

**Midwest Instruction and Computing Symposium**

# **Conference Proceedings**



**41<sup>st</sup> Annual Conference  
April 11<sup>th</sup>-12<sup>th</sup>, 2008  
La Crosse, Wisconsin**

© Copyright 2008 by the Midwest Instruction and Computing Symposium

Copyright 2008 by the Midwest Instruction and Computing Symposium. Permission to make printed or digital copies of all or part of this material for educational or personal use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies include this notice and the full citation on the first page.

# Table of Contents

## **Undergraduate Research**

- **Interdisciplinary Undergraduate Research in the CS Curriculum** .....1  
*D. Brown, D. Guster, B. Jansen (St. Cloud State University)*
- **Creating Visions for Computing Research**.....13  
*K. Sutherland (Augsburg College)*

## **Pedagogical Techniques**

- **Using Clickers to Enhance Computer Science Classes**.....18  
*J. Morrison (University of Wisconsin-Eau Claire)*
- **Grafting Technology onto Disciplinary Courses**.....29  
*B. Bultman (University of Wisconsin-Fox Valley)*
- **The Root Causes of the Students' Programs Quality Improvement in the TBC Method** .....42  
*Shawon Rahman (University of Wisconsin-Platteville)*

## **Algorithms**

- **Randomly Generating Well-Formed Postfix Expressions** .....54  
*A. Ng (University of Wisconsin-Parkside)*
- **Automated Process for Classifying Text Documents using K-Means and kNN** .....64  
*M. Evans, M. Lietzke, S. Huls, D. Svendsen (Saint John's University)*
- **Sectioning Points into Fixed-Size Sections**.....79  
*D. Edwins, L. Schlather, O. Hall-Holt (St. Olaf College)*

## **Computing Architectures**

- **Enhancing the Price/Performance for a Clustered Multiprocessor System**.....84  
*J. Myre, D. Ernst (U. of Wisconsin-Eau Claire)*
- **Performance Evaluation of Java RMI in Parallel and Distributed Discrete Event Simulation**.....96  
*T. Gamage, A. Ramadani, D. Hammes (St. Cloud State University)*
- **Securing the Border Gateway Protocol** .....109  
*M. Nickasch, J. Cavanaugh, M. Kohl, M. Dolfen, S. Rahman (University of Wisconsin-Platteville)*

## **Imaging and Video Processing**

- **Content-Aware Image Resizing** .....125  
*K. Wienkes, K. Hunt (University of Wisconsin – La Crosse)*
- **Tracking a Rat in Three Dimensional Space using Stereo Cameras** .....135  
*M. Meierpolys, M. Krahulec, D. Weibe, O. Hall-Holt, R. Price (St. Olaf College)*
- **Scientific Visualization of Magnetic Dipoles in a Lattice** .....145  
*Z. Oler, T. Urness (Drake University)*

## **Software Engineering**

- **Statistical Process Control of Software Processes for Obtaining CMMI Level 5**.....157  
*M. Rowe (U. of Wisconsin-Platteville), E. Farver, T. Bragg, M. Kelley, C. Hale (AVISTA Inc.)*
- **Introducing a Certificate in Software Testing for Non-Majors**.....172  
*J. Drake (University of Northern Iowa)*
- **The Characterization and Identification of Object-Oriented Model Defects** .....178  
*M. Rowe, R. Hasker (University of Wisconsin-Platteville)*

## **Web Technologies**

- **Exploring the Web Programming Jungle**.....193  
*C. Morrison (University of Wisconsin-Eau Claire)*

## **Application Programming**

- **Transversal Homomorphism and Orthogonal within OR/MS/DS Tools into VB.NET 2005**.....208  
*E. Tembe (University of Dubuque)*
- **PID Control in a Real-Time Embedded Systems Programming Course**.....229  
*J. Clifton (University of Wisconsin-Platteville)*
- **Simulation and Development of a Range Control Information Display System for UAS Operations in North Dakota** .....239  
*R. Marsh, S. Buettner, K. Ogaard (U. of N. D.) , J. Nordlie (Regional Weather Information Center)*

## **Algorithms**

- **An ANNTI (Artificial Neural Network Text Image) Spam Filter** .....250  
*J. Barr, C. Ashbacher (Mount Mercy College)*
- **Reflections on a Classic Trio of Graph Problems** .....265  
*T. O'Neil (University of North Dakota)*
- **Generative Programming Considerations for the Matrix-Chain Problem**.....272  
*A. Anda (Saint Cloud State University)*

## **Scientific Computation**

- **Simulation of Nitrogen Flow using the St. Olaf Beowulf Cluster**.....280  
*A. Waldschmidt, R. Brown, J. Schade (St. Olaf College)*
- **The Player is Always Right** .....285  
*S. Marquis (Lawrence University)*

## **Artificial Intelligence**

- **Chess AI**.....291  
*J. Odom (University of Wisconsin-Parkside)*
- **Course Scheduling with Genetic Algorithms** .....299  
*C. Davidson (Simpson College)*
- **A Learning Natural Language Parser** .....312  
*D. McKee (Lawrence University)*

## **Database and Information Retrieval**

- **Application of BLAST-based Techniques for Musical Information Retrieval** .....322  
*F. Korsakov (University of Northern Iowa)*
- **Mapping Application Attributes to Object/Relational Mapping Solutions**.....326  
*K. Hawkins, E. Towell (Carroll College)*

## **Software Tools**

- **Towards Musical Analysis Tools** .....343  
*C. Hill, S. Hagen (Valley City State University)*
- **A Policy-Based Scheduling Tool for Networking Labs** .....352  
*J. Yu (DePaul University)*

## **Algorithms**

- **A Dynamic Algorithm for Computing Periodicities of Misère Impartial Games**.....364  
*T. McConville (St. Olaf College)*
- **Dispersing Search and Rescue Robots** .....373  
*Binod K.C., K. Sutherland (Augsburg College)*
- **An Exploration of Implementing the A\* Algorithm Under Limited Resources in Lego Mindstorms**.....382  
*A. Horn (Graceland University)*

## **Classroom Innovation and Management**

- **Customizing MediaWiki for Project-based Courses** .....395  
*O. Hall-Holt (St. Olaf College)*
- **Integration of CodeLab into Programming Courses**.....401  
*M. Hall (University of Wisconsin-Marathon County)*
- **The Capstone Experience: Learning to Manage Uncertainty and Ambiguity in a Project Management Environment**.....413  
*S. Lynch (University of Wisconsin-Superior)*

## **Database and Information Retrieval**

- **An Algorithm to Restore Data Base Content to Past Dates in Real Time** .....424  
*C. Brown, D. Guster, B. Jansen (St. Cloud State University)*
- **A Methodology for the Design, Development, and Implementation of a Data Warehouse Project**.....436  
*C. Azarbod, M. R. Farooq (Minnesota State University at Mankato)*

## **Outreach and Recruitment**

- **An Experience in Teaching a Short Summer Robotics Course for High School Students**.....451  
*A. Lopez, E. Machkasova (University of Minnesota, Morris)*

## **Software Engineering and Ethics**

- **Lightweight Software Cost Estimation Model** .....462  
*I. Alsmadi (North Dakota State University)*

- **ISOMER: Enhancing Test Coverage Using Constrained Random Tests** .....468  
*D. Kulp, D. Ernst (University of Wisconsin-Eau Claire)*
- **Detecting Source Code Plagiarism** .....481  
*J. Degiovanni, I. Rahal (College of St. Benedict and St. John's University)*

### **Parallel and Distributed Computing**

- **Parallelizing the Computation of the SPT Statistic** .....494  
*T. Fredrick (St. Olaf College)*
- **Applications of Beowulf Cluster Computing to Problems in Biology** .....503  
*S. Debenport, R. Brown (St. Olaf College)*
- **Contemporary Technologies and Platforms for Electronic and Mobile Commerce Systems** .....508  
*W. Hu, Y. Zuo (U. of N.D.), L. Chen(Sam Houston State U.), A. Gopalakrishnan (U. of N.D.)*

### **Software Tools**

- **A Live View of the World**.....523  
*D. Hickok, M. Rowe (University of Wisconsin-Platteville)*
- **Computer Supported Collaborative Learning in the Geology Explorer** .....532  
*O. Borchert, B. Slator, G. Hokanson, L. Daniels, J. Reber, D. Reetz, B Saini-Eidukat, D. Schwert, J. Terpstra (North Dakota State University)*

### **Pedagogical Issues in Programming**

- **The Role of Writing Efficient Programs in a Data Structures Course** .....547  
*C. Hu (Carroll College)*
- **Effectively Apply Boundary Value Analysis Method in Student's Program Testing** .....557  
*S. Rahman (U. of Wisconsin-Platteville)*

# Interdisciplinary Undergraduate Research in the CS Curriculum

Richard Brown  
Department of Mathematics, Statistics, and Computer Science  
St. Olaf College  
Northfield, MN 55057  
rab@stolaf.edu

## Abstract

Undergraduate research, in the strict sense, means student inquiry leading to an original contribution to a discipline. We consider the processes associated with research at three developmental stages: guided discovery; independent investigation; and scholarly inquiry. Interdisciplinary research offers certain advantages for undergraduates, including accessible, interesting problems for students. In order to extend the benefits associated with research to a wide range of undergraduates, we provide interdisciplinary experiences at the three developmental stages in appropriate CS courses, spanning all academic levels in our curriculum.

We present examples of research-related activities in courses that represent six strategies for bringing research-related activities into the undergraduate classroom, and discuss the costs and benefits of those activities in terms of student learning and of human, curricular, and physical resources.

# 1 Introduction

Interest in undergraduate research has blossomed nationally on college campuses on an unprecedented scale. From announcements of tenure-track openings to professional publications for college presidents and deans, evidence abounds that project-based independent investigations has become an expectation for a strong undergraduate education, across the sciences, and increasingly in social sciences, arts, and humanities.

Interdisciplinary projects can provide some of the most fruitful opportunities for undergraduate research. In recent years, we have pursued interdisciplinary undergraduate research and related activities in connection with our computer science courses. These activities have increased the presence of undergraduate research in our CS program, enriched those courses, and developed interest in collaborative projects, within the context of a professor's teaching load (vs. independent research projects contributed by faculty outside of the ordinary load). We will explore our choices and discuss trade-offs inherent in bringing undergraduate research activities into the classroom.

## 2 Undergraduate research as a process

What should qualify as undergraduate research? The Council for Undergraduate Research (CUR) defines the term as “*An inquiry or investigation conducted by an undergraduate student that makes an original intellectual or creative contribution to the discipline*” [6]. Everyone agrees that independent work leading to publication in a professional journal would qualify. But not every student can achieve this high standard, at least, not in every discipline. Can something in the direction of undergraduate research but less than professional-level publication nevertheless be worthwhile for students?

CUR identifies the following benefits of undergraduate research [6].

- Enhances student learning through mentoring relationships with faculty.
- Increases retention in the science, technology, engineering and mathematics (STEM) pipeline.
- Increases enrollment in graduate education and provides effective career preparation.
- Develops critical thinking, creativity, problem solving and intellectual independence.
- Develops an understanding of research methodology.
- Promotes an innovation-oriented culture.



Other sources point out additional benefits. For example, the 2003 Bio2010 report states that students pursuing undergraduate research gain experience in working as part of a team, and learn effective oral and written presentation of scientific results [11, Ch.5].

Observe that students can derive a measure of each of these benefits from inquiry-based projects that do not result in publication at the professional level. Producing a peer-reviewed publication and presenting at a national or international scientific meeting is a unique and highly desirable experience in its own right, giving students an edifying sense of satisfaction and accomplishment as well as a valuable credential. But efforts that do not reach as far can still be worthwhile.

The Bio2010 report recognizes this by encouraging “independent investigation” experiences for all students in the biological sciences, even if they cannot all “experience the rewards and frustrations of original research.” Also,

While the richness of experience for the student likely will not be the same as working in a research group, it also is possible to provide meaningful research experiences for undergraduates in research-based courses or in teaching laboratories that are designed to be open-ended and to encourage independent investigation [11, Ch.5].

Therefore, we recognize a progression of developmental stages in undergraduate research activity.

1. Structured *guided discovery* exercises develop problem-solving skills and provide patterns for further investigations. These exercises can be included in even elementary courses.
2. *Independent investigation* projects involve a student or team in producing results that are new to them, using research methods, and in presenting those results in an appropriate forum.
3. *Scholarly inquiry* projects result in original contributions to a discipline, and public presentation at a professional level.

The boundaries between these categories are somewhat fluid. For example, a guided discovery exercise may include opportunities for open-ended pursuit of a question using (at least some) research methods. A strong independent investigation may be almost indistinguishable from scholarly inquiry, differing mostly in factors beyond a student’s direct control, such as the significance of the result to professional researchers.

Projects in one of these categories may achieve the benefits of undergraduate research to a greater or lesser degree. Even a project that results in a significant professional paper might not span all the benefits; for example, a student might “get lucky” in finding a significant result quickly, but miss the benefits of an extended mentoring relationship.

Therefore, we will consider guided discovery, independent investigation, and scholarly inquiry as general stages in the *undergraduate research process*. Through this process, students may benefit from attaining at least some of the benefits identified above, and progress towards achieving more of those benefits—depending on the student’s effort and insight, the choice of research problem, the influence of a research mentor (or mentors), and whatever happens to be discovered.

### 3 Interdisciplinary undergraduate research

Interdisciplinary problems offer many advantages for undergraduate research [10, p.2ff].

- By definition, interdisciplinary research involves two or more disciplines and a problem whose solution is beyond the scope of any one of those disciplines. Thus, if a problem can be identified that does not require graduate-level depth in one of its disciplines, an undergraduate in that discipline may well be capable of making a valuable contribution toward significant original interdisciplinary research.
- Interdisciplinary research is “real-world,” representing the inherent complexity of nature and society.
- Exploring problems that involve multiple disciplines is interesting, and may connect with multiple interests of a student.
- Undergraduates are strongly attracted to interdisciplinary issues that have societal relevance.
- Interdisciplinary research is typically collaborative, which helps to support and motivate students.
- New technologies help to drive interdisciplinary thinking, which provides added opportunities for CS students.

On the other hand, interdisciplinary research has its own set of challenges. For example, identifying collaborators and suitable problems may be difficult. Also, working together with people of disparate backgrounds may require extra time for building consensus and for learning new methods, languages, and cultures. These issues may intensify if multiple professors in different fields must co-supervise a project that neither fully understands.

In our experience, we have developed relationships over time with faculty members in other disciplines, identified research problems that cross the boundaries of each of those disciplines while being accessible to undergraduates, and found students interested in working on several of those research problems. Some key helps in bringing this about have been

considerable patience, a mutual desire among the faculty members to make the collaboration work, and tangible support and encouragement from department chairs and administrators.

## 4 Strategies for incorporating undergraduate research into the CS curriculum

The following six general strategies indicate ways that undergraduate research or its preparation may find its way into courses. These strategies may represent any or a combination of the three developmental stages for undergraduate research identified above. [3].

- *Project-centered courses*, in which a structured large project effectively organizes much or all of a course.
- *Project-accompanied courses*, in which a semester-long serves as a reference throughout a course.
- *Interdisciplinary courses with projects*, which may lead to new discoveries because of their interdisciplinary nature.
- *Team project courses*, solely devoted to undergraduate research activities in teams.
- *Individual project courses*, including independent research with individual or multiple students.
- *Client courses*, in which a course benefits from the results of undergraduate research efforts, whether or not that course includes research activities itself.

We will indicate these strategies in the following examples of interdisciplinary undergraduate research in the curriculum.

## 5 Examples

St. Olaf's computer science major is relatively young (approved in 2002), which gave our program an opportunity to incorporate interdisciplinary undergraduate research throughout our curriculum as we created the new major.

## 5.1 Earliest experiences

A locally developed programming environment used in our introductory course, CS 121 [4], provides an opportunity for cross-disciplinary guided discovery exercises and student-motivated “entry level” independent investigations.

We have modified MediaWiki, the open-source wiki software created for Wikipedia, to process additional languages besides wikitext for authoring pages, including Scheme, the CS 121 programming language [5]. We have also added features to support multimedia computation via Scheme, including production of XHTML pages represented as Scheme function calls, processing of sound waves represented as Scheme lists, generation of music using a MIDI-related protocol expressed in Scheme lists, rudimentary production and computation with grayscale and RGBA images, and object-oriented animation using a Scheme representation of the SVG standard. Students use these features in homework, as applications of standard course concepts, from nested function calls and simple recursion to object-oriented programming and project management. These are interdisciplinary applications: for example, the sound-wave material is presented in terms of the Physics of wave forms, with connections between musical overtones and harmonic frequencies; and elements of art and motion graphics design arise in an animation project.

In addition, students typically carry out three team projects during CS 121. For example, one project may call on students to create three or more contrasting sounds, whether by programming with sound waves or using the MIDI-based system for producing melodies and counter-melodies. The wording of the assignment invites teams to explore the possibilities of the system, with grade incentives for doing so.

The final project, to create a multi-scene animation with sound track, directs students to follow a staged development process, the first stages of which overlap with ongoing presentation of course material. In other words, this project is conducted as a guided discovery exercise, with opportunities to exceed the basic assigned expectations. Since the system implements the SVG standard, these beginning students are invited to learn more about that standard and animation techniques than appears in the course syllabus. In fact, students’ independent discoveries in past projects have informed teams in later terms, creating new thresholds that those later teams expect to surpass, and generally advancing the quality of the results. Teams present to each other and their friends in informal symposia at the end of the term, in a final open class period or an evening event.

Beyond team projects, we find each term that some students take on extracurricular independent investigations of their own. For example, one recent student applied the MIDI system (which supports arbitrary fractional pitch values) to explore the sonority of a non-logarithmic scale system she discovered on the Internet, and also a multiple diminution-stretto composition technique of her own invention. She did this work on her own initiative outside of the regular coursework, during the first third of the term.

CS 121 represents the strategy of a “project-accompanied course.” It is also a “client

course,” since undergraduate research has played a significant role in developing our specialized wiki.

## 5.2 Foundation courses

Further foundation courses in hardware and software design include guided discovery exercises designed to develop skills and methods that are useful in subsequent projects.

In particular, our second course CS 251 concludes with a multi-week team software development project in C++ using a waterfall-model methodology. That methodology is carefully prescribed as a series of deliverable goals for students to meet. Some teams choose to build interdisciplinary applications for their projects, although this is not required. Apart from such an application, these projects seldom have the nature of independent investigations. However these students do present their work to each other and friends at the end of the term (sometimes in a combined event with CS 121), a modest precursor to research presentations.

The great value of the CS 251 project for undergraduate research is in providing a common experience of carrying out a team software design project using a standard development methodology. Our program does not go into software engineering in depth, but this early exposure gives all students a baseline of design, implementation, and interpersonal skills they will need for any software-development aspects of future projects.

One required aspect of the CS 251 project has a somewhat interdisciplinary nature. While proposing, designing, and building their projects, students also conduct a simple formal ethical analysis of their software, using concepts derived from the ImpactCS project [9]. Computing ethics belongs as a field in computer science, but the methods in this analysis derive from the social sciences, and most of these students have never before associated notions such as “stakeholders” and even “ethical issues” with the field of computer science.

CS 251 illustrates the strategy of a “project-centered course” in its final weeks.

## 5.3 Core courses

Students taking core computer science courses in areas such as operating systems, programming languages, and ethical issues in computing carry out projects with a guided discovery framework that offer opportunities for extension into independent investigation. The subjects of these investigations are frequently interdisciplinary in nature.

We will consider two examples of such courses, both of which represent the “project-accompanied course” strategy.

- Our course in ethical issues, CS 263 [8], expands on the interdisciplinary theme in computing ethics begun in CS 251. Rather than keeping an abstract aloofness from applications, CS 263 focuses on ethics in the context of practical applications and professionalism. To make this emphasis concrete, the course requires a semester-long applied team project to perform ethical analysis of an actual system, in a service-learning context. Subject systems for ethical analysis have included the college registrar’s electronic information system, computing plans for a new science center, and a web-based system used by a local high school for distributing assignments and managing grades. These systems are analyzed broadly as *Socio-Technical Systems*, consisting of computing hardware and software, people, facilities, and other context.

Traditional social-science methods are presented in the course and used as tools for analysis. The applied ethical analysis project provides a natural connection and context for the other course content, and represents guided discovery with an opportunity for independent investigation in an interdisciplinary setting.

- Our operating systems course, CS 273, uses Linux as a primary example for illustrating operating system principles and their implementation. One of the projects involves modifying the Linux kernel in order to add one or more new system calls, or entry points into the operating system. The documentation describing requirements and procedures for this project provides a guided discovery framework for learning how system calls are implemented, and how to modify a large, complex code that one hasn’t written and which one can’t thoroughly understand.

The assignment is open-ended: a required system call must demonstrably extend the features of the kernel, but imaginative extensions receive an incentive in grading. This open-ended quality has led many students to delve deep into implementation issues, armed only with their conceptual understanding of operating systems principles and whatever experience they gain as they go. In one example, such an independent investigation led to a subsequent team undergraduate research project that explored a strategy for creating a more secure kernel [1].

- When a combinatorics researcher presented a problem in partition theory to our algorithms course CS 253 last Fall, students collaborated to create an algorithm that yielded significant new results in that mathematical field. By parallelizing that algorithm and implementing it on a Beowulf cluster, one student was able to achieve significant performance improvements. [7]

In terms of strategies for research, CS 253 is both a “project-accompanied course” and a “client course,” benefiting from resources developed by the Beowulf research team.

These two courses illustrate the strategy of “project-accompanied courses.”

## 5.4 Research courses

Some courses, e.g., a regularly offered advanced team project course, focus explicitly on independent investigation, typically into interdisciplinary questions, and work in those courses sometimes qualifies as true scholarly inquiry.

We offer two courses in this vein:

- The senior capstone seminar, CS 390, presents a one-semester experience in which every CS major participates in conducting a (brief) investigation into some aspect of an interdisciplinary research project, reads from the literature, performs a team ethical analysis of their project, documents their work, writes up their findings in a paper, and delivers those findings in a public presentation. This constitutes a model for the phases of research. Although a work of scholarly significance has yet to emerge from these one-semester forays into research, the results have local significance, and over time past investigations feed into other investigations taking place in the course (similar to the passing on of animation techniques in CS 121).

All CS majors, including weaker students, carry out CS 390 projects. When summer researchers and others with their own prior research projects take the course, CS 390 requirements adapt to the needs of that student. For example, one recent student completed work begun in a prior term and wrote up computing aspects of his interdisciplinary results [13].

CS 390 students collaborate on ethical analysis and may collaborate on other elements of the course, but their contributions to their project are accounted individually, and they write individual papers. Thus, this capstone primarily represents the strategy of an “individual project course.”

CS 390 also illustrates the “client course” strategy, in that most projects undertaken in the course arise from prior undergraduate research. Some of the most effective projects in CS 390 either complete some aspect of an earlier research project or extend one in some useful way.

- CS 350 is an applied team project course. Its purpose is to collect multiple team research projects so they can take place in a single course, providing students with a research community and increased access to mentoring, and providing a professor with course credit [2]. Most of the projects have had an interdisciplinary nature, and projects have ranged from guided discovery to scholarly inquiry, depending on the abilities and experience of the team and on the problem they tackle.

Besides illustrating the “team project course” strategy, CS 350 can also represent a “client course” that follows up on prior research. Some projects in CS 350 may produce results that other “client” courses can use. For example, St. Olaf’s first Beowulf cluster was constructed as a CS 350 project, and clusters deriving from that first system have served several “client courses.”

## 5.5 Interdisciplinary courses

Interdisciplinary courses with projects can offer a wealth of possibilities for undergraduate research activities. Some of these activities may produce original results.

Our course in bioinformatics, CS 315, requires students to read and present a research paper, then to undertake implementation projects applying CS to questions of concern to biologists. While these small-team implementation projects may be modest in scope, when selected by a bioinformatics specialist, they may represent original and useful small applications in bioinformatics. At the very least, they represent independent investigations that are new to the students.

## 6 Resources

This incorporation of interdisciplinary research throughout the curriculum requires resources of various kinds.

- *Faculty time and effort* must be an early consideration, since mentoring is so essential for successful undergraduate research. Effective mentoring requires time and effort of its own, for example, background research so that a mentor will have the information he or she needs in order to recommend research problems and strategies. Interdisciplinary projects typically require more preparation time, as a mentor must learn about another discipline (or other disciplines).

Longer term collaborations between individual faculty in different disciplines can significantly reduce the time for mentor background preparation. This typically narrows the range of knowledge needed for the collaboration: knowing about riparian plants may be enough; one needs study all of environmental science. Of course, background acquired for one shared project will often apply to other shared projects with the same collaborating mentor.

- *Course allocation* in a department must allow for scheduling research-rich classes. A sustainable and scalable effort cannot depend for long on overload activities, such as independent research without teaching credit. Integrating research into courses makes it part of a professor's teaching load. But scheduling such courses usually represents a trade-off, in which something else isn't scheduled.

St. Olaf's CS major was introduced recently (2002), and research-related courses were part of the original major proposal. Starting with a clean slate doubtless made this course allocation issue much easier to address than starting with a full curriculum of existing courses, as is the case for an established major.

- For established courses, *syllabus time* is usually a premium commodity. The prospect of dropping or curtailing the treatment of standard topics in favor of a lengthy re-



search project may be quite daunting, and even prohibitive, for example, in core courses.

However, activities at lower stages in the undergraduate research process may require less syllabus time. For example, a guided discovery exercise might replace another project assignment in a lower level course; or, an invitation to independent investigation may be added to an intermediate project, as an opportunity for a higher grade.

This approach was taken in the introductory course CS 121 with its wiki-based applications. With the help of this convenient and accessible development environment, we have been able to preserve all but a small number of peripheral topics in the prior course plan, a trade-off we have gladly made in exchange for the effectiveness and student interest in project-based activities.

Incremental syllabus modifications may not serve in courses pursuing more mature research activities. curricular debate about standard topics vs. new pedagogical approaches can be healthy from time to time, for any academic program.

- Suitable *research platforms and infrastructure*, e.g, specialized software or a Beowulf cluster, provide an environment for study and may offer capabilities that attract interdisciplinary collaborations. Some such resources, e.g., acquiring a GIS system, may require an initial investment of funds from a grant or institutional sources.

Other times, student efforts can effectively substitute for purchases. For example, St. Olaf's first Beowulf cluster [12] was built as an undergraduate research project from computers and network switches that the college had retired from regular use.

- *Summer research support* for students feeds the cycle of interaction with interdisciplinary research in an undergraduate curriculum. While research activities in courses need not depend on the existence of summer research, an extended period of investigation by a strong student or two can energize the work done on a series of problems. This effect is mutual: course-related research can also spur on summer research.

## 7 Conclusion

We have examined the nature and benefits of interdisciplinary undergraduate research, and described three stages in the development of research skills: guided discovery; independent investigation; and scholarly inquiry. We have also identified six types of course situations that offer strategic opportunities for including one or more of those developmental stages.

Several examples of interdisciplinary undergraduate research activity in courses were presented and discussed relative to the three stages and six strategic types of courses. These examples range from the early days of the introductory course to the senior capstone seminar. Resource issues concerning faculty time and effort, course allocation, syllabus time, equipment, and summer research support were considered.

## References

- [1] BONGARD, M., ENGLE, T., GERBER, A., HANDLEY, M., JOHNSON, T., AND BROWN, R. A. Secure Linux project. St. Olaf College, 2002–03.
- [2] BROWN, R. The advanced team project course, or how to manage a six-ring circus. In *Proceedings of the Midwest Instruction and Computing Symposium* (April 2006).
- [3] BROWN, R., ALLEN, R., HUFF, C., AND RUTHERFORD, R. Six strategies for merging computer science undergraduate research into the classroom. Poster presented at the Tenth CUR National Conference, June 2004.
- [4] BROWN, R., AND HALL-HOLT, O. CS 121, Principles of Computer Science. Retrieved March, 2008 from <http://www.stolaf.edu/depts/cs/academics/courses/list.html#CS1>, 2008. Online description of introductory course in CS.
- [5] BROWN, R. A., AND HALL-HOLT, O. Teaching computer science using a wiki with a general-purpose authoring language. In *Proceedings of the Midwest Instruction and Computing Symposium* (April 2007).
- [6] COUNCIL ON UNDERGRADUATE RESEARCH. CUR At-A-Glance. Retrieved March, 2008 from <http://www.cur.org/factsheet.html>, 2008.
- [7] FREDERICK, T. Parallelizing the computation of the spt statistic. In *Proceedings of the Midwest Instruction and Computing Symposium* (April 2008).
- [8] HUFF, C. CS 263, Ethical Issues in Software Design. Retrieved March, 2008 from <http://www.stolaf.edu/depts/cs/academics/courses/list.html#ESD>, 2008.
- [9] HUFF, C. W., AND MARTIN, D. Computing consequences : A framework for teaching ethical computing. *Communications of the ACM* 38, 12 (December 1995), 75–84.
- [10] NATIONAL ACADEMIES. *Facilitating Undergraduate Research*. National Academies Press, Washington, DC 20001, 2004.
- [11] NATIONAL RESEARCH COUNCIL OF THE NATIONAL ACADEMIES. *Bio2010: Transforming undergraduate education for future research biologists*. National Academies Press, Washington, DC 20055, 2003.
- [12] ST. OLAF COLLEGE BEOWULF TEAM. The St. Olaf Beowulf Project. Retrieved March, 2008 from <http://devel.cs.stolaf.edu/projects/bw/>, 2008. Wiki for project documentation.
- [13] WALDSCHMIDT, T. Simulation of nitrogen flow using the st. olaf beowulf cluster. In *Proceedings of the Midwest Instruction and Computing Symposium* (April 2008).

# Creating Visions for Computing Research

Karen T. Sutherland  
Department of Computer Science  
Augsburg College  
Minneapolis, MN 55454  
suther@navigation.augsburg.edu

## **Abstract**

A new organization, the Computing Community Consortium (CCC) has recently been funded by the National Science Foundation through the Computing Research Association (CRA). The purpose of this organization is to encourage the development of new visions for all areas of computing research.

One goal of the CCC is to get undergraduates excited about studying computer science and thinking about major research issues by involving them in the visioning process. For that reason, the CCC is sponsoring undergraduate poster contests at each regional meeting of the Consortium for Computing Sciences in Colleges (CCSC) as well as at the Midwest Instructional Computing Symposium (MICS).

This paper and the accompanying session will explore additional ways of involving undergraduate institutions in this development.

# 1 Introduction

The Computing Community Consortium (CCC) has recently been funded by the National Science Foundation through the Computing Research Association (CRA) with a goal of encouraging the development of new visions in computing research.

The research institutions which are involved in CCC projects will be holding “Crystallization Workshops” to involve more scientists and students in their project planning. CCC would like to see undergraduate faculty representation at these workshops and the projects infused at the beginning of the planning stage with ideas on how undergraduates can be involved.

As CCC supported projects begin to apply for external funding, they will be strongly encouraged to use the ideas gleaned during previous stages of development to involve the undergraduate community in a meaningful way. This could be in terms of collaborative grant proposals with undergraduate institutions, using REU supplements, exploiting opportunities to fund undergraduate researchers that are already written into calls for proposals, or actively engaging undergraduates in testing project results. The goal is to make undergraduate involvement an integral part of each project.

The CCC Web site will eventually have a page of links to opportunities for undergraduate students, faculty and institutions that have been created by CCC supported projects.

The purpose of this paper and session is to gain MICS attendee input on the poster contest, Crystallization Workshop participation and good ideas for undergraduate participation in these major projects. We are on the cutting edge of the development of the CCC’s mission and have an opportunity to influence outcomes that will have a major effect on the undergraduate computer science community.

## 2 What is a “Vision” in computing research?

Visions tend to run deep and broad. They involve fundamental questions in the field. They tightly integrate technological innovations with societal needs. They are often termed “audacious.” As an example, Jeannette Wing, the Assistant Director of the Computer and Information Science and Engineering Directorate (CISE) of the National Science Foundation has challenged the research community to articulate a research agenda to answer one of computing’s fundamental questions [7]:

“Is there a science for understanding the complexity of our networks such that we can engineer them to have predictable behavior?”

She defines networks broadly, including the physical layer at the bottom, the multiple architectural and protocol layers in the middle and the top layer of people and organizations. She argues that there are fundamental basic research questions underlying the scientific questions, technological innovations and societal demands that drive all areas of computing. The complexity of large-scale networks should be understood. New architectures

should be developed to manage future networks. New applications should insure security and privacy.

Webster defines “audacious” as daring or bold. It is not difficult to agree with Wing’s claim that “the vision for understanding the complexity of networked systems is audacious.”

Based on this vision, she calls on the research community to develop a compelling research agenda for the science and engineering of evolving, complex networks.

### 3 CCC Funded Projects

The CCC is currently funding the following projects:

- The Global Environment for Network Innovations (GENI):  
At the age of three years, GENI existed before the CCC was formed but is now under its auspices. The GENI project’s goal is to build an experimental facility designed to allow experiments on a wide variety of problems in communications, networking, distributed systems, cyber-security, and networked services and applications. Researchers will have the opportunity to experiment at a large scale with real user populations [1]. The GENI project is addressing the vision articulated by Jeannette Wing in Section 2.
- Big-Data Computing:  
The Big-Data Computing project held two CCC sponsored study groups in March 2008 to explore opportunities for research in high-performance, data-intensive computing systems. Applications range from astronomy to machine translation.
- Robotics:  
The goal of this project is to provide a comprehensive view of the use of robotics today and in the future, and to identify the necessary key competencies which need to be developed in order for the field to progress. Applications range from robotic surgery to planetary exploration to cars that park themselves.
- Theoretical Computer Science:  
The Theoretical Computer Science (TCS) group will be holding a CCC sponsored workshop in May 2008 with the following goals: “Identify broad research themes within theoretical computer science that have potential for a major impact in the future.” and “Distill these research directions into compelling ‘nuggets’ that can quickly convey their importance to a layperson.” The underlying goal of the project is to understand the capabilities and limitations of efficient computation. Applications range from public-key cryptography to quantum computation.

More can be read about all of these projects on the CCC Web page: [www.cra.org/ccc](http://www.cra.org/ccc).

## 4 Undergraduate Institutional Involvement

Knowing how important undergraduate education is to the computer science community, including the research community, both the National Science Foundation and the CCC are committed to involving undergraduate students and faculty in this new initiative. They would like our input as to exactly how we could best both contribute to and gain from this opportunity.

The poster contest that the CCC is sponsoring at MICS is a first step. The rationale for the contest was that the posting of the regional winning posters on the CCC Web page would raise awareness in the research community of the bright, talented undergraduates in our programs and raise awareness among our students of the cutting edge projects being undertaken which will affect not only the academic community, but all of society. It is hoped that knowledge of the societal impact that this research could have will help erase some of the negative stereotypes associated with our field, particularly for those students from underrepresented groups [2, 3, 4, 5, 6].

If the research institutions with CCC supported projects are going to be encouraged by the CCC to incorporate undergraduate participation into their projects, they need specific ideas as to how they could do so.

Some ideas for involvement, which we will discuss and augment in this session include:

- writing collaborative grant proposals with undergraduate institutions.
- encouragement to incorporate REU supplements into projects.
- providing support for undergraduate researchers from undergraduate institutions.
- providing collaborative summer research opportunities for faculty from undergraduate institutions.
- actively engaging undergraduates in testing project results.

While the above involvement is currently included in many NSF funded research projects, there remain many projects which include no such opportunities only because the proposers never thought of doing so, or thought of it but did not know how to approach it. Our goal should be to change this scenario.

As was stated above, the poster contest is only a first step. A window has been opened up for us to state our views to an audience that is eager to hear what we have to say. The vision of there being a single computer science community rather than there being “the research community” and “the CS undergraduate educational community” may be the biggest, boldest vision of all.

## References

- [1] Global Environment for Network Innovations. <http://www.geni.net>.
- [2] FISHER, A., MARGOLIS, J., AND MILLER, F. Undergraduate women in computer science: Experience, motivation, and culture. In *Proc. of the SIGSCE Technical Symposium on Computer Science Education* (Feb. 1997), pp. 106–110. Published in SIGSCE Bulletin, Vol. 29, N. 1, March 1997.
- [3] GARCIA, O. N., AND GILES, R. Research foundations for improving the representation of underrepresented minorities in the information technology workforce, June 2000. [www.cise.nsf.gov/itminorities/it\\_minorities\\_final\\_report.pdf](http://www.cise.nsf.gov/itminorities/it_minorities_final_report.pdf).
- [4] HALIBURTON, W. Gender differences in personality components of computer science students: A test of Holland's congruence hypothesis. In *Proc. of the SIGSCE Technical Symposium on Computer Science Education* (Feb. 1998), pp. 77–81. Published in SIGSCE Bulletin, Vol. 30, N. 1, March 1998.
- [5] MANNIX, M. Getting IT right. *Prism* (Mar. 2001), 15–20.
- [6] SACKROWITZ, M. G. An unlevel playing field: Women in the introductory computer science courses. In *Proc. of the SIGSCE Technical Symposium on Computer Science Education* (Feb. 1996), pp. 37–41. Published in SIGSCE Bulletin, Vol. 29, N. 1, March 1996.
- [7] WING, J. Network Science and Engineering: A Research Agenda. <http://www.cra.org/ccn/NSE.ppt.pdf>.

# Using Clickers to Enhance Computer Science Classes

Joline Morrison  
Department of Computer Science  
University of Wisconsin-Eau Claire  
Eau Claire, WI 54702  
morrisjp@uwec.edu

## **Abstract**

Research on learning approaches indicates that active and cooperative learning techniques improve educational processes and outcomes. One way to implement these techniques involves student response systems (clickers). This paper explores different knowledge types and learning approaches and argues that clickers can be used to aid the learning process with minimal risks. It provides examples of questions and techniques that can be used to enhance learning processes in computer science classes, and presents survey results of student attitudes towards clicker use in a CS 1 class. It also provides recommendations for effective clicker use.



## Introduction

The lecture is the mainstay of the college classroom. Lectures often degrade into instructor-led monologues with minimal interaction between the students and the instructor or among the students. Research on educational techniques (e.g., (1), (2), (3)) indicates that techniques such as active and cooperative learning improve educational processes and outcomes. One way to implement these approaches in the college classroom is by using student response systems, which are commonly referred to as “clickers.” The purpose of this paper is to explore how to use clickers to support different knowledge objectives and learning processes with minimal risk for both instructors and students, and to provide computer science-related examples. It also presents student attitudes towards clicker use in a CS 1 course, and provides recommendations for introducing clickers into your computer science classroom.

## Learning Processes, Approaches, and Risks

### Learning Processes

In 1956, Benjamin Bloom proposed a taxonomy of cognitive thinking to guide instructional pedagogy. Bloom’s taxonomy (1) consists of the following components:

1. *Basic knowledge*, which involves memorizing facts, figures, and base processes;
2. *Secondary comprehension*, which involves organizing, comparing, translating, interpreting, giving descriptions, and stating main ideas;
3. *Application*, which involves generalizing the facts or processes to other contexts and situations;
4. *Analysis*, which involves examining and breaking information into parts by identifying motives or causes, and then making inferences and finding evidence to support generalizations;
5. *Synthesis*, which involves compiling information together in different ways by combining elements in new patterns or proposing alternative solutions;
6. *Evaluation*, which involves presenting and defending opinions by making judgments about information, validity of ideas, or quality of work based on a set of criteria.

The first three items in this taxonomy are hierarchical: basic knowledge must be presented to gain comprehension, and comprehension must be attained before application can occur. The final three items are considered higher-order skills that cannot be attained before the first three are mastered. Figure 1 illustrates this hierarchy. The

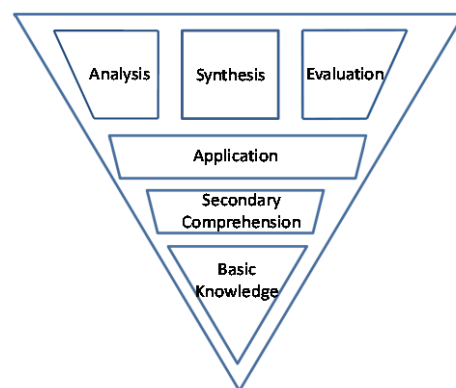


Figure 1 – Hierarchy of educational objectives

next section explores different learning approaches that can be used to achieve these objectives.

## Learning Approaches

Instructor lectures are the cornerstone of the college learning experience, but research (e.g., (2), p. 9) has shown that monologue lectures produce minimal learning gains. In response, educators have proposed techniques to help engage students and facilitate achieving educational objectives. One technique is *active learning*, which involves having students participate in activities beyond simply listening. A subset of active learning is *cooperative learning*, which occurs when teaching emphasizes social interactions between the instructor and students or among student groups (e.g., (3), (4)). The combination of active and cooperative learning is called *engaged learning* (5).

The most common and least risky approach to achieve engaged learning is to pose a question to the class and wait for a response. Unfortunately, one or two reliable students usually answer these questions, and the rest remain quiet, unengaged, and confident that the answer will be provided by one of their reliable colleagues. The "fast responders" enable the other students to remain quiet and unengaged, and might even prevent the other students from having time to think about the question and form an answer (6). To avoid this pitfall, the instructor might present alternate choices and then require all students to make a selection by raising their hands. This approach also has limited effectiveness: an unsure or unengaged student can still abstain from voting, and can easily alter his or her vote if it appears to be unpopular.

Math and science courses often involve require learning problem solving approaches or techniques. An engaged learning approach for this type of knowledge is for the instructor to first explain and illustrate the problem solving approach, and then have the students apply the approach to a different but similar problem on their own. After everyone is finished, the instructor and students discuss the solution as a group. This approach can be extended to cooperative learning by having students develop problem solutions in small groups, whereby a learning advantage is achieved by having students teach and learn from one another (7). This approach has been extensively used in computer science courses (e.g., (8)).

A more avant-garde engaged learning approach involves kinesthetic learning activities (KLAs), which force students to physically act out an activity in a way that teaches or reinforces the material (4). For example, a computer science instructor might teach students the difference between passing parameters by reference or by value using a note card with a value written on it to represent the parameter. The instructor passes the actual note card to represent passing by reference, and passes a copy of the note card to represent passing by value. In another example, the instructor might have students come to the front of the room and actually represent linked list nodes or values in a sorting algorithm.

Student response systems (“clickers”) have emerged as a popular approach to enable engaged learning. Clickers are similar to TV or stereo remote controls, and have numbered buttons that allow the student to select a response to a multiple choice question (1-5, A-E, etc.). Typically, the instructor displays a multiple choice question on the front screen, and students press clicker buttons to transmit their responses using an infrared or radio frequency. Responses are received by a base unit that relays them to the instructor’s computer, from which the aggregate results are displayed. The data is generally stored for later analysis. Responses can be anonymous or associated with a specific student (9).

## Engaged Learning: Rewards and Risks

Previous studies suggest that engaged learning techniques have a positive impact on educational outcomes, including improved student interest, involvement, confidence, retention, and higher order thinking (4). One source of these benefits involves student attention spans: multiple studies on student concentration levels indicate that concentration peaks at about 10-15 minutes into a lecture, but declines significantly thereafter [e.g., (10), (11)]. Engaged learning techniques improve classroom energy levels and give students the opportunity to synthesize, critically assess, and apply the concepts that the instructor presents. Furthermore, they allow the instructor to determine whether he or she has successfully conveyed target concepts (5). Studies specifically addressing the use of clickers suggest that these devices positively impact factors such as attendance, advance preparation for class, attentiveness, enthusiasm, in-class participation, and student confidence (e.g., (12), (13)).

McConnell (6) analyzes risk levels of engaged learning activities based on a variety of dimensions, as shown in Table 1.

<b>Dimension</b>	<b>Low Risk</b>	<b>High Risk</b>
<b>Class time</b>	Short	Long
<b>Amount of structure</b>	More	Less
<b>Amount of planning</b>	Careful	Spontaneous
<b>Controversy potential</b>	Low	High
<b>Student subject knowledge</b>	Strong	Weak
<b>Instructor experience with pedagogy</b>	High	Low
<b>Interaction type</b>	Student – Instructor	Student - Student

Table 1 – Levels of risk for engaged learning activities

This analysis asserts that carefully planned and highly structured activities that take shorter amounts of class time and rely solely on student/instructor interaction pose less risk than less structured activities that take longer and require interaction among students. Hence, an instructor asking students a pre-planned question is a low-risk strategy, while group activities and KLAs involve more risk. Low risk learning activities are more suited to the lower-level learning activities on Bloom’s hierarchy (e.g., conveying basic

knowledge and assessing secondary comprehension). Higher-level learning involving analysis, synthesis, and evaluation seem more suited to the higher risk activities such as group activities and KLAs. The higher-risk techniques can result in superb educational experiences or, if things don't go well, negative experiences and reluctant, confused, embarrassed, and/or disgruntled students.

Clicker technology involves minimal risk in the sense that it involves instructor-student interaction and is fairly structured. It certainly has the potential to enhance lower-level learning approaches involving instructor-student questioning and polling. When used with ingenuity and creativity, however, it also has the potential to enhance the higher-level processes while still minimizing the risks. The remainder of this paper describes approaches for using clickers effectively to enhance different learning at all levels within the domain of computer science.

## Using Clickers to Achieve Engaged Learning

Learning to operate clicker technology in your classroom is fairly easy. Using clickers effectively to enhance student learning is more challenging. The most obvious clicker use is to obtain *survey knowledge*, which asks for responses about existing knowledge (True/False: I have written computer programs using Java), experiences ("Which Web browser do you use?") or opinions/attitudes ("Which Web browser do you think most people use?"). This is especially useful when teaching freshmen, non-majors, or other groups that have diverse backgrounds or experiences.

Beatty et. al (14) propose that non-survey classroom clicker questions should address one of the following pedagogical goals:

1. *Content*, which involves presenting irrefutable facts that students must memorize and learn by rote.
2. *Process*, which involves teaching a cognitive process for solving a problem. This includes recognizing which process is applicable to a given situation, and identifying the information that is required to perform the process.
3. *Metacognitive*, which involves applying content and process knowledge to identify alternative solutions to problems with multiple competing solutions. This requires cost/benefit/risk analysis and viewing a problem from multiple viewpoints.

Beatty et. al (14) emphasize the importance of employing a "question cycle" that involves posing, pondering, answering and discussing. The following paragraphs describe strategies for using clicker technology to achieve these pedagogical goals both in terms of question types and how to manage the question cycle.

### Content Knowledge

During a lecture, the instructor attempts to convey and illuminate multiple points. Clickers can help students gain basic content knowledge by simply asking the students to

restate an idea. They can also be used to gain secondary comprehension by forcing students to summarize and synthesize a series of ideas. Figure 2 shows examples of content knowledge questions. Question 2(a) tests recall of rote facts, 2(b) requires students to consolidate multiple ideas to form a definition, and 2(c) requires students to apply previous-presented knowledge to a new situation.

Content knowledge questions for rote facts are usually answered fairly quickly (within a minute or so), and mainly serve as a wake-up call to gain student attention and ensure that notes are complete and correct. As a result, the question cycle is straight-forward: the instructor displays the question, waits until a high percentage of the students have answered, and then displays the answer distribution. These questions usually result in a high percentage of correct answers: For a question such as shown in Figure 2(a), you can simply state “The correct answer is b, 8 bits”, and then move on.

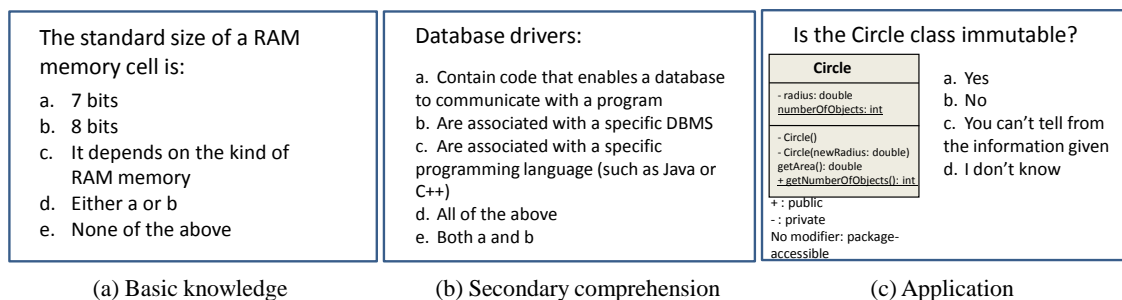


Figure 2: Examples of concept knowledge questions

The secondary comprehension question in Figure 2(b) and the knowledge application question in Figure 2(c) require more thought, and often result in some incorrect responses. You can use different approaches for managing the question cycle. When the percentage of incorrect answers is fairly low (less than about 25%), an effective approach for clearing the confusion of the minority is to state the correct answer, and then ask a volunteer to describe why he or she chose the correct answer rather than one of the incorrect ones (14). This tactic reinforces the correct answer for the correct responders while posing minimal risk for the incorrect ones.

When the percentage of incorrect answers approaches 50%, an effective management approach is to require students to compare their answer with their neighbors and then re-vote. This utilizes cooperative learning: it benefits the students who know the correct answer by forcing them to state their reasoning to convince their peers. Likewise, it benefits the students who do not know the answer by having it explained to them in a different way by their peers. It poses minimal risks to the unsure or incorrect students because no one is certain of the answer at this point, and the correct students must defend their reasoning.

It is useful to include a “None of the above” selection in most questions (14) because it provides insight into student reasoning that is unanticipated (and often correct!).

However, it should be the correct and intended answer often enough so as to be unpredictable.

## Process Knowledge

Computer science classes often require students to understand a series of steps for completing a task, such as converting a decimal number to binary, or writing program code to create a specific type of loop to complete a task. Usually, these processes are too complex to be reinforced or synthesized with a single clicker question. An alternative is to use a clicker question to reinforce a single step within the process. Figure 3(a) illustrates this approach: in this case, the process involves representing a floating point number using the binary numbering system, and the question reinforces the concept of normalizing a number using exponential notation.

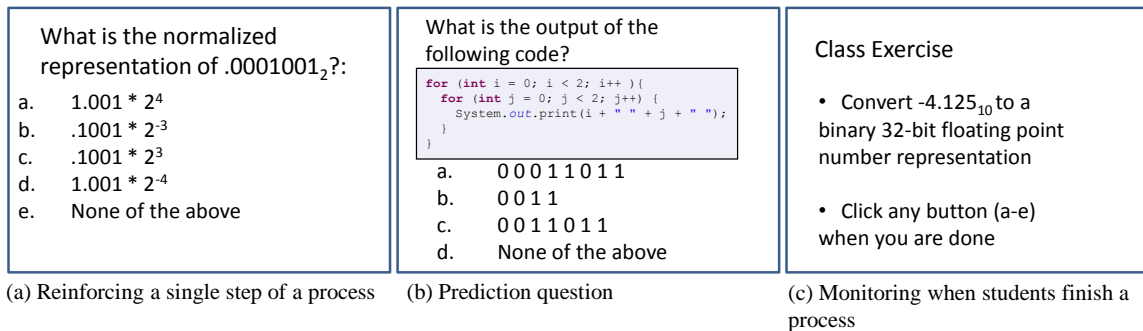


Figure 3: Example process knowledge questions

Another approach for reinforcing process knowledge is to use prediction questions such as the one shown in Figure 3(b). These questions are very effective for helping students understand programming concepts and program execution, and computer science students seem to enjoy their challenge. The question management cycle approach described in the previous section again applies: if the incorrect answers number 25% or less, ask a volunteer to describe the process that to their answer. If the incorrect answers approach 50%, ask students to describe their process to their neighbors and then re-vote.

Figure 3(c) illustrates how to use clickers to monitor student progress for performing all of the steps in a complex process. The instructor can monitor how many students have completed the process and judge when to discuss the solution. If the process solution is taking an undue amount of time, the instructor can modify the instructions and tell the students to signal when they have completed a portion of the problem, and then discuss the part of the solution that is causing the difficulty.

## Metacognitive Knowledge

Metacognitive knowledge involves applying content and process knowledge to identify alternative solutions to problems with multiple competing solutions. It encompasses the analysis, synthesis, and evaluation components in Bloom's hierarchy. At first glance, it

might seem that multiple choice questions are not compatible with these higher knowledge levels. However, it is possible to use clicker questions to spur discussions and higher-level thinking. Figure 4 shows three different approaches.

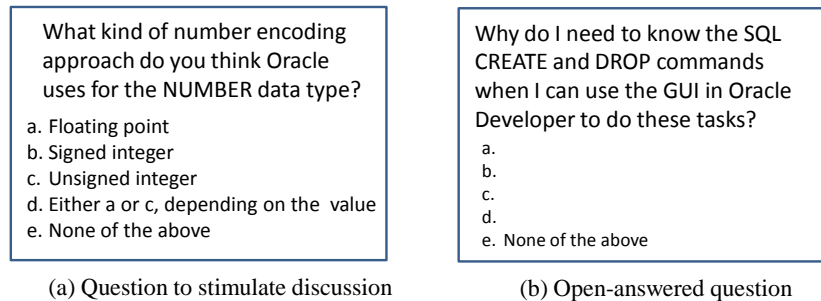


Figure 4: Examples of metacognitive knowledge questions

The question in Figure 4(a) occurs after defining the Oracle NUMBER data type, and requires students to connect how an Oracle database defines different types of numbers with previous knowledge on how computers encode number data in general. Students will first vote, and then will be asked to defend their choices. The question in 4(b) requires students to generate alternative, multiple defensible responses to a question. The instructor adds the alternatives on the fly, allows the students to vote, and then leads a class discussion on the results.

## Rewards, Drawbacks, and Best Practices

Studies on clicker use in science courses (e.g., (2), (13), (15)) suggest that students respond in a generally positive manner to clicker use in terms in making them feel more involved and engaged in a course. Clickers also tend to motivate students to come to class when attendance or answers contributed to their course grades.

Few (if any) studies have evaluated clicker effectiveness in computer science courses. Figure 5 shows survey results of clicker attitudes from approximately 40 students in a CS 1 course during the Fall 2007 semester. (Complete survey questions are shown in the appendix of this paper.)

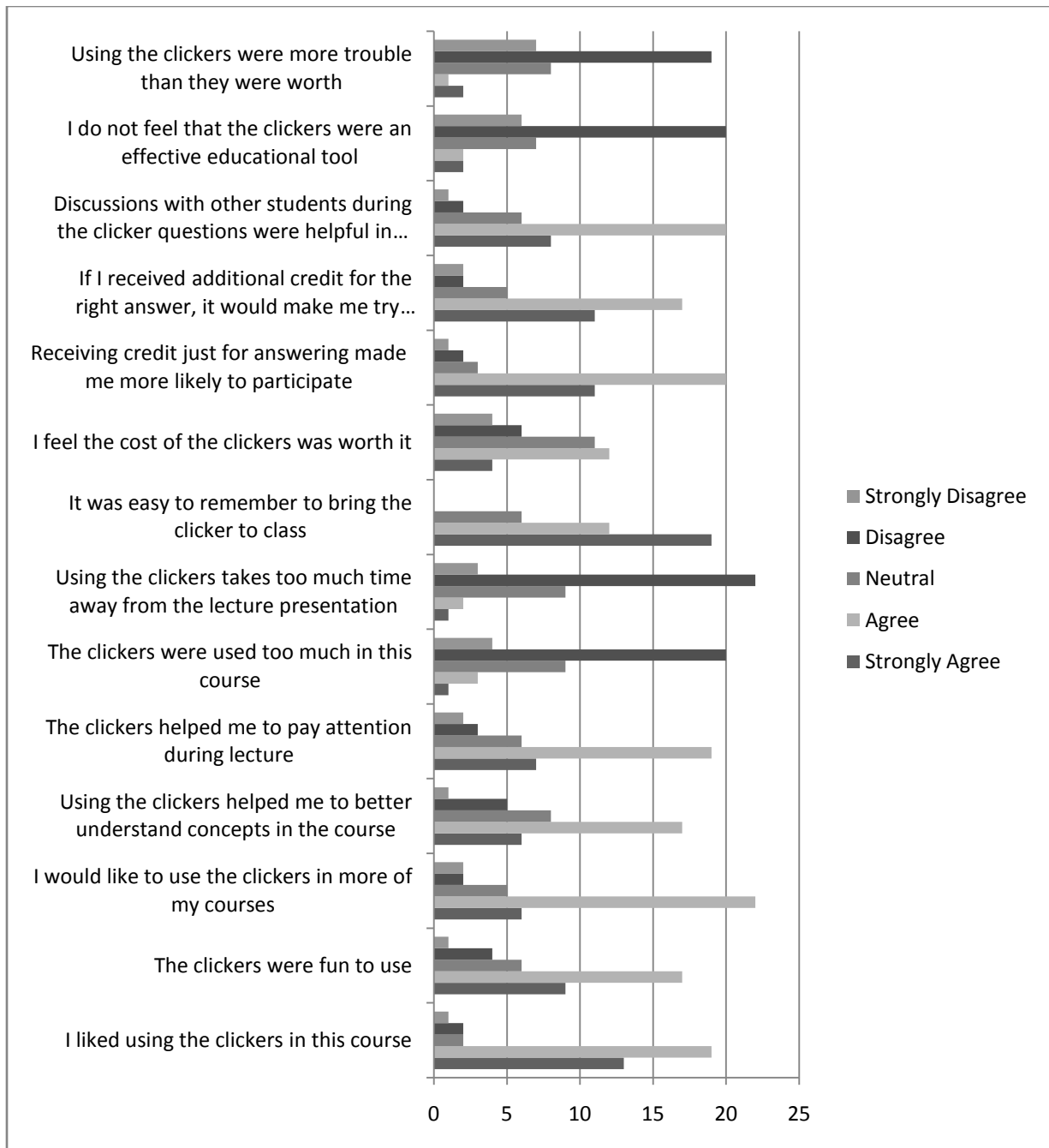


Figure 5: Survey data results

These results are similar to the ones seen in published studies in terms of clickers aiding student attention and understanding. Students were least enthusiastic concerning whether the cost of the clickers was worth their benefit. (At our bookstore, students purchase clickers for approximately \$35, and can resell them for \$20 at the end of the semester. I also advise students that they can purchase their clickers on EBay for a reduced amount.) Overall, clickers seem to be an effective educational tool in the college classroom, and this includes the computer science classroom as well.



As a result of a literature survey, personal experiences, and conversations with other instructors who are using clickers in the classroom, here is a list of recommendations to help make your transition to clickers successful:

1. Connect with other instructors at your institution who are using or are interested in using clickers, and agree upon a single vendor. This way, students can purchase a single clicker and use it in multiple classes.
2. Select a clicker type that minimizes the cost to the student.
3. Use the clickers regularly in every class. A common complaint among students is that they were forced to purchase the clicker and then used it sporadically or not at all during the semester.
4. Provide “loaner” clickers to lend to students who invariably forget to bring their clickers to class. Students complain if they don’t get credit for coming to class just because they forgot their clicker.
5. Use a variety of question types and question cycle techniques. Using the clickers for the same types of questions and in the same way becomes boring.
6. Avoid placing a lot of emphasis (more than 5%) on grades for getting the correct answer. This adds to student anxiety, especially among students who take longer to grasp new concepts.

## Acknowledgement

The author would like to thank Dr. Matt Evans of the Department of Physics and Astronomy at the University of Wisconsin-Eau Claire for his efforts in bringing clickers to the classroom at UW-Eau Claire, and for his insight and suggestions for improving this paper.

## References

1. **Bloom, Benjamin S.** *Taxonomy of educational objectives: Cognitive domain*. New York : David McKay and Company, 1956.
2. **Duncan, Douglas.** *Clickers in the Classroom*. San Francisco, CA : Pearson Education, Inc., 2005.
3. *A Systematic Approach to Active and Cooperative Learning in CS1 and its effects on CS2.* **Gonzalez, Graciela.** Houston, TX : ACM, 2006. SIGCSE 06. pp. 133-137.
4. **McConnell, Jeffrey J.** Active and Cooperative Learning: Tips and Tricks (Part I). *Inroads - The SIGCSE Bulletin*. June, 2005, Vol. 37, 2.
5. **Lazar, Alina.** Engaged Learning in a Computer Science Course. *Journal of Computing Sciences in Colleges* . October, 2007, Vol. 23, 1.
6. **McConnell, Jeffrey J.** Active and Cooperative Learning: More Tips and Tricks (Part II). *Inroads - The SIGCSE Bulletin*. December, 2005, Vol. 37, 4.
7. **Johnson, D.W., Johnson, R. T. and Smith, K. A.** *Learning: Cooperation in the College Classroom*. s.l. : Interaction Book Company, 1991.
8. *Cooperative Learning Techniques in CS1: Design and Experimental Evaluation.* **Beck, Leland L., Chizhik, Alexander W. and McElroy, Amy C.** St. Louis : SIGCSE 2005, 2005.

9. **Herreid, Clyde Freeman.** "Clicker" Cases: Introducing Case Study Teaching Into Large Classrooms. *Journal of College Science Teaching*. October, 2006.
10. **Stuart, John and Rutherford, R.J.** Medical student concentration during lectures. *The Lancet*. September 2, 1978.
11. **Thomas, E. J.** The variation of memory with time for information appearing during a lecture. *Studies in Adult Education*. April, 1972, Vol. 4, 1.
12. **Bullock, D.W., et al.** Enhancing the student-instructor interaction frequency. *Phys. Teacher*. 2002, Vol. 40, 535-541.
13. **Suchman, Erica, et al.** Evaluating the Impact of a Classroom Response System in a Microbiology Course. *Microbiology Education*. May, 2006, Vol. 7, 5.
14. **Beatty, Ian D., et al.** Designing effective questions for classroom response system teaching. *Am. J. Phys.* January 2006, 2006, Vol. 74, 1.
15. **Trees, April R. and Jackson, Michael R.** The learning environment in clicker classrooms: student processes of learning and involvement in large university-level courses using student response systems. *Learning, Media and Technology*. 1, 2007, Vol. 32, March 2007.

## Appendix – Student Response System Clicker Attitude Survey

(a) Strongly Agree (b) Agree (c) Neutral (d) Disagree (e) Strongly Disagree

1. I liked using the clickers in this course.
2. The clickers were fun to use.
3. I would like to use the clickers in more of my courses.
4. Using the clickers helped me to better understand concepts in the course.
5. The clickers helped me to pay attention during lecture.
6. The clickers were used too much in this course.
7. Using the clickers takes too much time away from the lecture presentation.
8. It was easy to remember to bring the clicker to class.
9. I feel the cost of the clickers was worth it.
10. Receiving credit just for answering made me more likely to participate in using the clickers.
11. If I received additional credit for the right answer, it would make me try harder to get the right answer.
12. Discussions with other students during the clicker questions were helpful in understanding the concepts in this course.
13. I do not feel that the clickers were an effective educational tool.
14. Using the clickers were more trouble than they were worth.

# Grafting Technology onto Disciplinary Courses

William Bultman  
Department of Computer Science  
University of Wisconsin-Fox Valley  
Menasha, WI 54952  
bill.bultman@uwc.edu

## Abstract

We describe a novel course to teach students the technology skills they need to succeed in another linked disciplinary course outside of computer science. We believed that students would learn the technology skills better by having a context in which to apply them. They would also benefit in the linked course from the ability to immediately use these skills.

A one-credit course was developed with assistance from instructors in various disciplines to determine the skills that *they* felt would be useful in their courses. Two sections were offered each semester for four semesters, each linked to a different cluster of disciplinary courses.

Along with improving technology skills, the course also sought to increase students' engagement in the university and improve their ability to work in groups. We measured the level of success in achieving each of these goals with pre-and post-semester surveys, and report the results of that assessment.

# 1 Inspiration

The UW-Fox Valley is one of the two-year transfer institutions of the University of Wisconsin System. The Computer Science department has had a cluster of one-credit computer applications courses that have remained relatively unchanged for two decades. With Evelyn Li from the Instructional Technology department, I set out to design a course that addressed changes to the student population over that time.

We attempted to address three needs in particular, based on our experience with today's students. First, students learn skills much more easily when they can see an immediate application of the material in the context of a "real" use for it. Second, students need to be guided into engagement, both with the course material and with the university overall. Third, students are increasingly asked to use software and hardware to create group term projects and larger presentations that they are unprepared to accomplish.

## 1.1 Technology Skills Applied in Context

The first of these needs ended up being the inspiration for the title of our course: *Technology Skills Applied in Context*. Students want to learn material on a need-to-know basis. A typical student would be one who completed a spreadsheets course, but only has that "Aha Moment" after applying a particular skill in an accounting course later. The standard courses attempt to weave in "real world" applications, but without the larger context they are still disconnected.

To address this need, we wanted our course to be linked to a non-computer science disciplinary course such as speech, business, or laboratory sciences. For clarity in this paper, we will refer to this course as the "linked course" and ours as the "technology course." Each offering of the technology course would teach students the particular skills they needed to succeed in that linked course.

This goal of teaching in context leads to another aspect that turned out to be fundamental to the nature of our course. Since different disciplines require different skills, we needed to find a manageable way to custom tailor the skills in each section of the technology course to those needed by the linked course. The instructor in the linked course should have an easy way to have significant influence on the topics covered in our technology course.

## **1.2 Increase Student Engagement**

A second need that we hoped to address was a general decrease in engagement with the university on the part of incoming freshmen. This was in part inspired by an initiative that the university was undertaking at the time, and we hoped that this course could make a contribution toward that goal. That initiative also brought with it grant opportunities, and we received a \$2,000 grant to support the development of the course.

One of the most difficult aspects of the course inspiration was trying to get practical about what is meant by “engagement.” Various dictionaries define it as “holding the attention of or interest in,” “involvement in activity,” or “emotional involvement or commitment.” We worked toward two aspects with the course: engagement in the classroom and engagement in the university.

In the classroom we focused mainly on the “activity” and “attention” aspects of the definitions. We would encourage active participation during class time, avoiding traditional lectures as much as possible. We would try to introduce nontraditional but relevant hardware (e.g., digital cameras) and software (e.g., video editing) along with the traditional application software topics, in hopes of engaging students’ interest.

Outside of the classroom we focused on the “involvement” and “commitment” aspect of the definitions. We would create small group activities to promote bonding between students. We would incorporate extracurricular activities such as noncredit technology workshops into the grading structure of the course. We would require participation in an online discussion forum.

## **1.3 Group Dynamics and Presentation**

The third need that we hoped to address was driven by increasing technology expectations for group projects and presentations. This was actually requested by the potential instructors we chatted with in preliminary discussions about the course.

Due to time constraints, most courses with projects and presentations do not provide instruction on how to use the technology. They assume that most students could figure it out on their own or get assistance from technology staff, which is true to some extent. But even those adept with the mechanics of technology often have little understanding of the principles of the systems. Most of us can recall PowerPoint presentations that attest to the difference between ability to use a tool and understanding of *how* to use the tool. We would give instruction targeted at both aspects: mechanics and principles.

Another need related to projects and presentations is that most students have little training in how to effectively work in groups. We would give instruction to students in how to effectively work together in groups. We would assign practice group projects as part of our technology course. When the linked course itself included projects or presentations, we would incorporate a separate assessment of their work from the point of view of technology and group work for the technology course.

## 2 Development

The course details were developed primarily during the 2005-06 academic year. For practical considerations of instructor and student workload we limited ourselves to a one-credit course. Initial queries yielded nine instructors who were potentially interested in linking to the course, and we relied extensively on their input throughout the development process.

For mechanical and somewhat political reasons we decided to develop the course outside of the computer science department. The course fit fairly nicely into an existing course called a Lecture Forum, LEC 104: *Course Supplement*. However, this decision led to some problems that we will discuss in Section 3.3.

### 2.1 Linked Course Selection

The typical model for linking courses at our university is to have a one-to-one correspondence between pairs of courses: students enroll in either both courses or neither. However, we wanted participation in the technology course to be optional, supplementing the linked course for those that wanted technology help. This implied that if we wanted to fully enroll our course, we needed to link multiple sections of linked courses to the same technology course section.

It was determined that the department would run only two sections per semester during the pilot stage of the course. Thus, even clustering several linked course sections together, we could not handle as many linked sections as we had interested instructors. We interviewed potential instructors for the types of technology skills needed, projects or presentations expected, and in the end chose two or three sections to link to each technology course section.

### 2.2 Topic Selection

The process of interviewing the instructors for compatibility with the course actually yielded an unexpected harvest: a list of potential topics that we needed to cover. We compiled the complete list of potential topics, homogenized their descriptions, and analyzed them for commonality. We made a second pass with the instructors selected for participation, asking them to prioritize the topics for their linked course.

One goal for the design of the technology course was to keep the content manageable for instructors. We didn't want to burden them with designing the technology course totally from scratch for each new linked course section. We hoped to *design* the course once, and allow it to be *implemented* easily many times.

		ZOO 101	CHE 145	BUS 101	COM 103	BIO 109	GEO 106
Required Topics (all)	<b>Windows Basics</b>						
	File Management	1		1		1	1
	Internet Searching	1	2	2	1	1	1
	<b>Spreadsheet</b>						
	Data Manipulation	1	1	1		1	1
	Formatting	2	1	1		1	1
	Graphing/Charting	1	1	1		1	1
	Analysis/Statistics	2	2	1		1	1
	<b>Presentation</b>						
	Basic Presentations	1	1	1	1	1	1
	Dynamic Slide Show	1	1	2	2	1	1
	Presentation Skills	1	1	1	1	1	1
	Group Dynamics	1	2	1	2	1	1
Supplemental Topics (choose)	<b>Database</b>						
	Database Structures	2				2	1
	Queries	2				2	1
	Reports/Forms	1				1	1
	<b>Desktop Publishing</b>						
	Poster Making	1				1	
	<b>Multimedia</b>						
	Scanning	1			3	1	2
	Video Editing	2	1		3	2	
	Digital Cameras	2			3	2	
	Image Manipulation	2			3	2	2
	Audio Editing	2	2		3	2	
	<b>Web Page Publishing</b>						
Page Formatting	3				3		
Publishing	3				3		

Table 1: Chinese Menu of Topics with Priorities

What we ended up with was the list of *Required Topics* and *Supplemental Topics* shown in Table 1 that we affectionately refer to as the Chinese Menu. Required topics were those that appeared as a high priority (marked with 1 in the table) for most of the linked courses. These topics would make up 60% of the course. Though they would be given a disciplinary focus depending on the linked course, they would provide a welcome core of stability for the technology instructors. Linked instructors would also be given the opportunity to add their own supplemental topics.

## 3 Implementation

As mentioned, one of the goals of the development process was that we would perform the complicated job of developing the course once, but make the job of implementing a particular section of the course relatively straightforward. To that end, we staffed one of the two sections each semester with a member of the development team, while the second section was staffed by an instructor did not participate in the development process. We felt that the feedback from that improvised course preparation would give us insight on the wider applicability of this model. In this section we describe that process as it works now, after three semesters and several evolutionary changes.

### 3.1 Constructing a Section

The process begins by selecting a group of course sections to cluster together that have similar needs. The instructor in each potential linked course would prioritize the items on the Chinese Menu, as illustrated above. If there are enough sections of a single course, this is a good candidate, but the individual instructors from those sections should still prioritize their needs, since use of technology differs widely between instructors in some disciplines.

The first time this is implemented at a location, we recommend casting a wide net initially and selecting out a cluster of courses with similar needs based on the topic selection. We found some clusters that are not intuitive, e.g., the cluster of *Concepts of Biology* with *Geographic Information Systems* would not have occurred to us without this analysis.

### 3.2 Materials Required

The most important materials are the hardware and software needed by students in the course. Though most classes were held in the standard instructional computer lab, we relied heavily on a Media Lab available to all students for instruction in some of the higher end topics. You will need to consider the technology facilities available at your location and either adapt the course to them or budget in advance for the necessary upgrades of equipment.

Though the Media Lab has fairly advanced technology, we tried where practical to use hardware and software that the students would have access to after leaving our university. For example, we taught video editing with Windows Movie Maker rather than Vegas Pro used by Com Arts courses, and allowed students to bring in their own digital cameras if they had them.



We were not able to find a single textbook that met all of the needs of this course. Yet, being a one-credit course, it would have been unreasonable to force the students to purchase two textbooks. In the end, we selected a “brief” edition of a Microsoft Office and Windows textbook. It was still both too much (400 pages, ~\$60) and too little (missing topics). But it gave the students a security blanket for much of the course material.

Supplemental materials need to be obtained in advance for any topics not covered by the textbook. In our case, handouts were available from the IT department for some topics. Other materials were obtained directly from Microsoft or from sources on the Web, used with permission.

The final sort of “materials” requirements that you need to take into consideration is instructor expertise. Just as with the computer hardware, if the staff is not sufficient to the needs of the course, you will either need to adapt the content of the course to them or budget for the necessary upgrades (training) of the equipment.

### 3.3 Enrollment

Enrollment history for the LEC 104 technology course is indicated in Figure 1. As you can see, the course has never been enrolled to its full potential of 30 seats (the computer lab size). Also as indicated, several of the sections had such low enrollment that they were cancelled before the semester started up.

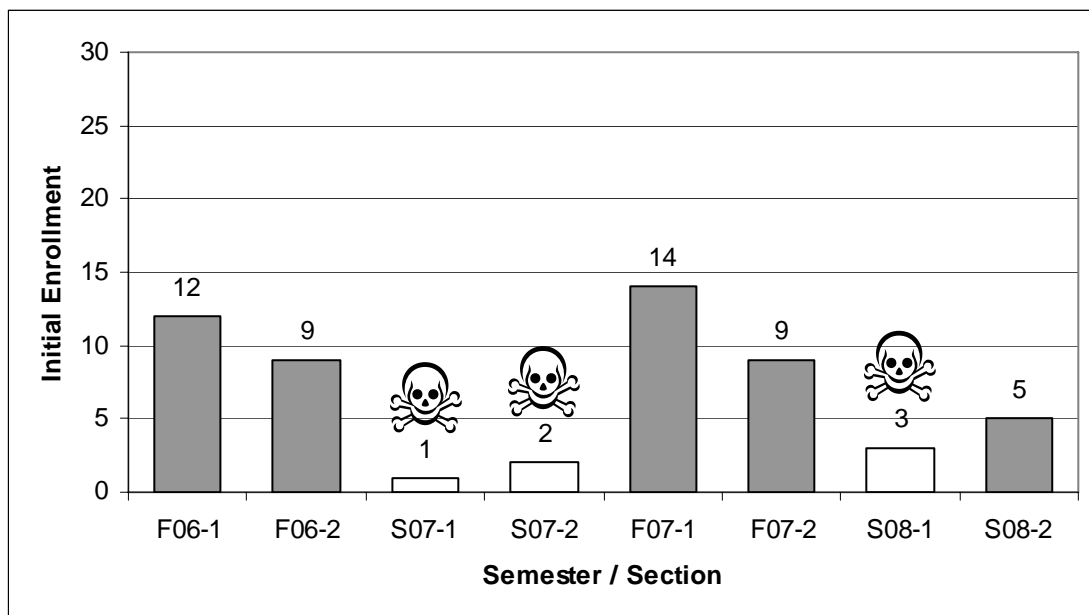


Figure 1: LEC 104 Enrollment History

Causes for the low enrollment remain unclear. As we show in Section 5, the course is well received by linked instructors and well reviewed by students. We show in Section 4 that the course is succeeding in most of its goals. It has been “advertised” to advisors who register students. Linked course instructors have encouraged their students to add the course on the first class day of the semester.

In the second year of the course we attempted to address the low enrollment by “widening the net” considerably, i.e., increasing the number in each cluster of linked courses. Section one was linked to any science course with a laboratory component (~15 courses) and section two linked to any business course (~10 courses).

As you saw from Figure 1, that alone did not improve enrollments. But just as importantly, we felt a philosophical disconnect with the inspiration for the course. By linking to so many possible sections we are severely encumbered in the ability to custom tailor the content to the needs of the linked course.

Our current best hypothesis for the cause of the continued low enrollment is confusion over the existence and purpose of the course. Since it is not listed in the computer science section of the print or online course schedule, students and advisors just do not think of it when advising. Cross references have been added to the course schedule in all linked courses to increase awareness. The course is also listed in the section with the mandatory freshman seminar course, LEC 100, which adds to the confusion about the course.

## **4 Assessment**

One of our goals in developing the course was to be able to quantitatively assess whether this course is meeting the goals we set for it. We did not want to just add another ordinary computer applications course to the assortment.

As indicated in Section 1, there were three primary goals for the course: technology skills, student engagement, and group dynamics. Thus, the questions in our assessment focused on those three aspects. Most quantitative measures were gathered on a five-point Likert scale and normalized where necessary.

### **4.1 Measuring Improvement**

Determining the effectiveness of the course could not be done on an absolute scale like grades, because they do not take into account the student’s skill level before entering the course. What we needed was a way to measure the *increase* in skill over time.

Furthermore, we could not measure only the increase in *LEC 104* students’ skills, since in general students’ skills increase just by virtue of their other class work. Thus we needed a control group of students in the same linked courses but not in *LEC 104*. So what we really needed was a way to measure that increase in skill *relative* to the control group.

We also wanted to use the demographic data to see whether there were any populations of students who were better served by such a course. In order to do so, we needed to collect relevant demographic data such as age, sex, and experience level.

We developed pre- and post-semester surveys that we would administer to students in the linked courses, both those in LEC 104 and those not. The surveys were administered via e-mail and a Web site. They included 30-40 questions with demographic data, previous experience and future intentions, and self-assessment of skills and attitudes. The full survey is available on request. We set a goal of at least 5% improvement in each area in order to consider the course a success (marked with a ☺ below).

## 4.2 Sample Size

Before giving any results, we should emphasize that with the relatively low enrollment in LEC 104 and only partial participation from those few students, the sample size on all of these statistics is relatively low.

We received survey results from 267 different students. Unfortunately, only 111 of them responded to both the pre- and post-semester survey, so only those could really be used for our intended analysis. Of those, 25 were enrolled in LEC 104, while 86 were only enrolled in the linked course. Though the number of LEC 104 responses was quite low, it does represent about 2/3 of the students who completed the course.

## 4.3 Technology Skills Results

The assessment of the effect of LEC 104 on students' technology skills was a clear win across the board. Technology skills were assessed both overall and individually for the six possible computer applications components of the course shown in Table 2. As you can see, students' self-assessment of skill level in all areas increased by more than 5%.

Did LEC 104 course increase students' self-assessment of computer skill in...	Control	LEC 104	Difference
word processing	+1%	+7%	+6% ☺
spreadsheets	+1%	+18%	+17% ☺
database	-3%	+17%	+20% ☺
presentation software	+5%	+22%	+17% ☺
audio/video editing	-1%	+14%	+15% ☺
desktop publishing	-1%	+19%	+20% ☺
technology skill overall	+0%	+7%	+7% ☺

Table 2: Improvement in Technology Skills

## 4.4 Student Engagement Results

Student engagement was measured indirectly through several of our best attempts to indicate engagement in terms that the students could comfortably quantify. They included the various measures shown in Table 3. As you can see, the assessment was a mixed bag on individual items, but is judged a positive result overall.

Did the LEC 104 course increase student engagement, as measured by...	Control	LEC 104	Difference
willingness to approach instructor	+8%	+4%	-4%
interest in joining student clubs	-7%	+4%	+11% 😊
participation in campus workshops	-5%	-2%	+3%
plans to remain at UW-Fox next year	-3%	+4%	+7% 😊
interest level in their linked course	-10%	-17%	-7% 🙅

Table 3: Improvement in Student Engagement

## 4.5 Group Dynamics Results

Group dynamics was measured directly by their self-assessment of the two factors shown in Table 4. In the first year's survey, a weak positive result was indicated, but with additional second year data, the overall results are weakly negative. This variability may be a real difference in instruction between the two semesters, or just statistical variability due to the small sample size. Either way, the results cannot be considered positive.

Did LEC 104 improve students' ability to work in groups, as measured by...	Control	LEC 104	Difference
their comfort in working in groups	+1%	-7%	-8% 🙅
their self-assessed skill level	+8%	+10%	+2%

Table 4: Improvement in Group Dynamics

## 4.6 Demographic Differences

Finally, the results in Table 5 create meta-measures of the three primary goals above by averaging the "Difference" column for various focused subgroups of students. So effectively each single number entry in the table below represents an amalgamation of all rows and columns in one of the three tables above, but restricted to only the students in the LEC 104 and Control groups that are in the subgroup indicated on the left.

The data in this table is complicated to read, so an example might be useful. Take the -6% 📉 entry in near the bottom right of the table. This course appears to have increased the computer skills of first semester freshmen 6% less than those who did not even take the course. (This is not quite as discouraging as it might seem; in fact both actually improved, but those in LEC 104 improved less.) Compare this to the +11% entry at the top of the column that indicates that in general LEC 104 students improved by 11% more than students who did not take the course. Overall then, what this indicates is that the course is doing a significantly worse job (by 17%) helping first semester freshmen than students in general.

Each subgroup listed is one that before the course began we predicted would be better served by this course, mostly based on intuition. So the fact that we were doing significantly worse in assisting first semester freshmen is at least counterintuitive to us, and perhaps even disappointing. Each such negative entry deserves closer consideration to see how we might improve delivery to those subgroups.

As you can see, the sample sizes on such restricted populations of students are very small, sometimes to the point of rendering the results completely unreliable. Sample sizes are represented by (n=#L/#C) where #L indicates the size of the LEC 104 sample and #C indicates the size of the Control sample.

Did students in LEC 104 in groups below ↴ increase more or less than control in goal to right? ⇨	student engagement	group work ability	computer skill
<i>all LEC 104 (n=25/86)</i>	+2%	-3%	+11%
more previous experience (n=2/22)	+9% 😊	-20% 📉	+16%
female (n=17/56)	+1%	-1%	+11%
over 23 (n=5/16)	+4%	-8% 📉	+23% 😊
having their own computer (n=10/60)	+3%	-8% 📉	+12%
first semester freshmen (n=15/34)	+4%	+0%	-6% 📉
no declared major (n=6/32)	+0%	+0%	+9%

Table 5: Improvement by Focused Populations

## 5 Feedback

We obtained systematic but subjective feedback from three sources: the post-semester survey of students mentioned previously, the standard student course evaluations, and a survey of linked course instructors.

## 5.1 Student Feedback

One question on the post-semester survey of LEC 104 students was particularly relevant to overall feedback. There were 26 responses, but a fair representative sample is collected below.

*Q: What aspects of LEC 104 helped you the most in the linked course?*

- Learning more about powerpoints, and how to present them in class.
- It is a good class, but requires a lot of time and work for 1 credit course.
- Learning research techniques and how to use the different office products better
- The Excel because we had to use Excel for our labs
- Although it is a nice refresher on stuff I already knew, I knew almost all of it.

The second source of student feedback is the standard Student Survey of Instruction instrument used by the university to evaluate instructors. The overall student evaluation numeric data were good, on par with other computer applications courses. One free response question asks students to talk about the overall course. This feedback was almost universally positive, as represented by the samples below.

*Q: What did you like best about the course?*

- Ask[ed] our opinions. Tailored class to BUS 101.
- I like how we learned all different skills on the computer and the group work.
- The business related material.
- It helped me understand how to make graphs a lot better.
- We learned how to solve for formulas on Excel and learned how to make movies with audio and pictures.

## 5.2 Instructor Feedback

We asked a sequence of eight free response questions of the linked course instructors after they had completed the semester. The purpose was mainly for improvement of the next course offering, and many addressed specific issues with the courses. But two questions addressed overall evaluation of the course.

*Q: Did those students perform any differently than the other students?*

- With respect to their grades, they did above the class average. However, I would like to point out that, especially in terms of lab work, they were much more proficient with the software we used (in particular, Excel) than most of the other students.

- The stock project required a display and presentation which these students seemed to do a little better. It was hard to say for sure and what they would have done without LEC 104, but I believe it made a difference.
- Their semester projects were more sophisticated and creative than the students who did not participate. For instance, they had more photos of their river sites, they used more graphics, they were more creative, and one student did a video!! It was awesome by the way.

Q: *Would you choose to link your course to LEC 104 if it were an option in the future?*

- I think all the CHE 145 students would benefit from this course. The less time I have to spend teaching them how to use the software, the more we can focus on the concepts and results of experimentation.
- Yes, definitely.
- I hope to continue to link it if that option is possible. I spent much less time giving personal tutorials on Excel. For instance, because some of the students knew how to use it, and showed other students. I was lucky in that all the students in the LEC 104 linked to Zoology were also in my lab class, and they did not need any special help from me to learn the technology of D2L, or other Microsoft programs.

## 6 Conclusion

In the end, we feel confident that we have developed a model that achieves most of the goals we set out to accomplish. It succeeds most strongly in increasing students' *technology skills* in a variety of key areas directly applicable to the courses in which the students are enrolled. It also succeeds, though less strongly, in increasing *student engagement* in the university. The course did not succeed in improving students' skills in *group dynamics*, which is an area we will seek to improve in future offerings.

Along the way, we developed a framework that can be used to create another "link" with minimal effort on the part of the instructors. The course can be customized to the linked course without overburdening the technology instructors. We hope that the techniques we used and the lessons we learned can be applied in a different context, and would be interested to hear of any such efforts.

In the end, however, we are left disappointed and befuddled by one important factor: lack of student enrollment. In the end, it is possible that this will doom this course. We built it... but they did not come.

# **The Root Causes of the Students' Programs Quality Improvement in the TBC Method**

**Syed (Shawon) M. Rahman, Ph. D.**

Assistant Professor, Dept. of Computer Science & Software Engineering

University of Wisconsin – Platteville

1 University Plaza, Platteville, WI 53818, USA

Phone: (608) 342-1625, Fax: (608) 342-1965

Email: Rahmans@uwplatt.edu

## **Abstract**

In our research, we have introduced and implemented a new software development method, Testing Before Coding (TBC), to bring in the benefits of using light-weight software development lifecycle in introductory computer programming courses. TBC improves students' programs quality and makes testing as a fundamental part of programming practice and students' prior knowledge in software lifecycle are not expected. In this paper, we have discussed the root causes or activities that we believe play roles in improving students' program quality.



## 1.0 Introduction

Testing Before Coding (TBC) method is a Light-weight Software Development method (such as Test-driven Development) and it has been applied in several introductory programming courses to improve students' program quality. We have published the TBC method's experimental procedure and results in other papers before (such as [1, 2, 3]). In this paper, we have briefly explained different phases of the TBC method. The TBC method introduces several additional activities or phases compared to the classical (Waterfall) or the XP (eXtreme Programming) methods. We believe these activities play a significant role in improving students' programs quality. In other papers before [1, 2, 3] we have explained how the TBC method has been exercised in five different programming courses and has improved students' programs quality by at least 24%, in terms of black-box testing.

The TBC method introduced the following activities that Waterfall or XP does not acknowledge or to which it pays very little attention:

- ◆ Specify the input and output values of the program; identify boundary values and data types.
- ◆ Create black-box test cases before coding; writing test cases before coding forces the students to understand the requirements better.
- ◆ Students are forced to execute the test suite and test their codes; make testing as an integral part of the program development.
- ◆ During testing phase, students create similar more test cases if any test case fails and execute the whole test suite again to make sure that the recent changes in the codes do not break the program somewhere else.
- ◆ Refactor codes to combine them and apply a couple of iterations that keep the codes as simple as possible and make the program ready for any change that comes along.
- ◆ Create a "working" prototype quickly; the prototype provides a clear indication about the final product.
- ◆ Provide an easy framework and do not need any significant training or a new learning curve for students.

Later in this paper, we have discussed briefly regarding the above activities.

## 2.0 TBC software development method

The TBC method does not require prior knowledge of the software development lifecycle; knowledge of any specific application software or design syntax and semantics; or domain-specific expertise to apply TBC in the program development and improve program quality.

In this section, we have introduced the TBC method and explained its different phases. The TBC method is divided into a number of framework activities or phases (Figure 1). We can classify the TBC method in the following seven phases:

- ◆ Gather and analyze requirements.
- ◆ High-level design.
- ◆ Data modeling and data specifications.
- ◆ Generate test cases and create a test suite.
- ◆ Develop the program.
- ◆ Execute the test suite and test the program.
- ◆ QA and customer evaluations.

### **2.1. Gather and analyze requirements**

During the requirements gathering phase, the developers attempt to identify what information is to be processed, what functions or performances are desired, what system behavior can be expected, what interfaces are to be established, what design constraints exist, and what validation criteria are required to define a successful system. The key requirements of the system need to be identified [4].

If there is any customer involvement in the process, the TBC method requires effective communications between customer and developer. The requirements gathering process is emphasized and alerted specifically on software. To understand the nature of the program(s) to be built, the software engineer must understand the information domain for the software, as well as required functions, behavior, performance, and interface [4]. Requirements must be reviewed with the customer. The requirements document should be clear, unambiguous, understandable, consistent, and complete.

### **2.2. Draw high-level design diagrams**

The developers in the TBC method draw a few high-level design diagrams. The developers make a list of everyone who will interact with the system, what output they will receive, and what input they will provide. They draw the high-level design or context diagram, show all the inputs and outputs in the system, and ignore the inside detail of the system. The developers identify the overall program objectives, available resources, and ultimate product.

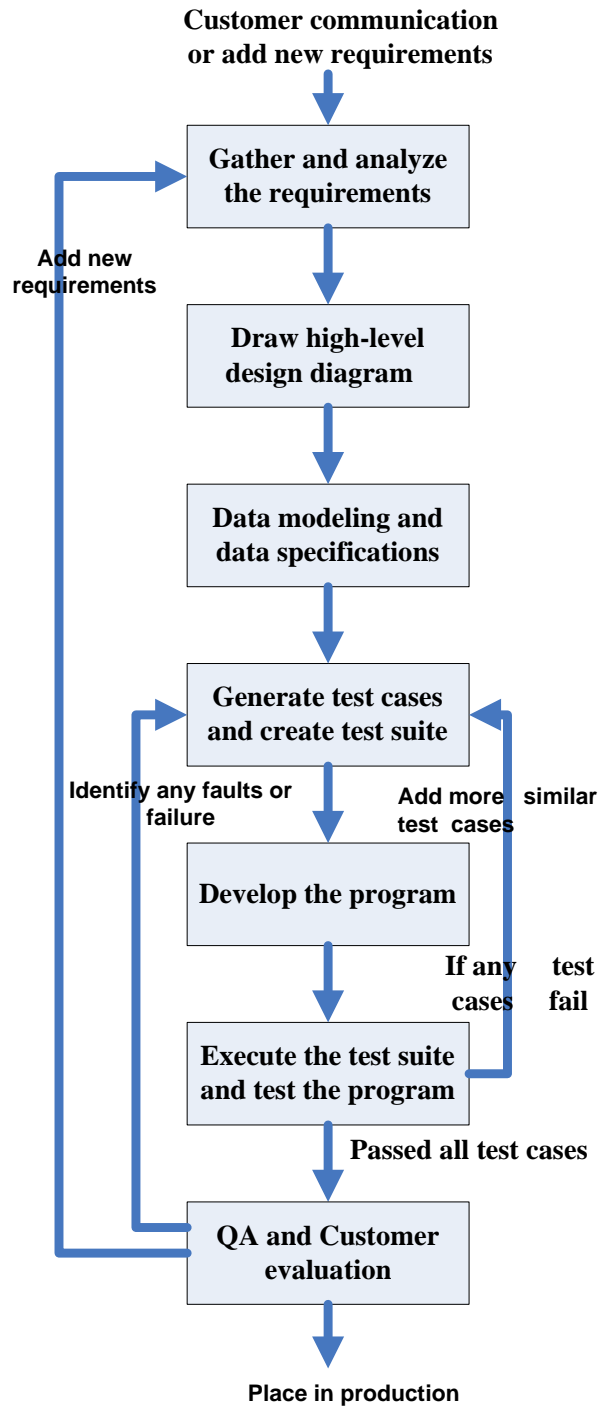


Figure1. The TBC software development lifecycle.

The design phase should discover requirement inconsistencies, missing system components, and unexpected development needs. The TBC model does not spend much time on initial or detailed design phase. However, it focuses on creating test suites, developing codes, executing test suite, and overall testing process.

### **2.3. Data modeling and data specifications**

The TBC method defines data modeling and data specifications. If we get the data model and data specification wrong, our application might not do what users need, might be unreliable, and might fill the database with garbage.

In the data modeling and data specification cycle, the developer answers a few general questions, such as what are the input or output data types, data ranges, boundary values, and maximum or minimum values. The developer checks if there are any possibilities for integer division, division by zero, blank values, or any other exceptional situations. In order to perform data specification or data modeling, the developer must understand the requirements and the system properly. It is an extremely important phase for creating a test suite.

### **2.4. Generate test cases and create a test suite**

The TBC model emphasizes the testing cycles. In the TBC model, the testing process begins before writing code; developers generate test cases and create test suites before writing code. Each developer follows the requirements document, data model, and data specifications to write test cases.

The developer creates the test cases including both normal/valid and abnormal/invalid conditions. The developer intentionally attempts to make things go wrong to determine if things happen when they shouldn't or things don't happen when they should. The test cases make sure that the program can handle any unavoidable or exceptional situations.

### **2.5. Develop the program**

The requirements and high-level design diagrams must be translated into a machine-readable form. The developer writes code to develop the required software or program. The programmers of the TBC model also conduct both the functional and non-functional testing. Developers make sure that their programs are doing what they expected the programs would do, but also that the programs are producing the correct output and a complied version of the code.

### **2.6. Execute the test suite and test the program**

In the TBC model, the developer tests the program extensively and makes sure it passes every single test case before it is released into the production. The developer

executes the program, feeds every test case in the test suite, and records the corresponding test results.

If any test case fails, i.e., the expected output of the program and the actual output are different, the developer goes back to previous steps where he or she generates more related test cases and fixes the program. Once the program is fixed, the developer executes the entire test suite again, including the newly generated test cases used to fix the known bug. The developer also makes sure that the recent changes do not create any more faults or failures in the program.

## **2.7. Quality assurance and customer evaluations**

Once the program passes the test suite, the program is passed to the quality assurance (QA) team. The QA team evaluates the program and makes sure that the program satisfies the customer requirements. The customers, or stakeholders, also evaluate the program before it is placed into production.

If the customer finds any missing requirements or he/she would like to add a new one, the developer takes the program in the initial state where the requirements would be analyzed and reviewed. The developer would follow the entire process by iteration.

## **3.0 Activities that leads to develop higher-quality product**

In Section 2, we discussed how the TBC method acknowledged software different activities in the development lifecycle. The TBC method acknowledged the classical or traditional development method activities, such as requirement gathering and analysis, design, implementation, or coding and testing. However, the TBC method introduced several additional phases that we believe play a significant role in improving software quality and producing higher-quality software at a lower cost.

We introduced the following activities in the TBC method that the Waterfall method does not acknowledge or to which it pays very little attention:

- ◆ Specify the input and output values and identify extreme cases.
- ◆ Create test cases before coding.
- ◆ Developers execute the test suite and tests their codes.
- ◆ Create similar more test cases if any test case fails.
- ◆ Refactor the code and apply a few iterations.
- ◆ Create a “working” prototype quickly.
- ◆ Provide an easy framework and do not need any training.
- ◆ Constant customers’ involvement in the project.

### **3.1 Specify the data and identify the extreme cases**

In the data modeling and data specification cycle in the TBC method, the developer answers a few general questions, such as what are the data type, data range, and the maximum and minimum values. The developer checks whether there is any possibility of integer division, division by zero, blank values, or any other exceptional situations. In order to do data specifications or data modeling, the developers must understand the requirements and the system properly.

Data specifications and identifying the extreme values in the database force the developer to know the system better. Understanding the problem better would be helpful while writing the program and the developers would produce better quality programs. Data specification is also extremely important for creating a test suite. Developers generate test cases for the extreme values and other exceptional cases, such as boundary conditions, outside boundary values, blank input, or no input, etc.

### **3.2. Create test cases before coding**

One of the unique requirements in the TBC method is creating test cases before writing the code. These test cases primarily follow functional black-box testing technique and unit testing technique. The developer follows the requirements documents, context diagram, data model, and data specifications to write test cases for both valid and invalid conditions and creates a test suite. The TBC model emphasizes testing cycles and the developers generate test cases and test suites before writing the codes.

Writing test cases before coding cycle forces the developers to understand the requirements better and to be aware of exceptional values or extreme points where the software would usually fail. Spending more time up front writing the test cases would pay off in the later phases of the development. The developers would develop the program quickly. While developing the program, they would know the requirements and exceptional situations that lead them to create high quality software.

After developing the codes, the testing phase is shorter than traditional development methods. Tester teams would have a complete test suite to test their codes; however, in the industry, the tester teams would further test their codes. We recommend using automated tools in the testing phase, such as JUnit, XUnit, BlueJ, etc. Besides these open source tools, there are many other commercial tools, also available in the market, that industry could use.

### **3.3. Developers execute the test suite and test their codes**

The TBC method forces developers to test their code. Developers are responsible for both white-box and black-box testing. Developers test their code and make sure that their codes are doing what they expected it to do. They execute the test suite, create more similar test cases, and fix the codes. They execute all test cases again until all test cases pass to make sure there are no faults and that the changes do not break the code somewhere else.

After the codes implementation, the codes are transferred to the testing team. Testers in the testing team execute the predefined test suites and perform other testing. For small projects, sometimes the developers who implemented the code and the tester could be the same person. In the testing phase, the main goal would be trying to break the code, making sure that the program can handle any possible odd situation. Other studies found that developers are not very good testers for their own code; they could misunderstand the problem in the first place or not try to break their own codes. A different person might see the problem in different ways and might find new faults in the programs. The testing team can also use some automated testing tools, such as JUnit, XUnit, etc. Besides these free tools, there are many other commercial tools available in the market that industry could use.

Software testing phase in the TBC method is shorter than the traditional or classical software development method. Chaplin [9] expressed that, in the test-driven development, the testing phase would be one-tenth of the traditional method. We do not have any exact data to support our claim; however, we predict that the testing phase will not be as long as Waterfall and that it would be similar to TDD or longer than TDD, but not as long as Waterfall. In the TBC method, the tester will have a complete set of test suites that will be very helpful to the tester team. The code will be less faulty as the process iterates several times and constant customer involvement in the team clarifies the requirements and the problem domain better than the classical development method.

### **3.4. Create similar test cases if any test case fails**

In the TBC method, the developers create a few similar test cases, if any test case fails. It would find the similar problem in the code, if one exists. The developers fix the code and execute the complete test suite once again to make sure that the recent changes in the codes do not break the program somewhere else. In the Waterfall method, the developers are generally reluctant to make any changes in their code; they fear that making changes in one place would break the code in other places. Even if it does break, in the TBC method, the developer would know the exact test case that failed and would fix the program quickly. This process boosts the developers' confidence about their programs correctness and works as a driving force to go forward.

In each of the iterations in the development cycle, the developers execute the entire test suite. If any test case fails, the developers go back to the previous phase where

they generate more similar test cases and fix the code and execute the entire test suite again. The reasons are to find similar bugs in the code and make sure that the program does not break somewhere else.

### **3.5. Refactor the code and apply a few iterations**

When we find two methods that look the same, we refactor the code to combine them. When we find two objects with common functionality, we refactor to make sure there is just one. Refactoring reduces the code volume and cleans up the unnecessary code from the program. Refactoring keeps the codes as simple as possible and ready for any change that comes along. William Wake wrote in his book that through refactoring the codes become clearer, better designed, and of higher-quality [6].

Fowler [5] explained that the entire point of the engineering-based approach is to say, if we get the design right, we won't need to refactor because our code will be just hunky-dory. In reality, of course, it's hard to come up with a design that good. In general, the Waterfall method follows a similar concept that make the “design is right” and implements the software based on the design. But in reality there are many reasons the design could go wrong. For example, the developers could have misunderstood the requirements, or customers could fail to explain in first place, etc. However, refactoring can work with any methodology. There are many developers who use refactoring with the Waterfall-style approach. Whenever we have a substandard body of code that we need to improve, refactoring is a technique to consider [5].

Developers in the TBC method do a few iterations for developing any software and refactor their code. The main goal of refactoring in the TBC method is to improve the internal structures of the code. We can refactor anytime as needed, however, there are three cases we must refactor [5]:

- ◆ If there is duplication.
- ◆ If we perceive that the code and/or its intent is not clear.
- ◆ If we detect problems in the code, whether subtle or not so subtle, that may indicate that there is a problem. It could be a design issue or something else.

The coupling in TBC would be less than in Waterfall, as the requirements in the TBC method are more clear and simplified than in the Waterfall method. The TBC demands frequent customer communications and iterates the cycles that verify or simplify the requirements. We also predict that the TBC code would be highly cohesive, because the TBC method forces the developers to understand the requirements and problems better and would reduce the scattered use of other methods or actions in the code.



As results of this refactoring process, constant customer communications, writing test suites before coding, etc., makes the code cleaner, simpler, highly cohesive, and maintains loose coupling between classes.

### **3.6. Constant customer involvements in the project**

Customer involvement or customer communications is one of the main focuses in the TBC method. Customers engage in different activities with members of the development team. One of the main complaints in the Waterfall method is that the final product might not be the product that the customer wanted. In the Waterfall method, the customer cannot see a prototype or get a clear idea until the project is close to being finished. There is always a risk that the developers did not understand the customer requirements or that the customer couldn't properly explain their needs in the first place. If this were the case, the team could produce something that might not be suitable or different than the customers' expectations.

In the TBC method, customers and developers' frequent communications overcome this problem. Both party's involvement make sure that the developers "develop the product right." Customers are involved in different activities in the development and work side by side with developers verifying the requirements, creating test cases, testing the application, etc. It would be very helpful if the customers had knowledge about the problem domain and a clear idea about the application's requirements. Sometimes the TBC method includes marketing personnel in the requirements analysis phase to obtain a better understanding the requirements and the market demands.

Customer and developer roles are different in the TBC method. They are in the same team, but need to make different decisions. At the early stage of the development, in the requirements phase, the customers provide the requirements and choose the priority of the requirements while developers estimate the work and analyze the requirements. Later in the development phase the developers write the code while the customers write test cases for functional or acceptance testing and answer developers' questions.

The roles of customers and developers in the TBC method would be similar to the XP roles, as Kent Beck [7, 8] and William Wake [6] listed in their books: The customers decide scope (what the system must do), priority (what is most important), composition of releases (what must be in a release for it to be useful), and dates of releases (when the release is needed). On the other hand, the developers decide estimated time (how long it will take to add a feature) technical consequences (developers explains the consequences of technical choice, but the customers make the decision), process (how the team will work), and detailed schedule (when parts will be completed within an iteration).

### **3.7. Create a “working” prototype quickly**

In the TBC method, developers create a quick prototype in their earlier iteration which is verified by the customers. In many cases customers cannot really understand the process or the requirements until they see something workable. The prototype provides a clear indication to the customers about their final products, and developers and customers get an opportunity to make adjustments. However, in the Waterfall method, the customers cannot see a prototype until the project is in very close to being finished and then it is very late to make any adjustments. There is always a risk that the developers would develop software that was different than the customers' needs. To make any change the developers may need to redo the entire development process.

Another potential danger in the Waterfall method is that the developers won't know if the solution is successful until very close to launch, leaving little time and room for correction. Oversights and flawed design work can seriously affect the launch date.

Changes in the requirements in the later stages could be very expensive and in some cases the reason for failing the entire project. However, if we develop software following the TBC method, the developers or customers would see a working prototype in an earlier iteration and have enough time to adjust the project or requirements, if necessary. The next topic in this section, we discuss more about the change of requirements cost activities.

## **4.0 Conclusions**

We have introduced and implemented the Testing Before Coding method that is a subset of eXtreme Programming. Our research has concluded and presented experimental results that the TBC method improves students' program quality and it brings many other benefits. In this paper, we have discussed the reasons why TBC makes a difference compare to Waterfall and XP in terms of students' program quality. We have pinpointed those reasons in our research and explained briefly in this paper.

There are several reasons or activities that play a significant role on applying the TBC method for developing students' programs and improving quality such as students forced to write test cases before writing code which help students understanding the requirements better; make testing as an integral part of development process; and help students while writing the program.

During testing phase, students write similar more test cases, if one test case fails and execute the whole test suite to make sure that fixing one bug did not create more bugs in the program. Students refactor their codes and create a prototype quickly that provide an indication about their final product and it helps for gathering or adjusting requirements.

In our research, we have also measured these activities using software several metrics that play role in improving programming quality and produce higher-quality product.

## References

1. Rahman, Syed; “Applying the TBC Method in Introductory Programming Courses”; IEEE Computer Society and ASEE sponsored conference, The 2007 Frontiers in Education Conference (FIE 2007), Milwaukee, Wisconsin, October 10–13, 2007.
2. Rahman, Syed and Juell, Paul; ‘Applying Software Development Lifecycles in Teaching Introductory Programming Courses’; IEEE Computer Society and ACM SIGSOFT conference, 19th Conference on Software Engineering Education and Training, April 19-21, 2006, Hawaii, USA.
3. Rahman, Syed and Juell, Paul; “Testing Before Coding: A cultural change approach for teaching and developing computer programs”, Association for the Advancement of Computing in Education (AACE) conference; World Conference on Educational Multimedia, Hypermedia & Telecommunications (EdMedia 2006), June 26-June 30, 2006, Orlando, Florida.
4. Pressman, R., S.: Software Engineering: a Practitioner’s Approach, 6th Ed, McGraw Hill, NY, 2005, Pages 467-594.
5. Fowler, M.: “Agile development: what, who, how, and whether,” <http://www.fawcette.com/resources/managingdev/interviews/fowler>, web retrieve on April 24, 2007.
6. Wake, W.: *eXtreme programming explored*, Addison-Wesley, NY, 2002, Pages: 28, 134-152.
7. Beck, K.: *Test-driven development: by example*, Pearson Education, Inc., NJ, 2003, Pages: 27, 123-125, 178.
8. Beck, K.: *eXtreme programming explained: embrace change*, Addison-Wesley, NY, 2000, Pages: 27, 131-133.
9. Chaplin, D.: “Test driven development,” Wednesday, December 17, 2003, <http://www.byte-vision.com/TestDrivenDevelopmentArticle.aspx>, web retrieve on March 6, 2008.

# **RANDOMLY GENERATING WELL-FORMED POSTFIX EXPRESSIONS**

*applying biological processes to computer programming*

Allen Ng  
Computer Science Department  
University of Wisconsin--Parkside  
Kenosha, WI 53141  
ng@cs.uwp.edu

## **ABSTRACT**

A model of biological inheritance is applied to develop a random search algorithm which yields an optimal solution. The underlying biology is presented, and application is made to the design of the software. The genetic algorithm's search speed is compared to a purely random search and shown to be much faster at producing well-formed solutions despite having no information on the criteria for correctness.

# **1.0 INTRODUCTION AND BACKGROUND**

Of all systems in existence, none are as optimized and efficient as those found in nature. Indeed, the human genome is so advanced that even with all our efforts we have only been able to observe and record it; no method to exactly duplicate its functionality has been discovered as yet. While human cloning may still be the stuff of science fiction, there are some observations to be made that can be applied to the technology of today. First, a brief primer on the underlying biology.

## **1.1 Chromosomes**

Until recently, the young science of genetics was virtually unheard of outside the domain of a few, highly specialized scientists. Today, some high-school students can tell you that the chromosomes contained in cells are the building blocks of life. Chromosomes are made up primarily of Deoxyribonucleic Acid, or DNA. The nucleic acid portion of DNA comes in the form of nucleotide bases that encode the information for making the proteins necessary for life.

## **1.2 Mutation and Synapsis**

As cells reproduce, they must obviously make more DNA. Our bodies contain trillions of cells and, though at any one time only a portion of them are reproducing, the amount of DNA being replicated at any one time is still enormous. With so much activity going on in a system so complex, mistakes are inevitable. As car salesmen are fond of pointing out (usually while having you apply for extended warranties): if you make 100,000 copies of anything, a lemon is bound to slip through.

DNA is no exception and can be damaged or in other ways altered by many factors most often while replicating. It is important to note that the exact location of these alterations within the chromosome is totally random. As a result, the effects of these alterations are highly unpredictable. Because of redundancies within the DNA sequence, many changes can occur that go unnoticed as they have no net effect. Given that life depends so heavily on the enzymatic action of proteins and that DNA encodes the instructions for producing those proteins, alterations to the DNA sequence that do produce an effect are usually lethal. From time to time however, by chance, an alteration to the DNA sequence, or mutation, can result in a beneficial effect.

These mutations are important as they provide variety within a species. If DNA were always replicated exactly and with no errors and no variation, then every organism within a species would be identical. More important than the damping effect this would have on social gatherings, this would equate to placing all of one's genetic eggs in one

environmental basket. Anything in the environment that affects one of them, affects all of them. This is a recipe for extinction.

So important are these sequence mutations, that organisms frequently do it on purpose. Organisms that reproduce sexually produce sex cells, called gametes, during a process called meiosis. The gametes contain the parent's genetic contribution to their offspring. During meiosis, an event called synapsis occurs which partially re-arranges the DNA sequence within the gametes resulting in a wide variety of DNA sequences that could be passed on to offspring.

### **1.3 Survival of the fittest**

It is the DNA that is passed on to offspring that determines the offspring's traits. Though previously stated, it is important enough to reiterate: without mutation or synapsis, an organism's offspring would only contain those traits present in itself. It is these alterations in the DNA sequence that allow for occasional improvements to occur. Be it a beak that is more suited to nut cracking or bacteria that are more resistant to toxins (or medicines, depending on your point of view), DNA mutations that help rather than hinder an organism, help to ensure the survival of the organism so that it may reproduce and pass on the mutation to further generations. This weeding out of weaknesses and preserving of strengths is known as survival of the fittest.

## **2.0 APPLICATION AND DEVELOPMENT**

Our goal here will be to apply the genetic model to a computer program in order to develop an algorithm which yields an optimal solution. For simplicity, we will use randomly generated, arithmetic expressions with the intent of generating well-formed, postfix expressions (if you are unfamiliar with postfix notation, see note at end).

### **2.1 Solution encoding**

First, a method of representing the solution as a linear sequence, or chromosome, must be found. Postfix expressions are linear, the methods of manipulating them are well understood, and most importantly, it is very easy to assign a numerical value to them; namely, by evaluating them. Therefore they are well suited to this task. For our purposes, we will deal with expressions containing the digits 0-9 and the arithmetic operations of addition (+), subtraction (-), multiplication (\*), division (/), and taking a modulus (%). In an organism's DNA sequence, there is typically a large percentage of the sequence that does not contain information for protein production. These are referred to as non-coding sequences. Non-coding characters are also allowed in our expressions and, as we have chosen arithmetic expressions to model a chromosome, the letters A-Z

will be used as non-coding characters. It should be noted that a digit or operation could conceivably be a non-coding character as well, if it doesn't make sense in the expression. For instance:

+ 2 2 + \*

contains two non-coding characters (namely, the first "+" and the "\*\*") neither of which is a letter.

It is important to note that since we are randomly generating the initial expressions, our evaluation of these expressions needs to be very fault tolerant. In other words, the presence of non-coding characters will not deter us from evaluating the expression and assigning it a value. Non-coding characters will simply be ignored.

## **2.2 Random variation and cross-over**

The choice of using arithmetic expressions as our chromosome is of benefit here since making random mutations to the sequence is almost trivial. When the sequence is cloned, there is a chance that a mutation will occur, resulting in a random character (a digit, operation, or letter) being substituted into the sequence. Two expressions are also allowed to "mate" to generate a new expression consisting of randomly selected subsequences from each of the parent's sequences. This process is analogous to the synapsis that takes place during meiosis.

## **2.3 Solution evaluation and fitness testing**

Survival of the fittest predicts that those organisms that are the most fit, or have the most advantages, have a higher probability of surviving long enough to reproduce. By reproducing, an organism is able to pass on its traits which, in turn, gives future generations the same advantages and allows the cycle to repeat. However, not all traits are advantageous; only those traits that help an organism to better adapt to its environment represent true advantages.

In order to weed out the weak and preserve the strong, some method is needed to rank our expressions and determine which will pass on their strengths to further generations. An obvious approach is to rank the expressions according to their evaluated value. This does not quite work, however, as a bit of thought reveals that expressions with a large value are not necessarily well-formed (and that is our ultimate goal). We need an environment that does not reward simply ever larger values. To that end, we will define an expression's fitness to be its evaluated value divided by the length of time it took to process the expression.

expression value / time to evaluate

Each chromosome is also examined to determine how well-formed it is. A chromosome's correctness score is calculated by:

$$\# \text{ of operands} / \# \text{ of operations} + 1$$

A well-formed expression will have 1 more operand than operation giving a correctness score of 1. Each non-coding character encountered during expression processing is deducted from either the number of operands or operations as appropriate.

Our choice of a correctness score also reinforces our choice of a fitness score. While proof of this is outside the scope of this paper, it is intuitively obvious that a well-formed expression should have a higher fitness score than a mal-formed one. This is evidenced by the observation that a non-coding character will add to the expression's processing time while not adding to the expression's value (recall how we defined fitness).

Only the most fit of the candidates in each generation are allowed to reproduce (no, this isn't *1984* or *Handmaid's Tale*...they're only arithmetic sequences). The expressions in the top 25% produce 4 offspring each: 1 from mating with a randomly generated chromosome and 3 from mating with the next fit-ordered chromosome.

It is tempting to want to combine the correctness score into the overall fitness score, but not doing so turns out to be more appropriate. Our goal is to use a model of biological inheritance to guide a random search for an optimal solution. We do not wish to simply compare randomly generated expressions until we match a predefined pattern. By leaving the correctness score out of the fitness calculation, we are assured that this is so.

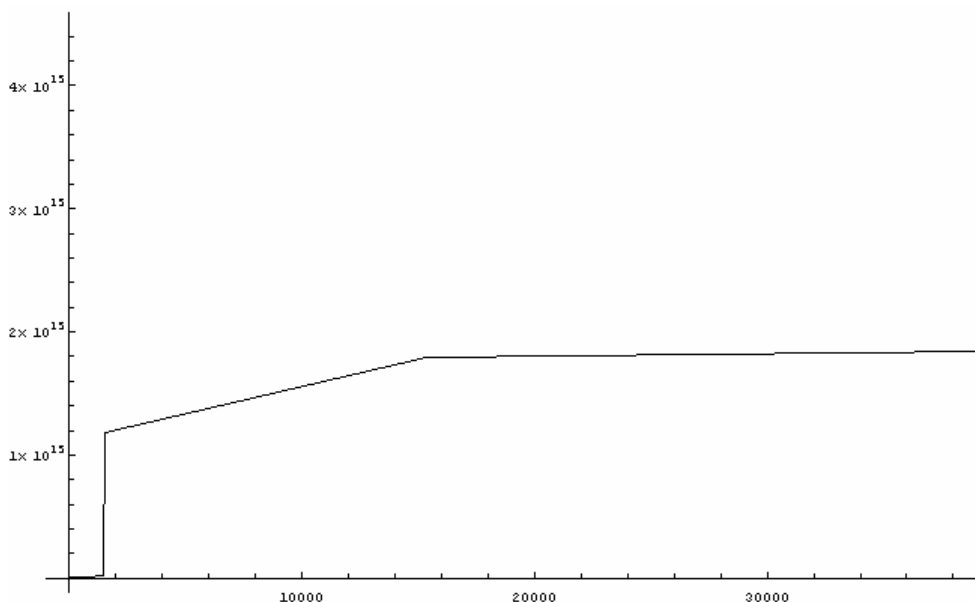


Figure 3.1.1 Fitness scores for successive generations of randomly generated postfix expressions



## 3.0 ANALYSIS AND SUMMARY

### 3.1 Results

Using the genetic model as defined above, successive generations of inherited expressions are observed alongside a control group of expressions generated randomly (without inheritance). The figures shown display a plot of value vs. number of generations, with the fitness associated with a particular generation being the highest fitness observed up to that generation.

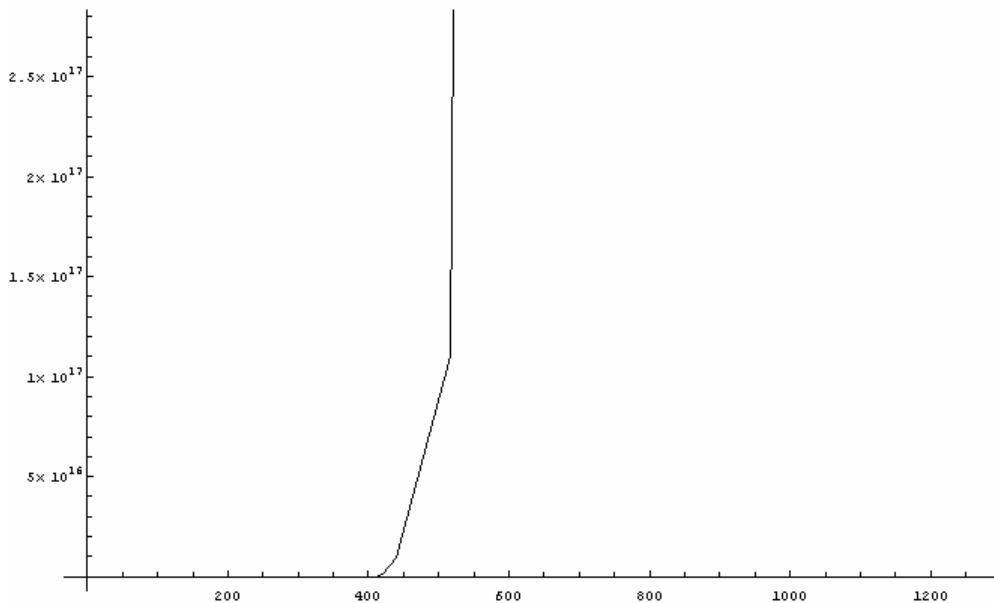


Figure 3.1.2 Fitness scores for successive generations of inherited postfix expressions

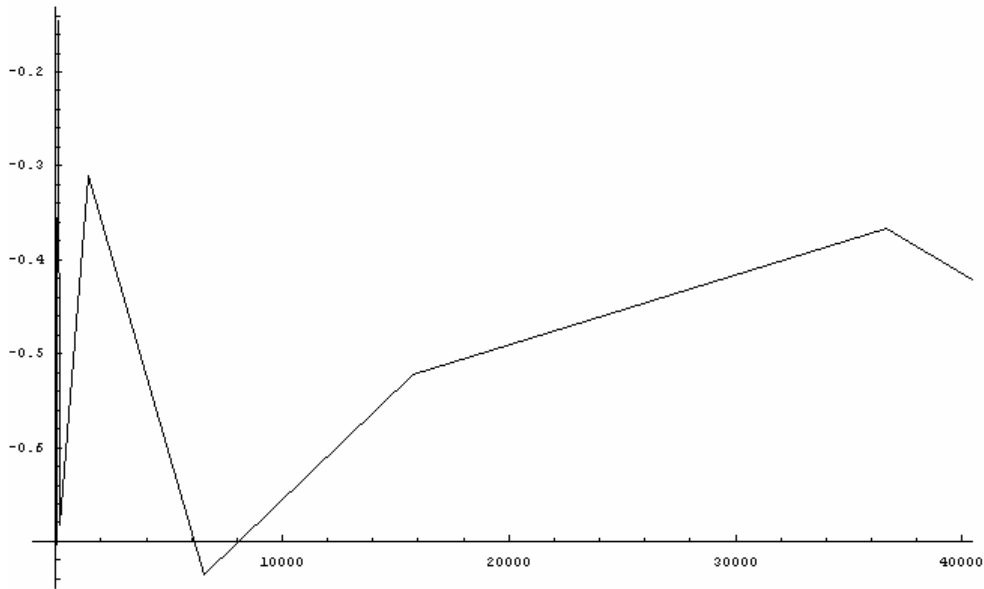
### 3.2 Observations

#### 3.2.1 Fitness scores of inherited expressions compared to random expressions

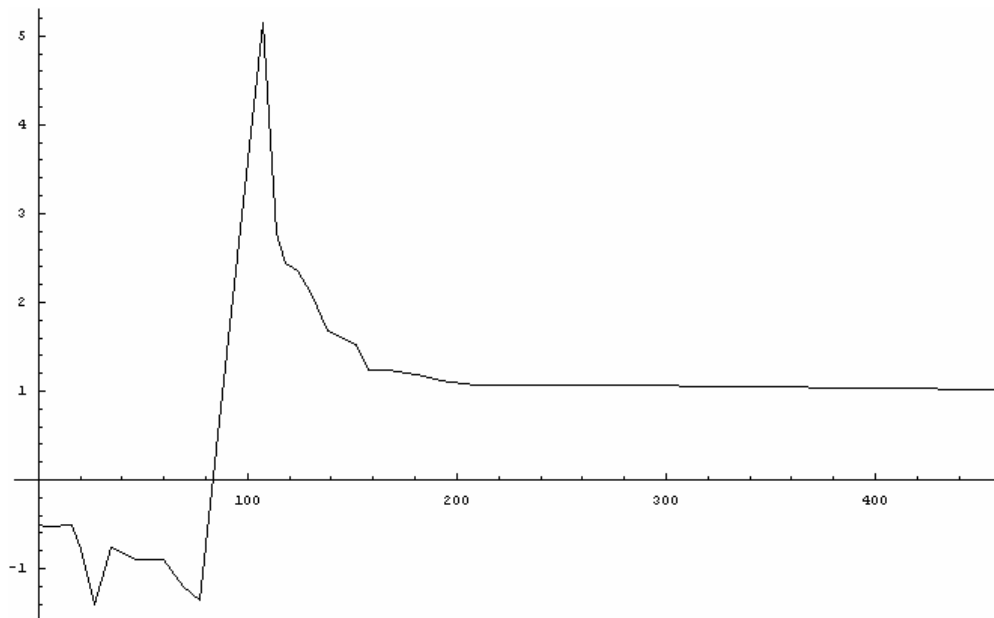
The most obvious observation is to note that the comparison of fitness scores from randomly generated expressions to those from inherited expressions shows a drastic disparity after fewer than 600 generations. In fact, it takes the randomly generated expressions over 10,000 generations to reach the fitness level of inherited expressions from around 400 generations.

Most importantly however is to note that inherited expressions tend toward being well-formed. Figure 3.1.3 shows that even as the fitness scores of randomly generated expressions increase, their correctness scores do not necessarily follow suit. Figure 3.1.4

shows that inherited expressions do indeed tend towards being well-formed (see section 3.2.3).



**Figure 3.1.3 Correctness scores for successive generations of randomly generated postfix expressions**



**Figure 3.1.4 Correctness scores for successive generations of inherited postfix expressions**

### ***3.2.2 Emergence of recurring sub-expression sequences within a population***

Examining the actual sequences from successive generations shows recurring sub-sequences being passed on from generation to generation. This appeals to a sense of symmetry with the biological model, but more importantly shows that sub-sequences that improve overall fitness are more likely to get passed on and sub-sequences that detract from fitness are more likely to get passed over.

### ***3.2.3 Slight alterations in configuration lead to drastic variations in patterns***

Some of the primary parameters of the program (and their values used for this study) include:

- sequence length (100 characters)
- mutation rate (1%)
- cross-over frequency (40%)

These parameters have shown to be very sensitive in that slight alterations in value can lead to drastic variations in output. This is a hallmark of chaotic systems, a subject worthy of treatment in itself.

One example is mutation rate. It was noted that inherited expressions tended towards being well-formed. This observation requires qualification however, as this property seems to depend on a low, but still greater than 0%, mutation rate. For instance, if the mutation rate is changed to 10% or 0%, the inherited expressions no longer tend toward correctness.

## **3.3 Summary**

Our aim here was to show how the application of a biological model to computer programming can produce a random algorithm that yields an optimal solution (or at least tends to one). This problem was intentionally simple so that focus could be given to the process. The way is now clear to apply the process to more complex situations.

As with all science, this success leads to more questions.

1. What types of problems are suitable for a genetic inheritance approach?
2. What is the relationship between mutation rate and form convergence?
3. Are genetic algorithms more efficient than purely random searches? Than hard-coded techniques? What are the trade-offs?

### **3.4 Further reading**

- Goldberg, David. Genetic Algorithms in Search, Optimization, and Machine Learning. Boston: Addison Wesley, 1989.
- Moriarty, David and Risto Miikkulainen. "Discovering Complex Othello Strategies Through Evolutionary Neural Networks."  
<<http://nn.cs.utexas.edu/downloads/papers/moriarty.discovering.pdf>>.

## **A NOTE ON THE DATA**

The reader may have noted a potential loss of precision due to our comparison of floating-point numbers. The software for this research was written in Java. Java's *double* provides around 16 decimal digits of precision. The lengths of the expressions used in this research generate values on the order of  $10^{17}$ . This enables us to resolve values that differ by at least a factor of 10. Observation of the raw data suggests that values in the range used in this research are not greatly affected by this.

Further, the graph in figure 3.1.2 seems to suggest that the inherited expressions improve at an exponential rate. This requires a bit of clarification. The data points included in the graph were the highest fitness scores observed up to that generation. The last point on the graph is merely the highest fitness score observed from all generations.

## POSTFIX NOTATION

The following arithmetic expression should be familiar to most:

$$5 - 2$$

This expression uses what is called *infix* notation, where the operation is *in*-between the operands. *Postfix* notation is nothing more than swapping the position of the operator so that it appears *post*-expression like so:

$$5 2 -$$

The advantage of using postfix rather than infix on a computer is that postfix needs no parenthesis. As an example, take:

$$5 - 2 * 4$$

Depending on which operation you wanted to do first, parenthesis might be required as in:

$$(5 - 2) * 4$$

With postfix, no such ambiguity exists. If the subtraction is to be performed first, we have:

$$5 2 - 4 *$$

and if the multiplication is to be performed first, we likewise have:

$$5 2 4 * -$$

Just like infix, postfix can become malformed. For example:

$$2 + 3 4$$

contains too many operands and so is not well-formed infix. Likewise:

$$+ 2 2 + *$$

is not well-formed postfix. In general, for a postfix expression to be considered well-formed, it must contain 1 more operand than operation and there must be two operands prior to any operation.

# Automated Process for Classifying Text Documents Using K-Means and kNN

Mike Evans, Matt Lietzke, Stephanie Huls, Doug Svendsen  
Computer Science Department  
Saint John's University  
Peter Engel 209, SJU campus  
Computer Science Department  
Collegeville MN 56321  
irahal@csbsju.edu

Monday, May 07, 2007

## ABSTRACT

Correctly classifying text documents among a set number of topics is a difficult task. We will present an automated system using text mining techniques to do just that. This paper presents analysis on documents describing aircraft malfunctions. We investigate the effectiveness of classifying these documents utilizing variations of kNN. The variations of kNN consist of Euclidean Distance and Cosine Similarity for distance metrics. Preprocessing techniques, such as Doc2Mat and IDFs, are used to format the data. Clustering, using K-Means, on the formatted data is used to identify class labels utilized by kNN. Our results showed that both variations of kNN classified the airline documents accurately. Cosine Similarity showed slightly better results. We concluded that our text mining classification system can be used to accurately classify any dataset of text documents.

## BACKGROUND

Text mining is a special subset of data mining; instead of the non-trivial extraction of previously unknown interesting facts from a set of data, text mining extracts these facts from a collection of documents. The key element in text mining is the linking together of the extracted information. This information can be used to form facts or hypotheses to be used or tested by other forms of experimentation. The difference between regular data mining and text mining is text mining patterns are extracted from a natural language text rather than from structured databases of facts, which are designed for programs to process automatically whereas text is written for people to read.

Text mining basically originated from attempts in the fifties to understand the information processing capabilities of the human brain and to model these capabilities. Original approaches analyzed text at the level of individual sentences in order to create a semantic representation of the sentence by using relations between the important words of the sentence. A corresponding semantic construction was associated with each individual sentence. This did provide a good way to understand the meaning of the text. However, there are many different ways an English sentence can be constructed, which led to a list of constructions that grew large very rapidly. As a result, this approach works well only for a limited subset of natural language texts.

A person must keep in mind that text mining is *not* the same as information extraction. An example of information extraction is a program that reads resumes and extracts out people's names, addresses, job skills, etc. Generally these programs have 80+ percent accuracies. The people's names, addresses, etc are *not* new pieces of information. However, some related approaches that pull out information from documents can be turned into data mining, such as finding overall trends in textual data such as looking at crime scene statistics or pulling out key terms in police reports to find overall trends in car theft.

Turning information extraction into text mining is relatively simple. We can extract names of people and companies in texts surrounding the topic of wireless technology to try to conclude who the top participants are in that field. These facts must be interesting and this poses a problem for some approaches where it is difficult to recognize which relationships are truly interesting. People who are familiar with the wireless technology field are more than likely already aware of key players; which makes this information previously known.

Text mining can even be applied to biosciences. For example, researchers in the biosciences are looking for indirect links in different subsets of literature in biosciences to try to come to a hypothesis of the causes of rare diseases. Also it can be used to help discover which proteins interact with other proteins in genomics. In this type of research, text mining is used to try to look for words that co-occur in articles about protein in order to predict interactions. Text mining does not look for direct mentions of pairs, but rather looks for articles that mention the proteins' individual names, record of other words that



occur in articles, and search for other articles containing the same sets of words. However the meanings of the texts and findings are interpreted by humans; not by machines.

Hospitals can also benefit from text mining. Text mining can be used on patient's charts from hospitals in order to discover ways to improve patient outcomes. The University of Louisville in Kentucky has done just this. They have found that some medications prescribed and taken during hospital stays can prolong the hospital stays for patients, especially diabetic patients. Diabetic's blood sugar levels can be altered by certain medications. If the diabetic patient's sugar levels vary, their risk of infection after cardiac surgery increases. A draw back to cases like these, and many other text mining applications, is that each individual hospital records their patient information differently and the application for one hospital's data has to be tweaked or drastically changed to be used on another hospital.

There are limitations of what text mining can do. One of the major limitations of text mining is that we are currently unable to write programs that fully interpret text like what the human mind is capable of doing. Also, the information needed in some cases is not recorded in a textual format. Some needed information may be in the format of spoken conversations, radio shows, television, etc. Characteristics of text mining also pose various problems. Text mining has to deal with very high dimensionality because each word is usually considered a dimension. There is high dependency in text documents. Dependency results from words having a different meaning as a result from the various words used with it and relevant information is usually a complex conjunction of words and phrases. Ambiguity, such as word and semantic ambiguity, is a difficult obstacle has been proven hard to solve even today. Word ambiguity refers to pronouns, such as he and she, synonyms, such as buy and purchase, and words with multiple meanings, such as bat (mammal/baseball bat). Semantic ambiguity refers to sentences that may have multiple meanings such as "The chicken is ready to eat." Noisy data poses another obstacle. Spelling mistakes, abbreviations, slang, and acronyms are examples of noisy data. Also, very informal texts such as emails, "R u available?" are problematic. Mining of historical texts poses its own set of problems. Most text mining tools focus on present-day English and the natural language of many historical text documents are dependent upon when and where they were created.

Some of the overall goals of text mining are improved document classification, automatic semantic explanation of documents, improved searches by semantic and concepts, and improved clustering of documents by concept. [2]

## **PROBLEM DESCRIPTION**

The idea for this project was obtained from the Seventh Annual Data Mining Conference put on by the Society for Industrial and Applied Mathematics (SIAM), which we decided to "non-officially" participate in. In this project, aviation safety reports are being analyzed and the overall goal is to group together the reports depending upon the

problem(s) they describe. Some of the documents describe more than one problem, therefore, having the potential to belong to more than one class; however, no class labels are given.

The dataset obtained contains 21,519 records, with one report per record. All of these reports are in one file, which is the standard text mining format. Each document is about a paragraph long. Refer to Figure 1 for a sample of the dataset.

Unfortunately, the dataset repeatedly had words that ran together, such as “loudnoise” or “engineindicationandcrewalertingsystem.” This posed a huge preprocessing problem. With the frequency of this problem suggests that it is too common for simple typos and may have been introduced by the SIAM for competition. These 'words' were treated as one attribute and eventually were thrown out since they had little frequency of occurrence.

To begin, we will preprocess our data using Doc2Mat, to take care of the errors with the dataset and to format the data so the K-Means algorithm could be used to cluster the documents and to identify the class labels on the training set. These processes will be discussed in further detail later in the preprocessing section of the paper.

Next the classification algorithm K-Nearest Neighbors (kNN) will be used to identify the problem(s) each document in the data refers to and to classify the documents using the class labels produced by K-Means.[1]

Finally we use the results obtained from the training process on the test documents. At this point, the precision, accuracy, recall, and F-1 measurements will be computed so the performance of the classification results can be analyzed.

```
1~AFTER takeoff ON runway _ A
loudnoise WAS hear come FROM
FRONT AREA OF aircraft.FOR A
WHILE I AND CREW THOUGHT IT
WAS THE AIR drive generate THAT
deploy FROM right NOSE OF
aircraft.UPON FURTHER
troubleshoot FOUND THAT THE
AIR drive generate COULD NOT
HAVE deploy DUE TO ABSENCE
OF icon AND message ON THE
engineindicationandcrewalertingsyst
em system.WE immediate return
TO THE airport FOR AN
UNEVENTFUL land.FURTHER
examine AT THE GATE show THE
OXYGEN accesspanel pop OPEN
AFTER takeoff cause THE
NOISE.PRIOR TO flight THE
normalpreflight show NO AJAR OR
OPEN panel ON THE
aircraft.moderateturbulence WAS
encounter AFTER takeoffdue TO
STRONG crosswind AND
lowlevelwindsh earadvisories IN
EFFECT.
```

```
2~taxi OFF THE parkingramp THE
brake system fail TO STOP THE
aircraft...
```

Figure 1: Sample Dataset

## APPROACH

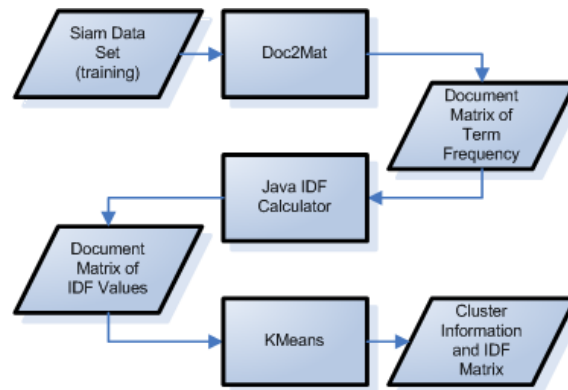


Figure 2: Process Flowchart

## Preprocessing

In our text mining experiment, preprocessing was essential. Our data set contained a high volume of noise with added terms and non relevant data. The preprocessing initiative was a complex set of issues that we dealt with in turn. The first issue was to represent our data as a mathematical model in order to work with frequency and occurrence counts. In doing so, we can reduce the noise produced from common, useless terms. The next step is to normalize our data with respect to each document and adjust for over saturation of relevant terms. This is accomplished by working with Inverse Document Frequency terms. Finally, we must cluster these terms together to discover key words which would describe the dataset as “class” labels.

## Doc2Mat

Doc2mat was the first tool used for preprocessing. This tool transforms our twenty thousand documents into a matrix of documents by term frequency counts. The program starts with the first document and counts the words present; it then marks the appropriate term frequencies. Then, the program proceeds to the following document, updating the first document with empty columns for the new terms discovered. This matrix uses document number as a row reference and the term number for the column heading. The program uses a stop-list which is a built in list of words that are to be excluded from counting. This stop list consists of common words in the English language. Filtering out these words is a common practice for text mining. The process of establishing term frequencies allows the data to be represented as a statistical model. The process for this tool is as follows:

## Algorithm of Doc2Mat

1. Gather user inputs for stop list creation and filename
2. Tokenize text documents and maintain term counts
3. Output results of term frequencies by document

The output contained 20,000 rows which represent documents and approximately 10,000 terms which represent keywords. Each cell value then is the term frequency for that specific document.

## Java IDF

The output from Doc2Mat is the first overall step for preprocessing our data. However, this output, (Term Frequency), is inadequate for analysis and modeling. These terms need to be normalized with respect to the overall dataset. We accomplished this objective by converting term frequencies into Inverse Document Frequency (IDF) values. These values then are again normalized using a sum of squared formula. This section of the preprocessing is done by our own Java IDF calculator.

## Algorithm of Doc2Mat

1. Parse through vector output from Doc2mat and create internal Matrix
2. **For** each term frequency  $t_j$  in a document  $d_k$ 
  - a. Apply IDF Calculation for each term in document.  
$$IDF = \log(\text{DocNum}/\text{TermCard}) * t_j$$
3. **For** each IDF value  $i_j$  in a document  $d_k$ 
  - a. Apply Normalization Formula  
$$i_j / \sum_k (t_j)^2$$
  - b. Store Normalized IDF Value

DocNum is the number of documents in the dataset, and TermCard is the cardinality (or total number of documents in which the term occurs).  $t_j$  is the term frequency for that entry  $j$ .

At this point, our data is generalized for all documents and is the appropriate normalized IDF value. There were several issues operating on this data set. The experiment hardware supplied was inadequate for running our program successfully. We employed the use of the Saint John's Melchior cluster in order to accomplish this goal (See Hardware Section). This data is ready to be clustered for class label extraction.

## KMeans Clustering

With the dataset ready to be clustered, we chose the KMeans clustering algorithm to use on our dataset. The KMeans algorithm “defines a prototype in terms of a centroid, which is usually the mean of a group of points, and is typically applied to objects in a continuous n-dimensional space.”[1] Since our data is normalized and is generally uniform and continuous, this algorithm suited our needs best. The algorithm for KMeans is as follows:

### Algorithm of KMeans

1. Establish K number of points and initialize them randomly
2. **Do**
  - a. Parse through input points and assign each to a specific cluster based on distance metric
  - b. Reset each centroid based on relevant points contained in cluster
3. **While** centroids do not move

In our implementation, we use a random number generator to establish the attributes for our initial centroid values. This process of selecting random start points can potentially lead to misinformation as centroid selection is key to useable output. Therefore we ran this aspect of the preprocessing data several times and correlated the extracted data to better our clustering model output. Also, after our initial calculations using Euclidian distance measure, we decided to implement cosine similarity COSINE SIMILARITY since our vectors were similar. This algorithm on such a large dataset took several hours to per run. We used nine tenths of our data set for training for our K nearest Neighbors algorithm. [5]

## EXPERIMENTATION

After our data had been pre-processed, we were able to perform the actual classification. Our last step of preprocessing, clustering with K-Means, gave us two files to run through the kNN classification algorithm. Unfortunately, there is no way to know if those class labels derived through K-Means are completely accurate, aside from manually checking. However, this is somewhat typical for real world data. On that note, we used kNN to give us our final results, outputting metrics for analysis.

Our K-Means algorithm, described in the preprocessing section, outputted a file consisting of documents and their cluster labels to be used for training and another similar, yet 1/10 the size, file to be used for testing. We decided not to use the test dataset given by the competition simply because it was easier to take in one dataset and

split it apart into a training and test set than make two datasets work together. This had to do with the output of doc2mat and the fact that we would have needed to align the words from one dataset to the other in order to do our distance equations on them.

For all of our preprocessing and processing we used a computer with two dual core AMD Opteron 275 processor's clocked at 2.1 gigahertz along with 8GB of physical memory (RAM). Our programs were all single threaded so they utilized one core on one processor, allowing us to run multiple trials at once.

To do the actual classification, we used the K-Nearest Neighbors algorithm. kNN takes a training set and plots each entry on an n-degree plane. It then takes test entries one at a time and computes the distance from the test entry to every other entry in the plane. Then the average label from the k nearest entries is assigned to the test entry as its classification. We implemented this algorithm ourselves using both cosine similarity and Euclidian distance for our distance measures. Euclidian distance needs no explanation, but cosine similarity may be less familiar. Cosine similarity is a distance metric that in essence turns the two entries to be compared into vectors and measuring the cosine of the angle between them. Cosine Similarity is frequently used as a distance metric between text documents, due to the unique properties of the matrices created by them.

P1 = Document 1

P2 = Document 2

i = One attribute

Euclidean Distance =  $\sqrt{\sum (P1_i - P2_i)^2}$

Cosine Similarity =  $\sum P1_i * P2_i / \sqrt{(\sum P1_i^2 * \sum P2_i^2)}$

### Algorithm of KNN

1. Let K be the number of nearest neighbors and D be the set of training examples
2. **For** each test example z =(x,y), do
  - a. Compute the distance between z and every point in D
  - b. Establish subset of D as nearest neighbors
  - c. Establish class labels based on relevant K Nearest Neighbors[1]

Our kNN algorithm takes in two files, one training and one testing. It then outputs various metrics about how the test entries were classified. These metrics are accuracy, precision, recall and F1 measure. Accuracy is the percentage of correct classification predictions. Precision is the percentage of correct positive predictions. Recall is the percentage of positive entries that were classified. Finally, F1 measure is a combination of precision and recall. While these metrics show that will show that our results are good, with this method, we are completely dependant on the clustering from K-Means, which is not certain to give us good results.

Lastly, we decided to create two datasets from our main dataset, one large and one small. The small dataset was 1/5 of the size of the main dataset, so roughly four thousand documents. The large dataset was 1/2 the main dataset, or roughly ten thousand documents. We did this due to memory and time constraints. We then created random centers, one set for each dataset. Next, we ran kNN with a variety of k values, which will be described in the next session.

## RESULTS

Overall, our results seem promising. Our K-Means implementation gave us clusters that seemed to be well separated, and kNN gave us class labels with decent consistency, as we will show. Assuming that K-Means with Cosine Similarity gave us clusters of truly similar documents; we have achieved a reliable way of automatically classifying text documents.

Our K-Means program was given an input of  $k = 20$ , or 20 clusters to be made. We ran it multiple times and chose the run that appeared to be an average run in terms of the spread of documents across the clusters. We originally had K-Means run using a Euclidean distance metric, but this gave us poor results. Out of the twenty asked for clusters, we on average got less than three clusters. When we implemented Cosine Similarity, a part of our enhancements to the program, our clusters became much more diverse. K-Means took on average an hour to run on the large dataset, twenty minutes on the small one.

One run of K-Means on the large dataset:

- Cluster 0 count: 710
- Cluster 1 count: 770
- Cluster 2 count: 385
- Cluster 3 count: 587
- Cluster 4 count: 531
- Cluster 5 count: 899
- Cluster 6 count: 296
- Cluster 7 count: 486
- Cluster 8 count: 398
- Cluster 9 count: 197
- Cluster 10 count: 579
- Cluster 11 count: 676
- Cluster 12 count: 329
- Cluster 13 count: 666
- Cluster 14 count: 127
- Cluster 15 count: 358
- Cluster 16 count: 881
- Cluster 17 count: 240
- Cluster 18 count: 244
- Cluster 19 count: 359

The goal of our project was to accurately classify text documents. We implemented kNN for this. We ran kNN with both the large and small datasets outputted from K-Means, along with their corresponding test datasets also output from there. We implemented both Euclidean Distance and Cosine Similarity for comparison reasons. In each distance metric, we printed out accuracy, precision, recall and F1 metrics for each cluster, along with their confusion matrices. The attached graphs show the comparisons of both distance metrics at various k's. Surprisingly, both distance metrics achieved strikingly similar results, but of note, not exactly the same results. Cosine Similarity was a percent or two better in almost all instances. These results are somewhat surprising; we expected the Euclidean classification to perform much worse than Cosine Similarity. Either way, our results were quite good. Below are some graph comparisons using differing values of k.

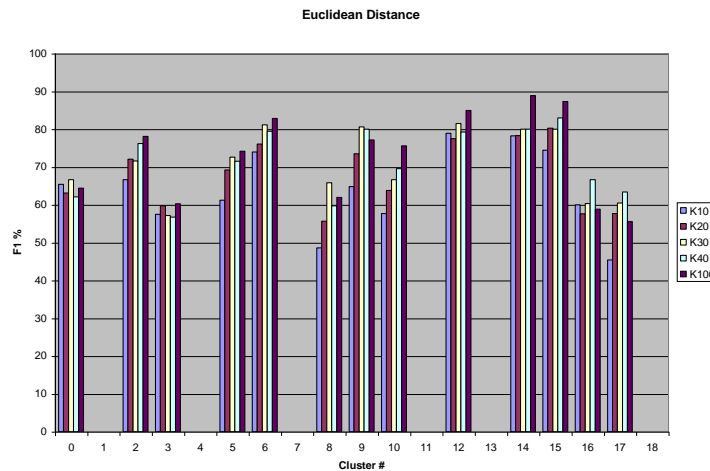


Figure 3: Euclidian Distance Calculations

Figure 3 is a comparison of the F1 measures in each of the clusters for the 5 runs of kNN we performed using Euclidean Distance. The measure kept getting better with increasing K's, signaling that the clusters were most likely well separated. Each cluster in the dataset used on this had at least a hundred documents in it, but computation time restraints kept us from running a higher K.



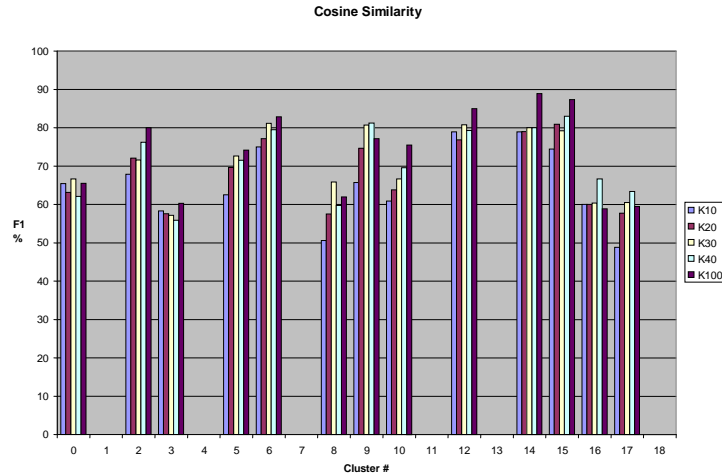


Figure 4: Cosine Similarity Distance

Figure 4 also shows a comparison among the various runs using F1 measure, only this time it is with Cosine Similarity. Again, on average, scores got better as k values went up. Next up is a comparison of the two different distance metrics.

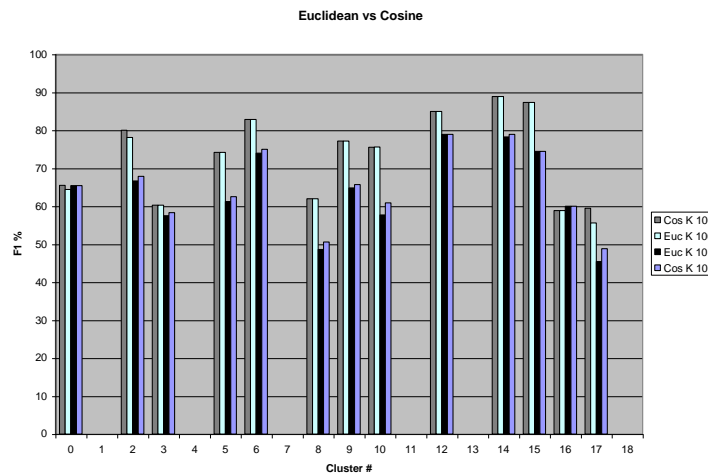


Figure 5: Euclidian vs Cosine

In this comparison chart, Figure 5, you can see that both metrics came up with similar results. We found this to be fairly surprising, as we had heard that Cosine Similarity would function much better than Euclidean Distance. It is probable that through our preprocessing we derived clusters that were well-separated enough to reduce the differences between the two metrics.

Lastly, this next figure, Figure 6, is a comparison of the two distance metrics on a dataset that was twice as large as the original. This chart cannot be compared to the others, since the clusters for this run are different than the ones for the smaller dataset run.

The clustering with this larger dataset worked very well when compared to the smaller. Only one cluster had an F1 measure of 0, where there were many of those in the previous dataset. The average F1 measure over all clusters was higher. This was most likely the case due to the larger model that was created, which, if the documents were well separated, would only help our kNN. We believe we could have had even better results on this dataset if we increased the k to 200 or 300, since each cluster had over 250 documents in it. However this would take prohibitively long to compute.

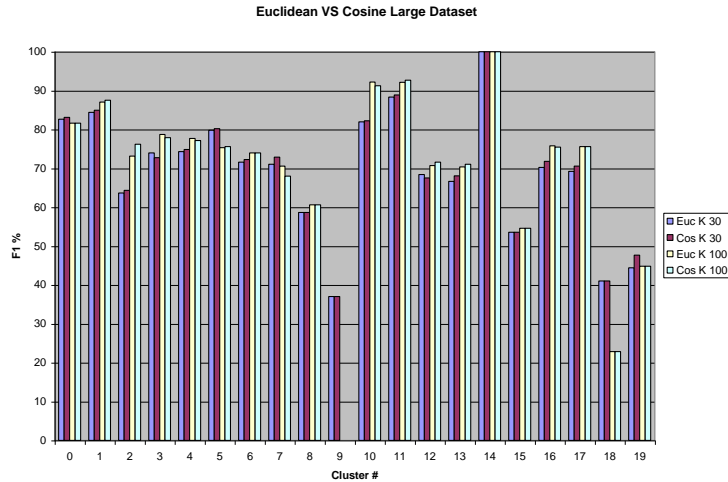


Figure 6: Large Dataset Distances Comparison

## MODIFICATIONS

Our team decided to modify our text mining experiment by changing our distance metric. We began by using Euclidean distance to measure the distance between vectors in our implementation of K-Means. Using Euclidean distance did not yield very good results, so we decided to implement Cosine Similarity as well. Through using Cosine Similarity we were able to cluster documents slightly more accurately.

In addition to utilizing a different distance metric, we also chose to implement our own clustering algorithm to identify class labels for our data, rather than utilizing the external Weka software. By doing this the entire process could be automated from preprocessing to classification using kNN. This allowed our team to run tests quicker and free of potential human error.

## FUTURE WORK

Future work on this data set could utilize additional distance metrics, beyond Euclidean Distance and Cosine Similarity. For example, Chebyshev Distance and Manhattan Distance could both be implemented. Chebyshev Distance defined as  $\max |p_k - q_k|$  returns the maximum difference between two attributes of a vector. When applied to the text dataset studied in this experiment Chebyshev would produce the maximum difference a

word appears between two documents. The results using Chebyshev Distance would be resistant to outliers. Outlier resistance is important in text mining, because words carry meaning with them. The difference of one word between two documents could change their entire meanings. The goal is to find neighbors to a document which have similar symptoms leading to a problem. An outlier word could potentially cause an incorrect classification.

In addition to Chebyshev Distance, Manhattan Distance could be implemented. In this metric, the absolute value of difference of all corresponding attributes between two vectors is summed. This metric is also resistant to outlier. This metric may be helpful in finding correct classifications, because the distance will be significantly increased when a document has words which are not found in the document it is being compared with. Again, this is important because the existence of a word in a document can change its meaning.

Research which could extend directly from the implementation in this study would be to add an implementation of the Shared Nearest Neighbors (SNN) clustering algorithm. SNN involves finding the number of nearest neighbors two vectors have. This first step in the SNN algorithm is to find the k-nearest neighbors of all points. Since we implement kNN in our study, the extension to SNN merely involves finding the number of nearest neighbors two vectors share if they are among the k-nearest neighbors of one another and assigning that value to an edge between those two vectors. If the vectors are not nearest neighbors of one another, then a 0 is assigned to an edge between them. Vectors can then be clustered together based upon a threshold number of shared nearest neighbors. These clusters could be based upon the class labels identified in kNN. SNN can be useful, because it works well to find clusters datasets of varying densities. This could potentially give further insight into the dataset.[1]

## CONCLUSION

In this project, we created an application which classifies documents about aviation problem(s). After preprocessing the data using Doc2Mat and finding the IDF values for each word in the document, class labels were identified by clustering the document on these values. Our clustering algorithm was an implementation of K-Means. Once the class labels were found, classification could be run on the given dataset. The classification algorithm implemented was kNN. Two distance metrics were used in the implementation of kNN: Cosine Similarity and Euclidean Distance. Both distance metrics gave similar results; however, Cosine Similarity produced slightly more accurate results. From these results we can conclude that our algorithms correctly classified the documents. We now have a completely automated process for classifying taking a dataset of text documents given no class labels.

## REFERENCES

- [1] Introduction to Data Mining by PN Tan, M Steinbach and V Kumar (ISBN 0-321-32136-7)
- [2] Fan, Weiguo, Linda Wallace, Stephanie Rich, and Zhongju Zhang. "Tapping the Power of Text Mining." Communications of the ACM 49.9 (2006) : 76 - 82
- [3] Arora, Ritu, and Purushotham Bangalore. "Text Mining: Classification \& Clustering of Articles Related to Sports." ACM-SE 43: Proceedings of the 43rd Annual Southeast Regional Conference. Kennesaw, Georgia, .
- [4] Hotho, Andreas, Andreas Nürnberger, and Gerhard Paaß. "A Brief Survey of Text Mining." LDV Forum - GLDV Journal for Computational Linguistics and Language Technology 20.1 (2005): 19-62.
- [5] Raymond Y.K. Lau. "Context-sensitive text mining and belief revision for intelligent information retrieval on the web." Centre for Information Technology Innovation, Faculty of Information Technology, Queensland University of Technology, GPO Box 2434, Brisbane, Qld 4001, Australia
- [6] Larsen, Bjornar, and Chinatsu Aone. "Fast and Effective Text Mining using Linear-Time Document Clustering." KDD '99: Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. San Diego, California, United States,

# Sectioning points into fixed-size sections

Daniel Edwins  
MSCS Department  
St. Olaf College  
Northfield, MN 55057  
edwinsd@stolaf.edu

Luke Schlather  
MSCS Department  
St. Olaf College  
Northfield, MN 55057  
schlathe@stolaf.edu

Olaf Hall-Holt  
MSCS Department  
St. Olaf College  
Northfield, MN 55057  
olaf@stolaf.edu

## **Abstract**

The Minnesota State High School League needs an effective method to separate schools in a given athletic division into a fixed number of sections. This problem is different from other clustering problems because of the fixed number of sections. Currently, these sections are chosen by hand. Certain schools are greatly disadvantaged, having to travel much farther than their neighbors. In our research, we have devised a method to minimize the traveling distance in each section, which in turn, minimizes the total traveling distance for the division as a whole. Through this research we propose a sectioning based on this metric.

Although clustering problems like this are NP-hard, we are confident that we have found the global minimum with our polynomial time algorithm. If needed, we can test each sectioning with our objective function, which returns the best sectioning based on our metric.

# 1 Background

Most sectioning algorithms tend to focus on creating good sections of whatever size the program deems appropriate. Our algorithm, in contrast, requires that all groups maintain a fixed size. For this general situation, many algorithms rely on a fixed point to use as a mediod or centroid for the cluster. Other algorithms, such as the DBSCAN (Density-Based Spatial Clustering of Applications with Noise) algorithm, require that each point has a certain number of other points within a certain radius [2] [3]. Our algorithm takes a novel approach, and analyzes all distances involved in each cluster.

# 2 Quality

Our research began with a search for an objective measure for the quality of a given graph. For our purposes, we sought to minimize the total driving done by the sports teams at the various schools. Given an objective value representing the cost of the graph, we can determine, if nothing else, which of a group of sectionings is most practical.

For the purposes of the Minnesota League, we looked at 96 schools in a division, and how to divide them into eight sections. To compute the distance, we used the schools' geographic coordinates, and computed the distance between them by the difference between their geographic coordinates as if the coordinates were in a cartesian plane. Our current system uses T. Vincenty's formula for the computation of distances in an ellipsoidal model of the earth [4]. A more accurate system might query Google Maps or Mapquest to find the distance by road between the schools, or even compare driving times. In any event, the particulars of determining distances are not radically difficult to implement. The algorithm described in this paper assumes a matrix exists providing the distance between any two given points in the graph to be sectioned.

Given this matrix, we analyzed two methods to determine a value to describe the total amount of driving a sectioning requires as compared to other sectionings. One followed the K-means approach, minimizing the distance to a centroid [1]. The second used the complete graph of each cluster, adding the distances between every point in a section together. The K-means approach, though initially attractive for efficiency, proved less effective, and also less efficient over the course of the algorithm.

Therefore, the quality  $d_g$  for an individual section is defined as the sum of the lengths of all of the edges in a graph representing the distances between a group of points. Computing this value takes quadratic time with respect to the number of points in a section  $p$ , given that the number of edges between  $p$  points, given a graph showing all possible edges, is quadratic with respect to  $p$ . To determine a total quality  $d_G$  for a given sectioning of a graph, we then sum the individual qualities  $d_1 + d_2 + \dots + d_g = d_G$  for  $g$  sections. Given this equation, combined with the quadratic nature of finding an individual  $d$ , the total quality  $d_G$  for a graph of  $n$  points may be found in time  $\Theta(\frac{n^2}{g})$ .

$$d_G = d_1 + d_2 + \dots + d_g \quad (1)$$

### 3 Proposed Algorithm

This quality function forms the basis of our proposed sectioning algorithm. Suppose we are going to section  $p$  points into  $g$  sections. First, each point is randomly assigned to a section. Then through a two-point rotation (i.e. a swap), these two points trade sections. If point  $p_1$  was originally assigned section  $g_1$  and  $p_2$  was assigned  $g_2$ , after the rotation,  $p_1$  is assigned section  $g_2$  and  $p_2$  is assigned section  $g_1$ . This process is continued exhaustively. Our algorithm swaps every point with every other point until we reach a local minimum. To see how this is done, refer to Algorithm 1.

---

**Algorithm 1** *exhaustiveSwap()*

---

```

 $d_{G_i} = \text{quality}()$  {initial quality}
 $d_{G_c} = 0$  {current quality}
while  $d_{G_c} < d_{G_i}$  do
  for all  $p_1 \in P$  do
    for all  $p_2 \in P$  do
       $\text{swap}(\phi(p_1), \phi(p_2))$ 
    end for
  end for
   $d_{G_c} = \text{quality}()$ 
end while

```

---

In our test runs, the two-point rotation function gives a wide spread of local minima. To “jump” out of these local minima, we decided to allow more rotations, such as three-point rotations. In practice, rotations of more than three points become unfeasible. The new process performs all possible two-point rotations, then all possible three-point rotations and repeats these two steps until a local minimum is reached.

This algorithm does the following (refer to Algorithm 2). Assign every point a random section. Store the initial quality as calculated by the quality function. Set the current quality to 0. While the current quality is less than the initial quality, perform all possible two-point rotations and then all possible three-point rotations. Then calculate the current quality and go through the while loop again until a local minimum is reached.

In order to eliminate bias when performing these exhaustive rotations, we have a  $\phi$  function that randomly chooses the starting point for the exhaustive rotations for each iteration of the while loop so that the rotations are not performed in the same order every single time.

---

**Algorithm 2** *newExhaustiveRotation()*

---

```
 $d_{G_i} = \text{quality}()$  {initial quality}  
 $d_{G_c} = 0$  {current quality}  
while  $d_{G_c} < d_{G_i}$  do  
  for all  $p_1 \in P$  do  
    for all  $p_2 \in P$  do  
       $\text{rotate}(\phi(p_1), \phi(p_2))$  {same as swap() function}  
    end for  
  end for  
  for all  $p_1 \in P$  do  
    for all  $p_2 \in P$  do  
      for all  $p_3 \in P$  do  
         $\text{rotate}(\phi(p_1), \phi(p_2), \phi(p_3))$   
      end for  
    end for  
  end for  
   $d_{G_c} = \text{quality}()$   
end while
```

---

### 3.1 Complexity

The complexity of Algorithm 2 is  $\Theta(\frac{kp^4}{g})$ , where  $p$  is the number of points and  $g$  is the number of sections. The value  $k$  is the number of iterations of the while loop, and from our tests, we have never seen  $k$  greater than 5. The quality function has a complexity of  $\Theta(\frac{p}{g})$ .

## 4 Results

We have broken our original data into two datasets: the first dataset uses the euclidean distance measure and the second dataset uses T. Vincenty's formula for distance.

### 4.1 First Dataset: Euclidean distance

Once Algorithm 2 reaches a local minimum, we cannot determine if the local minimum is a global minimum. We have a relatively small sample size in comparison to the millions of possible initial sectionings we could use for the algorithm. However, we ran four tests, each of which ran the algorithm repeatedly on different initial random sectionings. In total, we ran the algorithm on 700 different initial sectionings, and each one of these four tests resulted in the algorithm producing the same value we believe to be the best quality possible for this graph. Also, with each additional run, the number of local minima decreased relative to the total number of initial sectionings used.



Number of Runs	Number of Distinct Local Minima	Best Quality	Frequency of Best Quality
300	165	552.396	20
100	76	552.396	4
100	71	552.396	10
100	69	552.396	9
<b>700</b>	<b>289</b>	<b>552.396</b>	<b>51</b>

Table 1: Results from our test runs.

## 4.2 Second Dataset: Geodesic distance

After 107 runs with different initial sectionings, our algorithm arrived at the same local minimum each time:  $2.47784 \times 10^4$ . This value is twice the sum of every path between two points in a group in each group of the sectioning. Thus, the average distance between two schools is 24.42 km by this value. We are not as confident that this is the global minimum.

We would like to thank our algorithms class for their support and Professor Weimerskirch for bringing the problem to our attention.

## References

- [1] K. Alsabti, S. Ranka, and V. Singh. An efficient k-means clustering algorithm. *First Workshop on High-Performance Data Mining*, 1998.
- [2] M. Ester, H. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. *Published in Proceedings of 2nd International Conference on Knowledge Discovery and Data Mining*, 1996.
- [3] M. Halkidi, Y. Batistakis, and M. Vazirgiannis. Clustering algorithms and validity measures. *Proceedings of the 13th International Conference on Scientific and Statistical Database Management*, 2001.
- [4] T. Vincenty. Direct and inverse solutions of geodesics on the ellipsoid with application of nested equations. *Survey Review*, XXII, 176, 1975.

# **Enhancing the Price/Performance of a Clustered Multiprocessor System**

Joseph Myre      Daniel Ernst  
Department of Computer Science  
The University of Wisconsin - Eau Claire  
Eau Claire, WI 54701  
{myrejm, ernstdj}@uwec.edu

## **Abstract**

In this paper, we will explore the design considerations of a low cost, high performance microprocessor cluster, including both hardware and software factors. The goal of the designed system is to maximize the ratio of performance to price, creating a powerful computer at very low price point. This paper will explore the target capabilities of the platform, the design process used to select components and configurations, and metrics by which microprocessor clusters can be evaluated.

## 1. Introduction

Historically, a researcher wishing to perform computationally demanding studies has required access to a high performance supercomputer. For institutions with a grand problem to solve, investing in a large parallel system was often the best (and only) way to achieve the capabilities that they desired.

Unfortunately, access to this much computing power was traditionally only available to those with hefty budgets. Traditional “supercomputers” often involve a lot of specialized parts, including custom microprocessors, enormous memory banks, and elaborate network topologies. Therefore, it is no surprise that the institutions which showcase the fastest machines are often only government labs [11], mega-corporations, or large research centers [10]. In addition, given the high cost of these custom parts and the low volumes that these companies deal in, even smaller versions of these machines are prohibitively expensive for anyone for whom it does not provide a major capability advantage.

However, with the advances in performance of off the shelf hardware, clustered systems of commodity microprocessors have become an extremely popular alternative for institutions where both price and performance are concerns. A cluster is defined as a scalable architecture based on commodity hardware, a private system network, and usually running on open-source software [4]. The benefit provided by clusters is twofold: First, by using off-the-shelf parts and free software, cost can be greatly mitigated. Second, clusters enable institutions without the benefit of extraordinary budgets to establish their own modestly scalable high-performance computing (HPC) environment without breaking the bank.

Recently, a group at Calvin College designed and built Microwulf, a personal, portable Beowulf cluster, which provides over 26 billion floating point operations per second (Gflops) of measured performance at a total cost of less than \$2500 [1]. This level of performance attained by Microwulf was one of the first to break the \$100 per Gflop barrier at \$94 per Gflop. Given the natural decay of prices over time, Microwulf could be constructed for roughly half the cost today. Microwulf, along with these other projects, provide plentiful evidence that market trends are making the construction and usage of clusters progressively more accessible.

It is the primary objective of this research to explore the means and metrics by which low-cost microprocessor clusters can be constructed and evaluated for academic use. This includes looking at price and performance numbers for components, installing and running industry standard evaluation utilities, as well as experimenting with machine configurations to determine how various hardware and network configurations impact performance. In order to accomplish this goal, we have designed and implemented our own low-cost, high-performance computational cluster.

The remainder of this paper is organized as follows. Section 2 outlines our approach for

designing the system and selecting components. Section 3 describes our experiences with the hardware and software issues that arose during the implementation of our cluster. Our evaluation metrics for the cluster is explained in Section 4. Section 5 presents the cluster's performance and cost efficiency results. Section 6 proposes some future directions and conclusions.

## **2. System Design**

### **2.1 System Goals and Criteria**

There are many considerations to account for when designing any computing system, including clusters. The first issue to address is the eventual cost of the cluster. Through a differential tuition grant, our project was allocated \$3000 to complete our system. This limitation forced us to be very selective about what portions of the machine we spent money on.

The second issue to address is to determine the purpose of the final system. In our case, we hope to use our resulting system for both teaching purposes and as a shared research resource. In particular, both avenues of usage ("toy" multi-threaded applications and architectural simulations) will entail highly CPU-bound computation with fairly small and/or non-critical network requirements. This goal will help us prioritize where we spend money within our budget.

In addition, our department has an acute lack of available space to house our cluster, so our design should be as space efficient as possible. Ideally, the machine should be able to sit on or under a desk in a faculty office, if so desired.

Combining the above goals, we came up with the following criteria for our system:

1. Since processor performance is our most important metric, our budgetary resources will be focused on getting the best processor possible.
2. To leave as many resources as possible for processing power, all other components will be budgeted only at a level which is sufficient to not limit processor performance.
3. The physical size of the system should be minimized, if possible. (*i.e.* choose small form-factor parts when available)

### **2.2 Analysis of Available Materials**

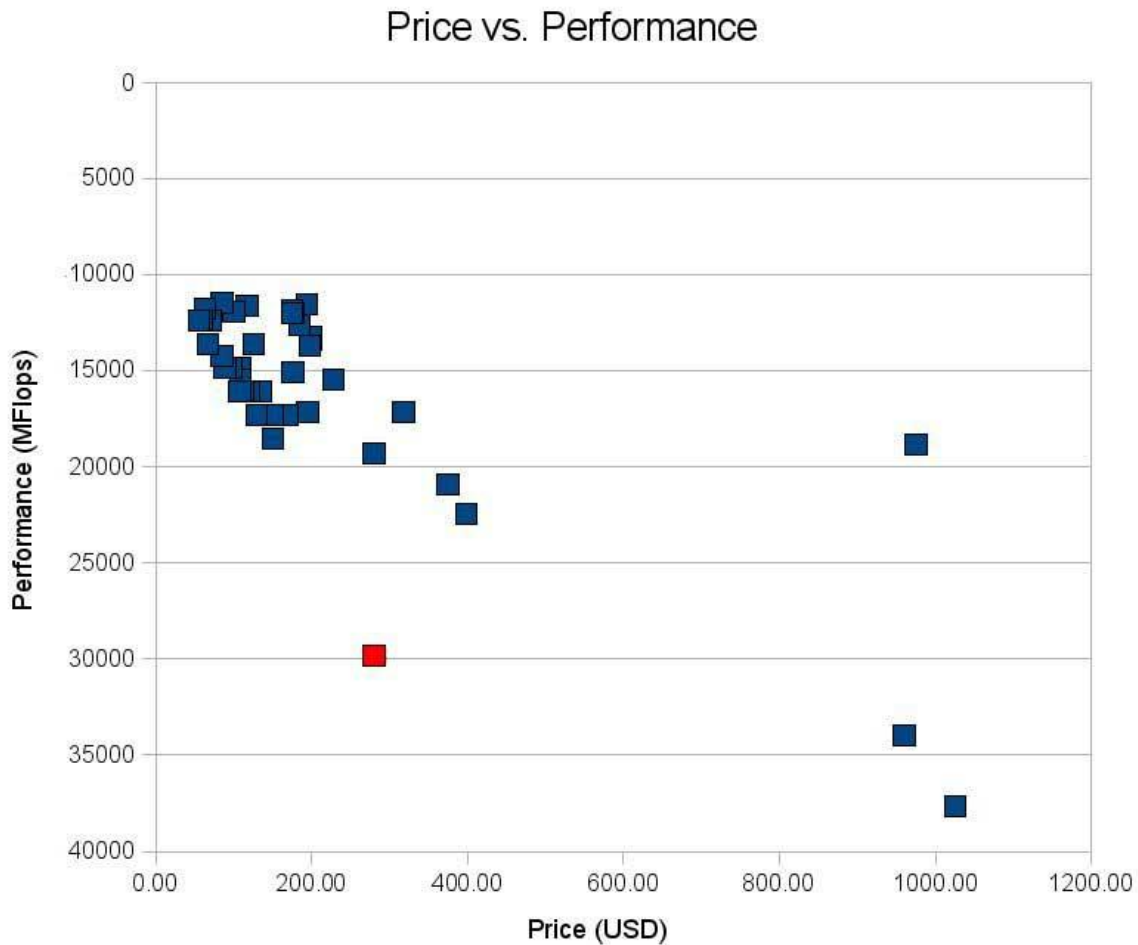
#### **2.2.1 Microprocessors**

The long march of Moore's Law over the last three decades has steadily packed more performance into individual microprocessors. In recent years, the advances in chip technology have changed from more aggressive uni-processor designs and systems to

providing multiple cores (processors, essentially) on each individual chip. While these advances don't do much for single-threaded programs, they are a great boon for multithreaded workloads intended for multiple processors. In addition, multi-core processors, because they are held within a single package, cost far less than two more powerful single-core processors. Therefore, for the purpose of our cluster, we will get the largest amount of compute performance at the lowest price by maximizing the number of cores in our processors (assuming that each core has reasonable performance).

The speed at which Moore's Law advances processor technology also aids us in finding affordable options that still have high performance. Because new processors are being developed and released at very high rates, they drop in price relatively quickly, causing the performance difference between brand new chips and those out only 3-4 months to be very close in overall performance.

We catalogued the processor options available to us in October 2007 and plotted them on the scatter plot shown in Figure 1.



**Figure 1: Pareto analysis scatter plot of microprocessor performance versus price.**

In order to find an optimal microprocessor to meet our dual goals of high performance and low price, Pareto analysis can be used. Pareto analysis is a data analysis technique which eliminates sets of data such that the remaining data points are optimal given a set of criteria. In this case, the data eliminated will be microprocessors that are either lacking in computational power (up on the chart) or are too expensive to fit our budget (to the right on the chart). This analysis allows us to quickly analyze the wide range of available processors on the market visually as well as quantitatively.

Figure 1 reveals a very obvious microprocessor choice. The one chosen was that with the highest amount of performance per unit cost, a 2.4 GHz Intel Core2 Quad Q6600. In Figure 1 this microprocessor is represented by the data point at (\$279, 29823). This selection was near trivial since that microprocessor provided performance that was near that of microprocessors with almost four times the cost. Similarly, the selected microprocessor provided almost 1.5 - 2 times the performance of microprocessors near its own cost.

This processor was one of the very first 4-core processors released, and so was also the first to drop drastically in price. Because it was the only quad-core processor among its peers in cost, it was a clear choice.

### **2.2.2 Memory and Network**

When maximizing machine performance for the general case, care must be taken when selecting other components to avoid damaging performance. As far back as 1970, Gene Amdahl [2] observed that a computer system must provide sufficient resources in memory and network I/O to provide a balance to the CPU performance. This balance should ensure that the maximum amount of processing power is available in the system, but also requires the remaining components be at a level that doesn't inhibit the processing power through starvation.

A long accepted rule-of-thumb that expresses this balance says that, for every Hz of processor speed, a system needs one byte of RAM and 1 bit-per-second of network bandwidth [2]. Extrapolating out this rule of thumb for our chosen "sweet-spot" processor dictates that we need  $2.4 \times 10^9 * 4 \text{ cores} \approx 10 \text{ GB}$  of RAM and 10 Gbits of network bandwidth per node.

Putting 10 GB of RAM (or even 8) on each node is not very feasible, as the largest single-DIMM RAM size available is 2GB, and we'd prefer to not limit our motherboard choices to only those with 4 memory slots. Placing 4 GB on each node would also cost roughly as much per node as the processor. Analyzing our budget options, we were therefore presented with a choice between using 4 nodes, each with 4 GB of RAM, or 6 nodes, each with 2 GB.

To determine how far away from the "rule of thumb" we were willing to go, we returned to analyze our initial goal applications for the cluster. All of these applications rely heavily on processor performance and each uses easily less than 256 MB of RAM per

thread. We therefore opted to provide more total nodes, each with 2 GB of memory.

We faced the same dilemma with network bandwidth. As 10 Gbit Ethernet is still very new (and extremely expensive) and Myrinet technologies [9] are also cost-prohibitive, we chose to use standard Gigabit Ethernet to connect our nodes together. Because our target applications do not depend heavily on network bandwidth or latency and as GigE is the standard interface included on most motherboards, this was an easy choice.

Most implementations of MPI support the use of multiple network interface cards on each node, successfully multiplexing traffic across both interfaces. Our budget had room for an additional GigE card per node, however, because our network usage estimate was low, we chose to not make that purchase. In the future, if more bandwidth is needed, adding the cards will be a simple upgrade.

### 2.2.3 Other Parts

Most of the remaining parts were chosen based on simple functionality constraints. The full parts list is shown in Figure 2.

The one remaining “interesting” choice is that of the motherboard. In order to satisfy our size criterion, we chose to use a micro-ATX small form-factor motherboard from GIGABYTE.

Part Name	Price	Qty.
Intel Core 2 Quad Q6600 2.4GHz	\$279.99	6
GIGABYTE GA-G31MX-S2 Micro ATX Motherboard	\$76.99	6
Rosewill RV350 ATX 1.3 350W Power Supply	\$20.99	6
Transcend JETRAM 1GB 240-Pin DDR2 SDRAM DDR2 800	\$23.49	12
Seagate 250GB 7200 RPM IDE Ultra ATA100 Hard Drive	\$64.99	1
DVD-ROM Drive	\$16.99	1
Rosewill RC-410 10/100/1000Mbps Black 8 Port Switch	\$39.99	1
Various Case Fans	\$32.34	1
<b>Grand Total</b>	\$2779.00	

**Figure 2: Parts list for our cluster system.**

## 3. Implementation

### 3.1 Hardware

The actual construction of the physical system was nearly trivial. The system consists of 6 compute nodes connected by a standard star network topology. Each compute node consists of a single quad-core processor and 2 GB of RAM on the MicroATX motherboard. One node (the “head”) includes the system hard disk, a basic CD/DVD drive, and an additional external network interface to connect the cluster to the outside world.

Instead of enclosing each node’s hardware within its own case, we chose a more space-efficient method of “stacking” the motherboards together on a frame created from a few simple threaded rods. Besides minimizing the cluster’s footprint, minimizing case hardware also saves on extraneous costs.

### 3.2 Software

What remains for the implementation is the software that is required to allow the cluster of microprocessor systems to act as a cohesive unit. Due to our goal of keeping the project cost low, it is preferred for the majority, if not all, of the software used to be free.

The first and most critical software need to address is that of an operating system. If possible, we wanted a system that has proven itself capable within the HPC community. To meet this requirement, a version of the Linux operating system is used. Linux has an excellent reputation for stability and is currently running 85.20% of the Top 500 supercomputers in the world [14]. We specifically chose Ubuntu Linux for its ease of use and fast setup time.

The method of inter-node communication that we use implement for this cluster is MPI. MPI (Message Passing Interface) is a standardized and portable message passing package designed to function on a wide array of parallel computers [12]. By using MPI, clustering functionality is implemented in user space and is in the hands of the users themselves. This does not limit functionality or hinder usability in any manner however, due largely in part to most MPI systems implementing job queuing. MPI is also a dependency of the industry standard benchmarking suite, LINPACK, which we ran to benchmark our system.

It is also necessary to provide a mechanism by which users can submit compute jobs to the system in such a manner that the multitude of jobs running do not interfere with one another. The tool used to handle this situation is a job queuing system. A job queuing system is responsible for controlling access to compute resources by controlling how user batch jobs are scheduled. Due to the job queuing system, scheduling conflicts between jobs are no longer a problem since the users are not allowed to control the scheduling of their own jobs. This makes the systems easier to use because of that fact. Users no longer need to concern themselves with how their job will be scheduled against other users’ jobs



since the computing time will be allocated fairly among all jobs.

The job queuing system we plan to implement on this system is Torque. Torque is a derivative of OpenPBS, which in turn is an open source version of the Portable Batch System (PBS) project. PBS was originally developed by NASA Ames Research Center, Lawrence Livermore National Laboratory, and Veridian Information Solutions, Inc. Torque provides numerous improvements over PBS (a list of which can be found at the Torque website) and is available as a free download [5].

## 4. Evaluation

In evaluating our cluster, we will use two approaches. First, in order to get a general sense of the overall system performance, we will evaluate our cluster using LINPACK, the industry standard HPC floating point computation benchmark. Second, we will measure the performance of the cluster in executing a load indicative of our original target applications – a batch of architecture simulations.

### 4.1 LINPACK

The current industry standard for basic HPC capability analysis is the benchmark LINPACK [6]. It is with the LINPACK benchmark that the Top 500 supercomputers in the world are currently determined and ranked [13]. The benchmark measures a system's double-precision floating point computational power. To determine this performance, LINPACK provides numerous numerical linear algebra routines which are run on the system being analyzed. These routines typically consist of sets of  $n$  by  $n$  linear systems  $Ax = b$  that the machine must solve.

The solving of these linear systems is a relatively good metric of expected computational performance because such linear algebra operations occur frequently in science and engineering computing applications. By measuring the number of routines the system can handle, as well as the time it takes the system to complete the routines, theoretical system performance can be calculated and reported in terms of how many billion floating point operations can be performed per second (GFlops).

It is necessary to understand that the results of LINPACK are still a very theoretical performance metric because it does not stress the full I/O capabilities of a machine. This is an important observation to make note of since the I/O performance of clustered microprocessor system, especially given our very thin network, is typically poor when compared to the larger custom supercomputers. Custom supercomputers have very high speed interconnect which are designed specifically for connecting compute nodes with high bandwidth and much lower latency. In contrast, microprocessor clusters typically use simple switched Ethernet for connecting compute nodes. While Gigabit Ethernet is a reliable and affordable performer for clusters, the operational latency of Ethernet is much higher than that of specialized high speed interconnect. As an example, one such interconnect, Myrinet, has an operating latency of one fifth to one tenth to that of Ethernet [7]. In some computational applications, there is a much higher dependence on

high-speed communication between nodes. In those cases, the shortcomings of Ethernet quickly become apparent.

## 4.2 SIM-MASE

In any performance analysis or computer architecture course, it is emphasized that the best benchmark to use to judge system performance is the program that you intend to use most often on the system. As stated earlier, one application we will be using on our cluster is an architectural simulator, *MASE* [8], which is a derivative of the *SimpleScalar* research simulator toolset [3].

*MASE* simulates the operation and performance of an aggressive out-of-order microprocessor. The simulated processor is capable of running real applications so that performance impacts of different architectures can be clearly shown.

However, since simulating the behavior of a processor entails far more computation than running the program itself, the simulations are exceptionally processing intensive.

## 5. Results and Analysis

### 5.1 LINPACK Results

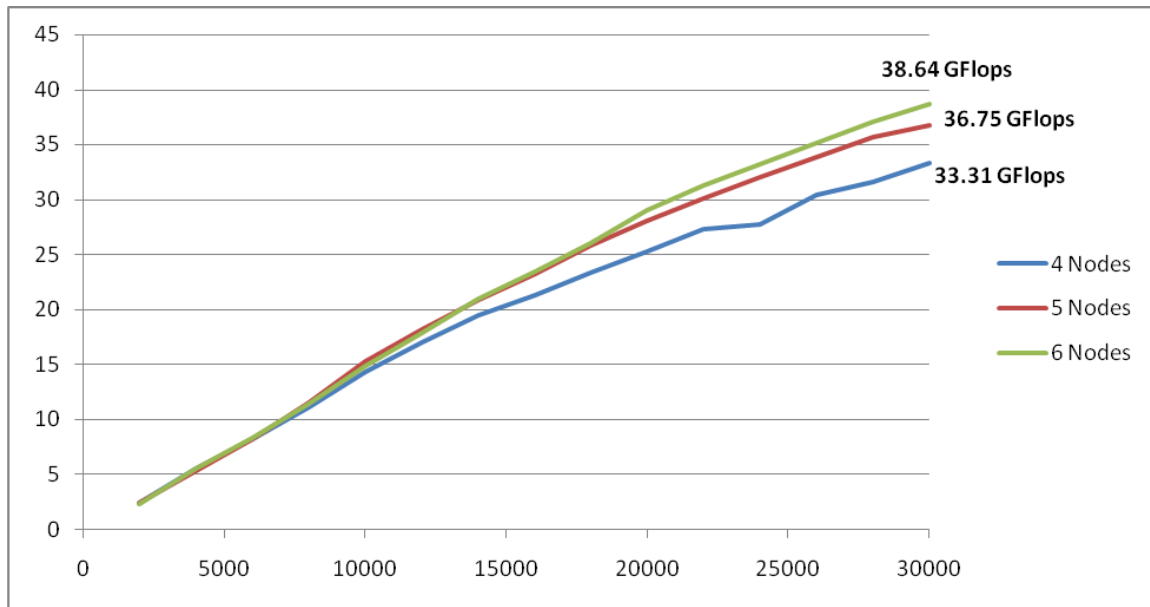
Figure 2 shows the LINPACK results for our cluster for varying problem sizes (N) and using different numbers of active nodes.

You can see that, as N increases, the performance also increases. This is because the larger the matrices are in LINPACK, the more computation that occurs between network sends/receives. Our cluster tops out at around 38.64 GFlops, for a cost efficiency of \$71.92/GFlop. As a peer comparison, MicroWulf reported 26.25 GFlops of performance with a maximum N of 30,000. Its cost efficiency at the time of construction was \$94.10/GFlop.

Interestingly, for most of the way up to our maximum problem size (anything over N=30,000 gives “out of memory” errors), using only 5 nodes nearly matches the performance of using the full 6. This non-intuitive result comes from the ratio of computation to transmission not being high enough to allow the processor to compute any faster (there is more transmitting with 6 active nodes, counterbalancing the extra processors).

The network performance of the benchmark was monitored using “ntop”, and the peak bandwidth used never exceeded 175 Mbits. This demonstrates that the communication overheads were not from bandwidth overload, but from the long latency of Ethernet.

The true limitation for LINPACK on our cluster is the small amount of memory. With more RAM, the processing to communication ratio would be significantly higher, allowing our processors to shine more fully.



**Figure 3: LINPACK performance for various N using 4, 5, or 6 nodes.**

## 5.2 MASE Results

Instances of MASE were run on variable numbers of processors, from one up to the full 24. Performance of the simulators was not impacted in any way as the number of processors scaled upwards. As previously stated, each instance of the program uses minimal memory and produces very little network traffic (enough to transfer files to and from the head node). In all cases, the simulator was able to maintain a throughput of over 660 million instructions per second on each node.

## 5.3 Analysis

The relative ease with which our design handled the architectural simulation comes as no surprise – the system was designed with that specific workload in mind. However, the performance of the LINPACK and MASE benchmarks taught us a few things.

First, that there is definite wisdom behind Amdahl’s rule of them, at least when looking at a computing system with a generalist benchmark such as LINPACK (which exercises every component at least somewhat). We chose, because of our budget and our target applications, to use an unbalanced amount of memory. This decision greatly affected the LINPACK performance of the system. Extrapolating out the performance curve to an N of 45000 (which would use roughly 2x the memory space) would yield a hypothetical system performance of 48.29 GFlops. However, the cost efficiency would be very nearly the same, as the cost of doubling the RAM is a significant investment.

Second, this study shows that general benchmarks such as LINPACK do not necessarily reflect the potential performance of a system on a workload that doesn’t have the same characteristics as the benchmark. Had we used LINPACK solely to judge our

performance, we might have concluded that our system would have run into the same limitations when running MASE. The result of that evaluation would have likely been the misguided decision to pay a heavy premium for more memory that we will likely never use.

Finally, our system provides a proof-of-concept that cost savings can be easily obtained for a cluster when it is constructed for a specific purpose, by building the system in a traditionally unbalanced way, as long as the final application space is well understood.

## 6. Concluding Remarks

Our goal for this project was to design a small, affordable cluster system that will meet our computational needs. Criteria for its construction were created and applied analytically to the parts available in the marketplace. Once the cluster was built, it was evaluated using both an industry standard general benchmark and a more specific targeted application, each showing a clearly different assessment of the system's capabilities.

## References

- [1] Joel Adams and Tim Brom. "Microwulf: A Beowulf Cluster for Every Desk." *Proceedings of the ACM Symposium on Computer Science Education (SIGCSE)*, 2008.
- [2] G. Amdahl, "Storage and I/O Parameters and Systems Potential," *Proceedings of the IEEE International Computer Group Conference (Memories, Terminals, and Peripherals)*, June 16-18, 1970, Washington, D. C., 371-372.
- [3] Todd Austin, Eric Larson, and Dan Ernst, "SimpleScalar: An Infrastructure for Computer System Modeling," *IEEE Computer*, February 2002.
- [4] beowulf.org. <http://www.beowulf.org/overview/index.html>, 2007.
- [5] Cluster Resources. Torque Resource Manager. <http://www.clusterresources.com/pages/products/torque-resource-manager.php>, 2007.
- [6] Jack J. Dongarra and Piotr Luszczek and Antoine Petit, "The LINPACK Benchmark: Past, Present, and Future," *Concurrency and Computation: Practice and Experience*, Volume 15, pp 1-18, 2003.
- [7] *HPCWire*. "Myricom Demonstrates Low-latency 10-Gigabit Ethernet." <http://www.hpcwire.com/hpc/708822.html>, 2006.
- [8] Eric Larson, Saugata Chatterjee, and Todd Austin, "The MASE Microarchitecture Simulation Environment", 2001 *IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS-2001)*, June 2001.
- [9] Myricom Home Page, <http://www.myri.com>.

- [10] “Pittsburgh Doubles Capability of BigBen”, Pittsburgh Supercomputing Center news release, Nov 21, 2006. <http://www.psc.edu/publicinfo/news/2006/2006-11-21-bigbengrows.php>
- [11] Neal Singer, “Red Storm upgrade boosts Sandia supercomputer to #2 in world,” Sandia National Laboratory news release, Nov 24, 2006. <http://www.sandia.gov/LabNews/061124.html>
- [12] Marc Snir, Steve Otto, Steven Huss-Lederman, David Walker, and Jack Dongarra. *MPI - The Complete Reference: Volume 1, The MPI Core*. The MIT Press, Cambridge, MA, 1999.
- [13] The Top500 Supercomputing Sites. <http://www.top500.org>.
- [14] The Top500 Supercomputing Sites. Operating System Family Share for 11/2007. <http://www.top500.org/stats/list/30/osfam>, 2007.

# Performance Evaluation of Java RMI in Parallel and Distributed Discrete Event Simulation

Thoshitha Thanushka Gamage  
Ahmad Ramadani  
Donald O. Hamnes

St. Cloud State University  
St. Cloud, MN 56301  
{gath0601, dohamnes} @stcloudstate.edu

## Abstract

*Java threads and RMI have gained a lot of popularity in recent years as a medium of distributed computing. In a distributed simulation, the efficiency as well as the accuracy of the simulating system will depend both on how well the simulation code is written and on how the nodes in the system communicate with each other. In an attempt to look into this issue, we carried out several experiments in Distributed Discrete Event Simulation using Java Threads and RMI. The main focus of our study was to evaluate the performance of Java RMI in a distributed simulation with a small number of processors. For the purpose of this study, we developed a distributed queuing network using a four (4) computer cluster which follows the conservative approach to distributed discrete event simulation. The empirical test results of our study are summarized and presented in this paper.*

## 1. Introduction

Implementation of real world systems in many cases is preceded by a simulation study which is carried out on a cluster of computers. This has been practiced not only because simulations are easy, but also because it is very expensive as well as dangerous to implement time critical and complex real systems without a proper study. In some cases, the need arises such that future events need to be assessed before time as part of a disaster preparation plan. For example, research such as worm propagation [11] in computer and communication networks is focused on analyzing and modeling epidemic style spread of attacks to study and to take appropriate counter measures to alleviate or minimize damage. Not only that, these studies help system engineers to build early attack detection and prevention systems.

The traditional approach to simulation is using sequential algorithms. Here at a given instance of simulation time, only one event gets executed or modeled with corresponding changes in the system [1]. In time critical systems, use of sequential simulators might not be the best solution as sequential execution of concurrent events is time consuming. An alternative is to use a parallel and distributed approach where several events could be modeled simultaneously and gain potential speedup. In many real life systems, events when they actually occur are rather discrete as well as unpredictable. Discrete events might occur at different locations or nodes in the system and there is every possibility for more than one event occurring at the same time as well. The notion of Parallel and Distributed Discrete Event Simulation (PDDES) takes into account all these concepts in developing models to observe the time based behavior of systems. Applications of PDDES can be observed in many domains including computer and network security, communication networks, game simulation, weather forecasting, battlefield simulation, queuing Networks, traffic modeling and digital circuits. Several other applications of Parallel Discrete Event Simulation can be found at [4].

The popularity of Java as a programming language in Discrete Event Simulation has shown steady increase during the past few years. Various features of Java such as platform independence, portability, scalability and even object oriented features are some of the reasons behind this increase in popularity and use. Not only that, Java provides several APIs that provide support for distributed and parallel Programming. Java Threads, RMI, CORBA and socket programming are well developed and documented APIs that come with the standard Java Development Kit (JDK). A recent addition to this collection is Java MPI [12, 13]. Thus various research groups are using Java for Discrete Event Simulation modeling. [9] shows the use of RMI for a web based simulation while [6] has implemented a Java based simulation engine in Conservative approach using Java RMI and Threads. IDES [7] targets portability and use in heterogeneous computing environments and selected Java for their Distributed simulation engine. An Initial investigation of the use of Java in PDDES on windows platform with small number of processors was carried out by [10]. Although [6] has used Java RMI and Threads in their implementation, there is the potential for more distinct comparisons of the technologies.

They do demonstrate that the use of Java with RMI can speed up a simulation when it is distributed over several machines. However, their emphasis has been more on evaluating the effect of network heterogeneity.

Section 2 is a general introduction to the concepts in PDDDES. Section 3 describes the implementation criteria we have used in this study which is followed by the implementation methodology. In Section 4 we describe the hardware and software and test parameters. Section 5 shows the results of our study and Section 6 lists the conclusions.

## 2. PDDDES

Sequential simulations are inherently slow, and processor intensive. The alternative is to break the code into several distributed versions and run them in parallel. During the execution, the system changes its state accounting for the events at discrete simulation time intervals. For the results of the simulation to be correct at a given time  $T$ , all events which occurred in the system up to time  $T$  should have been executed in increasing order of their simulation time. Thus the simulation should be identical to the behavior of the real system that is under study. Here the notion of time refers to the simulation time rather than real time or actual wall clock time. Since several logical processes (LPs) are running across multiple machines, it is necessary to synchronize their parallel events. Two main approaches to this problem are using Conservative Simulation [1] and Optimistic Simulation [2].

In the Conservative approach, each LP will wait until each of its incoming channels contain messages and select out the message with the smallest timestamp for execution. All LPs in the system progress towards a common simulation limit which could be a certain time constraint or number of events. In doing this, each LP insures that messages sent out are in the increasing order of their timestamp. The local simulation time of a LP also gets updated according to the timestamp of the message that has been selected. This mechanism ensures that all the LPs in the system maintain time synchronization and when a particular LP hits its local simulation time limit, it will not process any further event messages along any of its input channels. A mechanism to avoid possible deadlock due to waiting and how to recover from a deadlock situation has been described in [1].

While the Conservative approach maintains a strict ordering of event message execution in increasing timestamp order, the Optimistic approach on the other hand would allow any event message received to be executed as soon as possible. Here, when a particular LP receives message with a timestamp less than its clock value, a "*Rollback Mechanism*" is used to send anti-messages to alleviate the effect of previously sent incorrect event message(s). In the Optimistic approach, each LP works under its own local simulation time (a virtual time) and at a given instance of the wall clock time some LPs might be ahead of others. Nevertheless, this fact is invisible to the processes themselves [2]. The progress of the system globally is maintained using a Global Virtual Time (GVT) which is the minimum of all local virtual times and all messages that have been sent and still are in transit [2].



Performance of PDESs can be evaluated using the PHOLD model [8] which is an extension of widely used HOLD model for Sequential Event List Algorithm. This model consists of several parameters as described in [8]. In this study we have used the PHOLD model to assess the behavior of queuing systems under various test parameters.

### 3. Implementation

We implemented a queuing network following the Conservative approach. The code was primarily based on the Conservative approach PDES algorithm defined in [1] and was developed in Java. The queuing network consists of several queuing elements. The architecture of a single queuing element is depicted in Figure 1. A LP in this study is basically a single queuing element which consists of a queue, server and a branch point.

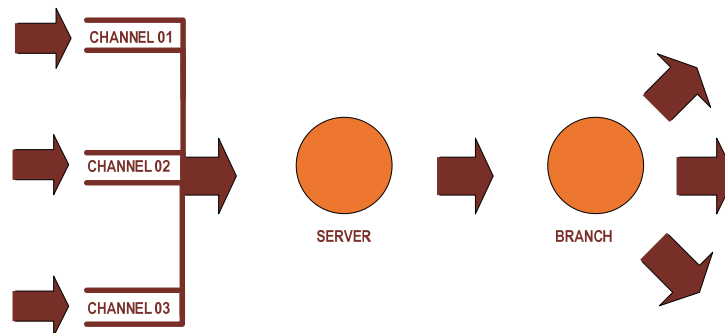


Figure 1: Single LP with 3 incoming Channels

Each LP has several inbound channels on which messages from other queuing elements arrive. These are dedicated channels between the corresponding sender and the receiver and in our study any LP can send messages to any other existing LP in the system. Thus, each LP had  $(n - 1)$  number of incoming channels where  $n$  is the number of LPs in the network. The server selects the smallest timestamp event message from the set of incoming channels and processes it. The processing takes a random time after which the message is sent out to its next destination by selecting a receiver randomly. The simulation was carried out using the following configurations.

- Configuration 01: A single Processor hosting  $n$  number of LPs using threads – ***SingleProcessorTHREAD***
- Configuration 02 : A single processor holding  $n$  number of LPs using Java RMI – ***SingleProcessorRMI***
- Configuration 03 :  $n$  number of processors each hosting one or two LPs using Java RMI – ***MultiProcessorRMI***

Here a processor is a computer with a single CPU.

## 4. Methodology

### 4.1 Hardware and Software

The simulation was carried on with the use of 4 DELL Dimension 2400 computers running Fedora core 5. Each machine was equipped with a 2.4GHz CPU and 512MB of RAM. The computers were interconnected with the use of 100 Mbps links and a 5 port switch. Figure 2 shows the interconnection network of the simulator with 4 LPs. Each LP has an incoming channel from each of the other LPs in the system. Java JDK 1.6 was used as the development platform.

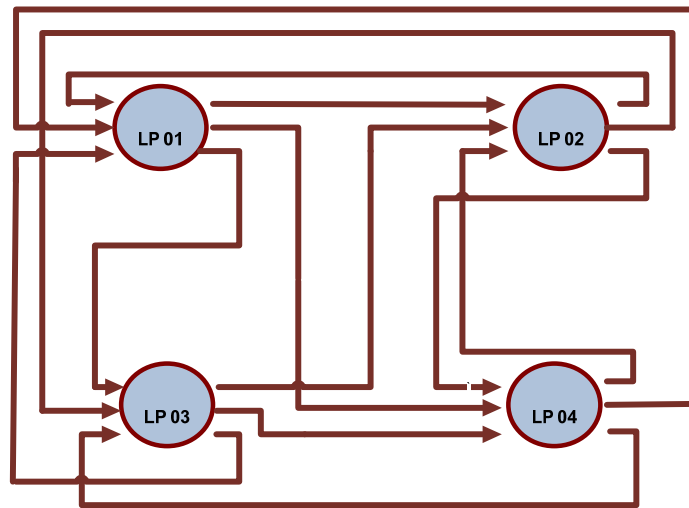


Figure 2: Interconnection Network with 4 LPs

Using 4 computers we simulated a system of 4 LPs and 8 LPs. In *SingleProcessorTHREAD* and *SingleProcessorRMI* configurations, a single computer hosted all 4 or 8 LPs. In the *MultiProcessorRMI* configuration, first, each computer acted as a single LP and hosted a single instance of the queuing system and later each computer was assigned 2 instances to simulate 8 LPs using 4 computers.

### 4.2 Test Parameters

The PHOLD model was used to evaluate parameterized behavior of the system. The PHOLD model defines six model parameters [8]. Table 1 shows the values of four model parameters that we have used in our study. Additional model parameters are discussed afterwards.

Model Parameter	Value
Number of LPs	4 and 8
Message Density	Refer Table 2
Computation Grain	10 milliseconds
Movement Function	Random selection out of $(n - 1)$

Table 1: PHOLD Model Parameters Used in the Study

The **Timestamp Increment Function** (TIF) was defined as follows. Whenever a message – whether it is an event message or a null message – is selected to be executed, its timestamp is checked against the present simulation time of the LP to see whether it is greater. If so, the simulation time is increased to the message timestamp. We used a biased exponential distribution to generate random service time values for each server. The definition of the service time generation function (ST) we used was,

$$ST = bias + \{rand * fAST\}$$

Here *fAST* stands for Fixed Average Service Time. We used a bias of 0.9 and *fAST* of 19. The random numbers were generated using the *java.Random* class. After servicing an Event message, the simulation time is updated for a second time with an increment of ST. Thus the message leaving the server reflects the effect of service time as well as the current simulation time of that particular server. **Computation Grain** is an artificial real time delay introduced at the server. After processing, the next receiver of a message is also selected randomly over a range of  $(n - 1)$  candidate destinations (queuing elements) - Thus termed as the **Movement Function**. The message densities used for both the 4 LP setting as well as the 8 LP setting were 16, 32, 48, 64 and 80. Initially, all the LPs start with the same number of messages. This defines the sixth parameter of the PHOLD model which is the **Initial Configuration**. For example, with a message density of 48 and a 4 LP setting, a single process would initially hold 48 messages, distributed equally as much as possible among all of the incoming channels. Table 2 shows extended test parameters for different test cases followed in each setting.

LP Setting	4 Logical Process	8 Logical Process
Message Density	16, 32, 48, 64, 80	16, 32, 48, 64, 80
Simulation Time Limit	225000	25000
Number of Runs	10	6

Table 2: Test Case Configurations

Each test case was executed several times as defined by Number of Runs and the following observations were made and recorded.

- Total Elapsed Time
- Total Number of Event Messages
- Total Number of Null Messages
- Total Service Time

These values were used to derive Average Service Time, Null message count and Speedup. Null messages are used to avoid and recover from Deadlocks [1]. Comparisons were made between different communication methods within a Configuration as well as between them.

## 5. Results

### 5.1 Test Model with Four LPs

For each configuration defined in Section 3, the queuing network was evaluated with message densities as defined by Table 2. Figure 3 shows the effect of message Density on average elapsed time. As the message density increases more event messages get serviced at the corresponding server. Thus, the TIF is primarily invoked due to event message servicing. This in turn increases the local simulation time in greater quantities ultimately leading the queuing element to reach stopping condition sooner.

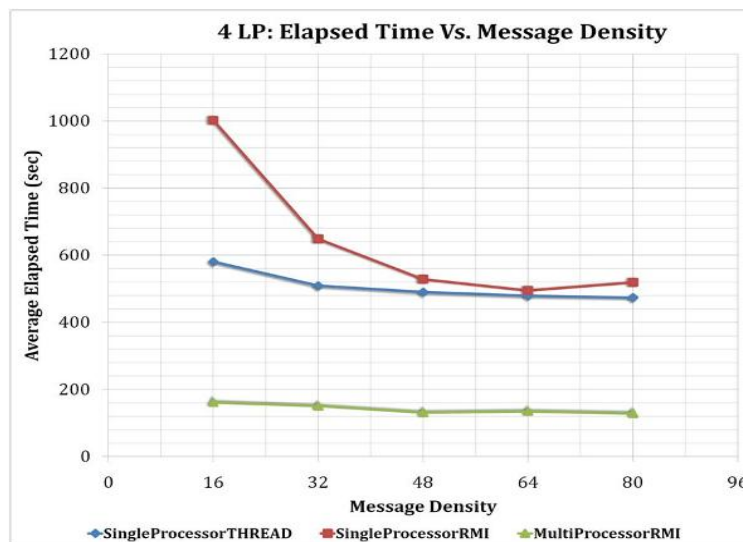


Figure 3: Variation of Average Elapsed Time with Message Density in 4 LP setting

The *MultiProcessorRMI* has the least elapsed time while the *SingleProcessorRMI* takes the most time. The *SingleProcessorTHREAD* version runs faster than *SingleProcessorRMI*. This might be caused due to overhead in RMI calling system calls in kernel and lower levels of communication for loopback connection establishments. Threads on the other hand are executed at higher level which returns quicker. Also running several LPs on a single processor is slower than executing on several processors. RMI is performing much faster than local threads in multi processor distributed environment.

The number of Event and Null messages received in the *MultiProcessorRMI* configuration for the 4 LP setting as a function of Message Density is shown in Figure 4. When the Message Density increases, Null message count rapidly decreased. At the same time the number of Event messages received increased although the rate of increase was less than that of rate of decrease in Null message receipt. With more Event messages, the parallelism of the system increased thus the need for Null messages for the purpose of preventing deadlock decreased.

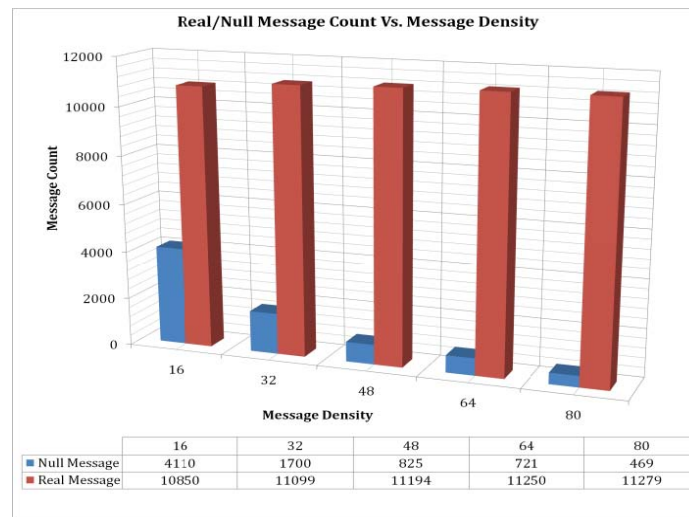


Figure 4: Received Event and Null message count variation with message Density in 4 LP

The average wait Time is defined as the average simulation time an event message has to wait in the queue (channel) before being serviced. With low message densities an event message has a higher chance of being selected as soon as it arrives. But as the number of event messages in the system increases, so does the time an event message has to wait.

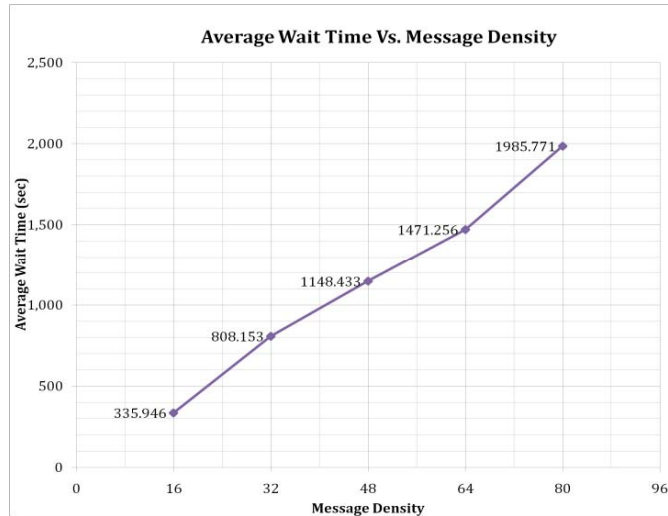


Figure 5: Average Wait Time variation of 4 LP setting with Message Density

The wait time behavior in 4 LP setting with configuration 03 is shown in Figure 5. The waiting time increases almost linearly with the increase of Message Density.

## 5.2 Test Model with Eight LPs

A similar set of tests were carried out for the Eight LP setting. Since we only used 4 machines in our study, a single computer was assigned to host 2 LPs. Here again we used Message Densities of 16, 32, 48, 64 and 80.

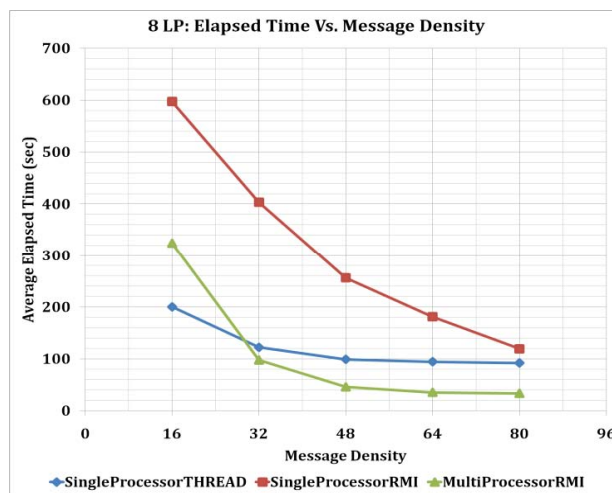


Figure 6: Variation of Average Elapsed Time with Message Density in 8 LP Setting

Figure 6 is a graph of Average Elapsed Time against Message Density in 8 LP setting. The behavior is similar to that of 4 LP setting, where as **MultiProcessorRMI** outperforms **SingleProcessorRMI**. The performance of **MultiProcessorRMI** in the 4 LP setting appears more uniform across message densities than it does for the 8 LP case. A reasonable explanation is that since we used only 4 machines to represent 8 LPs, a communication overhead might have been introduced in name resolution or virtual IP resolution. Better results might be expected with 8 individual processors.

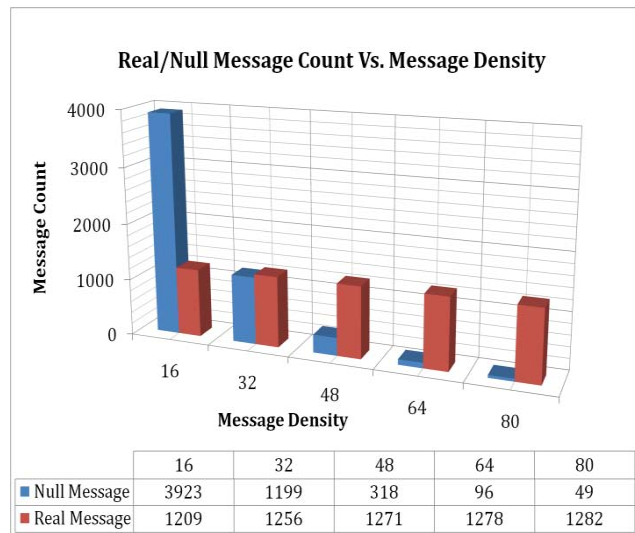


Figure 7: Received event and Null message count variation with message Density in 8 LP

Figure 7 shows the Number of received Null messages and event messages for each Message Density in a 8 LP **MultiProcessorRMI** configuration. Number of received Null messages decreased rapidly with the Message Density.

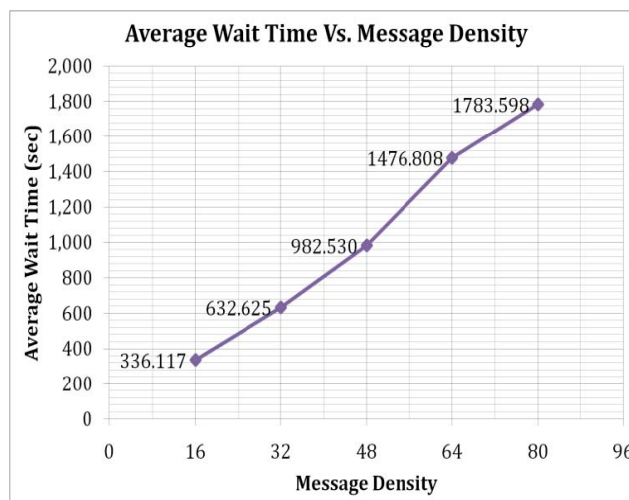


Figure 8: Average Wait Time variation of 8 LP Setting with Message Density

Wait Time behavior with varying Message Density for 8 LP setting with **MultiProcessorRMI** configuration is shown in Figure 8. Here again the wait time almost linearly increases with Message Density. By comparison with 4 LP setting, the wait time for a specific Message Density has increased in 8 LP setting.

### 5.3 Speedup

Speedup comparisons were made in both 4 and 8 LP configurations. These comparisons are shown in Figure 9.

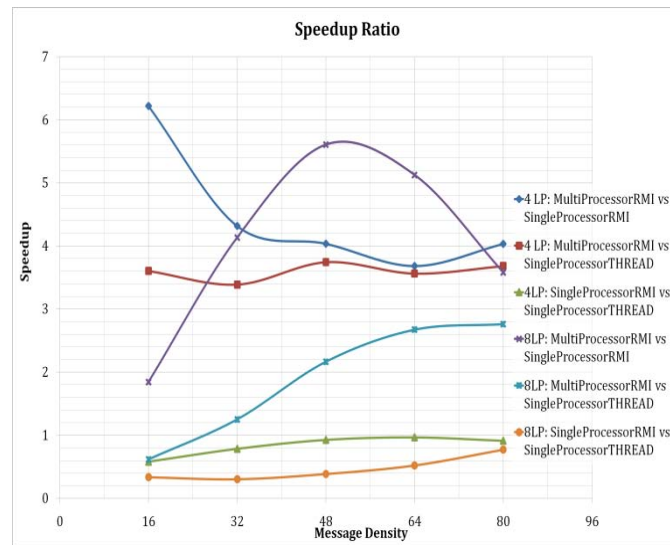


Figure 9: Speedup Ratio for 4 LP and 8 LP Settings

In 4 LP setting, the potential speedup of using **MultiProcessorRMI** was on the average around 4.5 times than that of using **SingleProcessorRMI**. The speedup variation with increasing message density was more prominent in 8 LP case ranging approximately from 2 to 6. The average speedup of 8 LP yields to around 4.0 which implies that running too many simultaneous RMI codes in a single processor slows down the performance. On a single Processor, threads runs faster than local RMI with speedup less than 1 for both 4 and 8 LP setting. In general, **MultiProcessorRMI** shows better performance over **SingleProcessorRMI** and **SingleProcessorTHREAD** versions for both the 4 LP setting as well as the 8 LP setting.



## 6. Conclusion

In this study we implemented the Conservative approach to Parallel and Distributed Discrete Event Simulation with Java RMI and Threads. Our results led to the following conclusions about the performance of Conservative simulation in our environment:

- On a single processor, using RMI communication results in poorer performance relative to a thread-only simulation; this is especially true at the lower message densities.
- On a single processor as the message density increases, the performance of RMI becomes much closer to that of Threads running several processes in the same machine as both execute with nearly the same average elapsed time.
- For a small number of LPs, using RMI to communicate among independent processors is faster than a single processor hosting either multiple THREAD or RMI based processes.
  - For a distributed simulation on a small number of processors, using Java RMI speeds up the overall simulation relative to a thread-only simulation on a single processor.
  - For a distributed simulation on a small number of processors, using Java RMI speeds up the overall simulation relative to an RMI based simulation on a single processor.
- Generally, elapsed times decrease with increasing message density.
- Overall performance behavior with 4 LPs and 8 LPs is similar as measured by elapsed time.

However, use of RMI for distributed computing might not be the most optimal solution. As shown by [14], Sockets outperform RMI. Also, RMI has shown considerable overhead in object marshalling and un-marshalling as shown by [15].

Further work in this area would be to measure the performance of Java MPI [12, 13] compared with RMI or implementing an Optimistic approach simulator on the same set of criteria to analyze the system behavior.

## 7. References

- [1]. Misra, J. Distributed Discrete Event Simulation. *ACM Computing Surveys, Vol 18, March 1986.*
- [2]. Jefferson, D.R. Virtual Time. *ACM Transactions on Programming Languages and Systems, Vol 7, July 1985.*
- [3]. Teo, Y.M., Ng, Y.K., Onggo, B.S. Conservative Simulation Using Distributed Shared Memory. *Proceedings of the 16<sup>th</sup> Workshop on parallel and Distributed Simulation (PAD'S 2002.)*
- [4]. Trooper, C. Parallel Discrete Event Simulation-applications. *ACM Journal of Parallel and Distributed Computing, Vol 63, 2002.*
- [5]. TEO, Y.M., NG, Y.K. SpaDES/Java: Object Oriented Parallel Discrete Event Simulation. *Proceedings of the 35<sup>th</sup> Annual Simulation Symposium 2002.*
- [6]. Ferscha, A., Ritcher, M. Java Based Conservative Distributed Simulation. *Proceedings of the 1997 Winter Simulation Conference.*
- [7]. Nicol, D.M., Johnson, M.M. IDES: A Java Based Distributed Simulation Engine. *6<sup>th</sup> IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunications Systems (MASCOTS) 1998.*
- [8]. Fujimoto, R.M. Performance of Time Warp under Synthetic Workloads. *Simulation Councils, Inc. Vol 22 1990.*
- [9]. Page, H.E., Moose, R.L., Griffin, S.P. Web Based Simulation in SimJava Using Remote Method Invocation. *Proceedings of the 1997 winter simulation Conference.*
- [10]. Ramadani, A. Conservative Parallel and Distributed Discrete Event Simulation using Java. *A starred paper submitted to the graduate faculty of St. Cloud State University 2006.*
- [11]. Wei, S., Mirkovic, J., Swamy, M. Distributed Worm Simulation with a Realistic Internet Model. *Proceedings of the 19th Workshop on Principles of Advanced and Distributed Simulation (PADS) 2005.*
- [12]. WeiQin, T., Hua, Y., WenSheng, Y. PJMPI: pure Java implementation of MPI. *Proceedings of the 4<sup>th</sup> International Conference/Exhibition on High Performance Computing. 2000.*
- [13]. <http://www.hpjava.org/>
- [14]. Ahuja, S.P., Quintao, R. Performance Evaluation of Java RMI: A Distributed Object Architecture for Internet Based Applications. *IEEE Proceedings of the English International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems 2000.*
- [15]. Kazi, I.H., Chen, H.H. Techniques for Obtaining High Performance in Java Programs. *ACM Computing Surveys, Vol. 32, No.3, September 2000.*

# SECURING THE BORDER GATEWAY PROTOCOL

<b>Matthew Nickasch, Joe Cavanaugh, Michael Kohl, Matt Dolfin</b> Comp. Science and Software Engineering Students	<b>Syed (Shawon) M. Rahman, Ph.D.</b> Assistant Professor
--	--

Dept. of Computer Science & Software Engineering  
University of Wisconsin - Platteville  
1 University Plaza, Platteville, WI 53818, USA  
Phone: (608) 342 1625, Fax: (608) 342-1965  
Email: {nickaschm, rahmans, cavanajo, kohlm, dolfinm}@uwplatt.edu

## Abstract

BGP, or Border Gateway Protocol, is the primary inter-domain routing protocol used today in the Internet. Due to the complexity and high-profile nature of the protocol, BGP is particularly vulnerable to many types of attacks; therefore putting large networks at risk.

This paper first analyzes the basics and fundamentals of the Border Gateway Protocol, including modern applications of BGP in the Internet today. Further analysis of the Autonomous System (AS), route selection and building, as well as network advertisement, are defined in the analysis of the protocol.

Secondly, the security risks and vulnerabilities of the protocol are analyzed. From BGP message vulnerabilities, to route redirection, black holing, and route flapping, the protocol faces some significant security-related challenges. There are several types of attacks that can be made against BGP and its propagation of routing information to others. Any of these attacks may be instigated by gaining access to BGP-enabled routers by using brute-force password attacks, or abusing a router's Operating System (OS) vulnerabilities.

Finally, recommendations of securing the Border Gateway Protocol are described. From PGBGP (Pretty Good BGP) to SBGP (Secure BGP), there are several draft recommendations to further secure the protocol.

## **Introduction**

Routing protocols often go unnoticed in the day-to-day usage of the Internet. Anything from checking your email from home, to exchanging information between Tier1 Internet Service Providers, routing protocols make up a very important part of the Internet.

The need for routing protocols is immense. From the seemingly simple task of exchanging information of routes between multiple routers, to providing services for ISP multihoming and algorithmic best route selections, routing protocols are needed to complete the job. This task of learning and advertising routes occurs at Layer 3. Almost all application-level protocols rely heavily on Layer 3 routing, so the importance of sharing routing information between multiple networks is absolutely necessary [1].

As always, there are multiple methods to accurately and efficiently communicate available routes, and therefore, wide arrays of routing protocols have been developed to facilitate this sharing of information. Border Gateway Protocol (BGP) is used primarily on very large networks, typically between Internet Service Providers (ISPs), or large companies or organizations who connect to multiple ISPs.

## **Routing Protocol Overview and Comparison**

With a plethora of routing protocols available for network architects to consider, the difference between such protocols can be confusing. However, routing protocols fall into one of two categories: Interior Gateway Protocols (IGP) and External Gateway Protocols (EGP).

### **Autonomous Systems (AS)**

When determining the difference between IGPs and EGPs, it is important to consider the AS, or autonomous system, in which these protocols will be exchanging information. An AS is often an abstract classification given to a collection of networks that are subject to the same routing policy. This AS can contain networks of more than one organization, and can be classified as a single administrative domain in which these collections of networks fall [2].

### **Interior Gateway Protocols (IGP)**

Interior Gateway Protocols are used to communicate routing information internally within an AS, but not outside of the AS. There are four primary protocols used for intra-AS routing: Routing Information Protocol (RIP), Interior Gateway Routing Protocol (IGRP), Open Shortest Path First protocol (OSPF), and Intermediate System to Intermediate System (IS-IS).

- 1) RIP, or Routing Information Protocol, is perhaps the simplest routing protocol of all IGP and EGP. RIP is a distance-vector protocol, and uses a hop count as a simple metric for calculating best routes. A router actively listens for other routers communicating routing updates and adds or modifies new routes into its routing table in order to communicate with other routers running RIP in the AS. RIP's logic is based on the Bellman Ford Algorithm. There are several limitations of RIP, with the most significant being a 15-hop limit. Also, a network with frequent topology changes can generate a lot of RIP-generated bandwidth in order to communicate these frequent routing table updates. Finally, RIP is considered to be a very "slow" protocol in realizing link changes across networks.
- 2) IGRP, or the Interior Gateway Routing Protocol, is another distance-vector protocol. This protocol was developed by Cisco, and therefore runs only on Cisco routers. In comparison with RIP, IGRP allows for faster convergence to identify topology changes.
- 3) OSPF, or Open Shortest Path First protocol, is not distance-vector based, and is therefore architected significantly different from RIP and IGRP. This protocol is link-state based, and is therefore significantly more complex. OSPF utilizes DRs and BDRs (Designated and Backup Designated Routers) elected on a subnet basis to handle information about link-state changes. These "supervisory" routers collect the initial network topology information and replicate it out to all other router-peers on the network. Once this process is complete, the Shortest Path First algorithm is executed on each router, and uses that information to populate its routing table with the best routes. This common exchange of routing information, using the same SPF algorithm, allows all routers on the network to converge on routes, eliminating any possibilities of routing loops.
- 4) IS-IS, or Intermediate System to Intermediate System, is also a link-state routing protocol. The OSPF protocol was developed out of influence from IS-IS. It is important to note that only OSPF and IS-IS natively support VLSM, or Variable Length Subnet Masks, which will be discussed later in the paper. VLSM support is essential in the modern application of networks that are heavily subnetted and partitioned [3].

### **External Gateway Protocols (EGPs)**

While IGPs are necessary for communicating routing information within an AS, EGPs are necessary for communicating routing information between Autonomous Systems. In fact, interior routes maintained by IGPs may also be used in upstream routing, hence, in routes maintained by EGP. BGP, or Border Gateway Protocol, is considered to be the standard in inter-AS routing protocols, because of its wide acceptance, open standards, and efficiency.

## **History and Development of BGP**

EGP, which was originally specified in 1982, was used during the earlier days of the internet to connect ASes. It is commonly used between hosts on the Internet to exchange routing table information. The routing table contains a list of known routers, the addresses they can reach, and a cost or metric associated with the path to each router so that the best available route is chosen.

Each router polls its neighbor at intervals of varying length and the neighbor responds by sending its complete routing table. EGP is a simple reachability protocol, and, unlike modern distance-vector and path-vector protocols, it is limited to tree-like topologies [6].

### **BGP version 1**

BGP version 1 was created to replace the EGP routing protocol to allow fully decentralized routing. The message size for BGP version 1 varied in size from 8 to 1024 bytes, this was changed to a message size from 19 to 4096 bytes in the subsequent versions. This was largely done due to the protocol expanding and becoming redefined to support newer and better path mapping [17].

### **BGP version 2**

In version 2, the concept of "up", "down", and "horizontal" relations between autonomous systems that were present in version 1 were removed. BGP version 2 also introduced the concept of path attributes and further elaborated on parts of the protocol that were considered "under-specified" [9].

### **BGP version 3**

In version 3, several restrictions on the use of NEXT\_HOP path attribute were removed. Along with this, version 3 clarified the procedure for distributing BGP routes between BGP speakers within an ASes [9].

### **BGP version 4**

BGP version 4, which has been the standard for the internet since 1994, has yet to be replaced. This version redefined the previously class-based network layer reachability portion of the updates to specify prefixes of arbitrary length in order to represent multiple classful networks in a single entry. The AS\_PATH attribute has been modified so that sets of ASes, as well as individual ASes may be described. In additions, the INTER-AS Metric attribute has been redefined as the MULTI-EXIT

DISCRIMINATOR. The LOCAL-PREFERENCE and AGGREGATOR attributes have been added. BGP version 4 also became the first version to support CIDR. Very few vendors still support earlier versions of BGP [7][9].

## **Fundamentals of BGPv4**

### **Overview**

The Border Gateway Protocol (BGP) uses a Transmission Control Protocol (TCP) connection through port 179. Since it does not run directly on top of the Internet Protocol (IP) or use User Datagram Protocol (UDP), it is possible to discover neighbors through sending broadcast and multicasts. After establishing a TCP connection, BGP information is sent in the form of messages. The messages have a header which contains a marker, the length of the BGP message, and the type. The marker checks for sender and receiver synchronization. The type indicates the messages purpose. The type can be open, update, notification, or keep alive [2].

### **Routing Algorithm**

BGP is often thought of as a distance path protocol. Unlike other protocols, BGP doesn't keep track of just a hop count, yet it doesn't keep track of the entire topology of the network either. Instead, every router receives information on the availability of only its neighboring routers. It chooses the shortest path between ASes, updates the routing tables and announces the information to its neighbors. This way BGP doesn't have to keep track of each network within the AS, but only the route between them [2].

### **Autonomous Systems (AS)**

An Autonomous System is defined in the IETF document, RFC 4271:

“The classic definition of an Autonomous System is a set of routers under a single technical administration, using an interior gateway protocol (IGP) and common metrics to determine how to route packets within the AS, and using an inter-AS routing protocol to determine how to route packets to other ASs. Since this classic definition was developed, it has become common for a single AS to use several IGPs and sometimes several sets of metrics within an AS. The use of the term Autonomous System here stresses the fact that, even when multiple IGPs and metrics are used, the administration of an AS appears to other ASs to have a single coherent interior routing plan and presents a consistent picture of what destinations are reachable through it.” [10]

Each AS is specified by a number from a 16-bit number field, allowing 65,536 possible fields. The numbers 1 through 64511 are available for use with internet routing while the numbers 64512 through 65534 are reserved for private use. AS 0, 65535, and 23456 are also reserved for various purposes. The AS numbers are used by ASes to determine where a message started and how it got to its final location [2][10].

### **Network Advertisement**

Route announcement says which AS has control over a certain IP address space. If an IP address space were to go down, the AS has to notify every other AS to update its routing tables. If the network comes back online it must again announce a change in the table configuration. The repeated on/off is commonly referred to as “route flapping” and causes major congestion between the ASes.

### **Interdomain Routing**

Classless Inter-domain Routing (CIDR) replaces the notion of network classes (such as A, B, C) by freely allowing the AS to choose of the number of bits for the network part of an address. According to Beijnum, “instead of looking to see if an address is Class A, B, or C every route has an explicit indication of the number of bits that belong to the network part of the address, either in prefix format or as a netmask.” [2]

### **Route Replication**

Route replication is where a redundant link is configured to the internet to increase services and reduce costs. BGP is used as a key tool for achieving internet connection redundancy. Multihoming increases the reliability of a connection by using two or more ISPs. This insures that the IP range of the network is still accessible when one of the connections or ISPs fail. BGP uses algorithmic intelligence at the router to decide which ISP has the most efficient path [10].

### **Types of BGP Sessions**

According to the Request For Comments (RFC's) for BGP there is a list of sessions that BGP can be in. These include idle, connect, active, opensent, openconfirm, and established. The state BGP is currently in will determine the behavior of the router. Idle is when the router is not setting up a BGP connection and any neighbors setting up TCP connections will be refused. Connect is when the router is waiting for its own TCP establishment to complete. Active is when BGP is waiting for some TCP



session. OpenSent is when the open message has yet to be received but the open message has been sent. OpenConfirm is the open message from a neighbor has been received and is waiting for the completion of the BGP setup phase. Lastly, Established is the initial keepAlive message is received and is ready for the other message types. [2]

The propagation of BGP routes happens when a new route is received in a BGP update message. When a BGP route is updated the router checks the filters to see if it's allowed and inserts the route if it is. It compares the route to other routes and determines if it is the best. If it is the best the old route is removed and tells all its neighbors it has received a new best route. If the filters on the external neighbor ASes allow for the new route, it is propagated. The neighbors in the local AS are then updated with the new best route, usually without any additional filtering [2].

### **Route Building / Selection**

Route selection enables a router to select what route is the best amongst those presented by the BGP. Three processes build and maintain the routing tables: The BGP process, the route table itself by accepting information from BGP, and the forwarding process which requests information from the routing table to make a packet forwarding decision [10].

Routes are built and chosen in the routing table based on the BGP administrative distance. Routes learned from protocol with the lowest administrative distance are installed in the routing table. If there are multiple paths to the same destination from a single routing protocol then the multiple paths will have the same administrative distance and the best path is selected based on the metrics. Metrics are values associated with specific routes, ranking them from most preferred to the least preferred. The parameters used to determine the metrics differ for different routing protocols, as mentioned in the History and Development of BGP [10].

### **Security Considerations**

The Border Gateway Protocol (BGP) allows for the passing of information packets by the use of routing tables through separate Autonomous Systems (ASes). ASes are usually large Internet Service Providers (ISPs). BGP is a key tool for ensuring multiple connections between two separate ASes. To achieve this, BGP uses algorithmic intelligence at the router to decide which ISP has the most efficient path. Routes are built and chosen in the routing table based on the BGP administrative distance. Routes with the lowest administrative distance are installed in the routing table. If there are multiple paths to the same destination from a single routing protocol then the multiple paths will have the same administrative distance and the best path is selected based on the metrics. Metrics are values associated with specific routes, ranking them from most preferred to the least preferred. [18]

It is common knowledge that the internet contains numerous vulnerabilities. Some prominent examples include Internet viruses, worms, and spyware. Often, these attacks are attributed to a lack of protection in software, including operating systems and commercial programs. In addition to these software deficiencies we often find a lack of security inside of the network layer of the TCP (Transfer Control Protocol) Model. Since the routing structure of the Internet consists of many “peers” that work together to route traffic, malformed routing tables can easily and quickly replicate across the entire Internet. Human error in routing configurations can also significantly affect large portions of the Internet, causing outages and malformed routing tables. Specific to BGP, most would assume that vulnerabilities arise from the protocol itself, while many potentially damaging attacks may result from implementation. Therefore, repairing and securing the protocol itself will not solve all security-related problems [18].

There are several types of attacks that can be made against BGP and its propagation of routing information to others. Any of these attacks may be instigated by gaining access to BGP-enabled routers by using brute-force password attacks, or abusing a router’s Operating System (OS) vulnerabilities. Some of the specific types of attacks include false route updates, route redirection, route instability, and blackholing. False Updates are a very clear-cut type of attack against the protocol. This simply occurs when an Autonomous System advertises a route or a prefix that it does not administratively own. Route Redirection causes traffic to a specific destination to be routed away and possibly to another compromised destination. Route Instability or Route Flapping occurs when routes to networks are advertised and quickly retracted, causing route dampening algorithms to activate in upstream routers, causing a delay in establishing valid routes. This could potentially cause frequent short-term outages during these frequent topology changes. Finally, blackholing is when an entire network or prefix is inaccessible from a large segment of the Internet. There are some cases where blackholing is valid, particularly for private and restricted network ranges (192.x.x.x Class C networks) that should not be advertised. Malicious blackholing is the mis-advertisement of valid routes, causing routers to drop valid traffic to the “blackholed destinations.” [18]

### **BGP Message Vulnerabilities**

Message authenticity is largely vulnerable. These types of vulnerabilities span most or all parts of the BGP message structure. First, the message header, or the leading part of the BGP message, contains information that requires specific syntax [19].

Any errors, especially syntax-based, will cause the BGP session to drop. In addition, route-calculating processes will re-run and therefore consume router resources to determine appropriate routes. Malformed message headers pose a significant risk for router’s CPUs. Message header errors can potentially cause wide-reaching problems

in the event that large ASes are compromised, sending out malformed or syntactically-incorrect BGP message headers [19].

In addition to the message header, potential vulnerabilities exist at the various BGP message forms. An OPEN message calls the router to invoke its route-decision process in anticipation of new routes being calculated and added to the routing table. OPEN message arrivals may indicate that an attempted new connection, or that an existing connection has dropped and is trying to be re-established. Frequent, or abusive OPEN messages cause problems related to the respective router's CPU utilization, similar to the vulnerabilities that exist in BGP message headers [19].

The BGP KEEPALIVE message is not transmitted when BGP connections are being established. A transmittal of a KEEPALIVE message while in the states of Connect, Active, and OpenSent would cause the connection-in-progress to fail. Therefore, an improperly-placed KEEPALIVE message would cause connections to fail [19].

BGP NOTIFICATION messages can easily be spoofed, and transmittal of these messages to BGP listeners will cause an immediate loss of the active BGP session. Large amounts of spoofed NOTIFICATION messages could significantly impact routing via a "cascade" effect, and may make connections between large numbers of ASes impossible [19].

## **Route Redirection**

Route redirection is when traffic flow is forced to take a different path to reach a particular network. The path that the traffic is redirected to is often either incorrect or is potentially compromised. The main objective of a redirection attack is to have the compromised destination impersonate the true destination. This way the attackers can get confidential information. The redirected traffic can also be used to congest or completely collapse a different network [13].

In order to instantiate a route redirection, a hacker first needs to get into a networks router. From there the hacker changes the routing tables to isolate parts of the network and direct network traffic elsewhere. Another way a hacker can get your computers to send data to the wrong address is to send Internet Control Message Protocol (ICMP) redirect packets to the router. An ICMP redirect packet instructs the router that an IP packet is being sent to the wrong router and that there is another route to the destination address that is more efficient, faster, or capable of avoiding a network problem. It is difficult to forge ICMP packets, however, because they must appear to come from the router closest to the originating computer [13].

## **Denial of Service (DoS)**

A denial of service attack is an attack meant to make a resource unavailable to the intended recipients. Often these attacks are intended to be malicious and target Internet services that receive a lot of traffic to begin with. Common types of denial of service attacks include the following [6].

### **SYN attack**

In the Transport Control Protocol (TCP), a small buffer space exists to handle the initial “hand-shaking” exchange that sets up the session between a server and a client [6]. The packets used to establish the connection have a SYN field identifying the status of the hand-shaking process. An attacker can perform a SYN attack by sending a large number of connection requests quickly without responding to the reply from the server. The packets fill up the buffer thereby blocking the legitimate requests of other clients. The packets in the buffer are eventually dropped without a reply, but the effect of many of these connection requests still makes it difficult for legitimate requests to get established [6].

### **Physical Infrastructure Attacks**

A Physical Infrastructure Attack is where a piece of the hardware is forcibly disconnected. An example of this attack can range from simply cutting a cable to damaging one of the routing devices. This attack is easily handled by redirecting the traffic to another available route [6].

### **Teardrop Attack**

The internet protocol requires each packet size to be within a certain range. If a packet is too large for the next router to handle, the packet should be divided into smaller fragments. Each of these smaller packets identifies an offset from the first packet which is then used to piece together all of the fragments of the original packet. An attacker can send a packet with a misleading offset value in subsequent fragments that then make it difficult for the receiving to reconstruct the original packet. If this is not handled by the receiving system, this can cause the system to crash [6].

### **Viruses**

Computer viruses replicate across networks and can act as denial-of-service attacks in that they make a resource unavailable to the intended recipients. A virus usually doesn't specifically target one system but simply anyone who happens to get the virus [6].

## **Smurf Attack**

In a smurf attack, the attacker sends a ping request to multiple servers using the IP address of target. This attack causes the recipients to all send the ping back to the target host which causes congestion at the receiving host's buffer [6].

## **Blackholing**

Blackholing is a technique used to deny a particular IP address or domain access to an AS or specific machine. Through Blackholing, packets are discarded based upon some assigned criteria. For example, an ISP might blackhole packets coming from a known spammer or from a file sharing application such as Bit Torrent. In this way blackholing can be used to protect systems from malicious software. Blackholing can also be used with a malicious intent in mind. Malicious blackholing refers to false route advertisements that aim to attract legitimate traffic to the wrong router and then drop it. This would cause packets to be lost and for information to no longer be delivered to its intended source [6].

## **Route Flapping**

Route announcement says which AS has control over a certain IP address space. If an IP address space were to go down, the AS has to notify every other AS to update its routing tables. If the network comes back online it must again announce a change in the table configuration. The repeated on/off is commonly referred to as "route flapping" and causes major congestion between the ASes [19].

Route Instability or Route Flapping occurs when routes to networks are advertised and quickly retracted, causing route dampening algorithms to activate in upstream routers, causing a delay in establishing valid routes. This could potentially cause frequent short-term outages during these frequent topology changes [19].

Route flapping is caused by a variety of conditions within the network which cause information on the reachability of an area of the network to be repeatedly advertised and then withdrawn. Configuration errors and sporadic errors in communications links are the most common causes for route flapping. Route flapping often forces routers to recalculate new routes to the offending network, while traffic on its way to that network is already in transit through the routers [20].

An example of the effects that route flapping can have is given below.

"In October 2002, a seemingly small misconfiguration of a router caused widespread outages. Improper filtering rules added to a router caused the routing tables of WorldCom's internal infrastructure to become flooded with external routing data. The internal routers became overloaded and crashed repeatedly. This caused prefixes and paths advertised by these routers to

disappear from routing tables and reappear when the routers came back online. This repeated advertisement and withdrawal of prefixes, known as route flapping, served to destabilize the surrounding network.” [21]

## **Pretty Good BGP (PGBGP)**

There is significant ongoing research on techniques and strategies to improve the Border Gateway Protocol. BGP's vital importance to the routing structure of the Internet lacks the necessary security to protect the Internet against large-scale routing attacks. Pretty Good BGP (or PGBGP) is an effort to improve the existing BGP implementation by making well-formed modifications [15][16][17].

The main focus of the PGBGP effort is in direct response to "bogus routes" that compromise routing tables across multiple ASes. PGBGP significantly protects against bogus routes by adding a time delay for learning new routes, adding a enough time for a logical analysis and "weeding-out" of bogus or poor routing decisions [15][16][17]. Since the introduction of bogus routes into routing tables is a prime attack vector, the PGBGP effort significantly closes a possible attack vector against BGP. The primary foundation of PGBGP is that even recognizable, pre-established routes are treated with vigilance before used as a primary route within intra-AS routing tables [15][16][17].

There are specific metrics that PGBGP considers when analyzing intra-AS routes for consideration. As routes are newly formed and prioritized, PGBGP places them in a quarantine to ensure their authenticity and quality. The protocol analyzes time, the update's prefix, and routing tables to verify the dependability of the proposed update or new route. Also during this intermediate period, PGBGP accepts updates to the specified route "in waiting" and recalculates metrics if such an update occurs [15][16][17]. After the intermediate time has expired, PGBGP commits these time-tested routes to the "normal routing table". This verified routing table consists of trusted routes, and so-called bogus routes are usually eliminated in quarantine. Bogus routes are classified as suspicious, and the router relies on routes from the verified table to route traffic [15][16][17].

PGBGP also communicates known bad routes to an Internet repository for other routers to analyze. This peer-contribution of information allows network operators to quickly identify bad or bogus routes. Mailing lists such as NANOG (North American Network Operators Group) frequently communicate and tabulate known bad routes to ensure that such routes do not affect intra-AS traffic [15][16][17]. This proposal, while containing some flaws, significantly closes a well-known attack vector for the existing BGPv4 protocol: bogus routes. The verification and quarantine of new and updated routes would significantly reduce such attacks on intra-AS networks [15][16][17].

## **Secure BGP (SBGP)**

BGP is highly vulnerable to a variety of attacks, secure BGP addresses these issues. In order to address them, SBGP uses three tools: Public Key Infrastructure (PKI), an optional BGP transitive path attribute (attestations), and IPSec. In large part, the SBGP has not been deployed because of the stalemate between Internet registries, router vendors, and Internet Service Providers (ISPs). Each organization cannot justify investing in SBGP without the others having consented to do so also [14].

### **Public Key Infrastructure**

According to Internet research department from BBN technologies, “PKI is used to support the authentication of ownership of IP address blocks, ownership of Autonomous System (AS) numbers, an AS's identity, and a BGP router's identity and its authorization to represent an AS. This PKI parallels the IP address and AS number assignment system and takes advantage of the existing infrastructure (Internet registries, etc.)” The PKI will use an existing certificate management system with a few modifications. The resulting new Certificate Authority (CA) will be set up at an Internet registry. The CA establishes an organization to use a block of IP addresses. An IPAddrBlocks certificate allows for verification that an organization indeed has the right to the IP addresses by verifying a signature within the certificate. Secondly, it establishes the right for an organization to have an AS number (ASN) and a way to identify AS's. Lastly it allows for routers to have an association with ASes. However, there has not been any implementation of PKI in routers to determine its real-world performance. BBN claims this is because of the aforementioned stalemate [2].

### **Attestations**

The aforementioned additional transitive path attribute uses digital signatures to protect attestations in order to cover the information in a BGP UPDATE. This is used in conjunction with the PKI to allow for authentication of both address and path information. Attestations are described best by BBN itself:

“They enable each S-BGP speaker that receives a route advertisement to verify that each AS along the path has been authorized by the preceding AS to advertise the route, and that the originating AS has been authorized to advertise those prefixes by the entity with the right to use each IP address prefix contained in the UPDATE [14].”

Attestations come in two forms: Route Attestation (RAs) and Address Attestations (AAs). The route attestations are used in the optional transitive path and are

additionally protected by the digital signatures. Address attestations are when the sender (signer) of the attestation has the rights to route to one or more ASes specified in the prefix address [14].

## **IPSec**

According to BBN IPSec “is used to provide BGP control traffic with data and partial sequence integrity, and with peer entity authentication.” IPSec protects the reliability in TCP connections of organizations using BGP by implementing it at the IP layer. In order to prevent the DoS attacks commonly associated with BGP, SBGP uses anti-replay mechanisms. It is faster than a normal TCP detection system thereby reducing the effect of the DoS attack [14]. SBGP focuses mainly on closing the holes in security by allowing for the scalability of authenticating users and the legitimacy of BGP control traffic. It attempts to be operable with current BGP settings, but has many opponents. Any actual implementation of a new standard will be many years in the future [14].

## **Conclusion**

With the various security issues discussed throughout this paper we have come across several key problems. There are many vulnerabilities in BGP messaging. Messages are easily forged and disrupted when transmitted. Also, route redirection causes the flow of traffic from a network to be rerouted. The traffic is often routed to a compromised location or forced to a different network to cause overflows or flapping. Denial of service is simply an attack intended to prevent an organization or user from getting the resources they need. Different types of denial service attacks are SYN attacks, physical infrastructure attacks, teardrop attacks, viruses, and smurf attacks. Blackholing is used in preventing access to an AS or specific machine from a specific IP or domain. The packets are then dropped based on definite criterion. Route flapping occurs when routes to networks are advertised and quickly retracted. This could cause frequent short-term outages. Pretty good BGP is a protocol that will improve upon the existing BGP. Implementing PGBGP will help close common attacks that occur upon current BGP networks. Secure BGP would update the existing BGP network by including a variety of verifications in to the system while maintaining the speed of the existing network.



## References

- [1] "Routing Protocols." Network Dictionary. 24 Oct. 2007  
<<http://www.networkdictionary.com/protocols/routing.php>>.
- [2] Van Beijnum, Iljitsch. BGP. Sebastopol, CA: O'Reilly Media, 2002.
- [3] Mattias, Jansson. "RIP." KTH School of Computer Science. 4 Feb. 2004. KTH.  
26 Oct. 2007 <<http://www.nada.kth.se/kurser/kth/2D1490/04/lectures/rip.pdf>>.
- [4] Saaristo, Sampo. "Implementation of IS-IS Routing Protocol for IP Versions 4 and 6." Tampere University of Technology. 4 Oct. 2002. Tampere University of Technology. 27 Oct. 2007  
<[http://www.cs.tut.fi/tlt/npg/icefin/documents/isis\\_impl.pdf](http://www.cs.tut.fi/tlt/npg/icefin/documents/isis_impl.pdf)>.
- [5] "Border Gateway Protocol." Wikipedia. 27 Oct. 2007. 27 Oct. 2007  
<[http://en.wikipedia.org/wiki/Border\\_Gateway\\_Protocol](http://en.wikipedia.org/wiki/Border_Gateway_Protocol)>.
- [6] "Exterior Gateway Protocol." Wikipedia. 2 Oct. 2007. 30 Oct. 2007  
<[http://en.wikipedia.org/wiki/Exterior\\_Gateway\\_Protocol](http://en.wikipedia.org/wiki/Exterior_Gateway_Protocol)>.
- [7] "BGP." Advanced Internet Routing Resources. 26 Sept. 2007. 30 Oct. 2007  
<<http://www.bgp4.as/>>.
- [8] "BGP." Network Sorcery. 31 Oct. 2007  
<<http://www.networksorcery.com/enp/protocol/bgp.htm>>.
- [9] Huston, Geoff. "Exploring Autonomous System Numbers." Cisco. 4 Nov. 2007  
<Exploring Autonomous System Numbers>.
- [10] Davis, David. "How to Use BGP to Achieve Internet Redundancy." Tech Republic. 11 Mar. 2002. <[http://articles.techrepublic.com.com/5100-1035\\_11-1039765.html](http://articles.techrepublic.com.com/5100-1035_11-1039765.html)>.
- [11] Nordstrom, Ola, and Constantinos Dovrolis. Beware of BGP Attacks. Georgia Institute of Technology. ACM, 2004. 4 Nov. 2007.
- [12] Kranakis, Evangelos, P.c. Van Oorschot, and Tao Wan. Security Issues in the Border Gateway Protocol (BGP). Carleton University. 2005. 4 Nov. 2007.
- [13] "Articles on Security". Dec 02, 2007. <<http://www.appinlabs.com/articles-on-security.php>>.
- [14] Mikkelsen, Joanne and Seo, Karen and Lynn, Charles. "Secure BGP (S-BGP)". June 2003. Dec 02, 2007. <<http://www.ir.bbn.com/sbgp/draft-clynn-s-bgp-protocol-01.txt> (used in SBGP)>.
- [15] Karlin, Josh and Forrest, Stephanie and Rexford, Jennifer. "Pretty Good BGP: Improving BGP by Cautiously Adopting Routes". Dec 02, 2007.  
<<http://www.cs.princeton.edu/~jrex/papers/pgbgp.pdf>>.
- [16] "Internet Alert Registry". Dec 02, 2007. <<http://iar.cs.unm.edu/>>.
- [17] Convery, Sean and Franz, Matthew. "BGP Vulnerability Testing: Separating Fact from FUD v1.1". Dec 02, 2007. <<http://www.nanog.org>>.
- [18] "Denial of Service". Oct 23, 2007. Dec 02, 2007.  
<[http://searchsoftwarequality.techtarget.com/sDefinition/0,,sid92\\_gci213591,00.html](http://searchsoftwarequality.techtarget.com/sDefinition/0,,sid92_gci213591,00.html)>
- [19] Murphy, S. "BGP Security Vulnerabilities Analysis". Jan, 2006. Dec 01, 2007.  
<<http://www.rfc-archive.org/getrfc.php?rfc-4272>>.
- [20] "Route Flapping". Dec 02, 2007 <[http://en.wikipedia.org/wiki/Route\\_flapping](http://en.wikipedia.org/wiki/Route_flapping)>.

[21] Farley, Toni and McDaniel, Patrick and Butler, Kevin. "A Survey of BGP Security Issues and Solutions". AT&T Labs Research. Dec 02, 2007.  
<[http://www.cs.purdue.edu/homes/ninghui/readings/TruSe\\_fall04/td-5u](http://www.cs.purdue.edu/homes/ninghui/readings/TruSe_fall04/td-5u)

# Content-Aware Image Resizing

Kirk Wienkes and Dr. Kenny Hunt

Computer Science Department

The University of Wisconsin-La Crosse

La Crosse, WI 54601

email: wienkes.kirk@students.uwlax.edu, hunt.kenn@uwlax.edu

## Abstract

Recent advances in image display and document layout have given rise to the notion that resizing of images should not necessarily resample an entire image but that resizing should consider the content of an image. This article is based upon previously published research on the content-aware image resizing technique known as *seam-carving* as presented by Avidan and Shamir [1]. The seam carving technique uses seams which are defined as an 8-connected path of pixels from top to bottom, or from left to right. This technique uses the repeated removal of seams of least importance where importance is based on an energy function image. By removing seams the algorithm retargets the image size to the users desired width and height. By removing the least important seams those parts of the image having the most information content are preserved while truncating the least significant image pixels. Unfortunately, a pure seam carving implementation suffers from poor performance on a certain class of images, mainly images with a strong diagonal edge. Here we present an implementation of the seam carving method, a verification of known results, and an explanation on the seam carving algorithm for implementing effective image resizing.

# 1 Background

In today's world, the internet is integral to everyday life and can be accessed by people throughout the world using a vast variety of computing systems. Unlike at its origins, the internet is no longer bound to personal computers and mainframes, and thus web pages can no longer assume that they will be displayed by a computer monitor using a prescribed resolution and aspect ratio. The internet can be accessed and displayed on everything from small-screened cell phones to large-screened high-definition monitors. Thus web designers have to be able to dynamically adjust their web page content to adjust to the capabilities of the displaying device. While dynamic webpage layouts have given web designers the ability to scale and move textual information, dynamic webpage layouts generally do little with respect to retargeting images. Web designers therefore require tools to dynamically retarget images in their online documents to accommodate any type of display. Currently, there are three main methods of resizing: cropping, padding, and retargeting. All of these common methods are based solely on the dimensions of an image rather than the content.

The first method, cropping, requires that the desired dimensions be smaller than the original dimensions. The cropping method removes pixels from around the edges of the image. Thus all the content that is on these portions of the image that are thrown away and their information is completely lost. There is no effort on the part of this method of image resizing to save the complete image only just a sub rectangle of the image. The second method, padding, requires that the desired dimensions be larger than the original dimensions. Padding as a method of resizing simply adds background pixels around the image in order to make it conform to the desired dimensions. This can be done by always adding pixels off the right and bottom edges of the image so that the content of the image remains in the upper-left hand corner or equally around the image so that the content of the image remains in the center. Padding adds no content or added clarity to the image; it simply adds a border.

The third method, retargeting, unlike cropping and padding has no restriction on the size of the desired dimensions in relation to the original dimensions of the image. The method of retargeting uses a mapping function between the source and retargeted image. Most retargeting methods use an inverse mapping function that defines a mapping function from the pixels of the destination image to the pixels of the source image. This prevents holes that can occur when using a regular mapping function, which maps a function from the pixels of the source image to the pixels of the destination image. Some of the problems that occur with retargeting deal with how it can distort the images since the functions are not a one-to-one correspondence and they often point in between pixels. The two main ways of dealing with this is done by either picking the nearest pixel or the using some type of interpolation to approximate what that point in between pixels would be. The main feature that these traditional resizing methods lack is that they do not take into account the content of the image. They only concern themselves with the dimensions of the image. The deficiency in this approach is that all image pixels are treated as of equal importance when retargeting such that a relatively unimportant *background* pixel takes on as much significance as a pixel that defines an object of interest in the scene.

Realizing the deficiencies and limitations of a purely geometric resizing algorithm, the next step in image resizing needs to take into account what is in the image or the content of the image. This brings us to content-aware image resizing methods. Due to the lack of efficient content-aware resizing algorithms, there is a lot of interest in content-aware image resizing. A content-aware image resizing algorithm will look at the features of an image and try to protect these features as it resizes the image. This way the image only loses unimportant information and retains the important information. In 2007, Shai Avidan and Ariel Shamir published a paper describing a content-aware image resizing algorithm named seam carving [1]. At the end of the paper, the authors expressed the limitations of their algorithm. According to the authors, there are certain types of images that break this algorithm: images with too many features, images with relative spatial requirements such as faces, and images that have content that prevents the seam from bypassing them. By using face detection algorithms, Avidan and Shamir were able to get a handle on this problem with faces, but they left the other two cases up to future work. Most of this paper will be heavily based on the work done by Avidan and Shamir in their paper. Our purpose will be implementing their seam carving algorithm, verifying their results, and eventually expanding on their research. In this paper, the technical explanations are derived from the paper by Avidan and Shamir, but the implementation of their seam carving algorithm is our own contribution.

## 2 Overview of Seam Carving

### 2.1 Energy Function

Seam carving makes use of an *energy function* in order to determine the relative importance of pixels within an image. A pixel that is important to the content of the image is said to have a *high energy* and a pixel that is not important to the content of the image is said to have a *low energy*. There are many different energy functions that can be used to determine the energy of a pixel. Using the observation by Avidan and Shamir, we decide to implement and use the two energy functions that they deemed *best* in most circumstances:  $e_1$  and  $e_{HoG}$ .

The  $e_1$  energy function is an approximate measure of the magnitude of the gradients in an image. It combines the image gradients in the vertical and horizontal directions by adding their absolute values. For an image  $I$  the energy function  $e_1$  is defined as

$$e_1(I) = \left| \frac{\delta I}{\delta x} \right| + \left| \frac{\delta I}{\delta y} \right| \quad (1)$$

For computing the oriented gradients  $\delta x$  and  $\delta y$  of an image  $I$  we convolve with the proper Sobel mask. The histogram of gradients energy,  $e_{HoG}$ , attempts to measure image gradients but in a more sophisticated fashion. The  $e_{HoG}$  function is defined by the following equation.

$$e_{HoG}(I) = \frac{\left| \frac{\delta I}{\delta x} \right| + \left| \frac{\delta I}{\delta y} \right|}{\max(HoG(I(x, y)))} \quad (2)$$

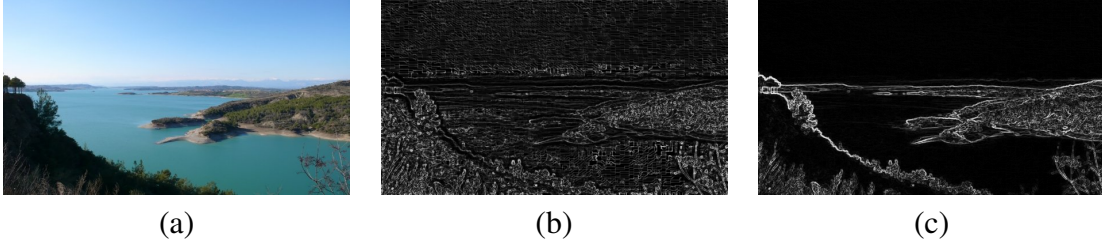


Figure 1: Source image (a) and the pixel energies as computed by  $e_{HoG}$  (b) and  $e_1$  (c).

$HoG(I(x, y))$  is the histogram of the oriented gradients at every pixel. This is computed using an 8-bin angular histogram computed on an 11 by 11 mask around each image pixel. The denominator of this energy function takes the seam toward the edges of an image while the numerator helps to ensure that the seams run parallel to the edges so that they do not cross to edge. This energy function has been used in the area of computer vision in the detection of humans in an image [2]. Figure 1 gives a visualization of these energy functions. The center image is a visual representation of the histogram of oriented gradients energy function of the source image on the right [5] while the rightmost image is a visual representation of the  $e_1$  energy function on the image on the left.

## 2.2 Seams

A vertical seam is an 8-connected path of pixels from top to bottom containing one, and only one, pixel in each row of the image. A mathematically precise definition of a vertical seam on an image  $I$  with dimensions  $n \times m$  is the following:

$$S^x = \{S_i^x\}_{i=1}^n = \{(x(i), i)\}_{i=1}^n, \forall i \{|x(i) - x(i-1)| \leq 1\} \quad (3)$$

Similarly, a horizontal seam is an 8-connected path of pixels in an image from left to right containing one, and only one, pixel in each column of the image. A mathematical definition of a horizontal seam on an image  $I$  with dimensions  $n \times m$  is the following:

$$S^y = \{S_j^y\}_{j=1}^m = \{(y, y(j))\}_{j=1}^m, \forall j \{|y(j) - y(j-1)| \leq 1\} \quad (4)$$

Hence, when removing either a vertical or horizontal seam from an image, the image is reduced in either width or height by exactly one pixel. It is also important to note that when removing a seam from an image a shifting of pixels occurs. For horizontal seams, all the pixels below the seam are shifted up to fill the space where the seam once was. Similarly for vertical seams, all the pixels to the right of the seam are shifted left to fill the space where the seam once was. In order to select the seam that contains the least information with respect to the content of the image we choose the seam where the sum of the energies of the pixels in the seams is minimized. The energies of vertical seams  $S^x$  and horizontal seams  $S^y$  are defined as

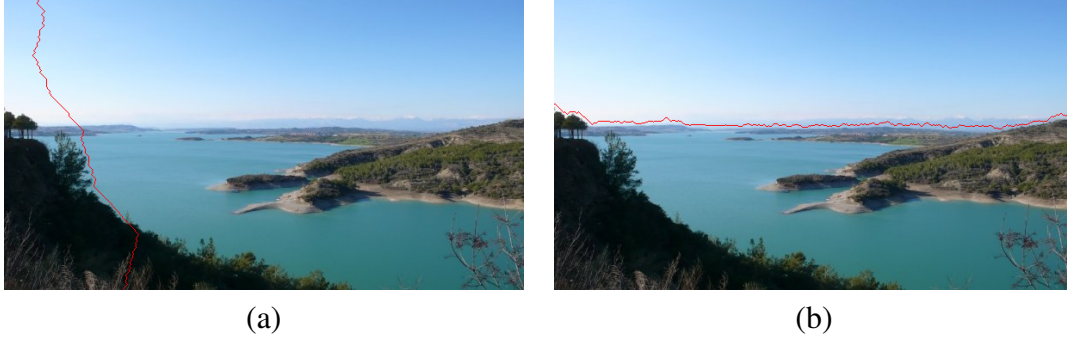


Figure 2: Minimal cost vertical seam (a) and horizontal seam (b) shown in red using  $e_{HoG}$ .

$$e(S_x) = \sum_{i=1}^n e(S_i^x) \quad (5)$$

$$e(S_y) = \sum_{j=1}^n e(S_j^y) \quad (6)$$

For maintaining the most energy or content in the image it is important to remove the optimal seam in an image, designated as  $S^*$ . The optimal seam is defined as the seam with the least energy.

$$S^* = \min(\forall S \in I\{e(S)\}) \quad (7)$$

To find this optimal seam, use can use a dynamic programming algorithm. Let  $M(i, j)$  be the energy of the minimum partial path at location  $(i, j)$  of an image  $I$  as the seam is traversed from top-to-bottom in a vertical seam. Note that the technique for horizontal seams is a straightforward extension of this description. For all pixels in the top row of an image, the minimum partial path energy is given as  $M(0, j) = I(0, j)$  for all column indices  $j$ . For all other rows of the image, the minimum partial path energy is computed using a linear raster scan using the following formulation:

$$M(i, j) = e(i, j) + \min(M(i-1, j-1), M(i-1, j), M(i-1, j+1)) \quad (8)$$

The minimum cost vertical seam is the path having the minimum  $M$  value in the bottom row of the image. To figure the rest of the path we simply, backtrack through  $M(i, j)$  collecting the smallest connected partial paths on the way back to the first row. Figure 2 illustrates how the minimal cost seam captures the notion of a path of least important image content. The image on the left displays in red the minimal energy vertical seam using the histogram of oriented gradients. The image on the right displays in red the optimal horizontal seam for removal using the histogram of oriented gradients energy function.

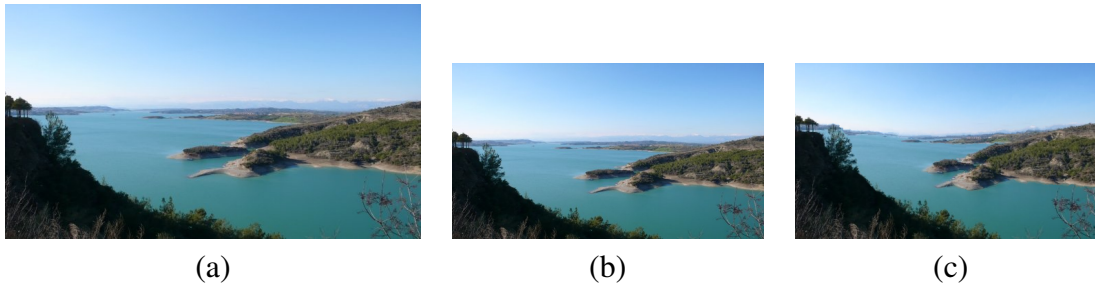


Figure 3: Source image (a) is resized to 75% of its original size using seam carving with  $e_{HoG}$  to produce (b).

## 2.3 Image Resizing

Having definitions for energy functions and seams, we can describe image resizing using seam carving. For this paper, we will only be discussing reducing images, a discussion of enlarging images can be found in the Avidan and Shamir paper. Consider image  $I$  with dimensions  $n \times m$  and assume that we want to resize  $I$  into  $I'$  with dimensions  $n' \times m'$ . Using seam removal, we will need to remove  $n - n'$  horizontal seams and  $m - m'$  vertical seams from  $I$  to create  $I'$ . For this paper, there are four approaches for selecting the seams to be removed: 1) remove all horizontal seams first 2) remove all vertical seams first 3) alternate between removing horizontal and vertical seams and 4) and alternate between vertical and horizontal seams. Our implementation chooses to alternate between vertical and horizontal seams.

Figure 3 shows how seam removal attempts to keep image content while eliminating pixels of little relative import. A source image is reduced by 25% using the traditional resampling approach to obtain the image shown in (b) while seam removal is used to obtain the image shown in (c). Comparing the skylines of the two reduced images reveals that seam removal has kept more of the foreground and eliminated more of the sky than the traditional approach. Also, the seam removal technique tends to dispense with pixels in the waterway rather than the terrain and hence the spatial relationship between scene elements is altered.

## 3 Implementation

Due to our familiarity with Java and the Java digital image processing library in the `awt` package, we decided to use Java as our implementing language of choice. Our main goal was to create a correct solution without regard to efficiency. In order to speed up our research, we used the Java framework as the basis of our implementation for the seam carving algorithm. In this framework there are two main classes that we need to be concerned about the `BufferedImage` and the `BufferedImageOp`. The `BufferedImage` class is the basic image class of the framework; it stores all the information about an image. The `BufferedImageOp` is a class that performs some one-to-one operation on a source image resulting in a destina-



tion image. This operation is done using the filter method, which is the main method that should be overridden when subclassing the BufferedImageOp.

In our implementation, we created subclasses of the BufferedImageOp including RetargetOp, CropOp, SeamCarveResizeOp, SingleSeamCarveOp, and ColorSeamOp. The RetargetOp is an implementation of an inverse mapping retargeting resizing method. The CropOp is an implementation of a crop method. These two classes provide a baseline for comparison with our implementation of the seam carving algorithm. The SeamCarveResizeOp is the main class for our implementation of seam carving. It works by repeatedly calling SingleSeamCarveOp which will remove the least important horizontal or vertical seam as defined by an EnergyFunction which is the interface that we used to implement a variety of energy function. The two implementations of EnergyFunction were the HOGFunction and the MagnitudeGradientFunction. The HOGFunction gives an implementation of the histogram of gradients energy function described above and the MagnitudeGradientFunction gives an implementation of the  $e_1$  energy function described above. Finally, the ColorSeamOp provides the same functionality as the SingleSeamCarveOp except that instead of removing the least important seam; it colors the least important seam to give a visual aid on what seams were being removed from an image.

## 4 Results

In order to find the classes of images for which this method performs adequately, we ran our algorithm on images that represented vastly different categories of image types. Our classification included *computer generated images*, *nature images with strong edges*, *natural images with weak edges*, and *binary images*. Our assessment of the quality of the results is based on a subjective aesthetic evaluation rather than a quantitative metric.

### 4.1 Computer Generated Images

This class of images involves images created on a computer using synthetic techniques. Clip art and icons are a few examples of types of images in this category. As a whole this class of images did not perform well. Since a basketball is round, a good resizing algorithm should result in a round or oval-shaped basketball. The seam carving method does not perform well with this type of image since it has difficulties with maintaining proportionality of its results. Figure 4 shows the results of reducing a source image [4] by 25% using both traditional resizing and seam carving techniques. Traditional resizing gives obviously superior results.

### 4.2 Natural Images with Strong Edges

Natural images are those that are acquired using direct environmental sampling. In this case, we have chosen a subjectively determined *strong edge* criteria to mean any photograph

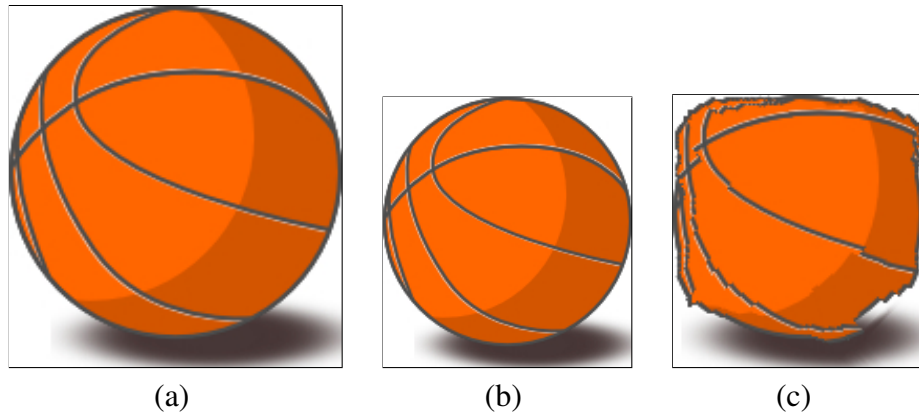


Figure 4: Source image (a) is resized to 75% of its original size using traditional resizing (b) and using seam carving with  $e_{HoG}$  (c).

having strongly outline objects within the scene of interest. Figure 5 gives a representative image of what we consider to be a natural image having a strong edge [6]. Strong edges within an image that span the entire width or height of an image are problematic, as was the case with the clip art example. In this case, the strong edge of the globe presents an edge that all horizontal seams must traverse and hence represent an edge that cannot be sustained throughout the seam carving retargeting.

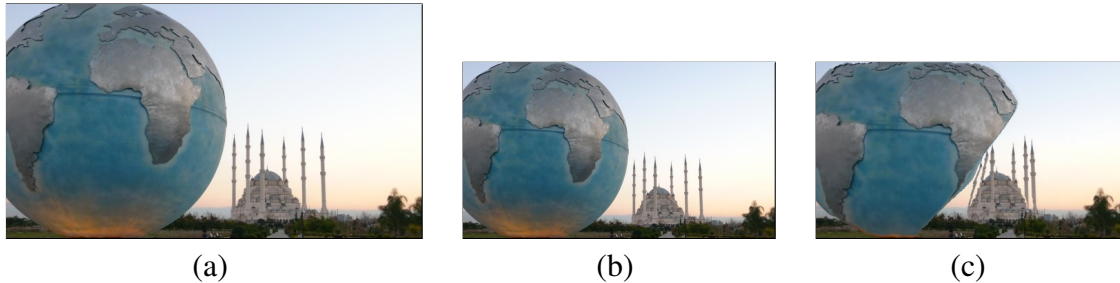


Figure 5: Source image (a) is resized to 75% of its original size using traditional resizing (b) and using seam carving with  $e_{HoG}$  (c).

### 4.3 Images with Weak Edges

Seam carving produces the best results with images that have large swaths of relatively weak edges throughout the image since wide horizontal or vertical seams can then be found that produce negligible effect when removed from the resulting image aesthetic. Figure 6 shows one such result [3].

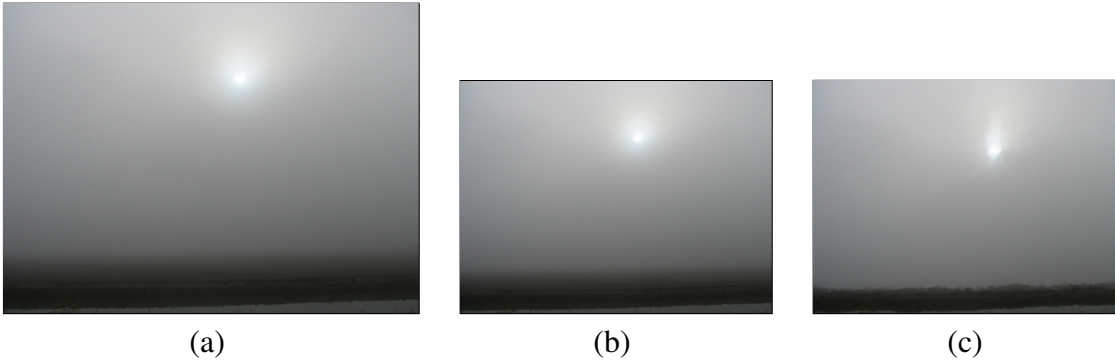


Figure 6: Source image (a) is resized to 75% of its original size using traditional resizing (b) and using seam carving with  $e_{HoG}$  (c).

## 4.4 Binary Images

It is interesting to see how this seam carving method works on a binary image. Figure 7 shows the results of easing seam carving a section of scanned text. Since there are relatively large pathways through the maze of lettering in both vertical and horizontal dimension, the algorithm produces reasonable results but produces characters that are spatially shifted in displeasing fashion.

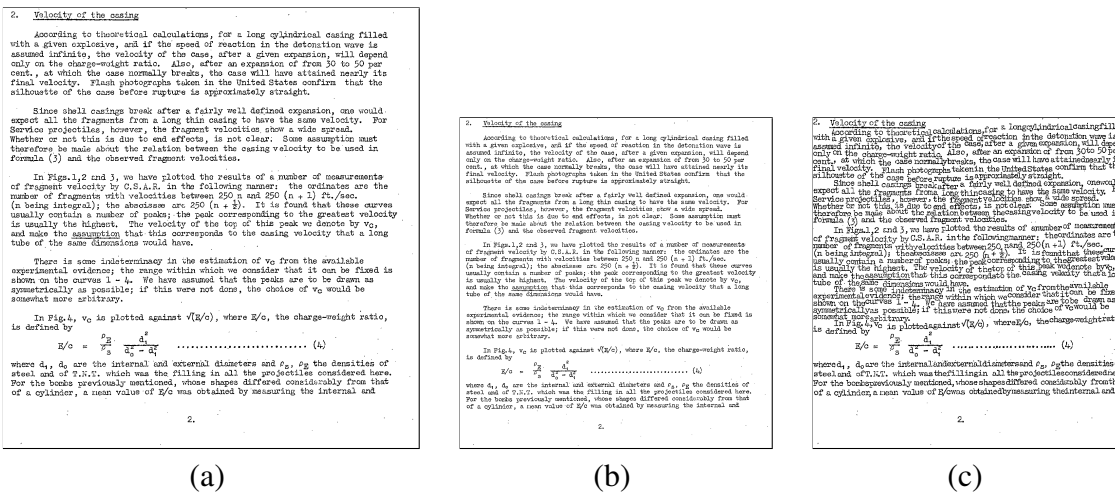


Figure 7: Source image (a) is resized to 75% of its original size using traditional resizing (b) and using seam carving with  $e_1$  (c).

## 5 Conclusion

Seam carving presents real potential for use in image retargeting as motivated by the desire to dynamically adapt document layout to various display devices. Seam carving does not,

however, present a general solution to image resizing since the content of an image may cause the image to be more or less amenable to seam carving. We hope to extend this research by developing a hybrid approach that attempts to dynamically examine an image and automatically determine those cases (or regions) which should be treated using the traditional hybrid approach and those that should be carved using the technique described in this article. Determining the correct way to *combine* these two very different approaches into a unified and automatic technique may present a powerful general solution to the problem of dynamic image retargeting.

## References

- [1] Shai Avidan and Ariel Shamir. Seam carving for content-aware image resizing. *ACM Trans. Graph.*, 26(3):10, 2007.
- [2] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. *International Conference on Computer Vision & Pattern Recognition*, 2:886–893, 2005.
- [3] emdot. <http://www.flickr.com/photos/emdot/73257388/>, 2005.
- [4] Stellaris. <http://openclipart.org/media/files/Stellaris/1178>, 2006.
- [5] A. Wellsfry. Personal photo used by permission of artist., 2008.
- [6] A. Wellsfry. Personal photo used by permission of artist., 2008.

# Tracking a Rat in Three-Dimensional Space Using Stereo Cameras

Nathaniel Meierpolys	Michael Krahulec	Daniel Wiebe
Computer Science Program	Computer Science Program	Computer Science Program
St. Olaf College	St. Olaf College	St. Olaf College
Northfield, MN 55057	Northfield, MN 55057	Northfield, MN 55057
meierpol@stolaf.edu	krahulec@stolaf.edu	wiebe@stolaf.edu
Olaf Hall-Holt	Reid Price	
Computer Science Program	Computer Science Program	
St. Olaf College	St. Olaf College	
Northfield, MN 55057	Northfield, MN 55057	
olaf@stolaf.edu	reid.price@gmail.com	

## Abstract

We consider the problem of determining the position and head orientation of a rat in three-dimensional space based off of two colored LED lights attached to the rat's head. In order to track the rat using only standard cameras (instead of range-finder technology), we use two pre-calibrated cameras for triangulation. To find the LED lights in the camera images, we segment the image based on color and gradient data, storing the information in a half-edge data structure. We then select specific faces as LEDs based on the expected color and relative size of the LEDs. Finally, based off the calibration information, we can triangulate the LEDs to within 0.3 cm. The results of this process will be used in conjunction with brain wave activity data as part of an interdisciplinary study of the neural basis for spatial navigation in a rat.

## Introduction

The problem of automatic detection and tracking of objects is a familiar one in the field of computer vision and has been analyzed in various contexts. The difficulty with this problem lies twofold in the large amount of video information to be processed and the inability of computers to recognize coherent objects and shapes the same way humans do. Basic objects that appear clear and concrete to the human eye are actually rather complex. Objects contain textures and shadows, potential for rotation and changing orientation, and often blend into backgrounds, making them hard to identify.

Using data from a pair of high-quality video cameras, we locate two LED lights mounted to the head of a rat. The front and back LEDs have a different color, and of the position of each colored LED allows us to measure the orientation of the rat at any given time. This information can be combined with measurements of individual neurons in the brain to study the neural basis for spatial navigation in the rat.

The most difficult aspect of our case is the nature of the object being tracked. When tracking a live animal, we encounter unpredictable difficulties with occlusion from objects in the maze. We also encounter moments when the physical LEDs reflect off walls to create virtual ones in the center of the reflection, confusing our system. If these complications weren't enough, the movement of the rat changes the orientation of the LEDs at varying speeds, which makes tracking the rat's movements very difficult.

Our approach relies on image data from two cameras, without need for range-finding equipment. Processing data is performed by a software pipeline involving four distinct components. Cameras are calibrated using a calibration target within view of both cameras. Images of the rat from two cameras are segmented in parallel with the help of a powerful Beowulf cluster environment. Images are then processed to determine the location of LED lights using a doubly-linked half-edge data structure. Finally, identified points are triangulated into three-dimensional space from pairs of stereo images. The process is structured to allow the processing of many frames automatically. We sacrifice real-time speed for highly precise measurement of LED light position.

## Calibration

In order to utilize images from cameras to triangulate the positions of objects in three-dimensional space, we must develop a concept of what a "camera" is, and how we can translate the images it gives into meaningful data. We consider a pixel on an image as color and position information about a particular ray of light that passes through the focal point of the camera and a specific place on the light receptor of the camera (See Figure 1). As seen in the diagram, the actual image taken by the camera can be placed in space such that the ray of light passes through the pixel on the image. If we can determine the three-dimensional position of the camera's focal point, and the size and appropriate position of where the image can be placed in space, then we know everything about the camera needed for triangulation. Throughout this next section, we will call the correct position and orientation of the image the camera's *screen*.

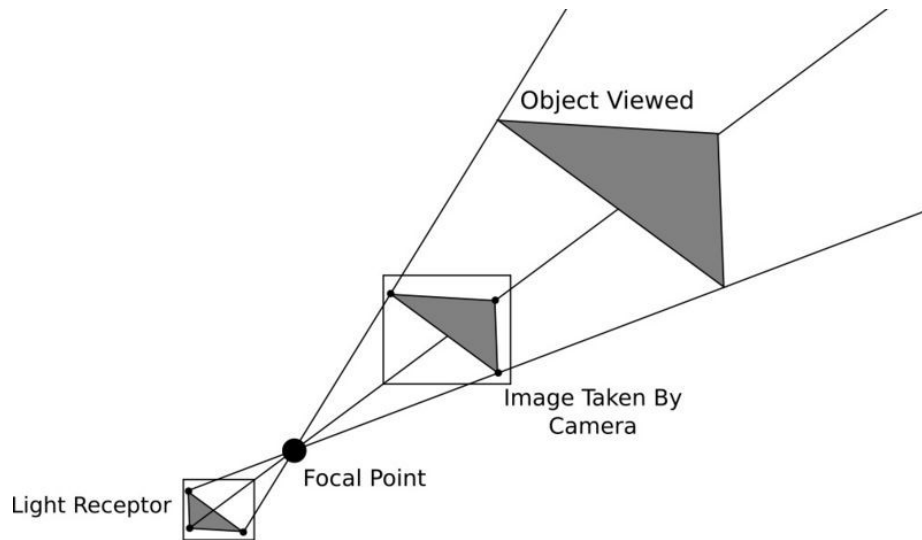


Figure 1: A Basic Camera

Calibration is the process of generating a three-dimensional coordinate system and determining the positions of the screen and the focal point of a camera in that coordinate system. In our method, we set up the cameras in the position they will be throughout the whole experiment. We then place a calibration target within clear sight of both cameras. This target is a grid of black dots on a white sheet of paper, with an odd number of rows and columns (Fig.3). The distance between the grid points is known, giving us a clear measure of length in the image. To help simplify our algorithm, the cameras must be set up such that pictures of the calibration target show the rows of dots to be parallel to the top and bottom edges of the image. The correct placement of cameras is displayed in (Fig. 2).

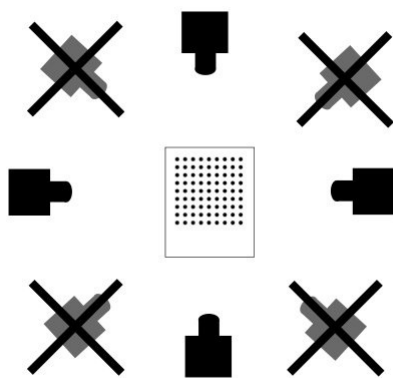


Figure 2: Example camera layout

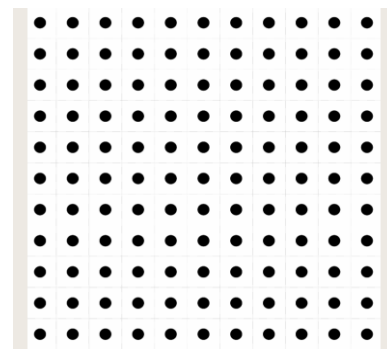


Figure 3: Calibration Grid

Once we've taken an image of the calibration target with both cameras, we continue with the experiment to its conclusion, for the calibration process can occur later, when we are actually processing the data.

Our calibration algorithm takes the images containing the calibration targets and produces

all the necessary position data about the cameras. The first step in this algorithm is identifying the dots in the picture and putting them in grid-order. In order to do this, we generate a mask of a standard black dot with a square white background of a size inputted by the user. Next, we loop through all the pixels, starting at the upper left-hand corner of the image, placing the mask over the image, centered at the current pixel. We then count up the differences in color of all the pixels in the mask compared to the image pixels. This value is normalized and stored as color data in a new image at the location of the original pixel. The result of this process is also known as a disparity map. In order to isolate the grid points, we filter this disparity map, blacking out any pixels that have a disparity higher than a given tolerance level. We then find the pixel with smallest disparity in each dot-sized region of interest to determine the positions of the dots in the image. As long as the dots in the image were well-lit, this algorithm clearly identifies the positions of the dots.

The central dot is assumed to be the origin of our three-dimensional coordinate system. In order to determine the camera parameters with reference to the grid, we first guess at a series of camera parameters. Then, based on these parameters, and the distance between the grid points, we generate a virtual grid of points, and display what would appear on this camera's screen (See Figure 4). We then continuously vary the camera parameters slightly until we minimize the difference between the virtual model and physical reality.

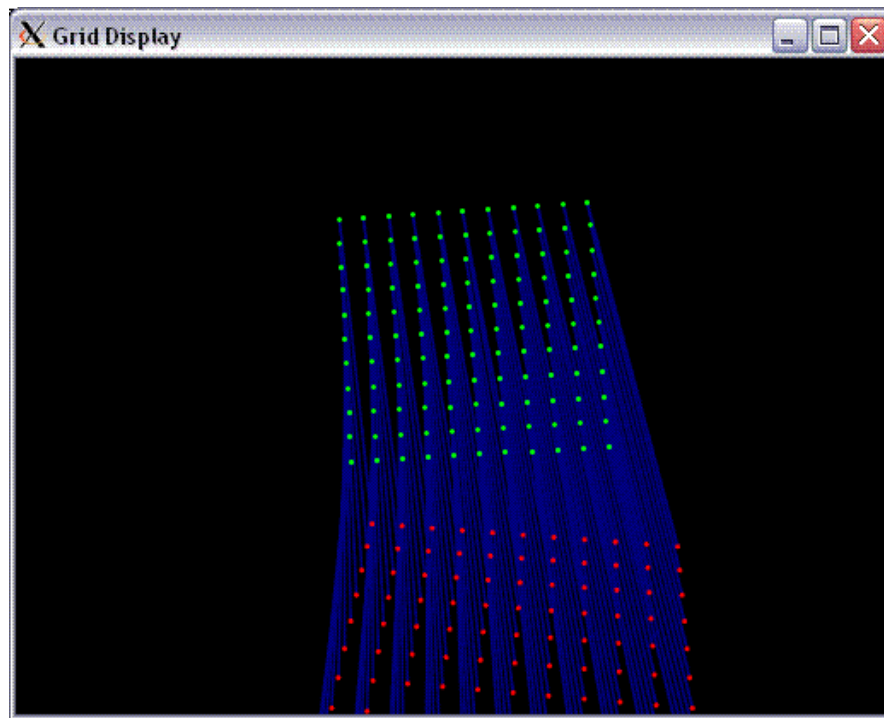


Figure 4: Virtual grid vs. Physical Grid

## Capturing Video Data

### Camera Settings

We use two Sony BRC-300 cameras which allow manual control of shutter speed, iris, gain and zoom. The task of picking out bright LED lights can be greatly simplified using these



manual settings. A high shutter speed means that the image sensor is exposed to light for a shorter period of time, causing LED lights to stand out against less bright elements of the background. Shorter length of exposure also reduces the risk of motion blur. Maintaining a low aperture setting opens the iris wider. This increases the depth of field, keeping more components of the image in focus. Both settings contribute to the reliability of our procedure for identifying LEDs.

### Scanline Issues

The cameras we use record interlaced video to effectively double the perceived frame rate of the video. The technique of creating interlaced video involves capturing all odd scanlines in one instant then all even scanlines in the next and finally piecing the two fields together to produce a single frame. This process improves the smoothness of video while reducing the amount of information to be broadcast or stored. In the context of our image-processing needs, however, the jagged edges make our analysis more difficult and less accurate. To deal with this problem and deinterlace image content, we first separate odd and even fields and then double each line to produce two distinct images from each frame recorded. The doubling ensures the maintenance of a consistent aspect ratio but unfortunately comes at the cost of losing vertical pixel precision since we cannot know exactly how closely our copied scanline matches the actual light entering the camera at that point.

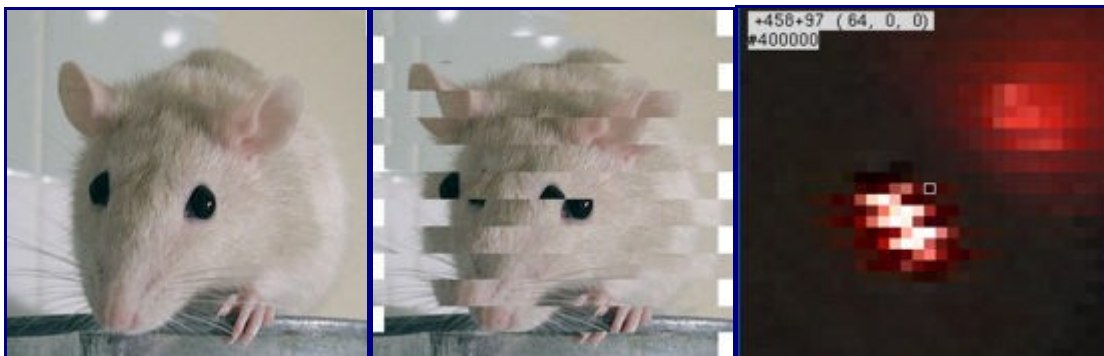


Figure 5: From left to right: Normal, deinterlaced image. The same image, with interlaced fields during movement. An actual interlaced LED image

Our relatively simple solution depends on knowledge of whether a line was initially copied up or down. For odd fields where all scanlines are copied down, we are essentially skewing the location of area within the face downwards. The same is true in the opposite direction for even fields where scanlines are copied up. To compensate for this, we adjust the vertical position of an LED's calculated center up by .5 pixels for odd scanline images and down .5 pixels for even scanline images. The extra information gained by processing both odd and even fields allows us to more easily overcome cases in which the LED light is occluded in one scanline and visible in the other as is generally the case.

### Segmentation

In order to get any meaningful information out of the images we collect, it is necessary to analyze the image data for coherent shapes and color regions. We use a program

developed by our co-researchers to segment our images. The program, called Eriol, breaks an image into shapes called faces and stores the information in a half-edge data structure. The Halfedge is a edge-centered data structure that is great for representing a polygon mesh, the exact structure of segmentations. The Halfedge Data structure received its name because instead of storing full edges of the mesh, we only store halfedges, where two halfedges together form a edge pair, and are referred to as "twins". This way each halfedge only belongs to a single incident face. Below is an illustration of a small portion of a halfedge structure.

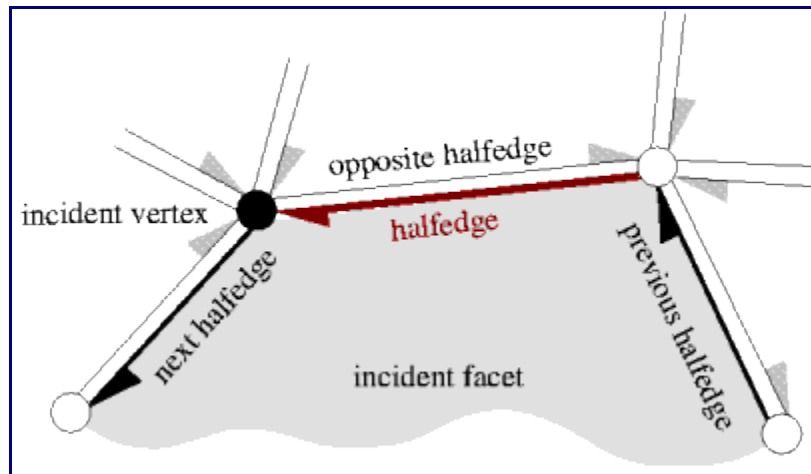


Figure 6: Halfedge example

The Halfedge data structure uses pointers to connect each face to its "outercomponent"(outer edge). Then, each halfedge has pointers to both the next edge, it's twin, and the incident face. This structure allows us to quickly move around the data structure and retrieve data in constant time per each piece of information gathered. We use the data stored in the faces of the structure to identify LED location and information more easily than sifting through raw image data.

The segmentation process itself is driven by the minimization of a cost function including four terms. These terms are *variance* (the change in color within faces), *boundary length* of faces, the *gradient* (change in color along edges), and the *number of segments* needed to describe the segmentation. The process starts with a grid containing the maximum number of pixel size squares and removes segments until the cost function reaches a minima.

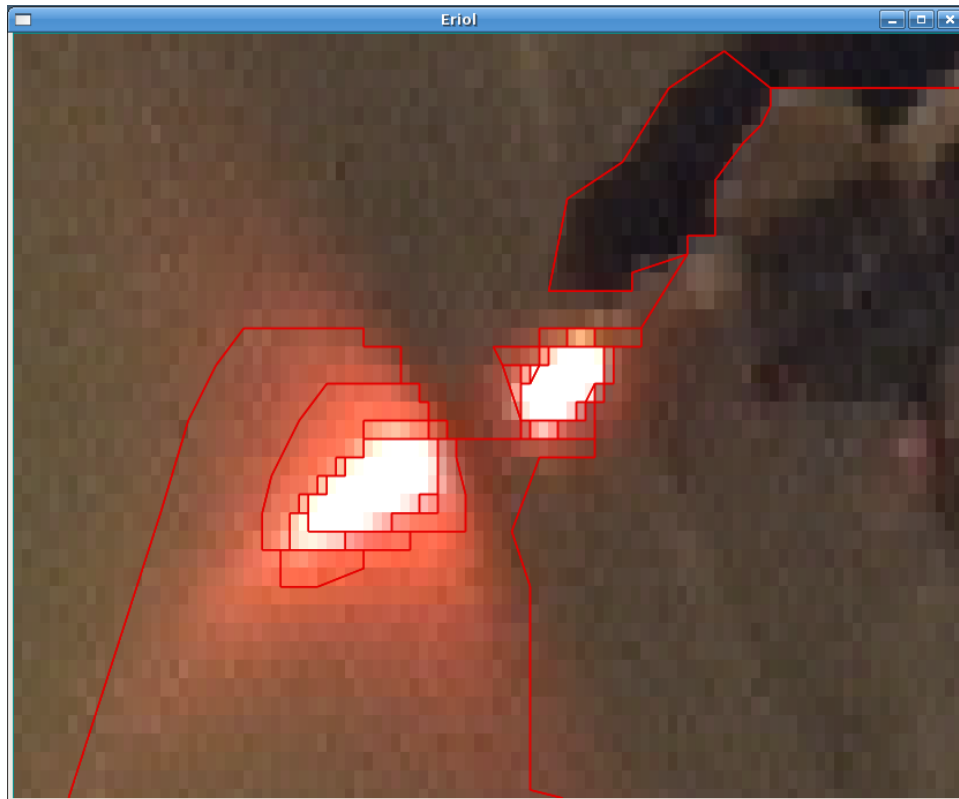


Figure 7: A segmented image in Eriol. The upper-right bright spot is the actual orange LED, whereas the bottom left one is its reflection. The remarkable size of the reflection bright segmentation is atypical, though it must be taken into consideration.

The very precise nature of this segmentation involves significant processing time. We employ the use of a Beowulf cluster of up to 34 machines, facilitated by the use of MPI message passing software. Together the cluster machines can segment many frames in parallel, increasing the speed of the process. Whereas a single machine processing a frame requires approximately 15 minutes to complete segmentation, many machines working in parallel reduce this task by a significant factor.

## Finding LEDs

### Picking out faces

Using the Halfedge data structure of the image segmentations, we can easily query the whole image to find the brightest faces in our segmentations. The LEDs are so bright that they appear overexposed in the image, creating a bright white spot where the center area of the LED is located. Because we have used the increased shutter speed to decrease the amount of light entering the camera, the two LEDs and possibly a reflection of one of the LEDs on the wall of the maze are the only bright spots that we find. Based on the area of the bright faces, we select the two largest bright spots as our LEDs, because the white center of the reflection of the LED on the wall is almost always much smaller than source

LED itself.

### Identifying Color

Once we have identified where the LEDs are located in the image, we need to find color information about them to distinguish between the two LEDs, red or green. To find the color of the LED, since the center face is only a bright white color, we must circle around the image to find information about the color of surrounding faces. Luckily, with the Halfedge data structure, we can easily find this information in constant time. We take the mean color of all of the faces around the LED, and then, because there are only red or green LEDs, we can look at which value, red or green, is greater in the mean RGB value of the faces, and based on this we know the color of the LED.

### Finding centroids

Once we have obtained the color of the LEDs, we need to know the exact location of the centroid of our LEDs on each image to enable us to find the correct three-dimensional position using triangulation. To do this, we use a function that uses the vertices of the face to find an extremely accurate position of the centroid. The function is as follows, where  $c_x$  is the x-coordinate of the centroid and  $c_y$  is the y-coordinate:

$$c_x = \frac{1}{6A} \sum_{i=0}^{N-1} (x_i + x_{i+1}) (x_i y_{i+1} - x_{i+1} y_i)$$
$$c_y = \frac{1}{6A} \sum_{i=0}^{N-1} (y_i + y_{i+1}) (x_i y_{i+1} - x_{i+1} y_i)$$

Figure 8: Centroid function

This function is great for use in our segmentation because it does not weigh the location of the centroid based on the distribution of vertices. The coordinates of the centroid on both the right and left images are then fed into our triangulation function to find the exact three-dimensional location in space.

### Triangulation

Once the two cameras have been calibrated, we are able to use pairs of camera images to locate an object in three dimensional space. Given the (x,y) position of a small object in each camera's image plane, we can generate the ray that passes through the focal point of the camera and the position on the camera's screen (see the figure below; for the definition of *screen*, see the Calibration section). The object observed necessarily lies somewhere along this ray in space. Now if we have two cameras, and they both observe the same object, the object must lie at the intersection of the two rays. In reality, there is naturally some error in the positions of the rays such that they do not intersect, but pass very close to

each other. Thus, our algorithm finds the midpoint of the shortest segment between the two rays, giving an approximation of the point of "intersection". A visualization of this process is shown in the figure below (Figure 9).

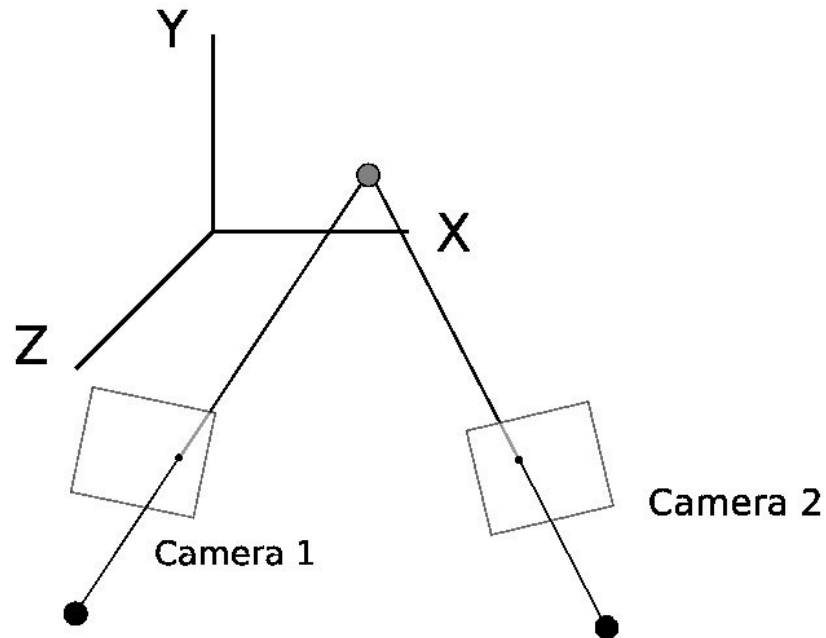


Figure 9: Triangulation Image

## Results and Error Analysis

Our algorithm successfully found the two LEDs in each frame only about 60% of the time. The rest of the time, it found one LED, two of the same color, or none at all.

For the triangulation portion of our algorithm, it is much more difficult to determine the error because we do not have an accurate "ground truth" set of images. To create a ground truth dataset, we would need to accurately pinpoint the LEDs' positions in some other way than with our cameras, whether with laser range-finding or with rulers. The former method is not cost-effective for a single test, and the latter is not accurate enough to give us any good comparison to our algorithm's results. Thus, the method we use is calculating one half the length of the shortest line connecting the two rays in the triangulation process. This measurement gives us a general idea of the error in our entire algorithm. During testing, we found that this error ranged from .06 cm up to .3 cm. We consider this to be a remarkable success, if indeed this is an accurate determination of our error.

## Conclusion and Future Direction

In conclusion, our algorithm is quite robust in triangulating the positions of the LEDs once they are found. However, our LED-finding algorithm will need improvement. We have a number of ideas on how to improve this aspect of our algorithm. First, although the size difference between the real LED and the reflection is generally a reliable distinguishing factor, we believe there could be instances when this characteristic may not be enough. Another potentially valuable method would be to look at the color spread around the two bright spots, since the color spread around a reflection is always significantly larger than the color spread around the real LED. It is also possible to devise a method to check accuracy and correctness based on the distance between the two LEDs found. Since the LEDs will always be the same distance apart from each other in three-dimensional space, information about the two potential LED positions makes it possible to check the distance between them. If they are not the correct distance apart, the algorithm must search further to obtain a new LED.

Occlusion of the LEDs is another problem that can occur in searching for the LEDs in our experiments. There is a wire that runs down from the ceiling to the head of the rat, and this cord can sometimes block the view of the LED for an instant, making it difficult to track. We are already looking at both of the alternating scanlines to help with this problem, but another possible resource could be the position of the LED in the image before and after the occlusion occurred. Considering information from these two sources would allow a very accurate estimate of the position of the LED, even in difficult cases.

These problems are of the foremost importance in the future direction of our research. Other explorations of the problem could involve moving the cameras to track the LEDs dynamically rather than our static setup where the cameras remain still. This would require constant re-calibration of the cameras and precise control of the camera movement. Once this is accomplished, the final step in our research would be to make all of these steps occur in real-time, rather than waiting for segmentations to be developed after the video has been taken and stored.

# Visualization of Energy Minimization in Ferromagnetic Systems

Zachary Oler  
Computer Science  
Drake University  
Des Moines, IA 50311  
zac.oler@gmail.com

Timothy Urness  
Computer Science  
Drake University  
Des Moines, IA 50311  
timothy.urness@drake.edu

## Abstract

Many different studies of ferromagnetism and anti-ferromagnetism models have presented theories on energy minimization. These studies, however, do not give visual confirmation of what is occurring during minimization. We wish to study how the energy minimizes locally in a ferromagnetic system. Where does the energy dissipate once the magnetic field is applied? Are there regions that exhibit a chaotic nature before eventually aligning with an external magnetic field? It has been demonstrated that the energy of the system will minimize in the presence of an external magnetic field. However, regions within the lattice may not converge at the same rate. Our goal is to develop and apply a visual tool to the system, which would allow users to visualize the minimization process. In this paper, we describe a model and visualization system designed to illustrate the principles of energy minimization in ferromagnetic systems.

# 1 Introduction

While magnets and magnetic fields are ubiquitous and largely understood, the subtle properties of atomic magnetic dipoles include complicated interactions between individual atoms and electrons [2, 5, 9, 10, 11, 12]. A better understanding of these magnetic, molecular interactions could be applied to various applications such as creating faster, more efficient random access memory (RAM) in modern computers.

We constructed software that models and visualizes magnetic dipoles in a lattice and studied how the dipoles interacted with one another. Our software models the energy minimization that naturally occurs in ferromagnetic materials using a basic model for the Hamiltonian. Then, our software uses OpenGL to produce real-time three-dimensional renderings of the interactions between magnetic dipoles.

Additionally, the software models the interaction of the dipoles with an external magnetic field. This serves to demonstrate real life scenarios and to provide a litmus test as to whether or not the energy minimization is being modeled appropriately and accurately. Finally, our software models thermodynamic effects. By modeling the thermodynamic energy (heat) of the system, our software can show things like the threshold at which a material loses its ferromagnetic properties.

## 2 Background

### 2.1 Physics

The Heisenberg model is a simple n-vector model that allows us to represent magnetic dipoles in a lattice and gives the Hamiltonian of an individual dipole. *The Hamiltonian directly corresponds to the total energy of the system.* A minimization of the total Hamiltonian indicates that the system has been allowed to align internally in the absence of an external magnetic field or has aligned with the external magnetic field. The Heisenberg model uses the Nearest Neighbor Principle in the calculation of the Hamiltonian. The Nearest Neighbor Principle states that only the surrounding dipoles' orientation will be considered to be most significant in calculation of the Hamiltonian. The surrounding dipoles in our system will be located above, below, left, right, in front and behind. The Heisenberg model defines the Hamiltonian of the  $j$ th dipole to be:

$$H_j = - \sum_i \vec{m}_i \cdot \vec{a}_j - \vec{B} \cdot \vec{a}_j \quad (1)$$

The  $\vec{a}_j$  is the vector that represents the  $j$ th dipole in the lattice. The  $\sum_i \vec{m}_i \cdot \vec{a}_j$  is the nearest neighbor sum where  $\vec{m}_i$  is the vector that represents the neighboring dipole.  $\vec{B}$  is the magnetic field of the system. An assumption of the Heisenberg model is that the magnitude of the dipole moment is one.



Additionally, we wish to model the effect of temperature on the Hamiltonian in the system. The energy of the  $j$ th dipole due to temperature is given by:

$$E_j = \frac{F}{2} k_b T \quad (2)$$

$T$  is the temperature of the entire system and  $k_b$  is the Boltzman constant.  $F$  represents the degrees of freedom. For our system, there are three rotational degrees of freedom. We will assume that this energy is due to the rotational kinetic energy of the  $j$ th dipole given by:

$$KE_j = \frac{1}{2} I \omega^2 \quad (3)$$

The  $I$  is the moment of inertia and  $\omega$  is the angular velocity defined by:

$$\omega = \frac{\Delta\theta}{\Delta t} \quad (4)$$

If we substitute  $\omega$  from Eq.(4) into Eq.(3) and set the kinetic energy equal to the thermal energy from Eq.(2) and solve for  $T$  we get formula for the temperature as a function of  $\Delta\theta$ :

$$T = \frac{1}{3} \left( \frac{I}{k_b \Delta t^2} \right) \Delta\theta^2 \quad (5)$$

Using this relationship between the temperature and  $\Delta\theta$ , we can model the temperature of the system by altering the amount of random rotation ( $\Delta\theta$ ). Another physical quantity that we will use is the magnetization. This is a measure of how the system has aligned with an external magnetic field. The magnetization is given by:

$$\vec{M} = \frac{1}{N} \sum_j \vec{a}_j \quad (6)$$

Typically we will take projections of the magnetization in the direction of the external magnetic field. This scalar quantity represents how much the dipoles have aligned with the external magnetic field.

Physically, we expect that applying these formulas will exhibit the following behaviors to system: If the temperature is zero ( $T = 0$ ), the system will minimize its energy and all vectors will align. If the temperature is not zero ( $T \neq 0$ ), then we expect that the energy will *stabilize* but may not become completely minimal. Lastly, if the temperature rises above the Curie temperature, the system will be unable to minimize and the dipole moments will not align. However, in the presence of a magnetic field, the system will exhibit paramagnetism, a form of magnetism that only exists in the presence of an external magnetic field. The algorithm we developed to model the system is presented in section 3.

## 2.2 Visualization

Many scientific visualization techniques have been developed for representing and understanding three-dimensional scalar and vector fields. Perhaps the most straightforward technique for visualizing a vector field is to use a series of lines or glyphs that

are tangent to the vector field. This technique is known as vector plots or hedgehogs [7].

In addition to the vector field, we also wish to visually represent the Hamiltonian that exists throughout the three-dimensional domain. Volume rendering is a classic visualization technique that uses color and opacity to represent a 3D scalar field. The process begins by assigning a scalar value to each point in the domain. A *transfer function* is defined to map each scalar value to a distinct color and level of transparency [8]. The image is rendered by taking a number of 2D slices through the 3D volume that is to be rendered. The slices are colored—or texture mapped—according to the transfer function at each point in the domain. The final image is constructed by compositing the transparency values within the slices to form a final 2D image that accurately depicts the 3D volume.

Field et al. have developed a technique that combines volume rendering with other vector field representations to visualize multiple quantities in the same three dimensional domain [3]. Our work is inspired by these techniques and we seek to expand these methods to better understand the minimization of the Hamiltonian.

### 3 Minimizing the Hamiltonian

We first randomly initialized the orientation of each dipole in a lattice consisting of  $20^3$  dipole vectors. Our initial implementation of a Monte Carlo method to minimize the Hamiltonian proved to be computationally expensive. Instead, we employed an iterative approach described in the next section.

#### 3.1 An Iterative Algorithm for Minimizing the Hamiltonian

After the dipoles were randomly initialized, an iterative approach is applied to find the orientation of the system that yields the minimum Hamiltonian. The technique we developed for updating the lattice involves a simple manipulation of Eq.(1). The Hamiltonian calculation involves a summation of its nearest neighbors added to the magnetic field. We take this to be one vector  $\vec{K}_j$  defined for the  $j$ th dipole to be

$$\vec{K}_j = \left( \sum_i \vec{m}_i + \vec{B} \right) \quad (7)$$

To update the lattice we add  $\eta(\vec{K}_j - \vec{a}_j)$  to  $\vec{a}_j$  for each dipole,  $\eta$  is a small user-defined constant that is on the order of  $10^{-2}$ . This constant relates to the type of material, the value of the time step, and the rotational inertia. It is defined by the user to attain the desired behavior of the system.

In figure 1, we show the results of our iteration method on the alignment of the dipoles for various magnetic field strengths. We apply the magnetic field in the direction of the x-axis. The alignment of the dipoles is represented by the magnetization projected

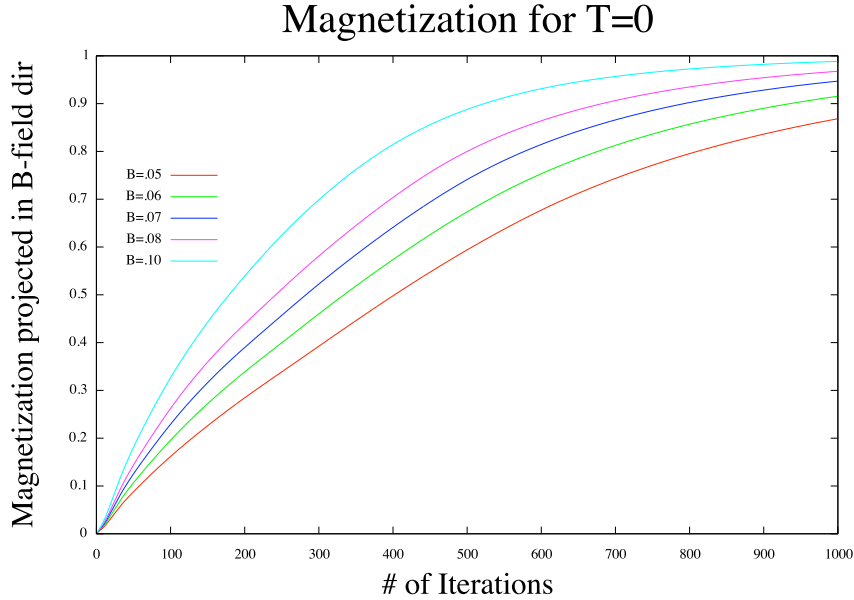


Figure 1: The magnetization approaches 1.0 for a system with no temperature and a magnetic field. If the magnetic field is stronger the graph approaches it more quickly. This figure demonstrates exactly how we expect the energy to minimize.

in the direction of the external magnetic field. Perfect alignment with the magnetic field would yield a magnetization of 1.0. As expected applying a stronger magnetic field causes the dipoles to align more rapidly than a weaker magnetic field.

### 3.2 Incorporation of Temperature

In order to incorporate temperature into our system, we evaluated  $Eq.(5)$  and assigned  $\frac{1}{3}\left(\frac{I}{k_b\Delta t^2}\right) = 1$ . This makes the relationship between temperature and the average change in the angle very manageable. However, we lose the physical interpretation of the temperature until further unit analysis is performed. The next step is to allow our system to have random rotations of the dipoles because we want our system to have a *freedom* in movement that non-zero temperature implies. We use the Central Limit Theorem (which was first explored by De Moivre[1]) to achieve a Gaussian-like distribution in which we could control the average angle and the variance independently. Figure 2 shows some plots of energy minimization at different temperatures. This figure demonstrates how temperature affects the minimization for a given magnetic field.

## 4 Visualization

In the next section, we describe visualization methods developed to better understand the orientation and alignment of the dipole vectors and the three dimensional convergence patterns of the Hamiltonian minimization.

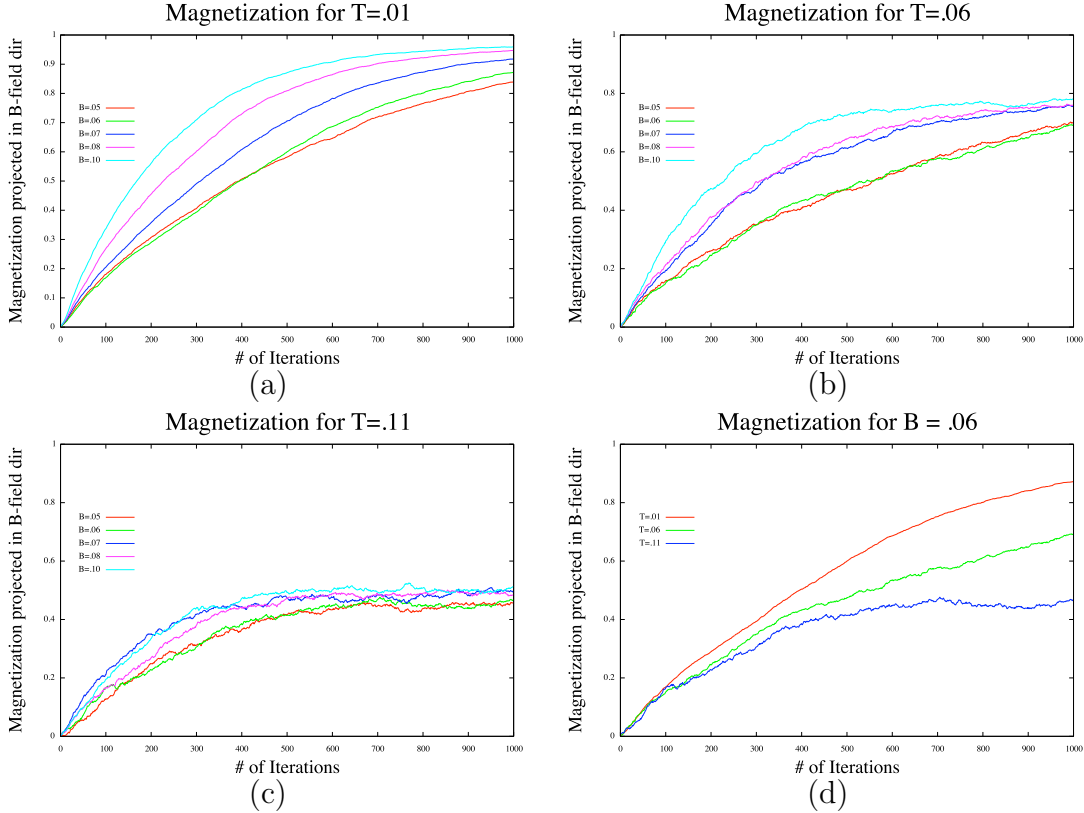


Figure 2: (a)-(c):These plots show the magnetization for three temperatures. These temperatures were experimentally determined to be *interesting*, because they showed how much temperature could affect the system. (d): This plot shows magnetization for a constant magnetic field with three different temperatures.

## 4.1 Dipoles

An initial visualization method for representing the dipole is to simply construct a line segment or glyph in the direction of each vector. This technique, known as vector plots or hedgehogs, is traditionally an effective method for representing vector fields. However, we are predominantly interested in the alignment of the dipoles and this simple technique is not entirely sufficient. Figure 3(a) illustrates how the orientation of the three dimensional vector glyphs within a lattice becomes difficult to interpret without additional information, as the orientation of the 3D glyphs become occluded by other glyphs.

In order to make the orientation of the vectors visually salient, we color the dipoles according to their orientation. We assign each of the coordinate axis a separate and orthogonal color in RGB color space (X⇒red, Y⇒blue, Z⇒green). Since each components range is -1 to 1, we developed a mapping function as follows:

$$f(w) = \frac{w + 1}{2} \quad (8)$$

where  $w$  is the X, Y, or Z component and  $f(w)$  corresponds to R, G, and B, re-

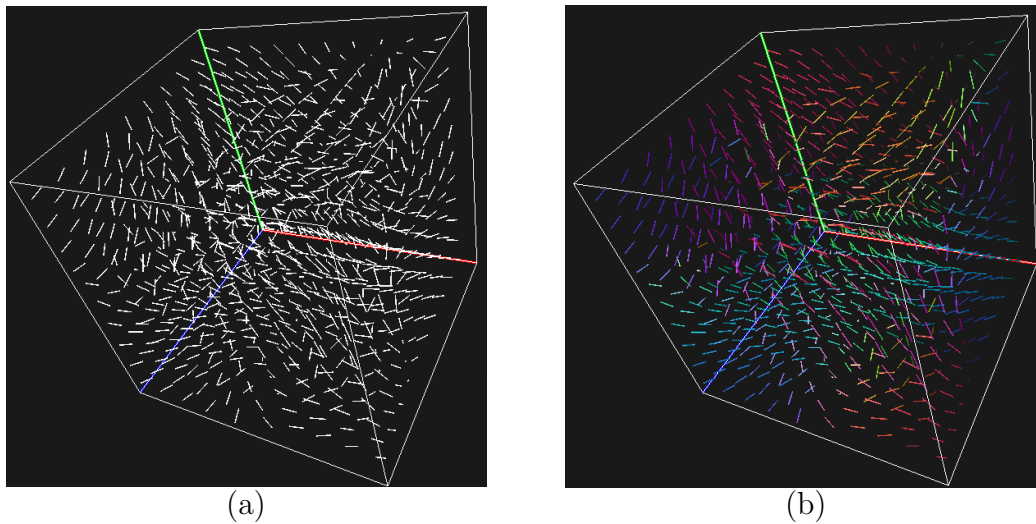


Figure 3: A visualization of the dipoles that have been allowed to partially align. (a): Shows no coloring (b): Shows a coloring of the dipole that indicates the direction of the dipole

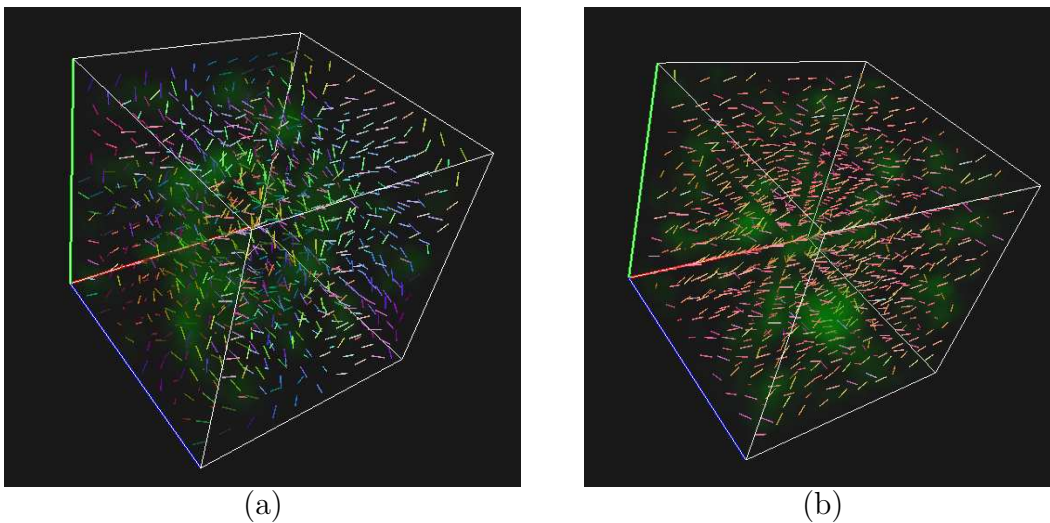


Figure 4: (a): This figure shows the lattice minimizing. We see there are definite regions of minimization. The volume-rendered regions of green indicate that the Hamiltonian has not completely minimized. (b): This figure is similar to (a) except that it has an applied external magnetic field. Here we can see that the Hamiltonian is minimizing in the direction of the magnetic field and volume render indicates regions where the system is *struggling* to minimize.

spectively. The result is that the vectors that are aligned are colored a similar color (Figure 3.b).

## 4.2 Volume Rendering the Hamiltonian

Volume rendering is a classic visualization technique that uses color and opacity to represent a 3D scalar field. We use volume rendering in order to represent the Hamiltonian of the system. This allows the user to directly identify regions where the Hamiltonian is minimizing and how the minimization occurs. Figure 4 shows two different volume renderings. It is clear that there exist regions where the Hamiltonian is not minimizing as quickly as other regions. It is also clear from these visualizations that the lattice does not uniformly minimize and that there are identifiable regions where initial dipole alignment greatly affects the convergence rate of the minimization of the Hamiltonian.

Figure 5 shows a plot of the magnetization similar to the plots in figure 2. Additionally, it shows the volume renderings that correspond to several time points throughout the minimization. Figure 5(a) show how the system is completely unaligned at the initial time step. It is clear from the volume rendering that there exists absolutely no minimization in the Hamiltonian. Examining figures 5(b) and (c), it is clear that as more iterations occur, the isolated regions of anti-alignment *shrink* in physical size. Eventually, the system becomes almost completely aligned with the magnetic field, which is clear from figure 5(d).

Figure 6 show that addition of temperature does not affect the existence of these anti-alignment regions. However it can be seen by comparison of figure 5(b) and figure 6(b) that a system with temperature *aids* the existence of these regions allowing them to be maintained longer. Additionally in figure 6(d), it is clear that there is a bit of variation in the dipole directions in neighboring dipoles. This variation is due to random rotations that are simulating the temperature.

## 4.3 User Interface

In order to allow the program be more useful and accessible a user interface was developed. We implement the Graphics Language User Interface (GLUI) extension for OpenGL. This is a open source extension that allowed for buttons, control of variables, movement of and through visualization, and display of current data. In Figure 7, the relevant data and controls are displayed for the user to manipulate, which allows the software to be more controllable.

# 5 Conclusions and Future Outlook

By visualizing magnetic dipoles, we are gaining insight into ferromagnetism and energy minimization. Our goal is to give scientists a better tool to look at ferromag-

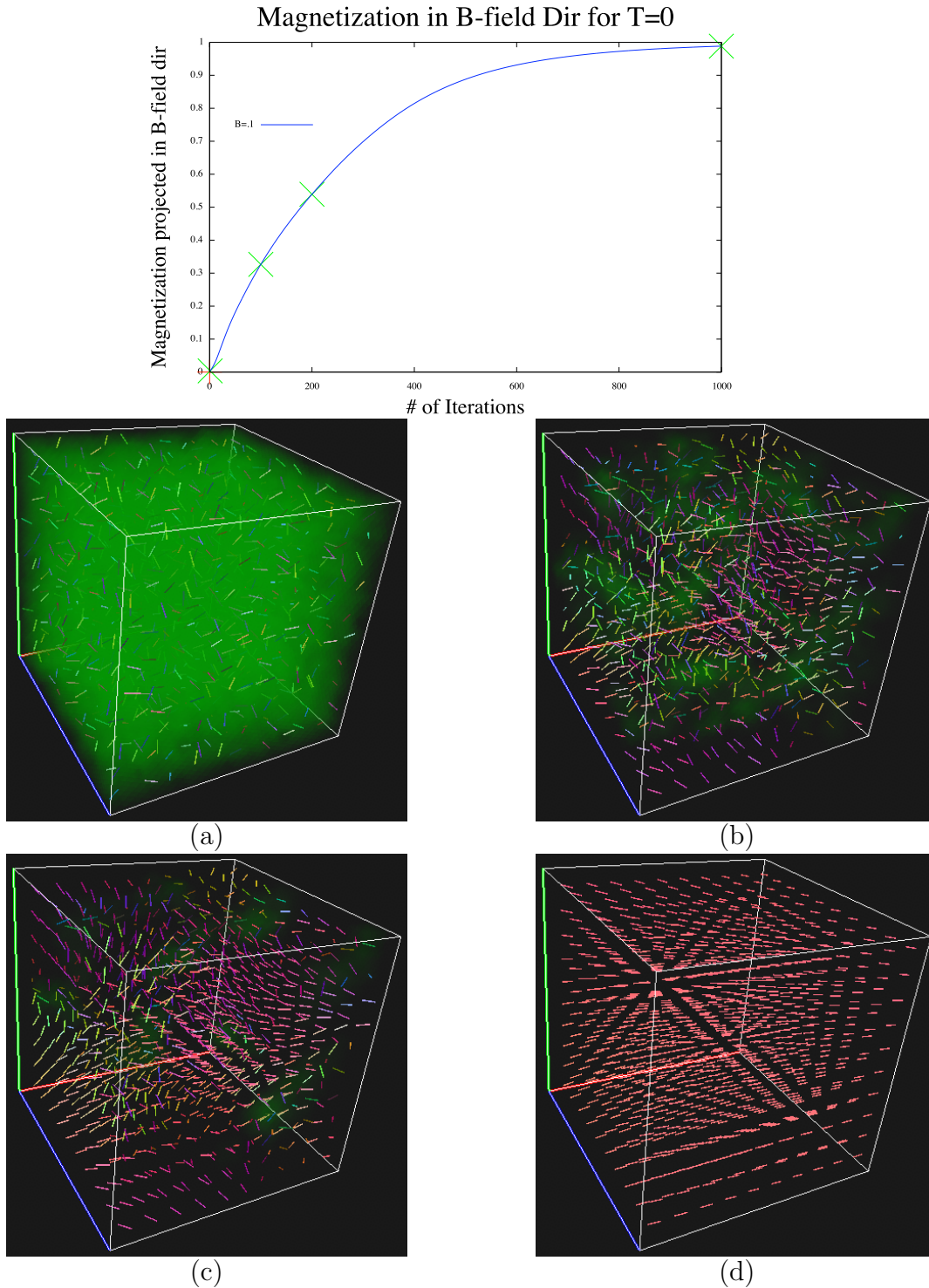


Figure 5: (top) A plot of magnetization projected in the direction of the magnetic field where temperature is not considered in the calculation. This plot and additional figures demonstrate the connection between minimization and volume rendering. The green X's on the plot of the magnetization indicate where we are taking snapshots of the volume rendering corresponding to (a)-(d), respectively. (b) and (c) clearly demonstrate that there exist regions where the initial dipole orientation lead to a slower alignment.

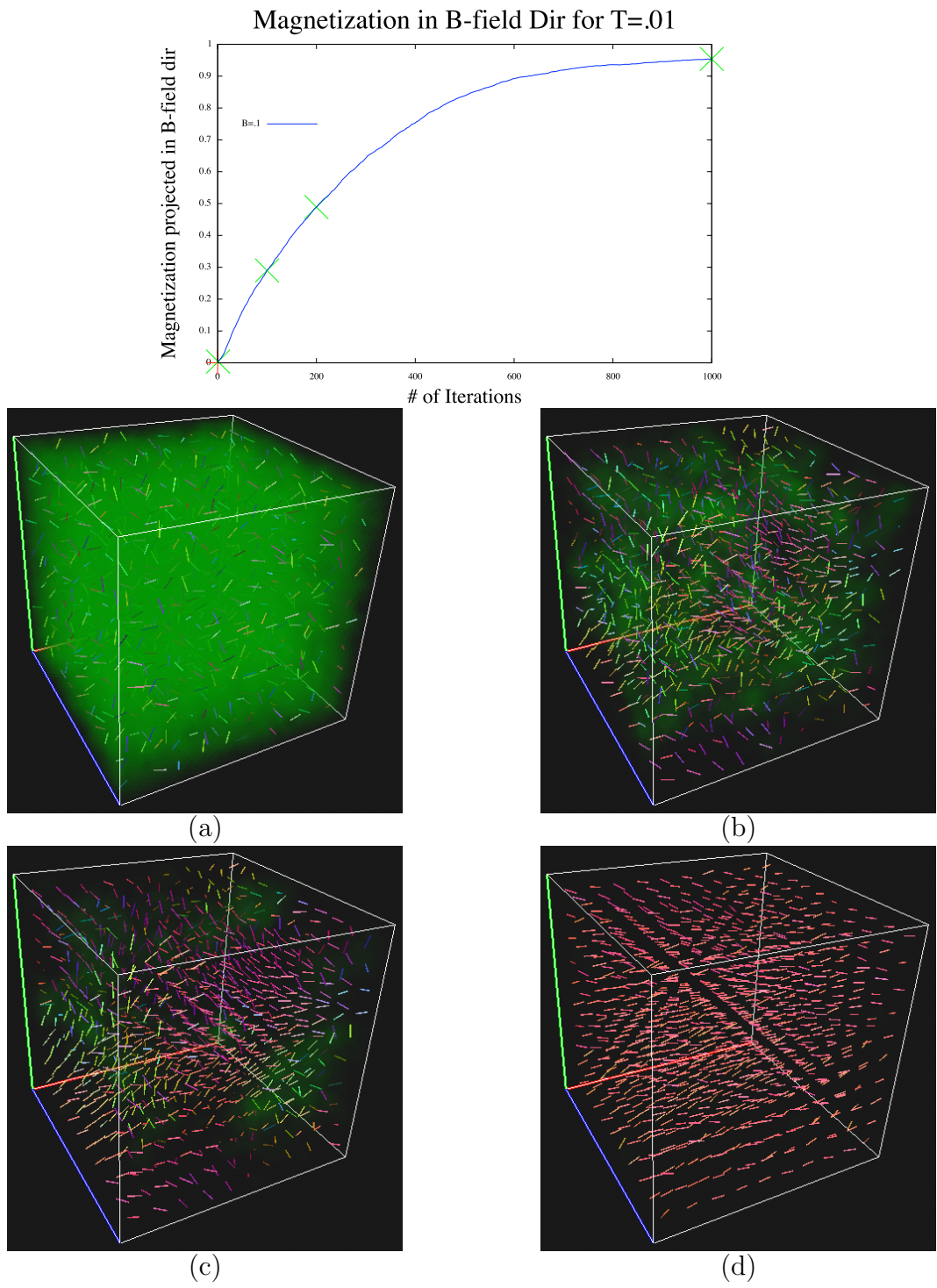


Figure 6: (top) A plot of magnetization projected in the direction of the magnetic field where temperature is considered in the calculation. This plot and additional figures demonstrate the connection between Hamiltonian minimization and volume rendering when temperature is considered. The green X's on the plot of the magnetization indicate where we are taking snapshots of the volume rendering corresponding to (a)-(d), respectively. Note: In comparison to Figure 5, these plots align more slowly because temperature is considered.



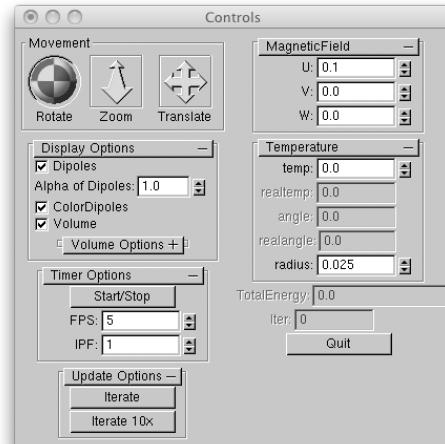


Figure 7: A user interface for our software that allow easy access to information and controls to operate the software.

netism. We hope that additionally we will be able to use our visualization to give individuals who have not previously studied ferromagnetism a better understanding of what underlying processes exist. Our visualization can demonstrate the Curie temperature—the temperature at which a system can no longer be minimized—to someone who has never had a single course in college level physics.

Our scheme for drawing dipoles and volume rendering is a starting point. We hope that our research is only the first of many visual tools to allow a better understanding of energy minimization in ferromagnetic systems. If our techniques were readapted to utilize the graphics processing unit (GPU), we might be able to see better real-time visualizations of large lattices. Additionally one could look at processing the lattice on a super-computer or multi-core system. This would allow us to have faster updating of large lattices. Lastly, this research could also benefit from visualization on a virtual reality system, as it would allow the user to have a better perception of the lattice.

## 6 Acknowledgments

We would like to thank Dr. Athan Petridis of Drake University for his expert advice on some of the physics. Additionally, Zack Kertzman provided several helpful insights at key times in our research. We would like to thank Drake Undergraduate Science Collaborative Institute for funding this research during the summer of 2007. Finally, we would like to thank the Drake University Student Travel and Research Fund.

## References

- [1] W. J. Adams. *The life and times of the central limit theorem*. Kaedmon Pub. Co., New York, 1974.
- [2] D. P. Belanger and A. P. Young. Random field ising model. *Journal of Magnetism and Magnetic Materials*, 100(1-3):272–291, 1991.
- [3] B. Field, S. O’Neill, V. Interrante, T. W. Jones, and T. Urness. Fieldvis: A tool for visualizing astrophysical magnetohydrodynamic flow. *IEEE Computer Graphics and Applications*, 27(1):9–13, 2007.
- [4] D. Giffiths. *Introduction to Electrodynamics*. Prentice-Hall, Upper Saddle River, New Jersey, 1999.
- [5] R. Guida and J. Zinn-Justin. 3d ising model: The scaling equation of state. *Nuclear Physics B*, 489(3):626–652, 1997.
- [6] C. Kittel and H. Kroemer. *Thermal physics*. W. H. Freeman, San Francisco, 2d ed edition, 1980.
- [7] R. Victor Klassen and Steven J. Harrington. Shadowed hedgehogs: A technique for visualizing 2d slices of 3d vector fields. In *IEEE Visualization*, pages 148–162, 1991.
- [8] J. Kniss, G. Kindlmann, and C. Hansen. Multidimensional transfer functions for interactive volume rendering. *IEEE Transactions on Visualization and Computer Graphics*, 8(3):270–285, 2002.
- [9] J. Lee and J. M. Kosterlitz. New numerical method to study phase transitions. *Physical Review Letters*, 65(2):137–140, 1990.
- [10] U. Löw, V. J. Emery, K. Fabricius, and S. A. Kivelson. Study of an ising model with competing long- and short-range interactions. *Physical Review Letters*, 72(12):1918–1921, 1994.
- [11] E. Manousakis. The spin-1/2 heisenberg antiferromagnet on a square lattice and its application to the cuprous oxides. *Reviews of Modern Physics*, 63(1):1–62, 1991.
- [12] A. W. Sandvik. Finite-size scaling of the ground-state parameters of the two-dimensional heisenberg model. *Physical Review B - Condensed Matter and Materials Physics*, 56(18):11678–11690, 1997.
- [13] R. A. Serway, R. J. Beichner, and J. W. Jewett. *Physics for scientists and engineers*. Saunders golden sunburst series. Saunders College Publishing, Fort Worth, 5th ed. edition, 2000.

# Statistical Process Control of Software Processes for Obtaining CMMI Level 5

**Mike Rowe, Ph.D.**  
**Computer Science and Software Engineering Department**  
**University of Wisconsin – Platteville**  
**Platteville, Wisconsin 53818**  
[rowemi@uwplatt.edu](mailto:rowemi@uwplatt.edu)

**Elisabeth Farver**  
**AVISTA, Inc.**  
**Platteville, Wisconsin 53818**  
[Beth.Farver@avistainc.com](mailto:Beth.Farver@avistainc.com)

**Thomas Bragg**  
**AVISTA, Inc.**  
**Platteville, Wisconsin 53818**  
[Tom.Bragg@avistainc.com](mailto:Tom.Bragg@avistainc.com)

**Mark Kelley**  
**AVISTA, Inc.**  
**Platteville, Wisconsin 53818**  
[Mark.Kelley@avistainc.com](mailto:Mark.Kelley@avistainc.com)

**Craig Hale**  
**AVISTA, Inc.**  
**Platteville, Wisconsin 53818**  
[Craig.Hale@avistainc.com](mailto:Craig.Hale@avistainc.com)

## Abstract

This paper discusses the implementation of Statistical Process Control (SPC) for software processes. SPC involves the tracking of key process parameters on a real-time (daily or weekly) basis to determine how projects are performing against historical baselines. SPC is critical for obtaining the Software Engineering Institute's (SEI) Capability Maturity Model Integrated (CMMI) Level 5. This paper contains an overview of CMMI, an overview of SPC, a discussion of the actual implementations of SPC and its application to software process monitoring, and a discussion of some of the benefits that were encountered implementing SPC while obtaining CMMI Level 5.

# 1 Introduction

In July 2007, AVISTA, Inc. became the 22<sup>nd</sup> U.S. and one of only a few Midwest software companies to attain Software Engineering Institute's (SEI) Capability and Maturity Model Integrated (CMMI) Level 5. Over the past 20 years, AVISTA has provided over 2.5 million hours of safety- and mission-critical software engineering services to the world's leading aerospace and medical companies.

Capability Maturity Model (CMM) and more recently CMMI has been promoted by government agencies in their effort to understand what makes software system providers successful. CMMI is broken into five maturity levels. The lowest, Level 1 – Initial, is characterized by no formal process, and the highest, Level 5 – Optimized, is characterized by an organization that is continually optimizing its process. Today CMMI Level 3 or higher is required to perform work on many major government contracts and is of increasing importance in the competition for non-government contracts.

Level 4 – Quantitatively Managed, requires establishment and maintenance of quantitative understanding of the organization's process performance. Quantitative understanding of process is necessary to determine if process optimization is actually occurring for Level 5. Statistical Process Control (SPC) is a mechanism for monitoring process performance.

## 2 SEI CMMI Overview

For an excellent resource on CMMI see *CMMI 2<sup>nd</sup> Edition, Guidelines for Process Integration and Improvement*[1]. Watts Humphrey[1] studied successful and unsuccessful software projects and determined that there were 25 major processes related to the level of success of software projects. This was the bases of the original CMM for Software that was released in 1993. In 1995 a CMM for Systems Engineering was released. In 2000, version 1.02 of CMMI Software and Systems Engineering standards were integrated into one standard. With CMMI, the 25 processes were reorganized into 22. Most recently, CMMI has been split into three standards CMMI for Development (version 1.2, 2006) Acquisition (version 1.2, 2007) and Services (version 1.2, 2007).

The 22 processes are subdivided into four process *categories*, including: Project Management, Process Management, Engineering and Support. An organization's maturity is evaluated and categorized into one of five levels:

Level 1: Initial, there is no formal process, and success can be attributed to the heroics of a few engineers;

Level 2: Managed, there is a minimal process and the status of projects is visible to management at major milestones. Process varies from project to project.

Level 3: Defined, there are organizational-wide standards, procedures, tools and methods.

Level 4: Quantitatively Managed, there are quantitative objectives for quality and process performance set for projects and the organization based on the needs of customers, end users, the organization, and process users. These objectives are statistically managed.

Level 5: Optimized, there is a constant effort to optimize the process by investigating the causes of inefficiency and defects. Statistical tools, including SPC, are used to spot possible inefficiencies and determine whether action plans are efficacious.

<b>Maturity Level</b>	<b>Process Management</b>	<b>Project Management</b>	<b>Engineering</b>	<b>Support</b>
1: Initial				
2: Managed		Project Planning (PP)	Requirements Management (REQM)	Configuration Management (CM)
		Project Monitoring and Control (PMC)		Process and Product Quality Assurance (PPQA)
		Supplier Agreement Management (SAM)		Measurement and Analysis (MA)
3: Defined	Organization Process Focus (OPF)	Supplier Agreement Management (SAM)	Requirements Development (RD)	Decision Analysis and Resolution (DAR)
	Organizational Process Focus (OPF)+ Integrated Product and Process Development (IPPD)	Integrated Project Management (IPM) + Integrated Product and Process Development (IPPD)	Technical Solutions (TS)	
			Product Integration (PI)	
	Organization Training (OT)	Risk Management (RSKM)	Verification (VER)	
		Validation (VAL)		
4: Quantitatively Managed	Organization Process Performance (OPP)	Quantitative Project Management (QPM)		
5: Optimizing	Organization Innovation and Deployment (OID)			Causal Analysis and Resolution (CAR)

**Table 1: CMMI processes organized by level and category.**

There is also a CMMI *Capability* level rating for organizations. Whereas *maturity* level of an organization is based on the lowest rating of any of the four process categories, an organization receives an individual *capability* level rating for each of the processes. Level 0 corresponds to none of the specific goals of a process being satisfied. Capability Level 1 corresponds to satisfaction of specific process goals, but none of these processes are institutionalized, and thus, without assurance that they will continue to be used. Levels 3 through 5 use the same names as CMMI Maturity Levels and represent similar improvements in process and process performance.

Of particular interest to this paper is the Quantitative Project Management (QPM) process that in part makes up Maturity Level 4. Quantitative Project Management (QPM) has the following specific goals (SG) and specific practices (SP) [1]:

SG1: Quantitatively Manage the Project

SP1.1 Establish the Project's Objectives

SP1.2 Compose the Defined Process

SP1.3 Select the Subprocesses that will be Statistically Managed

SP1.4 Manage Project Performance

SG2: Statistically Manage Subprogram Performance

SP2.1 Select Measures and Analysis Techniques

SP2.2 Apply Statistical Methods to Understand Variances

SP2.3 Monitor Performance of the Selected Subprocesses

SP2.4 Record Statistical Managed Data.

The next section of this paper will discuss how Statistical Process Control can be applied to help achieve the QPM goals.

### **3 Statistical Process Control (SPC) Overview**

SPC is a technique that facilitates the monitoring of process performance using *control charts*. Control charts plot key process parameters against historical measures of central tendency and variability. *Run rules* help identify non-random variations in processes control charts. When non-random variation occurs, a process is considered to be “*out of control*”. An out of control process triggers intervention to determine what if anything has caused the problem using *root cause analysis*. If a root cause can be found, the software process can be modified to hopefully prevent or minimize this root cause from happening in the future. This is the basis for process optimization, which is pivotal for obtaining to CMMI Level 5.

#### **3.1 Control Charts**

Control charts provide a real-time graphical presentation of how a process is performing in relationship to a target baseline. A control chart makes use of a *Target* value, *Control Limits*, in some cases *Specification limits*, *Zones*, and *Run Rules*. There are many

different types of control charts for many different data types and applications. The two major types of control charts are those for quantitative data and those for qualitative data. Examples of quantitative data control charts include the X-bar-R chart, which plots subgroup means and subgroup ranges; X-Bar-s chart, which plots subgroup means and subgroup standard deviations; XmR chart, which plots individual data values and a moving range. Examples of qualitative data control charts include p-chart, which plots percent of defective units or part; np-chart, which plots number of defective units; c-charts and u-charts plots number of defects (when a item can have more than one defect per unit).

### **3.1.1 Target Value**

The Target Value, also called centerline, is a central tendency value around which a process parameter is expected to perform. To help make this concept more tangible, this section will use the example of number of defects injected per KLOC. Often times this target value is based on historical performance of the process, for instance a company may have a historical average of 0.32 defects per KLOC. Unless this project is different from previous projects, one would expect that from week to week that the defects per KLOC would not be significantly different from this target of 0.32 defects per KLOC. One would always like to do better than this though. Note that all actual data are considered AVISTA proprietary, and for all examples, contrived data are used.

### **3.1.2 Control Limits**

Control Limits, upper control limit (UCL) and lower control limit (LCL), define the level within which one would expect a process to function. They are normally defined as 3.0 standard deviation above (for the UCL) and 3.0 standard deviations below the centerline for the LCL. The range of +/- 3.0 standard deviations from the centerline represents more than 99.7 % of normally distributed data. If data for a process falls outside of control limits, the project lead can suspect that something bad or in some cases good has happened. When a process goes outside of the control limits it is described as “*out of control*”. The detection of problems will be discussed below in the section on run rules. The standard deviation used for the control limits is commonly derived from historical process performance data, although sometimes it is set by less empirical methods. Note that some parameters are limited by logical boundaries, for instance, we cannot have a negative number of defects per KLOC, in fact, one might become suspicious of a testing process if very low defect numbers are consistently recorded.

### **3.1.3 Specification Limits**

Specification Limits, upper specification limit (USL) and lower specification limit (LSL), define the acceptable range of process output. Specification limits are commonly used to control manufacturing processes. For instance, there may be tolerances with respect to the diameter of a drilled hole. If it is too large or small it is unacceptable. Specification limits are not commonly used to monitor software processes.

### 3.1.4 Zones

Zones breakdown the region between the centerline or baseline mean and the control limits into 1.0 standard deviation bands.

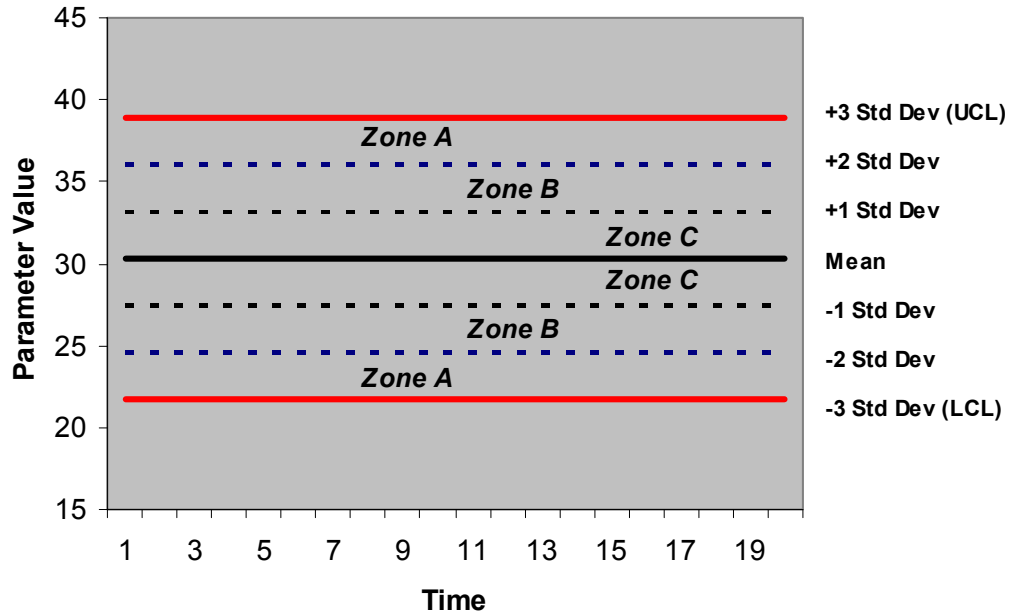


Figure 1: Control chart zones [2]

### 3.1.5 Run Rules

Run Rules were originally attributed to Western Electric [3] and make use of the zones to spot statistical outliers. These are simple statistical rules of thumb that can be computed by counting on one's fingers. Simplicity was essential because at the time when SPC was introduced, the 1940's and 1950's, even simple mechanical adding machines were unavailable and unsuitable for most manufacturing environments. Below are examples of some of the popular run rules.



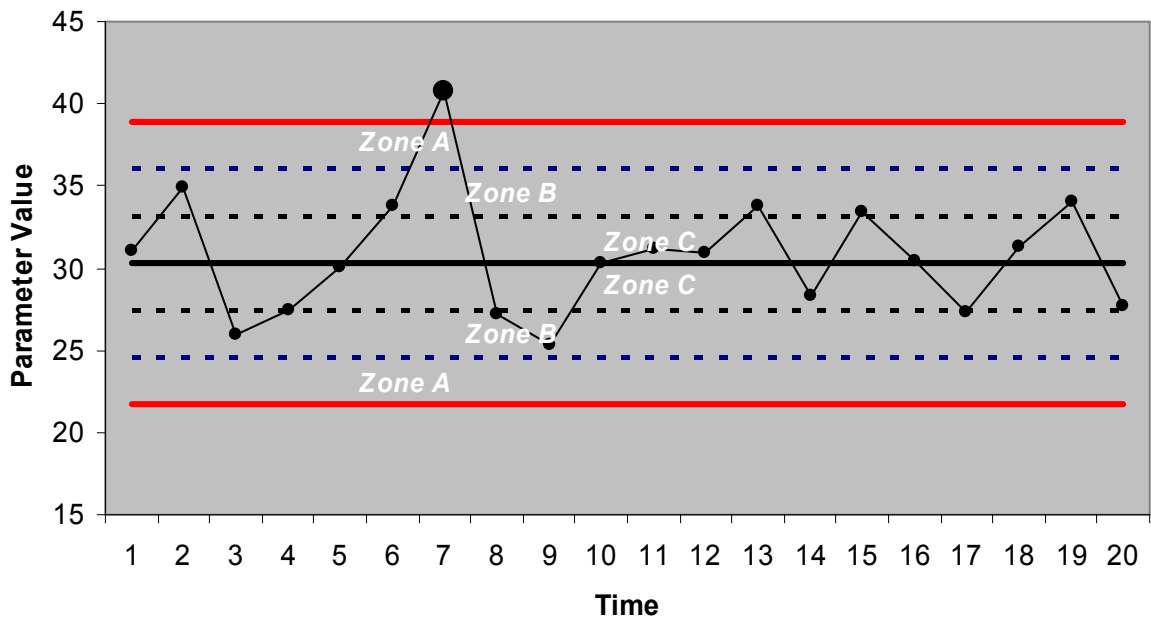


Figure 2: Run rule, One point above the UCL or below the LCL [2]

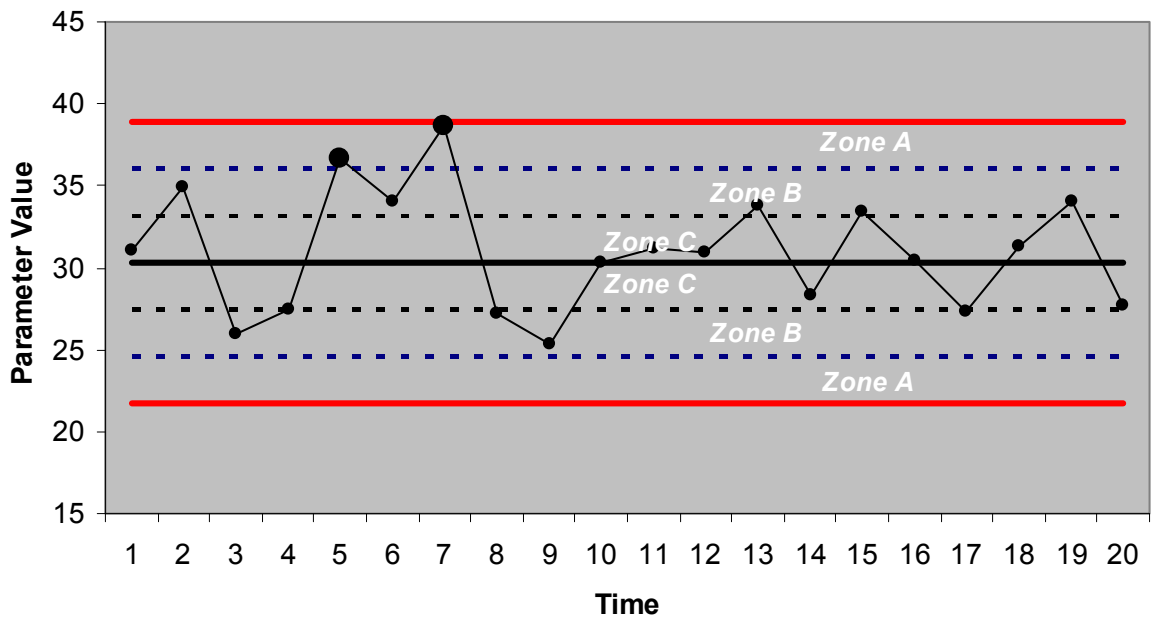
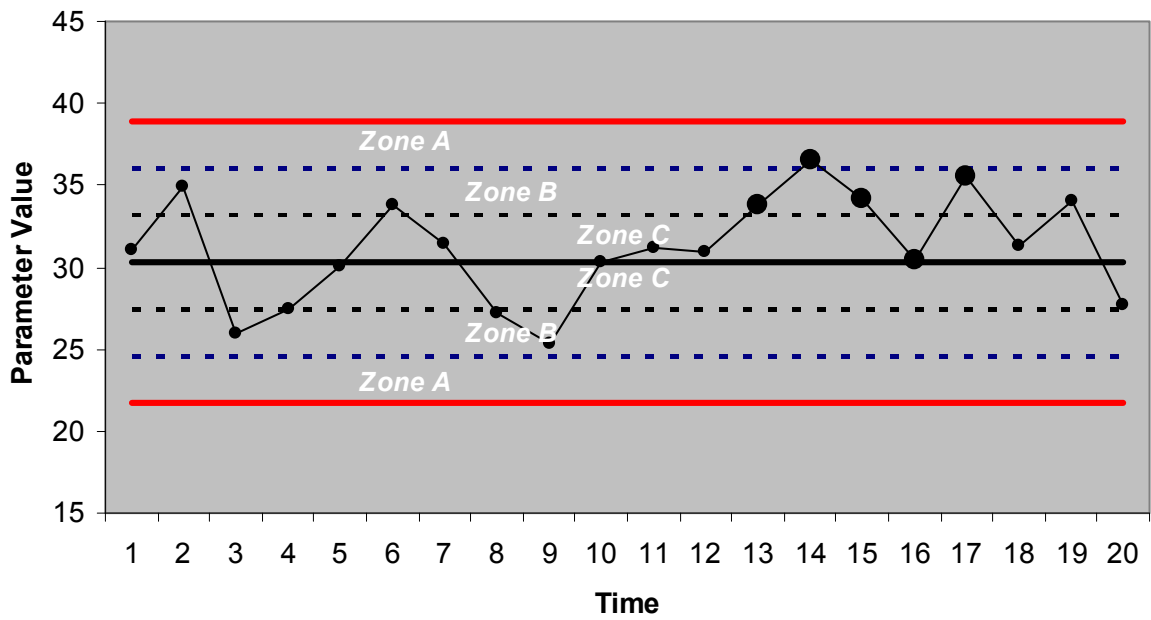
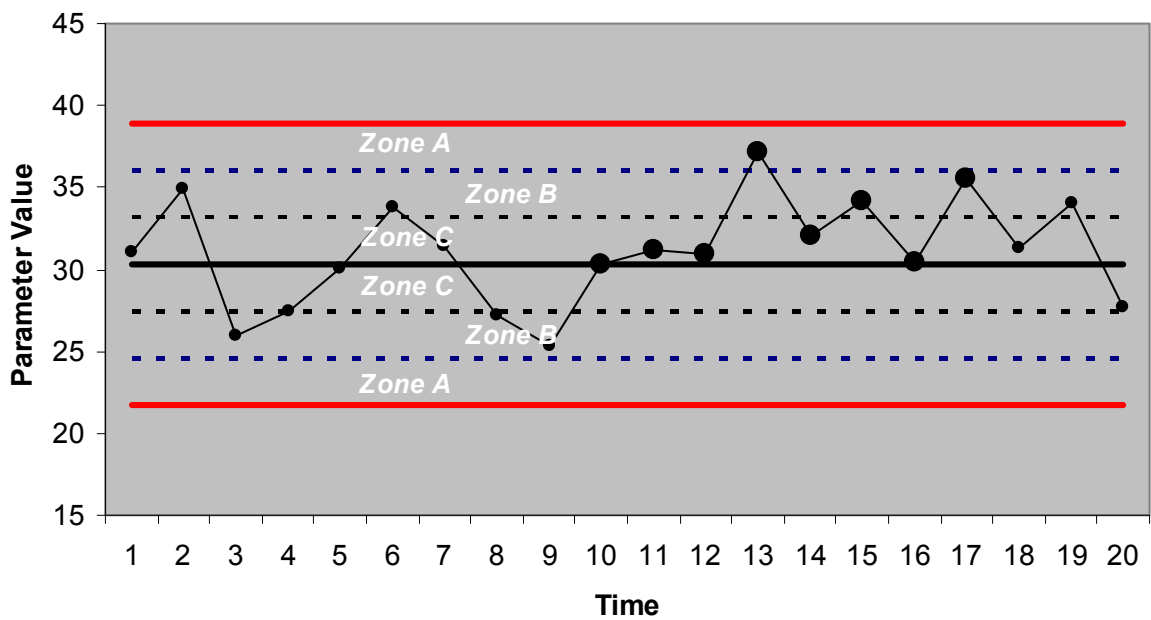


Figure 3: Run rule, 2 of 3 consecutive points in Zone A and all 3 points on the same side of the center line [2].



**Figure 4: Run rule, 4 of 5 consecutive points within Zone A or B and all 5 points on the same side of the centerline [2].**



**Figure 5: Run rule, 8 consecutive points fall on the same side of the mean [2].**

## 3.2 More on control charts

Quantitative control charts are generally used in pairs. One of the pair plots a measure of central tendency while the other chart plots a measure of variability. The reason for analyzing the pair of charts pertains to the nature of subgrouped data. The subgroup mean may not tell the whole story, for instance across the week the mean of all projects may hit the target perfectly. A closer look at the individual data for each project may show that not a single project's data fell within acceptable level – basically the projects offset each other. To illustrate this let's examine an X-bar-s (*X-bar* is the mean and *s* is the standard deviation) control chart example. The X-bar chart is sensitive to shifts in the mean and may have a CL=10.0, UCL=12.0 and a LCL=8.0. The raw data for the week might be 5, 4, 3, 15, 16, and 17; this yields an X-bar of exactly 10.0, which is right on the CL, thus no run rule would trigger. The corresponding s-chart is sensitive to variability and may have a CL=2.5, UCL=4.5 and a LCL=0.5. The raw data yields an s=6.62, which is significantly beyond the UCL of the s-chart. A run rule would trigger for the s-chart detecting this outlying standard deviation.

## 4 SPC at AVISTA

### 4.1.1 Background

AVISTA works in highly regulated mission critical and life critical domains and has been ISO 9001 certified for a several years. Because of the nature of this business, AVISTA already had several internal systems to track lifecycle activities, defects and labor efforts to satisfy regulatory needs. There are two primary tracking systems used for SPC data. The OMNI system tracks job level labor for numerous activities, earned values, lifecycle, etc. and reports Cost Performance Indices (CPI) and Schedule Performance Indices (SPI) to project and program management. Another tool, Data Item Review Tool (DIR Tool), is used to track lifecycle, requirements, issues, defect (severity, type, insertion points, detection points, etc.), resolutions, and signoffs. Combined, these two systems provide a tremendous variety of project information to consider for review.

Part of CMMI Level 3 is the institutionalization of processes across an organization. Over the years, AVISTA has developed standard processes for each stage in its development lifecycle and for different levels of product verification. Generally, these standard practices can be used with minimal tailoring for individual projects. This is of tremendous value to the individuals executing the process, as most team members are well practiced at performing all tasks necessary to complete a process. This also makes possible the analysis of individual projects against historical organizational baselines for each of the processes. Basically, one can compare a project to the historical baseline created from other projects that have used the same or similar process by means of control charts. The centerline, and control limits are set to the historically derived baselines. Any significant variance from these baselines, as determined by the run rules, triggers an alarm to study why the variance occurred. If the variance was caused by

something that can be fixed, the process is modified to help prevent this variance in the future.

#### 4.1.2 Initial Planning

Part of the AVISTA CMMI team spent around a year studying its software activities to characterize which process parameters were available and which parameters had the highest potential to help the business. It was discovered that although there were a lot of data being collected, additional parameters were still needed. The need for additional data entities required a few enhancements to AVISTA's internal tracking systems. On the recommendation of our lead CMMI appraiser, XmR control charts were selected to monitor AVISTA's processes. XmR control charts plot individual data and a moving average. Over 50 different control charts were initially planned.

#### 4.1.3 Implementation of SPC

The first step in actually producing a control chart for a baseline was to write requirements to query the database. Many of these queries are quite complex, involving several relational tables. Because of this complexity and risk associated with using the wrong data, very complete verification procedures were also produced. The verification procedures included reviewing the actual SQL and the data that resulted from the queries.

A query produced several data tables, one for the final data to be directly used in the control chart and additional intermediate tables that helped in the verification process. The query results were exported to comma separated value (CSV) files to facilitate easy import into Microsoft Excel™.

An Excel™ workbook was created for each of the planned control charts. These workbooks contained specific sheets for the query requirements, the verification process and results, notes for action items, intermediate data tables, excluded data and documentation as to why a data point should be excluded, the final data used in the control chart, and the control chart. Once the data were imported into the spreadsheets, the data were verified. In the early iterations of the baselines, the verifications identified several query defects. Once the data were verified, it was plotted in the control charts.

With the control charts populated, *outliers* are studied. An outlier is a data point that triggered run rule. Normal variation is expected and can result in some false-alarms. The study of outliers consists of interviewing the project leads to determine whether something out of the ordinary was occurring. Points that have some known and possibly preventable causes are classified as having "special cause variations" and are excluded from use in future baseline calculations. Project leads are required to produce *Correction Action Requests* (CARs) that document the reason for an outlier, including its root cause analysis and suggest actions that could help prevent this type of event in the future. For instance, some of the reasons for an outlier might include:

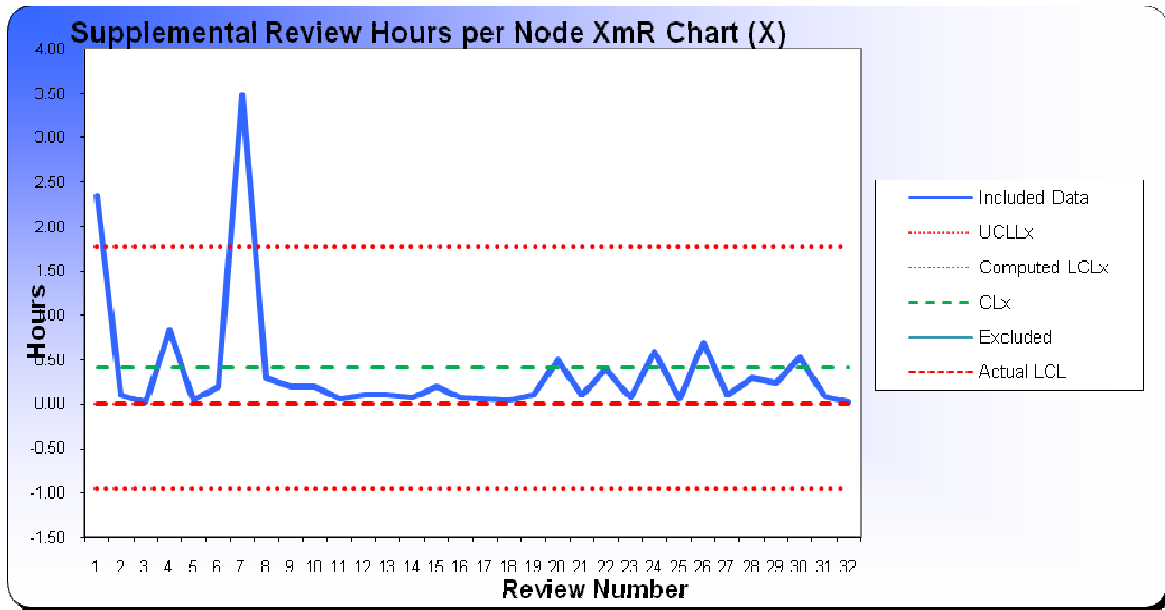
- A single requirement was very complex, being based on a six-page logic diagram rather than a simple “shall” statement. With very complex requirements, one would expect to encounter higher development time, longer review time, more defects during reviews and testing, higher rework time, and so on then with simple “shall” statement requirements.
- A new member was added to a team and their productivity was below that of experienced team engineers. There might also be misclassification of data items by inexperienced team members.
- A customer dictated a different process than one of the AVISTA standard processes. Because the process is different, we would expect somewhat different results than projects that follow one AVISTA’s standard processes.
- A customer may have substantially revised project artifacts, like requirements in the middle of a project.
- A project went into a temporary hold due to other companies involved with a project. Many of the projects that AVISTA works on are very large, like the Airbus 380 and the Boeing 787. Redesigns and their associated delays can sometimes happen. These holds are associated with project ramp-down and ramp-up inefficiencies.
- Delays in claiming earned value for work completed due to test environment issues can cause outliers. Much of the software AVISTA develops is targeted at real-time embedded systems. The work is not complete until it can be tested on the target hardware – sometimes the hardware development is behind that of the software development.

Outliers are documented on the exclude worksheet with a short description of why a data point should be excluded or not excluded. This documentation stays with the baseline so when new data is appended, we do not need to revisit this exclude or not exclude decision process for the older data points.

Below in Figure 6 and 7 are the XmR control charts for Supplemental Test Review hours per node. Supplemental Testing is a form of structural or coverage testing. A form of structural testing, Modified Condition Decision Coverage (MC/DC) testing, is required for AVISTA’s most critical avionics subsystems to certify at RTCA DO-178B Level A [4]. A failure of a Level A system results in catastrophic failure conditions for an aircraft, usually meaning it will crash. Figure 6 charts the actual test reviewing hours per test node and Figure 7 charts the moving range, the difference between the current and the previous values for review hours per node.

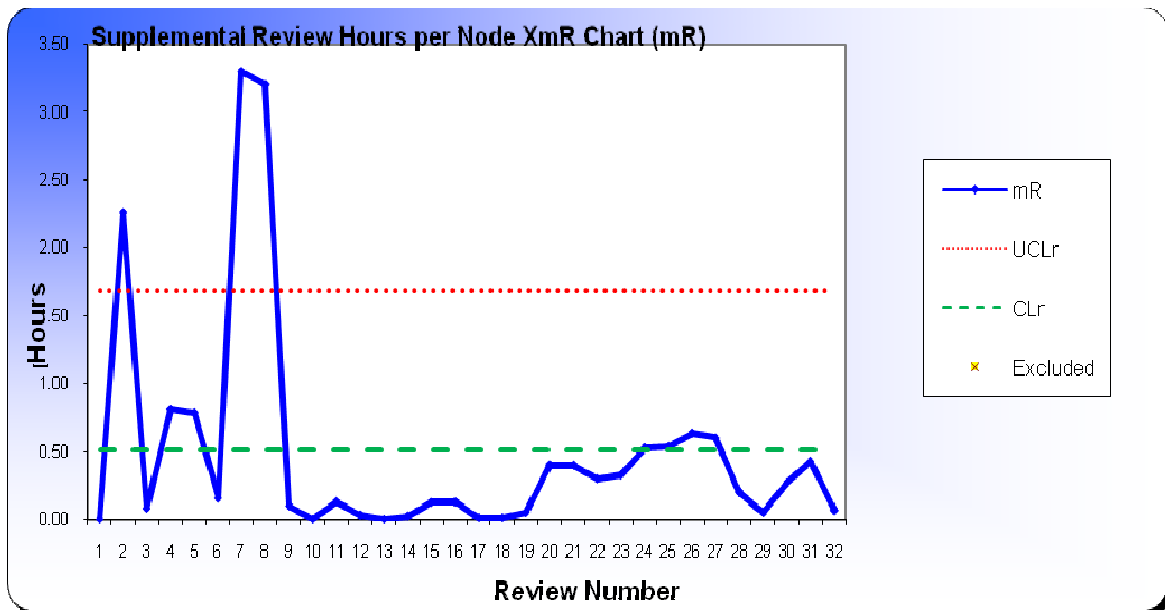
In Figure 6, there are two Lower Control Limits (LCLx red dotted and Actual LCL red dashed). The calculated, -3.0 standard deviations from the CL, LCLx is below zero. In the case of this measurement, hours per node, one would never expect to observe values

less than or equal to 0.0. Thus, we used an Actual LCL set at 0.0 so that any non-positive values will trigger investigation alarms.



**Figure 6: X chart of Supplemental Test Review Hours per Node**

The moving range (mR) chart in Figure 7 indicates a great deal of variability early in this project. This is normal for many projects during their ramp up. After Review Number 10, the variability drops down and remains low for the rest of the project. If the variability were to remain high, it would trigger an investigation.



**Figure 7: Moving Range (mR) chart of Supplemental Test Review Hours per Node**

To date around 10 control charts and baselines are already being used, and there are a couple more baselines that are in the final stages of development. Several of the 50 originally planned charts are associated with processes that have not generated sufficient data yet to produce statistically stable baselines. As more data become available, these baseline control charts will be produced. Examples of some of the control charts that AVISTA actually uses include:

- Number of AVISTA Developed Software Requirements-Based Test Results Defects Found per Total Software Requirements Covered in AVISTA Developed Software Requirements-Based Test Results Reviewed
- Software Requirements-Based Test Development Hours per Earned Value Equivalent Software Requirements Tested for average complexity requirements
- Software Requirements-Based Test Development Hours per Earned Value Equivalent Software Requirements Tested for low complexity requirements
- Number of Total Developed Software Requirements-Based Test Defects Found per Total Software Requirements Covered in Software Requirements-Based Tests Reviewed
- Software Requirements-Based Test Review Hours per Total Software Requirements Covered in Software Requirements-Based Test Review for average complexity requirements
- Software Requirements-Based Test Review Hours per Total Software Requirements Covered in Software Requirements-Based Test Review for low complexity requirements
- Total Supplemental Test Review Defects per Total Nodes Covered in Supplemental Test Review
- AVISTA Developed Code Review Defects Found per AVISTA Developed KLOC
- Total Developed Code Review Defects Found per Total Developed KLOC Reviewed

## **5 Future Plans**

AVISTA is currently working on producing some of the other higher business priority planned baselines.

Minor changes in processes, tools and the projects themselves over time can affect the baselines. For this reason, the baselines are reevaluated from one to four times per year to determine if the control limits or centerlines need to be change. The period for this analysis is based on the data volume for a particular baseline – the less data volume the less often the baseline is evaluated.

## 6 Conclusions

SPC has had several positive benefits for the work AVISTA performs.

Many process improvements have occurred based on *Corrective Action Requests* (CARs) resulting from the analysis of outliers. The process improvements are expected to help prevent these events from happening in the future by removing or mitigating the root causes of the problems. Many of these problems may not have been identified without SPC baselines to pinpoint outliers.

The baselines have allowed the company to understand very precisely its performance levels. AVISTA understands how long tasks take, defect rates, how much rework can be expected, the variability in the complexity of requirements, and more. This is a significant factor for bidding contracts as well as keeping projects on time and on budget.

As has been described above, deploying SPC into a software company is not an easy overnight process. It is dependent on several preconditions:

- Institutionalized software processes, so that results from historical projects can be generalized to future projects
- Strong data collection systems that do not burden engineers with record keeping. In the AVISTA's case, these systems actually make an engineer's job easier. AVISTA had already invested in these systems, so this was not as big an effort as it would be for organizations starting from scratch.
- Organizations must understand their business to determine which key parameters are actually useful to track and will help in the optimizing their business and processes.
- Organizations must be willing to study outliers to discover how and why their processes are producing outliers. Following this, the organization must be willing to try to prevent these events by modifying their process.
- Organizations must be willing to continue to update their baselines as processes and technologies evolve.



## 7 References

- [1] Chrissis, M. B. , Konrad, M., and Shrum, S., CMMI Second Edition: Guidelines for Process Integration and Product Improvement, Addison-Wesley, 2007.
- [2] DataMyte Handbook, 7<sup>th</sup> Addition, ASI DataMyte, Inc., 2005.
- [3] Thomas, D.W, et al., (too many to mention), Statistical Quality Control Handbook, AT&T, Indianapolis, Indiana (1956).
- [4] Radio Technical Commission for Aeronautics (RTCA), <http://www.rtca.org/default.asp>.

# INTRODUCING A CERTIFICATE IN SOFTWARE TESTING FOR NON-MAJORS

Janet M. Drake  
Department of Computer Science  
University of Northern Iowa  
Cedar Falls, IA  
[drake@cs.uni.edu](mailto:drake@cs.uni.edu)

## Abstract

Software testing is an important part of software development. Fifty to 70% of development cost is used in verification and validation. Software testing is the major activity in verification and validation. University computer science programs do not produce enough qualified software developers much less software testers. Industry hires people without computer science degrees and trained them to test software or testing is sent off shore. The result is that the most expensive part of the development process is done by the least well qualified people.

At UNI we are offering a Certificate in Software Testing for non-majors. These students have domain knowledge from their major field and the Certificate gives them the basic computer science skills to test software in their field.

## Introduction

Software testing is expensive – very expensive. The [NIST] report stated that 50% of software development costs are spent in Validation and Verification (V&V). For life critical software the V&V costs are even higher. Software testing is the largest part of V&V and therefore is the largest cost factor in software development.

The NICS report also said that we are not very successful at testing. The report said that software faults cost the US economy \$59.5 billion in 2002 (0.6% GNP). The quality of the software we produce is dismal. This is frightening because we are relying more and more on software for our physical and financial safety.

We spend lots of money and still don't do a good job. What is wrong? I believe that both industry and computer science education have to take some responsibility for this problem. We do not teach testing and industry does not put their best employees in testing. Industry is more focused on time to market and profit than quality. Tom DeMarco discusses this problem in [DeMarco].

As educators how can we attack this problem? First, I believe we have to teach testing in every computer science course. Second, I believe that we have to educate more people to be software testers. Third we have to recognize that software testing is a lucrative research area. In this paper I will focus on educating more people to be software testers.

## Motivation

I have taught testing in a variety of ways in a variety of courses. Still the great majority of my students are working as software developers or system administrators and not as software testers. We educate our students to value development or administration rather than testing. Industry hires our graduates as developers or administrators rather than testers. So both industry and educators are steering their best people away from testing – testing where good people could make the greatest financial difference. We need to change our value mindset.

## Testing in the Curriculum

The courses where I have taught software testing are:

*Introduction to Computing:* Introduction to boundary value and equivalence class testing were taught. Although boundary value and equivalence class testing sound initiative, they are only intuitive after being introduced. In introductory courses our goal is to teach basic

constructs. If we insisted on “bullet-proof” student programs, students would be spending all of their time validating input rather than learning the necessary concepts. Still students should realize that their programs usually do not pass minimum boundary value or equivalence class testing.

*Software Engineering:* Two weeks of lectures which include structural (white box) and functional (black box) testing are taught. For structural testing, students learn to use a Multiple Condition Decision Coverage (MCDC) approach. For functional testing, they learn to use boundary value, equivalence class, and transaction coverage. Even though they write and run test cases, this is only a short introduction. Testing is also addressed in the analysis/specification portion of Software Engineering. The Robertson and Robertson specification approach includes a “fit criteria” for each requirement. The fit criteria describes what the software must do to pass testing. This is the real starting point to testing. The fit criteria make a stronger, testable requirement.

*Software Testing:* Structural and functional testing are covered and student teams test a sizeable product. They use RequisitePro, an IBM-Rational tool that traced test cases to requirements. The course uses Boris Bizers book that introduces multiple ways to produces graphs from requirements and create test cases to cover graphs. The course only introduces other types testing (environmental, language, user interface, hardware ...). Automated testing is only introduced.

Even after all this testing education, we are not producing testers. Industry is still hiring non-computer testers. These people get on-the-job training and/or learn by doing.

Industry also out-sources testing to companies outside of the US. I believe industry does not want to send this work out of the country but they have little alternative because properly trained people are not available. Once these testing jobs are outside the US, it will be very hard to get them back. I believe that foreign testers do not have the cultural knowledge to adequate test products that will be used in the US. We are losing many good jobs by not providing an adequate workforce.

## **New Approach**

The UNI approach is to give non-computer science students a computer science/software testing background. They already have domain knowledge from their major area of study. With a little computer science education they are much more prepared to test software than the non-computer science people industry is currently hiring. In addition, the software testing skills provides work opportunities for graduates with majors in areas where fewer opportunities exist.

We offer a 15 credit Certificate in Software Testing. The Certificate attempts to model the work of software testers. Testers start work as part of the requirements team. They work to write requirement to ensure that each requirement is testable. When the development team

starts to design, the test team writes the test plan. Next the test environment must be developed and test cases created. Testers develop and execute test sets for many levels of testing. They can do structural testing on individual modules, testing on portions of a program during integration, and on the program as a whole. Test case creation is very creative and is most often done using a word processor, spread sheet, and configuration management tools.

Testes also often deal with automated testing tools. Regression testing tools are especially helpful. Tests can be automatically rerun after any change. Testers still have to develop the original test sets, validate the tests, and update them to accommodate any change in the requirements. Automated test tools are often built using Visual Basic and/or spread sheets. Some automated test tools help with structural testing by recording coverage during functional testing and finally reporting on missing coverage.

Software testers must have an overall understanding of the purpose of the software they are testing. Testers are closer to the customer in mindset than most other members of the software development team. The Certificate aims to take advantage of the students' major field of study for domain knowledge. Mathematics and actuarial majors with the Certificate will have the domain knowledge to test computational software. One example would be insurance company software. Physics majors with the Certificate will be naturals for testing engineering software. Biology majors could bring the domain knowledge and testing knowledge to medical and agricultural software.

## **Courses in the Certificate**

The courses in our certificate were selected to develop the skills that testers need.

*Visual Basic:* This course gives students the basic programming concepts – sequence, selection, and looping. Procedures and functions are also covered. Through the visual interface students learn an object oriented approach to interface design by using icons with properties and events. They learn to think at the basic level of programming and get an understanding of software development.

*Software Applications for Testing:* This course gives students experience with the type of tools they will use to create test cases and support testing documentation. The course covers advanced word processing techniques needed to create and update documentation. Spread sheets are widely used for tracking testing and actually making test cases. Several automated testing tools use spreadsheets for data storage. Database systems are also widely used in testing. Graphic modeling tools are also valuable to testers. In this class we are currently using Microsoft Office and Visio.

*Discrete Structures:* This course covers the logic needed to think like a tester. Conditional thinking, Boolean logic, and graphs are concepts needed by testers.

*Software Requirements Analysis:* In this course students learn about requirement specifications. They learn to elicit requirements from customers and represent the requirements in both graphic and textual forms. They learn to use case tools. The course focuses on making testable requirements.

*Software Testing:* Structural and function testing are covered in this course. MCDC structural testing is covered. Equivalence class, boundary value, and several approaches to graph base testing are covered. This is a project course and students work in a team to test a software application.

Visual Basic and Discrete Structures are prerequisites for Software Requirements Analysis. Software Requirements Analysis is a prerequisite for Software Testing.

## **Industrial Support**

The Computer Science Department at UNI has an Industrial Advisory Board. At a recent meeting the Software Testing Certificate was introduced and the board members were very enthusiastic. Although board members come from different companies, they all have difficulties in testing. They all found it difficult to hire testers and are anxious to support the Certificate program.

A Quality Control manager from Principal Financial Group visited the Visual Basic course to speak about careers in software testing and encouraged the students to consider the Certificate. He said that his company's starting salary for both testers and developers is the same.

## **Conclusion**

Software testing is a critical and expensive operation. Computer science graduates work as developers rather than testers and industry has difficulty hiring qualified testers. At UNI we are offering a Certificate in Software Testing for non-majors. The program gives non-majors computer science basics and, along with their domain knowledge from their major area, these students are prepared for positions as software testers.

## References

[NIST] Software Errors Cost U.S. Economy \$59.5 Billion Annually, NIST Assesses Technical Needs of Industry to Improve Software-Testing, [http://www.nist.gov/public\\_affairs/releases/n02-07.htm](http://www.nist.gov/public_affairs/releases/n02-07.htm), created 6/28/02

[DeMarco] DeMarco, Tom, "Why Does Software Cost So Much?", CrossTalk October 1994

# **The Characterization and Identification of Object-Oriented Model Defects**

**Mike Rowe and Robert W. Hasker**  
**Computer Science and Software Engineering Department**  
**University of Wisconsin – Platteville**  
**Platteville, Wisconsin 53818**  
[rowemi@uwplatt.edu](mailto:rowemi@uwplatt.edu), [hasker@uwplatt.edu](mailto:hasker@uwplatt.edu)

## **Abstract**

This paper presents a study of defects that commonly occur in object-oriented modeling. The study is based on experience from teaching more than a dozen sections of an Object-Oriented Analysis and Design course to sophomore and junior-level Software Engineering and Computer Science majors over the last eight years. The students use IBM (Rational) Rose as the design tool.

The goal of this research is to eventually provide real-time and anytime feedback for students as they develop their object-oriented models. It is hoped that this instant feedback will help students by discouraging them from developing bad habits and guiding them in the development of superior software models.



# 1 Introduction

While teaching the Object Oriented Analysis and Design course, we have observed that many software modeling defects occur multiple times, year after year. This paper categorizes these common defects by type of model: Use Case, Class, and Interaction and State. Examples of these defects as well as manual methods and potentially automated techniques for identifying some of these defects are described. This paper serves two goals: to provide a (non-exhaustive) catalog of errors that could be made available to students so they might be less likely to introduce the same defects, and to generate feedback from other instructors about these errors.

Eventually, it is hoped that satisfactory automated techniques will be made available for students and their instructors to help rapidly detect these defects and provide feedback to remedy the problem. The automated techniques proposed by this paper are of two types. The first automated approach involves the generation of C++ source code from the models, compiling it and studying the compilation errors. The second approach involves parsing Rose model (MDL) files to find specific defects directly.

The defects cataloged in this paper are based on having taught multiple sections of an Object-Oriented Analysis and Design course over the past eight years. This course covers modeling using the UML notation, though of course the concepts extend beyond any particular syntax. The students in the course are typically at the sophomore and junior level. The prerequisites are a course on fundamental data structures (CS2) and a project-based course on software engineering. Thus students are quite familiar with object-oriented programming and have already been introduced to some basic issues of object-oriented design. This course attempts to move students from applying the techniques on small problems towards modeling larger systems.

# 2 Related Work

Automated detection of defects in UML diagrams has received attention from a number of researchers [1][3][7][8][9][10][11][12]. However, these works focus on improving diagrams for professional developers. This paper focuses on errors typically made by students – people who are not only learning UML, but also learning basic issues about how to apply modeling in general.

This paper discusses errors typically made by students. [2] and [13] also examine errors made by students, but these catalog only a few errors and some of these are controversial. [5] and [14] discuss common errors in more depth, but focus on errors made by CS1 students. This paper discusses errors made by students at the sophomore/junior level. These are students who already have a basic understanding of object-oriented programming but do not yet appreciate certain subtleties of both modeling and object-oriented design.

### 3 Use Case Model Defects

There are a number of frequent defects in use case models: failing to use verb phrases for cases, misusing extends and includes, and use cases which capture insignificant interactions.

#### 3.1 Use Case Titles That Are Not Verb Phrases

**Description:** Use cases are common uses of a system that describe how an actor or actors obtain a significant benefit from a system. Since they describe this process of obtaining a benefit, they are best titled as verb or verb-object phrases.

**Detection:** Use case titles are generally concatenated strings of two or more words. This makes it difficult to parse the strings into separate words unless some syntax is enforced such as “camel case” or underscore word separators. If the syntax allows parsing, the component words in the use case title can be compared to a lexicon or use natural language technology to recognize parts of speech. If a use case title is used inside of the scenario, parsing the use case title in the syntax of the sentence may be able to reveal its part of speech.

#### 3.2 Misuse of Extends and Includes

**Description:** Use case extensions and inclusions are often confused. Most students can give the strong definitions of these constructs, but often confuse the direction of the arrows when producing use case diagrams. For an “include” relationship, the arrow should be on the included use case side, whereas with an “extension” relationship, the arrow should be on the far side of the extending use case.

**Detection:** This may be difficult to detect with only the use case diagram. A solution may be to cross check the diagram against use case scenarios. Formal use case scenarios typically list extensions and inclusions. By locating the use case scenario title in the use case model, we can analyze the model from this point, comparing the model’s extends and includes against the scenario’s. The automatic checking for this defect would rely on very specific use case scenario templates.

#### 3.3 Insignificant Use Cases

**Description:** The goal of a use case is to provide a significant benefit to an actor associated with that use case. Use cases without a significant benefit should be combined with others.

**Detection:** The above definition does not meet many of the IEEE STD 830 [6] characteristics of “good” requirements. To say the least, this does not meet the test of verifiability, in that we would be hard pressed to obtain universal consensus on precisely what is meant by “significant benefit”. We believe that certain cases can be defined and recognized, but further research is needed in this area.

## 4 Class Model Defects

Because they are richer, the number of potential errors in class models is larger.

### 4.1 Non-noun Class Names

**Description:** Classes are the stuff from which objects are made, and objects are nouns. Students frequently name classes using verbs, possibly because they are focusing on the actions performed as part of the class rather than the object which performs those actions.

**Detection:** The detection of non-noun class names could be partially automated by parsing scenarios, identifying which words are used most frequently as nouns, adjectives, or verbs, and using this information to identify misused words in class names.

### 4.2 Reversed Multiplicities

**Description:** The collection class of an aggregation or composition should have a multiplicity of '1', whereas the parts can have any multiplicity. This mistake is also commonly seen with other class relationships.

**Detection:** This can be detected with aggregation and composition, when the collection class has a multiplicity not equal to '1'. See Figure 1 for a UML example of reversed multiplicities and Figure 2 for a corresponding MDL file snippet. With non-collection relationships, this cannot be easily detected without domain knowledge about the classes and their relationship.

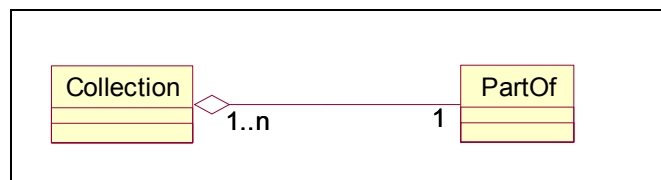


Figure 1: UML model with reversed multiplicities

```
// snippet from MDL file
root_category (object Class_Category "Logical View"
  quid "47BD93AC01B0"
  exportControl "Public"
  global TRUE
  subsystem "Component View"
  quidu "47BD93AC01B2"
  logical_models (list unit_reference_list
    (object Class "Collection"
      quid "47BD942B03D2")
    (object Class "PartOf"
      quid "47BD942F0346"
      documentation "A part of the collection.") // *
    (object Association "aggregation"
```

```

quid    "47BD943502E8"
roles  (list role_list
  (object Role "$UNNAMED$0"
    quid    "47BD94360384"
    supplier "Logical View::PartOf"
    quidu   "47BD942F0346"
    client_cardinality (value cardinality "1") // multiplicity of part of: 1
    is_navigable TRUE)
  (object Role "$UNNAMED$1"
    quid    "47BD94360395"
    supplier "Logical View::Collection"
    quidu   "47BD942B03D2"
    client_cardinality (value cardinality "1..n") // multiplicity of Collections: 1..n
    is_navigable TRUE
    is_aggregate TRUE)))) // indicates Collection is an aggregate

```

Figure 2: Rose MDL file snippet for defective aggregation multiplicities.

### 4.3 Only Public Operations and Attributes in Implementation-level Models

**Description:** An implementation-level class diagram should be at the detail that goes beyond interfaces and public operations. If almost all operations across all classes are public, then either the model not have enough detail to include private operations or the designer has not properly identified which functions should be private. In either case, there is a problem. Generally, it is hard to defend the concept that any class attribute should be public.

**Detection:** The detection of this defect is as simple as calculating a percentage of public to non-public operations in a class model. As long as the percentage is larger than some arbitrary value (well less than 100%) then this defect may be present. An acceptable percentage will depend on the domain, the design, and the expected level of detail in the design. Recognizing any public attributes can also be flagged as very likely defects

### 4.4 Classes, Operations, and Attributes without Documentation

**Description:** Rose class models allow the designers to add documentation to classes, operations, attributes, and operation parameters. This documentation is inserted into the source code that is generated from a class model. Documenting the elements while designing the system is an excellent practice since this is when designers are most likely to be intimate with both the requirements and the elements of the design that satisfy those requirements.

**Detection:** Identifying missing documentation is straightforward. The line marked by an asterisk in Figure 2 marks the documentation for class part of (“a part of the collection”). In contrast, class Collection, listed a couple lines earlier in the file, is missing its documentation.

### 4.5 Associations without Navigation Attributes

**Description:** Navigational attributes are reference or pointer variables that allow one class to access another class. In a class model, they are part of the reference and on the side of the navigational arrow. When code is generated from a class model, a navigational attribute produces a reference or pointer class variable of the type of the associated class.

**Detection:** Detection of missing navigational attributes can be detected by processing the Rose MDL file's references for missing navigational attributes. See Figure 3 for UML model, Figure 4 for MDL file associated with the UML, and Figure 5 for C++ code auto-generated by Rose from the model.

This “rule” is somewhat controversial: some instructors would prefer students to generate diagrams with less redundancy in them. This illustrates a basic requirement for any automated system: it must be possible for instructors to select which rules to apply for a particular course or even assignment.

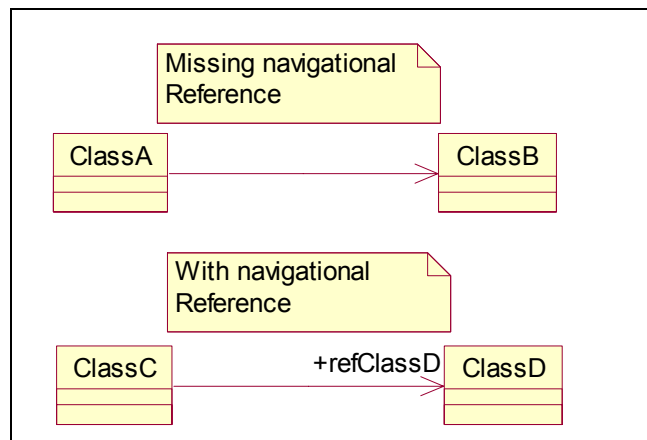


Figure 3: UML without and with navigational references

```

logical_models (list unit_reference_list
  (object Class "ClassA"
    quid "47BDA0820067")
  (object Class "ClassB"
    quid "47BDA08500A6")
  (object Class "ClassC"
    quid "47BDA087023C")
  (object Class "ClassD"
    quid "47BDA08A00F4")
  (object Association "$UNNAMED$0"
    quid "47BDA0B803B3"
    roles (list role_list
      (object Role "$UNNAMED$1" // Un-named navigational reference
        quid "47BDA0B903E2"
        supplier "Logical View::ClassB" // type of reference
        quidu "47BDA08500A6"
        is_navigable TRUE) // Navigable
    )
  )
)
  
```

```

(object Role "$UNNAMED$2"
  quid "47BDA0B903E4"
  supplier "Logical View::ClassA"
  quidu "47BDA0820067"))
(object Association "$UNNAMED$3"
  quid "47BDA0BD01EE"
  roles (list role_list
    (object Role "refClassD" // named navigational reference
      quid "47BDA0BE021D"
      label "refClassD"
      supplier "Logical View::ClassD" // type of reference
      quidu "47BDA08A00F4"
      is_navigable TRUE) // Navigable
    (object Role "$UNNAMED$4"
      quid "47BDA0BE021F"
      supplier "Logical View::ClassC"
      quidu "47BDA087023C"))))

```

Figure 4: MDL file snippet of relationships without and with navigational references

```


///##ModelId=47BDA0820067
class ClassA // notice no reference to Class B is generated
{
};

class ClassC
{
  public: // this is the generated reference to Class D
    ///##ModelId=47BDA0BE021D
    ClassD *refClassD;
};


```

Figure 5: Code auto-generated from Rose without (ClassA) and with (ClassC) navigational references.

## 4.6 Attributes and Operations that are not Typed

**Description:** At the implementation level, class model attributes, operations, and operation parameters need to be typed to support code development.

**Detection:** This defect can be detected by using Rose to generate source code from the class model and compiling it. Non-typed identifiers are not permitted in many high-level languages and will produce syntax errors. For example, the C++ code generated for NonTypedClass in Figure 6 is

```
Class NonTypedClass { public: opp(void agr1); private: att1; };
```

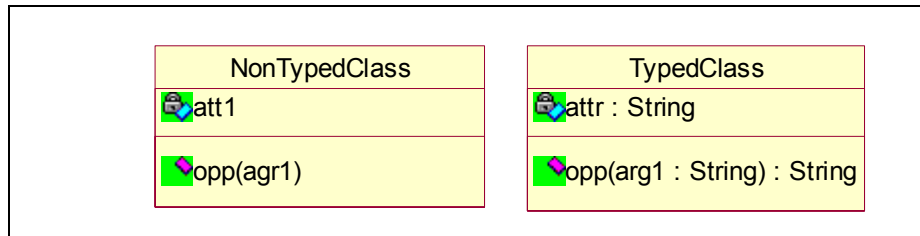


Figure 6: UML without and with typed attributes, operations, or operation arguments

#### 4.7 Illegal Identifiers for Target Language

**Description:** Classes should not be a dead end in a software development process. The class names, attributes, operations, and parameters of the model produce the identifiers in the generated source code. If illegal identifiers are used in the model, they will appear in the generated code.

**Detection:** This defect can be detected by using Rose to generate source code from the class model and compiling it. The compiler will produce errors relating to these illegal identifiers. Below is an example of a class with illegal C++ identifiers as they contain embedded spaces, “attr One”, “opp Two”, and “arg Three”. When the Rose-generated code is compiled, parse errors result on the identifiers.

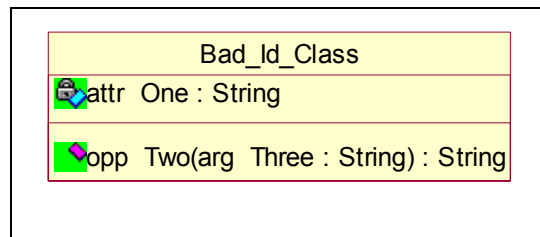


Figure 7: A class with incorrect identifiers for an attribute, an operation, and a parameter of the operation

#### 4.8 Inheritance Arrows in the Wrong Direction

**Description:** A common student error is that to reverse the inheritance arrows, placing the inheritance arrow on the child rather than the parent side of the relationship.

**Detection:** In general it could be very difficult to identify reversed inheritance arrows automatically. However, it is possible in some cases: given that multiple inheritance is rare in student solutions, a class with multiple outgoing generalization arrows suggests the presence of an error. Common cases in which multiple inheritance is encouraged, such as for the Composite Pattern [4] shown in Figure 8, can be detected as special cases.

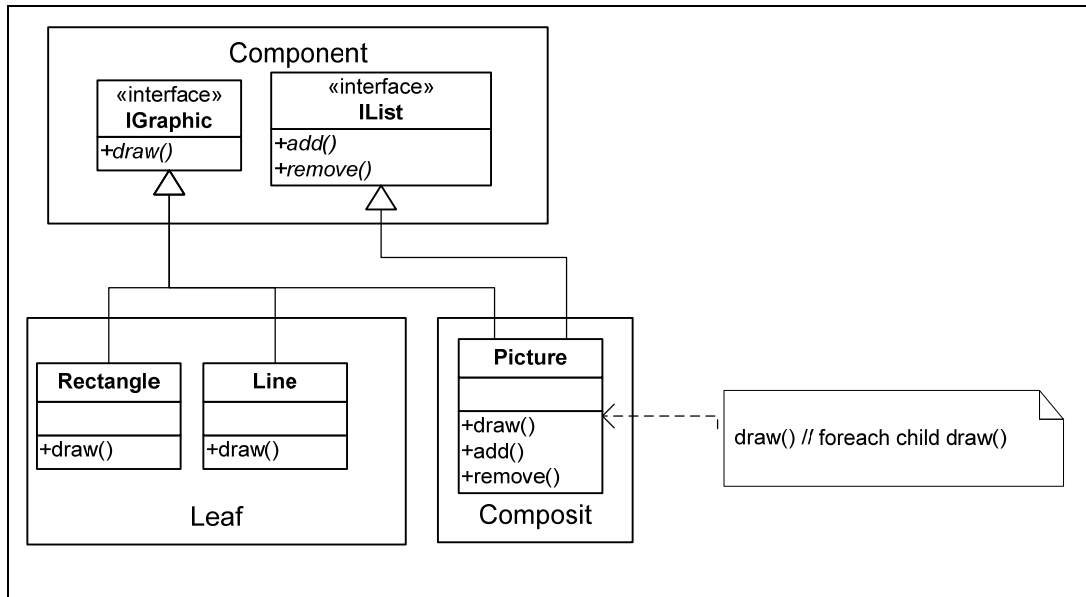


Figure 8: An application of the Composite Pattern in which Picture inherits two interfaces

#### 4.9 Duplicate Operations in Multiple Classes

**Description:** It is relatively rare for an operation to be duplicated in multiple classes (except to support inheritance) in class projects. Duplicated operations often indicate poor class cohesion or misplaced operations. While duplicated operations certainly do not indicate an actual defect, they can trigger a message discussing the concern.

**Detection:** This type of defect can be detected by scanning the Rose model file for duplicate operation names. The danger is leading students to believe that all duplication is invalid. Some generic operations, such as “sort” or “find” are likely to appear in several classes. Filtering out common operation names is likely to be necessary to avoid teaching students invalid concepts.

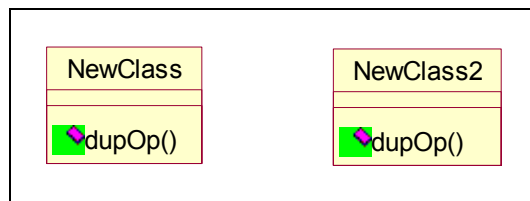


Figure 9: UML of two classes with the same operation

#### 4.10 Classes without Attributes and/or Operations

**Description:** Once a model gets to the design or implementation phases, it should have at least one unique attribute or operation. Classes without attributes or operations imply either that the class may have been motivated by the analysis but not be relevant to the final system or that the model is incomplete.



**Detection:** This type of defect can be spotted by scanning the Rose MDL file for classes that do not have “(object operation...” for class operations or “(object ClassAttribute...” class attribute entries.

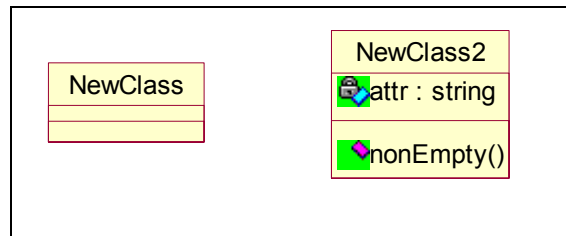


Figure 10: UML of an empty and non-empty class. The non-empty class contains the operation nonEmpty( ).

```

root_category (object Class_Category "Logical View"
  quid "47BD9B0402AA"
  exportControl "Public"
  global TRUE
  subsystem "Component View"
  quidu "47BD9B0402AC"
  logical_models (list unit_reference_list
    (object Class "NewClass" // empty Class
      quid "47CC0AAE03D8")
    (object Class "NewClass2" // non-empty Class
      quid "47CC0AB102EE"
      operations (list Operations
        (object Operation "nonEmpty" // non-empty Class has an Operation
          quid "47CC10170157"
          result "void"
          concurrency "Sequential"
          opExportControl "Public"
          uid 0))
      class_attributes (list class_attribute_list
        (object ClassAttribute "attr" // non-empty Class has an Attribute
          quid "47CC204E0177"
          type "string")))))
  
```

Figure 11: Rose MDL file snippet of empty and non-empty classes.

#### 4.11 Classes that are not Associated with Other Classes

**Description:** Classes interact with each other, providing and using services. To be useful in a system, classes need to be associated with other classes.

**Detection:** This type of defect can be spotted by scanning the Rose MDL for classes which are never referenced in the “(object Association . . .” lists. See the lines annotated with “//” comments in Figure 12.

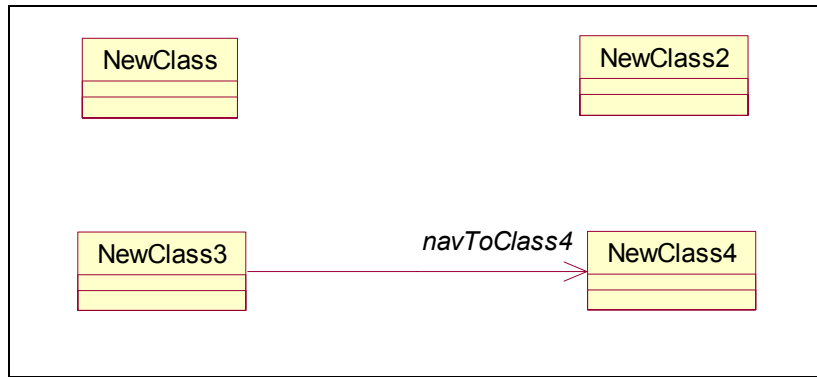


Figure 12: UML example of two classes without associations and two classes with associations.

```

root_category (object Class_Category "Logical View"
  quidu "47BD9B0402AA"
  exportControl "Public"
  global TRUE
  subsystem "Component View"
  quidu "47BD9B0402AC"
  logical_models (list unit_reference_list
    (object Class "NewClass"
      quidu "47CC0AAE03D8")
    (object Class "NewClass2"
      quidu "47CC0AB102EE")
    (object Class "NewClass3"
      quidu "47CC21E900DA")
    (object Class "NewClass4"
      quidu "47CC21EE038A")
    (object Association "navToClass4"
      quidu "47CC21F501B5"
      roles (list role_list
        (object Role "$UNNAMED$0"
          quidu "47CC21F60109"
          supplier "Logical View::NewClass4" // Associated Class
          quidu "47CC21EE038A"
          is_navigable TRUE)
        (object Role "$UNNAMED$1"
          quidu "47CC21F6010B"
          supplier "Logical View::NewClass3" // Associated Class
          quidu "47CC21E900DA")))))
  
```

Figure 13: MDL file of two classes without associations (NewClass1 and NewClass2) and two classes with associations (NewClass3 and NewClass4).

## 4.12 Very High Class Coupling

**Description:** Good object-oriented design strives for low coupling and high cohesion. High coupling is associated increased maintenance costs because when one class changes, the coupled classes are more likely to require changes.

**Detection:** The detection of high coupling is rather subjective in that the amount of acceptable coupling depends on the problem domain. However, a coupling metric can be computed for each class by processing the Rose MDL file associations and counting the number time each class name appears in the “(object Association(roles (object Role supplier)))” fields. A simple statistical analysis can be used to indicate which classes might be candidates for being coupled to too many others.

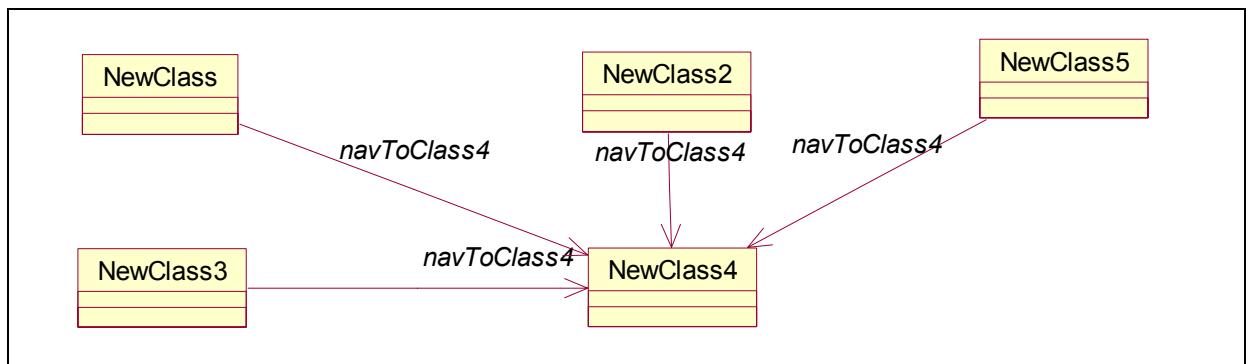


Figure 14: UML showing NewClass4 associated with the other four classes – high coupling.

```
logical_models (list unit_reference_list
  (object Class "NewClass"
    quid "47CC0AAE03D8")
  (object Class "NewClass2"
    quid "47CC0AB102EE")
  (object Class "NewClass3"
    quid "47CC21E900DA")
  (object Class "NewClass4"
    quid "47CC21EE038A")
  (object Class "NewClass5"
    quid "47CC24F702AF")
  (object Association "navToClass4"
    quid "47CC21F501B5"
    roles (list role_list
      (object Role "$UNNAMED$0"
        quid "47CC21F60109"
        supplier "Logical View::NewClass4"
        quidu "47CC21EE038A"
        is_navigable TRUE)
      (object Role "$UNNAMED$1"
        quid "47CC21F6010B"
        supplier "Logical View::NewClass3" // NewClass3 associated with
        // NewClass4
```

```

        quidu          "47CC21E900DA"))))
(object Association "navToClass4"
  quid   "47CC24E8034B"
  roles  (list role_list
    (object Role "$UNNAMED$2"
      quid   "47CC24EA01C5"
      supplier "Logical View::NewClass4"
      quidu   "47CC21EE038A"
      is_navigable TRUE)
    (object Role "$UNNAMED$3"
      quid   "47CC24EA01C7"
      supplier "Logical View::NewClass" // NewClass associated with
                                                // NewClass4

        quidu          "47CC0AAE03D8"))))
(object Association "navToClass4"
  quid   "47CC24EE0148"
  roles  (list role_list
    (object Role "$UNNAMED$4"
      quid   "47CC24F100CB"
      supplier "Logical View::NewClass4"
      quidu   "47CC21EE038A"
      is_navigable TRUE)
    (object Role "$UNNAMED$5"
      quid   "47CC24F100CD"
      supplier "Logical View::NewClass2" // NewClass2 associated with
                                                // NewClass4

        quidu          "47CC0AB102EE"))))
(object Association "navToClass4"
  quid   "47CC24FE02AF"
  roles  (list role_list
    (object Role "$UNNAMED$6"
      quid   "47CC2500000F"
      supplier "Logical View::NewClass4"
      quidu   "47CC21EE038A"
      is_navigable TRUE)
    (object Role "$UNNAMED$7"
      quid   "47CC25000011"
      supplier "Logical View::NewClass5" // NewClass5 associated with
                                                // NewClass4

        quidu          "47CC24F702AF"))))

```

Figure 15: Rose MDL file showing NewClass4 associated with the four other classes.

## 5 Interaction and State Model Defects

This section discusses the most significant defect in dynamic diagrams: failing to be consistent with static diagrams. Identifying additional issues is left as future work.

## 5.1 Message Arcs and Class/Objects that do not Correspond to the Class Model

**Description:** In a project, all of the object-oriented models model the same domain objects and such should be based on the same model components. Many students fail to make this connection. As a result, they build each model from scratch and ignore the work already done in previous modeling. Frequently, students will have wonderfully refined class models and then use different identifiers for classes, attributes and operations when producing interaction and state models. On the other hand, Rose and many other design tools provide mechanisms for ensuring consistency between diagram types. In Rose, drop-down menus give appropriate suggestions based on components from the class model – all a developer needs to do is click on the appropriate identifier. Students need encouragement to use such assistance.

**Detection:** The detection of this defect can be achieved by scanning the interaction and state model parts of the Rose MDL file to determine if all components are already part of the class model.

## 6 Conclusion

We have identified a number of frequent errors made by students when constructing UML diagrams. This list is certainly not intended to be exhaustive, but in our experience these defects have the distinction of being both easily recognized (at least by instructors) and very common. Future plans include developing tools to automatically recognize many of these defects. The intent is that students would use the tools to get anytime feedback on their models, presumably resulting in improved submissions. Thus the goal is an automated assistant: developing a system to identify relatively simple errors. This will hopefully allow instructors to spend more time on more significant issues.

## References

- [1] Cleidson, R. B., et al., Using Critiquing Systems for Inconsistency Detection in Software Engineering Models. SEKE 2003, pp. 196-203.
- [2] Coelho, W. and Murphy, G., ClassCompass: A Software Design Mentoring System. *ACM Journal on Educational Resources in Computing*, Vol. 7, No. 1, Article 2, March 2007.
- [3] Egyed, A., UML Analyzer Tool. Information available at [http://www.alexander-egyed.com/tools/uml\\_analyzer\\_tool.html](http://www.alexander-egyed.com/tools/uml_analyzer_tool.html). Accessed March 7, 2008.
- [4] Gamma, Helm, Johnson, and Vlissedes, *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, 1995.

- [5] Holland, S., Griffiths, R., and Woodman, M., Avoiding Object Misconceptions. *SIGCSE Bull.* 29, 1 (Mar. 1997), pp. 131-134.
- [6] IEEE Std 830-1998 IEEE Recommended Practice for Software Requirements Specifications, [http://standards.ieee.org/reading/ieee/std\\_public/description/se/830-1998\\_desc.html](http://standards.ieee.org/reading/ieee/std_public/description/se/830-1998_desc.html), IEEE, 1998.
- [7] Kaneiwa, K., and Satoh, K., Consistency Checking Algorithms for Restricted UML Class Diagrams. In Proceedings of the Fourth International Symposium on Foundations of Information and Knowledge Systems (FoIKS 2006), *Lecture Notes in Computer Science*, Volume 3861, Springer-Verlag, 2006, pp. 219-239.
- [8] Konrad, S. and Cheng, B.H.C., Automated Analysis of Natural Language Properties for UML Models. *Lecture Notes in Computer Science*, Volume 3844, Springer-Verlag, 2006, pp. 48-57.
- [9] Lange, C., Improving the Quality of UML Models in Practice. In *Proceedings of the 28th international Conference on Software Engineering* (Shanghai, China, May 20 - 28, 2006). ICSE '06. ACM, New York, NY, 993-996.
- [10] Lindland, O., Sindre, G., Understanding Quality in Conceptual Modeling. *IEEE Software*, March 1994, pp. 42-49.
- [11] Pap, Zs., Majzik, I., Pataricza, A., and Szegi, A., Completeness and Consistency Analysis of UML Statechart Specifications. In Proc. IEEE Design and Diagnostics of Electronic Circuits and Systems Workshop (DDECS'2001), Győr, Hungary, 18-20 April, 2001, pp. 83-90.
- [12] Pilskalns, O., and Andrews, A., Rigorous Testing by Merging Structural and Behavioral UML Representations. In 6th International Conference on the Unified Modeling Language (UML 2003), San Francisco, USA, October 20-24, 2003.
- [13] Thomasson, B., Ratcliffe, M., and Thomas, L., Identifying Novice Difficulties in Object Oriented Design. ITiCSE'06, June 26-28, 2006, Bologna, Italy, pp. 28-32.
- [14] Sanders, K., and Thomas, L., Checklists for Grading Object-Oriented CS1 Programs: concepts and misconceptions. ITiCSE '07, June 23-27, 2007, Dundee, Scotland, pp. 166-170.

# Exploring the Web Programming Jungle

Charles M. (Mike) Morrison  
Department of Computer Science  
University of WI – Eau Claire  
Eau Claire, WI 54702  
morriscm@uwec.edu

## Abstract

Selecting the content for a web programming class is complicated by the large number of protocols and technologies that can be used to accomplish similar tasks. A recent look at a number of university catalogs showed many computer science programs do not list a web programming class, and when they do they tend to focus on either Microsoft technologies, J2EE technologies, or something else entirely like PHP or Ruby on Rails. We propose a balanced course offering with topics taken from eight broad categories of web technologies that we define. These categories are: communication protocols, markup languages, web servers, server-side programming, client-side programming, frameworks, application deployment, security, and development environment.

## Introduction

A recent look at a number of university catalogs listing a computer science web programming class showed that a class of this type is often missing. When one is listed, the topics covered vary widely and tend to focus on a small set of technologies. So what are reasonable goals for a web programming class? First of all, we think a student should have completed at *least* two programming classes and have a reasonable understanding of common algorithms and data structures before taking this class. Next, our primary goal is for this class to be about creating web applications, not creating a web browser or writing the code for a web server. But before deciding what should be in this class, we made a list of well know Web technologies and categorized them.

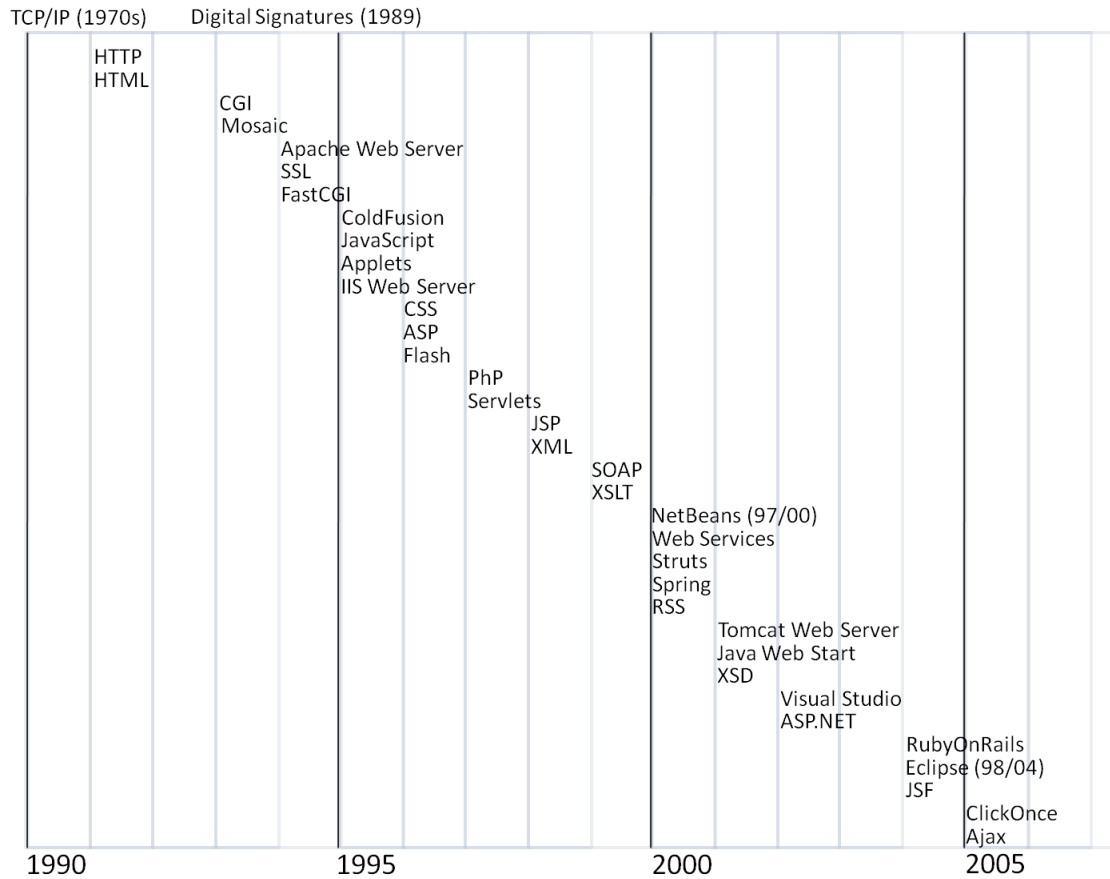
It quickly became apparent that there are many languages, development environments, and technologies that can potentially be used in this class, and they all can't be covered in depth. Should a web programming class educate students with a broad sampling of the technologies available to web programmers or go into depth on a small subset of technologies? We recommend teaching about a broad sampling of technologies and covering a smaller subset of technologies in more detail. We think that if students are experienced programmers, developing general programming skills and introducing additional languages are less important than understanding programming in a web context and making intelligent choices on what technology is appropriate in various situations.

This paper summarizes major web technologies available today and discusses how they can be incorporated into a web programming class. A curriculum is then suggested for this class that uses a variety of technologies.

## Web Technologies

TCP/IP protocols and the Internet were largely formulated during the 1970s. The Internet didn't hit the mainstream, however, until the 1990s when Tim Berners-Lee's work developing the World Wide Web converged with the development of graphical browsers and the ability to process user inputs submitted from HTML forms. Mosaic was the first graphical browser in 1993 and also in 1993, the Common Gateway Interface, CGI, provided a standard way for web servers to start executable programs (written in any language), pass HTML form inputs to them, and pass program outputs back to browsers. From 1993 onward, new web technologies appeared rapidly (Figure 1).





**Figure 1 - Web Technology Timeline**

Table 1 provides a brief description of the technologies shown in Figure 1. Although all of these technologies can be discussed in a Web programming class, they cannot all be covered in depth. Therefore these have been grouped into eight broader categories with the idea that a few technologies in each category can be given in depth coverage. Table 2 shows these categories.

<b>Technology (in chronological order)</b>	<b>Description</b>
TCP/IP	Essential Internet Protocols used in the Web
DS – Digital Signatures	Public Private Key Encryption
HTML - HyperText Markup Language	Document Markup Language
HTTP - HyperText Transport Protocol	Web Communication Protocol
CGI – Common Gateway Interface	Protocol Linking Server Apps to Web Server
Apache Web Server	Web Server
FCGI – FastCGI	Improved More Scalable CGI
SSL – Secure Sockets Layer	Protocol for Secure Internet Communications
IIS Web Server	Microsoft Internet Information Services Web Server
CF – ColdFusion	Server Framework and Language
JavaScript	Client Scripting Language
JA – Java Applet	Web Deployment Technology
CSS – Cascading Style Sheets	Stylesheet Language
ASP – Active Server Pages	Server Scripting Engine
Flash	Web Oriented Multimedia Technologies
PHP – Hypertext Preprocessor (R)	Server Scripting Engine
JS – Java Servlets	Server Framework
JSP – Java Server Pages	Server Scripting Language
XML – eXtensible Markup Language	Document Markup Language
SOAP – Simple Object Access Protocol	Protocol for Exchanging XML Messages via HTTP
XSLT – eXtensible Stylesheet Language Transformations	Language for Transforming XML Documents
NetBeans	Integrated Development Environment (open source 00)
WS – Web Service	App to App Web Based Communication Protocol
Struts – Apache Struts	Web Application Framework
Spring – Java Spring Framework	Application Framework
RSS – Really Simple Syndication	Formats for Publishing Frequently Updated Content
XSD – XML Schema Language	XML Schema Definition Language
Apache/Tomcat Web Servers	Web Server Supporting JSP and Servlets
JWS – Java Web Start/JNLP	Web Deployment Technology
Visual Studio	Integrated Development Environment
ASP.NET	Web Application Framework (C#, VB, etc.)
RoR – Ruby on Rails	Web Application Framework
Eclipse	Integrated Development Environment (open source 04)
JSF – Java Server Faces	Web Application Framework
ClickOnce	Web Deployment Technology
AJAX – Async JavaScript and XML	Group of Technologies for Interactive Web Apps

**Table 1 Significant Web Technologies**

<b>Category</b>	<b>Technology</b>
Communication Protocols	TCP/IP, DS, HTTP, SSL, SOAP, WS, RSS
Markup Languages (and related)	HTML, CSS, XML, XSD, XSLT
Client-Side Programming	JavaScript, Flash, AJAX
Web Servers	Apache, IIS, Apache/Tomcat
Server-Side Programming	CGI, FCGI, ASP, PHP, JS, JSP
Frameworks	CF, Struts, Spring, ASP.NET, RoR, JSF
Application Deployment	JA, JWS, ClickOnce
Security	HTTPS, DS, SSL/TLS, Certificates
Development Environments	VS, Eclipse, NetBeans, Text Editor, etc.

**Table 2 – Categorized Web Technologies**

## **Communication Protocols - TCP/IP, HTTP, SOAP, WS, RSS**

If this class is offered prior to taking a networks course a single lecture hitting the high points of the TCP/IP protocol should cover the aspects TCP/IP that are important to web programming. These include IP addressing, domain names, domain name servers, the host configuration file, and URLs.

Similarly a lecture should go over the high points of the HTTP protocol. This would include a discussion of HTTP clients (browsers) and servers (web servers). A brief summary of the HTTP request methods (with particular attention to GET and POST) and an explanation of typical HTTP request and response messages would also be appropriate – however – the web programming discussed in this paper is focused on creating web applications and not on creating browsers and servers.

The simple object access protocol, SOAP, provides the foundation layer for web services. This is an advanced, topic that adds additional protocols like WSDL and UDDI to the mix. It merits inclusion in a web programming class – but if presented as a programming assignment, this should be after the basics of creating and deploying web applications are understood.

Really simple syndication, RSS, is another advanced topic that is of interest, but would rate behind SOAP and web services in our hierarchy of what to cover in depth. It provides a way to use an RSS document to specify web content that can be sent from one web site to many other web sites. An intermediate company called an aggregator periodically searches the registered web sites for RSS documents and displays information about the feed so clients can link to documents of interest.

## **Markup Language (and Related) - HTML, CSS, XML, XSD, XSLT**

Web outputs are normally structured as HTML documents that are displayed in a browser. Therefore, a thorough understanding of the hypertext markup language HTML is essential to a web programmer. Although HTML is picked up quickly by most computer science students, you can't assume they already know it. It's important to allocate one or two lectures to discussing the structure of an HTML document.

Cascading style sheets, CSSs, are commonly used to format an HTML documents and merit part or possibly all of a lecture during the initial HTML part of the class. CSS id, class, and tag styles should be explained and demonstrated. When the time finally comes to write web programs, however, we recommend focusing on web programming issues and avoiding the extra complexity of CSSs.

A lecture on the extensible markup language, XML, will be needed sometime during the semester, but it doesn't have to come immediately after the HTML/CSS lectures. It's likely, however, that the development environment and framework chosen

for web programming projects will use XML configuration files. It is also the text format used with SOAP messages and RSS documents. XML also has a place in sending structured database data from one system to another. Where you include a lecture on XML depends on other technology choices you make for the class and when they are used.

XML schema definitions, XSDs, are used to specify data types, relationships, and cardinalities within sets of XML data. This is of interest to a database class; but less so to a web programming class. Keep in mind that anything done in a traditional database program can be done with a web interfaced program.

The extensible stylesheet transformations language, XSLT, provides programming commands and structures to process and format XML data. XSLTs are built on top of the XML Path Language, XPath. XSL templates are used to retrieve data, while XPATH provides the syntax for processing the data. We do not think XSLTs are an important topic for a web programming class. Although this is an interesting technology with an unusual programming style, reading a file in one format and generating a new file in a different format is something that can be done with any programming language and in any class – not just a web programming class.

## **Client-Side Programming – JavaScript, Flash, AJAX**

JavaScript is a client-side scripting language used within web browsers. It started life as a Netscape product named LiveWire and was informally renamed JavaScript and moved into the public domain as a joint effort by Netscape and Sun Microsystems in 1995. When it was placed in the public domain, the ECMA Organization was chosen to maintain it. JavaScript's official name is ECMAScript. Although Sun participated in moving JavaScript into the public domain, it did not convert JavaScript into a Java clone.

JavaScript is loosely typed, doesn't support creating custom classes, is blocked from accessing resources on the local computer, and relies on the web browser hosting it to display outputs to users. Students with prior Java programming experience will still find it useful to learn about JavaScript and work with it.

Adobe Flash is a multimedia design environment used to add interactive, dynamic content to web pages. It was introduced in 1996 and is widely used. Flash has a wide range of graphic design tools and includes a scripting language. It has a steep learning curve, however, and is aimed towards graphic designers more than programmers.

Asynchronous JavaScript and XML, AJAX, appeared on the scene in 2005. Its goal is to increase the responsiveness and interactivity of web pages by exchanging small amounts of data with the web server without requesting a new web page. AJAX uses JavaScript to dynamically display and interact with information. The XMLHttpRequest object is used to transfer text data (typically in XML format – but any text can be

transferred) between the browser and server. AJAX is a new technology that is used in map sites like Google Maps and Map Quest and might become widely used in the future.

## **Web Server - Apache, IIS, Apache/Tomcat**

There are many commercial and open source freeware web servers that can be used to deliver web content. Only two stand out, however. According to a Netcraft survey taken in February 2008 (<http://news.netcraft.com/archives/2008/02/index.html>) the Apache web server has 51% of the web server market and Microsoft's IIS accounts for 36% of the market. Both are readily available in most academic environments.

Both web servers have mechanisms for specifying home directories, accessible directories, allowed activities within directories, and more. IIS is configured with utilities displaying dialog boxes and wizards. Apache is configured by modifying the contents of XML formatted text files. Students should be familiar with the concept of virtual directories (IIS) and aliases (Apache). If you plan to use Java servlets or JSP pages, the Tomcat web server also need to be discussed, since it is normally combined with the Apache server and provides the servlet and JSP processing capabilities. (Tomcat uses contexts instead of aliases or virtual directories). A lecture should be allocated to showing how Apache, IIS, and possibly Apache/Tomcat are configured. When testing web programs the development computer will normally have a version of IIS or Apache/Tomcat locally installed – so an assignment configuring a web server could be worked out.

## **Server-Side Programming - CGI, FCGI, ASP, PHP, JS, JSP**

The common gateway interface, CGI, is a protocol specifying how a server-side program can get user inputs from a web server and send outputs back to the web server to be forwarded on to the browser as a web page. It is the first technology developed for this and is still widely supported. It doesn't specify a language, however. Any language capable of writing to stdout and reading from stdin (which means just about any language) can be used. In practice, the PERL scripting language is most often used in combination with CGI. FastCGI is similar to CGI. It scales up better than CGI; however, since the programs using it can be run in threads within the Web server process rather than having to start separate processes for each one as CGI does.

Active Server Pages, ASP, Hypertext Preprocessor Recursive, PHP, Java Servlets, JS, and Java Server Pages, JSP, specify both the language to be used and the technology connecting the web programs with the web server. All scale up better than the initial CGI

protocol. If ASP is used, the language is usually VB. PHP has its own unique language, and JS and JSP rely on Java.

Java Servlets are traditional Java programs extended with three javax.servlet packages. These imported packages add methods allowing the java program to retrieve HTML form inputs from a web server and send responses in HTML format back to the web server. Sending HTML to the browser takes this form:

```
PrintWriter out = response.getWriter();
java.util.Date now = new java.util.Date();
out.println("<html><body>HelloWorld! The time is " + now +
"</body></html>");
```

JSPs take a different approach. They make a page using normal HTML tags and when needed, embed special script tags containing java commands and special tags displaying results from the commands in the resulting web page. This model of embedding programmatic code with HTML is also used by ASP, PHP, and ColdFusion. For example:

```
<html>
<body>
<% java.util.Date now = new java.util.Date() %>
Hello World! The time is <%= now %>
</body>
</html>
```

JSPs are compiled into servlets when they are first accessed on the web server. If the page isn't modified, the compiled file doesn't have to be compiled again when it is accessed in the future. Its performance is therefore identical to that of a servlet (JSPs *are* servlets). JSP's are limited to embedding Java within HTML and lose the structure that can be provided in a traditional Java program. JSPs might be more popular than servlets, however, since the JSP model of embedding programmatic code within HTML is easier for many programmers to understand.

We think several of these technologies should be used in programming assignments. If the students are not familiar with Java, any of the technologies in this category are equally suitable. What should be stressed is the similarity of approach between ASP, PHP, and ColdFusion and how CGI and servlets differ.

## Web Application Framework - CF, Struts, Spring, JSF, ASP.NET, RoR

In general, a framework goes beyond simple web content processing by supporting a wider variety of tasks and common activities used in web development. Many frameworks provide libraries for database access, templating, and session management. Most provide support for the model view controller architecture, but this isn't necessary to be classified as a framework.

With the previous server-side programming technologies, you navigate from one page to another through links and code embedded within the JSP (ASP, PHP, etc.) pages. A common problem with these technologies is that mixing the application logic with information presented in the browser makes the code difficult to maintain and doesn't clearly separate the roles of the web programmers and web page designers.

A model-view-controller approach, MVC, improves this situation. Most of the frameworks in this section support MVC. If you plan to use Struts, Spring, or JSF, consider adding additional work with servlets and JSP. A model-view-controller architecture similar to that used in Struts, Spring, and JSF can be implemented using a servlet for the controller, ordinary java classes (sometimes called plain old java objects, POJOs) for the model, and JSPs for the view. Lecturing on this approach and having the students complete a lab or assignment using this before learning about frameworks will help them understand and appreciate what the Java oriented frameworks are doing.

ColdFusion was an early framework a wide range of tools. It does not appear to directly support the MVC architecture, but a number of third party products claim to add this. It does have an integrated development environment, support for AJAX, .NET integration, JavaScript support, RSS support, and more. Although it's a proprietary product, in the past, they had programs for distributing educational versions to schools and might still be doing this. If so this is a potential technology for assignments.

Apache Struts is an open source framework that provides servlets to developers supporting the MVC architecture. A Struts servlet acts as a switchboard that routes requests from browsers to the appropriate programs and pages – you do not write your own custom servlet to do this. In addition, POJOs are associated with *actions* in the struts-config.xml file. The switchboard (a servlet provided by Struts named ActionServlet) uses the struts-config.xml file to determine where to send incoming requests.

In addition Struts provides a number of tag libraries that can be used within JSP pages to reduce or eliminate the server side code normally seen in the non-MVC style JSP pages described previously. You can also add your own custom tags that are linked to Java classes you've written.

The source code and binary distribution of Struts can be downloaded from <http://struts.apache.org/downloads.html>. The struts-*version*-lib.zip file is about 45

megabytes in size and contains the Struts jar libraries and many additional starter programs and examples.

Getting everything correctly configured for Struts is challenging. Although you could develop a Struts application from scratch this doesn't have to be done. Apache provides a number of zipped "war" files with typical starter configurations for Struts. These are in the apps folder when you unzip the *struts-version-lib.zip* file. If you use Eclipse for your IDE, these should be imported into a workspace (not an Eclipse project).

Next on the list is the Spring framework. It was developed at about the same time as the Struts framework. Its focus is to provide a framework for organizing POJOs. It can be used in a web context which makes it a possible web application framework. It can also be used conjunction with Struts. We have not looked at Spring, although it has many proponents. For now, we will add it to the list of interesting technologies.

ASP.NET was introduced after Struts and Spring. Although model code can be separated from view code into separate files, they are tightly linked. (Microsoft recently released a preview of a MVC toolkit for ASP.NET. So this may change in the future.) An extensive set of complex, but easily used, user interface components is provided along with an integrated event driven coding model. Additional components that render JavaScript user input validation are also provided. As a result web applications with user interfaces rivaling standard desktop applications can be quickly created.

Java server faces, JSF, was created after ASP.NET. Its method for using components to rapidly create web pages is reminiscent of ASP.NET. JSF, however, uses a model-view-controller approach. It adds a set of user interface components allowing developers to rapidly add complex features to web pages along with an event driven programming model. The basic web enabled version of Eclipse provides JSF support (without having to resort to additional plug-ins or MyEclipse). Since JSF was built with a component model in mind, it provides the capability to build components in a variety of technologies and in ways that are amenable to drag and drop graphic design. Struts has custom libraries that can be hooked into forms, however, it isn't amenable to JSF's style of drag and drop or JSF's component oriented event model.

Ruby on Rails, RoR, isn't in the J2EE domain since it relies on the object oriented Ruby language which was developed in the 1990s. It has many supporters in the J2EE camp however. It follows the MVC architecture (almost identically to JSF), it's free, and it claims up to 10 times the productivity of traditional J2EE frameworks using less code than most frameworks put into their XML configuration files. Rails offers starting code templates called scaffolding covering a large number of common database oriented web page tasks. Its approach is to add or modify coding only when needed for unconventional aspects of an application.

Two additional frameworks not yet mentioned are Apache Tapestry, and Apache Wicket. Tapestry began as a SourceForge project in 2000 and joined Apache in 2003. Tapestry isn't a standard like Struts or JSF; there is one implementation currently at



version 5. Tapestry supports a component model to web development similar to JSF's. Tapestry view pages, however, are XHTML with additional Tapestry attributes rather than JSP pages. Tapestry is potentially more efficient than JSF since it uses separate processes to handle form submission as opposed to rendering a page with no user inputs. Wicket was released in 2005 and can be characterized as a simpler, easier to learn version of Tapestry.

## **Web Application Deployment – JA, JWS, ClickOnce**

Java Applets inherit from `java.applet.Applet` or `java.applet.JApplet`. This allows them to be displayed from within a browser as long as the Java Runtime Engine, JRE, and a plug-in for the browser is installed. Applets are designed to be executed in a sandbox preventing them from accessing local data, which makes them safe to run from unfamiliar web sites. Applets were popular initially when most browsers were configured out of the box to run them. They fell out of favor in recent years when this capability was removed from most browsers. Running applets on most browsers now requires installing the JRE and additional software.

Java Web Start, JWS, is a newer technology designed to allow running standard Java applications by clicking a link in a browser. Unlike applets, JWS programs do not run within a browser. JWS provides a sandbox the applications run within; however unlike an applet sandbox, it can be configured to be less strict and allow access to the local computer's resources. JWS doesn't appear to be targeted towards the general Internet, rather more towards an organization's private intranet where it could significantly ease problems with distributing software to organization members.

ClickOnce is a Microsoft technology that is similar to JWS. It employs a code security model protecting client computers from malicious actions and also allows security to be configured to allow access to local resources on the computer.

All of these technologies should be covered in lecture. None of them are particularly difficult to use, however lab restrictions on how much students can do with security settings might make it difficult to make assignments with JWS and ClickOnce.

## Security

Digital signatures, DS, secure sockets layer, SSL, its successor, the transport layer security, TLS and digital certificates, DC are security related network protocols that should be covered at some point in the semester. Digital signatures typically use public/private key encryption. The private key is used to encrypt the signature and the public key can be used to verify the signature was encrypted by the user of the private key. SSL and the newer TLS, which is similar to SSL, are used to encrypt communications between a browser and web server. Both rely on digital certificates provided by certificate authorities. This then allows a public private key style of encryption between the server and client.

These topics belong in a Web programming class and are probably best worked into lectures later in the semester.

## Integrated Development Environments

The J2EE community can choose between many different integrated development environment, IDE, choices. BlueJ is a free stripped down Java development environment suitable for teaching entry level classes but probably not best for working on web oriented projects. Eclipse and NetBeans are free, open source, full featured IDEs suitable for large Java projects and also suitable for web oriented classes. WebSphere, JBuilder, and Idea are proprietary Java development environments whose proprietary nature and cost makes them less desirable than Eclipse and NetBeans.

For a yearly subscription of \$30 to \$50, MyEclipse puts together a best of breed selection of Eclipse plug-ins. We have not contacted MyEclipse to see if they allow free (or lower cost) academic use. It would be worthwhile to investigate this and to investigate other free plug-ins that support Struts if you plan to use Struts in the class. It's easy to convert any web project into a Struts, Spring, or JSF project using MyEclipse tools. In addition, configuring struts-config.xml becomes a graphic point and click operation instead of a file editing task.

In the Microsoft camp, choices are limited to Visual Studio or the unbundled and free Express versions of the development tools included in Visual Studio. Microsoft allows academic departments to join their Microsoft developers network academic alliance, MSDNAA, at a yearly cost of \$320 (first years cost \$499), or on a three year basis of \$799 (first three years \$1,025). With the MSDNAA faculty members and students have permission to install Visual Studio and other Microsoft products. Permission is also granted to install them in labs. An alternative is to use the free Express versions of the software; however when working on web applications, the full featured

Visual Studio available through the MSDNAA has features missing in Visual Web Developer Express Edition and is recommended.

Ruby on Rails is fairly new and therefore available IDEs are still in the early stages. A quick search turned up RadRails, RoRed, and the Ruby Development Tool plug-in for Eclipse. All of these are free and potential class IDEs. We haven't worked with any of them and can't make recommendations.

## Proposed Class Outline

After identifying major web oriented technologies we're ready to propose what we'd like to see in a web programming class covering the 16 weeks in a typical semester. This schedule assumes the students have completed two programming classes and have Java programming experience. If not, we recommend making week 4 an introduction to Java programming and moving everything following week 4 back a week.

ASP.NET will be a Microsoft technology used in one programming assignment. JSF is the current favorite in the J2EE world and is probably the best J2EE candidate for in depth coverage. Arguments can be made for in depth coverage of Struts, Ruby on Rails and any of the others. The important point is that some of the chosen framework(s) supports MVC and highlights the differences between it, ASP.NET and the other frameworks.

Prior to the first programming assignment, students should understand how HTML forms submit data via the GET or POST methods. In the first assignment, an initial page collecting user inputs will be submitted to the servlet (or if skipping servlets, other type of server-side technology) and something will be done with the user inputs. Perhaps store user inputs in a file or database (assuming JDBC has been introduced in an earlier class)? And finally a response will be generated and sent back to the user's browser.

The second assignment should use a different server-side technology (JSP in table 3) and illustrate passing values among pages using cookies or session variables. This is the time to introduce techniques for creating login pages, preventing page caching, and preventing access to pages requiring a login if not yet logged in.

The third assignment should focus on creating a MVC framework using a servlet for the controller, POJOs for the model, and JSPs for the view. This will be done without using an application framework like Struts or JSF.

The fourth assignment will use Struts, the fifth JSFs and the sixth ASP.NET. Each of these assignments will be structured to highlight what its technologies add to the previous server-side technologies. The advantages and disadvantages of Struts versus JSF versus ASP.NET will also be highlighted.

Week	Topics	Labs	Programs
1	Protocols: TCP/IP, HTTP		
2	Markup: HTML, CSS, XML	Static Web Pages	
3	Client-side programming		
4	JavaScript	JavaScript	HTML Form/Servlet/JavaScript
5	Server-Side Web Processing	Servlets	
6		JSPs	Multiple Page JSP Application
7	Web Servers	Server Configuration	
8	Web Application Frameworks		Controller, POJO, JSP Application
9	Struts	Struts	Struts Application
10	JSF	JSF	
11	JSF		JSF Application
12	ASP.NET	ASP.NET	
13	ASP.NET		
14			ASP.NET Application
15	Web Application Deployment	JWS/ClickOnce	
16	Web Security		

**Table 3 – Suggested Class Outline**

## Conclusion

Selecting the content for a web programming class is easier when available technologies are listed and then categorized. In addition to the eight categories we identify, technologies within many of these categories can be broadly categorized as those supported by Microsoft, those that are Java centric, and everything else.

The Java centric technologies are wide spread and attractive to computer science curriculums since most computer science students have taken programming classes using Java. In the "everything else" category, PHP, CGI/PERL, Ruby on Rails and more are all widely used. Any of these could be substituted for JSP, Struts, and JSF as proposed in the course outline.

We chose a mix of Microsoft and Java development environments for the programming assignments in this class. Our students are familiar with Java (they will have completed two Java based programming classes prior to taking this class) which frees us to concentrate more on web aspects of programming rather than learning new languages like PHP, or Ruby.

Most of our students have no experience with Microsoft's languages and style of programming coming in to this class. We include ASP.NET because Visual Studio and ASP.NET are major players in the programming world and we believe our students should be familiar with them. It helps that Microsoft's C# language has a lot in common with Java, allowing our students to quickly adjust to it. (We have verified this in .NET electives we occasionally offer).

Finally, technologies like AJAX or Flash might add some excitement to a web programming class. We haven't included them in the schedule, but are still considering ways this might be done. Both of these technologies provide ways to make pages more responsive and interactive and could add an element of fun into assignments.

We are hoping the output of the class will be students who have some proficiency in creating web applications and who understand the tradeoffs in choosing web programming technologies.

# **Transversal Homomorphism and Orthogonal within OR/MS/DS Tools into VB.NET 2005**

**Dr. Elias O. A. Tembe**

Associate Professor

Department of Computer Information Systems

University of Dubuque

2000 University Avenue, Dubuque, IA 52001

Tel: 563-589-3681

Email: [etembe@dbq.edu](mailto:etembe@dbq.edu).

41<sup>th</sup> Annual Midwest Instruction and Computing Symposium (MICS)

The University of Wisconsin-La Crosse (UW-L)

La Crosse, Wisconsin.

April 11<sup>th</sup>-12<sup>th</sup>, 2008.

**Title: “Transversal Homomorphism and Orthogonal within OR/MS/DS Tools into VB.NET 2005: Application of Advanced Database Using Collections, Generics and Algorithms”.**

**Abstract.**

BASIC programming language has gone through revolutionary reengineering process since 1963 as software for decision support systems (DSSs), management support systems (MSSs), intelligent decision support systems (IDSSs), and intelligent management support systems (IMSSs). The evolution has resulted to the creation of one of the most dynamic and popular visual programming languages in the 21<sup>st</sup> century known as Visual Basic.NET 2005. VB.NET 2005 is both window and web applications. VB.NET 2005 is object oriented (OO), object oriented design (OOD), object centered approach(OCA) and object event -driven (OED) language within the Visual Studio. NET. The following are some of the orthogonal characteristic of VB.NET 2005 that are also shared by all: OO, OOD, OCA and OED visual programming languages within the Visual Studio.NET: class, connectivity, navigability, inheritance (generalization), polymorphism (specialization, many forms), attribution, aggregation, encapsulation, abstraction, coupling, association, composition, relationship, cohesion, multiplicity, etc. It is important to note that all these qualities are inherited from operational research, management science and decision science (OR/MS/DS) tools. These qualities have significantly enriched the development of VB.NET 2005 as OR/MS/MS tool. OR/MS/DS approach incorporates many concepts from wide a range disciplines of study that are both quantitative (i. e, mathematics) and qualitative (linguistics especially natural human spoken languages such as English).

However, the transversal homomorphism and orthogonal programming features within and from OR/MS/DS tools into VB.NET 2005 are some of the most important variables contributing to the emergence the dynamo of VB.NET 2005 within the Visual Studio. The cross and hybrid pollination of features (i.e., symbols, functions, modules, structures, operators, data types, algorithms, iterations, constructs, loops) within and from OR/MS/DS tools into VB.NET 2005 has empowered VB.NET 2005 to compute from the simplest (analytic) to the most complex (simulated) mathematical problems for real world modeling.

The proposition is that anatomy of VB.NET 2005 with over 2000 controls (objects, boxes, controls images, pictures), that configurations(patterns) from OR/MS/DS tools, VB.NET 2005 has ability to provide superlative computing techniques in such areas as: simulation(last resort, experimentation), telecommuting, bioinformatics, artificial intelligence(AI) soft computing (SC), cyber security, audio, multimedia, information assurance, biometrics, forensics, telecommunication(remote sensing such as mobile or cell phone), teleconferencing, artificial intelligence,(AI), business intelligence(data mining, data warehousing, data mart), networking and data communications, soft computing (SC), pervasive computing, quantum computations (space computation), globalization (internalization), distributed computing, etc. The paper brings the attention that many intelligent (intelligence, smart) tools have

been fused into VB.NET 2005 since September 11, 2001. The tools have oxygenated the propensity of VB. NET 2005.

The paper, however, contends that superior computational abilities of VB.NET 2005 have not been fully liquidated. One of the areas VB.NET 2005 which has not been fully utilized for real world modeling is the creation of advance data base with application of mathematical structures such as trees (special graphs). The paper demonstrates how mathematical trees in conjunction with some of the most recent developments in computer science such as collections, generics and exceptional handling are synergistically interlocked within OR/MS/DS for the creation of the advanced database programming within VB.NET 2005. Collections in other visual programming language such as C++, C# and Java, are referred to as containers. Collections hold many functions (verbs, functions, modules), iterators (references or pointers, links), libraries or packages, properties (adjectives), and overloading (polymorphic) member functions. Collections and generics and exceptional handling are potentially dynamic (late binding, vectors) for sorting, searching, filtering, clustering, and recursion of data.

The conceptual framework integrates concepts within the OR/MS/DS tools which are quantitative and qualitative in nature. Philosophical issues which include moral, ethics, cognitive, and emotive are also addressed. The impact of phenomenological issues as part of OR/MS/DS into VB.NET 2005 is also discussed. The tools used for presentation include: (1) PowerPoint for presentation (2) VB.NET 2005 Version 7 for programming using of data base collections (3) Pentium 1V as the hardware.

The epitome of the abstract is that BASIC (beginners all symbolic instruction code) which was for only beginners has been cloned with features from many fields of OR/MS/DS, and has now become the software for all types of end users as for multipurpose, multiprocessing, multiprogramming, multithreading and multitasking functionalities for teaching , learning, training and conducting research .

**Key words:** Orthogonal, data mining, OOP, OOD, OED, IDSSs, DSSs, MSSs, IMSSs, Visual Studio.NET, transversal, relation, relationship, homomorphism, normalization, collection, generic, container, Iterator, tree, graph, composition, data structure, abstract data type, business intelligence, object centered approach, abstraction, abstract data, user defined data type, late binding, vector, module, phenomenology.

### ***Introduction:***

For nearly three and half decades, BASIC programming language has been evolving as one of the main programming languages that can cater for all types of end users in computing in problem solving and decision making process. At the present, BASIC continues to reengineered, metamorphosized and ignited (igneous) as VB.NET 2005 has become one the most powerful and popular visual programming languages on earth. The emergence of VB.NET 2005 as a tool with propensity to manipulate simple( analytic) and complex (simulations) computation from is due to convergence of many variables. The paper contends that there is a transversal homomorphism, isomorphism, homoemorphism, diffeomorphism and integrated crossover of OR/MS/DS(quantitative and qualitative )tools, approaches and methodologies such as: structures, functions(methods, verbs, subroutines, procedures, modules),



patterns, operators, linguistics (grammar: verbs, adjectives, conjunctions, punctuations marks) into VB.NET 2005 which were not embedded within BASIC.

### ***Statement of the Problem.***

The variables which have continued to contribute to the emergence of VB.NET 2005 as one of the most superior, popular and dynamic(vectored) visual, object-Event drive(OVD, object oriented design(OOD), object centered(OC), and object oriented programming (OOP) languages have not been fully examined and integrated. The merits of the transversal homomorphism, isomorphism, homeomorphism, and orthogonal features within and from OR/MS/DS tools into VB.NET 2005 within the Visual Studio.NET have not been fully liquated (realized).

### ***The Purpose of the Paper.***

The purpose of the paper is to present the variables which have made of VB.NET 2005 as the most superior, popular, dynamic, visual and event -driven programming application software. The paper further explains and demonstrates that there is dynamic transversal homomorphism and orthogonal processing system within and from OR/MS/DS tools, approaches, techniques and methodologies into VB.NET 2005 which continue to contribute to the power of VB.NET 2005. The paper also demonstrates how VB.NET 2005(version 7) can manipulate from the simplest to the most complex OR/MS/DS computations for real-world applications such as the creation of advance data base systems using a special a graphical topology (structure)referred to in this study as binary search and sort tree.

### ***Hypothesis.***

H<sub>0</sub>: The merits of the transversal homomorphism and Orthogonal within OR/MS/DS Tools into VB.NET 2005(version 7) for advance database systems have not been realized.

H<sub>1</sub>: The merits of the transversal homomorphism and Orthogonal within OR/MS/DS Tools into VB.NET 2005(version 7) for advance database systems have been realized

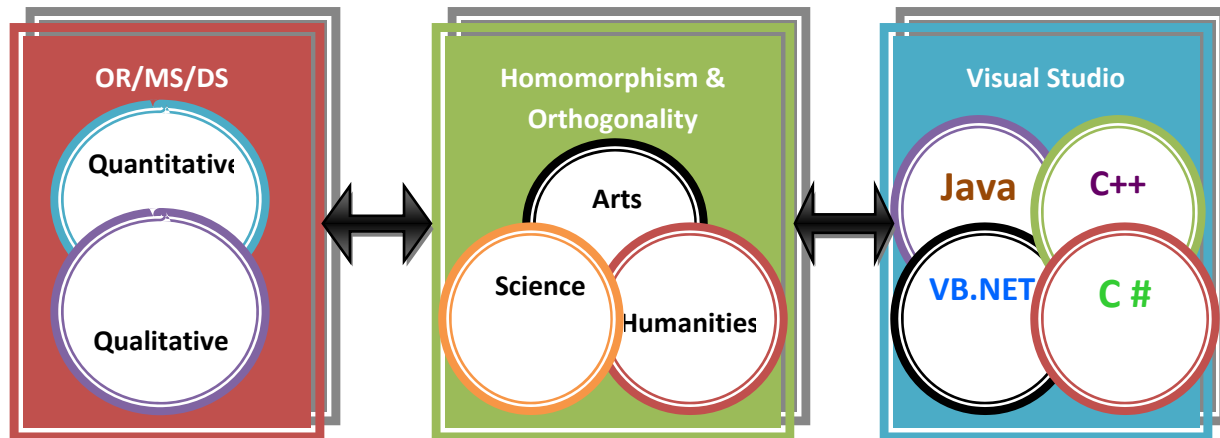
### ***Limitations of the Study.***

The paper is devoted and limited to the following: (1)literature review from primary and secondary sources (2 )OO, ODA, OCA, OCA, and EDA systems, (3) use of Pentium IV as the hardware, (4) advance object oriented relational database programming (OORDBMP) using VB.NET 2005 (Version 7) using binary search sort tree as a special OR/MS/DS/( mathematical graphical topology), (5) Philosophical issues (ethics, moral, cognitive, emotive) and phenomenological (lived experiences), economic, social, etc. which have impact in OR/MS/DS especially computer science database systems are also incorporated.

### ***Conceptual Framework.***

Many scholars such as (Dale 2003, Deitel and Deitel 2007, Malik and Sen 2004, Malik 2007) have written on visual programming languages (VPL), OOA, OOP, OOD, and OOC. However, there is **no** integrated conceptual framework that succinctly provides a holistic approach to show that there is transversal

homomorphism and orthogonal coexistence between and amongst OR/MS/DS tools, approaches, methodologies, and techniques into VB.NET 2005. The paper provides and recognizes: an enriched integrated, synergized, symbiotic a approach of OR/MS/DS tools, techniques and methodologies enriched VB.NET 2005. **Figure 1** shows the hybrid and cross pollinated conceptual framework ( schematic diagram ) used in this paper. The conceptual framework is dubbed as Integrated OR/MS/DS Transversal Homomorphism and Orthogonality into VB.NET (IORMSDSTMAOIVB.NET).



**Figure 1. IORMSDSTMAOIVB.NET Converged Approach**

**Assumptions.**

The assumption are that: (1) research is worthwhile (2) the conceptual framework can be utilized for programming for creating simple and complex computer programs (i.e. database, e.t.c) by many other types of end users and stakeholders (researchers, students, project managers, leaders) within the field of OR/MS/DS.

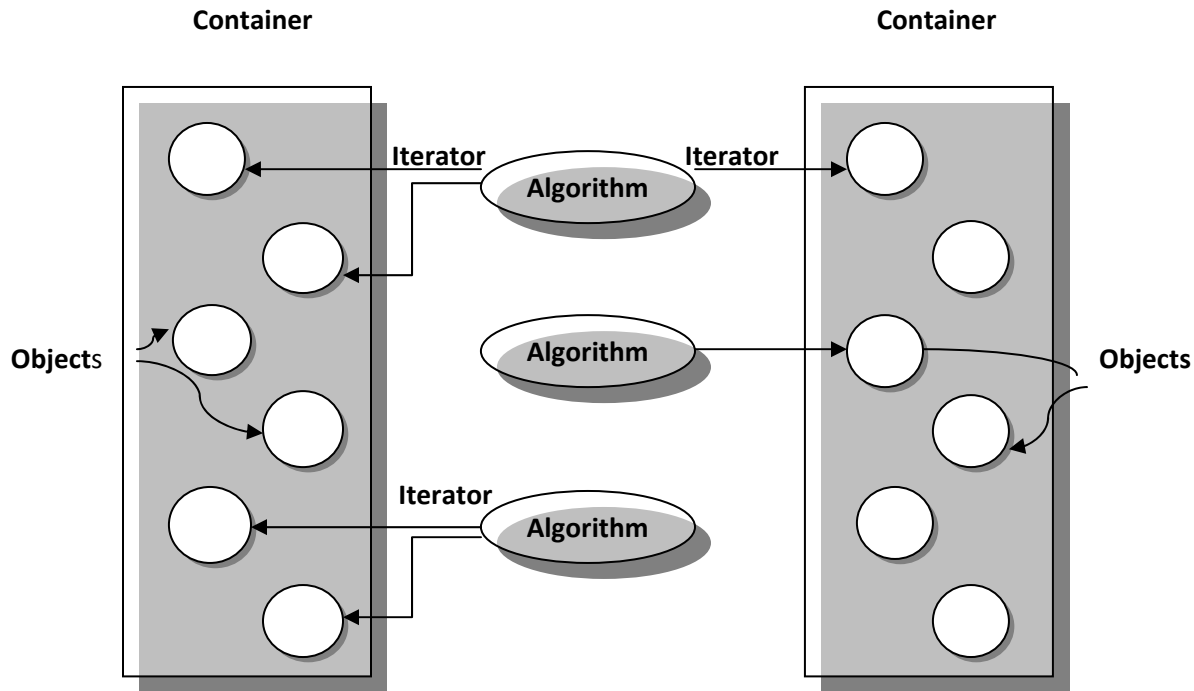
**Methodology.**

The study was conducted using literature review: (1) (primary and secondary sources, and (2) software. The presentation is into two phases: (1) Microsoft Office PowerPoint 2007 and (2) Visual Studio.NET: Visual Basic.NET 2005 (version 7) programming. Pentium IV is used as the main hardware. PowerPoint 2007 is used to provide explanation before the practical demonstration with VB.NET 2005(version 7).

**Programming and Experimentation with Visual Basic.NET 2005(Versio7)**

Visual Basic.NET 2005(version 7, although currently there is Version 8) is used to demonstrate and show a typical example of transversal homomorphism and orthogonal within OR/MS/DS tools, methodologies and techniques into Visual Studio.NET 2005. VB.NET 2005.The program provides an excellent mechanism for creating a library of collections of classes to handle storage and processing of data advance database with binary search and sort tree. The demonstration shows and includes the following key elements within the anatomy of program structure (code) for application of advanced database

using: (1) data structure (2) abstract data types (ADTS) or simply data abstraction (3) collections,(4) vectors (5) generics and (6) algorithms (7) comments or remarks (**see the definition of terms**). The VB.NET 2005(Version 7) program code is however, encapsulated (hidden). The code is however, exposed and explained during the presentation.



**Figure 2 Containers (Collections), algorithms, and iterators**

**Source:** Robert Lafore. Object –Oriented Programming in C++. Fourth Edition, 2005

**Data structure** refers to the process by which data or information is stored in the computer memory. In C++, collections are known as containers within the Standard Template Library (STL) while in other visual programming languages: VB.NET, C# and Java, the containers are referred to as collections. Collections are ways data is stores in computer and organized in the memory of the computer which include member functions for specific tasks. The type of collection used in this paper for demonstration belongs to type of collection referred to associative collection. Associative collections are not sequential; instead they utilize keys (primary keys) to access data. Typically keys are digits or string (words)which are used to automatically and dynamically sort, search, filter and cluster the stored elements in a specified organized order. In computer lingo, **Generics** are general (generalization) and dynamic concepts using one method (function) for many variables. **The print method ( )** is a typical example of generics. **The term vector** is OR/MS/DS especially in mathematics and computer science that refers to dynamic arrays of arrays (late binding). A Vector shrinks and expands automatically while the program is executing. Vectors with random access iterators can be plugged into any algorithm. **Algorithms** are sets (arrays of arrays) of dynamic (vectors) and late binding instructions, methods, functions (methods) procedures that are applied to containers to process data in various ways such as the use of sort ( ), copy ( ), search ( ) for data for solving problem and making decision (**see Figure 2**). In this paper algorithms

are used as non member stand alone methods used for both the non-discrete (analog) and discrete mathematical (digital) computations. Iterators as used in this paper are like (cables, links, connectivity, connections, wires) can be considered as smart pointers, intelligent, intellisense , points to, points at in order to connect or references on data or information “entry” in the collections. *Algorithms use iterators to act on objects in containers (collections).* Iterators determine which algorithms can be used with which collections and why it is necessary to do so such as efficiency of the algorithms Iterators also. Iterators are elegant match appropriate algorithm and collections. Iterators can be also considered as algorithms. Examples of iterators algorithms include: find ( ), input( ) reverse( ) , print( ) , merge( ) forward( ) , begin( ) , random( ) , output( ) , read line( ) , write line( ) , copy ( ) . It is important to point out that VB.NET does not have pointers like C++ programming language but it has methods and algorithms which act like pointers within the program code.

It is important to point out that the demonstration using VB.NET 2005 is the central purpose and theme of the approach to show the transversal homomorphism and orthogonality of OR/MS/DS tools into VB.NET

### ***Research Questions.***

The paper addresses the following questions: (1) What impact has transversal homomorphism and Orthogonal played within OR/MS/DS tools into VB. NET 2005? (2) What are some real-world application of advanced database using collections, generics , iterators, vectors, and algorithms, and (3) What is the future of VB.NET 2005 and other versions to be created?

### ***Literature Review:***

The real-world modeling applications using object oriented tools is documented as old as ancient civilization in the Indus valley, Mesopotamia (Tigris and Euphrates), Egypt (the Nile Valley), Yang-test-Chinese Civilization, Greeks, Romans, Mayas, Incas, Maoris of New Zealand, the Aborigines of Australia, Babylonians, Phoenicians, Medes, American Navajo Indians and other . The Egyptians developed great pyramids based on integrated incorporated OR/MS/DS tools. The Egyptian writings were hieroglyphics (picture writing, icons, graphs, charts, trees) using various tools(objects). In India the Alora and Ajanta caves with old magnificent with pictures (images, icons) and many mathematical architectural designs(OR/MS/DS tools). American Navajo developed coding systems called "Code Talkers" which was used during the World War II for security. The Code Talkers is an important real-world applications within database or information for encryption security (secrete communication, security, cryptology). Russell and Taylor (2007) comment that “history is full of amazing production feats –the pyramids of Egypt, the Great Wall of China, the roads and aqueducts of Rome. Malik and Sen 2004 point out those mathematical concepts which are discussed at length in programming recursion “represents work that is under the way in the temple of Brahma. At the creation of the universe, priests in the temple of Brahma were supposedly given three diamond pegs, with one peg containing 64 golden disks “. It is also authenticated from theological and biblical literature that the children of Israel wanted a god they could touch (Bible: Revised Standard Version).

Figure 3 shows some examples of ancient civilization tools which have been integrated into programming hieroglyphics.



Pyramids of Giza

Egyptian Hieroglyphs

Indian Temple

1	∩	11	∩ ∩	21	∩ ∩ ∩	31	∩ ∩ ∩ ∩	41	∩ ∩ ∩ ∩ ∩	51	∩ ∩ ∩ ∩ ∩ ∩
2	∩∩	12	∩ ∩∩	22	∩ ∩∩∩	32	∩ ∩∩∩∩	42	∩ ∩∩∩∩∩	52	∩ ∩∩∩∩∩∩
3	∩∩∩	13	∩ ∩∩∩	23	∩ ∩∩∩∩	33	∩ ∩∩∩∩∩	43	∩ ∩∩∩∩∩∩	53	∩ ∩∩∩∩∩∩∩
4	∩∩∩∩	14	∩ ∩∩∩∩	24	∩ ∩∩∩∩∩	34	∩ ∩∩∩∩∩∩	44	∩ ∩∩∩∩∩∩∩	54	∩ ∩∩∩∩∩∩∩∩
5	∩∩∩∩∩	15	∩ ∩∩∩∩∩	25	∩ ∩∩∩∩∩∩	35	∩ ∩∩∩∩∩∩∩	45	∩ ∩∩∩∩∩∩∩∩	55	∩ ∩∩∩∩∩∩∩∩∩
6	∩∩∩∩∩∩	16	∩ ∩∩∩∩∩∩	26	∩ ∩∩∩∩∩∩∩	36	∩ ∩∩∩∩∩∩∩∩	46	∩ ∩∩∩∩∩∩∩∩∩	56	∩ ∩∩∩∩∩∩∩∩∩∩
7	∩∩∩∩∩∩∩	17	∩ ∩∩∩∩∩∩∩	27	∩ ∩∩∩∩∩∩∩∩	37	∩ ∩∩∩∩∩∩∩∩∩	47	∩ ∩∩∩∩∩∩∩∩∩∩	57	∩ ∩∩∩∩∩∩∩∩∩∩∩
8	∩∩∩∩∩∩∩∩	18	∩ ∩∩∩∩∩∩∩∩	28	∩ ∩∩∩∩∩∩∩∩∩	38	∩ ∩∩∩∩∩∩∩∩∩∩	48	∩ ∩∩∩∩∩∩∩∩∩∩∩	58	∩ ∩∩∩∩∩∩∩∩∩∩∩∩
9	∩∩∩∩∩∩∩∩∩	19	∩ ∩∩∩∩∩∩∩∩∩	29	∩ ∩∩∩∩∩∩∩∩∩∩	39	∩ ∩∩∩∩∩∩∩∩∩∩∩	49	∩ ∩∩∩∩∩∩∩∩∩∩∩∩	59	∩ ∩∩∩∩∩∩∩∩∩∩∩∩∩
10	∩	20	∩∩	30	∩∩∩	40	∩∩∩∩	50	∩∩∩∩∩		

### Babylonian Alphabet

Figure 3 Ancient Object Oriented Writings.

The objects used for many years have been translated to real-world English application and some have been imported and embedded into computer programming. Visual Basic.NET is one of the visual programming languages which have been enriched by the transversal homomorphism of objects from OR/MS/DS tools and techniques for real-world application such as trees: binary search and sort trees for advance data base applications.

It can be deduced that mankind: (1) is object oriented by nature from time immemorial, (2) loves, appreciates and is motivated to work with tools that are visualize and touchable(tangible), (3) attracted to images, icons, pictures, shapes, cards (game programming, Sudoku) , buttons, graphs, trees, circles, oval, notations, symbols, volumes, dimensions, metrics, color, patterns, paints, and drawing, lines, curves, tables( mathematical relations for row and columns), figures, structures, boxes , etc , such the over 2000 tools which have been embedded into Vb.NET 2005 from OR/MS/DS tool, (4) appreciates objects are aesthetic (I. e, specially spiritual), inspiring, in nature and intrinsic appealing (intrinsically, intangible), (5) uses objects as tools relational for amusement and entertainment, and (6) innovates by

using objects. The creation of VB.NET 2005 is the integrated innovation involving broad transversal homomorphism and orthogonality within OR/MS/DS tools, methods, approaches and techniques.

It also interesting to note that there is trend in Europe for mankind to build pyramids to be cemeteries for the burial of the dead like the Egyptians did. This kind of is plan(strategic, tactical, operational), philosophical(spiritual, emotive and cognitive) feeling and phenomenological though is intertwined with the integration and utilization of OR/MS/DS tools including the creation of computer science systems, hence tranveral homomorphism and orthogonality of knowledge.

### ***Justification for Using VB.NET 2005***

There are many the main variables and justifications contributing to the strong attributes of VB.NET 2005 powerful software of the 21<sup>st</sup> century within Visual Studio.NET: hence its selections as application software for database using binary search graph(tree). The following are the variables and the justifications for selecting and strength of Visual Basic.NET 2005 as tool for with OOEDRDBMSs binary search and sort tree: (1) attractive and colorful (2) appealing (3) visual (4) easy to recognize and identify (5) eye catcher (6) simplicity for programming (coding), (7) statements (clauses, functions, verbs, modules) from one of the most powerful transform –oriented and declarative database software: Structures Query Language(SQL)are embedded within VB.NET 2005. **SQL** contains statements and clauses such as SELECT, WHERE, UPDATE, INSERT, DELETE, etc. which are useful OOEDRDBMSs. (8) more visual than most software because it contains with over 2000 tools such as controls (boxes, buttons images, picture, shape, lines, etc) for programming (9) user friendly and easy to program, (10) dynamic or late binding (11) any type of end user (all beginners) can learn it (13) it is currently being developed orthogonally by computer gurus to be at the same level with other visual programming languages. (14)homomorphism of functions, operators into VB.NET 2005 from OR/MS/DS: quantitative disciplines, namely: algebra, calculus, discrete mathematics, numerical analysis, and geometry such as: graphs, predefined mathematical functions( i.e., cosine, sine, tangent, floor, ceilings ), matrices, determinants , partially ordered sets (posets), congruence, summation, recursion, permutation, combinations, forecasting techniques, counting techniques, strings(words), relations, Big-O and theta notations, Venn diagrams, ordered pair and Cartesian product, cardinality, functions ( i.e, co-domain, target, injection, subjection )sets, sequences, series, vectors, paths, circuits , logic(negation, conjunction, disjunction, implication, bi-implication), constructs and loops(i.e, invariants), automata, probability and types, conics, rational, logarithmic and exponential functions, equations and inequalities. Bridging gap.(15)special functions and cardinalities such as inverse, restrictions, extensions, image, and pre-image, equivalent(equipotent), countable (16) mathematical domains such as discrete mathematics which is a form an important is an important and integral part of computer science. It has rich collection of structures which play an important part in programming visual Studio.NET hence VB.NET 2005 (17)counting principles such as union, intersection and Cartesian product of sets, Pascal’s triangle (18) RSA cryptosystem named after Ronald Rivest, Adi Shamir and Leonard Adelman). (19)Sorting and searching, filtering and clustering algorithms (20) Characteristics of object oriented concept : composition, inheritance(generalization), polymorphism (specialization), coupling, cohesion, aggression, attribution, association, relationship, encapsulation(hiding the information or data), abstraction (Malik and Sen 2004, Deitel and Deitel 2006), (21) Homomorphism of functions from qualitative disciplines

especially the English language in terms of terms of parts of speech (verbs, modules, subroutines) such as main( ). The qualitative verbs, adjectives (properties) include all the sorting and searching and algorithms used for database .(22) VB.NET 2005 has been cemented as a strong programming language that can handle simple and complex computations.(23) support of institutions of higher learning (colleges and universities) by incorporating discrete mathematics and data structures and algorithms into the curriculum within the field of computer science and other scientific related courses and topics and subtopics, (24) Support of software, network and telecommunications, data communication, hardware and procedure specialists and all types of end users( stakeholders), (25) computer globalization or internationalization( 26) social and economic disparity between the have and have-nots( the have-nots are looking for a software application system to meet the financial limitations). (27) contains within it over 2000 objects (controls, boxes, buttons) which accommodate OR/MS/DS: simple(analytic) and complex(simulation) and (28) The database systems developed by VB.NET 2005(version 7) meet the standard measuring litmus test and **12 Rules** of relational database specified by Edgar Codd and OOEDRDBMSs specified by Date's Twelve Rules for a DBMSs (Connolly and Begg, 2005, Rob and Caronel, 2009). In summary, VB.NET 2005 is easy to create: develop, analyze, design (i.e, reengineer), implement (code), test, and maintain and understand.

### ***Philosophical and Phenomenological Issues and Justifications for the Strength of VB.NET 2005.***

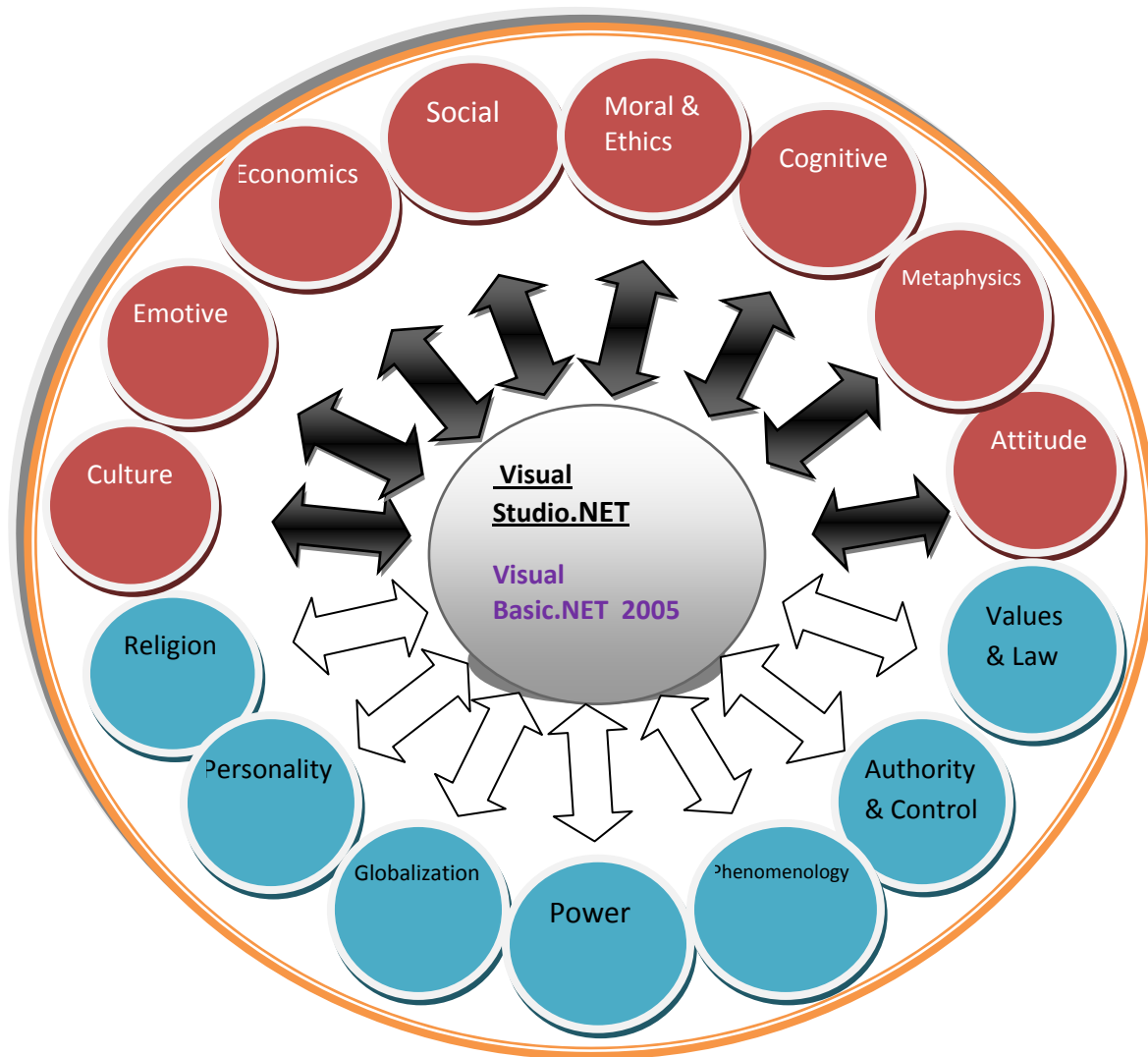
There are six elements of computer elements: software, hardware, data, networks and telecommunications, procedures, and people (peopleware). Most of the emphasis and attention concentrates on software, hardware, data, network and telecommunications, and procedures. Tembe (2006, Parker and Chase 1993), however, point out the most neglected and often forgotten (not given much or serious attention) computer element is the peopleware. Parker and Case (1993) confirms "the people component of management information systems is overlooked or underestimated". The paper supports that philosophical issues which include moral, ethics, cognitive, emotive, supernaturalism, metaphysics are interlocked with creation of software such as VB.NET 2005. **Figure 3** shows the interrelationship of the philosophical and phenomological issues (lived experiences) which are intertwined in the creating of software such as VB.NET 2005.

The rationale for including philosophical and phenomenological issues is because many outstanding and distinguished scholars in discrete and non discrete mathematics have included the life histories (autobiography) of pioneers in the fields of OR/MS/DS (i.e; mathematics and computer science)autobiography and found out that their academic resumes include studies and preparations in: ethics, moral study, philosophy, music, moral science, metaphysics, anthropology, history, theology, religion , and others, Malik and Sen 2004, Rosen 2007, Knuth 1998). It is further interesting to bring out that some of distinguished scholars in quantitative OR/MS/DS were deacons, priests and ministers of the gospel of churches. Some of these fields play a significant role in computing soft computing(SC) and artificial intelligence (AI), forensics, computer security and information assurance, encryption (i.e; RSA). Rosen (2007) cites the famous mathematician prodigy Srinivasa Ramanujan that" he considered mathematics and religion to be linked...an equation for me has no meaning unless it expresses a thought of God". As lover of music and pianist, the distinguished and exceptionally talented scholar (guru) Donald Knuth views computer science as both an arts and science. Phenomenology is defined in this

paper as lived experience approach which advocates that ideal objects(natural )such as numbers as conscious life (cultural world) enable mankind to find the know. Many scholars confirm that phenomenology has had impact on quantitative and qualitative approaches of many OR/MS/DS fields of study especially: mathematics, medicine, biology, chemistry, nursing, law, political science, architecture, philosophy(ethics, moral, mathematics, religion, philosophy, natural science, ecology, theology, economics, sociology, anthropology, physics, computer science(technology), gender education, culture, ethnicity, literature(linguistics), geography, psychology, dance, sports, music, education, theatre and film as early as mid 1890.The paper contends that phenomenological view points have contribute to the traversal homomorphism and orthogonality of OR/MS/DS tools for the creation and powerful functionality of VB.NET 2005. Some of the recently integrated and emerging areas in computer science such as bioinformatics, computer and ethics, cognitive computing and programming, SC, AI, computer forensics(FC) and many others can be considered as the by products of the concept of phenomenology.

The integration of ethics and moral play an important part into real world situations (i.e, curriculum, government control). For instance, The USA Congress, has enacted laws pertaining to ethical code of conduct such as Sarbanese- Oxley 2002 and USA Congress Ethics Act 2007. Such acts are control the unfair (moral, unethical), socially, economic, cultural irresponsible using mathematical and programming (software) and hardware (technological computation for business transactions elements of computer: software, hardware, people, data, procedures and documentation.





**Figure 3 Integrated Philosophical and Phenomenological Issues into VB.NET 2005**

The creation of virus by an end user to destroy for instance database systems is certainly dependant on how the end user ethically or morally views the side effects of the virus.

The paper argues that economic, social, geopolitical, internationalization (globalization), moral science, cultural, theological and religious, etc. are intertwined with creation of programming due to the role played by the various view points and concepts from disciplines. Synergistic dependency, interdependency and intra-dependent within OR/MS/DS tools help in with the creation of software (software development life cycle (SDLC) or also called program development life cycle (PDLC) such as VB.NET 2005 within Visual Studio.NET. In summary, the issues can be termed as external and internal generators.

***Justification for the Selection of Database Application.***

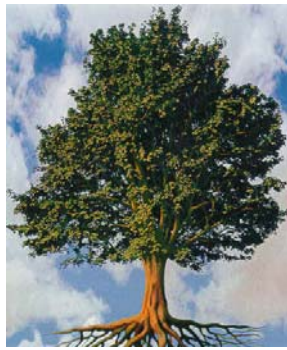
Many prolific writers and scholars such as (Shelly, et. al. 2006] indicate clearly that many experts “estimate that eighty percent (80%) of the world’s computer application are database applications”.

Although in this paper, there is no research to support this high outlier. The paper asserts that almost every one using computer system is involved in database application(s) consequently the percentage should be higher than 80 percent. In this paper database is broadly defined as a tool for storing, recording, retrieving, accessing, securing and protecting data or information using sorting, searching, filtering, collections, generics, clustering, iterate, algorithms methods(functions) and techniques. Database and information systems are expressed as collection of file of libraries (cabinets, banks, drawers, file cabinet).

### ***Binary search and Sort Tree and Justification for Database Systems with VB.NET 2005***

This section of the paper is devoted to the justification for selecting binary search and sort. Connolly and Begg (2005) assert that a tree pattern(topology) is utilized as data structure for data or information or indexes for much database management system (DBMSs). Though there are many types of trees such as B<sup>+</sup> - trees, AA trees, etc, used in computer science for real-world modeling for data base, in this paper however, binary search and sort tree has been selected and utilized as the main candidate tool. The main justification of using binary search and sort tree include: (1) an excellent tool for nonnormalization within the database with VB.NET 2005, (2) suitability for sorting and searching, clustering, filtering, thinning using techniques and algorithms(functions, modules) for simple and complex computations which advance data base systems (3) wholly embedded with algorithms from the OR/MD/DS: qualitative and well as quantitative techniques play a significant part in simple and complex database systems other computer related areas such as networking, software engineering, web programming, computer security, pervasive computing, (4) facilities duplicate elimination( no redundancy allowed at the node by intrinsically comparing, for instance, the OR/MS/DS methods from qualitative discipline (linguistics) such as: the **insert ( )** method is has the propensity and responsibility to eradicate any data that attempts to be duplicated by comparing similar values,(5) can be used as conceptual framework to **swap ( ), merge ( ), list ( ) sort ( ), compare ( ), heapfy ( ), reverse ( ),** (6) much faster and most efficient than other searching techniques such as sequential. (7) suitable for fast sorting **Savitch (2004) confirms that sorting is** the most widely encountered programming tasks, and certainly the most thoroughly studied".

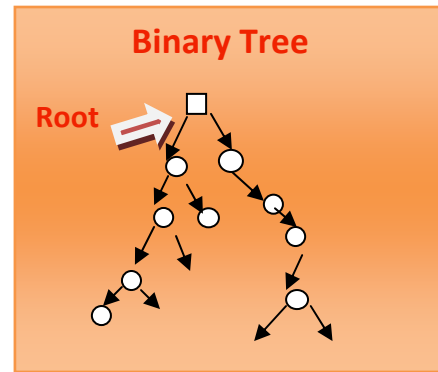
The diagram in **Figure 4** is an example that shows a pictorial (visual, object, icon, image) representation of binary search and sort tree. In compute science, the root node is the special node of the binary search and sort tree, and is drawn at the top instead of bottom up. The tree has specific two main routing links by using arrows (iterators, connectivity): (1) **lLink (left rotation)** for left as the starting from the node, (2) **rLink(right rotation)** for right connectivity and navigability. The operation starts from the left then to right. The left to right decomposition and linking is in conformity to the how the computer computes and manipulate data (left to right algorithm).



Top bottom



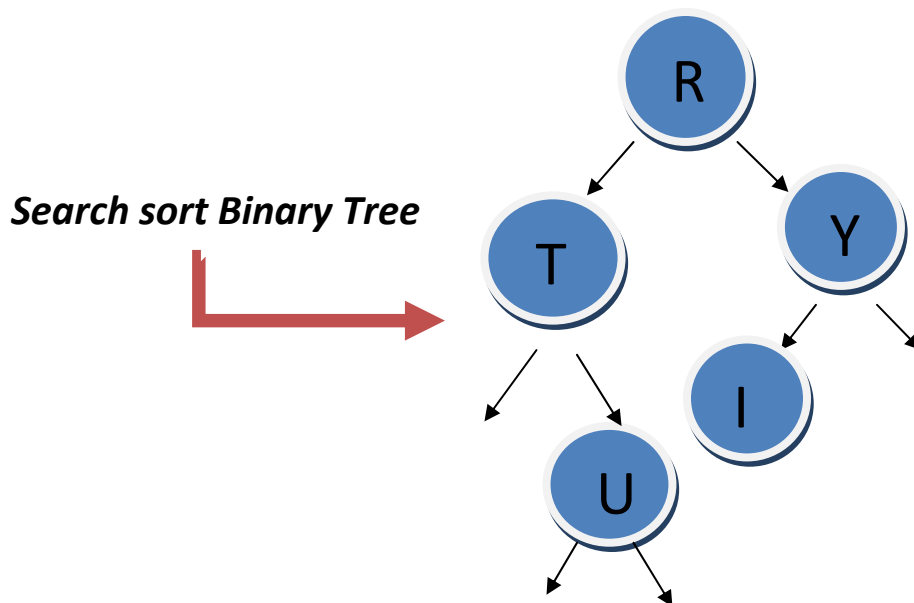
bottom up



Bottom up Actual modeling

**Figure 4** *Binary search and Sort Trees*

In summary, the binary search and sort tree is a typical example of a dynamic graph which is applicable for OOEDRDBMSs. The most profound advantage of binary search sort and tree is that it plays an important role the normalization of database system (data redundancy) which is the crux of the matter in creating a real-world model data base application. In this paper, it is referred to as binary search sort tree, instead of the usual binary search tree. The justification is that within the Visual programming language(s)code, search and sort are used combined as word search sort( ) algorithms.



**Figure 5** provides an elaborate structure how a binary Search Sort binary tree is decomposed to depict a typical logical sequence. Generally the most usual approach to apply binary search sort tree is to traverse or to visit every node of the binary search sort tree. The traversal must start at the root node because the reference of the root node. There are three main recursive algorithms utilized recursive :

inorder sequence, preorder, and post order sequence. In **Figure 5** shows the recursive sequence of binary search sort tree:

**Inorder Sequence : T U T Y I**

**Preorder Sequence : R T Y U I**

**Post order Sequence : U T I Y R**

It is important to note that before the implementation of binary search sort tree as ADT for database system, the following sequential algorithms in a form of protocol have to be followed: (1) determining to find out if the binary tree is empty (2) searching the binary search and sort tree for a specific entity, (3) inserting the entity in the binary, (4) deleting or removing an item from the binary search sort tree, (5) finding the height of the binary search sort tree, (6) finding the number of nodes in the binary search sort tree (7) finding the number of leaves in the binary tree (8) traversing the binary search sort tree (9) copying the binary search sort tree.

### ***Findings:***

The following are the findings: (1) There is cross pollination of objects (notations, symbols, characters, strings, modules, structures, patterns, objects) from OR/M/DS tools and techniques into VB.NET 2005 which continue to make it popular, (2) There is high demand of VB.NET 2005 by many types of users (3) user friendly, less costly, graphical, and OOD, OOPL, OOCA, (4) support innovative (theoretical and practical experimentation) of VB.NET 2005 by institutions of higher learning as a tool for teaching, learning, and training has made it stronger, (5) Good programming in VB. NET 2005 embraces: mental, moral, ethical, health, security and safety, legal, physical, affective, emotive, cognitive, psycho-motor, and spiritual domains for the success of the students in the real world now and future, (6) economic, social, geopolitical, internationalization (globalization), moral science, cultural, theological and religious, etc are interlocked, dependent, intra-dependent and interdependent with the creation of VB.NET 2005. Succinctly the issues are interrelated are synergistic, dependent, interdependent and intra-dependent with and for the creation of VB.NET 2005, for the success of student(s) now and in the future for learning, research and teaching, (7) The convergence, symbiotic dependency, understanding and application of mathematics contribute significantly to the critical success factors of students today and in future for problem solving and decision making process in real world application using VB.NET 2005, (8) experts estimate that over 80% of the world's computer application are database application; therefore many types of end users globally continue to use VB.NET 2005 database application, (9) lack of using integrated objects (boxes, images, icons, pictures, buttons, colors) such the ones used in VB.NET 2005 has the pull and inertia to discourage many learners to successfully enjoy programming, (11) OR/DS/MS tools and techniques are the determinant that are significantly important for the understanding of the dynamic VB.NET 2005 and other VB.NET 2005 (versions), (14) almost all fields of study are now inevitably dependent on the use of mathematics and the use of technology computer science, informatics) and computational functionalities and supporting devices (peripherals), (15) philosophical issues such as ethics, moral, cognitive, emotive, metaphysics, moral science, etc; and how and where they are interlocked within **Datalogy** (computing devices, computer science, informatics) and

(16)mankind loves to model by using real-world objects (icons, images, pictures). In VB.NET 2005(version 7) there are over 2000 objects intrinsic and extrinsic objects (buttons, controls, boxes, images, pictures which) within VB.NET 2005whichare attractive for programming. (17) phenomenology plays an important role in the creation of computer science systems especially database programming languages.

### ***Recommendations:***

End users intending to be majors and minors in computer science and related areas of study should understand how OR/MS/DS(quantitative and qualitative) tools, techniques and methodologies are homomorphism and orthogonal into programming language. An understanding of OR/MS/DS tools and techniques such as discrete mathematics(basic and advance), data structures, data abstraction, linear mathematics, calculus, numerical analysis are important for creation of advance database systems with Vb.NET 2005 because even the understanding of Oracle 10g is dependant on OR/MS/DS tools such as discrete mathematics properties, calculus, relational algebra and Java program a sibling of VB.NET 2005 within Visual Studio.NET .

### ***Summary and Future Prospect of VB.NET 2005.***

The transversal homomorphism and orthogonality within and from OR/MS/DS tools into VB.NET 2005 will continue to be re-engineered. VB.NET 2005 will be recognized as one of the visual software with “tsunami power” amongst the visual programming languages within Visual Studio.NET. The propensity of VB.NET 2005 to manipulate simple and complex data and information for real-world applications in many such areas as: telecommunication(remote sensing, mobile or cell phone, mobile entertainment), multimedia and hypermedia (sound, music, video), networking, web development, quantum computing, teleconferencing, artificial intelligence(AI), soft computing(SC), forensics and biometrics, bioinformatics, bioethics, pervasive computing, security ad information assurance (secret homomorphism, secret sharing of data as the RSA), self healing programming within self care computers, nano-technology, supply chain such as radio frequency identification(RDFI), software engineering, business intelligence (data mining, data mart, data warehousing), will continue to be realized (liquidated). VB.NET 2005 will and continue to have impact on: learning, teaching and research for problem solving and decision making process ranging from simple to complex manipulations and computations. VB.NET 2005 will dominate: multiprocessing, multithreading, on-line processing, multiprogramming, and multitasking processing systems. VB.NET is no longer be limited in some areas OR/MS/DS computing and non computing areas.

### ***Definition of Terms:***

The following are terms, concepts and their definitions as used in this paper.

***Aggression*** is term used to describe the concept of an object being composed of other objects

**Cohesion.** The term cohesion used for OOD application refers the class member very obviously and clearly belongs to that class and not possibly to some other related class. Cohesion in reality is one of the concepts of transversal homomorphism and orthogonality.

**Coupling.** As used within OOD, and Unified modeling language(UML), coupling is a term that refers to the number of relationships one class has with another class. Coupling in reality is one of the concepts of transversal homomorphism and orthogonality .

**Datalogy.** Datalogy is integrated computer science or data processing approach in which creators (all types of end users) associated with creation (development, analysis, design, implementation and maintenance ) of computer systems have to consider using holistic approaches such as the by the transversal homomorphism and orthogonality within OR/MS/DS into the software(i.e., VB.NET 2005) coined in this paper as Transversal Homomorphism and Orthogonality within OR/MS/DS Integration (**THAO OR/MS/DS I**). (see the conceptual framework, Figure 1).

**Diffeomorphism.** In mathematic diffeomorphism applies to mapping (linking, bundling) amongst diverse inverse. It helps to bundle (blend) . However, in computer science, isomorphism has real-world applications in relational object oriented database systems. Typical examples include Many-to-many relationship, database mapping, quantum computing useful for physics for space exploration such as NASA, computer security, database security, Artificial intelligence (AI), soft computing (SC).

**Isomorphism.** Isomorphism is a mathematical function (method, module, bijection) that links sets or groups of objects (entities) with similar (equal) attributes(state, qualities and properties, shape). However, in computer science, isomorphism has real-world applications in relational object oriented database systems. Typical examples include One-to-One relationship, database mapping. Isomorphism is another name for automorphism which is an object that automatically links it self to another but still retains it all identify (attributes, properties and state).

**Homomorphism.** Homomorphism mathematical concept associated with an application where one similar array are embedded, linked and mapped into another such in the form into Venn diagram where characters have relationship, composition, encapsulation, polymorphism, inheritance. Homomorphism I is most applicable when handling coordinates, vectors spaces, groups, and rings that have similarities of patterns OR configurations. In real world application. Homomorphism has application to many-to-many relationship, one-to-one relationship and one-to-many relationship database systems which are also OOERDBMSs. Homomorphism includes OR/MS/DS: mathematical concepts such as automorphism, isomorphism, homeomorphism and diffeomorphism.

**Homeomorphism.** In mathematics, homeomorphism is bijection with link that is continuous at the same time, the inverse is also continuous. Homeomorphism is applicable in many areas of computer science such as: networking, relational database (the emphasis of the paper). Homeomorphism is apart of *homomorphism*

**Orthogonal.** Orthogonal is mathematical term and as used in this paper , it is a structure with portions that are extended, shared and integrated with another element at right angles but in straight lines perpendicularly. In geometrical mathematics, the term orthogonal is most applicable when handling coordinates and vectors.

**Transversal.** Transversal is a mathematical concept used for real-world geometrical applications using a line or many lines (objects, drawings) to link, connect , cross, intersect two or more coplanar lines to help create homogeneous solution(s).

## References:

Anderson, R. David, Sweeney, J. Dennis, Williams A. Thomas and Martin, Kipp. [2008]. An Introduction to Management Science: Quantitative Approaches to Decision Making. Thomson .South-Western. Mason. Ohio.

Bible: Revised Standard Version.

Burden, L. Richard and Faires, J. Douglas. [2001]. Numerical Analysis. Seventh Edition. Brooks/Cole: Thomson Learning.Pacific Grove, California. USA.

Connolly, Thomas and Begg, Carolyn. [2005]. Database Systems: A Practical Approach to Design , Implementation, and Management. Fourth Edition. Addison Wesley. London.

Curwin, Jon and Slater, Roger. [1996].Quantitative Methods for Decision. Thomson Press. Fourth Edition.

Bradley, Case Julia and Millspaugh, C. Anita. [2007]. Advanced Programming Using Visual Basic 2005.McGraw-Hill. Dubuque.

Bradley, Case Julia and Millspaugh, C. Anita. [2006]. Programming in Visual Basic. Visual Basic .NET 2005 Edition. McGraw-Hill. Dubuque, IA.

Dale, Nell. [2003]. C++ Plus Data Structures. Third Editon. Jones and Barlett Publishers. Boston.

Deitel, H. M and Deitel, P. J.[2006]. Visual Basic 2005: How to Program. Upper Saddle River. New Jersey.

Knuth, E. Donald [1997, 1997]. The Art of Compute Programming The Art of Computer programmong. Volumes 1-2. Addison-Wesley Professional. New York.

Donald, Knuth. [1998]. Volume 3: Sorting and Searching (2nd Edition), 1998. Addison-Wesley Professional. New York.

Droezdek, Adam. [2005]. Data Structures and Algorithms in Java. Thomson:Course Technology. Boston. Massuchettes.

Gaither, Norman and Frazier, Greg. [2002]. Operations Management. South-Western:Thomson Learning. Cincinnati. Ohio.

Gaddis,Tony and Irvine, Kip. [2009]. Starting Out with Visual Basic. Fourth Edition Pearson:Addison Wesley. New York.

Hiller, S. Frederick, Hiller, S. Mark and Lieberman, J. Gerald. [2007]. Introduction to Management Science: A Modeling and Case Studies Approach with Spread Sheets Irwin Mc-Hill. Dubuque, Iowa.

Malik, D.S and Sen, M.K.[2004]. Discrete Mathematical Structures:Theory and Applications. Thomson. Course Technology.



Lafore, Robert. [2005]. Object Oriented Programming in C++. Fourth Edition. SAMS. Indianapolis. Indiana.

Larson , Ron and Hostetle, Robert. [2004]. College Algebra. Sixth Edition. Houghton Mifflin Company. Boston.

Larson , Ron, Hostetle, Robert and Falvo, C. David. [2007]. Algebra and Trigonometry. Seventh Edition. Houghton Mifflin Company. Boston.

Larson , Ron, Hostetle, Robert and Edwards, H. Bruce and Heyd, E David. [2006]. Calculus: With Analytic Geometry. Eighth Edition. Houghton Mifflin Company. Boston

Parker, Peter and Chase, Thomas. [1993]. Management Information Systems: Strategy and Action. Second Edition. McGraw Hill. New York.

Reeve, S. Carl and Reeve, M. J and Fees, E. Philip. [ 2005]. Accounting 22e. Thomson. South-Western. Mason, Ohio.

Ricardo, Henry. [2003]. A Modern Introduction to Differential Equations. Houghton Mifflin Company. Boston.

Rob, Peter and coronel, Carlos. [2009]. Database Systems: Design, Implementation, and Management. Eighth Edition. Thomson : Course Technology. Boston. Massachusetts.

Rosen, H. Kenneth. [2007]. Discrete Mathematics and Its Applications. Sixth Edition. McGraw Hill. Dubuque. Iowa.

Russell, S. Roberta and Taylor, III. W. Bernard. [2007]. Operations Management. Upper Saddle River, New Jersey.

Savitch, Walter. [2007]. Problem Solving with C++: The Object of Programming . Sixth Edition. Pearson: Addison Wesley. New York.

Shelly, B. Gary, J. Thomas , Cashman and Starks, L. Joy. [2006]. Java Programming : Introduction Concepts and Techniques. Third Edition. Thomson. Course Technology. Boston, MA.

Shelly, B. Gary and Vermaat, E. Misty. [2008]. Discovering Computers. Course Technology: Cengage Learning. Boston . Massachusetts.

Slack, M. James. [2000]. Programming and Problem Solving with Java. Brooks/Cole. Pacific Grove. California.



# **PID Control in a Real-Time Embedded Systems Programming Course**

**Joseph Clifton**  
**Computer Science and Software Engineering**  
**University of Wisconsin – Platteville**  
**Platteville, WI 53818**  
[clifton@uwplatt.edu](mailto:clifton@uwplatt.edu)

## **Abstract**

The Computer Science and Software Engineering department at the University of Wisconsin – Platteville offers a course entitled Real-Time Embedded Systems Programming. The focus is on the development of software for real-time embedded systems; however, we also provide considerable hands-on experience with the hardware. Example topics covered include microprocessors, languages, environments, real-time operating systems, simulators, emulators, timers, counters, digital I/O, ADCs, interrupts, reliability, fault tolerance, concurrency, process scheduling, synchronization, and communication.

About four years ago, members of our Advisory Board suggested that we add some control theory to our Real-Time Embedded Systems Programming course. From an electrical and hardware engineering perspective, control theory is typically covered in a semester-long course requiring several electrical engineering courses as prerequisites. For us to take such an approach was clearly infeasible. So instead, we focused on the digital implementation of the Proportional-Integral-Derivative (PID) controller. This paper discusses how we integrated PID control into the course.

# 1. Background

Most computers exist not on the desktop, but embedded in other devices. The computers in embedded systems can vary from tiny microcontrollers with a small amount of programming, as found in “low-end” toasters, to big computers running millions of lines of code, as found in large switching systems. As such, the issues in development of software for embedded systems vary greatly. Computer scientists, computer engineers, electrical engineers, and software engineers all have a somewhat different view as to what constitutes a course in real-time embedded systems. Such courses in electrical and computer engineering departments usually emphasize the hardware aspects of embedded systems. Software development is at best a secondary concern and usually done in assembly language. On the other hand, computer science departments that offer such a course tend to focus on the real-time theoretical aspects, in many cases to the exclusion of hardware. [2]

## 1.1 Real-Time Embedded Systems Programming Course

The Computer Science and Software Engineering department at the University of Wisconsin – Platteville offers a course entitled Real-Time Embedded Systems Programming. The name is somewhat of a compromise since it is specifically meant to convey the fact that the emphasis is on software development and not on hardware development. The name is somewhat deceiving since much more than just "programming" is covered. The course is taught as a senior-level course. The prerequisites are:

- CS/SE 2630 - Object-Oriented Programming and Data Structures II,
- CS/SE 3430 - Object-Oriented Analysis and Design, and either
- EE 3780 - Introduction to Microprocessors, or
- CS 3230 - Computer Architecture and Operating Systems

Typically, students have completed several other Software Engineering and Computer Science courses. The students are exposed to a wide range of topics associated with real-time embedded software development. A significant laboratory experience gives hands-on exposure to the topics. The students are expected to use analysis, design, implementation, and testing techniques learned in previous courses.

## 1.2 Platforms

One aspect of the course that distinguishes it from other such courses is the "evolution" approach to the platforms on which the students are expected to develop. There are five laboratory projects and one final project. Students start by using a very small PIC microcontroller (1K ROM, 64 bytes RAM) and programming in its assembly language. Next, they are given a PIC-C compiler, a bigger PIC (8K ROM, 368 bytes RAM) and

significantly more challenging programs to develop. The students spend about seven weeks with the PIC, doing four laboratory projects. After that, they switch to an 8051-based microcontroller with 32K ROM and 32K RAM. They are given a C-based development environment with a source-level debugger, including a hardware simulator and target monitor. They are required to do a large-sized multi-tasking program using a tiny Real-Time Operating System (RTOS) as their last laboratory project. For the final project, the students use an XScale processor running the Windows CE operating system and the .NET Compact Framework. It has 64 MB RAM, 64 MB ROM, a touch-screen, Ethernet, USB, and plenty of I/O. The students do a real-time UML design using Rational Rose and develop in C# using Microsoft Visual Studio .NET. Some sample final projects include an ATM, RFID cash register, milking parlor control system, sign language interpreter, automatic bowling score keeping system, and sales and inventory tracking system. There are multiple delivery dates for the last laboratory project and the final project, and these dates are interspersed over the last eight weeks of the semester.

### 1.3 Topics Covered

The topics covered in the course come from several areas. They roughly fall into three (somewhat arbitrary) categories: Overview, Practical, and Theoretical. [2]

The “overview” topics are those that generally include a listing of instances, brief discussions, and comparisons and contrasts. The students get laboratory experience only in a small number of the instances of a given topic. For example, the students will only use four different processors. The course gives an overview of:

- Real-time and embedded system examples
- Special considerations required for embedded systems development
- Processors and systems
- Development languages
- Development environments
- Platforms and platform standards
- Real-time operating systems, real-time kernels, and tasking shells

The “practical” topics are those for which an overview is also given; however, the students get more comprehensive laboratory experience. The practical topics generally include hands-on experience and considerable experience with the hardware. It should be noted, however, that this is not done at an electrical or computer engineering level. The practical topics include:

- Debugging tools, such as simulators, emulators, ROM monitors, and target debuggers
- Program loading: RAM, ROM, EEPROM, flash
- Timers, counters, interrupts, watch dog
- I/O: memory-mapped, port, DMA

- Device drivers, UARTs, SPIs, PPIs
- Interfacing and communications

Theoretical topics are those that are primarily software in nature and are generally hardware-independent. These topics allow traditional computer science material to be applied to the unique problems of embedded systems software development. With the exception of the last one, such topics are those that can be found in a text like [1]. The theoretical topics include:

- Reliability, fault tolerance, exception handling
- Concurrent programming, tasks, threads
- Synchronization and communication
- Process and resource scheduling
- Resource control: semaphores, monitors
- Device and inter-processor communications: buses, networks
- Real-time UML and object-oriented design of embedded systems

## **1.4 Software Engineering Principles**

Software engineering principles are stressed throughout the course. Software quality notions such as survivability, safety, and fault tolerance are discussed in lecture. Tools and environments, including tool analysis, selection, and operation are covered. For the laboratory component, students work in small groups for three of the projects. Some projects require the students to turn in requirements and design. Students are expected to adhere to good modularity principles in all the languages that they use. Coding standards are enforced. Methodical testing of functions, modules, methods, and objects is expected. Note that given the use of multiple platforms, one measure of how well the students have applied these principles is how easily their designs port to the next platform. Students are required to produce project plans, test documents, use version control, and log time spent on group projects.

## **2. Control Theory**

About four years ago, members of our Advisory Board suggested that we add some control theory to our Real-Time Embedded Systems Programming course. Our initial reaction was that our course is already very challenging and there is not room to cover another topic. The topics listed in Section 1.3 tend to fill the entire semester. However, the advisory board members represent companies that hire our graduates, so it was important that we attempted to cover the topic somehow.

From an electrical and hardware engineering perspective, control theory is typically covered in a semester-long course requiring several electrical engineering courses as prerequisites. For us to take such an approach was clearly infeasible. So instead, we

focused on the digital implementation of the Proportional-Integral-Derivative (PID) controller.

## 2.1 Control Theory Coverage

The PID control material discussed in the following section is covered in one lecture period. After that, the students are expected to apply it to the last laboratory project, which is implemented on an 8051-based microcontroller with a small RTOS. This project was a natural place to incorporate the PID control topic. The students choose their final project, so requiring it to be part of that project would not guarantee a thorough experience. The PIC controller's limited memory implies those laboratory projects are necessarily limited in scope. Typically, PID control is only part of a given application, so incorporating it into the laboratory project that uses multitasking is a good fit.

## 2.2 Proportional-Integral-Derivative Control Theory

A PID controller takes a measured value and a desired value as inputs and produces a correction value as an output. It uses the instantaneous error, which is the difference between the desired value and the measured value. For example, if the desired voltage is 3.5V and the measured voltage is 4.0V, then the instantaneous error is -0.5V. The value to be controlled can be almost anything, such as temperature, pressure, current, revolutions per minute, gallons per second, etc.

The correction value is applied to the controlling device. For example, the correction value may be the Pulse-Width Modulation (PWM) output that controls a motor or the amount to turn a valve to slow the flow of a liquid.

The PID control algorithm determines the correction amount using the sum of one or more of three calculations, assuming execution of the control loop at fixed intervals:

- Proportional – constant times instantaneous error
- Integral – constant times sum of instantaneous errors
- Derivative – constant times the change in error

A PID control loop is generally set up to cycle at precise time intervals so that time calculations for the integral and derivative terms can be incorporated into the PID constants.

Coming up with the constants is referred to as “Tuning the PID loop”. “The nice thing about tuning a PID controller is that you don't need to have a good understanding of formal control theory to do a fairly good job of it. About 90% of the closed-loop controller applications in the world do very well indeed with a controller that is only tuned fairly well.” [3]

### 2.2.1 Proportional

The proportional term is the easiest of the three PID terms. It is simply a constant times the instantaneous error. It is most likely what one would attempt to use if they had no knowledge of control theory. In fact, for some systems, using only a proportional term will produce reasonable results. However, the convergence from the actual value to the desired value often suffers from a couple of problems if only a proportional term is used. If the constant chosen is too small, it can take a long time to achieve the desired value. If the constant is chosen too large, there is a tendency to overshoot the desired value. A strong overshoot that continues to travel back and forth is known as ringing. [3]

### 2.2.2 Integral

The integration term is a constant times the sum of the instantaneous errors. Technically, because it is an integral, each sum factor should be multiplied by the time interval over which the current measurement and the previous measurement took place. In practice, the PID control loop is driven at a constant frequency so that the delta time factor is a constant and can be rolled up into the integration term constant.

The integration term is responsible for smoothing out the error over time. It can be thought of as the ‘history’ of the error. For example, suppose that the actual value is below the desired value. The longer the actual value stays below the desired value, the larger the integration term grows. The integration term starts to decrease once the actual value overshoots the desired value. However, since the integration term is still positive once this happens, the tendency of the proportional term to drive the system to an overshoot on the other side and/or a ringing state is diminished.

In practice, the integration term can often grow to very large positive or negative values. Therefore, the integration term is typically restricted to a range that is generally symmetric about zero, that is, it is restricted to a maximum absolute value. So besides coming up with an appropriate integration term constant, use of an integration term also requires the choice of limiter constants.

The integral term is rarely used by itself and is usually used in conjunction with the proportional term. A PID controller that only uses the proportional and integral terms is appropriately referred to as a PI controller. Reference [3] makes the following claim about the time sample accuracy needed for a PI controller:

“Because the integrator tends to smooth things out over the long term you can get away with a somewhat uneven sampling rate, but it needs to average out to a constant value. At worst, your sampling rate should vary by no more than 20% over any 10-sample interval. You can even get away with missing a few samples as long as your average sample rate stays within bounds. Nonetheless, for a PI



controller I prefer to have a system where each sample falls within 1% to 5% of the correct sample time, and a long-term average rate that is right on the button.”

### 2.2.3 Derivative

The derivative term is a constant times the change in error. Technically, because it is a derivative, the term should be divided by the time interval over which the current measurement and previous measurements took place. In practice, the PID control loop is driven at a constant frequency so that the delta time factor is a constant and can be rolled up into the derivative term constant.

The derivative term gives the instantaneous rate of change of error. It can be thought of as the “slope” of the error function, that is, as an estimate of future error. As the actual value initially moves towards the desired value, the proportional and integral terms will have the same sign but the derivative term will have the opposite sign. For example, if the current error is 10 and the previous error is 20, the derivative term will be -10 times a constant. This helps lessen the overshoot that can occur from the other two terms. Note that if the desired value is fixed, then the change in error becomes the difference between the previous actual value and the current actual value, and in this case, the derivative term is how fast the actual value is changing, that is, a prediction of the next actual value:

$$(\text{Desired} - \text{Current}) - (\text{Desired} - \text{Previous}) = \text{Previous} - \text{Current}$$

The derivative term tends to be the most difficult of the three. As stated in reference [3]:

“Differential control is very powerful, but it is also the most problematic of the control types presented here. The three problems that you are most likely going to experience are sampling irregularities, noise, and high frequency oscillations... When you use differential control you need to pay close attention to even sampling. I'd say that you want the sampling interval to be consistent to within 1% of the total at all times-the closer the better.”

## 2.3 Proportional-Integral-Derivative Control Algorithm

Assuming that the PID loop is executed at a constant rate, the PID control algorithm is surprisingly simple:

```
currentError = desiredValue - currentValue
pTerm = Kp * currentError
iSum = iSum + currentError
if iSum > iSumMax then
```

```

    iSum = iSumMax
elseif iSum < iSumMin then
    iSum = iSumMin
iTerm = Ki * iSum
dTerm = Kd * (currentError - previousError)
previousError = currentError
controlOutput = pTerm + iTerm + dTerm

```

The developer is responsible for choosing the three PID constants  $K_p$ ,  $K_i$ , and  $K_d$ , and the integration limiters  $iSumMax$  and  $iSumMin$ .

### 3. Project Description

As stated in Section 2.1, PID control was incorporated into the project that uses the 8051-based microcontroller and a small RTOS. The author chose to use a simulated analog device to control. This gave a cheap and easy way to introduce PID control. What follows is part of the description of the project given in the spring of 2007.

Implement a feedback loop to control a simulated analog device. The output of the device is an analog signal in the range of 0 to 4 volts. The output voltage is controlled via four discrete inputs (connected to 8051 outputs) whose functions are:

1. Decrease the voltage relatively quickly
2. Increase the voltage relatively quickly
3. Decrease the voltage slowly
4. Increase the voltage slowly

Writing a one activates an input, writing a zero deactivates it. For the "relatively quickly" inputs, the longer a one is written, the faster the rate of increase or decrease. The result of writing ones to multiple inputs is unspecified.

The feedback loop must drive the current output reading close to the desired output reading in a "timely" manner. When the current output reaches the desired output, it must be relatively stable and cannot bounce around much. You must use a PID loop as discussed in class.

You must have an LCD display as follows:

```

column  0123456789012345
        R:r.rrr  NNNN
        D:r.rrr

```

The actual reading (from the ADC) is on the first row. It is displayed as a floating-point number to three decimal places, even though that accuracy cannot be achieved. The 8051

boards have 10-bit A/Ds. I have connected the reference ground and reference voltage to 0 and 5 volts, respectively. The value to display is:  $5.0 * \text{ADC\_value} / 1024.0$

The NNNN field is the integer value sent over a serial channel. It is only for display. It *cannot* be used in any calculations or the PID loop. The desired reading is displayed on the second row, again as a floating-point number to three decimal places. It is based on keypad inputs from the user corresponding to the following keys:

1. Increase desired by 1.0
2. Increase desired by 0.1
3. Increase desired by 0.01
- A. Increase desired by 0.001
4. Decrease desired by 1.0
5. Decrease desired by 0.1
6. Decrease desired by 0.01
- B. Decrease desired by 0.001

Ignore the rest of the keys. Be sure to cap all desired values between 0.0 and 4.0. Update the display after each key press.

The serial data from the PIC is sent at 9600 baud, 8N1, not inverted. The target debugger uses the serial port on the 8051 board, so you must write a software serial driver that implements a handshake with the simulated device. When it is time to read and display serial data, enable external interrupt 0 (which is Port 3, pin 2) and then put a one on the serial enable pin (Port 3, pin 3) which will signal the PIC to send the serial data. Reset the serial enable to zero after the first bit arrives. When the interrupt for the start bit occurs, disable external Interrupt 0 and then enable Timer1 and its interrupt to read in the bits. Each bit occupies 104 microseconds. Sample approximately in the middle of each bit. The 8051 is clocked at 10 KHz, but it takes six clocks for each instruction cycle. Therefore, each instruction cycle is 0.6 microseconds. Each tick of Timer 1 is one instruction cycle.

The serial port data occupies two bytes, representing one 12-bit unsigned number. The format is:

first byte: 10xxxxxx, where xxxxxx are the low 6 bits of the data  
second byte: 01xxxxxx, where xxxxxx are the high 6 bits of the data

The bits are sent little endian (low bit first). Not counting the first start bit, you need to read 18 bits: 8 bits for the low byte, a stop bit, a start bit, and 8 bits for the high byte. If the high two bits of either byte are not correct, toss the data and display 9999 on the LCD. You must account for the possibility that the serial input wire could be broken.

You will be using the C515C boards, RIO MP boards and Keil uVision2 C. You must use RTX 51 Tiny. Besides the main task, you must have at least four other tasks that:

1. Reads the voltage from the ADC - used in PID calculations and for display
2. Gets serial data and displays it, and also displays the ADC voltage reading
3. Handles reading keys and updating desired voltage
4. PID Control - attempts to drive actual voltage to desired voltage

Note that you will need semaphores to guard the LCD, current reading, and desired reading.

## 4. Conclusion

The laboratory project has been a good way to introduce the students to some control theory. By using a simulated device, the author can change the characteristics of the device as well as the control inputs, assuring that the control constants do not migrate from semester to semester. The group that produces the best results is awarded a few extra points, which helps assure that the constants do not migrate between groups.

One improvement that could be made to the coverage of PID theory is that different methods of tuning could be discussed. Currently, the students are given little guidance on this beyond the suggestion to use trial and error, and to consider the output “units” to help judge the size of the coefficients. If the author chooses to cover tuning techniques in the spring of 2008, they will be discussed at the presentation.

## References

- [1] Burns, A. and Wellings, A, *Real-Time Systems and Programming Languages*, Addison-Wesley, 2001.
- [2] Clifton, J., "A CS/SE Approach to a Real-Time Embedded Systems Software Development Course", SIGSCE Bulletin, Volume 33 Number 1, March 2001, pp. 278-281.
- [3] Wescott, T., “PID Without a PhD”, *Embedded Systems Programming*, October 2000.

# **Simulation of & Development of a Range Control Center Information Display System For UAS Operations in North Dakota**

Ron Marsh, Ph.D.  
Steve Buettner  
Kirk Ogaard  
Computer Science Department,  
University of North Dakota  
Grand Forks ND 58203  
rmarsh@cs.und.edu  
steven.buettner@und.nodak.edu  
kirk.ogaard@und.nodak.edu

John Nordlie  
Regional Weather Information  
Center,  
University of North Dakota  
Grand Forks ND 58203  
nordlie@rwic.und.edu

## **Abstract**

The University of North Dakota (UND), in cooperation with the Federal Aviation Administration (FAA), is developing airspace within the state of North Dakota where unmanned air systems (UASs) can be tested/operated without the need for an on-board sense and avoid system or temporary flight restrictions (TFRs). The Computer Science Department has been tasked with developing a majority of the system software, including an airspace simulation and the range control center (RCC). Microsoft's Flight Simulator X was selected as a cost-effective solution for the airspace simulation, while OpenGL was selected for development of the display system. Since the Air Force wants UND to explore "new" ideas for information display systems (IDS), replication of an existing ATC system was deemed unsuitable. Thus, the UND RCC IDS is being designed from "scratch."

This paper provides an overview of the proposed system and the requirements and design of the RCC IDS.

# 1. Introduction

The University of North Dakota, in cooperation with the Federal Aviation Administration, is identifying airspace within the state of North Dakota where organizations interested in developing UASs can test/operate their systems without the need for an on-board sense and avoid system. Taking advantage of a relatively low population density, UND and the state of North Dakota are working to provide more than 13,000 square miles of airspace suitable for all manner of UAS operations without the need for implementation of temporary flight restrictions (TFRs).

Unmanned aerial systems (UASs) offer a unique range of features. Their small size and weight create lower manufacturing and operating costs; this also allows them to stay aloft for long periods of time. With no pilot on board UASs can be used in dangerous situations or for very routine and mundane operations. While the military applications of UASs are obvious, there are also many commercial applications that are being explored as well. In September of 2001 a proof of concept flight of a Pathfinder-Plus UAS over a plantation in Kauai, Hawaii was done to collect digital imagery indicating the ripeness of coffee beans [1]. In another proof of concept application a UAS was used to fly over the five thousand acre San Bernabe vineyard in Monterey County, California. The UAS was equipped with an infrared imaging system that was able to determine frost conditions in the vineyard and report them to a control station [2].

However, flying UASs in National Airspace can be problematic as current FAA regulations are not applicable to unmanned aircraft [3]. Therefore operations are limited to a lengthy Certificate of Authorization process that requires the UAS pilot to reliably see-and-avoid other air traffic in the area of operation [4]. There is also the issue of safety that UAS operators must contend with. A prior analysis of ground and midair collision risk found that most UAS operations would not meet FAA target levels of safety without the incorporation of a mitigation strategy. Mitigation techniques being considered include operating restrictions, mission scheduling, fault detection and accommodation, path planning and execution elements with a focus on emergency scenarios; including and not limited to collision avoidance and forced landings [5].

The John D. Odegard School of Aerospace Sciences, with funding from the United States Air Force UAV Battle Lab, is developing a ground-based ganged phased array radar system capable of detecting low observable aircraft such as sailplanes and hot-air balloons and is developing the software to optimally display the information to range controllers and operators of UASs. While previously available ground-based radar systems have not been sufficient for the Federal Aviation Administration to approve their use for airspace de-confliction mitigation, this system will employ new technology that will enable UAS operators to see potential conflicts before they become a problem and safely maneuver their craft away from non-cooperative aircraft. Sophisticated algorithms are being developed to determine optimum scan patterns, rates and data assimilation to provide the most comprehensive "picture" of the operating environment.

Funding of the program will allow UND to develop a comprehensive system that

incorporates all available data into the big picture. Ultimately, UND hopes to provide an interim mitigation strategy to allow UAS research and development outside restricted airspace thus aiding the Federal Aviation Administration in its efforts to develop appropriate regulations relating to UAS operations and certification.

## **2. Background**

There are many challenges to developing a comprehensive system for managing the airspace. The biggest challenge is that the ground-based radars are not yet in-place, yet we must forge ahead with the development of the RCC IDS and related system software. Thus, we have turned to software to simulate the expected environment. Several software simulations have been developed, including a radar simulation, a network simulation, and an airspace simulation. Of interest here is the airspace simulation. The airspace simulation must model the flight characteristics of the UASs, the airspace, and manned air traffic. The cost of developing such a simulation package must also be considered. We investigated several UAV/UAS simulation packages for potential applicability and found that Microsoft's Flight Simulator X (FSX) was the most cost-effective solution for the airspace simulation for our application, while OpenGL was selected for development of the RCC IDS.

### **2.1 UND Risk Mitigation Architecture**

The prototype architecture for the UND risk mitigation system is shown in figure 1. The core of the system is the three ganged phased array radars. Conceptual plans are to locate the radars in Park River, Nekoma, and Lakota ND, forming a triangular shaped airspace as shown in figure 2. Data from the radars and an ADS-B receiver would be fused and forwarded to the RCC (location TBD). Data from a weather station located at the UAS operations airport and Doppler weather radar would also be forwarded to the RCC.

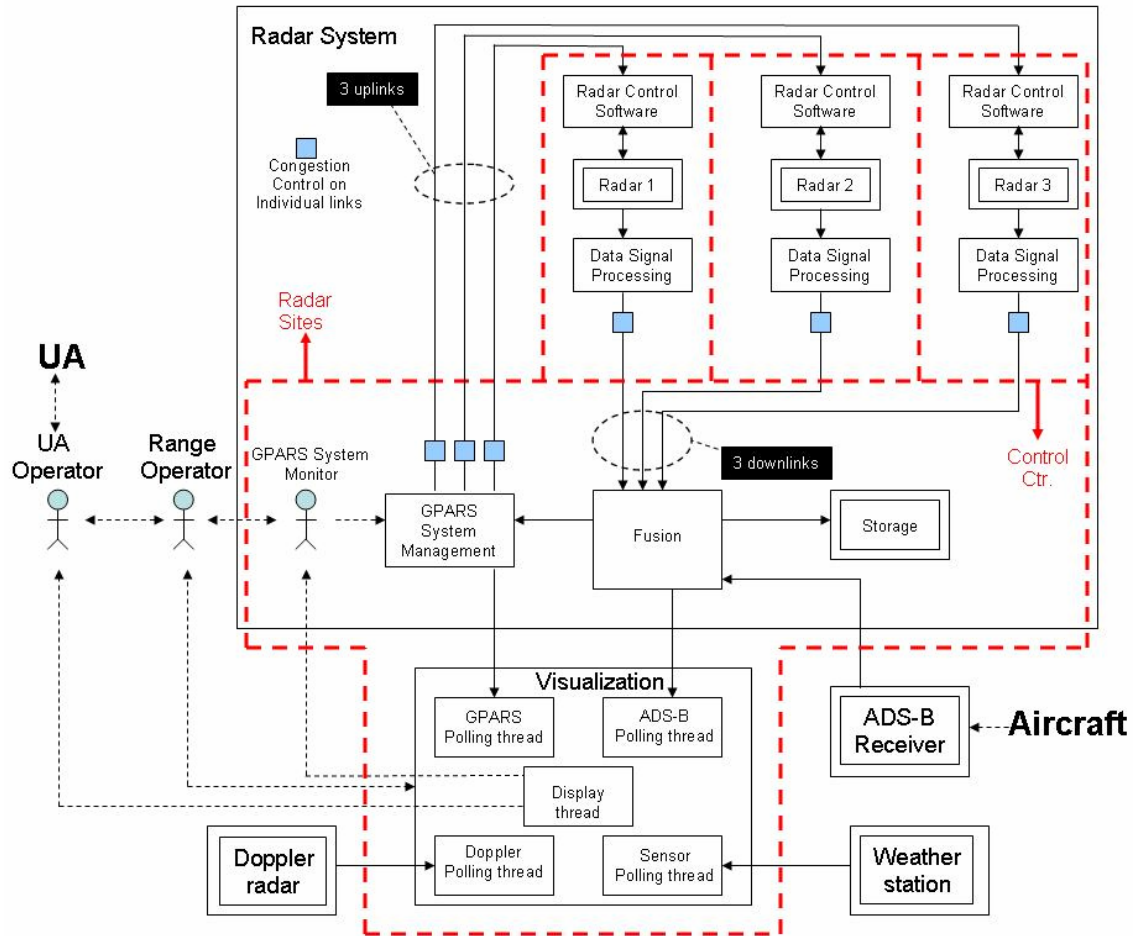


Figure 1. UND risk mitigation prototype architecture.



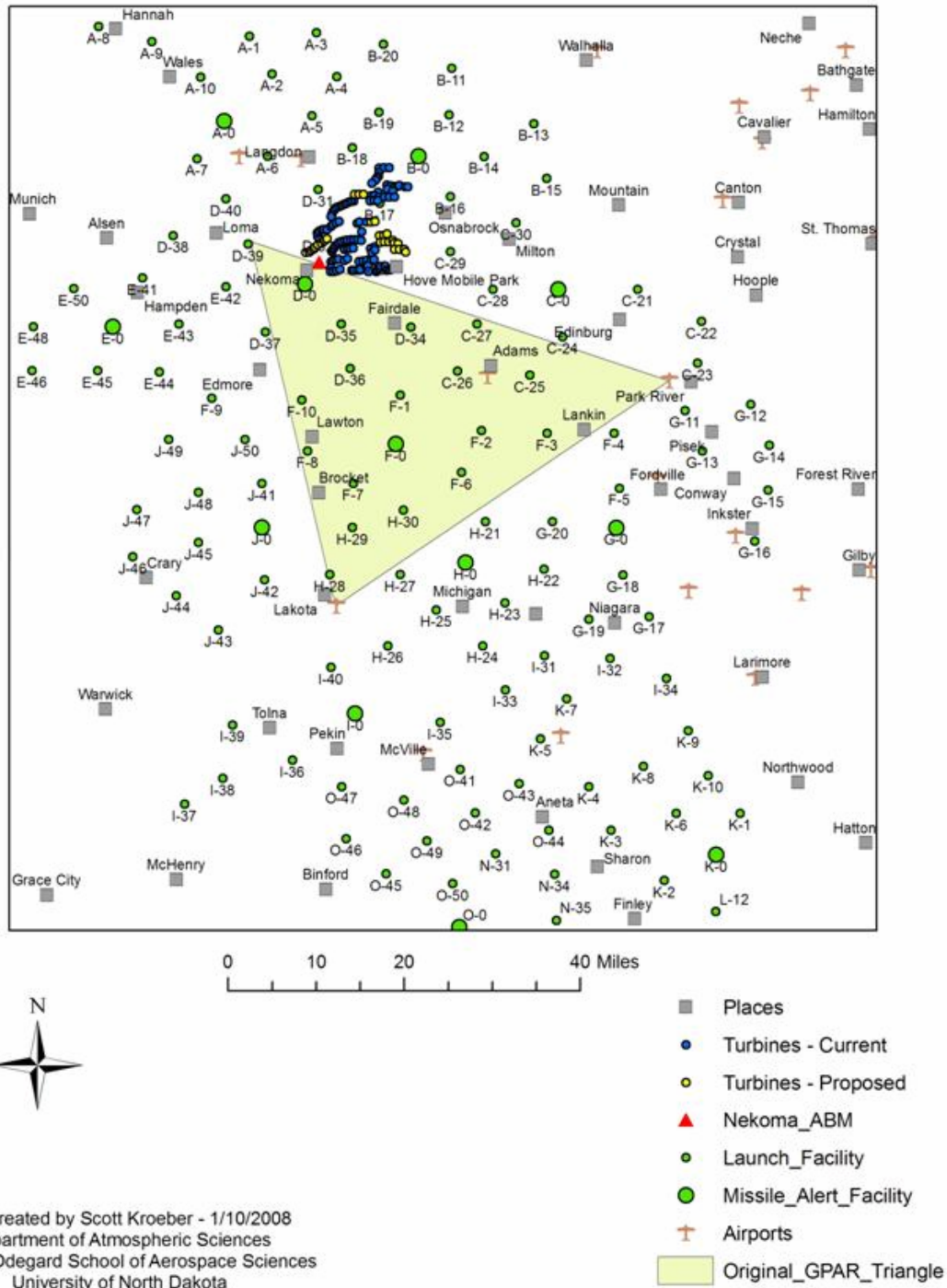


Figure 2. UND risk mitigation study area.

## 2.2 UND FSX Simulation Facility

Our primary task was to develop an FSX simulation of the airspace that included a human-controlled unmanned aircraft (in our case a UAS), several manned aircraft

(varying from private aircraft to commercial), and an air traffic control center (ATC). To accomplish our task, we installed Microsoft XP and FSX on three computers connected by Ethernet, as shown in figure 3. The intent was to operate FSX in multiplayer mode and have the leftmost computer generate the artificial intelligence-controlled manned aircraft, the center computer act as the ATC, and the rightmost computer support the human-controlled unmanned aircraft (note the joystick in figure 3). However, when operating FSX in multiplayer mode the ability to create artificial intelligence-controlled aircraft is disabled. Fortunately, Microsoft has an extensive suite of tools included with the FSX deluxe edition and we were able to use the SimConnect API to create the artificial intelligence-controlled manned aircraft while operating in multiplayer mode. Using multiplayer mode with FSX, we are able to recreate a good rendition of the airspace.



Figure 3. Phase 1 UND FSX simulation facility.

The first task in using SimConnect is to create the aircraft by calling a function named `SimConnect_AICreateNonATCAircraft`. This function requires a SimConnect handle and the name of the type of aircraft created, such as a Douglas DC-10 or Mooney Bravo. An initialization structure called `Init` requires an initial longitude, latitude and altitude for the aircraft. The second step in the process is setting up the flight plans for each of the aircraft. These plans are sent using a `SIMCONNECT_DATA_WAYPOINT` structure as an array. Each element in the array contains latitude, longitude, altitude, and speed in knots. A flag is also used that will allow the speed of the aircraft to change; another flag will make the plane loop back to the first waypoint. After the points have been entered

into the SIMCONNECT\_DATA\_WAYPOINT array, the data must be passed to a function called SimConnect\_SetDataOnSimObject. The third step is to use the MyDispatchProcSO function to connect to FSX to create the aircraft and load the flight plans.

For demonstration purposes, we also incorporated a model of the RQ-1 Predator UAS into FSX. The core of the model was purchased on-line. However, we encountered several problems with the model as the gMax modeling and rendering software and its FSX development plugin were not entirely compatible. Thus, the instrument panel does not render correctly. However, we did get the resulting model files integrated with aircraft performance and definition files to create a usable RQ-1 Predator simulation for FSX.

Using FSX in multiplayer mode we are able to recreate a good rendition of the airspace. However, the value of FSX can only be fully realized if we can extract the FSX airspace information and send it to the Linux-based RCC IDS. Thus, we must derive some mechanism for extracting second-by-second aircraft information (location, altitude, etc) from FSX such that it can be sent via a socket to the under-development Linux-based RCC IDS. To date, we have been only marginally successful at this task and we have begun to also consider using Flight Gear (open source, but not as capable as FSX) for the airspace simulation.

## **RCC IDS**

The design of RCC IDS is not as obvious as one might think as there is no one model to follow. As a 2004 DOT/FAA technical report sites [6], there are several different types of IDSs in use throughout the FAA's ATC facilities: towers, Terminal Radar Approach Controls (TRACONs), Air Route Traffic Control Centers (ARTCCs), and Flight Service Stations (FSSs) all have different IDSs. The variety of IDSs may be expected given the variety of tasks each FAA facility is expected to perform, however what is not expected is that supposedly identical IDSs have different interfaces depending on who the contractor was. Yet, one can argue that is to be expected given the work of Nielsen [7] who concluded that "No design standard can ever specify a complete user interface" and the work of Ahlstrom and Longo [8] who point out that the same (interface) standard may be implemented in a variety of ways.

Given the lack of a uniform IDS model and the unique requirements of UND's RCC IDS (as understood at this time: a 3-D display with the ability to rotate the view, the ability to predict flight paths for advanced deconfliction alerts, the ability to overlay Doppler weather radar, the use of MIL-STD-2525B Change 1 or NATA Specification APP-6A military symbols, the ability to incorporate ADS-B data and/or FAA air traffic data, and the ability to graphically show UAS operational boundaries), it seems prudent to design an IDS from first principles using a spiral model (such as Boehm's) where the IDS designers can work directly with those developing the rest of the system and with those who will use the resulting IDS.

Using the 2004 DOT/FAA technical report as a guideline, we see that an IDS should be well organized and that organization of the information and controls greatly affects the operator's ability to effectively use the system. The IDS must be navigable and consistent. The IDS should clearly indicate when pertinent information was last updated. Information displayed should be complete and relevant. Use of color and color combinations should be consistent. Buttons should be represented in shades of gray and use a consistent font size and font type. Hardware selection is also an important issue as the use of a keyboard for any required data entry should only be provided to operators who have the authority to enter data. The use of a mouse or trackball versus a touch screen display has advantages and disadvantages. Both facilitate interaction with the IDS. However, use of a mouse/trackball requires the operator to coordinate the position of the physical device with the icon on the screen and when used with multiple displays the operator can momentarily lose track of the icon during the screen-to-screen transition. Use of a touch screen can be problematic if the screen has a low touch resolution. Use of a touch screen also requires some form of adjustable mounting as the operator's arm will fatigue. Use of a touch screen also requires frequent cleaning to remove fingerprints which obscure information. The report indicates that touch screen users often preferred to use a trackball over their finger/stylus or a mouse. Screen size and resolution must be sufficient to clearly display the relevant information.

We have also investigated the use of color for altitude representation and are considering the work of Johnston, Horlitz and Edmiston [9] whose 1993 paper concluded that color is better than achromatic for altitude indication and that both discrete color bands or continuous color bands were applicable (depending on the situation/application). We have also reviewed the work of Xing [10] whose 2006 report cites the non-standard use of color schemes by the different manufacturers of ATC displays and whose report proposes guidelines for use of color in IDSs such as:

- Use color to capture attention. However, the effectiveness of color in this manner is highly dependent on the luminance and chromaticity differences of the colors used (i.e. how reliably and quickly can the colors on the screen be named) and on the consistent use of specific colors to represent specific situations across all components in the IDS.
- Use color to identify certain types of information to improve the operator's effectiveness in retrieving relevant information in complex/cluttered displays.
- Use color to segment complex display scenes to organize/cluster related information. However, in some cases segmentation is better achieved through a reorganization of the display.

However, there are also potential negative effects of the use of color:

- As the number of colors used in the IDS is increased, their effectiveness diminishes.
- Multiple colored objects that require attention should not be onset simultaneously within the visual field.
- Colors should be used in accordance with controller's experience.
- The luminance contrast between text and the background must be considered regardless of the desire to use color.

- Color alone should not serve as the sole method to convey meaning.

We are currently developing a prototype RCC IDS that renders the applicable airspace in 2-dimension (figure 4). The RCC IDS currently has the ability to:

- Acquire and display weather information (temperature, wind speed, barometric pressure, and wind gusts) on 5 minute intervals. The weather information is retrieved from a UND website that archives weather information from sensors located at a field research site in eastern North Dakota.
- Acquire and display Doppler weather radar. The Doppler weather radar is retrieved from the NOAA on a 15 minute interval.
- Import and display GIS shape files. Data currently exists for political boundaries, roads, railroads, towns, and terrain. We have also manually created GIS -like data files for high tension utility lines, schools, airports, and towers (TV/radio transmission and wind power generation).
- Import and display areas such as Military Operations Areas (MOA's). Data currently exists for the Tiger North and Tiger South MOA's. We also have the ability to create, import, and display UAS operational areas.
- Be menu-driven. The RCC IDS is completely menu-driven allowing the operator to display or to not display a variety of objects and allows the operator to pan, scroll, zoom-in, and zoom-out.

Once the Prototype UND RCC IDS has the ability to import real-time data from the UND FSX simulation facility, it will be incorporated into the FSX simulation facility

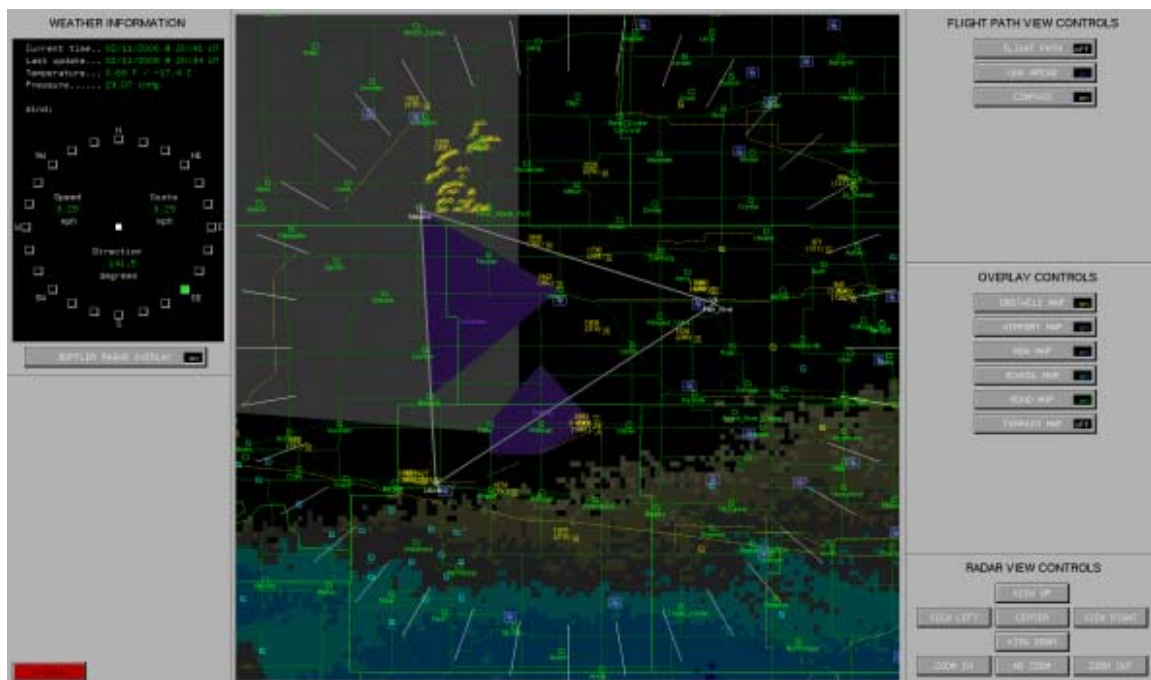


Figure 4. Prototype UND RCC IDS.

## Conclusion

UND is developing airspace within the state of North Dakota where UASs can be tested/operated without the need for an on-board sense and avoid system or temporary flight restrictions (TFRs) and the Computer Science Department has been tasked with developing a majority of the system software, including an airspace simulation and the RCC IDS. Since the Air Force wants to explore “new” ideas for IDSs, and since the current standards have not resulted in any one model to follow, replication of an existing ATC system was deemed unsuitable. Furthermore, the Air Force wants UND to develop a RCC IDS that is designed such that it reduces the workload on the operator. Obviously, given that UND does not yet have radars nor the supporting infrastructure in place, a dynamic simulation of the airspace is required for RCC IDS development and evaluation. Microsoft’s Flight Simulator X was selected as a commercially developed, cost-effective, adaptable, interactive, Internet-capable, and graphically rich solution for the airspace simulation and OpenGL for development of the display system. Use of FSX has greatly streamlined our development efforts. Much of the RCC IDS has been prototyped and a UAS operator IDS will be developed next.

## References

- [1] S. R. Herwitz, L. F. Johnson, S. E. Dungan, and G. Witt, “Orchestrating a Near-Real-Time Imaging Mission in National Airspace using a Solar-Powered UAV,” 2nd AIAA Unmanned Unlimited Systems, Technologies, and Operations-Aerospac, 15-18, San Diego, California. 2003.
- [2] S. R. Herwitz, K. Allmendinger, R. Slye, S. Dunagan, B. Lobitz, L. Johnson, and J. A. Brass, “Nighttime UAV Vineyard Mission: Challenges of See-and-Avoid in the NAS,” AIAA 3rd, “Unmanned Unlimited” Technical Conference, Workshop and Exhibit, 20-23, Chicago, Illinois. 2004.
- [3] R. E. Weibel and R. John Hansman, Jr., “Safety Considerations for Operation of Different Classes of UAVs in the NAS,” AIAA 4th Aviation Technology, Integration and Operations (ATIO), Forum 20-22, Chicago, Illinois. 2004.
- [4] R. E. Weibel and R. John Hansman, Jr., “An Integrated Approach to Evaluating Risk Mitigation Measures for UAV Operational Concepts in the NAS,” Infotech@Aerospace 26-29, Arlington, Virginia. 2005.
- [5] P. Narayan, P. Wu, D. Campbell, and R. Walker, “An Intelligent Control Architecture for Unmanned Aerial Systems (UAS) in the National Airspace System (NAS),” 2nd Australasian Unmanned Air Vehicle Systems Conference, 20-21. Melbourne, Australia. 2007.
- [6] T. Yuditsky, F. Friedman-Berg, and A. Smith, “Design of Information Display Systems for Air Traffic Control,” Federal Aviation Administration, William J. Hughes

Technical Center, Atlantic City International Airport, NJ. Retrieved from <http://hf.tc.faa.gov/products/bibliographic/tn0433.htm> on June 2, 2007.

[7] J. Nielsen, "Do Interface Standards Stifle Design Creativity?," Jakob Nielsen's Alertbox, August 22, 1999. Retrieved from <http://www.useit.com/alertbox/990822.html> on Feb 5. 2008.

[8] V. Ahlstrom and B. Kudrick, "Human factors design standard," Federal Aviation Administration, William J. Hughes Technical Center, Atlantic City International Airport, NJ. Retrieved from <http://hf.tc.faa.gov/hfds/default.htm> on June 2, 2007.

[9] J. C. Johnston, K. L. Horlitz, and R. W. Edmiston, "Improving Situation Awareness Displays for Air Traffic Controllers," Proceedings of the Seventh International Symposium on Aviation Psychology, Ohio State University.1993.

[10] J. Xing, "Color Analysis in Air Traffic Control Displays, Part I. Radar Displays (DOT/FAA/AM-06/22)," Civil Aerospace Medical Institute, Federal Aviation Administration, Oklahoma City, OK. 2006. Retrieved from <http://www.faa.gov/library/reports/medical/oamtechreports/2000s/media/200622.pdf> on October 18, 2007.

# An ANNTI (Artificial Neural Network Text Image) Spam Filter

Jeremiah Barr and  
Charles Ashbacher  
Department of Computer  
Science and Mathematics  
Mount Mercy College  
Cedar Rapids, IA 52402  
jrb62307@att.net and  
cashbacher@yahoo.com

## Abstract

Spam is an enormous problem in the information age. Spammers have begun to avoid traditional text analyzing spam filters by sending image files that display text messages. Since these so-called “text images” typically require more space and bandwidth than text-only spam messages, this newer type of spam presents a serious problem. Interestingly, text images typically have a large percentage of sharp contrasts across their pixel elements. This property allows OCR (object character recognition) techniques to detect characters within these images. Consequently, we devised a filtering algorithm that uses an OCR module and a Bayesian spam filter to recognize text image spam. We analyzed images built from an actively used e-mail message box to measure the algorithm’s performance.



## Introduction

Over the last few years, spam sent in the form of images displaying text has begun to replace traditional text-only spam messages. While e-mail filtering software packages can now filter some text image spam messages with rejection rules based on image checksums and similar filtering techniques, spam still presents a major issue as it accounts for approximately 70 percent of all e-mails on the Internet (Zeller). At the same time, spammers continue to produce new images with random noise in an attempt to slip past the rule-based filters. Worse yet, the new wave of text image spam requires more bandwidth than traditional text messages, which presents a major issue for system administrators and end-users alike in this age of bandwidth hungry Internet applications.

To help in the fight to protect valuable bandwidth and save the many millions of Internet users the time cost associated with deleting spam, this work describes an algorithm for recognizing spam text images based off of their contents. The algorithm incorporates OCR and a Bayesian classifier, which we will consider in brief. Afterwards, the results of an initial implementation will be discussed along with possible areas of improvement and future research directions.

### 1. The OCR Module

In general, OCR software must find and recognize characters within an image document. The OCR module considered here makes use of a BFS algorithm to find text image characters and an SOM (self organizing map) set to recognize character patterns. The character location algorithm roughly models the way the human eye tracks character objects and receives character information, while the SOM set roughly models the way the human brain processes that information to recognize words, phrases, sentences, etc...

The OCR module's text image content approximation algorithm begins processing at the upper left hand corner of a given image. After locating a character pixel pattern, the OCR module determines the pattern's boundaries and scales the pattern to a specified size. All scaled patterns possess the same dimensions so that the OCR module exhibits scale invariance for individual characters. An SOM set accepts the pattern vector and recognizes its contents to produce a literal text character. By literal text character, we mean some character with a text encoding such as Unicode as opposed to an image representation. The recognized characters belonging to a single word – as determined by the spacing between characters in the text image – are concatenated in series. The OCR content approximation algorithm concatenates these words together with space separators to produce a single result string. The OCR process terminates upon reading the bottom right hand corner of an image and returns the result string. In short, the OCR module maps text images to string approximations of their text contents. We will first analyze the primary subcomponent of the OCR module, the SOM set, and then glimpse the functionality of the OCR module as a whole.

## 1.1 The Artificial Neural Network: An SOM

Many different architectures exist for performing pattern recognition. Neural networks represent an attractive choice due to their statistical classification power and ease of implementation. Furthermore, due to the nature of the text image classification problem, where we have documents displayed by images that we must read in order to accept as legitimate content or reject as spam, pattern recognition techniques that roughly model the process of a human being reading a piece of paper and recognizing the characters with biological neural networks represent intuitive solutions. Let's now turn to the reasoning behind the selection of the SOM neural network architecture.

### 1.1.1 On the Choice of ANN

In choosing a neural network architecture for character pattern recognition, we kept the following implementation requirements in mind:

1. Light computation load – Considering the large number of pattern recognitions that can take place during a single image analysis, the neural network must not require excessive computational resources or the filtering application will become impractical for daily use. Moreover, the network training process must not interfere with the usability of the application.
2. Low memory overhead – The filtering application must store a large dictionary of e-mail words, all the while maintaining images and their vector representations in memory. Therefore, the application could require a prohibitive amount of memory if it utilizes a space hungry ANN architecture.
3. Tolerable accuracy – The implemented ANN must provide reasonably accurate character recognition capabilities, up to the point where the Bayesian filter can classify document text approximations with a tolerable accuracy.

With these constraints in mind, the first requirement renders the use of a multi-layered statistical ANN based on three layers – an input layer, a hidden layer with either  $\tanh(\dots)$  or  $\text{sigmoid}(\dots)$  activation functions, and an output layer – impractical. Statistical ANNs of two layers with  $\tanh(\dots)$  or  $\text{sigmoid}(\dots)$  activation functions in either or both layers do not provide accurate enough pattern recognition results. Threshold based ANNs akin to the classical McCulloch-Pitts architecture also lack sufficient accuracy and they display poor generalization characteristics.

The SOM due to Teuvo Kohonen (Kohonen) marks the most promising architecture surveyed here which meets the above constraints without extensive optimization. Furthermore, in contrast to statistical and threshold based ANNs, most training algorithms for this architecture take place in an unsupervised setting. This means that the SOM merely clusters input patterns into classes, or literal text characters for our purposes, without the user providing the desired outputs. In short, the SOM described here does not need user intervention.

The SOM has many applications, some of which include data compression, document organization, stochastic information processing, and the visualization of high dimensional data (Kohonen). The SOM also plays the role of a theoretical model of biological brain structures. In this work, we focus on the pattern recognition capabilities of this ANN.

### 1.1.2 The SOM in Brief

The structure of the SOM consists of two layers of nodes: an input layer and an output layer. The input layer of nodes is indexed 1, 2, ..., n, and each node contains a real value. Now, let  $x = (x_1, x_2, \dots, x_n) \in [-1, 1]^n$  be a stochastic input pattern vector normalized to length 1. For a vector  $y = (y_1, y_2, \dots, y_n) \in \mathfrak{R}^n$ , we attain the normalized vector  $x$  with  $x = y / \|y\|$ , where  $\|y\| = (y_1^2 + y_2^2 + \dots + y_n^2)^{1/2}$  denotes the Euclidean norm of  $y$ . To process an input pattern vector  $x$  from the set  $\{x\}$ , we first set the values of input nodes with indices 1, 2, ..., and n to  $x_1, x_2, \dots$ , and  $x_n$  respectively.

The output layer is similar to the input layer, in that each node is labeled by an index from 1, 2, to an integer d. Each output node represents a certain class, so that the SOM maps the vectors in  $\{x\}$  to particular classes. Each output node  $i$  contains a normalized reference vector  $m_i = (m_{i,0}, m_{i,1}, \dots, m_{i,n}) \in \mathfrak{R}^n$ . In a sense, the reference vectors serve as the memory of the SOM. The output algorithm for this SOM determines the index  $c$  of the output node with the reference vector most similar to  $x$  with  $x^T \cdot m_c = \max_{i=1,2,\dots,d} \{(x^T \cdot m_i)\}$ . Here,  $^T$  denotes the transpose operator. Consequently, only one output node activates for an input – the node with index  $c$ . We can then use the index  $c$  as an index to an ordered set of classes, like an array of literal text characters.

Notice that we treat the dot product  $x^T \cdot m_c$  as a measure of similarity between  $x$  and  $m_c$ . To understand this assertion, observe that  $x^T \cdot m_c = \|x\| \|m_c\| \cos(\theta)$ , where  $\theta$  represents the mutual angle between  $x$  and  $m_c$ . Since both  $x$  and  $m_c$  have length equal to unity,  $x^T \cdot m_c = \cos(\theta)$ . Provided  $x \neq (0, 0, \dots, 0) \neq m_c$ , if  $\theta$  equals  $\pi / 2$ , so that  $x$  and  $m_c$  are orthogonal and thus linearly independent, then  $x^T \cdot m_c = 0$ . On the other hand, if  $\theta$  equals 0, so that  $x$  and  $m_c$  lie on each other and thus constitute the same vector or a scalar multiple thereof, then  $x^T \cdot m_c = 1$ . In this way, the more  $x^T \cdot m_c$  differs from 0, the more similar  $x$  is to  $m_c$ . Since this SOM uses the dot product as the measure of similarity, it can be deemed a “dot product SOM”; other SOMs exist (Kohonen).

For the SOM to map character image patterns to output nodes, we must train the network. Training consists of varying reference vectors such that the SOM maps similar character pattern vectors to the same output node and thus to the same text character. To do so, we collect a set of normalized training input vectors  $\{x\}$ . Each  $x$  denotes the vector training pattern of a particular text character that we desire the OCR module to recognize. These patterns get mapped from training image pixels. Next, the reference vector components get set to random values. The training algorithm then enters a process consisting of a series of reference vector adjustment cycles. During every cycle, each training vector  $x$  is considered. The algorithm seeks to find the winning output node with index  $c$  for each  $x$

and to adjust its reference vector such that  $x$  and the reference vector become more similar. Therefore, it takes

$$m_c(t+1) = \frac{[m_c(t) + \alpha x]}{\|m_c(t) + \alpha x\|},$$

and sets all other  $m_i(t+1) = m_i(t)$ , where  $t + 1$  denotes the time of the current adjustment cycle. The variable  $\alpha \in (0, 1]$  expresses the learning rate of the algorithm. In the implementation, we let  $\alpha = 0.5$ . The training algorithm terminates when it completes a predetermined cycle count or when the biggest component change for all vectors  $m_c(t+1)$  falls below an arbitrary amount. We chose a maximum cycle count of 10,000 and a component change minimum of 1E-5. In this way, the training algorithm develops the SOM reference vector values such that the SOM classifies input patterns as their corresponding text characters.

Generally speaking, a great plethora of training algorithms for the SOM exist. For instance, we can modify the current algorithm such that  $\alpha$  denotes a continuous function of  $t$  that decreases monotonically. The implemented algorithm was chosen for its ease of computation.

### 1.1.3 The SOM Set

While an individual SOM can recognize the characters of different fonts, recognition accuracy tends to fall when the font of the training set does not represent the fonts displayed in actual text images well. For example, an SOM trained to recognize characters of the Courier font may perform unacceptably when tasked to recognize Arial characters. The OCR module's design reflects one solution to this problem: the use of a set of SOMs where each individual SOM recognizes characters from a particular representative font. Observe that the set need not contain an SOM for all fonts, but rather the fonts which best represent those applied in text images sent to a filter user.

Recall that the OCR module locates characters and sends their patterns to the SOM set for recognition. Each SOM in the set performs pattern recognition with a character pattern to give the corresponding literal text form of the character. Of course, this literal text character has a correspondence with a training pattern, so the OCR module compares the character pattern with the corresponding training pattern for each SOM's text character result. The output of the SOM with the training pattern most similar to the character pattern becomes the output for the entire SOM set. The measure of similarity is given by the ratio of the number of equal values between the vectors at each element index to the total number of elements in an individual vector. Keep in mind that all character images are scaled to have the same dimensions during both training and usage, so the image vectors possess the same number of elements. When the OCR module selects an SOM's output it increments a so-called vote counter for that SOM. Prior to reading an input image, the counters of all the SOMs get set to zero. Then, by the end of reading a document, the counter with the highest value serves as an indicator of the font

used in the entire document. Overall, the cross-comparison between SOMs gives the OCR module a degree of font invariance.

## 1.2 On the OCR Module

To reiterate, the SOM set recognizes located character patterns and translates them into literal text characters. In turn, the OCR module concatenates the character belonging to any single word together and then concatenates recognized words together with space delimiters to form a single result string for a given text image. We will now consider how the OCR module locates character patterns in text images and see the details of the interaction between the SOM set and the OCR module.

The OCR module does not accept images of jpeg, gif, png, or bmp formats directly. Instead, the OCR module requires text images to enter a feature extraction phase prior to processing. Feature extraction consists of converting an image  $\mathbf{I}$  of width  $w$  and height  $h$  into a binary vector  $v \in \{-1, 1\}^{wh}$ . This means that  $\mathbf{I}$  gets mapped into a monochromatic representation,  $v$ . Moreover, the character searching algorithm expects the “background” of a text image to possess the color white, denoted as  $-1$ , and the characters to have a black color value of  $1$ . If the opposite case holds true – as detected by a frequency analysis of white and black pixel counts in  $v$  based on the fact that background pixels tend to fill the majority of a text image – then the components of  $v$  with a value of  $-1$  get set to  $1$  and vice versa.

The conversion of  $\mathbf{I}$  from a full color image to a monochromatic representation serves two purposes. First, the reduction in the dimensionality of  $\mathbf{I}$ 's data leads to faster OCR execution. Second, this conversion gives the content approximation algorithm a degree of color invariance so as to remove the need to train the SOM set with patterns of different colors. More importantly, this conversion tends to filter out non-character graphical elements that could cause substantial OCR errors; however, in images where characters do not contrast greatly from background graphics, character data might get filtered out as well. Therefore, while the conversion process has many benefits, it can also result in the loss of significant information.

The OCR module accepts a binary vector  $v$  as an input, along with the dimensions of  $\mathbf{I}$ . Character pattern locating begins at the upper left hand corner of the text image. The location algorithm traces out the bottom and right sides of successively larger squares in  $v$  with upper left hand corners corresponding to that of the original text image. Tracing continues until the algorithm finds a position  $p$  of black color. Assuming that such a position exists, a sub-algorithm `determineRegionBounds` is invoked to determine a boundary rectangle that contains  $p$  and its neighboring black pixels, along with their neighboring black pixels, and so on. `determineRegionBounds` represents a breadth first search algorithm, wherein a scan of the neighbors to pixels polled from the head of a queue finds neighboring black pixels which in turn get inserted at the end of the queue. In this way, `determineRegionBounds` determines the groups of adjacent pixels that define character patterns. The left most, right most, top most, and bottom most pattern

pixel coordinates get stored in a rectangle data structure. This structure essentially consists of a 4-dimensional vector of integers where the first two components correspond to the x and y positions of the rectangle's upper left-hand corner and the second two represent the x and y positions of the rectangle's bottom right-hand corner. The populated structure gets returned. The design of the sub-algorithm relies on the assumption that character patterns possess contiguous groups of pixels. Note that while characters such as "i", ";", "?", and "!" do pose a problem given this supposition, the search for neighboring pixels can extend toward the top or bottom of the text image by an arbitrary amount of pixels. Consequently, this search extends to 7 pixels above the most recently polled pixel and 3 pixels below. The pseudo-code for determineRegionBounds follows.

```

Given a text image vector v and initial character pattern pixel position p,
begin
  if p is null then
    return null
  let L, R, T, and B denote the left, right, top, and bottom
    coordinates of boundary rectangle enveloping the current
    character pattern
  let Q be a queue of 2-dimensional integer vectors denoting pixel
    positions
  let visited be a static boolean array equal in size to v which
    persists throughout an execution of the OCR module on a single
    image
  set L = infinity
  set R = -1
  set T = infinity
  set B = -1
  insert(Q, p)
  visited[x(p) + y(p)*w] = true
  while not empty(Q) do
    let u be a pattern pixel position
    set u = poll(Q)
    let l, r, t, and b be integers denoting the boundary
      coordinates of the current scan area
    set l = max{x(u) - 1, 0}
    set r = min{x(u) + 1, w - 1}
    set t = max{y(u) - 7, 0}
    set b = min{y(u) + 3, h - 1}
    for j = l..r do
      for i = t..b do
        if v[j + i*w] is 1 and not visited[j + i*w] then
          let v be a pattern pixel position
          set v = (j, i)
          insert(Q, v)
          set visited[j + i*w] = true
          if x(v) < L then
            set L = x(v)
          if x(v) > R then
            set R = x(v)
          if y(v) < T then
            set T = y(v)
          if y(v) > B then
            set B = y(v)
    return (L, T, R, B)
end

```

**Code Section 1.2.1:** determineRegionBounds: (Text image vectors, position vectors) → (rectangle vectors).

The returned rectangle is then used to extract the character pattern into a vector on which the SOM set performs pattern recognition. The literal text values get concatenated onto the end of a temporary string called buffer. Subsequent to recognizing a character, the next pattern pixel position  $p$  is calculated by traversing the pixels of a rectangle to the right of the previous character pattern boundary rectangle. At this point, `determineRegionBounds` executes again with the new value of  $p$ . On the other hand, if such a point  $p$  does not exist in that rectangle – which has a width equal to the average space width between characters in a word and a height equal to the average character height – the algorithm considers the region enclosed by the rectangle a space and thus deems the string contained in buffer to be a single word. The matched string then gets inserted at the end of a result buffer, and buffer gets set to an empty string.

The character pattern locating algorithm executes the same buffer reading and resetting process whenever it encounters the right boundary of an image. Additionally, the algorithm begins reading at the left hand side of the image and at the bottom of the previous line of character patterns and searches for new patterns in the same way the first located character of the image is found.

OCR execution ends after the right hand side of the bottom most readable line is read. For the sake of concreteness, Code Section 1.2.2 gives a pseudo-code description of the text image content approximation algorithm.

```

Given a text image vector  $v$ , an image width  $w$  and an image height  $h$ ,
begin
  let resultBuffer be the result string
  let buffer be a temporary string for storing located words
  let  $r$  denote rectangle structure bounding a character pattern for
    analysis
  let  $r'$  be the rectangle structure to store the prior instance of  $r$ 
  let  $p$  be a 2-d vector representing a point found in a located
    pattern
  let  $p'$  be a 2-d vector for the first point from which to search when
    analyzing the line of character patterns below the one under current
    analysis
  let SOMSet be the SOM set
  let aveSpaceWidth be the average width of the spaces between adjacent
    characters belonging to one word
  set  $p$  = the located coordinate of the first found pattern
  set  $r$  = determineRegionBounds( $v$ ,  $p$ )
  if  $p$  is not null then
    set buffer = SOMSet.recognize( $v$ ,  $r$ )
  set  $p' = (0, \text{bottom}(r) + 1)$ 
  while  $r$  is not null do
    set  $p = (\text{right}(r) + 1, \text{top}(r))$ 
    set  $p$  = the located coordinate of the next character pattern to the
      right of  $p$ 
    set  $r' = r$ 
    set  $r$  = determineRegionBounds( $v$ ,  $p$ )

    if  $p$  is not null and  $p$  is not on the right boundary of the image then
      set aveSpaceWidth to new average space width including the
        distance between  $\text{right}(r')$  and  $\text{left}(r)$ 
      set buffer = buffer + SOMSet.recognize( $v$ ,  $r$ )
    else if  $p$  is null then
      set  $r = (\text{right}(r'), \text{top}(r'), \text{right}(r') + \text{aveSpaceWidth}, \text{bottom}(r'))$ 
      set resultBuffer = resultBuffer + " " + buffer
      set buffer to be an empty string

```

```

else if p is located at the right boundary of the image then
  set resultBuffer = resultBuffer + " " + buffer
  set buffer to be an empty string
  set p = the located coordinate of the first found pattern
    located on the character line below the previously analyzed one
  set r' = r
  set r = determineRegionBounds(v, p)
  if p is not null then
    set buffer = SOMSet.recognize(v, r)
    set p' = (0, bottom(r) + 1)
  return resultBuffer
end

```

**Code Section 1.2.2:** ocr\_execute: (Text image vectors, integers, integers)→Strings.

With the above functionality, we can approximate the text contents in a text image. Next, we shall see how to classify such images.

## 2. The Spam Filter: A Bayesian Classifier

Researchers have conducted studies of text classifiers employing Bayesian methods for well over a decade. Amidst this body of work we find e-mail spam filters which classify messages into one of multiple categories, including different types of legitimate messages and spam messages (Sahami, et al.). The success of these classifiers has reached the point where many commercial level filters either rely solely on Bayesian techniques or use them in conjunction with user-configured rules.

### 2.1 Bayesian Spam Filter Background

Before we delve into the design of the spam filter, let's consider some preliminary definitions along with Bayes' theorem. Let  $x$  and  $y$  be a pair of events or observations. We label the probability that  $x$  occurs as  $P(x)$  and we call the probability that both  $x$  and  $y$  occur,  $P(x,y)$ , the joint probability of  $x$  and  $y$ . Furthermore, the probability of  $x$  occurring given that  $y$  occurs, denoted  $P(x | y)$ , is called the conditional probability, which we define as  $P(x | y) = P(x, y) / P(y)$ . Two basic rules in probability are the sum rule and the product rule, which we express as  $P(x) = \sum_y P(x, y)$  and  $P(x, y) = P(y | x)P(x)$ , respectively. Notice the relationship between the definition of conditional probability and the product rule. Onto Bayes' theorem:  $P(x | y) = [P(y | x) P(x)] / P(y)$ . Of course,  $P(y) = P(x, y) + P(\sim x, y)$  by the sum rule of probability, where  $\sim x$  means the complementary event of  $x$ . Therefore,  $P(x | y) = [P(y | x) P(x)] / [P(y | x) P(x) + P(y | \sim x) P(\sim x)]$  gives an alternative formulation of Bayes' theorem. We can classify spam with these definitions, Bayes' theorem and some additional assumptions.

Using these concepts, we will now consider the spam filter. The inspiration for this filter stems from Sahami et al., Graham, and Shiffman.



## 2.2 The Spam Filter in Brief

To begin, we shall discuss the foundation of the spam filter's structure and behavior. To this end, every message is represented by a vector of word strings  $s = (s_1, s_2, \dots, s_n)$ . We treat each word  $s_i$  as a feature of the message. Each message  $s$  may belong to one of two classes: the spam class  $S$  or the legitimate e-mail class  $L$ . The basic assumption made in calculating the probabilities associated with message classification is that each probability  $P(s \in S \mid s_i \in s)$  is statistically independent of all other  $P(s \in S \mid s_j \in s)$ , where  $i \neq j$ . This basic supposition places the spam filter in the class of Naïve Bayesian classifiers. Note that more sophisticated feature dependence assumptions are possible. Furthermore, to simplify the filter's computations, we assume that the a priori probability  $P(s \in S) = P(s \in L)$  and that  $P(s \in S \mid s_j \in s)$  is the same regardless of whether  $s \in S$  or  $s \in L$ ; in these assumptions,  $s$  represents a random message variable. We desire to create a framework wherein we attain the probability  $P(s \in S \mid s)$  so as to flag  $s$  as spam or legitimate. To achieve this goal, the spam filter must have a mechanism to measure  $P(s \in S \mid s)$ . Bayes' theorem gives us that mechanism.

That is, Bayes' theorem plays a role in calculating the probability that a message represented by  $s$  is spam given that it contains a particular word,  $P(s \in S \mid s_i \in s)$ . We determine this probability as follows:

$$\begin{aligned} P(s \in S \mid s_i \in s) &= \frac{P(s_i \in s \mid s \in S) P(s \in S)}{P(s_i \in s \mid s \in S) P(s \in S) + P(s_i \in s \mid s \in L) P(s \in L)} \\ &= \frac{P(s_i \in s, s \in S)}{P(s_i \in s, s \in S) + P(s_i \in s, s \in L)}, \end{aligned}$$

as given by Bayes' theorem and the product rule. Therefore, in determining  $P(s \in S \mid s_i \in s)$ , the Bayesian filtering algorithm must calculate the probability that  $s$  is a spam message and the word  $s_i$  belongs to  $s$ ,  $P(s \in S, s_i \in s)$ , and the probability that  $s$  is a legitimate message and the word  $s_i$  belongs to  $s$ ,  $P(s \in L, s_i \in s)$ .

To find the required probabilities, the Bayesian filter training algorithm analyzes two text corpuses comprised of words from messages belonging to  $S$  and words from messages belonging to  $L$ . Now, let  $B$  represent the number of words contained by messages from  $S$ ,  $G$  express the number of words in messages belonging to class  $L$ ,  $b(i)$  denote the number of times the word  $s_i$  occurs in the spam word corpus and  $g(i)$  indicate the number of times  $s_i$  occurs in the legitimate word corpus. The filtering algorithm can thus apply the relative frequency  $b(i) / B$  to approximate  $P(s \in S, s_i \in s)$  and  $g(i) / G$  to approximate  $P(s \in L, s_i \in s)$ . After some substitutions, we have  $P(s \in S \mid s_i \in s) \approx [b(i)/B] / [b(i)/B + g(i)/G]$ .

For the calculation of  $P(s \in S \mid s)$ , we rely on our independence assumptions and the supposition that  $P(s \in S) = P(s \in L)$ . Based on these assumptions, the approximation of  $P(s \in S \mid s)$  is given by:

$$\begin{aligned}
P(s \in S | s) &= \frac{\prod_i P(s \in S | s_i \in s)}{\prod_i P(s \in S | s_i \in s) + \prod_i [1 - P(s \in S | s_i \in s)]} \\
&\approx \frac{\prod_i ( [b(i)/B] / [ b(i)/B + g(i)/G ] )}{\prod_i ( [b(i)/B] / [ b(i)/B + g(i)/G ] ) + \prod_i ( [g(i)/G] / [ b(i)/B + g(i)/G ] )}.
\end{aligned}$$

In the implementation, a message  $s$  receives a spam marking whenever  $P(s \in S | s) \geq 0.50$ . While the 0.50 threshold may appear exceptionally low relative to the thresholds used by traditional text e-mail Bayesian spam filters, this selection allows the filter to compensate for OCR errors. Additionally, the implementation's filtering algorithm only uses the 30 words in  $s$  with the greatest probability of belonging to a spam message or a legitimate message while approximating  $P(s \in S | s)$ .

### 2.3 The Interaction between the SOM Set and the Bayesian Filters

Each SOM in the set possesses a specific Bayesian filter. The voting system discussed at the end of section 1.1.3 The SOM Set determines which filter analyzes or trains on the OCR result string for a text image. That is, during the training phase, the filter belonging to the SOM with the greatest number of votes subsequent to an OCR pass over a training text image serves as the filter that gets trained. Throughout the usage phase, the SOM with the greatest number of votes provides the filter that accepts or rejects the recently read text image. This measure allows the various filters to learn the OCR errors of their respective SOMs during training and to account for said errors during analysis. During implementation verification it was found that when the entire SOM set shares a single filter, recognition performance tends to fall below reasonable usage standards.

## 3. Results

The verification process for the ANNTI spam filter implementation was comprised of five tests and a control run. Four out of the five tests corresponded to a particular font. That is, the first test required the implementation to filter text images in the Arial font, the second test required the filtering of text images in the Courier font, the third test consisted of analyzing text images in the Sans Serif (SS) font, and the fourth test was comprised by the filtering of text images in the Times New Roman (TNR) font. The fifth test required the filtering of a text image set with images containing text in a particular font individually but where the font varied from image to image. The fonts used in the mixed set were the ones mentioned above. The assignment of text image fonts in the mixed set was selected at random: 359 contained Arial text, 321 had Courier text, 341 possessed SS text, and 330 were composed of TNR text. Note that the test text images did not contain graphical objects besides characters such as pictures of people. No rotation of text image characters was performed either. The control run served as a measurement baseline for OCR accuracy where the Bayesian spam filter analyzed the original text comprising the text image sets instead of the text read by the OCR module.

Multiple datasets were compiled to verify the ANNTI spam filter. One such dataset consisted of images of the characters belonging to the Latin alphabet, the numerals 0-9 and the punctuation characters !, ", #, \$, %, &, ', (, ), \*, +, ,, -, ., /, :, ;, <, =, >, ?, @, [, \, ], ^, \_ , ` , {, |, }, and ~ in the Arial, Courier, SS, and TNR fonts. The training process for the SOM set produced the necessary reference vectors for four distinct SOMs to recognize image patterns containing single characters of a particular font. The same reference vector sets were used during each test for the sake of consistency.

The message box under analysis did not contain a sufficient number of text images to conduct a reliable test, so we employed a converter application to generate a series of gif images displaying the body text of each individual message. The spam filter training algorithm analyzed five text corpus pairs built by the OCR module from the generated text images. Each pair corresponded to a particular font; within the pairs, one text corpus contained the text from spam messages while legitimate message text comprised the other corpus. Throughout each test, the implementation trained with 677 text images and analyzed 674 total test messages made up of 321 spam images and 353 legitimate images. In the case of the control run, the original text contents of the 677 training text images acted as the learning set while the 674 test messages had the same treatment. Summaries of the results now follow.

Test – Message Group	Total Correct	Percent Correct
Arial – Spam Messages	307	95.6
Arial – Legit Messages	312	88.5
Arial – All Messages	619	91.8
Courier – Spam Messages	306	95.3
Courier – Legit Messages	320	90.7
Courier – All Messages	626	92.9
SS – Spam Messages	314	97.8
SS – Legit Messages	323	91.5
SS – All Messages	637	94.5
TNR – Spam Messages	306	95.3
TNR – Legit Messages	327	87.4
TNR – All Messages	633	93.9
Mixed – Spam Messages	262	81.6
Mixed – Legit Messages	321	90.9
Mixed – All Messages	583	86.5
Control – Spam Messages	287	89.4
Control – Legit Messages	335	94.9
Control – All Messages	622	92.3

**Table 3.1:** A summary of the accuracy attained by the implementation while filtering the test and control sets.

	P = 0.0	0.0 < P ≤ 0.1	0.1 < P < 0.5	P = 0.5	0.5 < P < 0.9	0.9 ≤ P < 1.0	P = 1.0
Arial – Freq.	208	114	4	79	9	36	224
Relative Freq.	0.309	0.169	0.006	0.117	0.013	0.053	0.332
Courier – Freq.	244	80	11	3	12	56	268
Relative Freq.	0.362	0.119	0.016	0.004	0.032	0.083	0.398
SS – Freq.	202	125	3	93	13	54	184
Relative Freq.	0.300	0.185	0.004	0.138	0.019	0.080	0.273
TNR – Freq.	210	128	4	81	4	84	163
Relative Freq.	0.312	0.190	0.006	0.120	0.006	0.125	0.242
Mixed – Freq.	197	179	4	74	5	72	143
Relative Freq.	0.292	0.266	0.006	0.110	0.007	0.107	0.212
Control – Freq.	239	119	11	10	20	84	191
Relative Freq.	0.355	0.177	0.029	0.015	0.030	0.125	0.283

**Table 3.2:** The frequencies (freq.) for intervals of spam probabilities assigned to test text image messages and the control run’s plain text messages.  $P = P(s \in S)$ , where  $s$  denotes a random message variable across a particular test/control set.

As Table 3.1 suggests, the implementation may function well enough for some users. However, while the spam recognition rate tended to fall within acceptable limits, the rate of misclassification of legitimate messages would probably prove too high for most users. Moreover, since spammers and legitimate message senders will vary message fonts, the mixed font test results suggest that the current implementation may need a very large training dataset to give useful classifications of real text image e-mails.

The difference in spam message recognition performance between the control run and the tests indicates that the Bayesian filter can perform better in classifying some messages read by the OCR module than in classifying their plain text counterparts. This apparent anomaly may arise from OCR errors since the Bayesian filter must account for these errors when analyzing text images. That is, the errors captured during text image filter training may have caused the filter to assign unnecessarily high spam probabilities to some words. This would explain why the implementation classified legitimate messages with better accuracy during the control run than during any of the tests. Along the same lines, the message box under analysis contained many spam messages constructed to cause misclassification by Bayesian filters. Such messages have large counts of unnecessary words that could indicate message legitimacy. The determination of the exact cause(s) of these recognition anomalies may require an intricate analysis.

The message spam probabilities tended to fall close to 0.0 or 1.0. In most tests, a substantial set of images also had probabilities close to 0.5, which implies that such images are neutral. Since most of these neutral images actually belonged to the spam class, the selection of a 0.5 classification threshold for the Bayesian filter proved appropriate for this dataset. Unfortunately, this threshold choice also led to a large number of false positives. We conjecture that filter training should also consist of modeling a threshold function that varies with message attributes like the sender and the message header.

## 4. Conclusions and Further Work

In regards to creating a commercial grade text image filter, the initial performance of the implementation falls short of most users' needs. Most notably, the false positive rate presents a practical issue. Many users of traditional spam filters do not check the junk e-mail folder to recover the lost messages. Additionally, in tests involving text images containing graphical data besides characters, the OCR module performs far worse. By its very construction, whenever the text image reading algorithm encounters say a picture of a person, it traces the entire picture's pixels and attempts to recognize the pattern as a character. Furthermore, the SOM set does not possess strong enough generalization capabilities to recognize rotated characters. OCR performance also suffers when characters overlap. These issues will need resolution before the implementation can serve the general public.

On the other hand, as an initial attempt at solving the spam problem with artificial intelligence tools, the performance of the implemented filter shows promise. The ANNTI spam filter may require little modification to solve the problems mentioned above. The OCR character location algorithm could read different image vectors corresponding to each distinct color channel, e.g. red, green and blue, while approximating the content of a text image. This solution may alleviate the severity of the overlapping character problem. The SOM set could also recognize characters as collections of specific components – such as an edge – in a hierarchical fashion so as to recognize the various features of characters independent of their orientation or size, much like biological neural networks. The ability to distinguish graphical objects besides characters may arise from a different OCR location algorithm that will abort upon reading a graphic that lacks subcomponents matching any learned character pattern. Finally, the implemented spam filter represents a degenerate Bayesian classifier in the sense that it only applies Baye's theorem to calculate the conditional spam probabilities for given words, but not entire messages. A full Bayesian treatment could produce more accurate results. The filter could also take into account message features such as e-mail headers and sender fields. In summary, the ANNTI spam filter gives us a nice foundation on which to build more sophisticated text image filters.

## References

- Bishop, Christopher M.. Pattern Recognition and Machine Learning. 1st. New York: Springer Science+Business Media, LLC. 2006.
- Bishop, Christopher M.. Neural Networks for Pattern Recognition. 1st. New York: Oxford University Press Inc. 1995.
- Graham, Paul. "A Plan for Spam." Paul Graham. 2002. 10 November 2007. <<http://www.paulgraham.com/spam.html>>.

- Heaton, Jeff. Introduction to Neural Networks with Java. 2007. 14 December 2007.  
< <http://www.heatonresearch.com/articles/series/1>>.
- Kohonen, Teuvo. Self-Organizing Maps. 3rd. Berlin, Heidelberg, New York: Springer.  
2001: 1-328.
- Sahami, Mehran, S. Dumais, D. Heckerman, and E. Horvitz. "A Bayesian Approach to  
Filtering Junk E-mail." AAAI'98 Workshop on Learning for Text Categorization.  
1998.
- Shiffman, Daniel. "Bayesian Filtering." daniel shiffman. 07 February 2006. 24 November  
2007. <<http://www.shiffman.net/teaching/a2z/bayesian/>>.
- Somervuo, Panu. "Redundant Hash Addressing of Feature Sequences Using the Self-  
Organizing Map." Neural Processing Letters. 10. 1999: 25-34.
- Zeller, Tom. "The Fight Against V1@gra (and Other Spam)." New York Times. 21 May  
2006.<[http://www.nytimes.com/2006/05/21/business/yourmoney/21spam.html?pagewanted=1&\\_r=2&adxnlnx=1194278478-jjggJZE%20qfIJIGxul7xv1g](http://www.nytimes.com/2006/05/21/business/yourmoney/21spam.html?pagewanted=1&_r=2&adxnlnx=1194278478-jjggJZE%20qfIJIGxul7xv1g)>.

# Reflections on a Classic Trio of Graph Problems

Thomas E. O'Neil  
Computer Science Department  
University of North Dakota  
Grand Forks, ND 58202-9015  
oneil@cs.und.edu

## Abstract

CLIQUE, INDEPENDENT SET, and VERTEX COVER are a classic trio of NP-Complete graph problems. There are polynomial-time reductions between these problems, and the reductions are so direct that they have traditionally been considered to be alternate definitions of a single underlying problem. This conventional belief was challenged by Richard Stearns and Harry Hunt in 1990. They conjectured that when a graph is represented with an adjacency list, CLIQUE is actually an easier problem than the other two.

In this paper we examine the reductions among the three problems to reaffirm their unity. The distinction between CLIQUE and the other problems demonstrated by Stearns and Hunt is completely dependent on using a list of edges to represent a graph. We could just as easily use the list of missing edges to fully define a graph, and when we do, there is no reason to believe one problem is harder than the others. We then reexamine the trio of problems giving the edges and missing edges equal emphasis.

# 1 Introduction

The graph problems CLIQUE and INDEPENDENT SET are classic NP-complete problems [1, 3]. They were used so frequently in reductions among combinatorial problems that Garey and Johnson [2] classified them as core problems in the NP-complete class. They are also very closely related, and for many years they were considered to be different descriptions of the same underlying problem, both sharing the same complexity. In 1990 Stearns and Hunt [5] challenged that conventional wisdom. They defined power indices as a measure of complexity among NP-complete problems, where problems with lower power index have lower complexity. They determined that CLIQUE has an algorithm with complexity  $O(2^{n^{1/2}})$  (index one-half), while most NP-complete problems, such as the SATISFIABILITY problem for Boolean expressions, have complexity  $O(2^n)$  (power index one), and there are no known linear reductions from index-one problems to index one-half problems. Since there was a known linear reduction from SATISFIABILITY to INDEPENDENT SET and there was no known linear reduction from INDEPENDENT SET to CLIQUE, this led to the conclusion that CLIQUE is actually a simpler problem.

The reduction from INDEPENDENT SET to CLIQUE requires the construction of the complement of a graph's edge set. A graph contains an independent set of  $k$  vertices if and only if the complement graph contains a clique of  $k$  vertices. Considering a graph  $G = (V, E)$  with  $n$  vertices in  $V$  and  $e$  edges in  $E$ , we expect  $e$  to be  $O(n)$  or  $O(n^2)$ . When reducing INDEPENDENT SET to CLIQUE, the worst case occurs when  $e$  is  $O(n)$  and the set of missing edges  $\bar{E}$  has size  $O(n^2)$ . The reduction is quadratic, not linear. Of course, as Stearns and Hunt point out, if we represent graphs with adjacency matrices, the reduction is linear, and we could claim that INDEPENDENT SET has index one-half along with CLIQUE. But the adjacency matrix is not a space-efficient representation if  $e$  is  $O(n)$ , and it is not acceptable to claim a lower complexity for a reduction or algorithm by choosing an inefficient representation for the input. So Stearns and Hunt concluded that CLIQUE is an easier hard problem than INDEPENDENT SET, and they advised against using CLIQUE for reductions in NP-completeness proofs, which would presumably "reduce" an easier problem to a harder one. This result was considered significant enough for Richard Stearns to feature it in his Turing Award Lecture [4] in 1994.

It is obvious that representations play a major role in these complexity comparisons. CLIQUE is easier than INDEPENDENT SET if adjacency lists are used to represent graphs, but not if adjacency matrices are used. In the next section we consider another possible representation. A graph could be represented by a list of missing edges.

## 2 Non-Adjacency Lists and INDEPENDENT SET

A graph  $G = (V, E)$  is a structure containing vertices and edges. The vertices are specified by the set  $V$ , and the edges are denoted as unordered pairs of vertices in the set  $E \subset V \times V$ . If  $|V| = n$ , then the number of edges cannot exceed  $n(n-1)/2$ . The edge set is



conventionally represented by a list of edges (adjacency list) or by an  $n \times n$  matrix (an adjacency matrix). We could, however, also represent the edge set with a non-adjacency list – a list of missing edges  $\bar{E}$ , where  $|\bar{E}| = n(n-1)/2 - |E|$ . We could even argue that this is the most direct and efficient graph representation for solving the INDEPENDENT SET problem, where we are looking for the largest subset of vertices that are not adjacent to one another. Actually, this is just a restatement of the classic reduction from INDEPENDENT SET to CLIQUE: the largest independent set in a graph  $G$  is the largest clique in the complement graph  $\bar{G} = (V, \bar{E})$ . If a graph is represented by a missing edge set instead of an edge set, we solve INDEPENDENT SET by finding the largest clique in the missing edge list. The point is that if CLIQUE has power index one-half, then so does INDEPENDENT SET.

The result that INDEPENDENT SET has power index one-half can be rigorously proven without resort to reductions. The algorithm and complexity proof are directly analogous to those employed by Stearns and Hunt [5] for the CLIQUE problem. The *MaxIS* algorithm below selects a vertex, computes a largest independent set  $S_1$  that contains the vertex and a largest independent set  $S_2$  that does not contain the vertex, and returns the larger of  $S_1$  and  $S_2$ . The input is the set of vertices and a list of missing edges.

Algorithm *MaxIS* ( $V, \bar{E}$ )

- ```

// Given a non-empty list of vertices and a list of missing edges for the graph
//  $G = (V, E)$ , this algorithm returns a largest independent set of vertices in  $G$ 
1) if  $|\bar{E}| = |V|(|V|-1)/2$  // all edges are missing  $\Rightarrow V$  is an independent set
2)   return  $V$ 
3)    $x =$  a vertex from  $V$  with fewest missing edges in  $\bar{E}$ 
4)   if  $|\bar{E}| = 0$  // no edge is missing  $\Rightarrow$  any single vertex is a maximal independent set
5)     return  $\{x\}$ 
// Compute a largest independent set containing  $x$ .
6)    $V' = V - \{x\} - \{y \mid (x, y) \in \bar{E} \text{ and } (y, x) \in \bar{E}\}$  // remove  $x$  and its neighbors
7)    $\bar{E}_1 = \bar{E} - \{(v, w) \mid v \in V' \text{ or } w \in V'\}$  // restrict  $\bar{E}$  to  $V'$ 
8)   if  $|V'| = 0$ 
9)      $S_1 = \{x\}$ 
10)  else
11)     $S_1 = \{x\} \cup \text{MaxIS}(V', \bar{E}_1)$ 
// Compute a largest independent set not containing  $x$ :
12)   $\bar{E}_2 = \bar{E} - \{(v, w) \mid v = x \text{ or } w = x\}$ 
13)   $S_2 = \text{MaxIS}(V - \{x\}, \bar{E}_2)$ 
14)  return the larger of  $S_1$  and  $S_2$ 

```

*Theorem 1.* INDEPENDENT SET has complexity  $2^{O(\sqrt{n+k})}$ , where  $n$  is the number of vertices and  $k$  is the number of edges missing from the input graph.

*Proof.* The proof follows the same method as used by Stearns and Hunt to prove the corresponding theorem for the analogous CLIQUE algorithm. We observe that the number of procedure calls during execution of *MaxIS* cannot exceed  $2^n$  for a graph of  $n$  vertices, since for each activation of the procedure, two recursive calls are made with

graphs of fewer vertices. We proceed to prove the following proposition by induction on the number of vertices in graph  $G$ : the number of procedure calls during an execution of the *MaxIS* algorithm on input  $(V, \bar{E})$  is no greater than  $n \cdot 2^{\sqrt{2k}}$ , where  $G = (V, E)$ ,  $|V| = n$ ,  $\bar{E}$  is the set of edges missing from  $G$ , and  $|\bar{E}| = k$ .

*Basis.*  $|V| = 1$ . In this case there are no edges and no missing edges. The condition tested at line 1 is true and  $V$  is returned. There are no additional procedure calls. So we have  $1 \leq 1 \cdot 2^{\sqrt{0}}$ .

*Inductive Hypothesis.* We assume the proposition to be true for any graph with  $n - 1$  vertices, where  $n > 1$ .

*Inductive Step.* We now consider a graph  $G = (V, E)$  with  $n$  vertices represented by the set of missing edges  $\bar{E}$ , where  $|\bar{E}| = k$ . For convenience, we define  $\bar{E}(x)$  to be the subset of edges from  $\bar{E}$  that mention vertex  $x$ . We examine two cases below.

*Case 1.*  $|\bar{E}(x)| < \sqrt{2k}$ . Here the number of vertices not adjacent to the selected vertex  $x$  is less than  $\sqrt{2k}$ . The set  $S_1$  is calculated using only these vertices, so the number of calls to calculate  $S_1$  is strictly less than  $2^{\sqrt{2k}}$ . The set  $S_2$  is calculated from  $V - \{x\}$ , so we can invoke the inductive hypothesis to establish that the number of procedure calls in the calculation of  $S_2$  is strictly less than  $(n-1) \cdot 2^{\sqrt{2k}}$ . The total number of procedure calls is  $1 + \#calls(S_1) + \#calls(S_2)$ , which is no greater than  $2^{\sqrt{2k}} + (n-1) \cdot 2^{\sqrt{2k}} = n \cdot 2^{\sqrt{2k}}$ .

*Case 2.*  $|\bar{E}(x)| \geq \sqrt{2k}$ . Since  $x$  is a vertex with fewest missing edges, we are assured that  $\sum_{v \in V} |\bar{E}(v)| \geq n \cdot |\bar{E}(x)| \geq n \cdot \sqrt{2k}$ . We also know that the sum of the missing edges over all vertices will count each missing edge twice, so  $\sum_{v \in V} |\bar{E}(v)| = 2k$ . Combining these observations, we have  $2k \geq n \cdot \sqrt{2k} \Rightarrow \sqrt{2k} \geq n$ . This bound on  $n$  allows us to bound the number of procedure calls at no more than  $2^{\sqrt{2k}}$ , which is no greater than  $n \cdot 2^{\sqrt{2k}}$ .

Having successfully bounded the number of procedure calls, we observe that each activation of the procedure performs a fixed number of operations on  $\bar{E}$ , and no operation takes more than  $k^2$  steps. So the total number of steps for *MaxIS* is less than  $cnk^2 \cdot 2^{\sqrt{2k}}$  for some constant  $c$ . Since  $k$  cannot exceed  $n(n-1)/2$ , we can express the upper bound as  $p(n) \cdot 2^{\sqrt{2k}}$  for some polynomial  $p$ , which is  $2^{O(\sqrt{k})}$ . The input to the algorithm has size  $n + k$ , and certainly  $\sqrt{k}$  is less than  $\sqrt{n+k}$ . So the total number of steps is  $2^{O(\sqrt{n+k})}$ .

*Theorem 2.* The power index of INDEPENDENT SET is one-half when a graph is represented by a non-adjacency list (under the assumption that SAT has index one).

*Proof.* Theorem 1 implies that the upper bound on the power index of INDEPENDENT SET is one-half when a graph is represented by a vertex set and a non-adjacency list. Reductions from SAT can be used to establish a lower bound for the power index. A sequence of textbook reductions can be applied to an instance of SAT to derive an instance of INDEPENDENT SET. The total step count is a sum of polynomials, and the size of each representation along the way is a linear function of the size of the previous representation. The entire process comprises a linear size reduction from SAT to INDEPENDENT SET. The final step, however, produces an adjacency list for a graph. When the adjacency list of size  $e$  is converted to a non-adjacency list, the size of the new representation is  $O(e^2)$  in the worst case, and the composition of a linear reduction with conversion to a non-adjacency list results in an  $O(n^2)$  reduction from SAT to INDEPENDENT SET where  $n$  is the length of the original expression. This reduction establishes that one-half is a lower bound for the power index of INDEPENDENT SET using the non-adjacency list representation.

Thus the result that CLIQUE has index one half can be duplicated for INDEPENDENT SET if we represent the graph with a non-adjacency list instead of an adjacency list. Can we claim that a non-adjacency list is less efficient than an adjacency list? Figure 1 contains a comparison of the space requirements for adjacency lists, non-adjacency lists, and adjacency matrices. Adjacency lists are most efficient for sparse graphs, all three representations are equally efficient for balanced graphs, and non-adjacency lists are most efficient for dense graphs. We can claim that the matrix representation is generally less efficient, since it is never better than  $\Theta(n^2)$ . But there is no distinction between the two list representations -- their efficiency characteristics are symmetric.

| Graph type | Edge count    | Missing edge count | Size of adjacency list | Size of non-adjacency list | Size of adjacency matrix |
|------------|---------------|--------------------|------------------------|----------------------------|--------------------------|
| Sparse     | $O(n)$        | $\Theta(n^2)$      | $O(n)$                 | $\Theta(n^2)$              | $\Theta(n^2)$            |
| Balanced   | $\Theta(n^2)$ | $\Theta(n^2)$      | $\Theta(n^2)$          | $\Theta(n^2)$              | $\Theta(n^2)$            |
| Dense      | $\Theta(n^2)$ | $O(n)$             | $\Theta(n^2)$          | $O(n)$                     | $\Theta(n^2)$            |

Figure 1: Space requirements for graph representation.

### 3 Observations on VERTEX COVER

The third problem in the classic trio is VERTEX COVER. A vertex cover in a graph  $G = (V, E)$  is a subset of the vertices that covers every edge. A set covers an edge if either end point of the edge is in the set. There is a well-known and very direct reduction between VERTEX COVER and INDEPENDENT SET. If  $IS \subseteq V$  is a largest independent set in  $G$ , then  $VC = V - IS$  is a smallest vertex cover. This reduction is easy to justify -- there are no edges between vertices in the independent set, so all edges are covered by the vertices that aren't in the independent set. It doesn't matter whether you search for an independent set or a vertex cover -- when you find one, you've found the other.

So the vertex set for a graph can be partitioned into the largest independent set and the smallest vertex cover. It is interesting to consider what set is represented by the vertices that are not in the largest clique. Shouldn't there be a fourth graph problem for finding this set? Actually, it's another vertex cover. But it covers the missing edges, not the edges. This leads to an analysis that eliminates one graph problem from the trio instead of adding a fourth. We need VERTEX COVER and either CLIQUE or INDEPENDENT SET, but not both. We'll choose CLIQUE and adopt the following notation for a graph  $G = (V, E)$ :

- $CL^+$  is the largest clique in the edge set  $E$ ,
- $CL^-$  is the largest clique in the missing edge set  $\bar{E}$ ,
- $VC^+$  is the smallest vertex cover for  $E$ , and
- $VC^-$  is the smallest vertex cover for  $\bar{E}$ .

The vertex set  $V$  can be partitioned into either  $CL^+$  and  $VC^-$  or  $CL^-$  and  $VC^+$ . We have

$$V = CL^+ \cup VC^- = CL^- \cup VC^+,$$

$$CL^+ \cap VC^- = \emptyset, \text{ and}$$

$$CL^- \cap VC^+ = \emptyset.$$

Step counts for a CLIQUE search are highest for a dense graph. Unfortunately, step counts for a VERTEX COVER search are highest for a sparse graph. When posed with the problem of finding  $CL^+$  in a dense graph, there is nothing to be gained by looking for  $VC^-$  instead. It would be much easier to find  $CL^-$ , but not much is known about the relationship between  $CL^+$  and  $CL^-$ . It's a relationship that should be investigated, since it's possible that a search for  $CL^+$  could be expedited by knowledge of what was in  $CL^-$ . Figure 2 below illustrates the relationship between the four subsets of  $V$ .

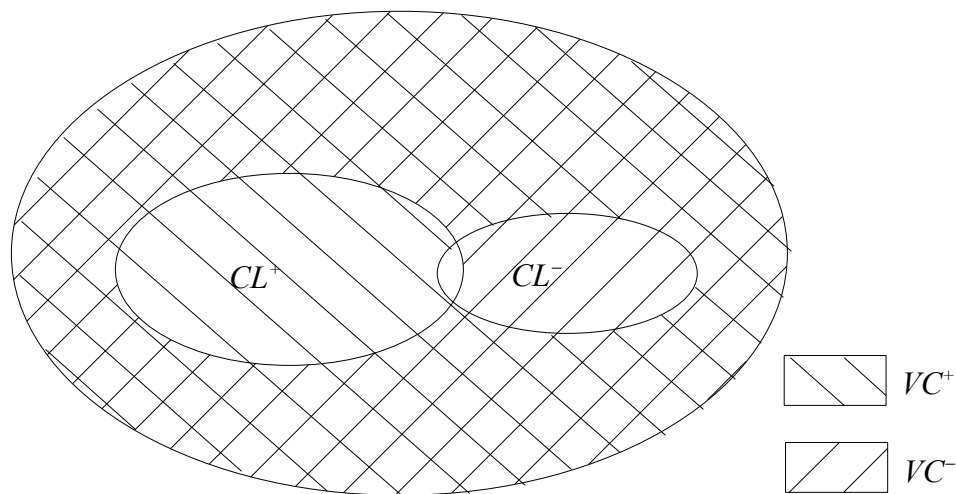


Figure 2: Subsets of  $V$ .

It's interesting that  $CL^+$  and  $CL^-$  can share at most one vertex. If the intersection of  $CL^+$  and  $CL^-$  contained two vertices, then they would have to be both connected to one another and not connected to one another at the same time. In some graphs we can find a  $CL^+$  and a  $CL^-$  that are disjoint. It is unknown whether we can always find a  $CL^+$  and a  $CL^-$  that have a shared vertex. When a graph is dense, we expect  $CL^+$  to be hard to find and  $CL^-$  to be easy to find. We could search for  $CL^+$  by finding  $CL^-$  first, searching the neighborhoods of the vertices in  $CL^-$ , and then searching the graph with  $CL^-$  deleted. It would be interesting to compare this approach empirically with the Stearns and Hunt algorithm [5], which chooses a vertex of lowest degree, searches its neighborhood, and searches the rest of the graph with the low-degree vertex removed.

## 4 Conclusion

The CLIQUE, INDEPENDENT SET, and VERTEX COVER problems have been studied for decades. They are excellent examples of seemingly simple graph problems that are intractable. There may be more to learn about these problems if we are willing to look at the missing edges in the graph instead of or in addition to the edges. We have seen that when the missing edge list is used to represent a graph, the INDEPENDENT SET problem has the same complexity as CLIQUE, and there is no reason to believe that one of these problems is harder than the other. We also speculate that for sparse or dense graphs, there may be an advantage to solving CLIQUE or VERTEX COVER for the edge list and the missing edge list simultaneously, allowing information from the shorter search to expedite the longer search.

## References

- [1] S. Cook, "The Complexity of Theorem-Proving Procedures", *Proceedings of the Third ACM Symposium on Theory of Computing*, ACM, New York (1971), pp. 151-158.
- [2] M. Garey and D. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, Freeman Press, San Francisco, CA (1979).
- [3] R. Karp, "Reducibility Among Combinatorial Problems", in *Complexity and Computer Computations*, ed. R. E. Miller and J. W. Thatcher, pp. 85-103, Plenum Press, New York (1972).
- [4] R. Stearns, "It's Time to Reconsider Time", *Communications of the ACM*, Vol. 37, No. 11 (1994), pp. 95-99.
- [5] R. Stearns and H. Hunt, "Power Indices and Easier Hard Problems", *Mathematical Systems Theory* 23 (1990), pp. 209-225.

# Generative Programming Considerations for the Matrix-Chain Product Problem

Andrew A. Anda  
Computer Science Department  
St. Cloud State University  
St. Cloud, MN 56301  
aanda@stcloudstate.edu

1 March 2008

## Abstract

Many who utilize C++ matrix classes which implement overloaded arithmetic operators are unaware that by allowing the matrix-chain product expressions they code to be evaluated via the multiplicative operator's intrinsic associativity, the performance of the evaluation might be significantly suboptimal relative to the performance of the optimal ordering. We propose that such a matrix class should be augmented to automate the process of identifying matrix-chain products in source code, determining an efficient ordering, and evaluating the matrix-chain using that ordering. We discuss considerations concerning: the influence of special algorithms and matrix types, the completeness of static specifications, the implementation environment, and future language standards.

Andrew A. Anda  
Computer Science Department  
St. Cloud State University  
St. Cloud, MN 56301  
aanda@stcloudstate.edu

# 1 Introduction

## 1.1 Overview

When a specialist in any specific discipline, an application domain, needs to formulate and implement a computational solution to a problem in that domain, they will specify, analyze, formulate an algorithm, then encode that algorithm into some programming environment, translating syntax and semantics from their domain into that of the programming environment. The more correlated the syntax and semantics of the programming language is to the application domain, the more efficient and reliable that process should prove to be, and the less the domain specialist is required to know about how to efficiently and reliably perform that translation to the programming environment. One aspect of the evolution of high-level programming languages is the ongoing goal of further *deskilling* the programmer, by automating routine tasks, raising the abstraction level of the interface, establishing reasonable, common, and expected (overridable) defaults, and enhancing the insulation of the programmer from implementation details. Seminal early high-level programming languages targeted specific broad application domains, e.g. COBOL for business and FORTRAN for science and engineering. Soon thereafter, the goal of more specific domain expression was satisfied via the development of source libraries of related modular subprograms. Examples in the context of numerical linear algebra include the BLAS, EISPACK, and LINPACK. Some years later, the trend emerged to design high-level languages to be primarily general-purpose. For these languages, external facilities for domain-specific specialization are essential. The *class* facility in object-oriented languages representing an abstract data type has evolved from the preceding library paradigm as the means to represent specialized domain knowledge and operability.

C++ evolved from C as an alternative to SIMULA67.[19] First classes, then operator overloading and templates were introduced. Operator overloading furthers the goal of matching the syntax and semantics of a specific application domain to that of the programming language. The operator overloading facility in C++ is not as general as one might want, as the set of overloadable operators is restricted to a well defined subset of the set of intrinsic operators with no allowance for overriding the default precedence and associativity of an operator (e.g. overloading the caret ‘^’ operator for exponentiation yields the wrong associativity). Templates, introduced to support generic programming, accidentally included an ability to perform static (compile-time) computations and even code generation.[25] Templates can contain recursive expressions. In fact, the template layer itself provides Turing-completeness. [22, pp. 312–313] [8, p. 407]

## 1.2 The Matrix Arithmetic Domain

Computational problems in an extensive set of disciplines are routinely solved via at least partial formulations as the solution of matrix algebra problems. A matrix facility in a programming language will better match its targeted domain if the arithmetic operators for addition, subtraction, and multiplication are overloaded for scalars, vectors, and matrices. Whereas matrices and their arithmetic operators are intrinsic in Fortran90 and its successors, in C++ these must be defined as extrinsic classes. Object oriented programming raised

the overall abstraction level, but at the expense of run-time performance.

Templates can provide type *genericity*, i.e. static specializations that allow the arithmetic operators to abstract commonality by processing different types of matrices, with minimal code duplication, uniformly or specially.[2] Additionally, templates can be exploited *generatively* to statically handle special cases and conditionals, evaluate expressions, and optimize code for execution (e.g. temporary reduction or loop unrolling) – this is termed *metaprogramming*. [24, 23, 25, 18, 8, 7, 1] Early matrix class libraries exploited templates for their generic facility only. Subsequent and more contemporary matrix class libraries (e.g. Blitz++, POOMA, MTL, and GMCL) implement templates metaprogramming methodologies to optimize run-time performance on a variety of matrix types.[26, 18, 17, 6] Matrices can multiply inherit a wealth of common properties including

**precision** e.g. single, double, quad

**number algebra type** e.g. integer, rational, real, complex

**symmetries** e.g. symmetric, non-symmetric, per-symmetric, Hermitian

**density** e.g. dense, sparse

**patterned** e.g. diagonal, banded, (upper/lower): triangular, Hessenberg

**storage format** e.g. CSR, CSC, recursive (tiling pattern: RBR, RBC, Hilbert, Z-Morton)

**rank** e.g. full rank, rank-one, rank-two, ...

**structure** e.g. circulant, Hankel, Vandermonde, Cauchy, Toeplitz, Fourier

**special** e.g. Hilbert, Krylov, stochastic

**shape** e.g. square, rectangular

**spectral properties** e.g. SPD

**blocking**

Without generics, a library handling each case would suffer from exponential combinatorial bloat. However, templates can be exploited to manage this complexity automatically. E.g. the GMCL library provides multiple template parameters for the matrix type and feature categories covering 1840 kinds of matrices with roughly 7500 lines of code.[26] Different matrix types and features are handled by a matrix configuration generator which is called by a matrix expression generator wherein overload operators construct expression objects representing the structure of the expression they are used in. The evaluation of the tree of expression objects is handled by the assignment operator.[8]

The overarching principal is that if a property is available at compile-time, it can be exploited at compile-time. Therefore, we can at compile-time perform partial, or even total, expression evaluation.



## 2 Motivation

If users learn about the properties of matrix algebra in a typical linear/matrix algebra course in the mathematics curriculum, they most likely learn that with respect to multiplication, matrices in general don't commute. But, matrices do associate. So, all associative orderings of the *matrix-chain* product of conformal matrices,

$$\prod_{i=1}^n A^{k_i \times k_{i+1}} \quad (1)$$

will give the same answer. However, math students are seldom introduced to the property that in general, for a matrix-chain product of conformal matrices, some of which are not square, associativity does not hold with respect to the total number of scalar operations. It is likely that a specialist, in a domain unrelated to computer science and combinatorics, would not know that by relying on the default left-to-right associativity of the multiplication operator, they might be performing an excessive and sub-optimal number of scalar products. For that reason, this instance of specialized matrix domain knowledge should be integrated into a generative matrix library. That goal is the focus of this paper.

The task of determining exhaustively the optimal (fewest operations) associative ordering becomes infeasible for longer product chains of matrices because the number of possible orderings grows proportional to the sequence of *Catalan numbers* which grow at the exponential rate of  $\Omega(4^n/n^{3/2})$ . [5, pp. 331–349] The optimal ordering can be determined in polynomial time  $\Omega(n^3)$  and space  $\Omega(n^2)$  [16, 10, 5] by performing *dynamic programming* [3] or *memoization* [5, pp. 347–349]. Hu and Shing developed lower complexity optimal solution algorithm with  $O(n \log n)$   $O(n)$  space requirements. [14, 15] Other lower complexity optimal matrix-chain algorithms have been developed which require either parallel processing, or finding a sufficiently optimal approximate solution. [4, 9]

## 3 Proposal

The preceding section motivates a proposal for research and development towards the following objective: create an C++ facility which

1. identifies matrix-chain products in source code,
2. determines an efficient ordering,
3. evaluates the matrix-chain using that ordering,

via C++ template metaprogramming.

### 3.1 Considerations

#### 3.1.1 Static/Dynamic

The proportion of work that can be performed at compile-time will be dependent on the degree of completeness of the set of parameterized features at compile-time. If any of the

matrices are dynamic or if any array slice dimensions are only known at run-time, then the class can perform the full ordering determination only at run-time. However, fully static subchains can be processed statically for optimality as any optimization problem amenable for solution via dynamic programming exhibits *optimal substructure*, i.e. an optimal solution comprises optimal subproblems.[5, p. 339]

The class will establish a default algorithm for the ordering determination. As it is likely that most matrix-chain products will be short, the simplest efficient algorithm should suffice. More efficient ordering algorithms, if they are part of the class, could either be selected through declaration by the user, or could be selected conditionally based on chain length. Additionally, a user could select an approximating algorithm. A facility whereby the user may optionally provide their own algorithm via functor would add desirable extensibility.

### 3.1.2 Binary product operation counting

Current examples of implementation of the matrix-chain product assume that the standard cubic complexity binary matrix product is applied to a fully dense and general rectangular matrix. However, if more specific characterizations of any of the two operands of a binary matrix product such as those itemized in the far-from-exhaustive list [1.2] of matrix types, those characteristics can be exploited by a more efficient algorithm yielding lower operation counts and/or execution rates than the standard general product algorithm. Even general matrices can be multiplied more efficiently, and with a lower computational complexity,(if square) via one or more recursive applications of the Strassen-Winnograd matrix product algorithm.[11, 13] Similarly, the more efficient 3M algorithm can be used for general complex matrix products.[13, pp. 437–438] If accurate operation counts for the products of special matrices and algorithms are provided to the ordering optimizer, then the optimal ordering may change. Therefore there should be a facility which provides a reliable exact or approximate operation count for any pair of matrices and product algorithm on those matrices. The operation counts could be generated *a priori* or alternatively, benchmarking can be used to generate functions which estimate runtime performance. Because of the significance of locality on modern computer architectures for algorithm performance, the relationship between operation count and performance is not as tightly coupled as it was pre-NUMA.

### 3.1.3 Implementation environment

Rather than start from scratch, this project would be more feasibly performed by augmenting an existing metaprogrammed C++ matrix library. Although, a stand-alone class could be feasible and practical if it is restricted to ordinary dense general matrices and the standard product algorithm. This might be a good target for a proof-of-concept project. Because the current template facility is Turing-complete, it is certainly powerful enough to perform everything we propose. However, certain essential generator and type related facilities have to be performed rather awkwardly and opaquely in the current C++ standard. The next major revision of C++ [C++0X] will add critical syntax and semantics (e.g. *concepts*) that will permit a more elegant implementation of generic and generative metaprogramming techniques.[12]

### 3.1.4 Beyond matrix-chain products

A more general project would be to parse matrix expressions to extract other optimizable sub-expression types. E.g. matrix polynomials of the types

$$\sum_{i=1}^n a_i X^i \quad (2)$$

$$\sum_{i=1}^n A_i x^i \quad (3)$$

$$\sum_{i=1}^n A_i X^i \quad (4)$$

can be evaluated efficiently via Horner's method in conjunction with the binary power algorithm. However, equation [2] can be evaluated more efficiently than by Horner's.[13, pp.102–103] [16, 21] In fact, any function of a matrix can be extracted and evaluated with some being identified as having more efficient solution algorithms known. [11, Chap. 11] Additionally, existing generative matrix arithmetic and algebra libraries can be augmented and extended to include any of the almost inexhaustible supply of not-yet-handled matrix types for which efficient product algorithms are known.

## 4 Summary

Because many who use matrix classes (having overloaded arithmetic operators) are unaware of the criticality of the associative ordering of the matrix-chain product expressions they code, we propose that the class should be augmented to automatically identify and evaluate the matrix-chain product using the optimal or a close-to-optimal ordering. This objective can be implemented via C++ template metaprogramming. This project can be partitioned into a set of coupled sub-projects. Extensions of this project to specific matrix types and expression types could spawn a wealth of related projects.

## References

- [1] ALEXANDRESCU, A. *Modern C++ Design: Generic Programming and Design Patterns Applied*. Addison Wesley, Boston, MA, USA, 2001.
- [2] BARTON, J. J., AND NACKMAN, L. R. *Scientific and Engineering C++: An Introduction with Advanced Techniques and Examples*. Addison Wesley, Reading, MA, USA, 1994.
- [3] BELLMAN, R. E. *Dynamic Programming*. Princeton University Press, Princeton, New Jersey, USA, 1957.
- [4] CHIN, F. Y. An  $o(n)$  algorithm for determining a near-optimal computation order of matrix chain products. *Commun. ACM* 21, 7 (1978), 544–549.

- [5] CORMEN, T. H., LEISERSON, C. E., RIVEST, R. L., AND STEIN, C. *Introduction to Algorithms*, second ed. The MIT Press, Cambridge, MA, USA, 2001.
- [6] CZARNECKI, K. *Generative Matrix Computation Library home page*. 2006, <http://nero.prakinf.tu-ilmenau.de/~czarn/gmcl>.
- [7] CZARNECKI, K., EISENECKER, U., GLÜCK, R., VANDEVOORDE, D., AND VELD-HUIZEN, T. Generative programming and active libraries (extended abstract). In *Generic Programming. Proceedings (2000)*, M. Jazayeri, D. Musser, and R. Loos, Eds., vol. 1766 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 25–39.
- [8] CZARNECKI, K., AND EISENECKER, U. W. *Generative Programming: Methods, Tools, and Applications*. Addison Wesley, Boston, MA, USA, 2000.
- [9] CZUMAJ, A. Very fast approximation of the matrix chain product problem. *J. Algorithms* 21, 1 (1996), 71–79.
- [10] GODBOLE, S. S. On efficient computation of matrix chain products. *IEEE Transactions on Computers* C-22, 9 (1973), 864–866.
- [11] GOLUB, G. H., AND VAN LOAN, C. F. *Matrix Computations*, third ed. The Johns Hopkins University Press, Baltimore, MD, USA, 1996.
- [12] GREGOR, D., JÄRVI, J., SIEK, J., STROUSTRUP, B., REIS, G. D., AND LUMSDAINE, A. Concepts: linguistic support for generic programming in c++. *SIGPLAN Not.* 41, 10 (2006), 291–310.
- [13] HIGHAM, N. J. *Accuracy and Stability of Numerical Algorithms*, second ed. the Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, USA, 2002.
- [14] HU, T. C., AND SHING, M. T. Computatation of matrix chain products. part i. *SIAM Journal on Computing* 11, 2 (1982), 362–373.
- [15] HU, T. C., AND SHING, M. T. Computatation of matrix chain products. part ii. *SIAM Journal on Computing* 13, 2 (1984), 228–251.
- [16] PATTERSON, M. S., AND STOCKMEYER, L. J. On the number of nonscalar multiplications necessary to evaluate polynomials. *SIAM Journal on Computing* 2, 1 (1973), 60–66.
- [17] SIEK, J., AND LUMSDAINE, A. Software engineering for peak performance. *C++ Report* (May 2000), 23–27.
- [18] SIEK, J. G., AND LUMSDAINE, A. The matrix template library: Generic components for high-performance scientific computing. *Computing in Science and Engineering* 1, 6 (Nov/Dec 1999), 70–78.
- [19] STROUSTRUP, B. *The C++ Programming Language*, third ed. Addison Wesley, Reading, MA, USA, 1997.

- [20] UEBERHUBER, C. W. *Numerical Computaton 1: Methods, Software, and Analysis*. Springer-Verlag, New York, NY, USA, 1997.
- [21] VAN LOAN, C. F. A note on the evaluation of matrix polynomials. *IEEE Trans. Automat. Control AC-24*, 2 (1979), 320–321.
- [22] VANDEVOORDE, D., AND JOSUTTIS, N. M. *C++ Templates: The Complete Guide*. Addison Wesley, Boston, MA, USA, 2003.
- [23] VELDHUIZEN, T. L. Expression templates. *C++ Report 7*, 5 (1995), 26–31.
- [24] VELDHUIZEN, T. L. Using c++ template metaprograms. *C++ Report 7*, 4 (1995), 36–43.
- [25] VELDHUIZEN, T. L. C++ templates as partial evaluation. In *Proceedings of PEPM'99, The ACM SIGPLAN Workshop on Partial Evaluation and Semantics-Based Program Manipulation*, ed. O. Danvy, San Antonio, January 1999. (Jan. 1999), University of Aarhus, Dept. of Computer Science, pp. 13–18.
- [26] VELDHUIZEN, T. L., AND GANNON, D. Active libraries: Rethinking the roles of compilers and libraries. In *Proceedings of the SIAM Workshop on Object Oriented Methods for Inter-operable Scientific and Engineering Computing (OO'98)* (Yorktown Heights, New York, 1998), SIAM Press.

# Simulation of Nitrogen Flow Using the St. Olaf Beowulf Cluster

Anthony Waldschmidt  
St. Olaf College  
Class of 2008  
waldschm@stolaf.edu

Dr. Richard Brown  
Computer Science  
St. Olaf College  
rab@stolaf.edu

Dr. John Schade  
Biology  
St. Olaf College  
Northfield, MN 55057

## Abstract

Riparian zones, or strips of vegetation located adjacent to flowing water, are vital ecosystem resources that benefit surrounding areas primarily by managing nitrogen runoff. Although the mechanism is unclear, researchers have demonstrated an empirical relationship between riparian plant root mass and levels of denitrification [3]. Denitrification is the export of nitrogen to the atmosphere by riparian plants and is the mechanism utilized to mitigate nitrogen runoff. Dr. Schade simulated these nitrogen dynamics and discovered that riparian zones appeared to fail at high nitrogen loads.

In order to explore the generalizability of this finding, Dr. Schade's model of nitrogen flow[4] was translated into the C language and adapted to run on the St. Olaf Beowulf cluster: Helios. An implementation of MPI was utilized to parallelize the algorithm once it had been translated to C.

This implementation was the first attempt to parallelize an existing scientific model on the helios cluster. A primary concern therefore was to establish guidelines and advice for repeating the process in the future. As expected, the implementation presented a plethora of technical and conceptual challenges.

The results of these cluster runs indicated that a wide array of riparian plant species with varying physiological characteristics experience this breakdown of riparian zone function at high nitrogen levels. Future research is needed to verify these findings and also to potentially expand this research to an ecosystem level (riparian linking) and into the classroom.

# 1 Introduction

In order to provide a meaningful understanding of the aims of this research project, it is necessary to briefly explain the basic ecological unit under investigation: the riparian zone. Riparian zones, as mentioned previously, are strips of vegetation adjacent to a body of continuously flowing water. Riparian vegetation may occur naturally, but in many cases farmers and natural resource managers plan and preserve these zones to act as biofilters for nearby agricultural fields, buffering the flow of excess nitrogen runoff.

The two processes that define the outcome of the nitrogen cycle in these zones are denitrification and plant uptake. Denitrification is the process of converting organic nitrogen to its atmospheric form at which point it escapes the ecosystem. While soil bacteria are the organisms actually responsible for this process, riparian plants modulate denitrification by virtue of their root mass. A low root mass will stimulate denitrification and thereby raise nitrogen export. Conversely, if root mass is high, denitrification will be lowered along with nitrogen export. Plant uptake, a process in which the plant draws in nitrogen through the roots and incorporates it into its tissue, has a direct relationship with root mass unlike denitrification.

The major concern presented by this model is the feedback cycle predicted when nitrogen runoff is too high. Under these circumstances, denitrification plummets, the riparian zone's nitrogen export is drastically reduced, and excess nitrogen unfortunately flows into the adjacent body of water. Dr. Schade uncovered this phenomenon during an initial, single-processor simulation[4]. The goal of this investigation was to utilize the St. Olaf Beowulf cluster to expand the preliminary sensitivity analysis and examine the generalizability of the discovered riparian failure.

## 2 Implementation

### 2.1 Model Types

In order to isolate the impact of the plant's response in terms of its nitrogen uptake, two model types were implemented. The first was the control type in which the plant was not allowed to adjust root size or tissue nitrogen levels. This constant control facilitated a comparative analysis against the response group in which the plant was allowed to respond physiologically.

### 2.2 Translation and Parallelization

Progressing from a model in a paper to a fully functional, parallel program presents a range of technical and conceptual challenges. As a result, a primary concern of this project was to establish guidelines for parallelizing future scientific papers and models on the Helios cluster. The created guides are available on the wiki for the St. Olaf Beowulf project [2]. This process consisted of two major stages.

The first stage was algorithm comprehension and extraction. Losing perspective in the scientific minutiae of the model was a potential concern; however, it was assumed that understanding the scientific basis of the model would be vital to correctly implement the model and to maximize the efficiency of our runs. After careful examination of Dr. Schade's paper presenting the model [4], it became clear that some of the details of the algorithm were not articulated with sufficient detail to easily repeat the implementation process. This problem was solved when Dr. Schade provided us with the original MATLAB files used in the single-processor simulation.

The second stage of this process was the actual coding and parallelization. Dr. Schade's model was translated from MATLAB to C and was parallelized with MPI. The parallelization component consisted of dividing the parameter space equally, sending the appropriate parameter space to each node on the cluster, processing this space, and then returning the data to the head node for further analysis.

### **2.3 Technical Complications**

During the course of the implementation process, which spanned from spring to fall during 2007, one major technical complication arose.

Aside from the relatively high downtime and frequent wipes of the cluster during the spring, a major technical obstacle did not arise until multi-processor runs were attempted. The issue appeared to be with the MPI software package installed on the experimental Castaway cluster. Although the model had been correctly implemented using the MPI framework, the `mpirun` command refused to correctly allocate runs to more than a single processor. This issue was ultimately resolved by a migration to the Helios cluster in Fall 2007. Sun Grid Engine was installed on this cluster and facilitated easy scheduling and multi-processor runs.

### **2.4 Conceptual Complications**

Two unexpected complications arose during the implementation of this model. Parameter space sampling was the first dilemma we encountered. The 6 parameters we were varying in our model runs unfortunately had no readily available natural limits. Our initial approach to sampling was to perform a geometric progression ranging from 25% to 400% of our baseline value for each parameter. Nothing in the nitrogen flow system dictates a geometric pattern and this progression choice was therefore somewhat arbitrary. Our plan to address this issue in the future is to use a series of geometric progressions to find a viable parameter space before performing a uniform linear progression across this space for comparison.

The second unforeseen complication we ran into involved early termination of the model. Termination occurs either after five years, or when plant productivity reaches a steady state. After performing several test runs in which a large proportion of the steady state model sets returned NaN (not a number) values, further investigation revealed that these outputs were occurring in cases where steady state was reached before denitrification ever started. It



makes sense that this would occur in cases where the parameter for maximum plant productivity  $P_{\max}$  was too low to allow the plant to physiologically respond to low nitrogen. However, we would like to investigate this phenomenon further to discover which other types of parameter sets may cause this early termination.

### 3 Model Results

After implementing Professor Schade's model in the C language and parallelizing the algorithm using the MPI interface, a large set of simulations were conducted on the Helios cluster over the course of several weeks in the Fall of 2007.

Each parameter's interval was sampled using a geometric progression ranging from 25% to 400% of each parameter's base value (provided by Dr. Schade). The initial results of this investigation appeared to uphold Dr. Schade's findings.

7,086,244 parameter sets were simulated creating approximately 43GB of output data. If each parameter set's simulation was allowed to run for 5 years, 100% of these sets exhibited the switch between feedback types at high nitrogen levels. If these sets were instead run until steady state was reached (defined as a state where plant productivity changes by less than .001 / day), 97.98% exhibited the expected feedback switch observed by Schade.

### 4 Conclusions

Dr. Schade's research into the dynamics of nitrogen nutrient cycling represents an area of pressing concern in ecology. Riparian zones are vital instruments for the mitigation of nitrogen runoff from human sources and are exploited for this purpose worldwide. The implications of riparian failure at a certain nitrogen threshold are far-ranging.

The 'Conceptual Complications' section of this paper has already outlined two areas for potential future exploration: new parameter space sampling methods and investigation of early steady state termination. These are two concerns that will need to be addressed before this research can be published and applied to current landscape planning practices.

Research may also extend this nitrogen model in new directions. Spencer Debenport's research during the '07 summer session illustrates this potential. Spencer's work aimed to extend my initial inquiry by simulating linkages between sequential riparian plants [1]. This project may be useful in discovering dynamic interactions between plants and understanding the wider ecosystem as a whole.

### References

- [1] DEBENPORT, S., BROWN, R., AND OTHERS. Application of beowulf cluster computing to problems in biology.

- [2] PROGRAM., S. O. C. S. Beowiki. Retrieved March, 2008 from <http://devel.cs.stolaf.edu/projects/bw/wiki.real/index.php/Docs> (2008). Excerpt of Beowiki Documentation.
- [3] SCHADE, J. D., FISHER, S., AND GRIMM, N. The influence of riparian shrub on nitrogen cycling in a sonoran desert stream. *Ecology* 82, 12 (2001), 3363–3376.
- [4] SCHADE, J. D., AND LEWIS, D. B. Plasticity in resource allocation and nitrogen-use efficiency in riparian vegetation: Implications for nitrogen retention. *Ecosystems* 9 (August 2005), 740–755.

# The Player is Always Right

Stephen Marquis  
Department of Computer Science  
Lawrence University  
Appleton, WI 54912-0599  
stephen.g.marquis@lawrence.edu

## Abstract

Working on a game design for any project can seem daunting. Finally putting together your work is a great success, until you see the final result. It does not feel right. The main question: what is wrong? The problem can be attributed to different problems: poor implementation, design flaws, or even incorrect modeling. In this paper, we aim to focus on problems in design that may slip notice, even for a well thought-out and organized project. We point out common aspects of design affected by differing ideas of intuition. Intuitively game designers think a well-modeled world effectively translates into good gameplay, but we suggest a flaw in this thinking. Computers are very precise and accurate, to a degree which humans are not. This paper points out several examples of this idea apparent in many modern game styles, as well as specific examples of this problem in our work.

# 1 Introduction

In game design, one of the most important aspects is player intuitiveness. Everything must feel natural perceptually. One of the larger conflicts that can arise from this necessity is human versus computer sensations. As developers we trust in the exactness of computers and almost feel that it is natural to be extremely precise in how we model ideas. This does not account for human feeling, however, and leaves much to be desired in the player experience for a game. Humans are not exact creatures and are imprecise by default. This must always be considered when diving into the world of game development.

Over the course of ten weeks we developed a 3-dimensional rendition of the classic game *Asteroids*. For the first half of the project, we did an in-depth study in the OpenGL graphics library [4] and common techniques used within. The focus was then shifted to game design and world modeling. When first working on the 3D asteroids project, we thought that a lot of the design would be a straightforward modeling problem. The controls turned the ship or gave it an immediate acceleration and everything was simulated in a 3D environment. We figured the rest should take care of itself. Once this was all coded up, however, it did not feel right at all. Emulating the real world exactly cannot work, because in real human experience there are three other senses which are not being triggered during a game. The three main subjects we will discuss that are affected by this need for a human feel are: reference frames, computer reaction times, and camera control.

## 1.1 Structure

we will first provide a brief background on the asteroids project that this paper is largely based upon, before discussing the main point. The first design element we considered was how the game world should be modeled in 3D. Since the objects in the game had to wrap at the world edges, we chose to model the world in the Real 3-dimensional Projective Space ( $RP^3$ ). This way the player would not experience any invisible walls in the game, but instead have a finite game world which wraps around, to play in.

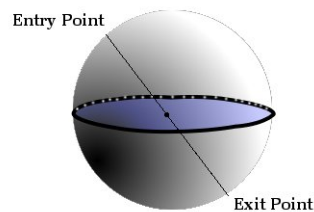


Figure 1: Any object leaving the sphere will reemerge on the other side.

This allowed us to place the ship at the center of a sphere, and have all movement calculations be relative to the non-moving ship. Once this was done we constructed the ship and asteroid models. We then began on the other design aspects of the game. Movement was

done through mouse-controlled turning and forward movement through the space bar. We will go into further detail on the aspects of movement in the following sections.

## 2 Reference Frames

The first basic thing that must be coded is a model of the 3D world. This model should have something more than a direct translation to the world in it; for example, any sort of movement in the game. The programmer's intuition may be that if you have a 3D world and move the models through that world, the player will experience movement. This is not necessarily the case. Some sense of movement in humans occurs physically even with their eyes closed, which cannot be done while playing a game. Thus the need for much clearer reference frames arises. For any visual reference frame, every possible direction of movement must be taken into account. If a player cannot explicitly see evidence of movement in each direction, then the experience will not feel natural to the player.

Take as an example walking down a textured hallway in some game. A simple design decision such as the orientation of the texture on the wall can help improve the feel of the game. If the walls are textured with designs that do not change with horizontal movement along the hall, then the feeling of movement is greatly reduced. Even if you can see the end of the hallway, the speed at which it is approached is much harder to recognize, because you have no further visual cues. Now if one added varying cracks in the wall, or even vertical lines as the texture, suddenly there is always movement on the screen. This in turn gives the player the feeling of true movement. This was the first important lesson we learned.

For *Asteroids*, the problem is even worse, because there were not even walls that you pass by. With just the asteroids floating in the distance there is no way to tell how far away they are and thus how fast everything is actually moving. It was not apparent immediately what was wrong. This seemed to model the vacuum of space correctly, which was our original goal. In looking at other space fliers, we saw that they are easy to use and did not seem to have any special attributes, however, there were always wisps of clouds, or broken debris in the air – something that is a consistent size, and common enough that one can judge distance to said object. To deal with this problem we developed a particle system. Succinctly, “A particle system is a collection of individual elements...that act mostly autonomously that is they don't care about what other particles are doing”[1]. Our solution to the problem was to add a particle system of white particles randomly distributed around the world, as seen in Figure 2. Now movement through space was always apparent, and there was no loss of data from the model's conception to the user.

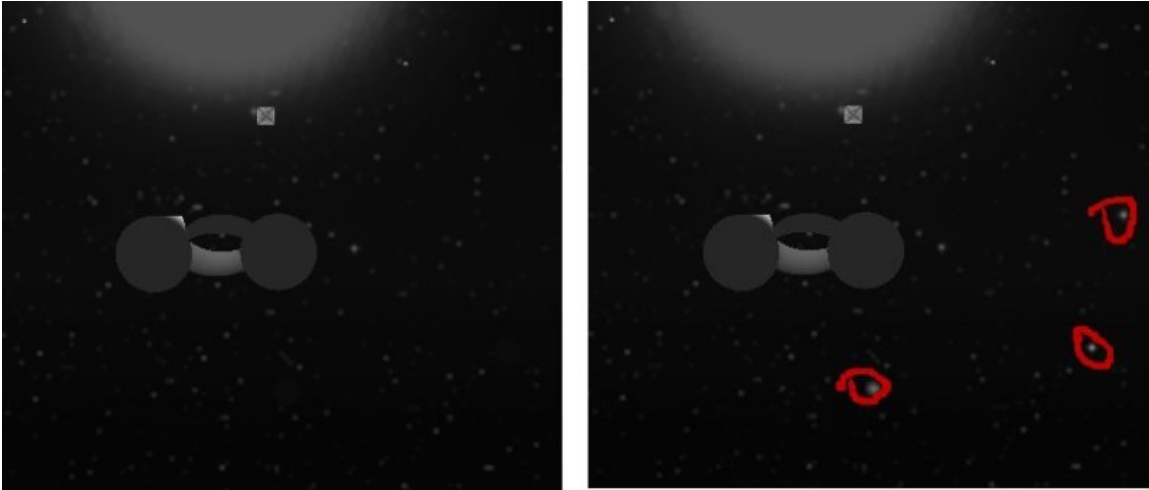


Figure 2: Comparison: In the second image, the circled points were particles in the 3D space, to accentuate movement.

### 3 Reaction times

In programming a game it is natural and usually easier for one to implement a design that does not take human ability into account; however, this does not always create a coherent interaction between what is happening and what the player feels in the game. For example, take any game that has computer opponents that are supposed to be the same as the player's character. The easiest design would be to give all available information in the environment to this opponent, and through some manner of search and heuristic find the best move to make. While any other approach may be difficult to implement and a less desirable design choice, this means information can be calculated incredibly fast. Any player of this game will immediately notice this gap in knowledge, however. From the information given on the screen, one cannot possibly have as much data as the opponent does. This gives an immediate disadvantage to the player as well as an overtly automated feel to the game. To apply a player intuitive approach to the design, one must consciously reduce reaction times. In a player's intuition, any other players in the game should appear to take time to think about things. This is especially evident in first person shooters. Even if we have a situation where the opponent is only updated every frame of the game in a game with sixty frames per second, the computer only takes one sixtieth of a second to react to new surroundings. If this involved all correct decisions, then it is not a well-designed game.

There are two approaches to this problem: constrain the data available to the opponent, or delay reaction times. By constraining data, the computer does not have the ability to optimally react to every stimuli. While this does not mean that the calculations will take longer, they will not be as accurate due to the constraints, and thus possibly require multiple calculations to get a full understanding of the situation. By delaying reaction times, one more closely approximate human behavior. Even a simple hack of giving a half-second delay between calculation and implementation would give the player a much better feel towards the game.

## 4 Camera Movement

Another example of where a different viewpoint is needed in game design is camera movement. In our design of *Asteroids*, we have a system which rotates the camera around the ship based on the location of the mouse. Initially, this rotation was too sudden. When we move the mouse back to the center, the ship stops turning immediately and no feedback is given to that effect. In contrast, when one turns a car, the driver does not keep his or her head locked straight ahead as the car turns, but instead looks back in order to get a broader view during the turn. We were curious about why things still looked so unnatural when turning, and it was this lack of familiarity with the turning process.

Take a look at any well-produced game and the camera has a distinct lag behind the player. When the player finishes turning, the camera catches up to them. It can be seen in earlier games that this was lacking. For example, in early first person shooters, there was no head-bob effect[3]. While many may claim that the games appear lower quality due to today's graphics standards, we see a larger factor in the way these effects were streamlined, whether there are more polygons or not. This blurs the line of when movement precisely stops. By using this approximation effect, we can have a huge impact to playability and once again the intuitiveness that a player feels when playing a game.

Another closely related aspect of camera movement is dealing with movement parallel to player direction. This adds even more to the aforementioned movement sensation. If the camera is just a fixed entity around the player, then it is too rigid. It must react to acceleration and show a noticeable lag between player control and display.



Figure 3: The overlaid(lighter) image shows the ship when not accelerating. Notice the camera is zoomed out on the accelerating ship.

When the player hits the space bar in *Asteroids*, an acceleration is given to the ship. With the particles, motion was much easier to spot and allowed for a constant appraisal of speed. Acceleration however is a different matter. The change in motion is a shorter process, and needs to be accentuated to catch the player's attention. We added a simple animation that pulled the camera away from the ship while accelerating. This simple technique allowed for the game to announce what was happening without being distracting.

## 5 Conclusion

We seek to instill in beginning developers the importance of usability and player feel. We must always consider the approach from a player's point of view. By making experiences explicit(which in the real world may be merely implicit), we are approaching design from a player's point of view rather than a programmer's. By taking this approach game design becomes a job for the consumer and producer, creating more realistic and engaging games.

## 6 Acknowledgments

Many thanks to the Provost and Dean of Faculty, David Burrows, for making it possible to present this paper and attend this conference. Thanks to Professor Joseph Gregg for advising me throughout the project and to Professor Kurt Krebsbach for reading and correcting many drafts of this paper.

## References

- [1] ASTLE, D., Ed. *More OpenGL Game Programming*. Thompson Course Technology PTR, Boston, MA, 2006.
- [2] HILL, JR., F., AND KELLEY, JR., S. M. *Computer Graphics Using OpenGL: Third Edition*. Pearson Prentice Hall, Upper Saddle River, NJ, 2007.
- [3] ID SOFTWARE. *Wolfenstein 3d*. Apogee Software, 1992.
- [4] SHREINER, D. *OpenGL(R) Reference Manual: The Official Reference Document to OpenGL, Version 1.4*. Addison-Wesley Professional, Reading, Massachusetts, 2004.



# Chess AI

Josh Odom  
Computer Science  
University of Wisconsin - Parkside  
Kenosha, WI 53141  
odom0002@uwp.edu

## **Abstract**

This was a research project to study game theory as it applies to computer chess, to use that theory to create a working implementation of a chess AI with a graphical interface, and to study the various efficiencies that could be applied to it.

## 1 Introduction

The purpose of this research project was to study the various aspects of Artificial Intelligence related to the problem of playing chess and to create a working chess AI. Over the course of the project, I was able to create a working implementation of a chess AI complete with a graphical front-end. I also did a significant amount of research on the various efficiencies that could be implemented on the program, the strengths and weaknesses of such efficiencies, and the changes to the code necessary to implement them.

## 2 Chess Theory

The reason that the game of chess can be solved by a computer is because it is a solvable finite problem. The game of chess can be thought of as a tree of possible moves with each move node having a finite number of children and each leaf node representing a game-over position. However, since the game-tree complexity of chess has a calculated lower bound of  $10^{120}$  [5], a full tree evaluation would take an astronomical amount of time using any conceivable hardware. Because of this, the process used by computers is to evaluate the game tree only to a certain depth from the current node, and then to make an educated guess about the strength of the leaf nodes of the search. The educated guess is typically an evaluation of the static board made by using heuristics to rate each player's material and the placement of their pieces. Thus, the strength of a chess program will be limited only by its search depth and static evaluation strength. These are both only limited by the time it takes to process them. Therefore, the principal challenge of building a computer chess AI is one of efficiency.

The minimax algorithm is used to search a game tree for the optimal move. It will find the best possible ending position (for the search depth) for the current player assuming that both sides make the strongest move they can. Since the minimax algorithm has to test each of the leaf nodes, and since a typical chess position has a branching factor of around 30 [5], then increasing the search depth by one ply (the move made by one player) will increase the computation time for a single position by a factor of about 30.

Alpha-beta pruning is an optimization to the minimax algorithm which will keep positions from being evaluated when data shows that those positions do not affect the strength of the current possible move. Intuitively, alpha-beta pruning will prune off sub-trees when it can be shown that the sub-tree in question can only be reached if one (or both) of the players make a weaker move than one already evaluated. Once the optimal move has been found, most sub-trees can be proven to be sub-optimal and little time is wasted calculating them.

### 3 Program Design

#### 3.1 Language

The decision of which language to use was a crucial one. My criteria for a language was that it needed to be cross-platform (because of the variety of operating systems I was using to develop in), that it needed to compile to be efficient and optimized, and that it needed to be flexible and modular so the code could easily adapt to new optimizations. I decided that Objective-C fit all three of these requirements. Objective-C can be compiled cross-platform because it can be compiled by GCC, and GCC has many optimization features which can be enabled [1]. Objective-C is a set of extensions to C which give it full object-oriented capabilities [2].

The decision of the language turned out to have advantages and disadvantages. The main advantages were that all of my code that was strictly Object-Oriented was very clean and readable (even if much of the procedural code was arcane) and that I had very few syntactical errors when developing the project. The main disadvantage lay in Objective-C's dynamic message-passing: namely, the expense of calling a method, since Objective-C must determine which method to call on the fly. Since efficiency was only a problem in the bottlenecks of the program, it was straightforward to replace the few frequently-called methods with C-style functions.

#### 3.2 Design Decisions

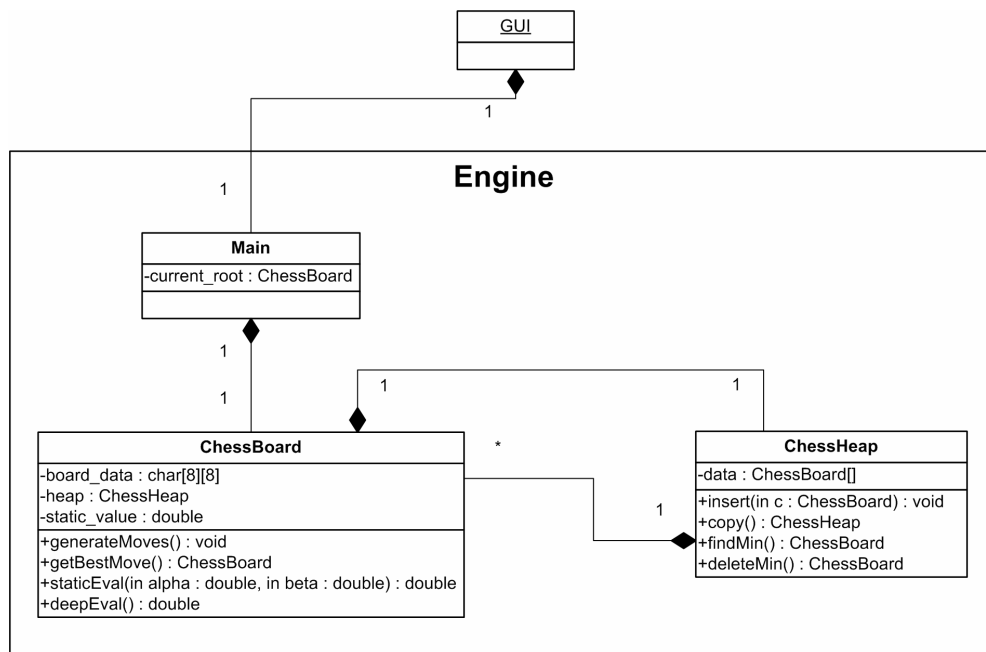


Figure 1: UML Diagram

My original design decisions ultimately led to many of the efficiencies and problems inherent in my program. First, I decided to implement the tree with each node representing a board object in memory, each board holding all positional information and a heap of child positions. If a heap has children, then its static value is that of the optimal choice of its children. If it has no children, its value is based on material. When performing the depth-first minimax search, it starts with the most optimal move as determined by the previous search, so the optimal move for the current search is likely one of the first to be evaluated. This allows alpha-beta pruning to efficiently prune the weaker moves on the tree.

Additionally, the boards are always represented as white's turn to move. This is done by flipping the board vertically between moves and changing the color of each of the pieces. The advantage of this is that only one move-generation method needed to be written. However, the disadvantage is that all of the board data needs to be copied and inverted between turns.

Because the game tree is stored in memory and since each game position contains all of the board data, doing a full 5-ply search can cause the engine to use a full gigabyte of ram. Therefore, this implementation is effectively limited to a 5-ply search, on modern computers.

### **3.3 Program Logic**

The most critical logic of the program is the move-generation code. It is a monolithic method which examines each piece on the board and determines which movement/capture rules apply to that piece. For each possible move, it creates the child board position dictated by the move and adds it to the current board's heap of moves. Normally, the child move only differs from the parent in terms of placement of the board's pieces. However, sometimes the child board also includes information about the move which was made. For example, when a pawn is advanced two spaces from its starting position, this is recorded so that en passant captures are possible. Also, whenever the king or a rook is moved out from its starting position, this is noted so that castling rules are upheld.

Another important section of the program is the logic to determine if the king is in check. This code was introduced late in the development cycle. My original plan was to determine whether a player's king was in check based on the outcomes of each move's child nodes. If any of its children resulted in the king being taken, then obviously the player put him or herself into check, so the current move should be flagged as invalid. Logically, if each board position evaluated possible moves before they gave the value of the position and if any of those moves caused the king to be captured, then the child could propagate a message back to its parent, informing that it was an invalid move. Thus, no invalid moves would be evaluated by the minimax algorithm.

The problem arose when I tried to implement castling. Castling cannot be performed when the king is in check, and check is only determined by whether the king can be

captured if the side in question forfeits a turn. To implement this, I would have needed to create a way to forfeit a turn and a way to do all move generation except castling (to prevent infinite recursion). I decided that it would be more practical and would make my code more readable and efficient to just create standalone logic to determine if the king is in check.

It turned out that the worst bottleneck in the program is the static evaluation code. This is because it tallies up the values of the pieces by checking each space on the board, and it gets called on every node. Code which determines check is called on each leaf node, but need not check each board square. Position-generation code, although more complex and processor-intense than static evaluation, only gets called on non-leaf nodes. Since I use only a naïve static evaluation of the board, I still get proper results if I keep track of a board's value by noting value changes made by captures. This bypasses the static evaluation function altogether.

### **3.4 Profiler**

After I completed a basic working game (albeit lacking many features), I determined that the program efficiency (based on its implementation) was essential to the project. In lieu of an actual code profiler, I wrote custom profiling code which performed instruction counts per method. The profiling code is enabled at compile time, so it does not affect performance when profiling is disabled. As useful as it was in measuring program efficiency and locating bottlenecks, it made all of the code much less readable.

### **3.5 Interface**

My initial interface for the engine was a simplistic one, which would textually provide a list of possible move choices and would make the operator choose from that list. Although this worked, it was very tedious. To be able to easily play a game, it required an operator to act as a mediator between the program and a physical chess board. Even so, move entry was slow.

When I started working on the GUI, my first idea was to make the engine to interface with XBoard/WinBoard (a free chess GUI client). That way, I would not have to bother with making any graphical software. However, because my engine's design was inherently incompatible with XBoard/WinBoard's interface, I determined it would be more straightforward to instead write my own GUI as a wrapper for the textual interface which I already implemented. Consequently, the graphical front-end turned out to be a kludge, more than anything.

I wrote the graphical front-end in Java, using the Java Swing API. I chose Java because it's cross-platform and because I've had considerable experience creating Java Swing GUIs. The GUI itself is an 8-by-8 grid of buttons, labeled per piece and colored to look like a chess board. The inter-process communication between the GUI and the engine is done by the GUI spawning the engine as a child process and getting a handle on the engine's I/O. It then sends move instructions to the engine's standard input when the user

makes a move, and the graphical board gets updated when the engine returns with new board data.

## 4 Difficulties

Because of the complexities of the rules of chess, along with the number of calculations needed to produce a single answer, the greatest difficulty I encountered was determining whether or not my code contained logical errors. This is because the program would still produce a legal move even with the errors, but the move would not be optimal. In other situations, the game would allow for invalid moves or would deny valid moves in obscure cases. These difficulties were only overcome through rigorous testing.

I had one such error after I implemented alpha-beta pruning. I had initially overlooked my program's side-swapping (between turns) when I coded the algorithm, and this caused the tree to be pruned based on arbitrary data, instead of being pruned from valid data. Thus, the game would make very foolish moves, and very quickly. However, since this was corrected, the program only prunes branches that can be proven to not contribute to the decided move.

## 5 Further Improvements

I did not implement many algorithmic optimizations to my program, but the testing and the code optimizations I did gave me an idea of how different optimizations will affect the program, and which would be more beneficial than others.

One good modification would be to modify the static evaluation function to pay attention to positional information, instead of merely material. This would result in the program playing much stronger even if no further modifications were done.

As far as implementation goes, it would be best to convert the code to pure C or C++. This would consist of turning all Objective-C methods into C-style functions or C++ classes, which would make method-calling more efficient. The code could be modified so the board stays represented from the same side between turns and merely indicates whose turn it is. Also, moves could be stored as a listing of the pieces that were moved. Thus, the game data would not need to be copied between moves. A single game board would be used to represent all game positions. The board would be modified to reflect the current position in the tree. Additionally, an index could be kept of pieces and their positions so a brute-force search through the board isn't necessary to find the position of a single piece. A transposition table could be kept so that board positions which occur multiple times need only be calculated once.

If all that were done, since evaluating moves would be considerably cheaper, there would be no reason to keep the tree in memory. It would be optimal to implement the MTD( $f$ ) modification to alpha-beta pruning. MTD( $f$ ) efficiently guesses at what the value of the best move is, and when it chooses its value, MTD( $f$ ) finds the proper branch to take.

Because it is guessing at a single value each pass, it can prune any sub-tree which cannot evaluate to that value. This efficient pruning is what leads to the efficiency of this algorithm as a whole [4].

Once this has been implemented, the program will inherently be very scalable. It would be straightforward to make the evaluation section of the code only evaluate a specific child (or specific children) of the root node, and thus you could have multiple threads evaluating at the same time. One could even implement a slave application which can sit on remote computers and can process sub-trees which are sent to them.

Finally, one could complement the rest of the optimizations by adding an opening and end-game database. If all of this is implemented, I predict that the program will be a very challenging opponent.

## **6 Psychological Effects**

I found two interesting psychological phenomena that resulted from this project. First, all of the people who played a full game against the program ended up referring to it as "he" during their game(s). My naïve guess as to why this happens is that my program passes the Turing Test [3] in a way: a chess player interacts with the computer no differently than he/she would interact with a human chess player, especially if a human operator is making moves on behalf of the computer.

Secondly, I have found that after looking at thousands of chess positions without context of logical reasoning (e.g. when debugging), I can no longer bring myself to logically analyze chess moves. I have not determined the cause for this.

## **7 Conclusion**

Because of the complexity of the game of chess, computers will not play a perfect game of chess in the conceivable future. Although my particular implementation is far from optimal, it's a working program which can play a perfect game up to 5 plies ahead. Since the code was modular, it would be a good base for starting future chess AI projects.

## References

- [1] *GCC 4.3.0 Manual*. Free Software Foundation, Inc. 10 Feb. 2008  
<<http://gcc.gnu.org/onlinedocs/gcc-4.3.0/gcc/Optimize-Options.html#Optimize-Options>>.
- [2] *The Objective-C 2.0 Programming Language*. Apple, Inc. 10 Feb. 2008: 11  
<<http://developer.apple.com/documentation/Cocoa/Conceptual/ObjectiveC/ObjC.pdf>>.
- [3] Oppy, Graham and David Dowe. "The Turing Test." *The Stanford Encyclopedia of Philosophy (Winter 2005 Edition)*. Ed. Edward N. Zalta. 10 Feb. 2008  
<<http://plato.stanford.edu/archives/win2005/entries/turing-test/>>.
- [4] Plaat, Aske. *MTD(f)*. 10 Feb. 2008 <<http://home.tiscali.nl/askeplaat/mtdf.html>>.
- [5] Shannon, Claude. "Programming a Computer for Playing Chess." *Philosophical Magazine* 41.314 (1950): 4.



# Course Scheduling with Genetic Algorithms

**Carl Davidson**  
**Computer Science Department**  
**Simpson College**  
**701 North C St. Box 4473**  
**Des Moines, IA 50125**  
**carl.davidson@simpson.edu**

## Abstract

This paper describes a research project using a genetic algorithm to generate schedules for college courses. The task to generate schedules for college courses is a combinatorial constraint satisfaction problem. The constraints include the requirements needed to graduate, the prerequisites of each course, the courses available each semester, the number of semesters before graduation, and the number of courses a student can take each semester. In the algorithm, a generation of schedules is created by permuting an array of requirements and replacing requirements in the array with courses that fulfill them. The algorithm then selects schedules from the population using a rank-based selection method that favors fit schedules. Fitness is determined by the ability to complete the most requirements in the least amount of time. Selected schedules crossover courses with one another to create a new generation of schedules. Schedules in this new generation may then be mutated by switching times courses are taken, removing repeated courses, and adding omitted courses. The algorithm creates new generations this way for a predetermined number of times and returns the best schedule created over all generations.

This paper discusses the organization of the data and the specifics of the basic steps in the genetic algorithm: creating the initial population, computing fitness, selection, crossover, and mutations. The paper also examines the effectiveness of the implemented algorithm through experiments performed under several different college scheduling scenarios. In the experiments, the algorithm demonstrated it could successfully generate several schedules in a short amount of time that would fulfill all requirements needed to graduate in tested scenarios.

## Introduction

Graduating from college with particular majors often requires strategy to finish before a given year. One effective strategy consists of creating a schedule for the next semester, based on what times courses are available, followed by creating possible schedules for each semester afterwards, based on whether courses are offered in the spring and fall, and whether courses are offered only every other year. The schedules for each semester are closely related—the schedules for which course hours are known affect what courses must be taken later, and schedules for which course hours are not known affect what courses must be taken before there is no other time in which the course is available. A common goal among students is to fulfill all requirements to graduate in the least amount of time. Depending on the situation, this goal may override any preference the student has for courses. Rather than continuing to manually create schedules in this manner each semester, a program was developed to perform the task.

Schedule generation is often a complex task without a definite goal. The task is further complicated when accounting for additional constraints that must be considered while scheduling college courses, such as times in which courses can be taken, and prerequisites needed before courses can be taken. As a result, typical approaches are often difficult to implement or inefficient to perform. However, genetic algorithms have been demonstrated to solve problems without easily defined solutions, including those involving schedule generation. It was decided that a genetic algorithm would be used to generate the schedules of college courses in the program.

## Genetic Algorithms

Genetic algorithms are used to find favorable solutions for problems among many potential solutions, in a manner inspired by natural selection. Genetic algorithms were first conceived in the 1960s by John Holland at the University of Michigan, and since then they have been employed in a wide variety of optimization problems, including coordinating robotic cameras, monitoring gas pipelines, developing more effective antennas, developing stealth planes, and decoding secret messages (Goldberg, 1989; R. Haupt & S.E. Haupt, 1998). Though many variations of genetic algorithms exist, the majority of genetic algorithms consist of four main operations: creation, selection, mating, and mutation (R. Haupt & S.E. Haupt, 1998). Each operation hosts a series of development decisions to be made.

In the creation operation, a series of potential solutions is randomly created. Each solution, known as an “individual” or “chromosome,” consists of a string of values that codes for characteristics of the solution. The series is collectively known as a “population.” Multiple development decisions relate to this operation, including how many individuals will be in a population, how chromosomes will represent solutions to the problem, how chromosomes will be generated, and whether to employ advanced techniques used in nature, such as diploidy or multiple chromosomes (Goldberg, 1989; R. Haupt & S.E. Haupt, 1998).

In the selection operation, individuals are selected to become the basis for a new population, using a chosen selection method. Selection favors individuals representing good solutions to the problem at hand, which are said to have higher “fitness.” Development decisions related to this operation include how fitness will be analyzed, and what selection method will be chosen, including the rank-based, roulette wheel, elitist, or tournament methods (Goldberg, 1989; R. Haupt & S.E. Haupt, 1998).

In the mating operation, a new population is created by mixing portions of the individuals selected during the selection operation. The combination of chromosome segments is referred to as “crossover,” the individuals participating in the crossover are known as “parents,” and the individuals produced in the crossover are known as “children,” “siblings,” or “offspring.” Development decisions related to this operation are limited to how many parents may contribute their chromosome to a single sibling, and how chromosomes are combined, such as through a fixed locus, random locus, or random two point locus method (Goldberg, 1989; R. Haupt & S.E. Haupt, 1998).

Finally, in the mutation operation, a percentage of individuals of the new population are randomly selected and alterations are randomly made to their chromosomes. Development decisions related to this operation include what percentage of the population receives mutation, whether individuals with high fitness are excluded from mutation, and how chromosomes are altered during mutation (Goldberg, 1989; R. Haupt & S.E. Haupt, 1998).

The selection, mating, and mutation operations are then repeated until a condition is met. One such repetition is known as a “generation.” Assuming the designed algorithm is appropriate for the problem it is designed for, the section of the population with the best fitnesses will increase with the population. This is summarized in what is known as The Fundamental Theorem of Genetic Algorithms:

$$M(H,t+1) \geq M(H,t) \frac{f(H)}{F} (1-p_1-p_2)$$

M represents the number of individuals in a population with a particular pattern in their solution (or scheme), H is a scheme, t is a population, f is the average fitness of all individuals with a particular scheme, F is the average fitness of all strings in the population, p1 is the probability that any scheme will be destroyed during crossover, and p2 is the probability that any scheme will be destroyed during mutation. In summary, the number of fit individuals will increase with population (Goldberg, 1989). Over many repetitions, the highest fitness in the population should also increase, and an individual should eventually solve the problem to a satisfactory degree.

## **Previous Schedule Generation Using Genetic Algorithms**

While no known research has investigated the use of genetic algorithms in generating college course schedules, some previous research has investigated the use of genetic

algorithms in general schedule generation. The two problems closest in resemblance to the problem at hand are the Traveling Salesman Problem and the Job Shop Scheduling Problem.

In the Traveling Salesman Problem, the most efficient route is determined through a series of cities that must be visited. In the context of schedule generation, the most efficient schedule is determined given a set of required objectives. The most common genetic algorithm used for the Traveling Salesman Problem represents individuals as an array of numbers representing objectives, in which each objective is mentioned only once throughout the array. During the creation method, permutations of a list of objectives are randomly generated to produce an initial population. During the mating method, two individuals in a mating population are copied. The two copied individuals exchange objectives at a random locus, which produces one duplicate objective in both individuals. The individuals then exchange objectives at the locus of one of the two duplicate objectives, which in turn produces additional duplicate objectives. Duplicate objectives are repeatedly exchanged this way until there are no duplicates remaining, at which point two new siblings are created with objectives from both parents. Finally, during the mutation method, pairs of random objectives are exchanged between loci in the chromosome of a randomly selected individual, producing a slightly different individual (R. Haupt & S.E. Haupt, 1998).

In the Job Shop Scheduling Problem, the most efficient schedule is determined given a set of objectives and a set of machines to perform them. Each objective has a set of activities that must be performed to fulfill the objective (Jackson, 1996). There have been many techniques used to solve the Job Shop Scheduling Problem. For instance, in a binary representation of the Job Shop Problem, the sequence of operations is represented by a binary array representing all possible links between operations, with two operations representing the start and stop of the sequence. Crossover could then occur as it does in basic genetic algorithms, provided that it was followed by a function to repair any broken sequences. In a permutation representation of the Job Shop Problem, sequences could be represented by an integer array, where each integer represents the operations themselves. Sequences could then be generated, reproduced, and mutated in a manner similar to the Traveling Salesman Problem (Yamada & Nakano, 1997).

Additional representations of problems could be implemented with the Job Shop Problem, though many such representations are not applicable to scheduling college courses. While the course-scheduling problem investigated here is very similar to the Job Shop Problem, there is only one machine doing the work—that being the student taking the courses. As a result, only the representations of the Job Shop Problem that can represent one-machine scenarios are applicable to the course-scheduling problem.

## College Course Schedule Generation Using Genetic Algorithms

A Java program was developed to generate schedules for college courses using a genetic algorithm. The program consists of several classes relevant to the operation of the genetic algorithm. A driver class, `ScheduleGeneratorDriver`, interprets information from files and user input in a command line interface, then passes the information to the class housing the genetic algorithm, `ScheduleGenerator`. A storage structure, `Set`, stores information relevant to course scheduling in a manner that can be easily rearranged randomly—an important feature when searching for a random course that fulfills a requirement. The class known as `Set` stores objects of the type `SetValue`, of which there are three subclasses: `Course`, `Session`, and `Requirement`.

Objects of the `Course` class represent college courses, objects of the `Session` class represent times at which college courses are available, and objects of the `Requirement` class represent conditions that must be fulfilled before graduation, usually by taking a given number of courses from a given list. The `Course` class stores a string representing a course title, an integer representing the number of credit hours awarded for a course, several Booleans representing when a course is available, a set of courses representing prerequisites, a set of requirements representing what a course fulfills, a set of sessions representing lectures available for a course, and a set of sessions representing labs available for a course. The `Session` class stores an integer representing how many semesters into the future a session will be offered, an integer representing start time, an integer representing end time, and several Booleans representing days of the week in which a session will be available. The `Requirement` class stores a set of courses that can fulfill a requirement, along with an integer for the number of courses from the set that must be taken before a requirement is fulfilled.

### Interface

The program begins by asking the user for the desired/likely number of semesters needed before graduation, the desired/likely number of courses that should be taken per semester, and a number designating whether the current semester is during the fall or spring, on a year whose numerical representation is even or odd. The program then asks the user for the names of three files.

The first file notates general course information. Every one to three lines of this file are dedicated to information on a single course, including the name of a course, credit hours awarded for a course, availability of a course during certain semesters, prerequisites for a course, times for a course's lectures, and times for a course's labs. The second file notates information on the courses needed to graduate with the student's desired major, including the list of courses that can fulfill a requirement, and the number of courses from the list needed to fulfill a requirement. The last file only notates courses that have already been taken. After receiving the three file names, the program processes the information from these files into objects of class `Set`. The program then uses the sets, alongside additional user input, to invoke the genetic algorithm.

For each generation of the genetic algorithm, the program displays the number of generations computed, the highest fitness found in a generation, the lowest fitness found in a generation, and the time it took to compute the current generation. After a desired number of generations, the program displays the best schedules up to a number specified by the user. If the schedules are not satisfactory, the user can choose to continue the process. Upon exiting, the program displays, in a more detailed form, the courses in the best schedule of the last generation.

## **Chromosome Representation**

The representation used for the course-scheduling genetic algorithm most resembles the permutation representation of the Job Shop Problem. The representation was chosen for its previous use in scheduling, its applicability to one-machine scheduling problems, and its similarity to the well studied genetic algorithm used for solving the Traveling Salesman Problem.

In the genetic algorithm, schedules are represented as arrays of objects known as nodes. Each node stores the name of a course, the name of a requirement intended to be fulfilled by that course, the name of a session at which the course can be taken, and the name of a lab the course the course may need. Each name is a numeric value, either determined by converting the string name of a course into a number (such as in the case of courses), or by the order in which they were read from input files (such as in the case of requirements, sessions, and labs). Names are set to zero by default, which bears different meaning depending on what else is stored in a node. Nodes storing class names of zero, termed “blank nodes,” show that the student would have wanted to take a course, but did not. Nodes storing class names of nonzero values, but storing session and lab names of zero, represent a course planned to be taken in a semester in which session and lab times are not known.

Arrays are subdivided into sections representing semesters prior to the student’s anticipated graduation. The size of each section is equal to the desired/likely number of courses to be taken each semester, and the number of sections is equal to the desired/likely number of semesters before graduation. As a result, the size of each array equals the desired/likely number of courses to be taken each semester multiplied by the number of semesters before graduation.

## **Creation**

During the creation operation, a two-dimensional array is initialized to represent the population. The length of the array equals the size of the population, specified by the user, while the width of the array equals the size of each schedule, specified in the manner previously mentioned. While the population is not full, new schedules are

created and added to the array. Schedules are created by finding courses to fulfill requirements in a permutation of an array of requirements needed to graduate.

Originally, schedules were created by first cloning the requirements set into a temporary set in which SetValues could be removed without repercussion. Then, while requirements were still present in the requirement set, an applicable requirement, course, and session would be randomly selected from the requirement set, course set, and the chosen course's lecture and lab set. If no applicable course could be found, a blank node would be created. Otherwise, the requirements fulfilled by the course would be removed from the temporary requirements set. When either no more requirements remain, or no more space in the schedule's chromosome remains, the schedule would be added to the population.

While this method created valid schedules, the time required to run the method was disproportionately large when compared with the rest of the program. As a result, a new method was devised. In the new method, an array of requirement names was created from the requirements set. For each schedule in the array, contents in the array of requirement names randomly switch positions to produce a permutation of the array. Courses and/or sessions available at that section of the array are found to fulfill the requirements named in this permutation, turning the array of requirement names into a schedule. If a course cannot be found to fulfill the requirement, a blank node is created. Modifications are then made to the schedule to accommodate for differences in size between the array of requirements names and the schedule. If the size of the array of requirements is greater than the size of schedules in the population, the schedule is truncated to fit into the population array. If the size of the array of requirements is less than the size of the schedules in the population, the remaining portion of the schedule is filled with blank nodes to fit into the population array.

## **Selection**

During the selection operation, the existing population array becomes the old population array, and a new population array is created to replace it. Individuals from the old population are selected by a combination of elitist and rank-based selection methods, favoring fit individuals that were recently created. Elitism was chosen for its ability to prevent the loss of individuals with high fitness. Rank-based selection was chosen for its ability to control the specific probabilities of certain individuals entering the mating population, which permits experimentation with the algorithm. Selected individuals are then used to fill the first half of the new population array, which represents the mating population. The first half of the array was used to store the mating population in order to prevent the loss of individuals with high fitness.

Originally, a tournament method was implemented in the selection operation, in which random pairs of individuals were selected, and the individual with the greatest fitness in the pair would enter the mating population. However, individuals with greater fitness would occasionally compete with one another, causing some individuals with high fitness

to be excluded from the mating population, in turn causing average fitness to steadily decline over several generations.

Later, a standard rank-based selection method was implemented, in which the old population would be divided into a given number of groups based on fitness. Groups would be selected through a weighted random method, and individuals from the chosen group would be selected through an unweighted random method. However, some individuals with high fitness would still be excluded from the mating population, again causing average fitness to steadily decline over several generations. The probabilities were altered so that individuals with higher fitness would have a greater chance of entering the mating population, but then only clones of individuals with high fitness would exist after a small number of generations. No arrangement of probabilities was found that consistently increased the greatest fitness in populations, while maintaining the lowest fitness in populations for genetic diversity.

To solve this problem, the rank-based selection method was altered to automatically select a given number of individuals within a given range of the best fitness in the population. Because of this alteration, the selection operation now employs a combination of elitism and rank-based selection. As a result, the highest fitness in a population increases while the lowest fitness in a population vacillates over several generations.

## **Mating**

During the mating operation, the second half of the new population array is created by crossing over individuals of the first half of the new population array. Cross over occurs in a manner similar to the genetic algorithm used in the Traveling Salesman problem. Two individuals from the first half of the new population array are randomly selected to represent parents. Two new individuals are then initialized to represent siblings. The two siblings begin as identical copies of these two parents. A random locus in the chromosomes of the two individuals is then selected for exchange. After the nodes at the loci are exchanged between the two siblings, one of the siblings is searched for any duplicate requirements that were created from the exchange. The nodes at the loci of this duplicate requirement are then exchanged. This is repeated until no duplicates remain.

## **Mutation**

As with other operations, the mutation operation employs a technique from the genetic algorithm used to solve the Traveling Salesman Problem. In this method, a given percentage of the population are selected, based on a mutation rate given by the user. Two loci in the chromosome of a chosen schedule are selected randomly, and the requirements at the loci are exchanged. Finally, applicable courses are randomly chosen to fulfill the exchanged requirements.



## Refining

Throughout the genetic algorithm, two operations are invoked for each schedule created or altered: an operation refining schedules, and an operation analyzing fitness of schedules. The refining operation attempts to improve the fitness of a schedule that is passed to it. This nonstandard operation was implemented to prevent the degradation of average fitness throughout several generations as a result of an increase in duplicate courses, blank nodes, and unfulfilled prerequisites. Originally, duplicate courses would appear when multiple requirements could be fulfilled by a single course, or when one requirement could be fulfilled only by taking multiple courses. Blank nodes would occur when a schedule that completes all requirements mates with a schedule that does not complete all requirements. Unfulfilled prerequisites would occur when a prerequisite is moved in a schedule's chromosome during mutation. However, these issues were eliminated when the refine operation is implemented.

During the refining operation, any duplicate or inapplicable courses are replaced with blank nodes. Then, any requirements not fulfilled by the schedule are found, and any blank nodes are replaced with applicable courses that fulfill such unfulfilled requirements. Finally, any remaining blank nodes are moved to the end of the portion of the chromosome that represents the semester they are in. This last step improves the schedule's fitness for reasons that will be discussed while explaining the fitness operation.

## Fitness

Fitness, used during the selection operation, is stored in an array of integers with the size of the population. In the array, the fitness value of an individual is stored at the same index as the individual stored in the population array. A fitness value in the array is determined by a fitness operation.

As with the refining operation, the fitness operation is invoked whenever a schedule is created or altered. For each filled node in the chromosome, requirements fulfilled by the course in the node are removed from the requirements set. This is done until there are either no more cells to examine in the chromosome, or no more requirements to fulfill. If all requirements are fulfilled, then the schedule is valid. In this case, the fitness value is returned as a positive number, representing the number of cells that remained before the desired date of graduation is encountered. If, however, unfulfilled requirements remain, the schedule is invalid. The fitness value is returned as a negative number, representing the number of requirements that remain.

To encourage invalid schedules to become valid, an additional factor was added to fitness values. Because invalid schedules typically occur when blank nodes exist in a schedule's chromosome, and because it is rare for the mating and mutation operations to successfully replace such blank nodes with courses, fitness values must also consider the

number of blank nodes that occur in each schedule's chromosome. Invalid schedules are more likely to become valid schedules if they have fewer blank nodes, so the fitness values for invalid schedules are decreased by the number of blank nodes in their chromosomes. However, valid schedules are more likely to improve when they have more blank nodes, so the fitness values for valid schedules are enhanced by the number of blank nodes in their chromosomes.

## Experiments and Results

Best fitness and worst fitness were examined for 50 generations for two simplified real-life scenarios. In the first scheduling scenario, a sophomore attempts a double major in Computer Science and Environmental Science. The scenario begins in the spring of an even-numbered year with some credits earned through a combination of AP courses, transfer courses, and courses from three previous semesters at the college. The scenario must be completed in five semesters, with five courses each semester. In the second scenario, a freshman attempts a major in Sociology and a double minor in Marketing and Spanish. The scenario begins in the fall of an even-numbered year with some credits earned through AP courses. The scenario must be completed in eight semesters, with five courses each semester. Each scenario was processed with population sizes of 100, 500, and 1000 members while the mutation rate was 5%. Afterwards, each scenario was processed with mutation rates of 1% and 10% while population size was 500. Figures 1 and 2 display the results of each scenario.

| Population Size | 100   |       | 500   |       |       |       |       |       | 1000  |       |
|-----------------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|
| Mutation Rate   | 5%    |       | 1%    |       | 5%    |       | 10%   |       | 5%    |       |
| Range Limit     | Upper | Lower | Upper | Lower | Upper | Lower | Upper | Lower | Upper | Lower |
| 1               | -2.2  | -10.2 | -1.2  | -12.2 | -1.2  | -11.2 | 4.1   | -10.2 | 4.2   | -11.2 |
| 2               | 4.2   | -9.2  | -1.2  | -12.2 | 4.2   | -11.2 | 4.1   | -10.2 | 4.2   | -10.2 |
| 3               | 5.0   | -8.2  | 4.2   | -12.2 | 5.0   | -11.2 | 4.1   | -9.2  | 4.2   | -10.2 |
| 4               | 5.0   | -7.2  | 6.0   | -12.2 | 5.0   | -8.2  | 4.1   | -7.2  | 4.2   | -10.2 |
| 5               | 5.0   | -7.2  | 6.0   | -12.2 | 6.0   | -8.2  | 4.1   | -7.2  | 4.2   | -9.2  |
| 6               | 5.0   | -6.2  | 6.0   | -7.2  | 6.0   | -8.2  | 5.0   | -6.2  | 5.0   | -8.2  |
| 7               | 5.0   | -6.2  | 6.0   | -6.2  | 6.0   | -6.2  | 5.1   | -6.2  | 6.0   | -7.2  |
| 8               | 5.0   | -6.2  | 6.0   | -5.2  | 6.0   | -6.2  | 5.1   | -6.2  | 6.0   | -6.2  |
| 9               | 5.0   | -5.2  | 6.0   | -4.2  | 6.0   | -6.2  | 5.1   | -5.2  | 6.0   | -6.2  |
| 10              | 5.1   | -4.2  | 6.0   | -6.2  | 6.0   | -5.2  | 5.1   | -5.2  | 6.0   | -5.2  |
| 20              | 6.0   | -2.2  | 6.0   | -4.2  | 6.0   | -4.2  | 6.0   | -5.2  | 6.0   | -5.2  |
| 30              | 6.0   | -2.2  | 6.0   | -3.2  | 6.0   | -3.2  | 6.0   | -3.2  | 6.0   | -4.2  |
| 40              | 6.0   | -2.2  | 6.0   | -4.2  | 6.0   | -2.2  | 6.0   | -4.2  | 6.0   | -4.2  |
| 50              | 6.0   | -2.2  | 6.0   | -3.2  | 6.0   | -4.2  | 6.0   | -3.2  | 6.0   | -4.2  |

**Table 1: Scenario 1 Fitness Ranges in Several Generations Under Different Population Sizes and Mutation Rates**

| Population Size | 100   |       | 500   |       |       |       |       |       | 1000  |       |
|-----------------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|
| Mutation Rate   | 5%    |       | 1%    |       | 5%    |       | 10%   |       | 5%    |       |
| Range Limit     | Upper | Lower | Upper | Lower | Upper | Lower | Upper | Lower | Upper | Lower |
| 1               | -4.3  | -17.2 | -4.2  | -18.2 | -4.2  | -19.2 | -3.3  | -17.2 | -2.2  | -9.3  |
| 2               | -3.2  | -14.3 | -2.2  | -15.3 | -2.2  | -16.2 | -2.2  | -16.2 | -1.3  | -11.3 |
| 3               | -1.3  | -14.2 | -2.2  | -14.3 | -1.3  | -14.3 | -1.2  | -15.2 | 3.5   | -10.2 |
| 4               | -1.3  | -12.2 | -1.3  | -12.3 | -1.3  | -13.3 | -1.2  | -15.2 | 3.5   | -9.3  |
| 5               | -1.3  | -12.2 | -1.2  | -12.3 | 4.6   | -13.2 | -1.2  | -12.2 | 4.4   | -9.2  |
| 6               | -1.3  | -10.2 | 3.7   | -10.2 | 4.6   | -11.2 | -1.2  | -9.2  | 4.5   | -9.2  |
| 7               | -1.3  | -11.2 | 3.7   | -13.2 | 4.6   | -10.3 | 4.6   | -10.2 | 4.5   | -9.2  |
| 8               | -1.3  | -9.2  | 3.7   | -13.2 | 4.6   | -9.3  | 4.7   | -10.2 | 4.5   | -10.2 |
| 9               | -1.3  | -10.3 | 3.7   | -13.2 | 4.6   | -9.2  | 4.7   | -8.3  | 4.5   | -10.2 |
| 10              | -1.2  | -9.2  | 3.7   | -13.2 | 4.6   | -9.3  | 4.7   | -9.2  | 4.5   | -8.3  |
| 20              | -1.2  | -7.2  | 4.6   | -9.3  | 4.6   | -10.2 | 4.7   | -8.2  | 8.2   | -9.3  |
| 30              | 4.4   | -6.2  | 4.6   | -9.3  | 4.6   | -8.2  | 7.1   | -8.2  | 8.2   | -10.2 |
| 40              | 4.4   | -5.2  | 4.7   | -8.3  | 9.2   | -8.3  | 7.2   | -9.2  | 8.2   | -9.3  |
| 50              | 4.5   | -4.2  | 4.7   | -9.3  | 9.2   | -9.3  | 8.1   | -9.2  | 8.2   | -9.3  |

**Table 2: Scenario 2 Fitness Ranges in Several Generations Under Different Population Sizes and Mutation Rates**

Both the highest and lowest fitness values in populations increased over the generations. However, the highest fitness values never decreased over the generations—undoubtedly a result of the elitist selection method. By the end of each experiment, the highest fitness value typically capped around a positive number. After 50 generations, increases in the highest fitness value rarely occurred.

Population size had a noticeable effect on fitness. Though the highest fitness values in initial populations were greatest with large population sizes, the overall highest fitness values were obtained with a population size of 500. Mutation rate likewise had a notable effect on fitness. The highest fitness values were obtained when the mutation rate was 5%—neither a high nor low value.

The best schedule found throughout all experiments was also recorded for each scenario. The best schedules are shown below.

**Scenario 1:**

Fitness: 6.0

Semester 1:

- bio 290
- geo 102
- cmsc 265
- com 101
- cmsc 365

Semester 2:  
cmsc 378  
soc 350  
cis 255  
bio 253  
cmsc 315  
Semester 3:  
eng 117  
cmsc 335  
cmsc 375  
math 255  
geo 375  
Semester 4:  
cis 260  
geo 101  
cmsc 360  
cmsc 355  
Semester 5:

## **Scenario 2:**

Fitness: 9.2

Semester 1:  
econ 102  
anth 110  
soc 350  
soc 311  
soc 319  
Semester 2:  
math 201  
soc 101  
eng 103  
mus 103  
magt 234  
Semester 3:  
magt 336  
rel 103  
span 201  
soc 313  
magt 324  
Semester 4:  
soc 347  
span 202  
eng 101  
soc 323  
span 302  
Semester 5:  
span 301  
chem 101  
span 303  
soc 320  
Semester 6:  
eng 117  
magt 131  
soc 321  
magt 335  
Semester 7:  
soc 340

As the schedules demonstrate, the genetic algorithm is capable of generating college course schedules. The college course schedules complete graduation requirements in a small amount of time without unfulfilled prerequisites or duplicate courses. Some generated schedules may require the student to take difficult or undesired classes. This is illustrated in the scenario 2 schedule by the 300 level courses taken during the freshman year.

## Conclusion

Genetic algorithms are promising means to find solutions to ill-defined problems. As this paper demonstrates, one such problem genetic algorithms can solve involves the generation of college-course schedules. The program that finds solutions to this problem quickly generates college-course schedules that complete graduation requirements in a small amount of time, while recognizing unfulfilled prerequisites and duplicate courses. With future improvements, this program could recognize additional constraints, such as schedule conflicts and student preferences. At the very least, the program could assist the user while developing schedules that are favored by the student. In any case, the genetic algorithm would decrease the difficulty that comes with manually creating college schedules each semester.

## Acknowledgements

I would like to thank Professor Lydia Sinapova for her advice in developing the program and preparing this paper.

## References

- Yamada, T., & Nakano, R. (1997, March 18-19). Genetic Algorithms for Job-Shop Scheduling Problems. *Proceedings of Modern Heuristic for Decision Support*, 67-89.
- Goldberg, D.E. (1989). *Genetic Algorithms in Search, Optimization & Machine Learning*. Boston: Addison Wesley Longman, Inc.
- Haupt, R.L.; Haupt, S.E. (1998). *Practical Genetic Algorithms*. New York: John Wiley & Sons, Inc.
- Jackson, K. (1996). "The Job-shop Scheduling Problem." *Simon Fraser University*. Retrieved February 14, 2008 from <http://www.cs.sfu.ca/research/groups/ISL/papers/jackson-lds/node2.html>

# A Learning Natural Language Parser

Duncan McKee

Kurt Krebsbach

Department of Computer Science

Lawrence University

Appleton, WI 54911

mckeed@lawrence.edu

## Abstract

A natural language parser is described that analyzes the syntactic structure of an input sentence in relation to a specified grammar and generates all possible syntax trees of the sentence, along with estimates of the probability of each being the correct parse. The grammar used is based on X-bar theory, and the parsing algorithm is a chart parse – a top-down parser which uses dynamic programming for efficiency in cases where the grammar leads to ambiguities. The parser has a database of the frequency of application of each syntax rule in the grammar as well as a lexicon of known words and their lexical categories and frequency of use in each category. These are used in a probabilistic context-free grammar model to yield the likelihood judgments of the candidate parses and are updated by user feedback, leading to more accurate subsequent estimations.

# 1 Introduction

Approaches to natural language understanding have usually modeled language as having levels of structure corresponding to traditional levels of linguistic analysis. Thus, speech input undergoes acoustic and phonetic analysis, then phonological, morphological, syntactic, and finally semantic analysis. Each level in this schema imposes a layer of structure on the input with the goal of eventually arriving at a logical representation of the information it imparts.

Natural language does not bend conveniently to these structural analyses, unfortunately. Ambiguity is prevalent at every level of analysis, and some ambiguity cannot even be resolved by a native speaker. When humans process language, structure and meaning are inferred using contextual information. The first problem this poses to computers is that, given a string of words, there are no hard and fast rules to govern the syntactic structure of that string. Information from the semantic level of analysis is required to disambiguate many grammatical ambiguities. Since, in other respects, semantic analysis takes place *after* syntactic analysis, this poses a problem for the level-by-level strategy, necessitating backtracking.

Many recent trends in linguistics and natural language processing move away from the multi-level, structure-heavy approach in favor of methods based on statistical analysis and positional relationships. These ideas have been very productive, but there is still room for structural approaches. By combining them with the statistical and probabilistic strategies, structural analyses can be made flexible enough to deal with ambiguities. This paper examines an example of how this can be accomplished, applied specifically to the syntactic level of analysis: parsing.

Parsing is the logical first step to computer understanding of natural language text. Most approaches to determining the meaning of a natural language statement first analyze the syntactical structure of the statement. The syntactical analysis of a sentence contains the information of what roles the words in the sentence are playing as well as the phrase structure of the sentence – that is, what parts constitute the subject and predicate of the sentence and what the adjectives, adverbs, and prepositional phrases refer to.

Unlike programming languages, which have relatively simple formal grammars, natural languages have not been successfully formalized, so modifications must be made to adapt traditional parsing algorithms to them. One major challenge is that natural language grammars frequently generate multiple possible parses for a single sentence. These are ambiguities that humans resolve through the application of a wealth of semantic and contextual knowledge that computers are not yet able to replicate.

This project's goal was to develop a system to parse grammatical English sentences into phrase structures in a way that deals gracefully with ambiguity. The resulting parser first generates all possible phrase structures using a chart parser algorithm. It then uses a probabilistic grammar to assign each of these possibilities a likelihood measure, which is

the estimated probability of that parse being the intended structure of the sentence. These measures are normalized probabilities of the incidence of each result based on the sentences that the system has seen in the past. Thus, as the parser is exposed to more sentences, it learns what are the most common and the least common grammatical constructions in the language, and therefore improves its judgments of ambiguous cases.

## 2 Background

### 2.1 X-Bar Grammar

The output of a parser such as the one described here is a parse tree, a construct describing the phrase structure of the input sentence in relation to a specified grammar. The internal nodes of a parse tree represent non-terminal symbols in the grammar and the leaf nodes are the words of the input sentence. The grammar used in this parser is a context-free grammar based on X-bar theory. Although X-bar grammar grew out of the Chomskyan generative grammar tradition that frowns upon probabilistic methods (Russell & Norvig, 2003), it still has many features that are useful in computational analysis. It lends itself well to abstraction, as its rules are at most binary branching and are categorized in terms of a few abstract patterns. It also serves as the base framework for much work on describing syntactic phenomena such as agreement, pronoun reference, and structural transformations (Haegeman, 1994).

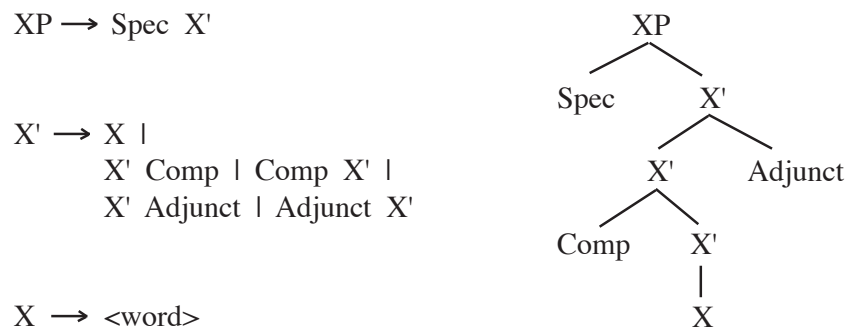


Figure 1: X-bar theory's categorization of non-terminal symbols means that the syntax rules conform to a set of abstract patterns. On the right is a generic XP schema.



There are a few types of non-terminal symbols in an X-bar grammar. First, there are lexical category symbols, which represent the grammatical functions of words, such as noun, verb, preposition, etc. These are always the parent nodes of the terminal symbols – the words themselves – in the parse tree. For each lexical category symbol X, there is an X-phrase symbol, XP, which represents a phrase headed by a word of category X. The motivation for grouping words in this way arises from the perception that an XP fills the same role in a sentence that a single X fills; often, an entire phrase can be replaced by a single word without changing the essence of the sentence, whereas this cannot be done with groups of consecutive words that do not make up phrases. For example, in the sentence, “The computer with the file crashed,” the subject is the noun phrase “the computer with the file” and can be replaced by a pronoun to make the sentence “It crashed.”

X-bar theory postulates another type of non-terminal symbol, usually written X' and pronounced ‘X-bar’. One or more of these symbols comes between an X node and its corresponding XP node; they provide branching locations for complements and adjuncts to X – words or phrases that support or modify X. Since both of these are optional, X' nodes often have only a single child node.

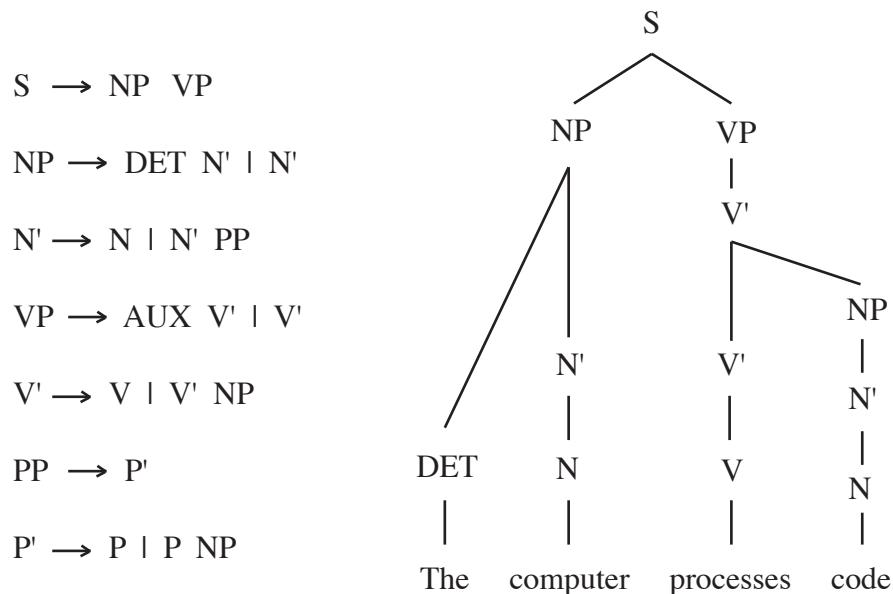


Figure 2: On the left is a small, incomplete X-bar grammar. On the right is a sentence and its parse tree according to this grammar. The root symbol of the sentence, ‘S’, is not technically a proper X-bar symbol, but it is used here for simplicity.

## 2.2 Chart Parser

The algorithm used to find the parse trees is a top-down chart parser, a search algorithm that uses dynamic programming techniques. Chart parsers are commonly used in natural language processing because they deal efficiently with ambiguous grammars, which can be combinatorially problematic (Russell & Norvig, 2003; Allen, 1995). This algorithm uses a data structure called a chart to store partial parses, which are parse trees of substrings of the input sentence. It builds up the set of all partial parses by extending and combining them as it processes the words of the sentence one at a time. The variant described here is adapted from Russell & Norvig (2003) and is similar to the Earley parser (Earley, 1970).

For an  $n$ -word sentence, the chart is a multigraph with  $n+1$  vertices, where a partial parse of the  $i^{\text{th}}$  to  $j^{\text{th}}$  words would be associated with an edge from vertex  $i$  to vertex  $j+1$ . This edge is added to the chart during the processing of the  $j^{\text{th}}$  word by one of three methods: PREDICTOR, SCANNER, and EXTENDER. Figures 3-5 show how these methods would produce edges when parsing the example sentence from Figure 2. When an edge is added for a partial parse tree with a non-terminal symbol  $X$  as a leaf node, the PREDICTOR method adds a new edge for each rule in the grammar for  $X$ , representing the possible branchings that could extend the tree (Figure 3). For each word  $w$ , the SCANNER method extends the edges with partial parse trees that need a word at this position of a lexical category that  $w$  belongs to (Figure 4). When an edge is added for a complete parse of some phrase  $Y$  that spans words  $i$  through  $j$ , the EXTENDER method combines it with any edges ending at  $i$  that were lacking a  $Y$  subtree; this adds a new edge that spans the combined spans of the edges that produce it (Figure 5). If the grammar admits a parse tree for the entire sentence, the algorithm succeeds by producing at least one edge from the first vertex to the last vertex that corresponds to a parse tree with the start symbol of the grammar as the root node and the words of the sentence as the only leaf nodes.

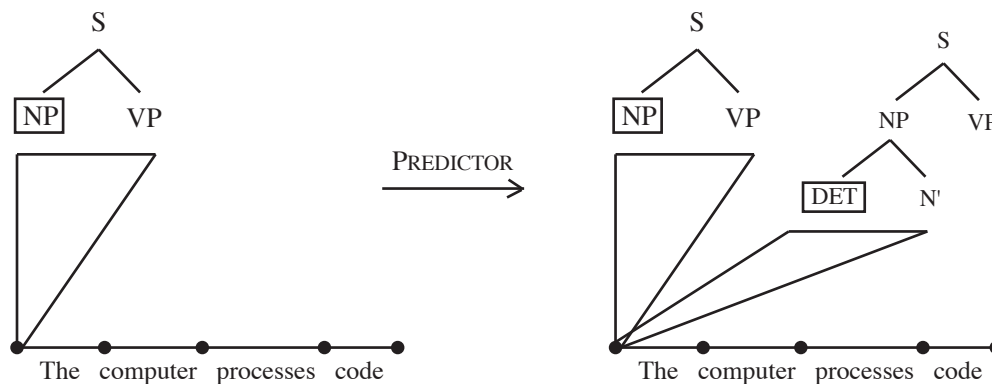


Figure 3: An edge added by the PREDICTOR method. The box indicates the next branch to be extended or specified.

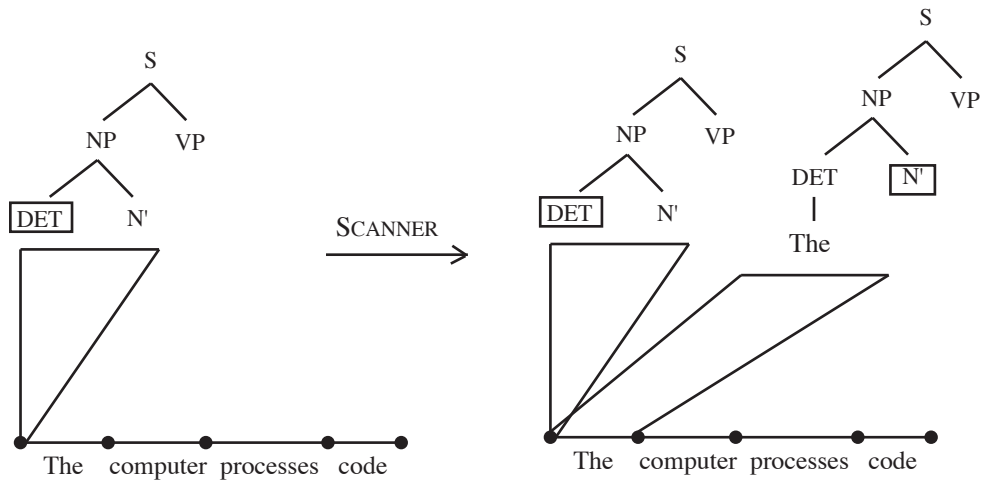


Figure 4: An edge added by the SCANNER method. Note that not all edges that would be in the chart at this point in the algorithm are shown.

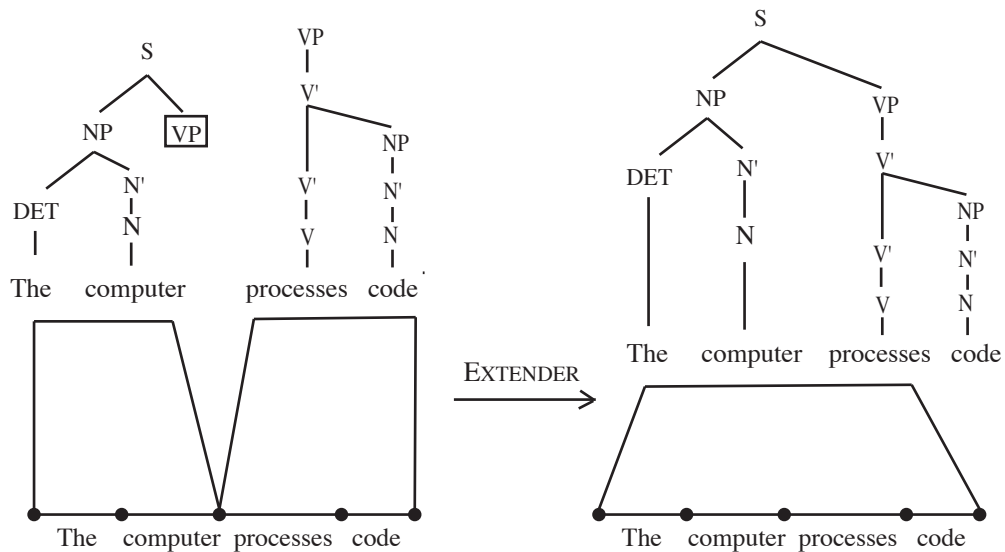


Figure 5: An edge added by the EXTENDER method. Note that the edges and partial parse trees shown on the left remain in the chart and can extend other edges – they are omitted on the right for clarity.

## 2.3 Probabilistic Context-Free Grammar

The statistical learning feature of the system uses a form of probabilistic context-free grammar, a model of syntactic structure that is more conducive to learning and more robust in ambiguous cases (Manning & Shütze, 1999). In a PCFG, every syntax rule – including those that attach words to lexical categories – has a probability associated with it. For each non-terminal symbol  $X$ , the rules in the grammar of the form  $X \rightarrow Y Z$  partition the set of possibilities for the structure of an  $X$  component. Thus, the normalization constraint dictates that the probabilities of the rules in this set sum to one.

The probability of a given complete parse tree is simply the product of the individual probabilities of each node in the tree. This probability can be understood as production probability; if the PCFG is used to generate a legal sentence by starting with the symbol 'S' and recursively replacing each non-terminal symbol with one of the relevant rules (chosen randomly using the given probability distribution), the probability of arriving at some sentence is the product of the probabilities of each rule choice.

## 3 The Parser

The PCFG technique is used in this parser not for making the parsing itself more efficient or flexible, but rather as a means by which to store and use information about the grammar that is collected over many applications of the parser, and hence provide a more accurate evaluation of future input. The parser is able to produce more than a list of the possible parse trees for an input sentence; it also judges their relative grammatical acceptability. This does not take semantic information into account, as this is solely the syntactic level of analysis. In theory, the output from the parser could be passed on to a semantic processor, which could use the parser's evaluations as a starting point and update the likelihood measures based on semantic information.

The system uses a database for knowledge persistence. When an input sentence is entered into the parser, each word is looked up in a database of all words known to the system and their possible lexical categories (e.g. noun, verb, etc.). For each word/category pair, the lexicon database has a count of how many times the word has been observed in that category. The count of each category is divided by the sum of the counts of all categories for that word – this yields the probability that the word exemplifies that category. This implementation of probability inference is crude but effective.

The chart parser algorithm is then used to generate all the possible parses of the input sentence, using a fully-specified X-bar grammar. This algorithm is efficient in most situations, but, due to the tree structures of the generated parses, there is a possibility of an exponential number of results. Fortunately, such sentences are not common in actual usage.

In addition to word category counts, the occurrence frequency of each grammar rule that is used in the candidate parses is selected from the database, which keeps a count of past appearances of each grammar rule. These counts are converted to probabilities in the same way as are the word/category counts. The number of times a certain nonterminal symbol has been observed to be rewritten by each rule is divided by the sum of the counts of all rewrites of that rule, effectively yielding the probability of that syntax tree node appearing in that environment.

After generating all the probabilities of the components of the parses, they are multiplied as is typical in the use of probabilistic context-free grammars, resulting in occurrence probabilities of each of the possible parse trees. These are then normalized against each other, representing the assumption that the correct parse is among the possible parses generated by the chart parser, and these can be compared as measures of their relative strength as solutions.

Figure 6 shows an example of the parser's analysis of a sentence with multiple parses. It portrays a hypothetical state of its database after having seen 30 input sentences. Via the ratio of the PCFG probabilities of the parse trees, the parser assigns a 90% probability to the solution on the left and a 10% probability to the solution on the right (under the assumption that one of them is the intended interpretation).

The user can then indicate which result was the intended structure, which will increment the occurrence counts in the database for the rules and lexical category assignments used in that parse. If the same input is fed to the parser a second time, it will favor the correct one slightly more than it did previously. Over time, the system learns what structures are most used in the language, and hence makes better guesses in ambiguous situations.

In practice, this has produced some positive results; the parser is often able to distinguish between the obvious, straightforward parse of a sentence and an anomalous result brought about by a grammar rule that exists to deal with some rare construction. For example, it favors the correct interpretation of the sentence "I can code" over the interpretation that implies "I put code into cans" by a 99.3% to 0.7% margin. Still, there are some instances where the incorrect parse scores higher, usually because determining the correct interpretation requires semantic information.

It is important to note that although the input is only being analyzed on the basis of syntactic structure, the way the system favors some rules over others is, to a degree, based on semantic information learned from human agents via its feedback system. Furthermore, it is being trained only on the dialect and sentence domain of the input it receives, which could make it inappropriate for input from a novel source.

| rules                           | frequencies | probabilities  |
|---------------------------------|-------------|----------------|
| $S \rightarrow NP VP$           | 30          | 1.0            |
| $NP \rightarrow DET N' \mid N'$ | 20 \mid 15  | 0.57 \mid 0.43 |
| $N' \rightarrow N \mid N N'$    | 35 \mid 5   | 0.88 \mid 0.12 |
| $VP \rightarrow AUX V' \mid V'$ | 10 \mid 20  | 0.33 \mid 0.67 |
| $V' \rightarrow V \mid V' NP$   | 30 \mid 15  | 0.67 \mid 0.33 |

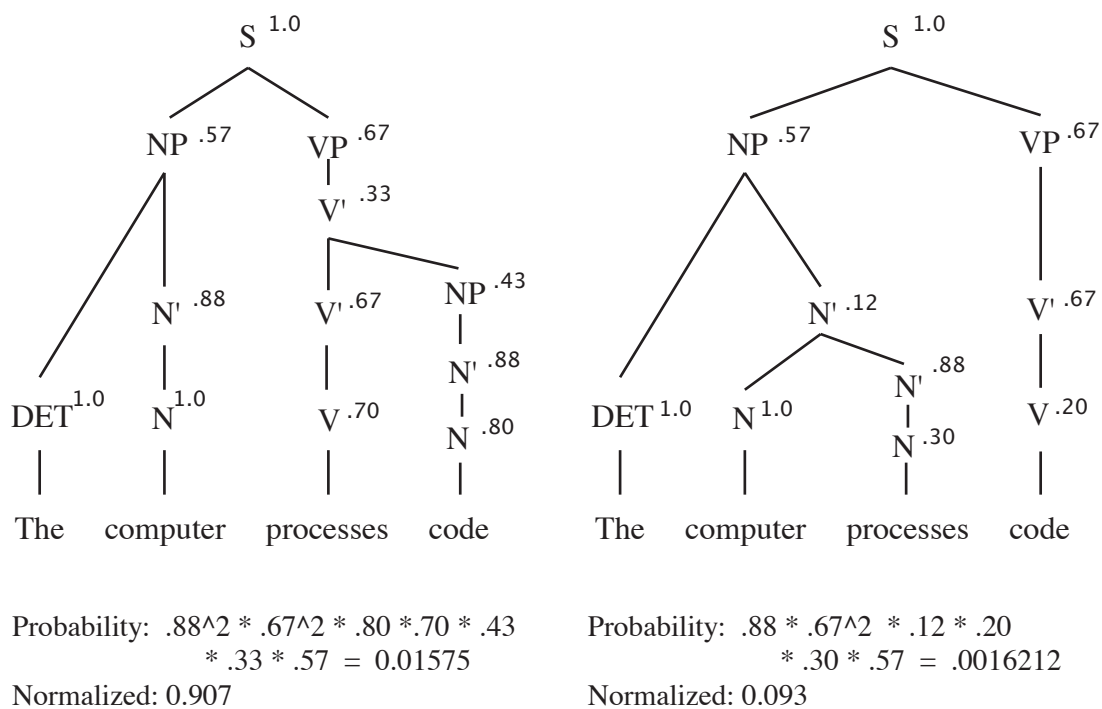


Figure 6: A rule for compound nouns has been added to the grammar, producing multiple parses for the example sentence. The table above shows a hypothetical state of the database after it has seen 30 input sentences. The nodes in the parse trees are marked with their individual probabilities and the tree probability calculations are shown below.

## 4 Conclusion

Without semantic and contextual knowledge, consistently determining the correct syntactic interpretation of a sentence is not possible. However, the strategy of generating multiple candidate interpretations and providing as much information as possible about the syntactic feasibility of each holds the promise of providing an advantageous starting point for semantic and logical analysis without necessitating backtracking to the syntactic analysis stage. In any case, the complete understanding of natural language – while currently an unsolved problem – has inspired much beneficial research.

## Acknowledgments

Many thanks to Lawrence University Provost David Burrows for supporting this work through a Lawrence *Enhancing Academic Distinctiveness* grant. This program was written in PLT Scheme and made use of Jay McCarthy's Sqlite package, available at the PLaneT Package Repository (<http://plt-scheme.org>).

## References

- Allen, James. 1995. *Natural Language Understanding*. Redwood City, CA: Benjamin/Cummings.
- Earley, J. 1970. "An efficient context-free parsing algorithm", *Communications of the Association for Computing Machinery*, 13:2:94-102.
- Haegeman, Liliane. 1994. *Introduction to Government and Binding Theory*. Oxford, UK: Blackwell.
- Manning, C. & Schütze, H. 1999. *Foundations of Statistical Natural Language Processing*. Cambridge, MA: The MIT Press.
- Russell, S & Norvig, P. 2003. *Artificial Intelligence: A Modern Approach*. Upper Saddle River, NJ: Pearson Education.

# Application of BLAST-based Techniques for Musical Information Retrieval

Fedor KORSAKOV  
(student)  
Department of Computer Science  
University of Northern Iowa  
Cedar Falls, IA 50614, USA  
korsakov@uni.edu

## Abstract

Content retrieval in musical collections has been dependent on textual metadata (e.g. ID3 tags) which can present problems when the title of a piece is forgotten, misspelled, or when the search revolves around the similarity of sound. Content-based MIR (musical information retrieval) could offer an alternative. BLAST, an algorithm widely used in bioinformatics to search for sequences of aminoacids within longer sequences, seeks similarities and homologies, which makes it interesting for MIR, because musical information can be expected to be imprecise, and because homologies can allow to draw connections between musical pieces. Increased availability of digital music necessitates MIR methods which would allow to search a polyphonic sound collection with polyphonic queries to retrieve individual files, and a question can be raised how viable is BLAST-based retrieval for this kind of data. This paper discusses an implementation of such a system.



## INTRODUCTION

Personal computers are being increasingly used for multimedia purposes. The emergence of new content distribution models contributed to the growth of collections of digital music. One of the effects of this growth is the need for the enhancement of content retrieval mechanisms, which historically have been dependent on textual metadata associated with files. In the specific case of musical data, a common approach is to incorporate tags (such as ID3) into the files. While this method is adequate for textual search, there are drawbacks in using data representation significantly different from data itself. Textual search, for all of its simplicity, may have difficulties with situations where the title of a musical piece is forgotten, misspelled, or typed in a different language, or where the search revolves around the similarity of the sound (for example, looking for a remix that incorporates a classical piece) rather than the title. While tags can be edited to compensate for some of these problems, it is obvious that devising means to perform MIR (musical information retrieval) on the basis of content could offer a promising alternative solution. Furthermore, such approach could lead to innovative user interfaces that would allow to perform searches with acoustic queries which are hummed, spoken, or played on an instrument.

A conceptually alike challenge is common in bioinformatics, where it is frequently necessary to perform a search for a sequence of aminoacids within a longer sequence (e.g. genome). BLAST is an algorithm widely used to address this need. BLAST seeks similarities and homologies, which makes it interesting for MIR, because musical information, especially user queries, can be expected to be inherently imprecise, and because homologies can allow to draw connections between musical pieces. Furthermore, BLAST is appealing because it is faster than the other two well-known similarity search algorithms: Smith-Waterman and FASTA.

## LITERARY REVIEW

A substantial body of information exists on the subject of MIR. Mongeau-Sankoff algorithm [4] is a seminal work which provides an effective similarity measure for pieces of music, but requires the information to be presented in a symbolic form. While it has been successfully used for MIR [1], the conversion of musical files into sheet music is beyond the scope of this research. Kline and Glinert [2] as well as Miura and Shioya [3] all agree that while pitch contours may be used for the purposes of fast retrieval, the accuracy is very poor. The latter research suggests using pitch spectrum (histogram of notes per bar) as a means to describe musical information, and even though the technique was developed for sheet music, the idea behind it may potentially be reapplied in the context of this research. While a fairly large number of works deal with monophonic (query by humming) or symbolic queries on symbolic databases, Yang's research [5] presents interest due to its focus on polyphonic queries on a polyphonic database. Yang uses spectral indexing and implies that this technique is rather unexplored. Short-Time Fourier Transform is used to generate spectrograms, and then the resulting data is processed to determine characteristic sequences. It would appear obvious that due to difficulties of automated music transcription, spectrograms are a promising way to address the problem of finding a suitable means of representation for the music.

## **METHODOLOGY**

The system I propose preprocesses the collection by performing Short Time Fourier Transform on the musical files. The results are normalized, expressed as 4-bit integers and stored as nucleotide base sequences (two bases per integer) in FASTA format and as PNG image files. The query (a musical excerpt) is preprocessed likewise. A Mega BLAST search is conducted on the spectrograms, returning files that match the query most closely and the place within the sequence where the match occurs.

The system is implemented in a predominantly Linux environment. The development has been done in C language using `fftw`, `libpng` and `libsndfile` libraries, and the project will be made available under GPL. Additionally, the project uses `SoX` for sound file conversion. Currently the system is in the state of a functional prototype.

## **DISCUSSION AND FINDINGS**

### **Current Results**

The most direct approach to the problem involved the application of Mega BLAST to the sequences of columns (i.e. sequencing along frequency axis) of STFT results. However, while this approach was fitting for similarity detection, it was not adequate for accurate retrieval, since it effectively disregarded pitch information, and sometimes even did not return the piece on which the sound clip was taken (although it was able to return numerous pieces by the same artist). A more promising method consists of creating sequences of that correspond to pitch bands over time (i.e. sequencing along time axis), and then selecting the matches where numerous offsets correspond to the same region of a file, and this is the method that is currently evaluated.

### **Questions to Address**

The primary question that remains to be answered revolves around the optimal representation of STFT results for BLAST processing. The main challenge consists of 1-dimensional representation of 2-dimensional data. Furthermore, there is a substantial room for optimization. It remains to be determined whether 4-bit integer representation of STFT results provides a good combination of descriptiveness and space efficiency, and it should be possible to assess varying bit depths once the suitability of sequencing along time axis is confirmed.

## **REFERENCES**

- [1] C. Gomez, S. Abad-Mota and E. Ruckhaus, "An analysis of the Mongeau-Sankoff

- algorithm for music information retrieval”, In *Proc. ISMIR 2007*, Austria, 2007, p. 109.
- [2] R. Kline and E. Glinert, “Approximate Matching Algorithms for Music Information Retrieval Using Vocal Input”, *MM’03*, 2003.
  - [3] T. Miura and I. Shioya, “Similarity among Melodies for Music Information Retrieval”, *CIKM’03*, USA, 2003.
  - [4] M. Mongeau and D. Sankoff, “Comparison of Musical Sequences”, *Computer and the Humanities*, vol. 24, 1990, pp. 161–175.
  - [5] C. Yang, “Efficient Acoustic Index for Musical Retrieval with Various Degrees of Similarity”, *Multimedia’02*, France, 2002, pp. 584-591.

# Mapping Application Attributes to Object/Relational Mapping Solutions

Kyle Hawkins and Elizabeth Towell

Computer Science

Carroll College

Waukesha, WI 53186

[khawkins@cc.edu](mailto:khawkins@cc.edu)

[etowell@cc.edu](mailto:etowell@cc.edu)

## Abstract

The object/relational impedance mismatch is the breakdown of communication between object-oriented programming languages and relational databases, and has been a problem developers have been facing for years. Two frameworks, Hibernate for the Java programming language, and ActiveRecord for the Ruby programming language, offer solutions to the problem but in fundamentally different ways. This research develops a model that a software developer could use to determine which framework to use dependent on the needs of their application. The model is based on the comparisons of the two languages, the discussion of the object/relational impedance mismatch, the comparison of Hibernate and ActiveRecord, the comparison of different types of web applications, and then ends with the comparison of how Hibernate and ActiveRecord handle the various types of applications.

# 1 Introduction

Web applications today have been similar in design and purpose, and generally have been utilizing the same types of technologies. Two technologies utilized by many web applications are object-oriented programming languages that communicate with a relational database. However these two technologies work in fundamentally different ways, and this has led to the discovery of what is known as the object/relational impedance mismatch. This mismatch refers to the difficulties in mapping classes and objects to tables in a database, thus resulting in the difficulty of developing applications that utilize these technologies.

Over the years this mismatch has led to various different types of solutions, and one of the solutions is known as object/relational mapping. Simply put this solution is mapping specific objects to tables through the use of metadata, and the goal is to have this all done transparently so that developers are not burdened by the impedance mismatch. Frameworks have been built to be used as middleware to handle this mapping, and the two frameworks that have had the most discussion in the past few years are Hibernate and ActiveRecord. Hibernate has risen as the enterprise standard for managing persistence of Java applications, while ActiveRecord manage persistence and mapping for the new Ruby on Rails web application framework.

Both frameworks offer a solid solution to the object/relational impedance mismatch; however they do so in fundamentally different ways. Since Ruby on Rails has been seen as counter to Java technologies because of its new approach to development, it is appropriate to compare the two frameworks together. This paper will do an in-depth analysis of the two frameworks starting with a comparison of the two languages, an in-depth analysis of the object/relational impedance mismatch and how each framework solves the problem, a discussion and categorization of various web applications, and finally a model will be presented for a developer to use that maps specific application attributes to one of the frameworks.

## 2 Programming Languages

A programming language is a set of syntactic and semantic rules that are used to give instructions to a machine such as a computer. The design and study of programming languages has been happening for a number of years, and has been utilized by computer scientists to execute and study algorithms, and will continue to progress and change with the progression and change of technology. As with any language, programming languages differ in various ways that offer different levels of readability, reliability, efficiency, and writeability. The difference lies in two major areas; the

language design methodology and the language implementation. The design methodology is the way the language speaks with a user and how the language utilizes machine architecture. The language implementation refers to the method in which a language communicates to the machine and performs commands given by the user. (Sebesta, 2006)

## 2.1 Java vs. Ruby

Table 1 maps out some key similarities and differences between the two. Ruby is a much more powerful and productive language than Java. But Java has the benefit of its years of experience and has proven itself to be a productive and powerful.

| Both               | Ruby                  | Java                      |
|--------------------|-----------------------|---------------------------|
| Strongly-typed     | Dynamically-Typed     | Statically-Typed          |
| Object-Oriented    | Fully Object-Oriented | Not Fully Object-Oriented |
| Garbage Collection | Interpreted           | Compiled                  |

Table 1: Comparison of Java and Ruby

## 3 Object/Relational Impedance Mismatch

Persistence, the ability for an object to live past its creation point, is an important factor in modern web applications. Almost all web applications of today utilize some sort of relational database backend to support persistence. A lot of these applications work with object-oriented programming languages such as Java and Ruby, the two that have been a focus of this paper. The constant use of these technologies has led to the discovery of what is known as the object/relational impedance mismatch, the name given to describe the differences between the two technologies that cause a lot of difficulties in development.

The problem lies in the fundamental differences between the object-oriented paradigm and the relational model. Relational technology is founded on proven mathematical principles of set theory while object technology is founded on proven software engineering techniques. Relational technology concentrates solely on the data aspects of a solution and the relationships of the data are expressed through foreign keys between tables. Thus one would access a particular entity and its relationships by joining rows. Object technology focuses on encapsulating data and behavior together and expresses relationships by aggregation. Therefore access of an object and its relationships is done by traversing through an object and the objects that it knows about. (Ambler, 2003)

This has resulted in several difficulties in creating efficient software applications especially in terms of object-oriented concepts. These applications run into the difficulty of persisting objects especially in terms of inheritance or polymorphism. The concepts of subtypes and subclasses aren't represented easily in relational databases, and even those databases that try to support these aspects do not have an industry-wide standard. Hand coded solutions to these problems result in software that is tightly coupled, has high granularity, and in general just a mess. Then as these software projects grow the difficulty of these sloppy solutions grows even faster, leading to applications that were difficult to extend and maintain. (Ambler 2003)

## **4 Object/Relational Mapping Frameworks**

An object/relational mapping framework also known as ORM is founded on creating a solution to the object/relational impedance mismatch by automating the persistence of objects through the use of metadata. This metadata is responsible for layering a blueprint on how the objects in the application map to specific tables and relationships in the database. ORM is also responsible for making this happen in a transparent way, therefore removing the developer from having to deal with the difficulties of the object/relational mismatch. Even though there are a plethora of different frameworks for various languages, the two that are the focus of this paper are Hibernate and ActiveRecord. (Bauer, 2007)

To better display how both frameworks work, a sample application was developed utilizing both frameworks to create examples for the paper. The application is a small inventory manager responsible for overlooking a company's hardware and software with relationships to specific employees. Employees can use many different pieces of hardware such as printers or workstations, and these pieces of hardware can be used by many different employees. Employees can also use different pieces of software such as Eclipse, Word, or Notepad, and these pieces of software can be used by many different employees. Specific maintenance and upgrades can be performed on hardware, which establishes a many-to-one relationship between the hardware and its upgrades and maintenance. This application was developed with an Oracle database behind it, and was developed in Java with the use of Hibernate and Ruby with the use of ActiveRecord. Let it be noted that this application is a simple solution and does not reflect all of the possible issues that can arise for an enterprise web application; it serves just as a simple example to how Hibernate and ActiveRecord work.

### **4.1 ActiveRecord**

Ruby on Rails has been one of the most popular fully stacked web application frameworks on the market, and has given fame to the Ruby language and community. Ruby on Rails is founded on a few principles: convention over configuration and the Don't Repeat Yourself principle or DRY. Within this framework the persistence is handled by the framework known as ActiveRecord, which follows the Active Record design pattern. This section will display how ActiveRecord handles persistence and how the framework follows its founding principles.

ActiveRecord works off of specific naming conventions, therefore removing the need for configuration files and mapping documents. Instead of reading a document or configuration file, ActiveRecord reads the database and matches the names of tables to the names of classes. Therefore when a developer creates a specific class, they don't even have to write code for all the attributes, the framework automatically generates the source code based off of the table. For example in our sample application if we create an Employee or Hardware object all we have to do is subclass the ActiveRecord base class and the framework will handle the rest for us. This is demonstrated by Figure 3.

```
class Employee < ActiveRecord::Base
end
class Hardware < ActiveRecord::Base
end
```

Figure 3: Two examples of ActiveRecord objects.

However your database does not have to follow the naming conventions because ActiveRecord has the ability to override the normal naming conventions and allow you to establish specific mappings. For example in our application if we were to call the Employee class the Worker class instead, we would have to override the naming conventions through Ruby specific method calls as such. This is demonstrated in Figure 4.

```
class Worker < ActiveRecord::Base
  set_table_name :employee
end
```

Figure 4: Example of overriding naming conventions.

This concept of convention over configuration can be seen in establishing relationships between the various objects, through the use of specifically named methods. These methods establish anything from many-to-many relationships to one-to-one relationships between objects. Establishing this relationship creates a set within the object or specifies the foreign key of the relationship. This foreign key can either explicitly referenced within the object or you can follow another simple naming convention like such: tablename\_id. For example in our application the Hardware class has a many-to-many relationship with the Employee class and a many-to-one with the Maintenance and Upgrade classes and the source code is demonstrated by Figure 5.



```

class Hardware < ActiveRecord::Base
  has_many :upgrades
  has_many :maintenances
  has_and_belongs_to_many :employees
end

```

Figure 5: Example of establishing relationships.

It can thus be seen from these examples that a result of the principle of convention over configuration that the source code is greatly reduced compared to other languages. In fact, when following all of the Rails naming conventions and not overriding anything there is only one configuration file needed that is the YAML file. This file establishes the various database connections used for the development, deployment, and testing parts of the application allowing for one centralized area for this configuration. YAML is also responsible for establishing which adapter for ActiveRecord to use on communication between the application and the database. One problem however that is present is that the framework does not come with adapters for all popular databases out of the box, instead you have to download those adapters from other sources, even perhaps coming down to developing your own adapter.

The overall concept of convention over configuration supports an even bigger principle that Rails has been pushing for its entire lifespan. This is the Don't Repeat Yourself principle otherwise known as the DRY principle. With the use of convention over configuration ActiveRecord removes the need to develop configuration files that establish connections and mappings that are already present within the software. Therefore a developer can follow the DRY principle because they are not duplicating code within configuration files. (Halloway, 2007)

ActiveRecord provides the developer with a simple solution that is easier to configure and easier to develop than other frameworks. Developers are freed from worrying about developing complex configuration files and are allowed to maintain an application much easier. Any changes to the database can be reflected through the application with little or no changes to the objects themselves. Also the logic to access the database is not kept in a separate layer instead this is all kept within the objects themselves. Therefore developers are freed from having to access another layer to perform database transactions or queries; instead the creation of an object is all that is needed.

However each solution has its weaknesses and ActiveRecord has a couple of key ones that a result of its design. One is that the classes themselves become tightly coupled to the database tables and for simple data models with simple database designs this is not a problem. However as the data model expands and becomes more complicated this problem leads to ActiveRecord losing its simplicity. Another weakness deals with the concept of convention over configuration and if you are using a data model that does not follow these conventions, such as a legacy system model, you lose the benefit of this

principle. The result in not utilizing the naming conventions is objects that are loaded with configurations to their tables and relationships, thus defeating the purpose of ActiveRecord's simplicity.

## 5.2 Hibernate

Hibernate is a persistence framework developed in the Java language that is based off of the Data Mapper Design pattern. This framework is known as a lightweight framework, as it is designed to work with the Java Bean design pattern. This allows Hibernate to run stand-alone or work with other frameworks that are considered lightweight. Hibernate has been around for awhile now and has been the industry standard for persistence.

Hibernate is founded on the use of configuration files and mapping metadata that describe the mapping between classes and their respective tables. The main Hibernate configuration describes which database the application is supposed to connect to and which adapter and dialect to utilize with that database. This configuration file is fed into a Configuration class which creates the SessionFactory object, who is responsible for generating Session objects. The Session class is the center of Hibernate, as it knows how to access the database as well as the state of the database. All of the methods to perform CRUD operations, queries, and transactions can be found within the Session class. An example of a configuration file for Hibernate can be seen in Figure 6, based off of our sample application.

```
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE hibernate-configuration
PUBLIC "-//Hibernate/Hibernate Configuration DTD//EN"
"http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">

<hibernate-configuration>

  <session-factory>

    <property name="connection.url">XXXXXXXX</property>
    <property name="connection.driver_class">oracle.jdbc.driver.OracleDriver</property>
    <property name="dialect">net.sf.hibernate.dialect.Oracle9Dialect</property>
    <property name="connection.pool_size">1</property>
    <property name="connection.username">XXXXX</property>
    <property name="connection.password">XXXXX</property>

    <!-- Mapping files -->
    <mapping resource="Employee.hbm.xml"/>
    <mapping resource="Hardware.hbm.xml"/>
    <mapping resource="Software.hbm.xml"/>
    <mapping resource="Upgrade.hbm.xml"/>
    <mapping resource="Maintenance.hbm.xml"/>
    <mapping resource="Login.hbm.xml"/>

  </session-factory>

</hibernate-configuration>
```

Figure 6: An example of a Hibernate configuration file.

The session factory tags in the XML give some light as to the purpose of the configuration file. Along with establishing the SessionFactory, the configuration file also is responsible for either mapping out classes to their respective tables or pointing to specific mapping documents that contain this information. This configuration file is very simple and will run an application with no problem, however there is much more that can be written into a configuration file such as optimization commands and techniques.

The SessionFactory and Session classes are not just built by the minimal information in the configuration file; they also rely heavily on XML mapping files. These mapping files describe which class maps to which table and what relationships that class has with other classes. Mapping files are directly linked to classes, and therefore there is one convention that Hibernate works with; all classes must follow the Java Bean Specification. This allows Hibernate to be lightweight and operate stand-alone or with other Java frameworks such as Spring or Struts. This allows Hibernate to create objects dynamically via reflection through the Session class, but it also allows a developer to establish relationships that exists between the objects. For example for our sample application the Hardware class can show its various relationships by simply having a Set attribute associated with the many-to-many relationship or an object for a many-to-one or one-to-one relationship. By following this convention the mapping files can specify what the classes are, what table they are associated with, and what relationships exist between the classes. Figures 7 and 8 show parts of a mapping document for the Hardware.

```
<hibernate-mapping>
  <class name="hibernate.Hardware" table="HARDWARE" >
    <id name="hardwareID" type="long" column="HARDWAREID">
      <generator class="increment">
        </generator>
      </id>

    <property name="hardwareName" column="HARDWARENAME"/>
    <property name="serialNumber" column="SERIALNUMBER"/>
    <property name="datePurchased" column="DATEPURCHASED"/>
    <property name="whoInstalled" column="WHOINSTALLED"/>
    <property name="vendorName" column="VENDORNAME"/>
  </class>
</hibernate-mapping>
```

Figure 7: Example of basic mapping.

```
<set
  name="employees"
  table="HARDWARE_EMPLOYEE_INT"
  cascade="save-update"
  lazy="true"
  outer-join="true">
  <key column="HARDWAREID"/>
  <many-to-many class="hibernate.Employee" column="EMPLOYEEID"/>
</set>
```

Figure 8: Example of establishing relationships via mapping.

Hibernate has been around for awhile and thus has proved itself through experience in web applications and enterprise software. The framework is lightweight, therefore is easily insert into an existing application that has its objects follow the Java Bean Specification and can easily incorporate other frameworks because of this. Another strength is that the framework creates a separate layer of persistent logic, therefore the objects themselves are not coupled to the database or the data model. Therefore changes to the data model, or switching to a new database will not affect the objects themselves, and maintenance and extension of these objects will be easier. This is a result of following the Data Mapper design pattern, whose goal is to decouple the objects from the data model.

Even though it is tried and proven, Hibernate does posses a few weaknesses that have paved the way for the Ruby on Rails framework. Configuration files and mapping documents create data redundancy because you are writing the same thing in multiple places. Changes to the objects lead to changes in the mapping documentation; this creates a problem especially for agile developers. These configuration files and mapping files also are very cumbersome, and are sometimes the hardest part of developing the application; therefore developers spend far more time creating these files than coding business logic, which should be the priority.

## **5.4 Hibernate vs. ActiveRecord**

Although there has already been a big comparison between the two frameworks, there are still different aspects that need to be discussed about each framework. The other areas that a developer should consider when comparing the two frameworks is how they handle queries and transactions.

### **5.4.1 Querying**

Hibernate and ActiveRecord both offer a plethora of different abilities to query the database for your object or a range of objects. However because of the design patterns they both use, they operate in very different fashions. ActiveRecord has all of the database access abilities tightly coupled to the objects themselves, so querying the database can happen just by creating an object. Hibernate on the other hand relies on passing the queries through the persistence layer, thus calling on specific querying methods from the Session class.

ActiveRecord offers the simplest methods; therefore these will be analyzed first. The framework itself generates a multitude of different methods for querying based off of the attributes the object has. For example in our sample application if we are looking for a particular employee, we can simply create the object and query the database in

many different ways. ActiveRecord also provides the abilities to query the database through the find method which can take any parameter you put in to find the object; you can even pass the parameter to find all the objects. The methods also have the ability to add conditions as well as order the results just by passing simple parameters. Figures 9 and 10 display the possible methods.

```
Employee.find_by_employeeName
Employee.find_by_salary
Employee.find_by_employeeName_and_salary
```

Figure 9

```
Employee.find(1)
Employee.find(:all)
Employee.find(:all, :conditions => "employeeName = 'Kyle'", :order => "salary ASC")
```

Figure 10

Hibernate on the other hand relies on utilizing the Session class to perform various queries and offers three specific types of querying to be performed. Because the Session class is a direct connection to the database it is important that a developer remembers that this Session must be closed to terminate this connection. The three different querying types are querying by HQL, by Criteria, and by Example. HQL is also known as the Hibernate Query Language, which looks exactly like SQL but with a little bit of an object twist to it. Querying by Criteria and by Example fall under the same API, therefore have similar qualities. Rather than worrying about executing SQL or HQL statements, querying by these two methods simple creates objects that the Session will go find the database. These Criteria and Examples can also specify specific conditions as well as ordering procedures, and they can also be set to ignore case or not. The benefit of the last two querying methods is the complete removal of the relational aspect of the application and the developer can work strictly with objects. Here are a couple of examples of the different querying methods with Figure 11.

HQL

```
List list = session.createQuery("from Employee employee where " +
    "employee.employeeName like " + this.getEmployeeName()
    + """).list();
session.close();
```

Querying by Criterion

```
List list = session.createCriteria(Employee.class)
    .add(Expression.eq("employeeName", "Kyle"))
    .addOrder(Order.asc("salary"))
    .list();
session.close();
```

Querying by Example

```
Example exampleEmployee=Example.create(Employee)
    .ignoreCase().enableLike(MatchMode.ANYWHERE);
List list = session.createCriteria(Employee.class).add(exampleEmployee).list();
```

Figure 11

### 5.4.2 Transactions

Hibernate provides an entire Transaction API that maps specific capabilities of the database that it is connected to. In Hibernate all of the operations are explicit; therefore you need to write in your code when and where you want your transaction to rollback, and when and where you want the transaction to commit. (Bauer) In ActiveRecord this is all done implicitly, therefore you do not have to worry about when the transaction needs to commit or needs to rollback. The framework will handle this through exception handling and therefore removes this concern from the developer. Again do to the design pattern; the methods for transaction are all contained with the objects themselves. (Halloway, 2007)

Hibernate offers support for transactions that ActiveRecord does not, and they are distributed transactions and transactions for container managed applications. Since many Java applications operate in a container managed environment Hibernate is an optimal choice. Hibernate also benefits from the ability to perform distributed transactions that span over different types of database and files systems. ActiveRecord however continues to push for its vast reduction in development time because the amount of source code is far less than an equivalent Hibernate application.

## 6 Web Applications

Web applications of today vary in size, demand, and purpose from simple directory search applications that are used by colleges, to huge auction sites that have hundreds of users such as eBay. Also there are applications that are used with corporations support dozens of users or sometimes fewer, and incorporate business logic of the entire enterprise, these applications are also known as enterprise applications. Even though these applications vary in so many ways, there are some similarities that allow us to categorize them into types of applications. This categorization will help us determine the applications that are best suited for the two frameworks.

The first category that we will declare will be known as the massive data load applications which provide services to a large amount of users. These applications usually have a massive amount of data being pushed into the database and being pulled from the database. Also they support hundreds to thousands of concurrent users accessing and sending data to the database. The priority of these applications becomes performance and scalability rather than development time, because of the push to have users operate in a real-time environment. Due to this demand, generally these applications are not written in dynamic languages or Java, but rather strictly customized

compiled languages such as C/C++ and a customized web server. These applications also try to reduce the trips to the database by queuing the requests into an intermediary, which sacrifices some real-time performance but saves on the stress to the database. Examples of these types of applications are Amazon and eBay and other sites that have a massive user base and performance demands.

The second category that web applications could be known as is the small to medium data load applications. These applications offer services to a much smaller user base, with a similar amount of data being passed through each request. Performance is a concern but scalability loses priority until the application is known to expand its user base. Generally these applications have very little optimization with the database, and sacrifice some performance with an out of the box database. An example of this type of application would be Twitter.com or a search directory used by a college.

Another category of web applications can be called enterprise applications, where the scope of the application does not reach outside of the corporation the application is built for. These applications are built to centralize the business process of an entire enterprise, hence the name. Though performance and scalability of each enterprise application can vary, generally since the pool of users is relatively small the scalability and performance are not an important concern. Instead enterprise applications are concerned with industry standard performance and quick development time, therefore these applications are usually written in a plethora of different languages and platforms. Also some of these applications rely on utilizing the powers of the database with triggers and stored procedures, in an effort to centralize business logic. An example of this type of application would be an ERP application that a corporation would use to manage all aspects of their business.

Another category would be simple applications, which should include a wide range of applications. These applications generally have a very simple domain model, database structure, and solve a simple problem. They can extend from a simple web application developed to add to a guest book, or an application that somebody would design within a textbook or a class for software development.

The final category that web applications is legacy applications, either applications that are working with a legacy database or applications that work with legacy code. These applications can vary for performance, scalability, and development time requirements. The main concern of these applications is that when a change happens, that nothing breaks the rest of the system. Legacy applications are different than other applications due to the fact that they are existing systems rather than new systems. Generally these applications are developed in whatever language or platform they were originally built upon or they are development on top of the original software.

## 7 What Framework Works Best?

Now that the basics of each framework have been covered and compared, and there is a categorization of web application attributes, the model to pick the best framework for the solution can now be created. The following section will discuss what applications a particular framework is weak with and what applications a particular framework is strong with. At the end a chart will present this information in a clear and easy way to decide which framework to pick depending on the type of application.

### 7.1 ActiveRecord

There are certain applications and development teams that ActiveRecord and Ruby on Rails is not the ideal solution for. While Ruby on Rails can be used to develop any application, certain applications will actually make Ruby on Rails difficult to develop in. The application are not just what hold ActiveRecord back, but also the structure of a development team and what practices that team follows.

The first and probably most important thing to consider is whether or not your development team is following an agile approach to software development. Meaning you are enforcing Extreme Programming, Test-Driven Development, conversational development with the client, and having everybody from developers to database administrators participate in the development process. ActiveRecord and Ruby on Rails are “opinioned software”, and do not just make assumptions about the application through defaults, but they also make the assumption that your team is following agile methodologies. To quote Neal Ford at the No Fluff Just Stuff Java Conference 2007 in Green Bay, “Those who do not do test-driven development and develop in Rails will fail.” Agile methodologies have been shown to produce development teams that are more productive and develop higher quality applications; however these practices have not been adopted by everyone in the industry. Those that have not adopted these methodologies should stay away from Ruby on Rails and ActiveRecord because they will lose the beauty and the simplicity of the framework.

Along similar details ActiveRecord loses its simplicity when you start working with legacy systems and when the development team is separated from the database administrators. The team loses the benefit of ActiveRecord’s migrations and convention over configuration. Therefore the once elegant Ruby code now begins to become cluttered with overriding defaults. ActiveRecord is also a poor solution for applications that have a huge data load on the database end and require the support for distributed transactions. Most Ruby on Rails applications experience problems of performance and scaling on the database end of the application. This is due to ActiveRecord’s lack of support for caching and other optimization techniques that hinder the framework when



put up against other O/RM frameworks. Though there are very few applications with data load this significant, if your application falls under this category ActiveRecord may not be the best choice.

Now to look at what applications ActiveRecord is an ideal solution for. First and foremost, if you're team is utilizing agile techniques such as test-driven development, agile database techniques, your clients are integrated into your development process, and everyone on your team is involved in every piece of your software, then you benefit most from ActiveRecord. This framework has a lot of functions and abilities that require that you have access to do anything to your database; therefore you should have an agile database administrator. ActiveRecord and the Rails framework is designed to not just support agility but demand it as part of its "opinion".

ActiveRecord works best with applications that have a simple domain model, therefore a simply object to table mapping. Applications that fall under the category of small to medium data load as well as application sthat fall under the simple category ActiveRecord can provide a great solution for. Due to the simplicity of the Ruby programming language and the ActiveRecord framework, Rails developers can create these web applications up to ten times faster than a J2EE developer with decent performance right out of the box. This quick development time is a result of convention over configuration that ActiveRecord can utilize well when developing these types of applications.

Enterprise applications become a little harder to determine whether or not ActiveRecord would be an ideal solution for. Though there have been successful Rails projects both in commercial and enterprise applications, there still have been very few projects that have been convincing to anyone. The debate of whether or not this framework can scale well with enterprise applications is on both sides of the fence with Neal Ford saying that Rails can scale for 99.9% of applications; hwoever you have other professionals such as Ted Neward waiting for that to be proven to them. Then because this debate so far has not been proven to either side, we will make an educated decision based off of how ActiveRecord has faired with other types of applications. If your enterprise application is relatively simple with moderate performance demands, then ActiveRecord will work just fine. However if you have high performance demands and a complicated application with a complicated domain model, then you may run into problems with using ActiveRecord. (N Ford, personal communication, SEP 9, 2006)

## **7.2 Hibernate**

At the 2007 No Fluff Just Stuff Java Conference in Green Bay, Neal Ford had said that Hibernate was the industry standard for object/relational mapping frameworks. And it was proven true was no other technology was used more by the attendees of this

conference than Hibernate. It was again proven true when Neal Ford described that a Ruby on Rails implementation of Hibernate was in the works to benefit from the robustness of Hibernate. The framework has been around for some time now and has pushed to standardize the methods of persistence of an object to a database. Through its rich domain model approach and the ability to cache and optimize many operations, Hibernate allows applications to excel in performance. However like all technologies it has both its strengths and weaknesses.

Hibernate is not ideal for any application that works with a simple domain model and thus a simple application. Hibernate is designed better for applications that have grown out of the simple stages and grown into much more complicated models. If you use Hibernate for a simple application, the result is that the application becomes far more complex to create and maintain than it has to be. Hibernate also is not ideal for applications under the category of massive data load, even though Hibernate is ideal for performance. The data load is too extreme even for Hibernate to handle, and thus application these applications are never developed utilizing Java. Even though Hibernate out of the box comes with ability to work with various systems, most of the stress of the massive data load applications lies in the database end. These applications excel in removing this stress, where Hibernate cannot achieve.

Enterprise web applications is where Hibernate becomes the industry standard for any corporation. Hibernate has the ability to handle much more complicated domain models with ease therefore providing better maintenance and extendibility of these applications. Through the use of caching, performance tuning, and Hibernate support for the creation of and use of stored procedures applications can run at optimal performance with very little trouble. It is because of these same reasons that Hibernate is also a wonderful solution for applications that fall under the category of small to medium data load.

Legacy systems are always difficult to deal with, since most of them have become much more than just old code and data models, but rather a huge mess of old code and data models. Hibernate's design pattern the Data Mapper was designed to better handle data models that were much more complicated and messier than your average data model. Java and Hibernate can be developed to either be the legacy code, if the system was originally developed in Java, or can be embedded within the original code and design. (N Ford, personal communication, SEP 9, 2006)

### **7.3 Conclusion**

In conclusion Table 2 describes a simple chart with the appropriate framework chosen for a specific type of application. The bottom line is that the Ruby on Rails framework is

still young along with ActiveRecord, and they have not proven itself in the production environment. Therefore there should be a great deal of caution when choosing this new technology, but its strength is simplicity and the ease of software development. Hibernate is an industry standard, proven, and therefore a great choice for any application, however there should be caution in picking this as well due to its complexity and increased development time.

<b>Web Application Type</b>	<b>Hibernate or ActiveRecord?</b>
Simple Applications	ActiveRecord
Small to Moderate Data Load Applications	Both
Massive Data Load Applications	Neither
Enterprise Applications (Simple)	ActiveRecord
Enterprise Applications (Complicated)	Hibernate
Legacy Systems	Hibernate

Table 2: Conclusion

## References

- [1] Ambler, Scott W (2007, May 16). Introduction to Object-Orientation. from Agile Data Web site: <http://www.agiledata.org/essays/objectOrientation101.html>
- [2] Ambler, Scott W. (2003). *Agile Database Techniques*. Danvers, MA: Wiley Publishing Inc.
- [3] Bauer, C, & King, G (2007). *Java Persistence with Hibernate*. New York: Manning Publications.
- [4] Halloway, S and Gethland J. *Rails for Java Developers*. Dallas: The Pragmatic Programmers LLC, 2007.
- [5] Johnson, Rod (2004). *J2EE Development without EJB*. Indianapolis: Wiley Publishing Inc.
- [6] N Ford, public presentation, AUG 2007
- [7] Sebesta, Robert W. (2006). *Concepts of Programming Languages*. Boston, MA: Pearson Education Inc.
- [8] Thomas, D., & Hansson, D. (2007). *Agile Web Development with Rails*. Dallas: The Pragmatic Programmers LLC.
- [9] Thomas, D., & Hunt, A. (2001). *Programming Ruby - The Pragmatic Programmer's Guide*. Addison Wesley Longman Inc.

# **Towards Musical Analysis Tools**

Curt Hill and Sara Hagen  
Valley City State University  
Valley City, ND 58072

[Curt.Hill@vcsu.edu](mailto:Curt.Hill@vcsu.edu) [Sara.Hagen@vcsu.edu](mailto:Sara.Hagen@vcsu.edu)

## **Abstract.**

The golden age of musical analysis is now dawning with computer assistance. Traditional analysis of music is a tedious and time consuming process for even the experienced analyst. There are many well known rules for this, but music is an art where a balance must exist between the expected and surprises.

The raw data for such a study abounds, but finding the preferred format is an issue. The data can generally found in two different classes. The first class is something that could be performed. The second class includes the file formats of numerous score producing programs. There are some tradeoffs for either class of data.

Several experimental programs are described. The first detects composition errors. The second recognizes chords and even this is not trivial. The third tabulates chord sequence pairs within collections of works to give statistical insight into the important topic of chord progressions.

## **Introduction**

The entire process of publishing has been revolutionized in recent memory. The time honored method was to write out the manuscript in longhand, have it typed and then typeset. Each copy introduced errors that had to be found and removed. The arrival of word processors and desktop publishing in the 1970s and 1980s streamlined the process considerably. A similar but less well known series of events has done similar things to music composition and publication. The advent of score production programs has saved countless hours in the production of publishable music. In about the same time frame Musical Instrument Digital Interface (MIDI) and a plethora of MIDI-compatible musical instruments has made it possible to play the score directly from the program.

The area of analysis of music is just beginning to benefit from these advances. Music analysis is a well known process of decomposing music into recognizable pieces. Just as a sentence may be analyzed to find its subject and verb so may music be analogously examined. Music analysis is looking for chord sequences, cadences, tonicisations and modulations rather than for nouns, verbs and adjectives. Although the low order techniques are quite different as the subject matter is different, the goal is the same – to find the structure placed there by its author.

Music and grammar analysis have something else in common: both require considerable knowledge of the subject domain and are time consuming and tedious. Tasks that are knowledge based and tedious are exactly the kind to which computer application is most suitable. Therefore, we will consider two important topics: the form of the data on which to begin the analysis and some types of analysis that have been accomplished. The former will focus on the kinds of data available and conversion to usable format. The latter will focus on some specific analyses that have been done as well as what could be done. We will also attempt at least an informal definition of the technical musical terms as we consider them.

## **Form of Musical Data**

Music exists in a number of data formats that are readily available [Castan,2008], but few are easy to process. What is desired is a format that is easy to process and captures as much usable information as possible. As always, these goals are in conflict, so finding the middle ground is important.

The obvious form is the recorded performance. There are a variety of formats for recorded music such as MPG, WAV, AIFF, etc. Although this is the obvious form it also has the obvious problems. Processing an audio file into its separate components, that is the separate instruments, is extremely difficult. The human has a very well developed sense of hearing and an extremely proficient audio processor, yet very few people can identify the next to lowest note played by the piano. In most cases our attention focuses on the highest notes. Of course, as the number of instruments increases the difficulty of

separating out each line of notes also increases. Sadly, we have a substantial way to go in order to render a score from a performance [Bello, 2000].

A much more tractable alternative is Musical Instrument Digital Interface (MIDI) format data. MIDI may be generated via a score program or recorded live from a large variety of electronic musical instruments. MIDI is composed of a series of signals indicating events and is relatively easy to parse. An event is typically the depressing or releasing of a key. This may include velocity information, which describes how vigorously the musician pressed the key. This velocity translates into the loudness of the note. In addition there is also information concerning the particular instrument tone that is being played. The modern digital piano can approximate the sound of virtually any instrument. Although, a Roland piano will never sound just like a Stradivarius violin, the differences in the two are not significant for the purpose of musical analysis.

In contrast to live recordings, either in audio or MIDI format, are the files used by the score production programs. Since a score is seldom finished in one session, these programs must store the data in a file in a retrievable way. An analysis program could parse the file and reconstruct the score as it would be published. The problem with this format is the difficulty of the parse. The manufacturers of such programs are not generally open about the format of the programs. There are at least two exceptions to this that are worth noting: ETF and MusicXML.

ETF is the acronym for Enigma Transportable File. This was an exchange format used by the MakeMusic! company (formerly known as Coda Music) in its Finale product line. The Finale series routinely stores the score in a proprietary format that typically changes with each new release in an upward compatible way. This format was a binary file with no specifications – it was not intended that any other program could process it. However, most of these could also store the score in ETF format. This was an ASCII file that was programmatically simple to read. It appears to be a people readable dump of the contents of the internal tables.

ETF was originally designed as a means to transport files from one version of Finale to another. Coda published the original specifications [Coda, 1998]. Margaret Cahill [Cahill, 2006] expanded these in her thesis, which seems to be no longer available. The types of coding used in ETF has increased over the versions and the available documentation has not. The first author has augmented this documentation as a side effect of constructing an ETF parser, however there is much that is not yet documented. This parser is the basis of several analysis tools to be discussed in the next section.

Unfortunately, MakeMusic! has ceased its support of ETF. Finale 2007 will not write out an ETF file, although it may still read it. According to their documentation it turned into a support problem as users tried to modify the ETF directly and produced inconsistent files. The Finale program could read the file but apparently did insufficient checking which led mysterious errors and eventually to finger pointing between the company and its users. This would appear to be a good corporate decision but disappointing from a short-term research perspective.

A superior format does exist and this is MusicXML. It was designed primarily to be an interchange format for score programs. Unlike ETF, which had little use outside of those who used Finale, it was designed to be able to represent the score in an independent way. It was pioneered by Recordare company [Recordare, 2008] but most of the score programs now support it. This includes Capella, Finale, Sibelius among many others [Recordare, 2008]. Strangely, there is no support for either the direct publication of a score nor the playing of a piece from MusicXML. The underlying assumption is that every person with an interest in MusicXML has one of the score programs, will import a MusicXML file into their score program and then print or play from there.

There are some tradeoffs between a score format like MusicXML and a performance format like MIDI. Both are relatively easy to parse. The score format will contain extra information that is tricky to extract from a performance format. It always contains both a key signature (such as key of Eb) and a time signature (such as 2/4), since these are essential parts of the score. When a piece modulates, that is moves from one key to another, the key signature generally changes on the score. The measure separators (that is the bar lines) should be of no consequence, but can give significant clues as to the structure. For example, a cadence (a sequence of keys that brings the piece to a temporary conclusion) almost always terminates at the end of a measure. Such things can be deduced from a performance but it is a complicated and error-prone process. Moreover, the performance of a score by a person always has minute variations from what is printed.

The comparison, however, is not one-sided. The performance format has some advantages of its own, when the file was created by a talented musician. Although a good musician should stay within close tolerances of the score, there is always room for interpretation that a score does not capture. For example, when looking for a cadence in a MIDI format file, one would look for slightly louder notes along with the other features. This is not evident from the score at all, but the musician has already done a de facto analysis and will emphasize things like a cadences.

## **Analysis Tools**

Even though MusicXML would appear to be the preferred format, for historical reasons an ETF parser was constructed and the results in this section are based upon that. Work on converting the work to MusicXML is proceeding.

Once a parser of the file is complete there are a variety of data available for processing. Clearly the most important form of data is the information on the individual notes. This will necessarily include the note value and octave, as well as starting time and duration. Auxiliary information includes the measure information, which includes its starting time, key signature and time signature. This is the bulk of the raw data that needs to be analyzed. What will be considered next are three experimental programs that perform differing analyses on this raw data.

Document authors have a variety of checking tools available for their use. These include spelling checkers, grammar checkers as well as the determination of reading level. Many



programming languages have types of verifiers as well, such as the lint program which checks for certain potential errors in source code. The first of the described programs does exactly that for compositions. Its intended audience was novice composers. It searches for a variety of conditions that should not normally exist in certain types of compositions. A screen shot of the dialog box that allows condition selections to be checked is in Figure 1.

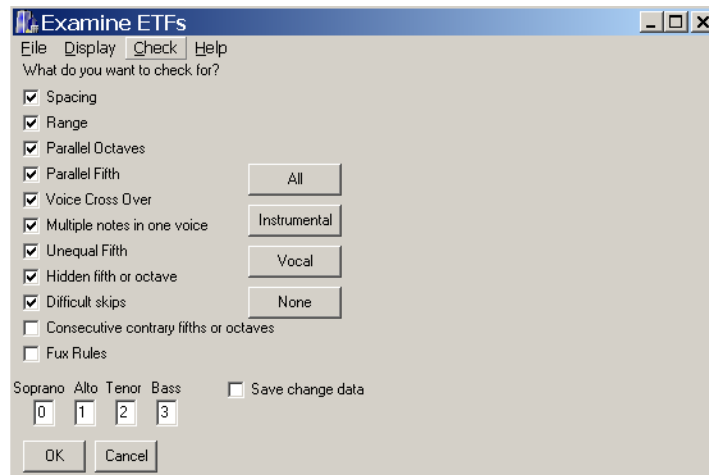


Figure 1. Checking dialog box.

There are several types of checking that may be done. Vertical checking compares a note with the other notes that are to be played at the same time. Horizontal checking looks at successive notes in the same voice. There are other possibilities as well including combinations.

Perhaps an explanation of a few of these conditions is in order. Most of these conditions are checked in a vocal piece for three or four part harmony. The idea of a spacing error is if there is more than an octave between the soprano and alto or alto and tenor voices. This is an example of vertical checking. A range error occurs when a note is beyond the usual range of that voice. This is neither vertical nor horizontal, just a case of demanding too much from a performer. A difficult skip is when a single voice is required to skip an interval that is beyond the average vocalist. This includes a skip of more than an octave, a seventh and the augmented fourth. This would be found with horizontal checking. A parallel octave or parallel fifth error occurs when two voices move from one note to another and maintain the parallel distance. This usually occurs in non-adjacent voices and requires a combination of horizontal and vertical checking.

Is this really necessary? Certainly to the experienced composer these rules are so ingrained as to require no additional work. However, consider the number of possibilities that a novice must deal with. In a ten measure piece, which is little more than an advertising jingle, there might be approximately 40 notes in each voice. This requires about 40 range checks, 156 horizontal checks, 240 vertical checks as well as 117 horizontal and vertical checks. This is more than 500 things to check for in a trivial piece and this does not take into consideration that checking for a parallel octave is slightly different than checking for a parallel fifth. When these numbers are considered, as well as

other conditions that may be selected but not here described, it should be clear that verifying a piece is a formidable obstacle for the novice composer.

A chord is usually three or four notes with well known intervals that are played simultaneously. For example a major chord has a major third interval between the lower two notes and a minor third interval between the upper two notes. This would seem to be a simple thing to recognize, but this is not the case because of the many variations. In actual music any chord may be complicated by inversions, duplicated or omitted notes and extra notes that are not part of the chord. The marvelous human ear can appreciate all of these as more or less equivalent.

An inversion is a rearrangement of the notes in any way other than the standard notation. For example a standard C major chord consists of the notes, C, E and G in the same octave. There are six permutations of this order. They have somewhat different tonality but they are for the most part interchangeable. In a four part harmony at least one of the notes needs to be duplicated, although probably in a different octave. Thus any note may occur more than once. It is also the case that we will recognize the chord even if one of the notes is missing, so that possibility must be accounted for as well. The more difficult problem is non-chord tones. There are a variety of these that have function in the linear structure of the piece but are not a part of the chord itself. Notice that the number of combinations of tones that can form the same chord is quite large and there are a fair number of chords that one can expect in a particular key. The pattern recognition capability of people is rather amazing, for chords with any combination of these items is recognizable, not to mention pleasing. Since chord progressions, that is the sequential motion of chords, is an important foundation for many higher level structures, it is essential that chord recognition be rather robust.

The pattern recognition used involves representing the chord as a set of integers. Ultimately, a chord should fit in an octave. Thus after the notes are represented in the set the range is gradually reduced by plucking off the highest note and adding it back in an octave lower until the entire chord fits in the octave. This will also collapse duplicates into a single note. There are approximately 50 bit patterns that capture the tonality of the chord. Thus a D chord has exactly the same shape (ie. bit pattern) as a C chord, but it is based upon a higher note. If the set matches none of the patterns and has too many notes a backtracking process removes a note and tries again. If the note removed was an extra-chord tone the pattern should be discovered, otherwise that path fails.

This process is embedded in several objects that have a general usage in the analysis process. A simple program that identifies a chord entered as a series of notes is used as a test bed for these objects. Figure 2 shows a screen shot of this program. An explanation of what is shown follows.

The series of notes is entered into the window. Each note must specify the note name, such as D# or G. This may be followed by an optional octave number. Octaves always start with C and end with B, so C4 is one half-step above B3. Once the notes are entered the user may get a simple ID by clicking the ID button or more information by using the Info button.

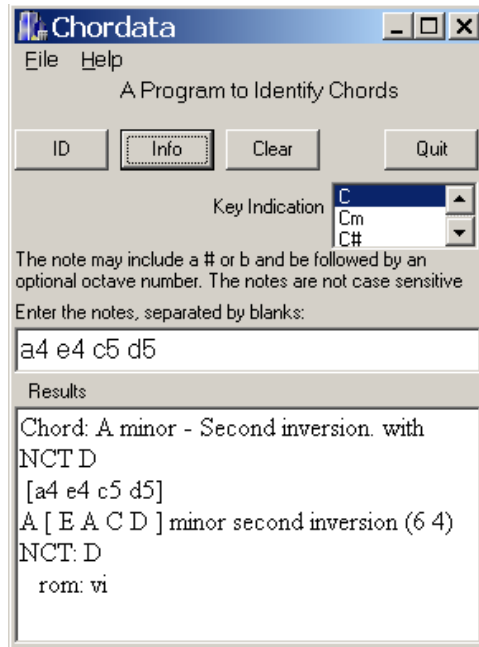


Figure 2. A chord analyzed.

In the example shown four notes are entered. The ID button shows the first two lines. It identifies this chord as an A minor. Since the E and not the A is the lowest note it is labeled as an inversion. The program has also identified the D as a non-chord tone (NCT).

The last four lines results from using the Info button. It gives mostly the same information as the ID button, but offers some additional information as well. It shows the figured base notation (6 4) as well as the Roman numeral designation: vi. The Roman numeral notation is handy because it is independent of key. A piece transposed into another key has exactly the same Roman numeral notation for its chords, which makes comparisons of chords and chord sequences much easier. Roman numeral notation requires knowledge of the key that is being used when the chord is used. The key indication in the screen shot shows that this should be in the key of C major. Within C major an A minor is the sixth chord. An upper case roman numeral indicates a major chord, while lower case signifies a minor chord. Thus the vi signifies a minor sixth chord.

A chord progression is a sequence of chords. Of course, not any random sequence will be enjoyable and an important question is what chord may follow another in a pleasing fashion. The question has many variants depending on whether the focus is on just pairs of chords or on longer sequences. A very common sequence is the IV, V, I. With the ability to find and process large quantities of music this can be considered from a statistical point of view.

A two step approach was attempted to shed some light on this. The first step is to use the parsing of ETF files and the recognition of chords to create temporary file that captured chord transitions. This would provide data that indicate a starting and ending chord as well as measure information. (The form of these temporary files was that of SQL Insert

statements even though a relational database was not used in this experiment.) The second step was that of a tabulator, that would select the data in some fashion and create a table counting transitions.

Statistical approaches are of dubious value with small sample sizes. It is of little interest if one or two Beethoven pieces are tabulated, but twenty would be of some value. Such a tabulator was created and the user may gather as many of the SQL temporary files as possible and create an in-memory database. This database may be selected by composer or by minor or major key and a table produced. Figures 3 and 4 show this program.

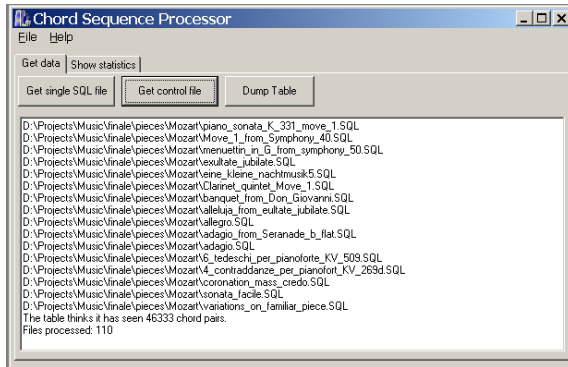


Figure 3. Loading data.

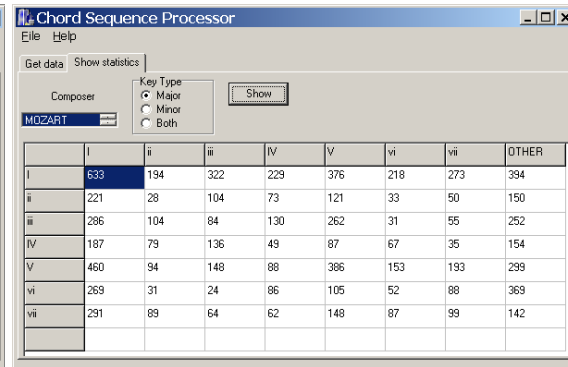


Figure 4. Displaying the table

Figure 3 shows the loading of the temporary files included over 100 separate scores. The bulk of these were obtained from the Finale Showcase [MakeMusic, 2008] which contains a large variety of scores suitable for downloading. A score could be a movement from a symphony or a much smaller piece. In this there were about 30 pieces from Mozart, which will be summarized in Figure 4. A particular piece may modulate through various keys, minor or major, but only those that are in a major key are summarized in the table.

The analysis of this table exceeds the scope of this paper. However, a casual glance asserts that Mozart was more likely to transition a V back to a I, which could hardly be a surprise since this is the basis of many cadences and was common well before Mozart and continues to this day. However, it seems that he was not a big fan of transitioning a IV to a V as is often common elsewhere.

## Conclusions and Future Work

The purpose of this paper was to demonstrate potential, so no conclusions are needed. Yet, it seems clear from Figure 4 that what could have taken hundreds to thousands of hours is now quick and painless. The analysis of the data will still engage us but the tedium of generating it should be done by program.

The authors still have much to do. The construction of a suitable MusicXML parser that will produce data in way parallel to the ETF parser is in process along with the

preservation of the existing analysis tools. The detection and classification of cadences is the next two project that is planned.

The goal of music analysis is to illuminate the structure of the work. Yet at the heart of it is the desire to find why we like some pieces and not others in order that every composition is a pleasing one. If computerized analysis assistance can bolster this effort in any way then it will be time and energy well spent.

## References

- Bello, Juan Pablo, Guiliano Monti and Mark Sandler(2000). Techniques for Automatic Music Transcription. Proceedings of the International Symposiumon Music Information Retrieval (ISMIR 2000). October 23-25, 2000, Plymouth, MA. [http://www.ismir2000.net/papers/bello\\_paper.pdf](http://www.ismir2000.net/papers/bello_paper.pdf). Date accessed 5 March 2008.
- Cahill, Margaret (2006). Publications. <http://www.csis.ul.ie/staff/Margaretcahill/publications.htm>. Date accessed 5 March 2008.
- Castan, Gerd (2008). Musical notation codes. <http://www.music-notation.infor/en/notationformats-a4.pdf>. Date accessed 5 March 2008.
- Coda Music (1998). Enigma Transportable File Specification. <http://www.xs4all.nl/~hanwen/lily-devel/etfspec.pdf>. Date accessed 5 March 2008.
- MakeMusic (2008). Finale Showcase. <http://www.finalemusic.com/showcase/> Date accessed 7 March 2008.
- Recordare (2008). MusicXML Definition, Version 2.0. <http://www.recordare.com/xml.html> and <http://www.musicxml.org/xml.html>. Date accessed 5 March 2008.

# **A Policy-Based Scheduling Tool for Networking Labs**

James T. Yu, Ph.D.

School of Computer Science, Telecommunications, and Information Systems

DePaul University

Chicago, IL 60604

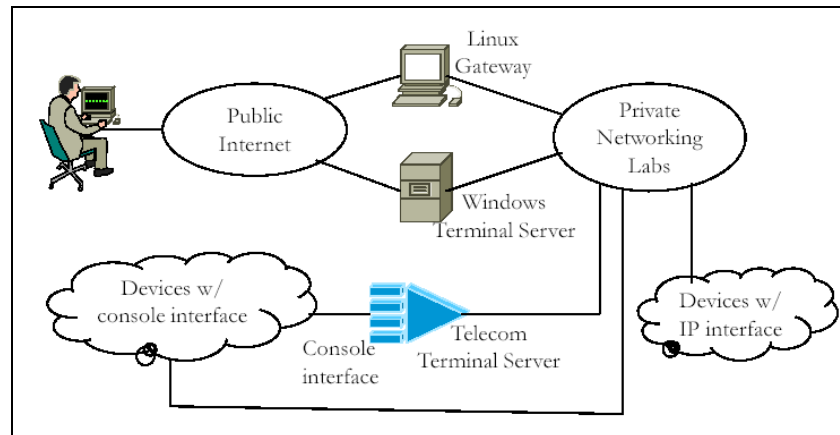
[jyu@cs.depaul.edu](mailto:jyu@cs.depaul.edu)

## **Abstract**

This paper presents a web-based scheduling tool for the networking lab at the School of CTI, DePaul University. The growing demand of distance learning requires an innovative networking lab to support students to perform hands-on exercises from anywhere with 24×7 availability. However, hands-on exercises are mutually exclusive as students cannot perform the same administration task on a device at the same time. Therefore, a scheduling tool is required to avoid interference. Unfortunately, most commercial tools do not meet the requirements for the academic environment where instructors need to manage the lab resource for fair use by all students. To address this issue, we developed a policy-based scheduling tool for our network lab. The tool has been used in our environment for six quarters, and the feedback from the instructors and students is very positive.

# 1 Introduction

Hands-on lab exercises are an integral part of the telecommunication and networking curriculum, and there are many publications discussing the needs and the environment to support it [1][2][3]. With the growing demand to support distance learning for networking curriculums, it becomes a new challenge to support hands-on lab exercises with remote access capability [4][5]. In the summer of 2005, we started a pilot program to support students to access the networking lab from the public Internet as illustrated in Figure 1.



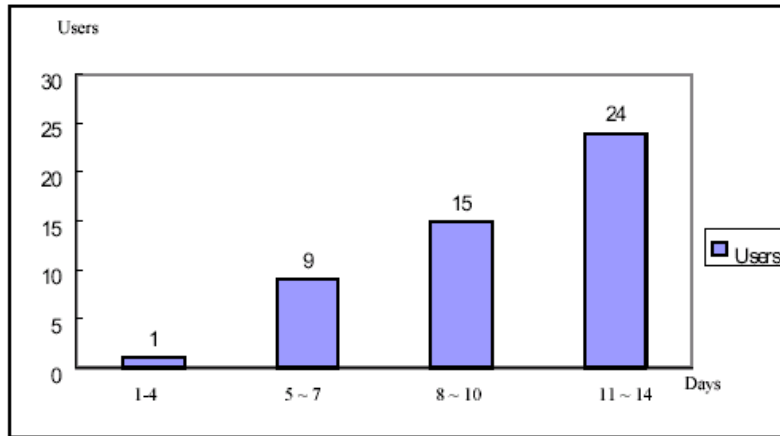
**Figure 1. Remote Lab Access Architecture**

An immediate challenge of this remote lab is the scheduling conflict. A lab configuration can be used by only one student at a time. Before we introduced the remote lab, students went to the physical lab for work assignments. If two students came to the lab at the same time, one would wait until the other completed the work. In the case of the remote lab, students would not see each other. It is possible that one student is doing the router configuration while another student is trying to **reload** the configuration. The pilot program had only nine students during the summer, so this scheduling problem can be easily resolved by using a sign-up paper.

When we decided to provide the remote lab environment for regular classes, it becomes obvious that a sign-up paper would not work as there are frequent requests to change schedules. This issue motivates us to deploy an on-line scheduling tool to support the remote networking lab. We first explored the Internet to search for commercial or shareware tools that could meet our lab scheduling needs. However, most commercial scheduling tools are designed for appointment rather than for management of shared resources. Examples are MS Outlook Calendar, Yahoo Calendar, OrgScheduler [6] and Mimosa Scheduling Software [7]. They are primarily for general purpose calendar events. Although they could be configured for lab scheduling application, they have no or limited capabilities to support the access over the public Internet. In addition, many of those tools require a software client installed on student workstations, while we prefer a simple and standard web interface for lab scheduling.

We implemented a prototype in the spring of 2006, and encountered a unique problem in the academic environment. While students were given two weeks to do each lab exercise, the majority of students would wait until the last day. Although all lab exercises were designed with ease-of-configuration so that students could complete the work in one hour, most student

requests for lab sessions were 2-3 hours. As a result, we could accommodate only 6-8 requests per day. For a typical network class of 30+ students, it is not possible to accommodate all student requests in the last 2-3 days of assignment due, and many complained that the scheduling is not fair for them to complete the lab work. Figure 2 illustrates the non-uniform distribution of lab requests, and this data is collected after we have implemented a strict policy on lab usage. Without the policy enforced in the tool, the situation is much worse.

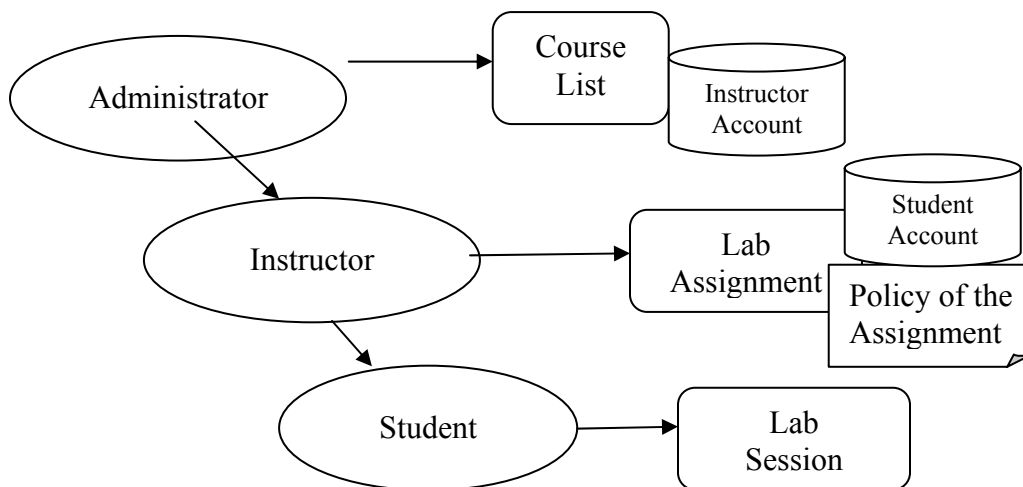


**Figure 2. Non-uniform Distribution of Student Lab Requests**

The problem of non-uniform distribution motivates us to develop a **policy-based** lab scheduling tool, and the policy could vary from classes to classes and labs to labs. The benefit of the tool is to support an effective remote lab environment to share the resources with little administration overhead for the instructors and the lab administrator. This paper presents the motivation, requirements, and design of the lab scheduling tool.

## 2 System Requirements

The objects and the control structure of the scheduling tool are illustrated in Figure 3.



**Figure 3. Objects and Control Structure of Lab Scheduling Tool**



The roles of administrator, instructor, and student are described as follows:

**Administrator** – The lab administrator is responsible for creating the instructor accounts and associating the accounts with the courses assigned to the instructors. The lab administrator is not involved in the scheduling policy which is the responsibility of the instructors. After each quarter, the administrator will clear all accounts (instructor and student), and move the scheduling data into the archive. The administrator is also responsible for the stability and maintenance of the tool, including issue resolution and feature enhancement.

**Instructors** – The instructors are responsible for designing lab exercises and determining the policy for each lab exercise. The attributes of lab exercises are given in Table 1.

**Table 1. Attributes of Lab Assignment**

1.	Lab Assignment ID
2.	Lab Assignment Name
3.	Lab Assignment Description
4.	Course ID
5.	Duration (starting time and due time)
6.	Number of lab sessions per day (limited to one by default)
7.	Number of hours per lab session (limited to 1-3 hours)

In addition to setup the policy for each lab exercise, the instructors are responsible for setting up the student lab accounts. To streamline the process for the instructors, we follow the same format of the student profile in the school database system. Instructors simply download the student file from the school database and then upload the file to the scheduling tool. Each student is given a personal password to access the lab scheduling tool. It should be noted that the password is associated with each course. If a student takes two different networking courses, he/she will have two different accounts. Instructors are also required to reserve time slots to use the lab. However, the instructors are not constrained by the policy as instructors could reserve *unlimited* sessions for class demonstration and technical support. Instructors also have the privilege to grant more lab sessions to students who have special needs.

**Students** – Students use their personal accounts to reserve a lab session. The tool would show only the available lab time slots based on the policy of each lab assignment. A typical example is that students could have one lab session per day, three lab sessions per assignment, and one hour for each lab session. Students may request or release lab sessions based on their availability. Students, of course, cannot release a lab session that is already passed.

Other requirements for the scheduling tool are:

1. An on-line tutorial for instructors to set up the policy and for students to reserve and release lab sessions.
2. A log to track all lab requests and releases. This information is important to track the lab usage and to perform postmortem analysis.

### 3 System Environments

We choose Linux as the development and target production environment due to the ease of using scripting language for task automation. Apache Server 2.0 which comes with the Linux Fedora 2.4 is the development platform. The database is MySQL which is included on the Linux system, and it is a high performance, multi-threaded, and multi-users SQL database server. MySQL server can support data records of up to 4 TB on the Linux kernel 2.4 [8]. Based on the data of two quarters, a networking course with 10 lab exercises and 50 students uses only 1MB of disk space. A typical MySQL server on Linux can handle up to 1,500 transactions/queries per second, and our environment for lab scheduling would peak at no more than 2 per second. Therefore, scalability would not be an issue on this environment.

We discussed between PHP and Perl as the programming language for the web development, where both languages provide effective API to interface with MySQL. We decide to use PHP because we can reuse much code from a similar system, and to use Perl for backend management. The performance of using Linux, Apache Server, MySQL Server, and PHP is also shown to be better than using the IIS server [9]. The only system requirement for end users is the Internet browser. Users can access the scheduling system via any browser which supports HTML and JavaScript. The system environments for the server and clients are summarized in Table 2.

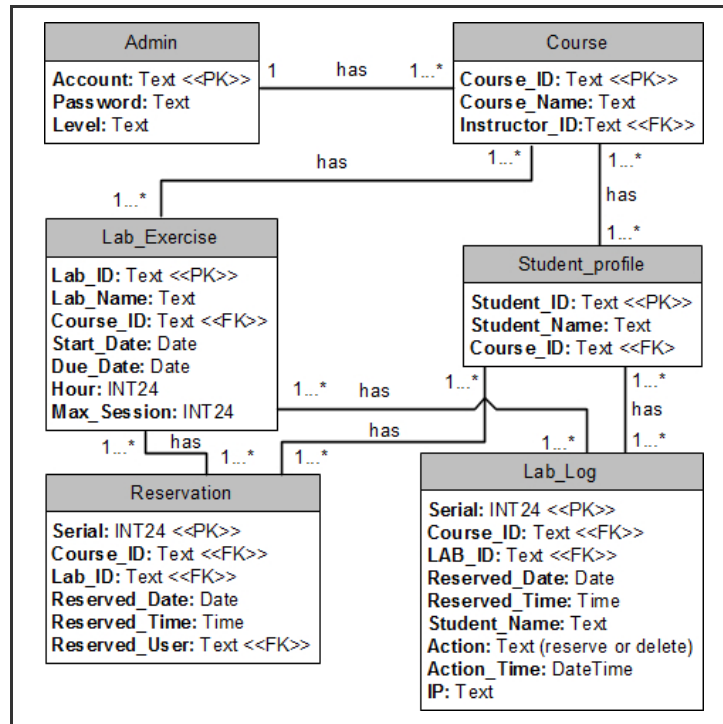
**Table 2. System Environment for the Server and Clients**

Server	Operating System	Linux Fedora core 2.4.20
	Web Server	Apache Server 2.0
	Web Development	PHP 4.3.10, JavaScript and HTML
	Server Task Control	Perl and Shell Script
	Database	MySQL 3.23.49
Client	Web Browser	Internet Explore, Firefox, Netscape

## 4 System Architecture and Design

### 4.1 Data Modeling

The relational data model of the scheduling tool is illustrated in Figure 4. From which, we can see two types of accounts in the admin table. One is for administrator, and the other is for instructors. An instructor has one or more courses, and each course has multiple lab exercises. The database also provides the lab reservation log for postmortem analysis. The Reservation table is the key component of the data model as it contains the actual lab scheduling data. This table also provides the reporting function of showing available time of each lab exercise. The system authenticates each student who follows the policy to request a lab session. Students can access the labs only during their reserved lab time.



**Figure 4. Physical data model for relation database.**

It should be noted that there is a Normal Form Violation [10] in the Student\_Profile table. If a student takes two classes, the student name and ID become redundant data which has the potential issue of data integrity. However, this table is generated automatically from an input file downloaded from the school database system. Therefore, it is unlikely to have a data integrity issue. An advantage of our approach is to avoid additional *join* operation which has negative impact on performance. In summary, this design has the performance advantage without the issue of data integrity

## 4.2 User Interface Design

The interface design is to support ease of use for the administrator, instructors, and students. Figure 5 shows the interface views provided for each user group. All users interface with the system via the web.

### 4.2.1 Interface Design for the Administrator

The Lab administrator is responsible for creating instructor accounts which include account ID, password, and course data. An example of creating an instructor account is given in Figure 6, where the required data includes:

- Instructor Account Name
- Password
- Course ID
- Course Name

A management interface page is provided for the instructor to modify and delete instructor accounts.

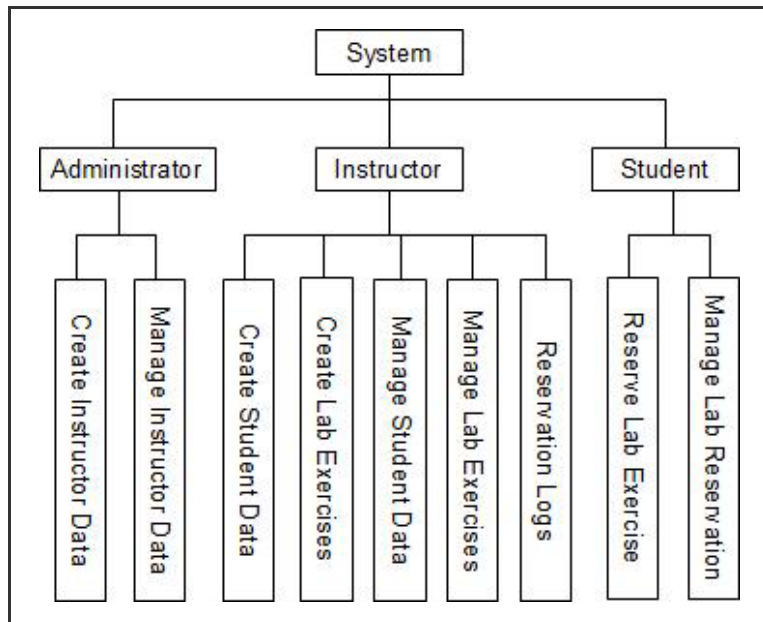


Figure 5. User View of the Scheduling System

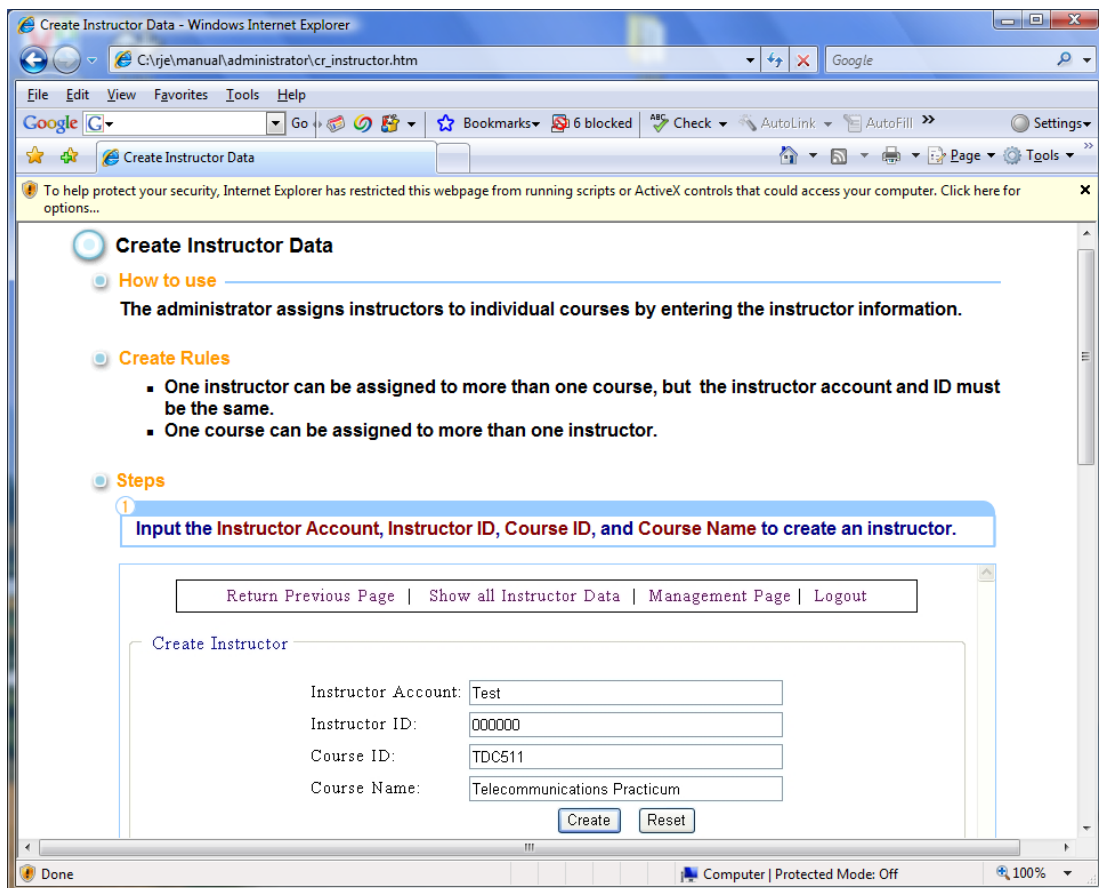


Figure 6. Sample Screenshot of the Administrator Interface

## 4.2.2 Interface Design for Instructors

The 1<sup>st</sup> task for instructors is to create student accounts and the task could easily be accomplished by uploading the student profile (in the Excel spreadsheet format) from the school database system. The PHP program reads the input Excel file and writes the data into the database table for each course. Instructors can use the management interface to delete the student data.

An important feature of the lab scheduling tool is the **policy** specified by instructors for individual labs. Instructors are provided with a management interface to define a lab exercise within a course and the system will create an internal ID of each lab exercise for internal tracking. The management interface for lab creation is illustrated in Figure 7, and the policy data includes:

- The starting and ending dates of a lab exercise,
- The number of sessions per lab exercise allowed for each student, and
- The number of hours for each lab session.

2

Fill in every field in this page. Please do not input the duplicate Lab ID and Lab Name. ( If you want to modify the lab data, please use maintain function. ) After finishing all data filling, click Create button.

Previous Page | Main page of management | Logout

Create the Lab

Instructor: test2

Course: TDC511-Telecommunications Practicum

Lab ID: lab01

Lab Name: STP and RSTP

Start Date: 9-13-2006 (mm-dd-yyyy)

Due Date: 9-22-2006 (mm-dd-yyyy)

Number of lab hours per session: 1 hour

Max sessions for this Lab: 1

Create Reset

Figure 7. Sample Screenshot of Instructor Interface

The system has a general policy that each student is allowed for only one lab per day. This policy is based on the experience that students have the tendency to wait for the last day and reserve multiple lab sessions. With the policy of only one lab session per day, students are

required to plan ahead. After creating a lab exercise, the instructor can use the management interface to modify the lab information or delete a lab exercise. Reservation Log interface provides instructors with the status of student reservation. Instructor may also overwrite and delete a student lab reservation. For example, if a student has a special need for more lab sessions, the instructor can use the management interface to grant additional lab sessions for that student.

### 4.2.3 Interface Design for Students

The web interface for students supports multiple classes from different instructors. When an instructor uploads the student data, the system uses personal data to derive the default password. Only authenticated students can reserve lab sessions. Each lab session has starting and due dates, and students must reserve lab sessions during this interval. The lab reservation is menu-driven and students are given the available time slots (day/hour) from a friendly calendar interface. Figure 8 and Figure 9 are the screen shots of lab reservation interface for students who can select only those dates that are between the lab starting and due dates. We use different colors to show the status of a given day: time slot available (cyan), no time slot available (red), and time slot already reserved (green).

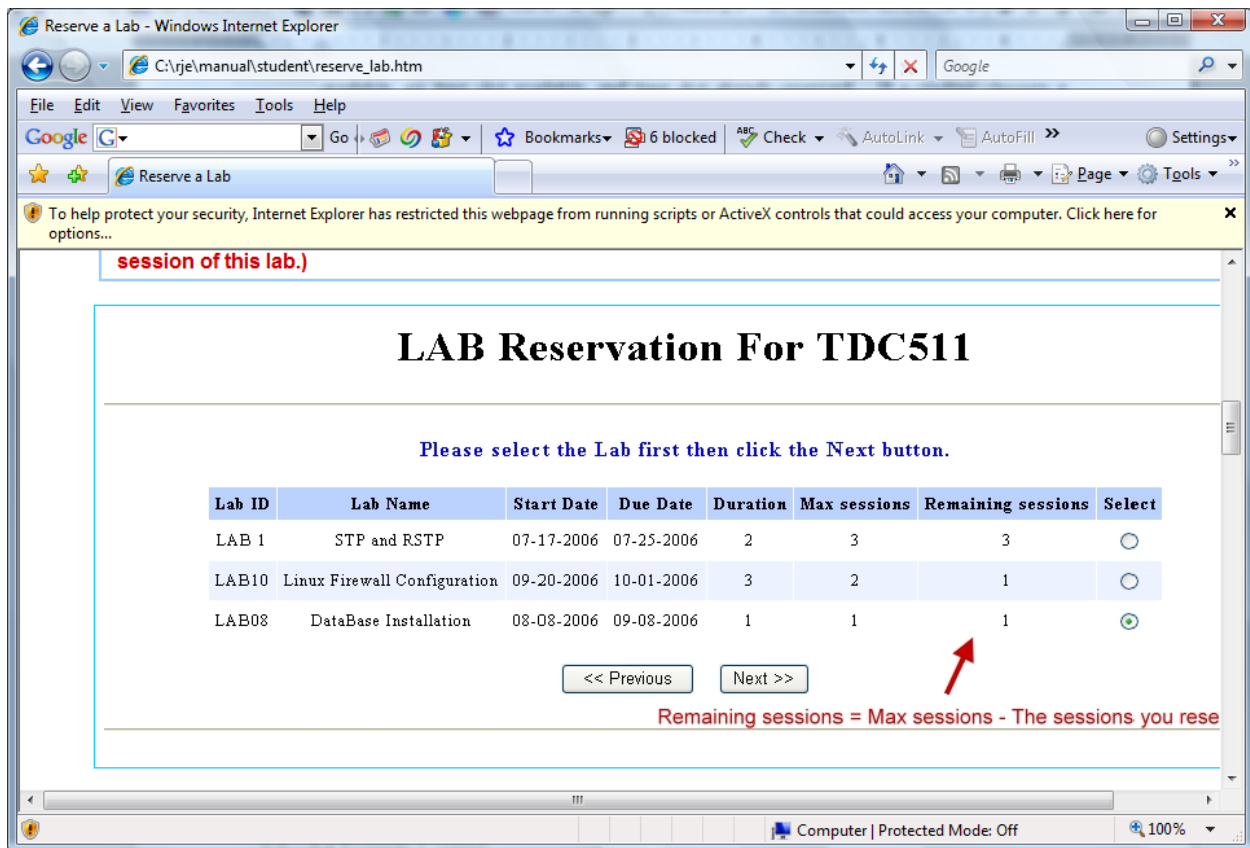


Figure 8. Sample Screenshot of Student Interface

The lab you choose is Lab-B - Frame Relay Configuration.

Please choose the date you want to reserve.

March, 2007						
Sun	Mon	Tue	Wed	Thu	Fri	Sat
*	*	*	*	1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	31

April, 2007						
Sun	Mon	Tue	Wed	Thu	Fri	Sat
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	*	*	*	*	*

Lab time.                       No more available time today.  
 Days in the past or not in lab time.     You have already reserved today.

The date you choose for this Lab is 08-31-2006

Please choose the start time of the Lab.

Time	Select	Time	Select
1:00	<input type="radio"/>	13:00	<input type="radio"/>
2:00	<input type="radio"/>	14:00	<input type="radio"/>
3:00	<input type="radio"/>	15:00	<input type="radio"/>
4:00	<input type="radio"/>	16:00	<input type="radio"/>
5:00	<input type="radio"/>	17:00	<input type="radio"/>
6:00	<input type="radio"/>	18:00	<input type="radio"/>
7:00	<input type="radio"/>	19:00	<input type="radio"/>
8:00	<input type="radio"/>	20:00	<input type="radio"/>
9:00	<input type="radio"/>	21:00	<input type="radio"/>
10:00	<input type="radio"/>	22:00	Reserved
11:00	<input type="radio"/>	23:00	<input type="radio"/>
12:00	<input type="radio"/>	24:00	<input checked="" type="radio"/>

**Figure 9. Screen Shot of Calendar for Lab Reservation**

If a student chooses a date with available time slots, he/she will be presented with a new page showing time slots that are available for reservation. With this function, the system does not need to check schedule conflict because only available time slots could be selected and reserved. The system also provides a management interface for students to view their reservation history with the option to cancel a reservation.

### 4.3 Security Control

#### 4.3.1 Data Control Structure

The system has three level of security control as specified for the administrator, instructors, and students. The administrator controls the data of courses and instructors who control the data of labs and students who control the data of their lab sessions. This control structure is illustrated in Figure 3.

#### 4.3.2 Password Encryption

Each account has its own password which is used for access control. In order to prevent hackers from sniffing password over the network, we use an encryption function of PHP [11] to protect the passwords. The passwords in the database are stored in the encrypted format as well.

#### 4.3.3 Activities Log

All lab activities and Linux server access are logged, including IP address and timestamps. If there is any unauthorized access to the system, the administrator can check the activity log to determine the cause and severity of the activity. Two examples of lab log are given below:

38	TDC511	lab01	09-19-2006	23:00	jyu	Delete	09-11-2006 15:34	140.192.33.153
34	TDC511	lab02	09-19-2006	17:00	jyu	Reserve	09-11-2006-14:01	140.192.33.153

**Figure 10. Examples of Activity Log**

## 5 Conclusions

The development of the lab scheduling tool was complete in the summer of 2006, and the tool has been used for six quarters and 12 networking courses with over four hundred students. The feedback from the instructors and students is very positive. Since the official use, we identified a few omissions in the requirements and enhanced the tool to correct those omissions.

The network lab environment represents a *limiting* resource that needs to be shared by a large student population. With the design of the remote lab, we are able to provide 24×7 service for students to access the networking lab from anywhere with the public Internet. The lab scheduling tool is required to support this remote lab environment so that students would not interfere with each other's work. One unique feature of this lab scheduling tool is the support of a *policy* for individual lab exercises, and the policy is managed by the instructors. Although this scheduling tool is designed for the networking lab in an academic environment, its general concept is applicable to any environment with contention for limited resources. With an established policy that is agreed by all the parties, an organization can use the approach presented in the paper to implement the policy in the scheduling tool. The benefit is fair and effective use of limiting resources with no administration overhead

Additional features are being considered to enhance the tool capability. One enhancement under consideration is to link the student reservation data with access control to the lab equipment. For example, if a student is trying to access lab equipment outside his/her lab session, the access would be denied. A major challenge in providing this capability is that lab equipment is shared by multiple lab exercises and their use changes constantly. The work scope involves the asset management of the lab resources. We are studying different approaches to support the development of this feature.

## Acknowledgement

The author would like to thank Jen-Wei Lai for developing the lab scheduling tool in the summer of 2006, and supporting the tool maintenance until the spring of 2007. The funding for the tool development is provided by the Competitive Instruction Grant from DePaul University.

## References

- [1] P. T. Rawles, "Developing and Supporting a Laboratory based Local Area Network Course," 1999 IACIS Refereed Proceedings, 223-229. Harrisonburg, VA: International Association for Computer Information Systems (pp. 223-229).
- [2] L. C. Hassan, et. al. "A Model for Telecommunications and Networking Technology Curricula," Telecommunications System Management Conference 2004, Louisville, Kentucky (April 2004)



- [3] Ted Mims, “Hands-on Laboratory Based Networking Courses,” Midwest Instruction & Computing Symposium 2002
- [4] S. Yoo and S. Hovis “Remote Access Internetworking Laboratory” In Proceeding of the Thirty-fourth SIGCSE Technical Symposium on Computer Science Education, Norfolk, VA, 2004
- [5] G. Scheets and M. Weiser, “Improvement a Remote and Collaborative Hands-on Learning Environment,” HRD in a Networked World, February 2001, Urbana Champaign, IL  
[http://learning.ncsa.uiuc.edu/ahrd/papers/virtual\\_lab\\_paper2.pdf](http://learning.ncsa.uiuc.edu/ahrd/papers/virtual_lab_paper2.pdf)
- [6] OrgSchedulers, <http://www.binary-house.com/orgschedulers.html>
- [7] Mimosa Scheduling Software, <http://www.mimosasoftware.com/mimosa.html>
- [8] MySQL Reference Manual, <http://dev.mysql.com/doc/refman/5.1/en/full-table.html>
- [9] U. V. Ramana, “Some Experiments with the Performance of LAMP Architecture”, The Fifth International Conference on Computer and Information Technology, 2005, pp. 916-921
- [10] Gerald V. Post, *Database Management Systems*, 2<sup>nd</sup> Edition, McGraw-Hill Publishing Co 2002, pp. 75-79.
- [11] Argerich Luis, *et. al.*, *Professional PHP4*, Apress, 2003, pp. 849-850

# A Dynamic Algorithm for Computing Periodicities of Misère Impartial Games

Thomas McConville  
Department of Mathematics, Statistics, and Computer Science  
Saint Olaf College  
Northfield, MN 55057  
mcconvit@stolaf.edu

## Abstract

A result of the Sprague-Grundy theorem, which forms much of the basis of the field of Combinatorial Game Theory, is that any impartial game in normal-play can be translated into a game of Nim. Since normal-play Nim has a simple solution and these translations may be handled automatically, this theorem effectively solves all impartial games in normal-play. However, no such unifying theory is known for misère impartial games.

We present an algorithm that, given a set of heaps and a ruleset, determines which player has a winning strategy in misère-play by first computing the associated misère quotient. Our implementation accepts a ruleset and an upper bound on the size of heaps and returns the periods and preperiods that exist within those bounds. Documented here are our results for the games 0.3122, 0.31011, and 0.3102.

# 1 Introduction and Background

This paper describes an algorithm for computing outcomes of impartial games in misère-play by approximating indistinguishability quotients. In this section, we provide an overview of the types of games played and the theory supporting the algorithm. Readers more knowledgeable about these topics are encouraged to skip to the Algorithm and Results sections.

The field of Combinatorial Game Theory (CGT) is concerned with solving two-player games of perfect information with no luck involved. Note the contrast with classical Game Theory used to study human or market behavior. CGT has been motivated in part by a desire to solve games such as Go or Chess that despite being very complex, have either a winning strategy for one player or a draw strategy for both players. With sufficient resources, a computer could completely solve either Go or Chess by analyzing every possible sequence of moves, but due to the vast complexity of these games, more elegant approaches are being developed.

Our algorithm is designed to solve *impartial* games, which are games with all resources shared among both players (i.e. every move available to the first player is also available to the second player). The most classic example of an impartial combinatorial game is Nim.

## 1.1 Nim

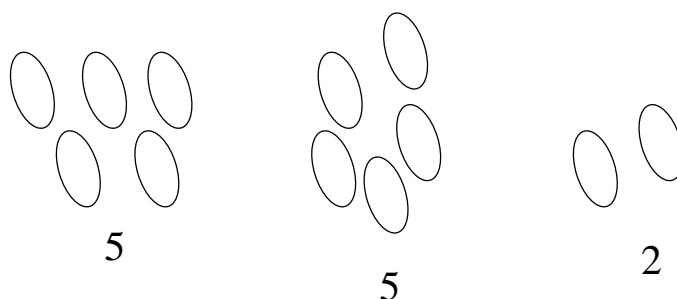


Figure 1:  $(0, 1, 0, 0, 2) = h_2 + 2h_5$

A game of Nim consists of a set of heaps of various numbers of beans. Each player alternately removes any positive number of beans from one heap until all of the beans are removed. In normal-play, the player who is unable to remove any beans loses, or equivalently (excluding trivial cases), the last player to remove beans wins. Like all other impartial games, Nim has no draw conditions. If  $G$  is a game of Nim, we write  $G = (x_1, x_2, \dots) = x_1h_1 + x_2h_2 + \dots$  where  $x_i \geq 0$  is the number of heaps of  $i$  beans. Since Nim games are finite,  $x_i$  is positive for only finitely many  $i$ . Since any impartial game can be expressed in terms of heaps of beans being removed according to some rules, we use this notation for all impartial games.

**Example 1.1** *The ruleset 0.3122 allows the following moves: 1 bean may be removed from any pile, 2 beans may be removed only from piles with 2 beans, 3 or 4 beans may be removed only from piles with strictly more than 3 or 4 beans, respectively, and no other moves*

are allowed. For example, the game  $(0, 1, 0, 0, 2)$ , which consists of one heap of 2 beans and 2 heaps of 5 beans, has moves to the games  $(0, 0, 0, 0, 2)$ ,  $(1, 0, 0, 0, 2)$ ,  $(0, 1, 0, 1, 1)$ ,  $(0, 2, 0, 0, 1)$ , and  $(1, 1, 0, 0, 1)$ .

The game  $(0, 0, \dots)$  is certainly a win for the previous player since the current player has no moves. Any game with a move to the endgame, which in the case of Nim is any game with a single heap, is a win for the current player since the endgame is a win for the previous player. Working backward in this way, we develop a recursive formula for computing the outcome of any game of Nim. If a game has any move to a game that is a win for the previous player, then the game is a win for the current player. Otherwise, it is a win for the previous player. This defines an outcome function  $o(G)$  that maps games of Nim to  $N$  or  $P$  depending on whether the game is a win for the current or previous player, respectively.

There exist simple algorithms for computing the outcome of any game by unpacking this recursion, but these methods are inefficient and yield little information about the structure of Nim or other impartial games. It turns out that Nim and every other impartial game in normal-play has a simple solution. However, when the misère condition is introduced - the player who cannot move wins rather than loses - the analysis used to solve normal-play Nim breaks down. Hence, our work focuses on solving misère-play impartial games, though with minor adjustments, the algorithm could handle normal-play games as well.

## 1.2 Indistinguishability Quotients

Instead of computing the outcomes of all games directly, we partition the set of games into groups with similar “behavior”. To test for similarity of games  $G, H$ , we define the equivalence relation  $=^-$  such that  $G =^- H$  iff  $o(G + X) = o(H + X)$  for any game  $X$ , where  $+$  is just the normal vector addition defined for games. If  $G =^- H$ , then we say that  $G$  and  $H$  are indistinguishable. One pair of indistinguishable games under any set of rules are  $(0, 0, \dots)$  and  $(2, 0, 0, \dots)$  [1]. Since the only move on a heap of size one is to the endgame, having two heaps of size one is equivalent to having zero heaps.

This partition  $Q$ , called the misère indistinguishability quotient, suggests a new approach to computing the outcome of a particular game  $G$ : compute the quotient, find the cell containing  $G$ , and determine the outcome of some game in that cell. Unfortunately, these quotients are not always well-behaved and may not even be finite for a fixed maximum heap size. Moreover, the quotients are difficult to compute since they require an infinite number of comparisons to assert the indistinguishability of two games (relying solely on the definition of indistinguishability). Instead of computing these quotients directly, we find a neat approximation to the quotient.

## 1.3 Quotient approximation

Our approximation to the quotient finds some of the pairs of indistinguishable games, but does not in general find all such pairs. The approximation is thus a finer partition (more cells) than the one given by  $=^-$ . It is formed by finding certain periodic relations in the

outcome function. In particular, for each  $i$ , we seek pairs of games  $G = r_i h_i, H = (r_i + d_i) h_i$  where  $r_i \geq 0, d_i > 0$  are the smallest values satisfying  $G =^- H$ . The vectors  $R = (r_1, r_2, \dots), D = (d_1, d_2, \dots)$  are called the preperiod and period respectively. These quantities are significant because it has been shown that if  $r_i h_i =^- (r_i + d_i) h_i$ , then  $(r_i + u) h_i =^- (r_i + t d_i + u) h_i$  for all nonnegative  $t, u$ .

## 2 Algorithm

Our algorithm for finding  $R, D$  values of a particular impartial game relies on the following theorem by Weimerskirch [1].

**Theorem 2.1** (Weimerskirch) *Fix a ruleset under misère play. Suppose that the outcomes for games of the form*

$$G = (x_1, x_2, \dots, x_{i-1}, r_i, y_{i+1}, y_{i+2}, \dots, y_n)$$

*agree with the outcomes of*

$$G^* = G + d_i h_i$$

*(i.e.  $o(G) = o(G^*)$ ) for fixed  $y_{i+1}, y_{i+2}, \dots, y_n$  and arbitrary  $x_1, x_2, \dots, x_{i-1}$ .*

*In addition, suppose that the outcomes for games of the form*

$$K = (x_1, x_2, \dots, x_{i-1}, r_i + t, x_{i+1}, \dots, x_n)$$

*agree with the outcomes of*

$$K^* = K + d_i h_i$$

*for arbitrary  $x_1, x_2, \dots, x_{i-1}$  and for  $(x_{i+1}, x_{i+2}, \dots, x_n)$  strictly preceding  $(y_{i+1}, y_{i+2}, \dots, y_n)$  in the colexicographic order and all  $t \geq 0$ .*

*Then  $o(G + t h_i) = o(G^* + t h_i)$  for all  $t \geq 0$ .*

The sequence  $(a_1, \dots, a_n)$  precedes  $(b_1, \dots, b_n)$  in the colexicographical order if  $a_i > b_i \Rightarrow \exists j > i \ni a_j < b_j$ . This relation is strict if the sequences are not identical. We write  $\leq$  and  $<$  to refer to the nonstrict and strict relations, respectively.

### 2.1 Partial periods and preperiods

This theorem suggests the notion of, and provides an algorithm for computing, a partial preperiod and partial period, which are valid for all games colexicographically smaller than some known maximum.

**Definition 2.2** *Let  $G = x_1 h_1 + x_2 h_2 + \dots + x_t h_t$  be a game with  $x_t \neq 0$ . For each  $i < t$ , define  $R_G(i), D_G(i)$  to be the smallest nonnegative integers such that  $o(R_G(i) h_i + X) = o((R_G(i) + D_G(i)) h_i + X)$  for all  $X < x_{i+1} h_{i+1} + x_{i+2} h_{i+2} + \dots + x_t h_t$ . Then,*

$$R_G = (R_G(1), R_G(2), \dots, R_G(t-1)),$$

$$D_G = (D_G(1), D_G(2), \dots, D_G(t-1))$$

*are the partial preperiod and partial period, respectively.*

Since we always set an upper bound  $M$  for heap size, the goal of the algorithm is to determine  $R_{h_{M+1}}, D_{h_{M+1}}$ . By the discussion earlier, this would provide a very quick way of determining the outcome of any game  $G$  whose largest heap is  $\leq M$ : Locate another game  $G'$  in the same “cell” as  $G$  by setting for each  $i$  either  $G'(i) = G(i)$  if  $G(i) < R_{h_{M+1}}(i)$  or  $G'(i) = R_{h_{M+1}}(i) + (G(i) - R_{h_{M+1}}(i) \bmod D_{h_{M+1}}(i))$ , otherwise. Then look up  $o(G')$ , which will have been determined when  $R_{h_{M+1}}, D_{h_{M+1}}$  were computed.

**Example 2.3** We find the outcome of the game  $(5, 3, 1, 8, 150, 0)$  under the ruleset 0.3122. It is known that  $R_{h_7} = (0, 3, 1, 0, 3, 2)$  and  $D_{h_7} = (2, 2, 1, 2, 1, 2)$ . Then  $o((5, 3, 1, 8, 150, 0)) = o((1, 3, 1, 2, 3, 0)) = N$ .

## 2.2 Computation of $R_G, D_G$

We use dynamic programming to compute the values of  $R_{h_{M+1}}, D_{h_{M+1}}$ . That is, the values of  $R_G, D_G$  are determined iteratively until  $G = h_{M+1}$  is reached. The *successor* of  $G$  is defined to be the smallest game whose outcome cannot be determined directly from  $R_G, D_G$ , and the list of previous outcomes as explained in Exercise 2.3. Rather, the outcome of the successor of  $G$  is determined by the recursive method discussed earlier of checking whether it is within a move of some  $P$  position. If it is, then  $o(\text{successor}(G)) = N$ ; otherwise,  $o(\text{successor}(G)) = P$ . Then the next successor is chosen and the process repeats until  $\text{successor}(G) = h_{M+1}$  for some  $G$  in the pipeline.

The *successor* algorithm is summarized below.

---

### Algorithm 1 $\text{successor}(G)$

---

```

 $m \leftarrow \max\{i : G(i) > 0\}$ 
for  $i = 1..m$  do
  if  $\text{RD-check}(G, i)$  then
     $G(i) \leftarrow 0$ 
  else
     $G(i) \leftarrow G(i) + 1$ 
  return  $G$ 
end if
end for
 $G(m + 1) \leftarrow 1$ 
return  $G$  {If  $m = M$ , we are done.}

```

---

This algorithm relies on the method *RD-check*, which checks for periodicities. For a given heap size  $i$ , *RD-check* searches for  $a_i, b_i$  with  $b_i > 0$  such that

- (1)  $a_i \geq R_G(i)$ ,
- (2)  $D_G(i) | b_i$ ,
- (3)  $a_i + b_i \leq G(i)$ , and
- (4) if  $G' = \sum_{k=i+1}^M G(k)h_k$ , then  $o^-(a_i h_i + G' + X) = o^-((a_i + b_i)h_i + G' + X)$  for all  $X < h_i$ .

If these four conditions are met, then  $a_i, b_i$  are stored into  $R_G(i), D_G(i)$ , respectively, thus updating the values of the partial preperiod and period. Conditions (1) and (2) verify that the new partial preperiod/period values are consistent with the previous values. Note that if  $R_G(i), D_G(i)$  are not defined, then (1) and (2) are considered vacuously true. In this case, put  $R_G(i) = 0, D_G(i) = 1$  for the purposes the *RD-check*. Condition (3) is necessary since we can only directly compute outcomes of games smaller than  $G$ . Moreover, since the values  $a_i, b_i$  with  $a_i + b_i < G(i)$  have already been tested, we can assume  $a_i + b_i = G(i)$ . Indeed, the first valid  $a_i, b_i$  that the algorithm discovers must be the smallest values satisfying the above conditions.

---

**Algorithm 2** *RD-check*( $G, i$ )

---

```

 $G' \leftarrow G(i+1)h_{i+1} + G(i+2)h_{i+2} + \dots + G(M)h_M$ 
for all  $j$  such that  $R_G(i) \leq j < G(i)$  and  $D_G(i)|(G(i) - j)$  do
  if  $o^-(jh_i + G' + X) = o^-(G(i)h_i + G' + X)$  for each  $X < h_i$  then
     $R_G(i) \leftarrow j$ 
     $D_G(i) \leftarrow G(i) - j$ 
    return true
  end if
end for
return false

```

---

### 2.3 Asymptotic Complexity

The running time of the algorithm is sensitive to the largest allowed pile size  $M$  and the values of  $R_{h_{M+1}}, D_{h_{M+1}}$  for a given set of rules. This complexity has two primary sources: the time spent on determining outcomes of new games and on discovering periodic relations. Note that this whole analysis assumes that the misère quotient is finite.

First, we determine the total time to compute outcomes. Assuming that splitting piles is not a legal move, the number of available moves is  $O(M^2)$ . The quadratic bound is achieved by Nim since a heap of size  $i$  may be reduced to a heap of size  $j$  for any  $j < i$ . It is simple to implement a structure that allows already computed outcomes to be queried in time  $\Theta(M)$ . Finally,  $\prod_{i=1}^M \{R_{h_{M+1}}(i) + D_{h_{M+1}}(i) + 1\}$  is the total number of games whose outcomes need to be computed by the recursive method. Hence, the total time spent computing outcomes is  $O(M^3 \prod_{i=1}^M \{R_{h_{M+1}}(i) + D_{h_{M+1}}(i) + 1\})$ . The product is roughly exponential in  $M$ , but the average base is dependent on the ruleset.

The complexity analysis for finding periodic relations is trickier. For each  $i = 1, \dots, M$ , we find the smallest distinct values  $a_i, b_i$  satisfying  $o(a_i h_i + K) = o(b_i h_i + K)$  for all  $K < h_{M+1}$ . In the worst case, for each pair of distinct  $x, y \leq R_{h_{M+1}}(i) + D_{h_{M+1}}(i)$ , the stored outcomes of all games  $xh_i + K, yh_i + K$  with  $K(i) = 0$  would need to be compared. There are  $\frac{1}{2} \prod_{i=1}^M \{R_{h_{M+1}}(i) + D_{h_{M+1}}(i) + 1\} \cdot \sum_{i=1}^M \{R_{h_{M+1}}(i) + D_{h_{M+1}}(i)\}$  pairs of stored games that differ in exactly one dimension. Due to the empirically small period and preperiod values, we make the simplifying assumption that  $\sum_{i=1}^M \{R_{h_{M+1}}(i) +$

$D_{h_{M+1}}(i)\} = O(M)$ . So the total cost of searching for period/preperiod values is at worst  $O(M^2 \prod_{i=1}^M \{R_{h_{M+1}}(i) + D_{h_{M+1}}(i) + 1\})$ .

Thus, the overall complexity is  $O(M^3 \prod_{i=1}^M \{R_{h_{M+1}}(i) + D_{h_{M+1}}(i) + 1\})$ . If the number of available moves is assumed to be linear rather than quadratic in  $M$ , then this bound is improved to  $O(M^2 \prod_{i=1}^M \{R_{h_{M+1}}(i) + D_{h_{M+1}}(i) + 1\})$ .

### 3 Results

This approach was designed by Weimerskirch in order to glean some information about specific impartial games with infinite misère quotients. These games are not managed well by current methods. We show several examples of games with infinite misère quotients, and for the smallest  $M$  at which the quotient becomes infinite, we show how our approach reveals clear patterns in the partial preperiods and periods. Our implementation is available online at <http://www.stolaf.edu/people/mcconvit/HEAP/>.

For each example, we explain the rules of the game, for which  $M$  the quotient becomes infinite and a list of  $R_{ah_M}, D_{ah_M}$  for the first few positive integers  $a$ .

**Example 3.1** *The quotient of 0.3122 described in Example 2.3 becomes infinite at  $M = 7$ . The results suggest that  $R_{ah_7}(5) = a + 2$  and all other values are constant.*

R1: 0 3 1 0 3 2 0	D1: 2 2 1 2 1 2 0
R2: 0 3 1 1 4 2 0	D2: 2 2 1 2 1 2 0
R3: 0 3 1 1 5 2 0	D3: 2 2 1 2 1 2 0
R4: 0 3 1 2 6 2 0	D4: 2 2 1 2 1 2 0
R5: 0 3 1 2 7 2 0	D5: 2 2 1 2 1 2 0
R6: 0 3 1 2 8 2 0	D6: 2 2 1 2 1 2 0
R7: 0 3 1 2 9 2 0	D7: 2 2 1 2 1 2 0
R8: 0 3 1 2 10 2 0	D8: 2 2 1 2 1 2 0
R9: 0 3 1 2 11 2 0	D9: 2 2 1 2 1 2 0

Table 1: 0.3122

**Example 3.2** *The quotient of 0.3102 becomes infinite at  $M = 12$ . The rules of the game are: 1 bean can be removed from any pile, 2 only from a heap of size 2, 4 beans from any pile with more than 4 beans, and no other moves are allowed. The first three partial preperiods are the same, then  $R_{ah_{12}}(7) = a - 1$  for  $a \geq 3$ . All other values are constant.*



R1: 0 1 1 2 2 2 2 1 2 2 2 0	D1: 2 2 1 2 1 2 2 1 2 1 2 0
R2: 0 1 1 2 2 2 2 1 2 2 2 0	D2: 2 2 1 2 1 2 2 1 2 1 2 0
R3: 0 1 1 2 2 2 2 1 2 2 2 0	D3: 2 2 1 2 1 2 2 1 2 1 2 0
R4: 0 1 1 2 2 2 3 1 2 2 2 0	D4: 2 2 1 2 1 2 2 1 2 1 2 0
R5: 0 1 1 2 2 2 4 1 2 2 2 0	D5: 2 2 1 2 1 2 2 1 2 1 2 0
R6: 0 1 1 2 2 2 5 1 2 2 2 0	D6: 2 2 1 2 1 2 2 1 2 1 2 0
R7: 0 1 1 2 2 2 6 1 2 2 2 0	D7: 2 2 1 2 1 2 2 1 2 1 2 0
R8: 0 1 1 2 2 2 7 1 2 2 2 0	D8: 2 2 1 2 1 2 2 1 2 1 2 0

Table 2: 0.3102

**Example 3.3** *The quotient of 0.31011 becomes infinite at  $M = 5$ . The rules of the game are: 1 bean can be removed from any pile, 2, 4, or 5 beans may be removed only if it empties the pile, and no other moves are allowed. The data settles down after  $a = 10$ , after which  $R_{ah_5}(4) = a - 5$  if  $a$  is odd and  $a - 6$  if  $a$  is even. All other values remain constant after  $a = 4$ .*

R1: 0 1 1 2 0	D1: 2 2 1 2 0
R2: 0 2 1 2 0	D2: 2 2 1 2 0
R3: 0 2 2 4 0	D3: 2 2 1 2 0
R4: 0 2 3 4 0	D4: 2 2 1 2 0
R5: 0 2 3 4 0	D5: 2 2 1 2 0
R6: 0 2 3 4 0	D6: 2 2 1 2 0
R7: 0 2 3 4 0	D7: 2 2 1 2 0
R8: 0 2 3 4 0	D8: 2 2 1 2 0
R9: 0 2 3 4 0	D9: 2 2 1 2 0
R10: 0 2 3 4 0	D10: 2 2 1 2 0
R11: 0 2 3 6 0	D11: 2 2 1 2 0
R12: 0 2 3 6 0	D12: 2 2 1 2 0
R13: 0 2 3 8 0	D13: 2 2 1 2 0
R14: 0 2 3 8 0	D14: 2 2 1 2 0
R15: 0 2 3 10 0	D15: 2 2 1 2 0
R16: 0 2 3 10 0	D16: 2 2 1 2 0
R17: 0 2 3 12 0	D17: 2 2 1 2 0
R18: 0 2 3 12 0	D18: 2 2 1 2 0
R19: 0 2 3 14 0	D19: 2 2 1 2 0
R20: 0 2 3 14 0	D20: 2 2 1 2 0

Table 3: 0.31011

## Acknowledgements

The implementation of Theorem 2.1 was the collaborative effort of Dylan Evans, James Bonebright, and myself. Thanks to Mike Weimerskirch for proposing the problem and teaching us the requisite mathematics. Also, thanks to Olaf Hall-Holt and his Fall '07 Algorithms and Data Structures class for their suggestions.

## References

- [1] M. Weimerskirch. On Infinite Indistinguishability Quotients in Misère Impartial Combinatorial Games.
- [2] T. E. Plambeck. Taming the Wild in Impartial Combinatorial Games. *INTEGERS: The Electronic Journal of Combinatorial Number Theory* 5 (2005) #G05
- [3] T. E. Plambeck and A. N. Siegel. Misère Quotients for Impartial Games. Forthcoming. <http://arxiv.org/abs/math.CO/0609825>.

# Search and Rescue Robots

Binod K.C. and Karen T. Sutherland  
Augsburg College  
Minneapolis, MN 55454  
kcb@augzburg.edu  
suther@navigation.augsburg.edu

This work was supported by National Science Foundation grant #0538740.

## Abstract

When unsafe areas such as collapsed buildings must be entered in rescue operations, the time it takes to find victims often determines whether or not those victims survive. Due to the fact that the area is unsafe, human rescuers are also in danger. One approach has been to send teams of small robots into the area. The rationale is that they will be able to move through small openings, spread out and find victims more quickly than a single robot would be able to do. The danger to human rescuers would also be decreased.

We have developed an algorithm using a tree structure for robot dispersion. This is in contrast to the two most common approaches of using a completely random dispersion or setting up a strong network of connections between robots.

We previously studied our algorithm in simulation using Player/Stage. The current paper discusses the implementation of the algorithm using a team of rescue robots built on the Handy Cricket platform.

We have also been able to test the Cricket's physical communication ability, how successful the bit array data structure we developed is for real robot communication, and have modified our algorithm to degrade gracefully when faced with the hardware failures that we have experienced.

# 1 Introduction

The motivation for this work was given in our previous paper [3]:

A dispersion algorithm is an algorithm that causes many robots to disperse in an area either systematically or randomly, trying to cover the maximum area and gather information collectively.

A number of researchers have developed dispersion algorithms for small robots. Das et. al. [2] used a team of heterogeneous agents, including both humans and robots, in rescue operations. They concentrated on forming strong networks between all participants. Their environment was dangerous in terms of fire or toxins, but was well structured. McLurkin and Smith [5] used ad-hoc communications network topologies formed by gradient floods. Although they paid attention to doors and hallways, the underlying assumption was that the environment was structured with no unexpected obstacles or narrow passages. Reich and Sklar [7] are developing algorithms for automatically reconfiguring robot sensor networks in urban search and rescue. They are using a large number of robots with limited mobility and sensing capabilities combined with a small number of more powerful robots. Their focus is on network configurations rather than dispersion. Damer et. al. [1] developed two distributed dispersion algorithms based on wireless signal intensities. The main difference between their work and what is being presented here is that they used cliques in a graph structure as their data structure as opposed to a tree structure which requires less overhead. Their intended robot of implementation was the University of Minnesota's Scout [8].

Our intention when we started to develop a dispersion algorithm was to extend the algorithm developed at the University of Minnesota for the Scout. We chose to develop a tree structure in an attempt to improve on the execution time of the graph-based algorithm used by the Minnesota group.

The algorithm we developed for growing this tree structure was described in [3]. We developed and tested the algorithm in simulation using an open source Javaclient for Player/Stage [6]. Due to possible obstacles or narrow passages, the tree expands from root out, with nodes being "pushed" down from the root to fill the space. The tree grows in a depth-first/breadth-first manner, branching out into large spaces and snaking through small passages.

Our next task, described in this paper, was to implement the algorithm using real robots. We did so using the Handy Cricket. Although robots used in actual search and rescue would have to be sturdier than the Crickets, we have used them to address issues involving limited communication and dispersion with real hardware. Player/Stage is a two dimensional environment. The Crickets exist in a more realistic three dimensional environment. Their limits in sensing and power are real. Each robot communicates only with parent and children (currently a maximum of three each). The tree structure is known only locally and consists of a bit string reflecting whether a given robot has a left, center and/or right child.

We do not add extraneous robots to our tree. As an example, if a narrow passage exists, a robot would sense an obstacle in the left and right positions and only request a center child. The robots would then be “pushed” through the narrow passage, separated by the maximum distance of their communication capabilities, forming a single line.

The same approach is used when a victim is found. A bit string consisting of representation for left, center and right at each position is sent back to the root.

## 2 Our Handy Cricket Robots

The Handy Cricket board [4] is a PIC microcontroller-based robotics platform with input/output and motor/sensor control. It weighs 0.2 pounds and has dimensions of 3x3x0.5 inches. It is powered by four AA batteries, has two motor outputs, two sensor inputs, and two bus connection lines. Figure 1 shows the Handy Cricket board as viewed from above.

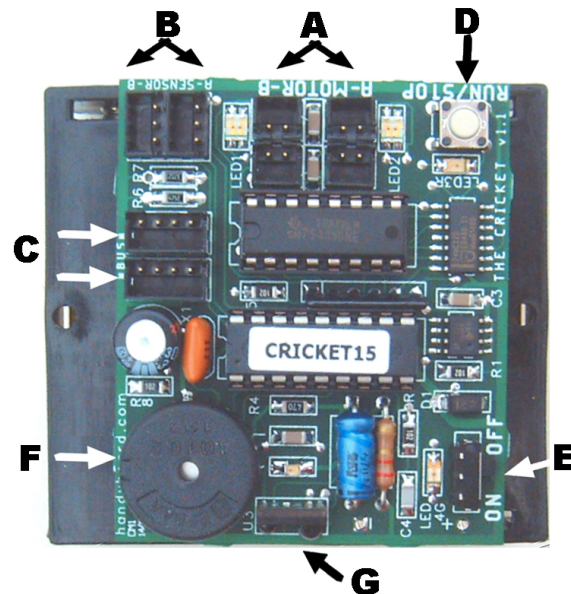


Figure 1: (A) Motor Output[2], (B) Sensor Input[2], (C) Bus Connections [2], (D) Run/Stop button, (E) On/Off Switch, (F) Piezoelectric Speaker and (G) infrared transceiver. Photo courtesy of the Cricket website.

The Handy Cricket can be expanded by connecting a 4-Digit LED number display, a servo motor control interface, a lamp/relay driver and/or a motor/sensor expansion board to one of the buses. The motor/sensor expansion board adds two more motor ports and four additional sensor inputs. All sensor inputs of the Handy Cricket can be used with both digital and analog sensors (0-255 range).

With the goal in mind of using as little sensing as possible, we limited our Crickets to the use of the infrared transceiver for communication and two photocell light sensors, one

mounted on each corner of the front of the robot for obstacle avoidance. The infrared transceiver allows for inter-robot communication up to approximately one meter. The maximum range of values for a light sensor is 0-255 where 255 denotes the sensor is not blocked at all and decreased values denote some blockage in light. Thus, the smaller the value, the closer an obstacle. Figure 2 shows one of the Crickets. We have also built a simulated destroyed environment in which to run the robots.

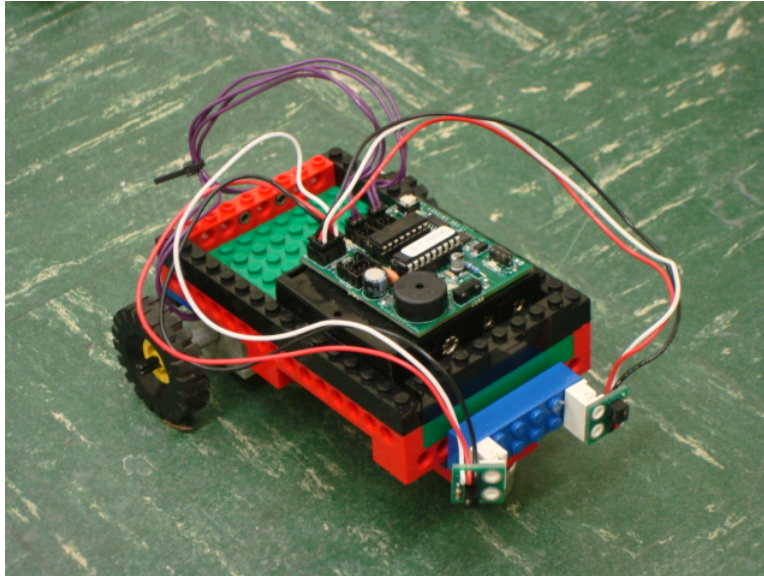


Figure 2: One of our Cricket robots. All robots are configured identically.

### 3 Programming the Handy Cricket

The Handy Cricket provides 4096 bytes of non-volatile storage for user programs. It is programmed using a version of Logo called “Cricket Logo.” Code is downloaded through an external communications board which connects to the host computer’s serial port. This board communicates with the Handy Cricket’s on board infrared transceiver. Due to the small amount of memory, code must be carefully crafted. All Crickets contain the same code. There is no central control.

A section of code which is run on each robot to update the local data structure, described in Section 4, follows:

```
to CONDITIONCHECK
// These are the condtns where it decides how the children change
if ir = 1 [
    setinc ir
    aset child 0 1
```

```

    aset child 1 0
    aset child 2 0
    wait 30
    WAITFORACHILD
  ]
if ir = 2 [
  setinc ir
  aset child 0 1
  aset child 1 1
  aset child 2 0
  wait 30
  WAITFORACHILD
]
if ir = 3 [
  setinc ir
  aset child 0 1
  aset child 1 1
  aset child 2 1
  wait 30
  WAITFORACHILD
]
if ir = 4[
  NEXTPHASE
]
END

```

## 4 Local Data Structure

We have used a maximum branching factor of three for our tree. Thus, each robot can have up to three children. Two single arrays, each of length three bits, hold local information for each robot. This could easily be changed to implement a larger branching factor.

Position	0	1	0
Children	1	0	0

Table 1: Each robot's bit array holds information on its own position and the position(s) of its children.

As shown in Table 1, each robot uses a bit array to store its own relative position in the tree plus whether or not it has a left, right or middle child. The array in the table reflects a robot in the middle position with only a left child.

The first robot to enter the area becomes the root of the tree. Its three bit position array has the middle bit set to 1. Its three bit children array is zero filled because it has no children at

this time.

The root checks ahead. If it sees space to be explored, it calls for a child. As soon as a robot answers that it is willing to become part of the tree, the root prepares to move forward, giving its root position to the new robot. It follows the following in a recursive fashion:

- If Children 0 , 0 , 0 go LEFT (considered default)
- If Children 1 , 0 , 0 go MIDDLE
- If Children 1 , 1 , 0 go RIGHT
- If Children 1 , 1 , 1 then go Left again

This forces the tree to fill left to right. Whenever an obstacle, such as a wall, is in the way of its progress in a given direction, it moves on to the next option. After reaching its designated destination, it sends back a confirming infrared signal and the robots make the required changes to their arrays.

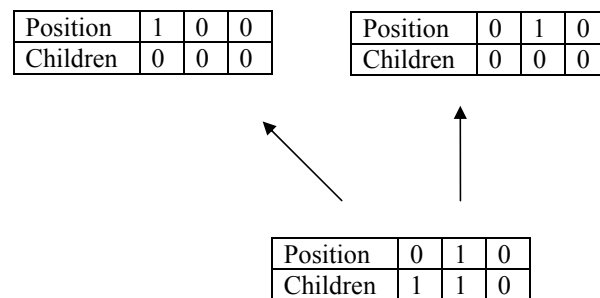


Figure 3: The bit arrays for the first three robots in a tree.

After three robots have been added to a tree, the arrays look as shown in Figure 3.

In Figure 4, there are seven robots in the tree. The robot in the right foreground is the current root and is located at the entrance to the area. The left leaf robot will be the next to request a child. It will be a middle child because expansion will occur down the passage straight ahead of it. The other two leaves will not request children because their sensors will report the walls ahead of them as obstacles. The left leaf robot's request will propagate back to the root, an eighth robot will become the new root and all ancestor robots of the left leaf will move forward, with the left leaf moving into the passage ahead. If it senses space in that direction, it will request another child. Communication is local, between parents and children only, eliminating the processing overhead of a strongly connected network of robots. Also, jams in the initial narrow passage, a common problem associated with random dispersions are eliminated. (The cricket has only one infrared transceiver on the front, so



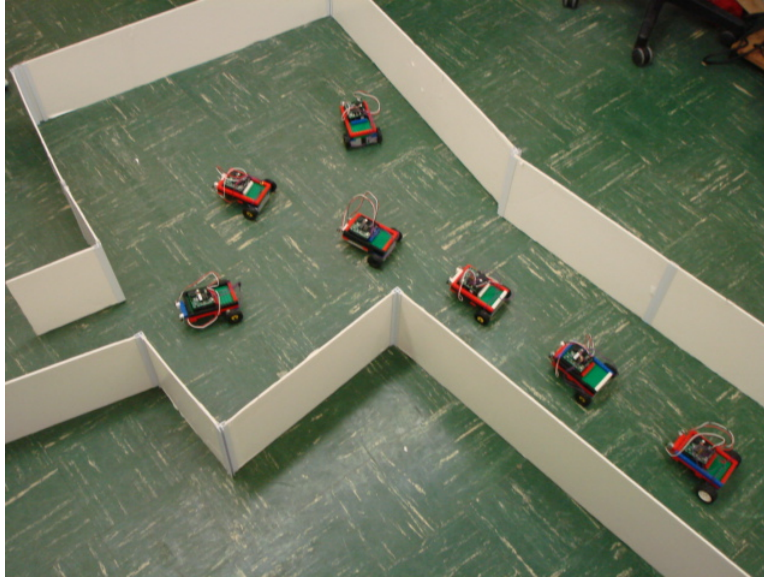


Figure 4: Dispersion of seven robots.

in order to communicate with its parent, who is behind it, the robot is required to rotate 180 degrees and back. This would not, of course, be necessary with more sophisticated hardware.)

One problem we have encountered and which we continue to work on is that a Cricket may respond to an IR signal that was intended for a different Cricket. As with the requirement to turn described above, hardware allowing each Cricket to have its own signal would solve this problem. Attaching a unique identifying bit string to each robot is also a possible solution. We are working on the latter.

## 5 Navigating the Environment

As was seen in Section 4, while they are building the tree, the robots must navigate an unknown, unstructured environment with minimal sensing.

As a robot moves, the obstacle avoidance module continues checking the light sensor values until the values reaches 150 or less, which means there is an obstacle in that direction. If the left sensor values decreases, the robot reverses the motors, moves backward for a second, and turns right about 60 degrees. Similarly, if the right sensor value decreases, it reverses and turns left.

Several trials were done using this simple procedure. It worked quite well unless the Cricket was facing a sharp corner as shown in Figure 5. When in a corner, the robot would sense a wall on its left, turn right and sense the other wall, turn left, and continue moving back and forth indefinitely. To solve this problem, a separate condition was created which checked

both the light sensor values at the same time instead of checking them separately. If both sensor values are below 150 at any instant, the cricket moves backward for one second and then turns. Whether it turns left or right is determined by comparing the sensor values. It turns 135 degrees in the direction of whichever sensor has the larger reading. This has helped avoid the “stuck in the corner” problem, which has plagued a number of robot dispersion algorithms when used with real robots.

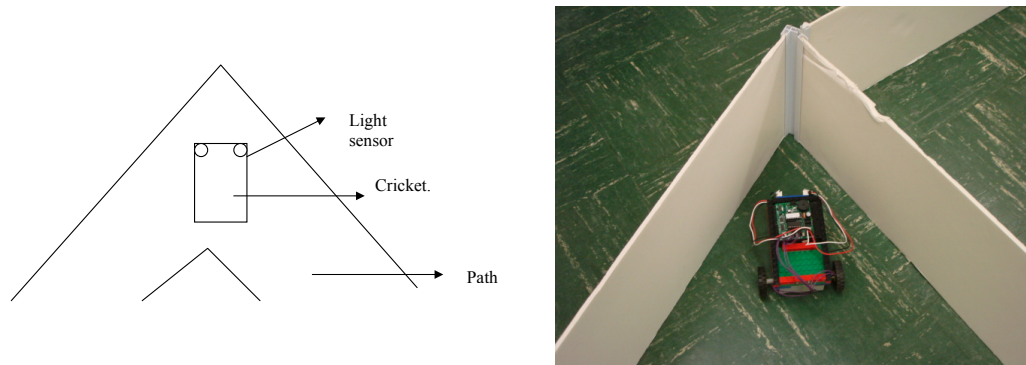


Figure 5: The diagram on the left and photo on the right show the robot facing a sharp corner.

## 6 Conclusion and Future Work

Trials thus far have shown

- a decrease in dispersion time compared to using both a random dispersion and a strongly connected network.
- a need for fewer robots than in using a random dispersion.
- a workable system that depends on minimal communication, as opposed to using a system with a strong network of connections.

Future work includes

- pursuing a solution to the IR response problem described in Section 4.
- running trials on the effectiveness of passing a bit string back to the root containing victim location.
- testing robot behavior in maze-like environments with numerous dead ends.

## References

- [1] Stephen Damer, Luke Ludwig, Monica Anderson Lapoint, Maria Gini, Nikoloas Papanikolopoulos, and John Budenske, *Dispersion and exploration algorithms for robots in unknown environments*, Proceedings of the 2006 SPIE Meeting, SPIE, April 2006.
- [2] A. Das, G. Kantor, V. Kumar, G. Pereira, R. Peterson, D.Rus, S. Singh, and J. Spletzer, *Distributed search and rescue with robot and sensor teams*, Proceedings of Field and Service Robotics, Japan, July 2003.
- [3] Lava KC, *An algorithm for dispersion of search and rescue robots*, Proceedings of the Midwest Instruction and Computing Symposium, 2007.
- [4] Fred Martin, *The Handy Cricket*, <http://www.handyboard.com/cricket>.
- [5] James McLurkin and Jennifer Smith, *Distributed algorithms for dispersion in indoor environments using a swarm of autonomous mobile robots*, Proceedings of DARS2004, 2004.
- [6] Esben Ostergaard, *Javacient code for Player/Stage*, <http://java-player.sourceforge.net/documentation.php>.
- [7] Joshua Reich and Elizabeth Sklar, *Toward automatic reconfiguration of robot-sensor networks for urban search and rescue*, First International Workshop on Agent Technology for Disaster Management (ATDM): Fifth International Joint Conference on Autonomous Agents and Multiagent Systems (Hakodate, Japan), ACM, May 2006.
- [8] Paul E. Rybski, Ian Burt, Andrew Drenner, Bradley Kratochvil, Colin McMillen, Sascha Stoeter, Kristen Stubbs, Maria Gini, and Nikolaos Papanikolopoulos, *Evaluation of the scout robot for urban search and rescue*, Proceedings of the AAI 2001 Mobile Robot Competition and Exhibition Workshop (Seattle, WA, USA), August 2001.

# An Exploration of Implementing the A\* Algorithm Under Limited Resources in Lego Mindstorms

Ammon Horn  
Computer Science Student  
Graceland University  
Lamoni, IA 50140  
[horn@acm.org](mailto:horn@acm.org)

## **Abstract**

One of the most important aspects of the Artificial Intelligence movement is pathfinding, of which a popular algorithm is the A\* algorithm. The goal of this project was to implement the A\* algorithm using the Lego Mindstorms Invention System 2.0 with brickOS, a firmware replacement for C++. The challenge in implementing this was the limited memory and other resources on the RCX brick that would otherwise be available on a workstation. There were several challenges to be overcome. The first was how to represent the state of the grid in as small a space as possible. The second challenge was modifying the algorithm to allow for discovery of the world. The last major challenge was to code the application so that it occupies a minimal amount of space in the robot's memory but still offers maximum functionality. The memory limit on the RCX brick proved insurmountable to solve these challenges.

# 1 Introduction

When building a mobile robot there are numerous considerations that must be dealt with, but providing a good movement algorithm is arguably the most important. My definition of a good movement algorithm is an algorithm that enables travel from point A to point B in 2D space with moderate obstacle intelligence. The obstacle intelligence does not need to be extremely robust, but it does need to have a basic avoidance component. This process of movement with obstacle intelligence is called pathfinding, with the most popular algorithm being the A\* Search Algorithm.

## 2 Pathfinding

So how should movement be implemented in a mobile robot? The simplest approach is to allow the robot to wander without giving it any sense of object avoidance, or path choosing. While this approach is simple, it invariably will create problems the moment the robot encounters anything in its way. Pathfinding in its simplest form allows for movement with a small measure of obstacle avoidance. [1] For example if the obstacle forms a wall covering over 180° of a circular area, the robot will go into the obstacle area and follow the wall until it circles the obstacle. (Figure 1)

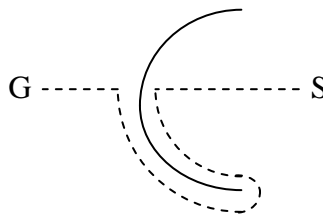


Figure 1: Pathfinding Concave Error

### 2.1 Simple Pathfinding Problem

This can be very costly in terms of both time and effort. There are simple workarounds for this problem: treat the obstacle as a solid object or mark it to be entered only if the goal is inside the shape. (Figure 2)

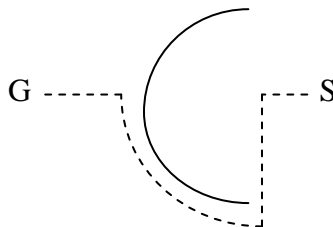


Figure 2: Pathfinding Concave Error Fix

---

<sup>1</sup> Stout, <http://www.gamasutra.com/features/19970801/pathfinding.htm>

However this is only possible if the size, shape and nature of the obstacle is known beforehand. Also the path has no guarantee of being the most efficient; it is simply more efficient than the previous example.

## 2.2 Simple Pathfinding Solution

The best way to avoid problems such as this is to use a more complex pathfinding algorithm. Over the years several have been proposed ranging from the simple breadth-first or the depth-first search algorithms to the complex A\* search algorithm. The breadth-first algorithm starts from the beginning node and examines every neighbor node. After the first tier of neighbors, the algorithm examines the next layer of nodes (nodes two steps away) and continues until it discovers the goal node. On the flip side the depth-first algorithm follows the starting nodes descendents until a specified terminating distance is reached. [2] Both of these algorithms will eventually discover the goal, but they are both costly in terms of time and effort required. But more importantly they do not guarantee the most efficient path. This guarantee is the specialty of the A\* algorithm.

## 3 A\* Search Algorithm Overview

The A\* search algorithm is arguably the most implemented method of optimizing the pathfinding process. [3] This is easily explained by the advantages that the algorithm offers. An important advantage is the scalability of the algorithm. The algorithm can be easily scaled up to large map systems and be modified to account for terrain movement costs. The algorithm also requires less time to execute than a straight breadth-first because it uses a heuristic to minimize the number of nodes searched and traveled to. This is important because using a heuristic guarantees the algorithm will find a solution if one exists. The heuristic used by the algorithm is extremely simple, but powerful. In its simplest form the heuristic is the cost of getting to the current node from the start node added to the estimated cost of getting to the goal node from the current node. [4][5] This value is generally represented by  $f(n)$  where  $n$  is the current node.

### 3.1 Heuristic Creation

The heuristic function  $f(n)$  consists of two portions added together:  $g(n)$  and  $h(n)$ . The  $g(n)$  portion of the function is the piece that calculates the cost of getting to node  $n$  from the start node. The other portion of the function is  $h(n)$ , which estimates the cost of traveling from node  $n$  to the goal node. The equations for these parts are simple to implement and understand.

---

<sup>2</sup> Stout, <http://www.gamasutra.com/features/19970801/pathfinding.htm>

<sup>3</sup> Patel, <http://theory.stanford.edu/~amitp/GameProgramming/AStarComparison.html#S3>

<sup>4</sup> Jones, Page 27 – 28.

<sup>5</sup> Patel, <http://theory.stanford.edu/~amitp/GameProgramming/Heuristics.html#S1>

The implementation of the  $h(n)$  portion of the algorithm ranges from the simple to the highly complex depending on the nature of movement on the map and the level of optimization. The simplest form of the equation is the Manhattan distance equation. (Figure 3) However the equation can be modified to allow the cost to represent any value necessary. In the simplest form the cost is equal to 1.0 which represents no cost in traveling from one node to another. This value may be changed as necessary to account for differing terrain types. For example if the robot is unstable over rocky terrain, set the cost equal to 2.0 or higher so that the robot will prefer the easier terrain. However changing the value of the cost can have dire consequences if the heuristic ever overestimates the distance to the goal. When this occurs the algorithm may choose a poor path because it is considered “better” than the true best.

$$h = cost * (abs(node_x - goal_x) + abs(node_y - goal_y))$$

Figure 3: Manhattan Distance Equation in A\* [6]

The second portion of the heuristic is the cost of getting to the current node,  $g(n)$ . This equation in its simplest form consists of two key parts: the parent nodes  $g$ -value and the alpha value. Since the simple version of the  $g(n)$  function only calculates the cost of traveling between the parent node and the current node, it is necessary to add in the sum total of the previous  $g(n)$  values to get an approximate value for the cost of traveling. The most robust way to setup  $g(n)$  however is to create a scalable equation. This will allow the robot to dynamically switch between navigating based solely on whether the adjacent node is available for travel to navigating using intelligent movement. The way to create this is to add the value *alpha* into the equation. (Figure 4) This *alpha* must consistent of a real number from 0 to 1 inclusive.

$$g(n) = 1.0 + alpha * (g(n)_{parent} - 1.0) + g(n)_{parent}$$

Figure 4: Cost of Movement between Nodes in A\* [7]

The final portion of the heuristic is the combination of  $g(n)$  and  $h(n)$ . (Figure 5) This value is used by the algorithm to rank each of the possible path nodes to ascertain the most efficient path to take. It represents the cost of getting from the start node to the goal node passing through the current node.

$$f(n) = g(n) + h(n)$$

Figure 5: Total Cost of Movement Passing through n [8]

### 3.2 Algorithm Mechanics

The basic structure of the A\* algorithm is simple to understand, easy to code and simple to customize, but it is not without its shortcomings. The general structure of the

<sup>6</sup> Patel, <http://theory.stanford.edu/~amitp/GameProgramming/Heuristics.html#S8>

<sup>7</sup> Jones, Page 29.

<sup>8</sup> Jones, Page 30.

algorithm uses two distinct lists: an open node list and a closed node list. Each of these lists serves to keep track of which nodes have been examined and which have yet to be examined. Each of these lists will be checked several times each time a new successor node is discovered which will result in an eventual bottleneck of resources. When looking at this issue it may be natural to assume that hit taken on the resources is likely to be minor in the grand scheme. Unfortunately, when implementing this algorithm on limited resources, resource management must be considered carefully.

### 3.2.1 Algorithm Mechanics – Resource Management

There are numerous data structures that might be used to store these lists, but to better understand the benefits of any particular data structure, a list of the operations necessary must be created. Considering the open list there are four operations that will occur, some more likely than others. These primary operations consist of finding the best node and removing it, checking if the successor node is in the list, inserting the new nodes into the list, and replacing an existing node with a newer version of itself. [9]

Knowing what these steps are is the first step to grasping the amount of work the lists will undergo. But it is not enough to know the operations, the number of times they run is also necessary. The operation used the least will be the replacing of an existing node simply because the algorithm usually does not cause backtracking. On the flip side, every time a successor node is found, the list will be searched for a previous version of the node. So deciding which data structure to use is based on the desired speed of execution and ease of access.

### 3.2.2 Algorithm Mechanics – List Storage Methods

The simplest method of storing the nodes is that of an unsorted array or list. Using one of these data structures it is easy to add an item, but time-consuming to find the best node and check for existence in the list. In addition the array or list would have to be searched in its entirety to determine the best node. A more optimized data structure would be an indexed array. This would allow the algorithm to quickly test for membership. This data structure is ideal when the number of nodes is finite and is limited in size, but memory consumption is extensive because memory must be allocated for every node before execution. [10] Even worse than the memory issue, the cost of removal on an indexed array is extremely high because the index would have to be reordered whenever an element is removed.

So what is the best possible solution? The answer to that question lies in the implementation details. If the map is large it might be better to use a hash table or some other memory limiting data structure. However, if performance is the goal implementing a hot queue or a hybrid data structure would allow for a faster response. Obviously there

---

<sup>9</sup> Patel, <http://theory.stanford.edu/~amitp/GameProgramming/ImplementationNotes.html#S3>

<sup>10</sup> Patel, <http://theory.stanford.edu/~amitp/GameProgramming/ImplementationNotes.html#S3>



is no easy answer to which data structure is the best overall, so careful consideration must be given to the specifics of the implementation.

The representation of the grid is also an important consideration. What is the best way to represent a grid? The answer again depends on the particular implementation details because what works great for one implementation is ill-advised for another. The implementation design can also play into the decision if there is a limit to the amount of resources to which the robot has access.

## 4 Resource Limitations

This brings me to the problem at hand, developing an implementation of the A\* Search Algorithm that is capable of running on the limited resources of the Lego Mindstorms Robotics Invention System 2.0. From the beginning I had to make a decision about whether to use Lego brick code or replace the firmware with an open source alternative. The choice over which firmware I choose also determined the language that I was going to use.

### 4.1 Lego Mindstorms Limitations

Originally released in 1998, the Lego Mindstorms Robotic Invention Kit has been utilized heavily by schools to teach the basics of embedded systems and hobbyists. The core part of the system is the RCX brick which functions as the brain of the kit. While there are several different models currently available for purchase, I decided to stick with the Robotics Invention System 2.0. The main reason for this choice was my familiarity with the platform. In the past I have built numerous robots using this kit and programmed them using leJOS, the Java-based firmware replacement. Secondary reasons included the easy access I had to several kits and the wealth of resources for this particular version.

Choosing to implement the A\* algorithm on this platform is not without its difficulties. The biggest limitation of the platform is the amount of memory available. The RCX brick contains a paltry 32 kB of RAM and 16 kB of ROM. [11] This presents a problem because each firmware replacement will use a portion of this memory. The 16 kB of ROM is reserved for the RCX executive application which runs the hardware and launches the firmware stored in the RAM. [12] This makes it more challenging to program for the platform because out of the 32 kB of RAM that exists, around 6 kB is available for user programs.

This limitation of memory reared its ugly head to me. As of this paper I have been unsuccessful in implementing the algorithm on the Mindstorms platform due to the

---

<sup>11</sup> Simpson, <http://web.mit.edu/6.933/www/Fall2000/LegoMindstorms.pdf>

<sup>12</sup> Nielsson, <http://legos.sourceforge.net/docs/kerneldoc.pdf>

memory issues. My simplest version of the algorithm to date is almost 575kB when running under a non-optimized g++ compilation. A value that surprised me because the same code running in Visual Studio 2005 clocks in at around 392kB. While this is far too large to run on the robot, I still tried anyway. The robot compilation of the algorithm crashed the RCX brick every time I tried to run it. I attempted to optimize the code by compiling the program using the -Os tag in g++, but ran into multiple errors due to my un-optimizable code.

## 4.2 Choice of Firmware and Programming Language

Knowing the limitations that I was going to have on memory I decided to use a C/C++ based firmware replacement in an attempt to minimize the memory consumption. After making this decision my choice of firmware replacements was limited to two: NQC and brickOS. NQC (Not Quite C) is a C-like replacement that is limited in both the number of variables you can create and in the maximum number of subroutines. [13] These limitations did not sound good to me so I decided to use brickOS. Formerly known as legOS, brickOS is a firmware replacement that runs precompiled C/C++ applications. I decided to use brickOS because it used code that was more familiar to me and seemed to be the most complete of the two firmware replacements.

Up to this point my only experience with programming for Mindstorms was using Brickcode or LeJOS. I understood the limitations and strengths of both. A working version would be impossible with Brickcode, and I did not believe a small enough version was possible with LeJOS. I chose C++ because I wanted to streamline the code by using pointers and structs in the algorithm and because of its object oriented features. I also had previous experience using Visual C++ 2005. In retrospect, NQC might have been a better choice for creating a smaller compiled algorithm.

## 4.3 Other Limitations

There were also practical limits on my time to research algorithms, languages, firmware options, and development. It took me longer than first anticipated to fully understand the A\* algorithm well enough to create my own version, even though I had multiple versions of the basic algorithm and multiple implementations to use as references. A majority of the time I spent working on the algorithm was spent deciphering code. I struggled with understanding the exact process the A\* algorithm takes until months into my study.

# 5 A\* Search Algorithm Implementation

As of the writing of this paper, my implementation of the algorithm is currently not small enough to be implemented on the Mindstorms platform. I am still pursuing several

---

<sup>13</sup> Baum, [http://bricxcc.sourceforge.net/nqc/doc/NQC\\_Manual.pdf](http://bricxcc.sourceforge.net/nqc/doc/NQC_Manual.pdf)

strategies to try to fix this and be able to fit the program execution footprint onto the RCX brick. General algorithm details and ideas on further improvements to the code follow.

## 5.1 General Implementation

The general implementation of my algorithm follows the general A\* algorithm. (Figure 6) There are a few minor changes in certain areas, but these changes are mostly attempts to optimize the implementation. The changes I made include changes made to storage methods of both the lists and the grid and the necessary functions to implement the new storage methods.

```
create Open and Closed lists
create start node
  start node g = 0, h = goal estimate, f = g + h, parent = null
  push start node on open
while Open not empty
  pop best node from Open // best node has lowest f
  if best node is goal
    construct path and return success
  for each successor node of best node
    set successor node parent to best node
    calculate successor node g, h, and f
    if successor node in Open or Closed and f < parent f
      skip
    remove successor node from Closed and Open if exists
    push successor node on Open
  push best node on Closed
return failure // if no path
```

Figure 6: General A\* Algorithm [14][15]

## 5.2 List/Grid Storage Implementation

The majority of the implementation changes that I have made to the algorithm consist of storage changes. The example algorithms that I used as reference stored the node g, h, and f values as doubles, they also stored the node location using two integers, x and y. In my implementation I created a struct for the node that contained a single short (loc), a pointer to another node, and two floats (f and g).

---

<sup>14</sup> Stout, <http://www.gamasutra.com/features/19970801/pathfinding.htm>

<sup>15</sup> Jones, Page 28 – 30.

### 5.2.1 Node Location Implementation

The loc variable stored the x, y coordinate pair using bitwise operations. With aid from Dr. Jim Jones, I created get and set methods for placing and retrieving the information. The get method simply performed a shift operation to move the four location bits to the rightmost bits and used a modulus to retrieve the value. (Figure 8) The set method in contrast performed a Boolean AND on the loc variable with specific hex codes to preserve the original bit values with the exception of the four to be changed. After the AND I performed a Boolean OR on the loc variable and the value to be written shifted the appropriate number of bits.

```
short get(int i, short node)
{
    return (node >> ((3-i) << 2)) % 16;
}
```

Figure 7: A\* Node Location Get Method

```
short set(int i, int x, short node)
{
    switch(i)
    {
        case 0:
            return (node&0x0FFF) | (x<<12);
        case 1:
            return (node&0xF0FF) | (x<<8);
        case 2:
            return (node&0xFF0F) | (x<<4);
        case 3:
            return (node&0xFFF0) | (x);
        default:
            return 0;
    }
}
```

Figure 8: A\* Node Location Set Method

When I implemented the get and set methods I ran across an error that I had not anticipated. Because I was returning the values from get as shorts the bits for the value I was getting were returned as a signed short. So if the value was stored in the four leftmost bits and the first bit was a 1, I returned a negative value. When dealing with a grid that has no negative locations, this is a problem. My temporary solution was to store the node x location followed by the node y location on the rightmost eight bits.

One issue that turned out to be a non-issue was concern over whether the difference between big-endian and little-endian would influence the bit operations. My personal computer is a Dell Laptop using an Intel processor which is a little-endian machine while the RCX brick is a big-endian machine. To test whether this would influence the results

Dr. Jones devised a simple application to test the effects the endianness of the machines would have. (Figure 9)

```

short temp = 0;
temp = (temp&0xFFF0)|(1);
if (temp == 1)
    cputs("hello"); // or printf("hello")
else
    cputs("world"); // or printf("world")
sleep(2);
cls();

union{char x[2];short i;} abc;
abc.x[0] = 0;
abc.x[1] = 1;
if (abc.i == 1)
    cputs("hello"); // or printf("hello")
else
    cputs("world"); // or printf("world")
sleep(2);
cls();

```

Figure 9: “Endianness” Test Code

The test code consisted of two simple tests. The first one tested the method that was to be implemented in the algorithm. It simply stored the value 0001 in the rightmost four bits of the variable temp. The code then tested the value of temp and output ‘hello’ if the value was 1 and ‘world’ if it was not. The second part of the code statically assigned the values into the char array which disallowed the compiler from swapping the bit pairs on my laptop. When this program was run on my machine it output ‘helloworld’ but the same program ran on the robot output ‘hellohello’. This showed me that the compiler in Visual C++ was swapping the bits on the first test but not on the second. Seeing this was a relief because it meant I did not have to write extra code to deal with the endianness of the machine that was running the code.

### 5.2.2 Grid Storage Implementation

In my continued efforts to shrink this algorithm to fit within the confines of the RCX brick I plan to convert my existing char grid to a less memory intensive integer array. The current grid stores each location as a char, and is of size [Y\_MAX][X\_MAX]. This solution is wasteful in memory because I can only store the information for a particular coordinate location. My plan is to implement an array of integers where each integer stores the grid state for five coordinate pair locations. (Figure 10) This can be accomplished by using the same bitwise operations as above with minor modifications to accommodate the larger variable memory.

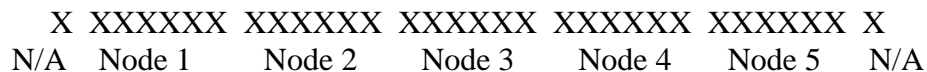


Figure 10: General Outline for Grid Node Storage

Each of the Node locations will contain six bits of information. The first two bits will state whether the node is goal or start respectively. The last four bits will state whether the north, south, east, and west walls are open (0) or closed (1). The outer bits on either side are unused and will be wasted space.

### **5.2.3 List Storage Implementation**

This implementation will free up some memory, but the largest offender in terms of memory consumption is the two lists. The general algorithm calls for two separate lists to contain the nodes. On the surface this doesn't sound too bad, but after considering that each list is fully initialized to the number of possible nodes it adds up quickly. My current implementation does not fix this problem, but I have a couple possible solutions in mind. The first one is to rewrite the storage methods to enable the removal of nodes that are clearly not ideal. The second idea is to store which list the node is a member of inside of the node struct using two booleans, one for open and one for closed.

## **6 Ongoing Work**

Based upon the work I have done so far, it is looking unlikely that I will get a complete working version of the algorithm on an RCX brick. With that in mind there are several modifications that could be made to allow some version of the algorithm to operate on the robot. These changes might become purely academic if I am able to successfully implement the changes outlined above.

My current implementation follows the program flow of the general algorithm, but this might change. If I am unable to shrink the memory footprint enough using the above modifications, it might become necessary to cripple the algorithm. This is obviously not an ideal solution, but if it is the only way to fit the algorithm onto the robot I will do it. A better solution would be to use an outside object to act as memory for the robot. This could range from another RCX brick to connecting my laptop and using it as the storage location for the data. This would free up the space on the robot to be used for the local temporary variables and the functions.

## **7 Observations and Conclusions**

While working on this project there were several observations that I made. The first observation is that it is important to plan out the algorithm that will be used to ensure that it is optimal for the end platform. The second observation is that the process of creating and optimizing code is not a simple matter of using a command line argument. This was brought home when trying to use the optimization argument (-Os) on g++, I was told by the compiler that I had un-optimizable code which is something I did not know was an issue. A third observation is that the libraries used by cross compilers for embedded

systems (i.e., RCX in this case) are minimal. For instance, the brickOS compiler does not have the absolute value function. I defined my own as an inline #define which turned out to be better for reduced size of executable. A final observation was that the process of using alternative data structures to attempt to optimize code is not without dangers. Using the bitwise operations to optimize data storage meant extra functions to keep things conceptually simple. That, however, cost more memory.

The amount of work required to understand a complex algorithm and optimize it for space considerations is significant. While it may not be impossible to achieve for the RCX brick, the limitations of programming knowledge, time, and resource availability can all conspire to hamper timely progress. The effort to reconcile the issue of implementing this complex algorithm under limited resources shall continue and more will be learned about what is possible.

## References

- [1] Jones, M. T. (2005). *AI Application Programming* (2nd ed.). Hingham, MA: Charles River Media, Inc.
- [2] Baum, D. (2003). *Definitive Guide to Lego Mindstorms* (2nd ed.). New York, NY: Apress.
- [3] Patel, A. (n.d.). *Amit's A\* Pages*. Retrieved November 2007, from <http://theory.stanford.edu/~amitp/GameProgramming/>
- [4] Stout, B. (1997, August). *Smart Moves: Intelligent Pathfinding*. Retrieved November 2007, from <http://www.gamasutra.com/features/19970801/pathfinding.htm>
- [5] Heyes-Jones, J. (2006, September 18). *A\* Algorithm Tutorial*. Retrieved December 2007, from <http://www.geocities.com/jheyesjones/astar.html>
- [6] Villa, L. (2000, October 22). *LegOS HOWTO*. Retrieved January 2008, from <http://legos.sourceforge.net/HOWTO/>
- [7] *brickOS C++ Documentation* (2004, February 16). Retrieved January 2008, from <http://brickos.sourceforge.net/docs/APIs/html-c++/>
- [8] Chen, D. (2004, July 28). *BrickOS Command Reference v2.0-0.2.6.10*. Retrieved January 2008, from <http://brickos.sourceforge.net/docs/CommandRef.html>
- [9] Nielsson, S. (200, September 27). *Introduction to the legOS Kernel*. Retrieved February 2008, from <http://legos.sourceforge.net/docs/kerneldoc.pdf>
- [10] Simpson, J. (n.d.). *A Native Transterpreter for the LEGO MindStorms RCX*. Retrieved February 2008, from <http://web.mit.edu/6.933/www/Fall2000/LegoMindstorms.pdf>
- [11] Baum, D. (n.d.). *NQC User Manual*. Retrieved January 6, 2008, from [http://bricxcc.sourceforge.net/nqc/doc/NQC\\_Manual.pdf](http://bricxcc.sourceforge.net/nqc/doc/NQC_Manual.pdf)
- [12] Baum, D., Hansen, J. (n.d.). *NQC Programmer's Guide*. Retrieved January 6, 2008, from [http://bricxcc.sourceforge.net/nqc/doc/NQC\\_Guide.pdf](http://bricxcc.sourceforge.net/nqc/doc/NQC_Guide.pdf)
- [13] Overmars, M. (2002, February 14). *Programming Lego Robots using NQC*. Retrieved January 6, 2008, from [http://bricxcc.sourceforge.net/nqc/doc/NQC\\_Tutorial.pdf](http://bricxcc.sourceforge.net/nqc/doc/NQC_Tutorial.pdf)



# Customizing MediaWiki for Project-based Courses

Olaf Hall-Holt

Department of Mathematics, Statistics, and Computer Science

St. Olaf College

Northfield, MN 55057

olaf@stolaf.edu

## Abstract

The increasing popularity of Wikipedia ([en.wikipedia.org](http://en.wikipedia.org)) suggests that the Wikipedia interface for collaborative document creation may become a de facto standard. Project-based courses in all disciplines frequently require collaborative document development, but the default configuration of the MediaWiki server software used by Wikipedia does not provide all the functionality required for an academic setting.

We describe several approaches to adding the necessary functionality into the MediaWiki server, as well as our experience using modified wikis in both technical and non-technical courses. Some of our specialized extensions to MediaWiki servers have exceeded our expectations, both in terms of convenience for instructors and student reactions.

As wikis become more popular for use in colleges and universities, they may grow to provide functionality currently associated with social networking sites. What would a social-networking wiki look like? What kinds of protocols would be important for sharing information between campuses?

# 1 Introduction

The increasing popularity of Wikipedia ([en.wikipedia.org](http://en.wikipedia.org)) suggests that the Wikipedia interface for collaborative document creation may become a de facto standard. Project-based courses in all disciplines frequently require collaborative document development, but the default configuration of the MediaWiki server software used by Wikipedia does not provide all the functionality required for an academic setting.

Project-based courses typically require one or more of the following functions:

- restricted access mechanisms, so that information intended only for the professor or a restricted subset of the class can be separated from public information about the course.
- assessment tools, which streamline the process of providing formative and summative feedback to students (and TAs).
- project management support for keeping team activities synchronized and on schedule.
- protocols and interfaces for integrating other types of applications.
- convenient and specialized communication pathways to facilitate group interaction.

One danger in integrating these types of functionality into the MediaWiki server is that adding all these functions may compromise the simplicity and ease-of-use that make the wiki attractive in the first place. For example, the mechanism for access restrictions must be transparent to all the students in the course, for if they do not know how to use it, the access restrictions will not be effective. As another example, if the interface to some homework-related function is not clear, new users may quickly become overwhelmed. The danger of introducing new complexities is even greater when the platform is used in a non-technical course or community.

In this paper, we describe several approaches to adding the above types of functionality into the MediaWiki server, as well as our experience using modified wikis in both technical and non-technical courses. Some of our specialized extensions to MediaWiki servers have exceeded our expectations, both in terms of convenience for instructors and student reactions.

As wikis become more popular for use in colleges and universities, they may grow to provide functionality currently associated with social networking sites. We have begun to wonder what a social-networking wiki would look like. Also, what kinds of protocols would be important for sharing information between campuses? The latter part of this paper begins to explore these questions as well.

## 2 Restricted access mechanisms

Information on a wiki is stored in pages in much the same way that information in a file system is stored in files. A page can be understood to contain a sequence of bytes, just

like a file, and so can be used for storing a wide variety of types of information. Pages on a wiki typically include some amount of meta-data that is separated from the page source proper. Perhaps the most important difference when comparing with a file system is the lack of directory structure: links between pages are assumed to be the dominant mode of local navigation, while search is used for long hops. Directory structure can be mimicked by choosing page names that look like pathnames, but there is no built-in navigation mechanism that relies on such naming schemes.

In this setting, the basic access restrictions control access to individual pages. We would like these restrictions to be visible and easily used by novice authors, yet not interfere with the page source itself. We also would like to encourage open-ness on the wiki, in keeping with the general tone of many MediaWiki projects. One option is to store these restrictions in the page meta-data. Another option is to write these restrictions directly into the page name.

In comparing these two alternatives, the main drawback to using meta-data may be that it requires a change to the editing interface for a page. Choosing the name of a page is a standard operation that is implicitly supported. Also, appropriate encoding of access restrictions into page names allows pages with less restrictions to be shorter and easier to type, thus encouraging open-ness. This is the route we have chosen.

The most basic access restrictions on pages are read/write privileges. To encode restrictions on who can edit a page, we prefix the page name as in the two examples below:

- User:Helga/My first page
- Group:CS121TeamBlue/Animation/Stills

These page names are intended to look something like files in folders, with the first page name indicating that Helga alone has write privileges, while the second everyone in CS121TeamBlue has write privileges. Note that the second / in the latter example is purely cosmetic.

To encode read restrictions, we then add a Restrict: clause anywhere else in the page name:

- User:Helga/Restrict:CS121TeamBlue/Project journal
- Group:CS121TeamBlue/Animation/Restrict:CS121A/Check this out

In the first example, Helga has created a page that can be read only by her team members in CS121TeamBlue. In the second, the team has a page for sharing ideas with anyone else in the same section of the class. Note that the placement of the Restrict: clause does not have any significance (it could be in the middle or the end, it just doesn't matter).

Another feature of MediaWiki engines is that each page comes with a 'talk' page for discussion around the page content. Access control can be simply extended as follows: if a page is readable by a given user, then the associated talk page is both readable and writable by that user.

We have found that this interface for access permissions is understandable and usable by technical and non-technical students, but not instantly. Students may be unfamiliar with the overall concept of differentiated read/write access restrictions, to say nothing of the details of this interface. To begin, it seems helpful to have authors use pages with predetermined

names (and thus, predetermined access restrictions). Then, when they are accustomed to the names and have run into a situation where their access was blocked, they are prepared to better understand and use these mechanisms.

The implementation of the above strategy for access permissions is relatively straightforward in MediaWiki, as long as you don't expect a totally secure result. That is, it requires about a page of PHP code, outside the main body of the server (in the 'extensions' directory) if you rely on the 'userCan' hook. This hook is advertised to be inadequate to prevent really clever users from getting access (particularly read access) to pages that they should not see, but the required level of cleverness to squirm around 'userCan' seems to be going up with time.

### **3 Assessment tools**

A major assessment activity for a wiki-supported course may often be the grading of homework. Homework assignments are easily posted on a wiki, and we have found it useful to make links for the students to use from the homework assignment page directly to the pages where the students are to do their work, so that there is no confusion about page names. We have arranged pre-designated names for homework pages either by asking a TA to create a link for each student, or by having a little javascript support on the page to customize a generic link each time a student visits the page.

We have found it helpful to ask students to spread each homework assignment across multiple pages corresponding to the smallest meaningful unit of each assignment. That is, on a homework assignment, we create a separate link for each exercise in each part of the assignment. When this division into small portions is made, it becomes easier to compare student responses, by aggregating a large number of responses into a grader's summary page.

The construction of a summary page is largely a matter of choosing how to insert one page's content into another page, and to do so in a way that is convenient for TAs. There are many possible approaches for how to implement this type of functionality, including approaches that generalize to other uses (see the section below on application interfaces).

Other assessment tools are certainly possible and potentially quite valuable. On these wikis, all user activities are recorded quasi-permanently, including all revisions of a page, all contributions to talk pages, and more. This information is stored in a relational database with an open structure for developers.

### **4 Project management support**

Our student projects include a planning phase, an implementation phase, and a presentation phase. For the planning phase, a team often finds it convenient to write their plan as a page on the wiki, which can then be viewed and updated by team members and TAs at any time as a coordination mechanism.

The implementation phase often involves team members that work in parallel. To keep everyone informed about the progress of different team members or subgroups, we some-

times ask everyone to keep a journal and a time log, again spread out over many pages with predefined names. Every team member has access to the journals and time logs of the other team members. As the team members write entries in their journals and logs, a 'page trail' accumulates that both facilitates communication and informs assessment activities.

In order to summarize the activities of all the people in a class, we have created a page to display a bar graph of everyone's time investment in the project to date, based on the content of their time logs and journals. The bar graph is clickable, so that different parts of the graph link to each time log and journal. Other summary information (such as the number of words in a given journal entry) pops up as the mouse moves over different parts of the graph.

From a pedagogical standpoint, this type of coarse overall data regarding project involvement has at least two benefits: first, it allows students to develop a shared understanding of a 'reasonable' time commitment for a project. Second, it provides an accountability mechanism for keeping up to date with journals and time logs, and thus supports the communication that can happen through those channels.

## **5 Application interfaces**

MediaWiki is gradually evolving more modularized approaches to integrating server- and client-side applications. On the client side, it is getting easier to edit the content of pages with applications other than a browser, and on the server side more hooks are becoming available by which a PHP wrapper can invoke server side executables with information from the wiki's database.

We have been particularly interested in using pages themselves as executables, rather than their default use as simple HTML documents. In order to do so, we have made it possible for at least some users to type code in a scripting language (Perl, Scheme) in a page, and then have the server execute that code when the page is saved. We also have some pages that, when any user accesses the page, the code on that page is executed.

We are also evolving scripting hooks by which these executable pages can access the wiki's database. The most important example of these scripting hooks is the ability for a script to get access to the content of other specified wiki pages. It is in this way that we currently implement the summary pages for TAs, the bar graph for projects, and various other tools. These scripts also have access to some 'real' executables on the server itself, and thus can invoke non-trivial functionality that may not exist on the client side. From the point of view of experimenting with a wiki, this approach has provided nice benefits in allowing lightweight implementations of new ideas. On the other hand, this approach has also required the development of a more thorough security framework than would otherwise be needed.

## **6 Communication pathways**

The vast majority of our students seem to use instant messaging, voicemail and e-mail, synchronous telephone calls, face-to-face meetings, dynamic web pages, and more in the

course of a typical day. It seems as though having specialized communication channels is helpful, especially if the channel is convenient at a given moment of time. The explosive growth of social networking sites suggests that having specialized database queries packaged for you can be interesting and useful to a large number of people on a collaborative content platform.

The examples of specialized communication we have used, such as the TA summary pages or the project time bar graph, have been hand-crafted from a general script language. However, it should be possible to create pages for users that allow a menu of query options, as well as 'push' style polling of the database to keep informed of the actions of other users. Mash-ups might also be a useful abstraction in such a wiki environment.

Someday, there may be wikis on many college and university campuses that have evolved to meet the needs of students and faculty. Would it ever be possible for these wikis to provide cross-institutional links, using shared query and mash-up protocols?

## 7 Conclusion

The response to our forays into customizing wikis has been largely positive. In computer science courses, including both introductory courses and more advanced project courses, the wiki has become a natural part of the work of our students, even to the point that the students have been building their own wikis and relying on the existing wiki for extracurricular projects. The main drawbacks for students so far have been the times that modifications to a wiki has made the wiki or the associated web server unstable. Happily, this problem hasn't occurred for more than a year.

Using the wiki as the basis for the assignments in a humanities class was also favorably received. Student essays were posted in appropriately restricted access areas of a wiki, and talk pages were used to discuss these assignments within sections of the class. The humanities students and TAs appeared to take to the wiki relatively easily.

From a faculty standpoint, the work on modifying these wikis has involved a significant time commitment. The approaches suggested in this paper did not develop all at once, and some of them have required considerable tinkering. However, as mentioned in the section on application interfaces, the MediaWiki server is getting easier to modify, and some of the design ideas represented in this paper enable a cleaner approach to many issues than we originally chose. Furthermore, as more faculty become interested in customized wikis, the code needed for these customizations can easily be shared.

If these approach become widely used in colleges and universities, students and faculty would benefit from a shared strategy for managing these communication resources.

# **INTEGRATION OF CODELAB INTO PROGRAMMING COURSES**

**Mark S. Hall**  
**Assistant Professor of Computer Science**  
**Computer Science, Engineering, Physics & Astronomy (CSEPA)**  
**University of Wisconsin – Marathon County (UWMC)**  
**University of Wisconsin - Colleges**  
**Wausau, WI 54401**  
**mark.hall@uwc.edu**

## **Abstract**

CodeLab is a web-based interactive programming exercise system for introductory programming classes in Java which is used for the first two CS core classes at the UW-Colleges. CodeLab has over 300 short exercises with each exercise focusing on a particular programming concept. Student submissions are automatically judged for the correctness of the solution, and offers hints when the submission is incorrect. If incorrect, the student modifies their solution, and submits it for verification.

During fall 2007, I integrated CodeLab into my CS1 course and my CS2 course. This paper will highlight features of CodeLab and my experiences using this tool to attempt to improve student's success in developing successfully algorithms. The paper will focus on the potential benefits to the student and the benefits to the instructor as well. In addition, the paper will discuss the changes to using CodeLab for the spring 2008 semester.

# **1 Introduction – Academic Environment**

Many journal articles and papers have documented the problems that are facing the Computer Science discipline in the first two years of the curriculum. The core Computer Science courses usually are programming classes with student drop-out rates higher than the discipline would like to see. Many new pedagogically ideas are constantly being offered as aids to combat this problem.

## **1.1 Background**

The thirteen 2-year University of Wisconsin (UW) Colleges serve students over the entire state of Wisconsin [2]. The Computer Science (CS) program in the UW Colleges is designed to be easily transferable to the 4-year institutions within the UW System. The nationwide decline in CS enrollments reported in the 2003-2004 Taulbee Survey [3] has affected the UW System, and has been particularly hard on the 2-year UW – Colleges (UWC), which are relatively small colleges, with a combined enrollment of approximately 12,000 students. In addition, the majority of our students are in the bottom 60% of their high school rank. The majority of our students enroll to prove that they can achieve at the University level in order to be accepted at a 4-year University.

## **1.2 Enrollment Issues**

A trend of declining enrollment in Computer Science (CS) is being felt at the 2-year University of Wisconsin (UW) Colleges, as is the case nationally. UWMC is the third largest two-year campus in the UW Colleges [4]. It is safe to say that those are NOT the numbers that were active in the classroom at the end of the semester. The loss of these potential students for the core Computer Science courses are very troubling. Many research articles have documented that students struggle with learning the syntax of programming languages. Something needed to be done to recruit and then retain these students.

## **1.3 Problems**

A major problem is that the students enrolling into the UW–Colleges are that many students do not have the necessary math requirements to enroll in any CS course. They are forced to take remedial math courses prior to actually taking their first CS course. It is easy to understand why these students struggle with their problem solving skills as they failed to take the necessary college prep courses while in high school. To succeed, these students need to be exposed to as many opportunities to solve problems, either in homework assignments or programming assignments. However, the disadvantage is that this requires intense effort on any instructor to grade these assignments. Students take graded assignments more seriously than if they are not graded.



## 1.4 Designing Algorithms and Critical Thinking Skills

As I looked at the guidelines established by the UW Colleges for teaching the general concepts for this course, the emphasis that course coverage for designing algorithms should take at least a minimum of six hours. And the reality is that every programming assignment typically forces students to design a totally different algorithm. In order to be successful in this course, for the students and the professor, somehow, someday, the critical thinking skills of students have to be improved. Improvements in critical thinking along with the understanding of the basic control structures will allow students to be successful writing simple programs using any language. It is obvious that more time on task can be very helpful for developing these critical thinking skills. However, more assignments that need to be graded so feedback can be given to the student can overwhelm any instructor.

## 2 Introduction -- Codelab

CodeLab [5] is the commercial version of an academic NSF project[6], WebToTeach [7, 8]. The pedagogy behind CodeLab mimics techniques used widely in other subjects, such as mathematics and foreign language study. The main idea of the pedagogy is to provide large numbers of self-paced, highly interactive exercises that focus on key ideas of programming. These exercises are intended to augment, rather than to replace, the traditional "whole program" assignments in the first year of undergraduate study. CodeLab is a web-based tool that enables faculty to assign exercises to students and monitor student progress.

For the student, CodeLab provides experience with fundamental elements of syntax, semantics, and basic usage of the programming language. The tool provides immediate feedback on correctness and often offers suggestions for fixing errors. Students can proceed at their own pace, subject to deadlines imposed at the instructor's discretion.

CodeLab has over 300 short exercises with each exercise focusing on a particular programming concept. The exercise is presented to the student and the student types in their solution to the exercise. The system immediately judges the correctness of the solution, and even offers hints when the submission is incorrect. If incorrect, the student modifies their solution, and submits it for verification. "Through this process, the student gains mastery over the semantics, syntax and common usage of the language elements." [9]

### 2.1 History

Turing's Craft was founded in 1999 by David Arnow and Gerald Weiss. They are both professors of Computer and Information Science at the City University of New York. To aid in teaching Computer Science concepts, "Arnow developed the WebToTeach system to address the limited opportunities for computer science students to practice the concepts taught in the classroom." [9] The early academic versions of WebToTeach were used by

thousands of students in several colleges and high schools since 1990. A National Science Foundation grant was awarded to them to further develop the technology in hope that this system could be applied to help reducing the attrition rate of beginning CS students.

“Responding to broad user interest and with the NSF's encouragement, the two started Turing's Craft in early 1999 to commercialize the WebToTeach technology and thereby make it broadly available. In June 2000, Turing's Craft received a technology commercialization grant from the Empire State Development fund in recognition of the outstanding potential of the technology. In the spring of 2002, Turing's Craft released the commercial version of WebToTeach, CodeLab.” [9]

## **2.2 Benefits of using CodeLab**

### **2.2.1 Benefits to the Student**

The benefits to the students posted on the web site are: [9]

1. Higher Test Scores
2. No Brick Wall
3. More Efficient Studying
4. Better Grades on Projects

#### **2.2.1.1 Higher test scores**

If a student receives immediate feedback, then that should help the retention rate of the material. This, in theory, should help the student score higher on exams with the increase in the retention of the concepts.

#### **2.2.1.2 No brick wall**

Ever Computer Science student has, at one time or another, and sometimes very frequently, been completely stuck on a homework problem. The “brick wall” syndrome is an experience in itself. In CodeLab, since every exercise comes with feedback, hints and other explanatory material, the brick wall syndrome should be reduced, and sometimes even eliminated.

#### **2.2.1.3 More efficient studying**

If students knew what concepts that they were struggling with, then they could spend more time doing assignments on that concept. Students would be getting more bang for their buck with their study time. Since CodeLab gives feedback within seconds of the solution submission, students can quickly understand if they know this concept.

#### **2.2.1.4 Better grades on projects**

The Codelab assignments help students prepare for the learning the concepts that are necessary to solve the larger programming projects. These projects should require less time since students will spend less time on the basics and more time on creating a solution that meets the specifications of the project.

### **2.2.2 Benefits to the Instructor**

The benefits to the instructor, as posted on the web site are: [9]

1. Take the high road!
2. Grading relief / Automatic class rostering
3. Frees up office hours
4. Decrease attrition

#### **2.2.2.1 Take the high road!**

CodeLab aids the students in learning the syntax, semantics and the basic usage of the programming language. Class time can now be spent on problem solving strategies, design and analysis, abstraction, algorithms, and style. This emphasis can aid in the development of the critical thinking skills of the students.

#### **2.2.2.2 Grading relief / Automatic class rostering**

Many articles have been written on the frustrations faculty feel with the overwhelming tasks of grading homework and programming assignments. CodeLab automatically checks student work for correctness and tracks student performance by maintaining their submission records.

#### **2.2.2.3 Frees up office hours**

With its built-in feedback mechanism and aids, students do not have to seek out the instructor to overcome the brick walls that they encounter. Now, when students choose to speak to their instructor during office hours, their problems will be at a higher level in which the instructor can provide useful knowledge. An instructor provided the following testimonial: *"I am getting more questions about concepts, software engineering and problem-solving and fewer questions about basics."* [9]

#### **2.2.2.4 Decrease attrition**

This was one of the primary goals when CodeLab was developed. If students can be helped as early as possible in the process, then more students should be successful in the classroom which would reduce the number of students dropping the course.

### **2.2.3 Benefits to the CS Department**

The benefits to the CS Department, as posted on the web site are: [9]

1. Retention
2. Language Switching
3. Confidence of Achievement Level
4. Automated Placement Exam

#### **2.2.3.1 Retention**

One of the primary goals of the NSF grant was to reduce attrition. Reduced attrition aids in the retention of CS students for future classes. “This is clearly having benefits. For example, one instructor reported that six weeks into the term, instead of the usual 30 student withdrawals (out of 150), only 2 had dropped out.” [9]

#### **2.2.3.2 Language Switching**

The UW-Colleges changed their CS curriculum to offer the first two CS core classes in Java and to offer CS3 (Algorithms and Data Structures) in C++ so students get a chance to experience multiple programming languages. This curriculum change was due to the desire to make the CS credits easier to transfer to the public four-year universities in Wisconsin. For the 2<sup>nd</sup> language, CodeLab makes it much easier to switch languages without having to devote class time to teaching the fundamentals. Now instructors can spend less time on teaching the language and more on teaching the topics that need to be covered for the course.

#### **2.2.3.3 Confidence of Achievement Level**

“CodeLab certifies that the student has correctly written code that solves a stated problem.” [9] Both the students and the instructor should have increased confidence that students in the introductory classes have mastered the introductory topics and concepts that are necessary to build the foundation for future courses.

#### **2.2.3.4 Automated Placement Exam**

Students transferring into the university or High School AP students can use CodeLab to prove their mastery of the necessary concepts so that they can be placed into the

appropriate class level. CodeLab can also be used as the automated placement exam. Students coming into the program can be accurately assessed with relation to their mastery. CodeLab will report the success rate of any set of assignments that has been set up to act as the placement exam.

## **2.3 CodeLab Exercises**

### **2.3.1 Short and Focused**

CodeLab exercises are short, focused on a particular topic and automatically evaluated. The exercises range from "one-liners" like variable declarations, arithmetic and boolean expressions, assignment statements to more complex problems involving loops, functions or methods or even small class definitions.

### **2.3.2 Graduated**

The first exercises are really simple so that even the most challenged student can be successful right away. The exercises build upon each other so as the student progresses within each concept, the exercises become more complicated.

### **2.3.3 Immediate Feedback**

Many of the exercises are similar to exercises found at the back of a typical textbook. The difference is that CodeLab gives the student immediate feedback on their answer with hints if their answer was not complete. Each successful submission is recorded in the roster for the instructor.

### **2.3.4 Topic Coverage**

The exercises start with the imperative programming core of Java and then go on to address procedural and object-oriented programming. These exercises are targeted at a typical CS1 syllabus.

### **2.3.5 Customized**

Instructors can add assignments to complement existing assignments or to add additional coverage.

## **3 Introducing CodeLab into Programming Courses**

During SIGCSE 2007, I was introduced to CodeLab when I met David Arnow at the Exhibit Hall. The demonstration of CodeLab sounded promising as a new educational tool. Once SIGCSE was over, I completely forgot about it until David called me early in the fall 2008 semester. During the phone call, I agreed to use CodeLab in two courses for that semester, a CS1 course in Java and a CS2 course, also in Java.

CodeLab has already been woven into many textbooks including the Morelli textbook which I currently use for my CS1 course. The existing CodeLab assignments are linked to the appropriate chapters in the textbook. So the integration into the course was very easy. The students were given the instructions for establishing their user ids and passwords for gaining entrance into the web site for the course. And off they went with enthusiasm...

### 3.1 CodeLab Effectiveness

As I wrote this paper, I searched for published papers on CodeLab reporting on the effectiveness of this tool. It was surprising how little evidence exists.

#### 3.1.1 Existing Anecdotal Evidence

Anecdotal evidence abounds as there are many testimonials about using CodeLab for introductory programming courses. The list of universities (Figure 1) that are using CodeLab, by itself, could be used to make a statement about the effectiveness of CodeLab.

Adelphi, Alabama, Alberta, Belmont Abbey, Bob Jones, British Columbia CSU - Long Beach, CUNY – Baruch, CUNY – Brooklyn, Cabrillo Col., Central Piedmont CC, Clemson, Col. of the Ozarks, Corban, Covenant, Dallas Baptist, De Anza, DePaul, Denison, Ellis, Fairleigh Dickinson, Florida Atlantic, Georgia Col. & State, Greenhills School, Illinois – Chicago, Illinois - Urbana/Champaign, Johns Hopkins, Johnson County CC, Kentucky, Kenyon, Loyola University Chicago, Merrimack Col. Michigan - Ann Arbor, Michigan Tech, Millikin, NE Miss CC, NW State U of Louisiana, Nebraska – Lincoln, Northern Colorado, Oklahoma, Ontario Inst. of Tech., SUNY - Stony Brook, South Florida, Texas A&M – Commerce, Toronto, Trinity Col., UC – Riverside, UT - San Antonio, Wisconsin – Madison, Xavier
----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Figure 1: Universities using CodeLab

Many instructor testimonials rate the product very highly:

*"I have to admit I liked the interface and the load that it took off of me. Overall I was impressed with the product and will use it in my intro class again."* [9]-- Ken Whitener, Professor, Iowa Lakes Community College

*"I now highly recommend the use of CodeLab to all other instructors and students. It finally provides both the programming practice and evaluation support that I have been seeking for an introductory course." [9]-- Michael Schultz, Instructor, University of Wisconsin*

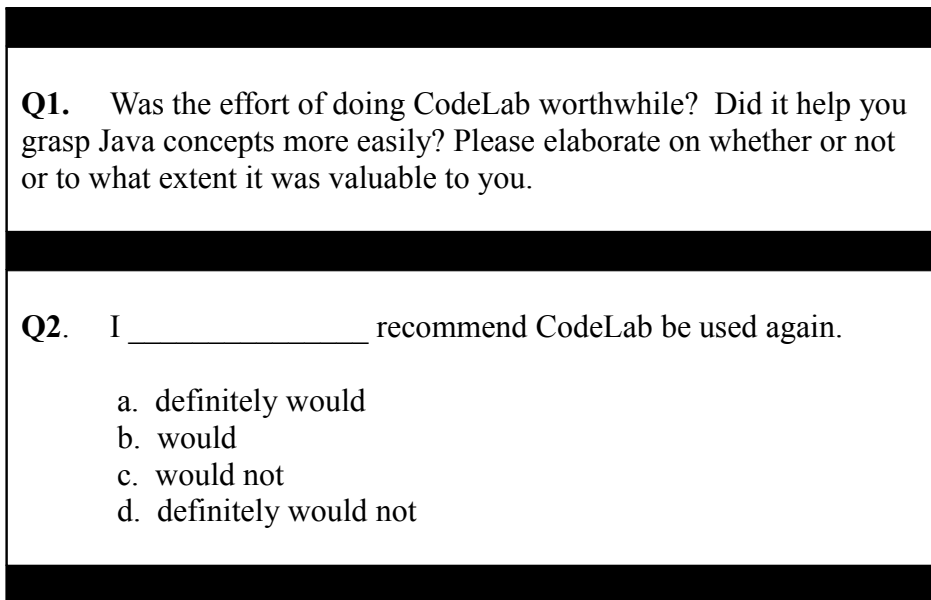
*"I really love your site. It has helped me cover more material with better comprehension than I've ever had before!" [9]-- Marcus Darden, Professor, Olivet College*

### 3.1.2 Empirical Evidence

I will admit I did not spend many hours searching but several quick searches did not reveal any published articles with empirical evidence.

### 3.1.3 My Anecdotal Evidence - Survey

With my small class size, I cannot overwhelm you with any empirical evidence with the effectiveness of using CodeLab in my CS courses. Near the end of the semester, I had my CS1 students take the following two question survey (Figure 2).



**Q1.** Was the effort of doing CodeLab worthwhile? Did it help you grasp Java concepts more easily? Please elaborate on whether or not or to what extent it was valuable to you.

**Q2.** I \_\_\_\_\_ recommend CodeLab be used again.

- a. definitely would
- b. would
- c. would not
- d. definitely would not

Figure 2: CodeLab Survey

The one student who did not recommend using CodeLab again reported it had been difficult to correlate the ideas from the textbook with those presented by CodeLab. This student said "CodeLab was worthwhile when I understood the material placed before me. When I didn't understand the material very well it helped me very little and in fact added greatly to my level of frustration."

The pedagogical goals that are designed into CodeLab were reflected in many of the comments. One student explained that it "helped me apply better what was taught in class and what I read in the chapters". Another said "It gave some good practice on specific pieces of Java programming, with immediate feedback on what went wrong". Yet another appreciated that CodeLab "helped me to have more contact with the vocabulary of Java". "The best part is that when I am truly stuck I can consult with my professor and receive more detailed feedback." was a comment that made an impression with me.

Numerous critiques of the CodeLab tool were offered by the students as well as well as recommendations for how to improve the tool. One student complained that "it had no style hints or help on elegance and the instructor harps on that all day long". I guess that student's do pay attention if they hear the same theme over and over as I do emphasize good programming guidelines in the CS 1 course.

Even in CS, the English language comes into play as several students complained that the wording on the questions or hints were not helpful at all. One student remarked "sometimes the questions were kind of worded funny"; another stated that "some parts were too easy and repetitious, and in other parts the wording confused me so much I didn't know what to put". CodeLab does not eliminate the use of a textbook. I believe that these comments could be reduced if students would spend time reading the textbook along with attending and paying attention during class would provide the students with the needed background to understand the vocabulary that is used in CodeLab.

Another concern related to the feedback, with one student writing "More help and better instructions would have been better". A student took this concern further with the statement "that CodeLab does not teach you how to accomplish the task".

### **3.1.4 Implementation Errors**

In my rush to implement CodeLab in my CS1 course, I failed to include in my syllabus to mention how the CodeLab exercises would be used in determining the overall grades. So I allowed students to use CodeLab as a review mechanism for the exams and to aid them if they were having any problems with the concepts. Students did use CodeLab early in the semester but when the "more complex" programming assignments were assigned, usage of CodeLab screeched to a grinding halt. This was not a smart decision on my part. I wanted to believe students would use the tool because it would help them. But it is obvious that the use of CodeLab has to be included as parts of the grade or the students do not see the value of the tool.

In my CS2 course, I envisioned using CodeLab as a review mechanism for my students who had taken CS1 in the spring and had spent no time reviewing Java before showing up for the fall semester. This particular group of five students was overall, a very talented group of students. After using CodeLab for the first two weeks, they also stopped playing with CodeLab when they started receiving additional homework and major programming assignments for the CS2 course.



## 4 Discussion and Future CodeLab Plans

CodeLab has lots of potential to help aid students in introductory CS programming courses. Any introduction of a new educational tool has to be integrated into the entire course in order for the tool to be successful. For the spring 2008 semester, I am teaching two sections of CS1, a face-to-face section and another section composed of students at the smaller UW-Colleges campuses where small enrollment issues do not allow F2F section to be taught locally.

This second section is a distance education course where I never get to physically see my students and they do not get to see me. A version of Microsoft Live Meeting is used where students can see the contents of my PC screen projected on a local large-screen display. Audio is over a bridged telephone hook-up so we can hear each other so the class can be interactive.

Both sections are using CodeLab and students must complete 80% of all CodeLab assignments to receive the maximum homework points which constitute 10% of their grade. Since office hours for the DE students are only virtual, CodeLab has a chance to be very beneficial for them as they cannot physically come to my office. Hopefully, the brick walls are reduced for them and their frustration levels are also reduced since they are using CodeLab for the homework assignments.

## References

[1] Shackelford, R., and LeBlanc, R. Introducing Computer Science Fundamentals Before Programming. *Proceedings of FIE '97*, 285-289.

[2] UW System, Two-Year Campuses <http://www.wisconsin.edu/campuses/twoyear.htm>

[3] Zweben, S., 2003-2004 Taulbee Survey, *Computing Research News*, Vol. 17/No.3, May 2005. <http://www.cra.org/CRN/articles/may05/taulbee.html>

[4] UW – Colleges,  
<http://www.uwc.edu/>

[5] Turing's Craft – The Exercises,  
<http://www.turingscraft.com/exers.php>

[6] Arnow, D. & Weiss, G. An Asynchronous Learning Network Tool for Improving CS Education and Retention Rates, Proposal to the National Science Foundation, EHRDUE CCLI-EMD Program.

[7] WebToTeach: A Web-based Automated Program Checker, *Frontiers in Education* (FIE99), San Juan, Puerto Rico, November, 1999. (With Oleg Barshay).

[8] WebToTest: On-line Programming Examinations Using WebToTeach, ITiCSE 99, Cracow, Poland, June, 1999. (With Oleg Barshay).

[9] <http://turningscraft.com>

# The Capstone Experience: Learning to Manage Uncertainty and Ambiguity in a Project Management Environment

Shaun M. Lynch, Ph.D.  
Department of Mathematics and Computer Science  
University of Wisconsin-Superior  
Superior, WI 54880  
slynch@uwsuper.edu

## **Abstract**

A capstone class can provide a quintessential experience for students in their program of study and is a critical component of an information systems curriculum. This paper examines uncertainty and ambiguity in a project management environment. The article considers the experience of students enrolled in CIS 456 Project Management—the capstone class for the Computer Information Systems major at the University of Wisconsin-Superior—during Spring Semester 2006. Discussion includes a description of the project, an overview of the class content and activities, and a step-by-step description of the process students engaged in. Finally, project outcomes and recommendations made by the students are presented with relevant observations about the influence of uncertainty and ambiguity on their project management experience.

## Introduction

A capstone course provides a quintessential experience for students in their program of study and is a critical component of an information systems curriculum. A project-based capstone class using real-world projects can expose students to the complexity they will certainly face as information systems professionals. Participants have the opportunity to practice formal project management methodologies while learning to manage uncertainty and ambiguity. Designing a course that balances project dynamics with the use of structured practices proves to be a delicate process worthy of consideration by those who teach capstone classes.

As with any class, the learning experience is often impacted by unforeseen events and circumstances. This article attempts to capture these extraordinary conditions in attempt to look how these dynamics contribute to the uncertainty and ambiguity students must contend with when immersed in a formal project management setting. It extends the article, *The Capstone Experience: Integrating Curricular Outcomes with Real-World Practice* [1] and follows, *The Capstone Experience: Balancing Formal Project Management Methodologies with Student Creativity and Innovation* [2] in a series intended to examine and document the intangible aspects of a capstone experience.

The content of this paper has been divided into four parts. The first part provides the background of the project from which the student project originated. The second part provides an overview of the capstone class to include content, activities, and performance metrics. The third part presents a chronological record of the capstone experience with milestones and a description of relevant activities. Finally, the fourth part presents the outcomes of the project along with the recommendations students made followed by relevant observations about the influence of uncertainty and ambiguity on the experience.

## Project Background

The impetus for the 2006 capstone project originated from a consolidation project at the University of Wisconsin-Superior to merge the two information technology and service providers, namely Computing and Media Services (CMS) and Network and Programming Services (NPS). The goal of the consolidation project was to find and implement a permanent solution that promoted department strengths and continued contribution to the University's mission while being conscious of staffing and resource limitations. Three goals were established to include:

1. Improving executive decision making within the Information and Instructional Technologies (IIT) organization,
2. Minimizing and eliminating operational redundancies (or perception of) that existed, and
3. Facilitating long-term management of increasingly complex information technology infrastructure and services.

Build-up to the project began in the summer of 2005 with a review by an external consulting team to assess the effectiveness of the two departments. In many areas, the reviewers praised the contributions the units made toward the University's mission. However, the consulting team also identified areas that needed attention, in particular, the need to integrate the departments and consolidate the two directorships into a single executive position [3].

In late August, the Chancellor extended an invitation to the author to facilitate the consolidation project under the oversight of the Provost. Upon accepting the assignment the project commenced and was announced to the membership of both departments with the charge to 1) integrate the two departments into a single cohesive unit with a common mission, and 2) rewrite the CIO position description to fold the two directorships into a single executive role.

Historically, the two departments had settled into niches aligned with the academic and administrative functions of the University. For the most part, these two niches were distinct enough to operate independently and provided a measure of autonomy for each department. In situations where missions overlapped, the two department directors and their staff were able to meet and come to some sort of an agreement how to proceed.

The introduction of enterprise-wide applications and systems exposed the dependencies between these two operations. As the cross-over expanded, positions once the domain of a single department now contained jobs that traversed organizational boundaries. Contention over resources or from philosophical differences in these overlapping areas tended to percolate up through the departments to be resolved by the directors. However, the number of instances where an agreement could not be reached seemed to be increasing and often entailed intervention by Administration.

Consolidating the two departments under a single executive position with a common mission appealed to Administration and potentially would lead to improved decision making within the organization. Merging the two departments meant identifying and documenting the various jobs performed by both organizations. Many of the jobs could be found in the existing position descriptions posted by each employee; however, new jobs or jobs that would change as a result of the merger would also have to be included. Once all the jobs were compiled, they could be assessed individually, regrouped, and incorporated into the position descriptions for the new organization.

This need to identify and assess the jobs performed between the two departments led to the original idea for the capstone project. A systematic job analysis would differentiate between those jobs specific to a given department and those jobs common to both. This approach had innate appeal in that it offered students a holistic view of an information technology organization based on the jobs the organization performs. In addition, project scope appeared manageable and the objectives clearly defined.

## Class Overview

CIS 456 Project Management is the capstone course for the Computer Information Systems major at the University of Wisconsin-Superior. The class is offered spring semester to seniors in the program and is generally the last class students take before graduating and entering the world of practice. Students who register for the class are expected to have completed a majority of classes in their program of study, including foundation, general education, core business, and major specific classes [4].

The semester-long course carries a 3-credit designation in combination with a mandatory lab. Class and lab times are combined to form a two hour and fifteen minute period that meets twice a week. Kathy Schwelbe's text, *Information Technology Project Management* [5], is used as the primary text for the class in conjunction with a supplemental reference by the Project Management Institute, *A Guide to the Project Management Body of Knowledge* [6]. In addition, class-specific materials are available online that include fact-sheets, check lists, forms, and document templates.

Normally, the class is organized using a matrix structure that integrates project activities with department services. Cross-functional teams form the backbone of the organization creating a dynamic, goal-oriented environment that achieves project objectives. However, only five students were enrolled so a compact form of the matrix structure was employed consisting of one project and two functional departments. Although small, it still required members to show initiative, communicate with one another, solve problems, think critically, and justify their actions.

The organizational development project was the sole project assigned to the capstone class of 2006. Best characterized as an analysis project, the goal focused on trying to figure out exactly what jobs each department actually did and presenting it a clear and systematic manner. Specifically, the project objectives were to 1) compile a set of uniform job descriptions; 2) record each position's constituents, systems, and affiliations; and 3) identify training opportunities for individuals filling supervisory roles in the organization.

Class content was organized around two basic threads: the objective-oriented project that emphasized deliverables and the function-oriented departments that emphasized services. Project activities included the project kick-off, scope review, project review, and the final presentation. Department activities included the qualification exam, functional review, and group exam. Both threads ran concurrently with various activities and milestones intermixed throughout the semester.

In addition, there were a number of activities designed to help students improve their project management abilities. For instance: Weekly activity reports helped students learn time management skills. Status reports and presentations helped students improve verbal communication skills. Cover letters and resumes helped students with job application skills. Position descriptions helped students learn responsibility and assignment skills. And, reports and essays helped students develop their writing skills.

A comprehensive set of metrics that included individual activities, group activities, administrative activities, and peer evaluation and participation were used to measure student performance. The first three areas constituted 90% (each category worth 30%) of the final grade and were based on external measures of performance established by the instructor. Peer evaluation and participation contributed 10% to the final grade and allowed students to assess the performance of team members using predefined criteria.

## Capstone Experience

The CIS 456 Project Management class of 2006 began on Tuesday, January 24<sup>th</sup> with a welcome letter presented to each student. Two attachments were included outlining the organization's structure and available positions. In the letter, students were asked to submit a letter of application stating their preferred project and department positions and a current résumé by Friday, January 27<sup>th</sup>. Prior to the deadline, the class met to discuss various strategies and approaches needed to write successful application materials for positions in business and industry in preparation for their application packets.

On Tuesday, January 31<sup>st</sup>, students received a formal letter indicating their position assignment. The organization consisted of two departments and one project group. Departments were formed around the functional activities of project management to include Finance, Procurement, and Human Resources (FPHR); and Scheduling, Risk, and Quality Management (SRQM). All participants were included in the project since there was only one. In addition, standing and ad hoc committees were convened to manage organization activities throughout the semester.

Once the organization was populated, students were required to write position descriptions for their respective department and project assignments. Position descriptions were processed for content and consistency through the newly formed Finance, Procurement, and Human Resources Department. Tuesday, February 7<sup>th</sup> was the deadline for submitting the position descriptions.

The project kickoff commenced on Thursday, February 2<sup>nd</sup>. Given that this was an internal project where the instructor also served as the project sponsor, the kick-off entailed an in depth discussion of the IIT Consolidation Project and the objectives being sought. It also provided an opportunity for students to ask questions and start formulating a plan to achieve project objectives.

Project and department teams began to meet on a regular basis after the project kick-off. A significant portion of class and lab time was dedicated meetings, however, teams often met outside of class. In addition, weekly activity reports enabled members to track activity progress and status reports scheduled throughout the semester updated progress made toward completing project or department activities.

The scope review was the first major presentation for the project team. The review was made during class on a date scheduled between February 16<sup>th</sup> and February 24<sup>th</sup> by the project team. In preparation, the project group was required to prepare a scope review

packet that consisted of a project charter, statement of work, a work breakdown structure with a tentative schedule, and a copy of their presentation slides. The project team passed the scope review on their first attempt.

After the scope review, a qualification exam was given on Tuesday, March 7<sup>th</sup> to simulate a professional certification exam. Members from the various departments created and disseminated learning materials to help each other prepare for the test. The exam was divided into two portions that covered the functional knowledge areas of project management to include finance, human resources, procurement, quality, risk, and scheduling. The general knowledge portion contained material the whole class was expected to know; whereas, the specific knowledge portion covered in-depth material associated with particular functional departments.

Functional reviews immediately followed the qualification exam and each department was expected to schedule a review between March 7<sup>th</sup> and March 18<sup>th</sup>. The review provides an opportunity to assess the competency of individual department members in their respective functional areas. Each department prepared a packet that contained a mission statement, services offered, methodologies, and a copy of their presentation slides. In addition, department members evaluated the presentations of their peers and performed a self-evaluation using a video recording of their own presentation. Both departments passed with only one member being asked to return for a second attempt.

Project reviews were the next item on the agenda after the functional reviews. The project team was required to set up a meeting between April 4<sup>th</sup> and April 15<sup>th</sup> in which to make their presentation. This review assessed project progress and offered a roadmap how the team would successfully complete the deliverables articulated in the statement of work. For the presentation, the team prepared a packet that contained a final work breakdown structure and schedule, progress to date, a plan for carrying out the remaining activities, and a copy of their presentation slides. The project team passed the review on their first attempt.

Once the project review was completed, a group exam was given on Thursday, April 20<sup>th</sup>. The two-part exam is collegial in nature and designed to strengthen team cohesion while evaluating project management knowledge. First, department teams competed in a challenge called *Put Your Best Foot Forward*. The objective was to formulate and present a response to address a hypothetical challenge a project might face. Teams were ranked based on the strategy employed and the content of their response. Second, project members played *Project Jeopardy* based loosely on the popular television show *Jeopardy*. The objective was to formulate a correct “question” in response to an “answer” provided in one of five project management areas.

As the end of the semester approached, the project team compiled and documented their work and recommendations into a final report, *IITCP Organization Development Project* [7]. The report was reviewed for content and prepared for electronic distribution. The final presentation came on Tuesday, May 9<sup>th</sup> in a closed session due to the sensitivity of the material being presented [8].



A lessons-learned session took place on the last day of class on Thursday, May 11<sup>th</sup>. This session included a systematic review of the activities accomplished over the semester with discussion of alternate approaches in light of their experience. An economic assessment of the monetary value of their work was conducted based on similar projects found in industry or funded by external grants.

Finally, class members were asked to submit a formal peer evaluation and an essay describing their experiences in class. The peer evaluation required participants to objectively rate the performance of members in their project and department teams. The essay gave students an opportunity to tell their story about what they did and offer advice to future students. All the essays were compiled in a document called the *Project Management Cookbook* [9].

## Project Outcomes

Before examining the particular outcomes of the capstone project, it is useful to consider the dynamics of the consolidation effort that stimulated much uncertainty and ambiguity. In hindsight, issues stemmed from tension between the two IT organizations; administrative turnover; and changes to project expectations. To begin with, department methods used to service administrative and academic functions brought to light a number of managerial differences. This led to increased concerns regarding the direction of the new organization, job security, and job satisfaction. The slow pace of the merger only agitated staff further heightening their sensitivity to external measures.

In addition, the consolidation project exercised its directive during a transitory period in the University's administrative ranks. The foundation of project was formulated and prepped under one Provost who resigned to pursue another opportunity as the project was being launched. Soon thereafter, an interim Provost was announced as a search for a permanent replacement was set in motion. By 2006, a new Provost had been recruited and was poised to take the operational reins of the University beginning that February.

If that were not enough, worsening fiscal conditions further impacted the operating budgets of the University. It was fully anticipated at the beginning of the merger that the new IIT Department would be allowed to fill vacant positions once complete; therefore, the proposed organization structure adopted a modest growth strategy. Near the end of the project, however, budget pressures mounted forcing a reconsideration of the strategy to a no-growth staffing plan that could survive a potential downsizing effort.

Fortunately, there was a measure of separation between turbulence created by these external conditions and the capstone project itself. This allowed the project team to formulate a concurrent strategy to pursue the three objectives outlined at the onset of the project. Nonetheless, students still faced uncertainty and ambiguity as they engaged the process.

Compiling a uniform set of job descriptions was the primary objective of the project. Neither organization maintained a detailed list of jobs that could be used to assess the

areas where the missions of the two departments overlapped. However, both departments maintained up-to-date position descriptions that contained employee duties and responsibilities. Most of these records were quite detailed although content and style varied by department and employee type.

Positions and jobs are complementary views of accomplishing work in an organization. A position defines a resource—an employee—available in fixed-time increments (full-time, half-time, etc.) to accomplish work; whereas, a job defines related tasks needed to accomplish work. Conceptually, positions and jobs exist as two distinct sets related by a many-to-many relationship. In practice, however, position descriptions are written for employees often without specific reference to a formally defined set of jobs as in this instance.

Indirectly creating a formal set of jobs from position descriptions caused considerable uncertainty and ambiguity for the project team. This situation is analogous to learning the dual solution for the Simplex Method. You have to really understand what the primary solution means before the dual solution makes sense. In response, team members created a structured form that included common attributes to describe each job, such as job title, job summary, qualifications, duties and responsibilities, evaluation, knowledge, and the individuals currently assigned.

Using this standard form, the project team compiled an inventory of 49 distinct jobs based on 18 position descriptions. (Note: This number includes full-time and part-time staff and supervisors, and excludes the two Director positions.) Some jobs could be clearly articulated while others needed further information to describe them completely. It is interesting to note that nearly a 2.5:1 ratio exists between the number of jobs and positions across this organization as a whole. Nonetheless, a standardized form created the framework in which to conduct a systematic inventory.

Recording each position's constituents, systems, and affiliations (CSA) was the second objective of the project. A CSA analysis documents relationships an employee has with constituents—the external population an employee serves, systems—the technologies and processes an employee works with, and affiliations—the other members of the organization an employee collaborates with. These relationships were crucial to finding the functional overlap between the two departments.

Originally, the option of letting students interact directly with the IT staff to perform the CSA analysis was available. Unfortunately, members of the two departments objected to any student involvement as the consolidation project moved forward. The author responded to these objections by limiting student participation to nonintrusive activities. The uncertainty and ambiguity caused by this midcourse correction required project team members to find a new way to acquire the data needed for the analysis.

Team members considered four alternatives to help the project facilitator gather the data needed for the CSA analysis. Alternatives included: implementing a survey, conducting one or more brainstorming sessions, using a Delphi Technique, and conducting direct

interviews. After evaluating each alternative, the project team recommended a survey to overcome access and timing limitations. The team crafted a survey instrument and proposed a methodology as a starting point to collect data.

Identifying training opportunities for individuals filling the coordinator positions in the new IIT organization was the third objective of the project. Coordinator positions serve two purposes. First, coordinators were expected to manage the day-to-day activities of a five to six member team. This enabled a division of labor strategy to minimize span of control problems as the two departments merged. Second, coordinators were expected to work with the CIO to develop tactical plans to reconcile strategic and operational activities. This was particularly important in light of the new executive responsibilities granted the CIO.

Identifying and selecting training opportunities to prepare coordinators for their new duties proved quite challenging. The project team had to adopt a new perspective of learning that included objective-based professional training presented as workshops and seminars. In addition, students needed to consider training objectives and assess how training programs met those objectives. This view of training is substantially different from the college experience most were familiar with and caused uncertainty and ambiguity for the project team.

Initially, the project team misunderstood the training objective and began to look at training programs for specific technical skills. They soon realized that they were ill-equipped to recommend training in specialty areas that members of the two departments already possessed. A meeting with the project team helped them revise their understanding of the objective and how to proceed. The resulting search yielded six potential training programs that addressed the supervisory and tactical planning needs of the coordinator position. For each training program, team members wrote a summary, listed pertinent topics, performed a cost analysis, and addressed procurement issues.

## **Summary**

This paper considers the influence of uncertainty and ambiguity in a project management environment. The capstone class of 2006 provided an exceptional opportunity to observe how students manage project dynamics while striving to learn project management skills and practices. A description of the project, class, and activities depicts the setting in which student learning takes place. The approach used by students to overcome uncertainty and ambiguity is presented in the context of achieving project outcomes.

This work recognizes the importance of learning to manage uncertainty and ambiguity. Learning is a dynamic process and includes conditions that elude quantification. Even the most experienced instructors have difficulty anticipating the all circumstances that give rise to unplanned events and behaviors in a capstone class. Yet, acknowledging these conditions is an essential part of capstone pedagogy. Recognizing the intangible aspects of education and openly discussing classroom experiences offers insight into the mechanisms that contribute to a successful learning environment.

Finally, this work continues to add to the body of knowledge regarding capstone experiences; however, there is more work that needs to be accomplished. Further studies may include identifying specific factors that precipitate uncertainty and ambiguity in a project management environment, the degree that structured methodologies lessen the impact of uncertainty and ambiguity, and measures to gauge an organization's ability to manage uncertainty and ambiguity.

## References

- [1] Lynch, S. M. The Capstone Experience: Integrating Curricular Outcomes with Real-World Practice. *Proceedings of the 37<sup>th</sup> Midwest Instruction and Computing Symposium*. Morris, MN, 2004.
- [2] Lynch, S. M. The Capstone Experience: Balancing Formal Project Management Methodologies with Student Creativity and Innovation. *Proceedings of the 40<sup>th</sup> Midwest Instruction and Computing Symposium*. Grand Forks, ND, 2007.
- [3] Meachen, E. Report on the UW-Superior Organization. University of Wisconsin-Superior, Superior, WI, 2005.
- [4] University of Wisconsin-Superior. *2004-2006 General Catalog*. Superior, WI, 58-59, 2002.
- [5] Schwelbe, K. *Information Technology Project Management*. Course Technology, 2006.
- [6] PMI Standards Committee. *A Guide to the Project Management Body of Knowledge*. Project Management Institute, 1996.
- [7] Lynch, S. M., Ferguson, K., Dahlberg, C., Samarasekera, K., and Padmasiri, D. *IITCP Organization Development Project*. IIT Consolidation Project: University of Wisconsin-Superior. Superior, WI, 2006.
- [8] CIS 456 Project Management. *Capstone Presentation*. University of Wisconsin-Superior, Superior, WI, 2006.
- [9] CIS 456 Project Management Class of 2006. *CIS 456—Project Management Cookbook, Class of 2006*. University of Wisconsin-Superior, Superior, WI, 2006.

# **AN ALGORITHM TO RESTORE DATA BASE CONTENT TO PAST DATES IN REAL TIME**

Christopher Brown, Dennis Guster and Brittany Jansen  
Business Computing Research Laboratory  
St. Cloud State University  
St. Cloud, MN, 56304  
chrisb@stcloudstate.edu

## **Abstract**

Although it is important to devise an accurate and reliable methodology for updating a data base (so that current information can be rapidly accessed), there are times when it is equally important to be able to roll back the transactions (Date, Darwen, & Lorentzos, 2003). This is so that reports can be generated that draw on that data in its past form. This need makes it important to have the capability to both draw from the actual data base table(s) as it is and draw from the historical data base table(s) as it was. Currently, there are three popular methodologies regarding this, which include restating, tracking or taking a snapshot of the history. However, a method was also devised, featuring an algorithm that addresses relational constraints. This algorithm, and its deployment on an operational level, makes it possible to go back and generate reports for any given day.

## Introduction

Although it is important to devise an accurate and reliable methodology for updating a data base (so that current information can be rapidly accessed), there are times when it is equally important to be able to roll back the transactions (Date, Darwen, & Lorentzos, 2003). This is so that reports can be generated that draw on that data in its past form. This need makes it important to have the capability to both draw from the actual data base table(s) as it is and draw from the historical data base table(s) as it was. An example of this might be a university system that reports grade point averages. Perhaps a comparison from year to year is desired. A report might be run in 2005, and when its users wish to run the program again in 2006, the people requiring the data wish to include a break out by gender, which was not included in the 2005 report. It is easy to get the current information, but in order to be able to generate the report for 2005, some type of roll back mechanism would be needed. Devising a methodology to provide such capability is not trivial and requires much thought, due to the volume of data that is probably involved. A quick overview of the concerns and frustrations appears in Date (2003).

Obviously, one method would be to create historical snapshots of the reported data on a periodic basis, such as the end of each business day. However, this may not provide the required flexibility to recover to a specific transaction point and would require the storage of massive amounts of data, which may also limit speed of access. Currently, there are three popular methodologies described in the literature. Mundy (2007) describes the Kimball Method which advocates the following three techniques.

Type 1: Restate history by updating the dimension attribute in place.

Type 2: Track history by adding a new row to the dimension table that contains the new view of the dimension member.

Type 3: Snapshot history by adding new columns to the dimension table that holds the attribute's values at a specific date, often year-end.

Although the effectiveness of these methods has been proven, there are still issues with overhead, performance or ease of use by end-users. All of these issues can be directly related to operational cost. Tao and Papadias (2005) recognize the need to cost optimize the history process and feel that other models that reduce cost merit investigation. For the application to be delineated in this paper, Type 2 offered much promise in regard to the output desired and the adaptability to the current database structure. However, the overhead and potential performance required lessened its attractiveness. An analysis of the temporal requirements revealed that it provided a higher degree of granularity than was needed.

The type two model provides granularity at a per transaction level, when all that was needed was a per day level. In other words, any updates to a given record will override each other that occur on the same day. More specifically, a method was devised, featuring an algorithm that addresses relational constraints. That is, the historical table is actually part of the same database, which allows for consistency within the search key methodology. For each table, a custom history table was devised using the current table

as a base template and adding a column for a history date, record action (Insert, Update, Delete), and a history surrogate key. Historical records were retrieved by sending the historical report date to a cross applied view of the history table and a calendar day table, where the appropriate date/time in history was reached by pulling records with the max date  $\leq$  the report date and where record action  $\neq$  Deleted.

The preliminary analysis has revealed that this method integrates well into an existing database and only generates about 110% more storage overhead. The additional processing overhead are minimal, due to indexing that can be done on the history table, and as the writes are minimal. The development cost of implementing this history method is relatively 10% of the total development cost of the ETL process. It is important to note the flexibility of selecting which stage in the ETL development process this history model can be implemented. Due to the trigger relationships, the cross apply function, and with the single stipulation of a required segregate key on the modeled table, the history method can be implemented at any stage in the ETL development process with negligible difference in development cost.

An added benefit of this method was observed in the simplicity of converting a regular dynamic report to a historical report by changing the table name to the view name and adding a report date qualifier. This change was realized through the table structure that mirrors history to the original table with the addition of the three prior mentioned columns. Through triggers and the use of a cross apply operation, it becomes essential, from a performance standpoint, to optimize the use of indexes and maintain a maintenance plan for each history linked table. Although it was devised on a university record keeping system, it appears the basic templates have wide transferability to a wide variety of data base applications. A more detailed description follows later in paper, including flow charts depicting the storing history and reporting history. Further, tables illustrating an historical inquiry will also be used to clarify the logic of the algorithm and trigger. View templates will be provided to assist the reader in adapting the methodology to other database designs.

## **Review of Literature**

The material synthesized in Date Darwen & Lorentzos (2003) indicates that are effective ways to deal with resetting a database to some date (or transaction point) in history. However, databases are complex in nature, and although the basic algorithm options appear sound, there is a need to customize the basic algorithms for the sake of performance or to reduce cost (Tao and Papadias, 2005). On the server side, one of the prime concern is performance, which is compounded by the change requirement of both data and structure associated with historical resets (Yang and Widom, 2003). One approach in dealing with this performance problem is to focus on the algorithm. The work of Moon, Lopez & Immanuel (2003) focuses on query processing as related to the temporal grouping. They report success and some of their parallel adaptation have achieved almost a linear speedup and scale up. They further state that this contribution is of particular importance given the fact that the rate of increase in data size and response



time requirements has outpaced achievements in processor and mass storage systems. Gao, et al (2004) agree that as data bases get larger it becomes more and more difficult to meet the desired performance targets and that parallel processing becomes a very useful tool in solving this problem. In addition to the basic algorithm, there is also promise in focusing on the indexing method. Stantic, Khanna and Thorton (2004) state indexing strategies may need to be customized to deal with and optimize the data. This, of course, is in the new time dependent world, where a substantial amount of the data may be *now-relative* instead of fixed within the basic defined structure of the database. In some cases, they have even achieved better efficiency by a factor of 20.

On the client side, where the inquiries are devised, there also concerns that is cost-related. Zimanyi (2006) reports that often the historical related systems have not been devised to take advantage of standard SQL or have the required complex SQL coding. His paper provides insight and methodology on how to realize temporal aggregates and temporal universal qualifiers using standard SQL. Kang, Chung & Kim (2004) also agree on the importance of temporal aggregates and have used and have developed a methodology to transform the time interval of a single record into a single value to make queries easier to define.

## **Details of the Algorithm**

Two flow charts and a series of four tables are used to describe the algorithm and methodology. The first flow chart provides an overview of how the history data will be used. This flow chart, Figure 1, is depicted below. The entry point into this chart is the “if condition”: Does the inquiry request information from a prior date? If not, then it is simply a matter of using an existing table and then ending the process. However, if a prior date is requested, then it is necessary to determine if a dynamic or a static report is required. The dynamic report reports all data (including past data) as it *is* and looks to the active non-history table for all record inquires. Whereas the static report reports history data as it *was*. This requires that the database be rebuilt and rolled back to the desired date using the where maximum history data < report data and action != delete conditional within the cross apply operation.

# Storing History

V 2.2

**Update** – Occurs when a PK in the original record matches the PK in the changed record and one or more non-PK field values have changed

**Delete** – Occurs when a prior PK entry is removed from the original table

**Insert** – Occurs when a new PK entry is added to the original table

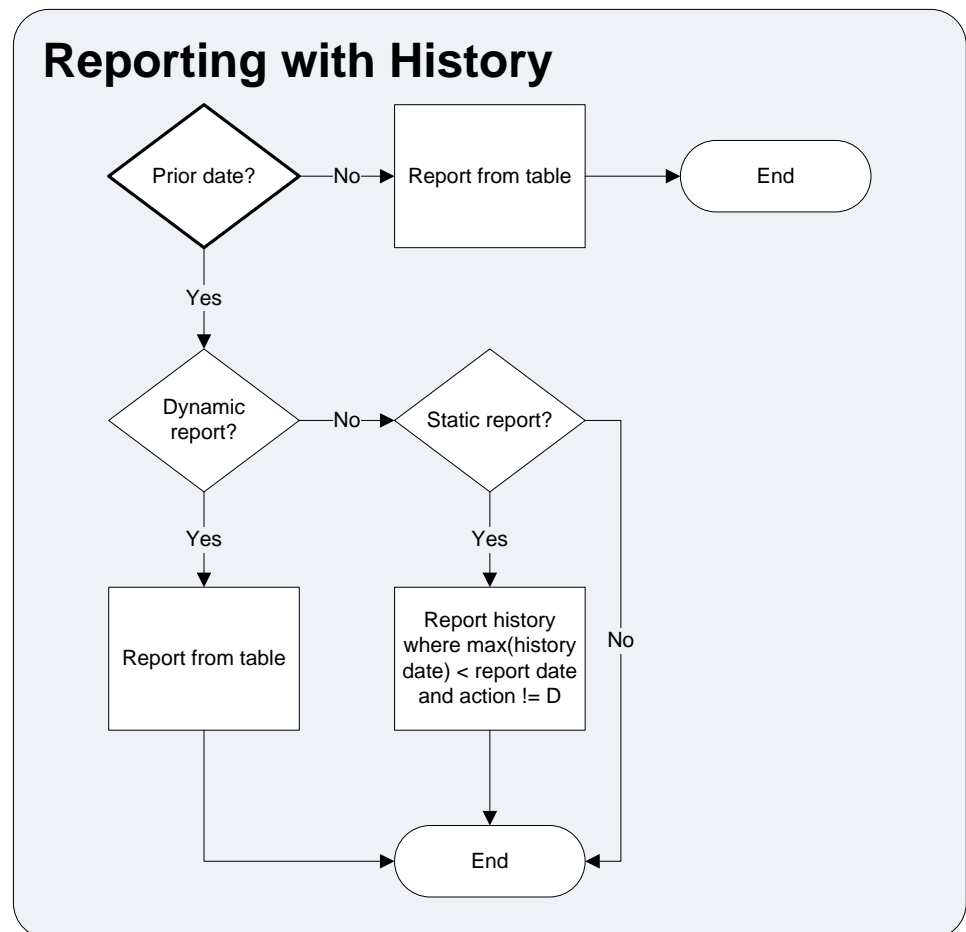


Figure 1: Overview of Storing History Data

The storing of history is depicted in Figure 2 and provides a detailed assessment of the logic used. The first level of logic depicted is whether or not the action is to be an update, add or delete. If update is selected, then it is necessary to determine if there was a prior entry in history today. If not, then a new record is added to history with an action flag of "U." If there is a history record for today, the action flag is checked, and if set to "I," the history record for today is updated accordingly. However, if there is an existing record for today and the action flag is not set to "I," then the action flag is set to "U" and no further updates are made to that history record.

If add is selected, then it is necessary to determine if there was a prior entry in history today. If not, then a record is inserted to history with an action flag of "I." If there is a history record for today, it is then an erroneous situation because the logic of the method does not allow it, as it would be in violation of a constraint requiring the unique occurrence of a record ID to one per day. This would be undoubtedly intercepted by the db engine error handling and would not progress to the trigger. However, if there is not an existing record for today, then the action flag is set to "U" and no further updates are made to that history record.

If delete is selected, then it is also necessary to determine if there was a prior entry in history today. If not, then a deleted record is added to history with an action flag of "D." If there is a history record for today, then the action flag is checked, and if set to "I," the history record for today is deleted accordingly, as the record was inserted earlier that day and would not require any historical indication of being added. However, if there is not an existing record for today, then the action flag is set to "D" and no further updates are made to that history record.

One observable logical inference within this model relates to the multiple pass operations. For example, when performing an add, update, update operation, the outcome will vary from an update, delete, add operation. The first example (A,U,U) would leave the history record with a value of the final update. Alternatively, the later example (U,D,A) would leave the history record with a value of the initial record value. This is important logical construct, as it enables retention of initial values with the exception of adding a record. This imbedded logical construct can be understood as preserving record data on a per day basis. One that is initiated with an update or delete action, and overwriting records change on a per day basis as well, originating with an add operation. Construction of a truth table can assist with mapping out these embedded constructs.

# Storing History

V 2.2

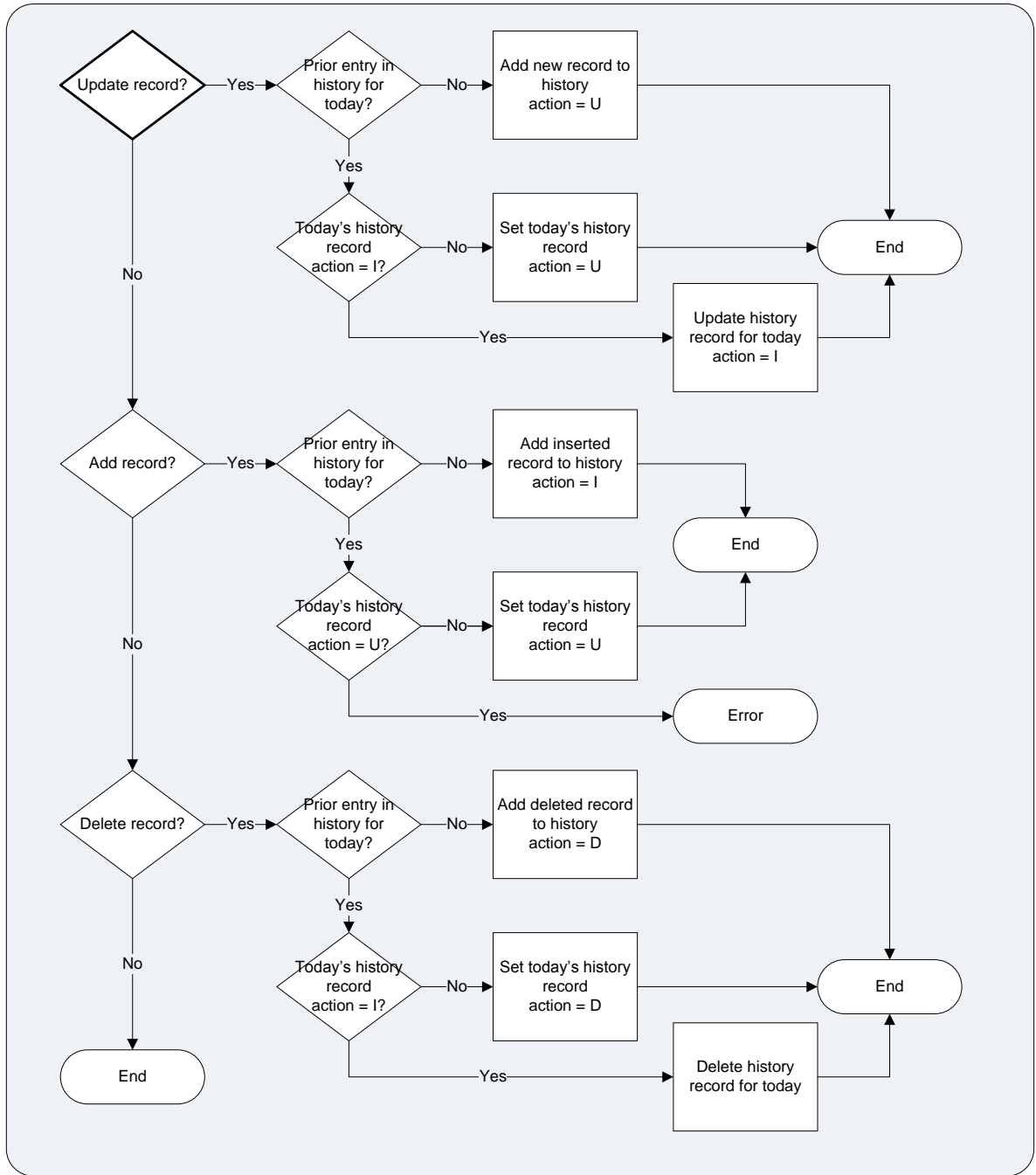


Figure 2: Storing History and Logic Use

Table 1 provides three examples that delineate the some of the possible action scenarios listed in the truth table that checks the validity of action combinations. The examples are

based in a database that keeps track of student grades. It is possible that grades could change over time due to such mechanisms as IP (in progress), I (incomplete) or a calculation error may even result in a change from one letter grade to another after the fact. The examples show how a historical related grade change would be handled by the database logic described herein. In example one, the sequence insert, update, delete is explored. In the first row there is no record for today so a value must be inserted. That value is inserted into the table record and then also placed in the history record. The second row depicts an update in that the table record has a value of A- so the history table receives the same value due to the initial insert earlier that day. The third row then shows a situation that should not occur because it would result in the generation of a duplicate primary key, and the record would have to be deleted before it could be re-added. The fourth row has similar logic to the second row except this is an additional change to the record. In other words, the record had been modified twice within the time period specified. The fifth row depicts a delete, which results in no record in either table. Again, adding or deleting records in history only occurs when the initial entry in history has an action type of insert. The sixth row then demonstrates an attempt to update a record that does not exist, which of course is not allowed. Finally, the last row illustrates adding a record, which would satisfy the database engine requirements as the insert follows a delete.

In the second example the sequence update, delete, insert is explored. In the first row, there is no historical action variable. The new action variable dictates an update so the history record is added with the action variable set to update and grade value set to A to reflect the grade prior to the edit. In the second row, a delete is requested, which removes the value from the table record, changes the history action to delete but leaves the history grade value untouched so it can be used in future inquiries if needed. The third row illustrates an insert, which adds a record to the current table with a grade of C+. The history action variable would be changed from delete to update. If the initial action had been insert, the history action variable would be set to insert and the history grade value would be set to C+; however, it is necessary to persist to the original value of the record, requiring the update action as a means of preserving the initial grade value. In the fourth row an update is depicted in which the C+ value in the table is updated to a B-. The action variable in history would not change as its state prior to the update is consistent with the action performed. The last row, in example two, depicts a second delete process that occurred within that allotted activity timeframe. The action variable in history would change from update to delete to reflect the action performed.

The third example depicts the delete, insert, update sequence. In the first row there is no historical action variable. The new action variable dictates a delete so the table record is deleted and an A remains in the history record. In the second row, an insert is requested, which changes the history action variable from a delete to an update while retaining the history grade value of A. The third row illustrates an update, which results in the table record being updated to an A and no change to the history action variable. In the fourth row a delete is taking place, which means no grade value is left in the history table; of course, the history action variable is set to update, and the history grade record remains untouched. The last part of Table 1 depicts allowable actions based on the database logic.

The check marks indicate the sequence is allowed and the x's indicate that the sequence is not allowed.

One challenge of using this logic model is the costly overhead to eliminate unnecessary records in history. An example of this occurs when the final result of the record is the same as the initial state. In this case, history would not require a record for that timeframe. The trigger would have to contain additional logic to "clean up" any history records reflecting a record final value equal to its initial state.

**Multiple changes on same day:**

Note the Action Variable and data record when same-day sequential data loads are performed.

**Example1: Insert Update Delete**

History Id	History Date	History Operation	History Action Variable	New Action Variable	Table Record Value	History Record Value	
1	11/1/2007	Insert	<i>none for today</i>	I	I	I	
1	11/1/2007	Update	I	I	A-	A-	
1	11/1/2007	Insert	<i>*** Duplicate PK - Operation Not allowed ***</i>				
1	11/1/2007	Update	I	I	A	A	
1	11/1/2007	Delete	I				
1	11/1/2007	Update	<i>*** No record to update - Operation Not allowed ***</i>				
1	11/1/2007	Insert	<i>none for today</i>	I	A	A	

**Example2: Update Delete Insert**

History Id	History Date	History Operation	History Action Variable	New Action Variable	Table Record Value	History Record Value
2	12/13/2007	Update	<i>none for today</i>	U	C	A
2	12/13/2007	Delete	U	D		A
2	12/13/2007	Insert	D	U	C+	A
2	12/13/2007	Update	U	U	B-	A
2	12/13/2007	Delete	U	D		A

**Example3: Delete Insert Update**

History Id	History Date	History Operation	History Action Variable	New Action Variable	Table Record Value	History Record Value
3	12/1/2007	Delete	<i>none for today</i>	D		A
3	12/1/2007	Insert	D	U	C	A
3	12/1/2007	Update	U	U	A	A
3	12/1/2007	Delete	D	D		A

Step 1	Step 2	Step 3	Ok?
Insert	Update	Delete	✓
Insert	Delete	Update	✗
Update	Insert	Delete	✗
Update	Delete	Insert	✓
Delete	Insert	Update	✓
Insert	Update	Update	✓
Insert	Delete	Insert	✓
Delete	Update	Insert	✗

**NOTE:**  
The X's indicate errant combinations, due to database constraints

Figure 3: Insert, Update and Deletion Examples

## Discussion and Conclusions

There are some applications in which being able to rollback the database to align its contents to some point in history is critical. If that is the case, there is always a concern over the required overhead to support that capability and provide adequate performance in regard to inquiries that need to access it. One method to address these problems is to examine the degree of granularity that is truly required for a given application. The ultimate level is on a per transaction basis. However, this level may not be needed for all applications. In the example described herein, involving the recording of student grades, a granularity on the day basis is adequate. In other words, being able to display the grade of record at the end of any given day is the goal of records department. Therefore, supporting per transaction granularity is not needed and would entail a waste of resources.

The strategy used herein, in which any updates to a record will overwrite each other in history on the same day resulting in a per day granularity, has several advantages over the per transaction level. First, the overwrite per day process results in a database that takes up less space. Second, because the database is smaller retrieving data is quicker and easier. Third, the generic process used makes it easier for developers and end-users. This is especially attractive to end-users and the people that need to support them because the complexity of the search is lessened. This means that a greater proportion of the search can be made directly by end-users without help from the database team.

From a design perspective it was also found that there were advantages to integrating this functionality into the existing database structure rather than create a second database just to support the history roll back function. To achieve the desired functionality, a new history table was devised for each table, requiring history tacking within the existing database structure and populated accordingly by an initial mirroring of the original table, and then sustained through three triggers (update, insert and delete). This made the design

much simpler, but there was a concern that this additional overhead might slow down the primary function of the database, which was real time access to the current records. By properly applying indexes, performance latency no more than doubled that of querying the original table. This concern can be reduced to a more acceptable point by building the historical records on a separate disk spindle.

In summary, as a result to this algorithm and its deployment on an operational level it is now possible to go back and generate reports for any given day that was not generated when the data was originally staged. A prime example of this type of need was a grade report generated by course ID on the department level that was seen during strategic planning by the Dean's office, who decided it would be useful to have that same report on that same day but generated for the whole college. The design described herein not only makes that possible, but it can be generated quickly, efficiently and, most importantly, cost-effectively. This scenario means that not only is the information somewhere in the computer system but is readily available in real time to support core decision making, which is a testament to the functionality and flexibility of the design.

## References

- Date, C. (2003). "On modeling history. *Database Debunkings*.
- Date, C., Darwen, H., & Lorentzos, N. (2003). *Temporal Data and the Relational Model*.
- Gao, et al. (2004). Main memory based algorithms for efficient parallel aggregation for temporal databases. *Distributed and Parallel Databases*, 16(2): 123-163.
- Kang, S., Chung, Y. & Kim, M. (2001). An efficient method for temporal aggregation with range-condition attributes. *Information Sciences—Informatics and Computer Science*. 168(1-4): 243-265.
- Mundy, J. (2007). Handling arbitrary restatements of history. *Intelligent Enterprise*.
- Stantic, B., Khanna, S. & Thorton, J. (2004, January). An efficient method for indexing now-relative bitemporal data. *Proceedings of the 15<sup>th</sup> Australasian Database Conference*. Dunedin, New Zealand, January 1, 2004 (pp. 113-122). Darlinghurst: Australian Computer Society, Inc.
- Tao, Y. & Papadias, D. (2005) Historical spatio-temporal aggregation. *ACM Transactions on Information Systems* 23(1): 61-102.
- Yang, J. & Widom, J. (2001). Incremental computation and maintenance of temporal aggregates. *International Journal of Very Large Databases* 12(3): 262-283.



Zimanyi, E. (2006). Temporal Aggregates and temporal universal quantification in standard SQL. *ACM SIGMOD Record* 35(2): 16-21.

# **A Methodology for Design, Development, and Implementation of Data Warehouse Project for University Upper Level Course using Microsoft SQL Server Analysis Services**

Cyrus Azarbod, Muhammad Rizwan Farooq  
Department of Computer & Information Sciences Department  
Minnesota State University at Mankato,  
273 Wissink Hall, Mankato, MN 56001  
[cyrus.azarbod@mnsu.edu](mailto:cyrus.azarbod@mnsu.edu)

## **Abstract**

Data warehousing is an emerging technology that provides users the ability to perform better analysis and to make business decisions easier. In order to gain competitive advantage, organizations use systems that automate business processes to offer more efficient and cost-effective services to their customers. As a result of that typically everyday an organization captures large amount of data in its operational systems. Organizations need to turn their archives of data into a source of knowledge, so that a single integrated / consolidated view of the organization's data is presented to the user. A data warehouse was deemed the solution to meet the requirements of a system capable of supporting decision-making, receiving data from multiple operational data sources. Many institutions and universities offer data warehousing in their advanced database course or a course by itself. Most of the curriculums emphasize on providing a theoretical overview of various components and basic concepts of the data warehouse systems. However, they do not include hands on project dealing with complete life cycle of designing and implementing data warehouses.

This paper will propose a curriculum for a data warehouse course to offer not only theoretical concepts of data warehousing but also to involve students in a hands on experience of designing and implementing a data warehouse system using Microsoft SQL Server Analysis Services.

## Why Data Warehousing?

Everyday organizations typically capture large amounts of data to offer more efficient and cost-effective services to the customer. This data provide some indication of how businesses currently operate. For example, a retail store chain captures all sorts of data for every product, customer, purchase orders, return orders, backlogged orders, and promotions it is offering. This data provides facts such as:

- ✓ How many customers the store currently have
- ✓ What are the products it currently has in stock?
- ✓ What products its waiting for vendors to deliver

However, the captured raw data does not provide information on trends or predict how the organization may operate in the future. Business Information can be derived by exploring and analyzing the raw data in several different perspectives and from several different points of view. The process involves identifying facts that describe a condition and extracting the critical data that reflects the overall business goals and objectives [1].

In the case of a retail store chain typical questions that may arise after analyzing data are:

- ✓ What products are they selling on different year?
- ✓ What products are they not selling on different year?
- ✓ What is the effect of product promotions?

Operational databases are not designed to support such business questions. Organizations need to turn their archives of data into a source of knowledge, so that a single integrated / consolidated view of the organization's data is presented to the user. Data Warehouse systems are specifically designed data stores that support an organization's business analysis process. Often implemented as enterprise wide decision support architecture, a data warehouse system provides a reporting environment to facilitate data analysis. As opposed to operational systems, data warehouse systems are designed to get strategic information out of the database [2, 5]. A data warehouse system transforms and integrates data from heterogeneous sources into a single repository database. "A data warehouse is a subject oriented, integrated, time variant and nonvolatile collection of data in support of management's decision" [6].

## Data Warehouse Architecture

A data warehouse system begins with source systems that capture the business transactions. These source systems can be in heterogeneous environments and consist of multiple operational systems. Data from these operational systems then transform, consolidate and integrate into a staging environment, and from there the data is moved to data marts periodically depending on the business requirements. Figure 1 shows the basic components of a typical data warehouse system.

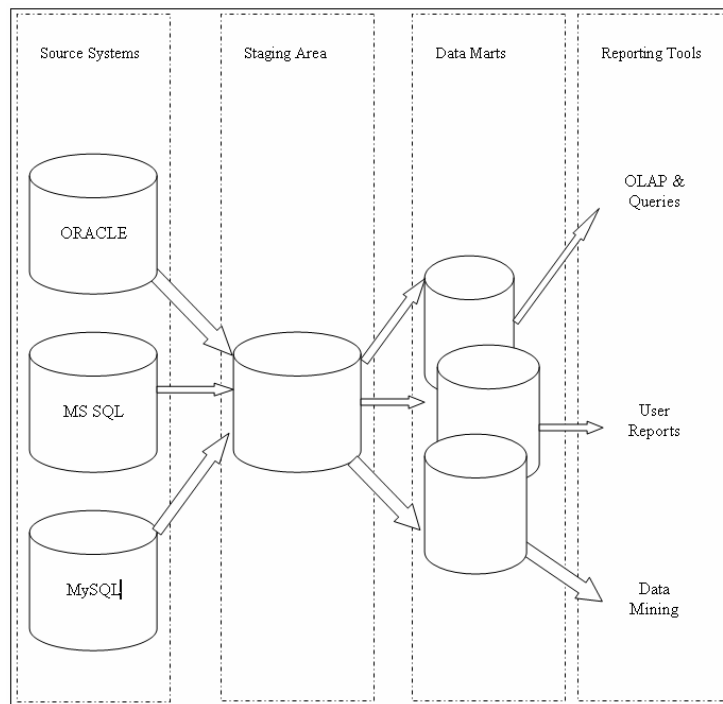


Figure 1: Typical Architecture of the Data Warehouse System

## Data Warehouse Design Life Cycle

The main purpose of designing the data warehouse is to provide a system that can be used to perform data analysis. Careful planning is required in order to achieve this goal. The following are different components of the design and implementation of a data warehouse system:

- ✓ Business requirements, scope and content
- ✓ Data sources
- ✓ Data cleaning / filtering
- ✓ Design DW schema
- ✓ Data migration
- ✓ Staging database
- ✓ OLAP database
- ✓ Analysis

## OLAP in Data Warehouse

OLTP systems are designed on relational models and support day-to-day business processes. Standard Query Language (SQL) is the standard interface to perform such activities in the systems based on relational model. SQL is a data manipulation (select, update, insert, or delete data) and definition language (create table, alter table etc.). Basic reports can be generated from OLTP systems using SQL. However, these reports do not

provide multidimensional view of data to the users. Once these reports are generated, users can not drill down to lower levels to find interesting results [2, 10].

Data warehouse systems, on the other hand, are designed to provide substantial analysis of the data to the users. Data marts are stored in multidimensional structures to support dimensional analysis. Meaningful analysis typically involves retrieval of large amount of data, number of calculations and summarization of the data on the fly. Many queries follow each other in order to get interesting results. SQL could be use to perform such analysis, but in that case SQL statements will be very complex, and likely to involve multiple joins, full table scans, grouping, complex calculations. Hence, performance of the system will be badly effected [2, 10].

On-line Analytical Processing (OLAP) is the answer to provide multidimensional analysis and to support ad-hoc queries with a fast execution time. E.F. Codd, in 1993, defines OLAP as [1]: “On-Line Analytical Processing (OLAP) is a category of software technology that enables analysts, managers and executives to gain insight into data through fast, consistent, interactive access in a wide variety of possible views of information that has been transformed from raw data to reflect the real dimensionality of the enterprise as understood by the user.”

## **Data Model**

This section describes the different type of data modeling techniques used to design data marts. Data mart is a prevailing extension of the data warehouse and provides the data at a very granular level. One of the main goals of data warehousing is to provide faster query performance. To achieve this goal data warehousing must be designed very carefully. There are two distinct types of data marts based on their design.

- ✓ Relational data marts
- ✓ Multidimensional data marts

### **Relational Data Marts**

OLTP or operational systems are typically designed using Entity Relationship (E-R) modeling with no data redundancy. Like OLTP, Relational data marts are designed based on ER modeling. Relational data marts store detailed and summarized data in two dimensional structures, supported by relational database technologies [8, 9].

### **Multidimensional Data Marts**

Multidimensional modeling is a logical design technique that is used in data warehouse systems. The purpose of dimensional modeling is to design business dimensions that support data analysis and reporting requirements. Multidimensional modeling produces de-normalized data structures that simplify the database complexity by reducing the

number of tables. As the structure is de-normalized, the number of joined tables in queries is much lower than in a relational model. It is highly optimized for queries and provides for fast and easy data access to the users. Due to its flexible design, multidimensional modeling can more easily accommodate new data elements and design changes [2, 8, 9]. The main components are:

- ✓ Fact Table. A central table in a data warehouse or data mart presenting numeric data in a context that describes a specific event within a business.
- ✓ Measures. A quantitative, numerical column in a fact table. Measures typically represent the values that are analyzed.
- ✓ Dimension Table. A table in a data warehouse or data mart representing a business entity.

## **Data Warehouse Schema Design**

### **Star schema**

Star schema is characterized as having one central fact table surrounded by many dimension tables that contain de-normalized description of the facts. Fact tables contain numeric measures and references to the dimension tables. The measure should generally be additive and numeric because these values are the basis of a calculation. Reference to the dimension tables are implemented by using foreign key columns [2, 7, 8].

### **Snowflake Schema**

Snowflake schema is the normalized form of Star schema. It consists of a fact table and multiple related dimensions table. In OLTP systems, normalization increases the performance of the system by reducing the redundancy of the data. Transactions can easily update, delete, and insert new data in normalized OLTP systems. On the other hand, normalization degrades data warehouse performance because of the additional joins in the queries [2, 7, 8].

### **Starflake Schema**

Starflake schema is a hybrid structure that contains a combination of normalized and de-normalized structures. It is a compromise between Snowflake and Star Schema. Figure 3 shows a starflake schema [2, 7, 8].

## **Microsoft Data Warehousing Strategy**

The main objective of Microsoft data warehousing strategy is to simplify the procedure of designing, implementing, and managing the data warehouse system. The components comprising the strategy are [10]:

- ✓ **OLTP systems** conduct core business processes by tracking real-time transactions. [2, 10].
- ✓ **Data Transformation Services** is a workflow that is bundled into SQL Server.
  - Transform data between data sources.
  - Automate loading of OLAP cubes.
  - Automate the imports, exports, and transformations of the databases.
  - Define the import, export and transformation as packages that can be used repeatedly
- ✓ **Data Warehouse Storage**
- ✓ **Analysis services** is bundled with SQL Server.
  - Analysis Services can be used with any ODBC / OLE DB complaint relational data sources e.g Microsoft Access, Oracle [10].
  - Analysis Services accesses data from the relational database and provides data to the reporting application. It functions as an intermediate layer that converts relational warehouse data into a form that makes it fast and flexible for creating analytical reports [10].
- ✓ **Client Applications**
  - Excel 2000 is Microsoft's own OLAP client. Within Excel 2000, PivotTables have been enhanced to work smoothly with Analysis Server.
  - Most major third-party OLAP client products that interface with other OLAP tools also interface with Analysis Server.
  - Custom front-end applications can also be created to access Analysis Server by using OLE DB for OLAP [10].

## Proposed Curriculum

The proposed curriculum is based on the study and overview of concepts of data warehouse system, life cycle of DW design and Microsoft Analysis Services data warehouse strategy. It aims to offer a data warehouse course to students, in which they will learn not only the theoretical concepts of data warehousing and various components of data warehouse system, but it will also give them hands on project dealing with complete life cycle of designing and implementing data warehouse using Microsoft Analysis Services, as a class project. Table 1 shows the proposed curriculum for a project based DW course.

1. Teach the Main Concept of DW, Introduction to Microsoft Analysis Services and concept of data warehouse development life cycle.
2. Identify the specifications and scope.
3. Study and understand given specifications
4. Analyze the participating operational databases
5. Identify related sub-schema of individual databases
6. Perform Data Cleaning / Filtering Tasks
7. Identify Global Schema
8. Identify the Entities required to solve the problem

9. Design DW schema
10. Data Migration
11. Develop cubes by administrator
12. Analysis of implementation
13. If proposed design does not meet the specification go back to the step Design DW schema (step 9)
14. If objectives still do not meet specifications then go back to the step Identify related sub-schema of individual databases (step 4)
15. If the objectives still does not meet specifications then go back to the step Identify the specification and scope step (step 2)
16. Develop appropriate reports and evaluations

Table1: Proposed Curriculum for a Project Based Data Warehousing Course

## Implementation of Proposed Curriculum

To demonstrate the proposed curriculum, an order entry database is used to implement a data warehouse project using Microsoft Analysis Services (OES\_DW). Order Entry System (OES) database used as the data source for this project. Figure 2 shows the data model of OES databases. OES keeps track of information such as customers, orders, branches, employees, and vendors. Figure 3 shows the flow chart of the methodology.

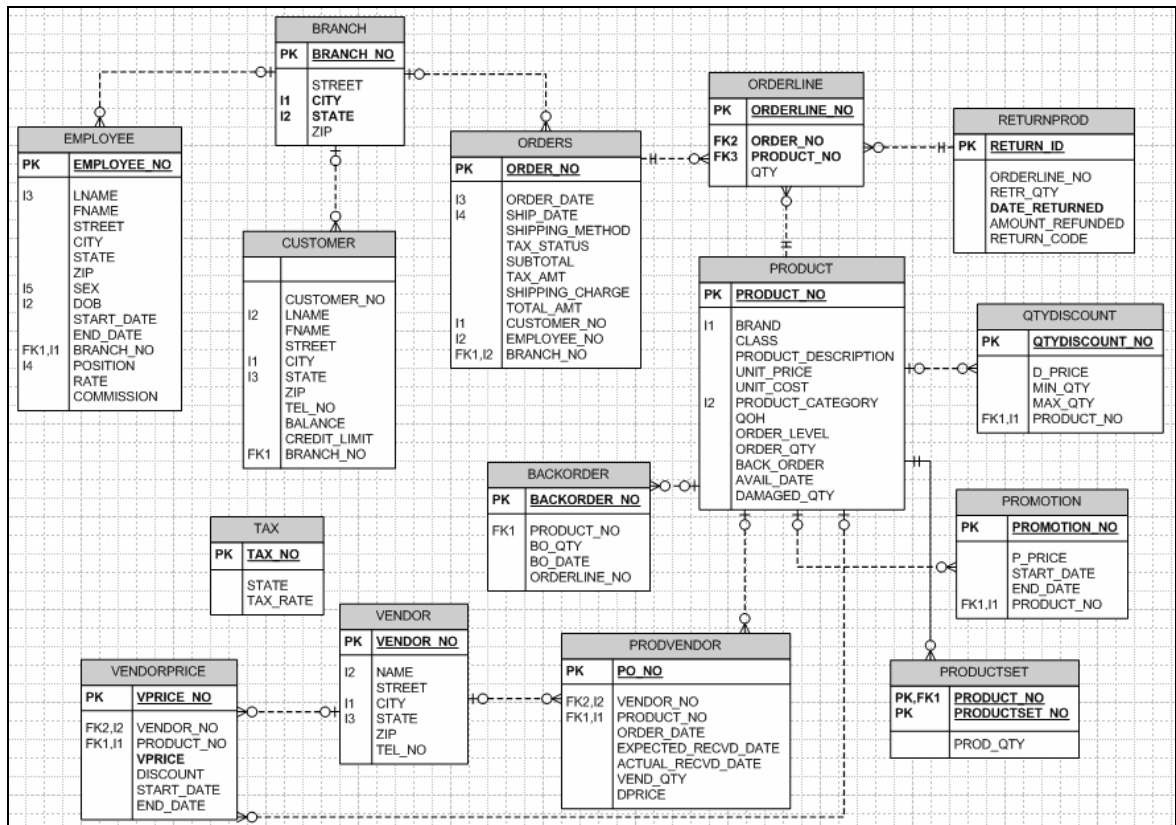


Figure 2: OES Data Model



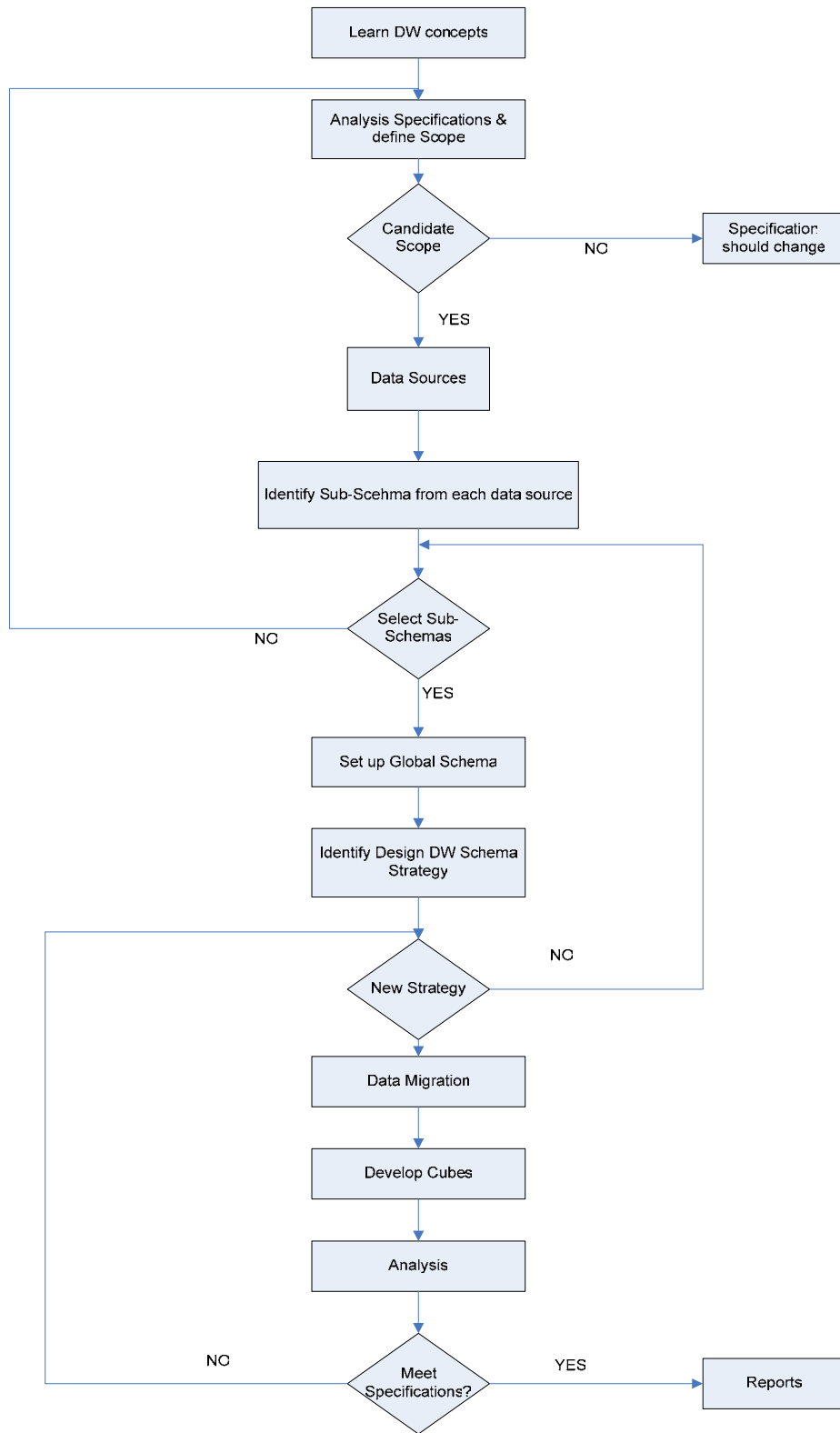


Figure 3: Flow Chart of Proposed Methodology

## **Brief discussion of the steps involved in the proposed methodology**

Step 1: Teach the Main Concept of DW, Introduction to Microsoft Analysis Services.

Step 2 & 3: Identify and understand the specifications and scope. For the OES project, specifications derived from OES database were:

- ✓ Analyze all the sales by products, customers, and employees that were made in different geographical areas
- ✓ Following Queries were developed after examining the specification
  - What products are selling?
  - What products are not selling?
  - In different states what is the trend Customers that means is there increase of customers in that area or are we losing them
  - What is the progress of Employees?
  - How many sales Employees have made?

Step 4: Analyze the participating operational databases. All tables of OES databases were stored in Oracle database. In order to eliminate distributed query processing, it was decided to bring in all the data from Oracle into SQL Server.

Step 5: Identify related sub-schema of individual databases: It was decided to use the whole OES schema in our DW design.

Step 6: Perform Data Cleaning / Filtering Tasks: Since there was no missing data in OES and there was no other database therefore there was no need to clean the database. It was also decided to bring all rows for each table and no filtering was needed.

Step 7: Identify Global Schema. As OES schema was from one database in Oracle so there was no need of creating a global database.

Step 8: Identify the Entities required to solve the problem

- ✓ DIM\_PRODUCT
- ✓ DIM\_CUSTOMER
- ✓ DIM\_EMPLOYEE
- ✓ FACT\_ORDER\_DETAIL ( provides consolidated view of data from underlying tables FACT\_ORDER and FACT\_ORDERLINE)

Step 9: Design DW schema. From the previously identified entities, the Star DW schema was designed (Figure 4).

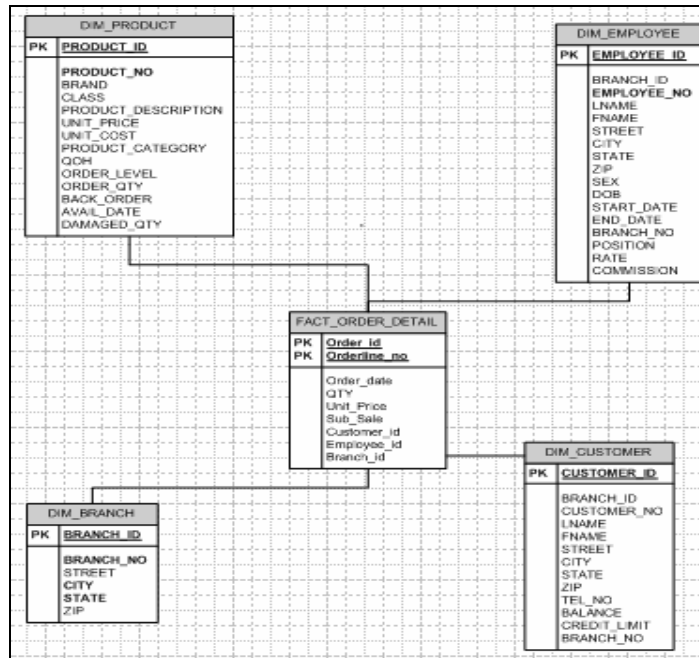


Figure 4: OES Star Schema

Step 10: Data Migration to Staging environment is another very critical step of implementing DW project. Staging Environment is used

- ✓ To secure Production system
- ✓ Avoid extra load and effect performance of Production systems
- ✓ To Keep Historical Data as opposed to Production that only keeps current state of the data

Microsoft Data Transformation Services (DTS) is used to bring the data from OES production database in Oracle to SQL Server and from there to OES\_DW staging database (Figure 5).

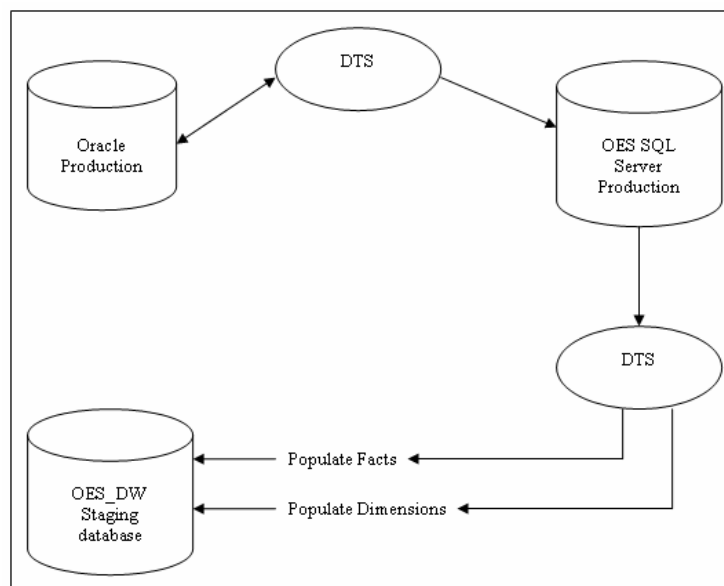


Figure5: Migration of data from OES Oracle database to OES\_DW Staging database

Step 11: Develop cubes by administrator. This step involves the OLAP administrator to implement the DW and develop cubes. The cubes were built by the following steps

- ✓ Analyze the staging database and design of DW to verify all components are available to create the Cubes. Possible problems can be:
  - What is on Staging database that's not in DW design
  - What is in DW design that's not in Staging database
  - Or any other obvious error
- ✓ Implement the design by creating Cubes and Dimensions.
- ✓ Give Read Only Access to the Users

Step 12: Analysis of implementation. This is a very important step in Life Cycle of DW design. This step involves initial evaluation and analysis of the report generated to find out any issue or missing information in the DW.

Cube Browser feature of Analysis Services was used to generate report and browse the data. Following Report (Figure 6) was generated that shows the Product sales by Employees.

					MeasuresLevel	
- State	- City	+ Fname	- Product Category	+ Brand	Sub Sale	
All Employee	All Employee Total	All Employee Total	All Product	All Product Total	1,341,075.00	
			- Game	Game Total	1,341,075.00	
				+ Microsoft	1,284,425.00	
				+ Samsung	56,650.00	
				+ Sony		
+	Total	Total	All Product	All Product Total		
			- Game	Game Total		
				+ Microsoft		
				+ Samsung		
				+ Sony		
- MN	MN Total	MN Total	All Product	All Product Total	1,341,075.00	
			- Game	Game Total	1,341,075.00	
				+ Microsoft	1,284,425.00	
				+ Samsung	56,650.00	
				+ Sony		
	- Minneapolis	Minneapolis Total	Minneapolis Total	All Product	All Product Total	1,341,075.00
				- Game	Game Total	1,341,075.00
					+ Microsoft	1,284,425.00
					+ Samsung	56,650.00
					+ Sony	
		+ Jake	All Product	All Product Total	1,341,075.00	
			- Game	Game Total	1,341,075.00	
				+ Microsoft	1,284,425.00	
				+ Samsung	56,650.00	
				+ Sony		

Figure 6: Report showing Product sales by Employees

**Step 13:** If proposed design does not meet the specification go back to step 9 (Design DW schema). In our implementation the initial analysis revealed that there is no information about time that when these sales were made? So it brought the need to change the design of DW and introduced the need for the time dimension. Process was repeated and the DW Schema was re-designed.

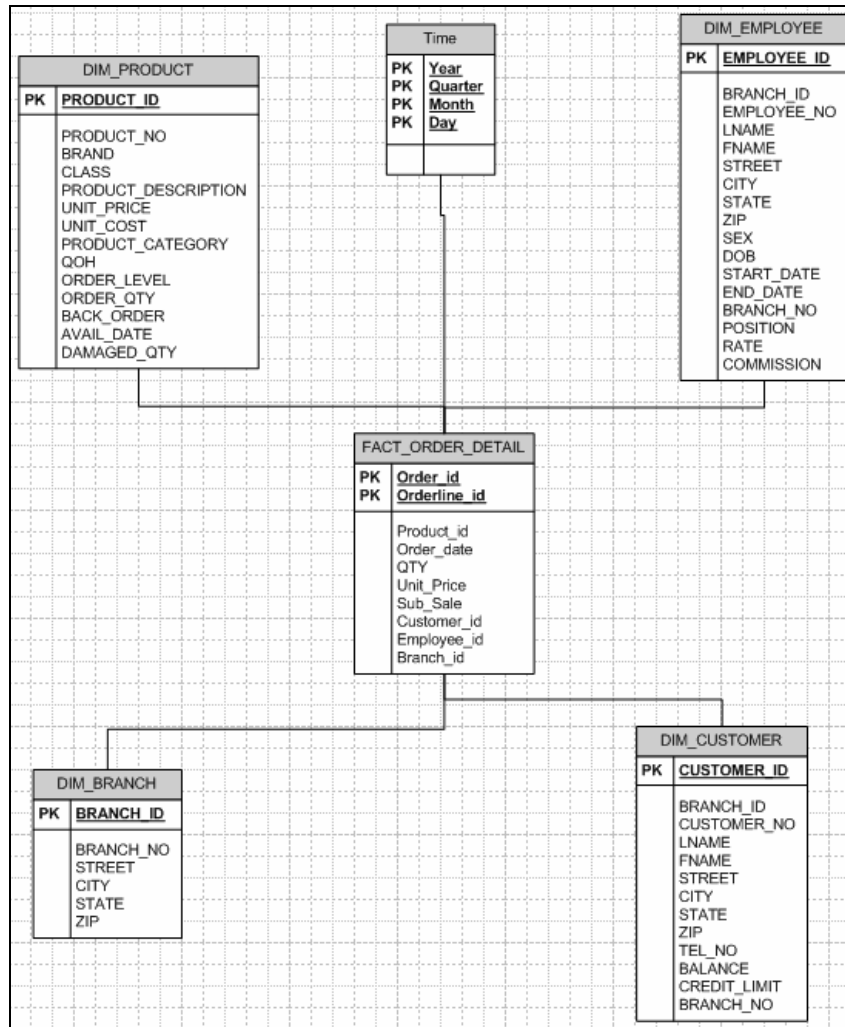


Figure 7: Implemented DW design

Figure 7 shows the revised star schema model. Following report (Figure 8) was generated based on new DW design using Cube browser feature of Analysis Services to do the initial analysis.

Report from Figure 7 reveals that Microsoft brand were sold mostly in 2nd Quarter of 2003. It also revealed that Samsung Brand was mostly sold during 1<sup>st</sup> and 2nd Quarter of 2003. There was no sale for Sony products.

**Step 14:** If objectives still does not meet then go back to he step Identify related sub-schema of individual databases (step 4)

+ State	- Product Category	+ Brand	- Year	+ Quarter	MeasuresLevel
					Sub Sale
All Customer	All Product	All Product Total	All Time	All Time Total	1,341,075.00
			- 2003	2003 Total	1,341,075.00
				+ Quarter 1	177,600.00
				+ Quarter 2	788,825.00
			+ Quarter 3	374,650.00	
			- Game	Game Total	All Time
	- 2003	2003 Total			1,341,075.00
		+ Quarter 1			177,600.00
		+ Quarter 2			788,825.00
	+ Quarter 3	374,650.00			
	- Game	+ Microsoft			All Time
			- 2003	2003 Total	1,284,425.00
				+ Quarter 1	176,350.00
		+ Quarter 2	733,425.00		
		+ Quarter 3	374,650.00		
+ Samsung		All Time	All Time Total	56,650.00	
		- 2003	2003 Total	56,650.00	
			+ Quarter 1	1,250.00	
+ Quarter 2		55,400.00			
+ Quarter 3					
+ Sony	All Time	All Time Total			
	- 2003	2003 Total			
		+ Quarter 1			
+ Quarter 2					
+ Quarter 3					
+ AK	All Product	All Product Total	All Time	All Time Total	6,100.00
			- 2003	2003 Total	6,100.00
				+ Quarter 1	
				+ Quarter 2	6,100.00
			+ Quarter 3		
			- Game	Game Total	All Time
	- 2003	2003 Total			6,100.00
		+ Quarter 1			
		+ Quarter 2			6,100.00
	+ Quarter 3				
	+ Microsoft	All Time			All Time Total
		- 2003	2003 Total	6,000.00	
			+ Quarter 1		
	+ Quarter 2	6,000.00			
	+ Quarter 3				
+ Samsung	All Time	All Time Total	100.00		
	- 2003	2003 Total	100.00		
		+ Quarter 1			
+ Quarter 2	100.00				
+ Quarter 3					
+ Sony	All Time	All Time Total			
	- 2003	2003 Total			
		+ Quarter 1			
+ Quarter 2					
+ Quarter 3					

Figure 8: Report showing Product purchase by customers in 2003

Step 15: If the objectives still does not meet then go back to reviewing the Specification (step 2)

**Step 16:** Develop appropriate reports and evaluations. Once the initial analysis shows that the specifications has been met. Final reports will be generated and evaluations will be done. We used Microsoft Office Excel to generate reports. Figure 9 and 10 shows two reports developed with Microsoft excel component of Microsoft Analysis Services.

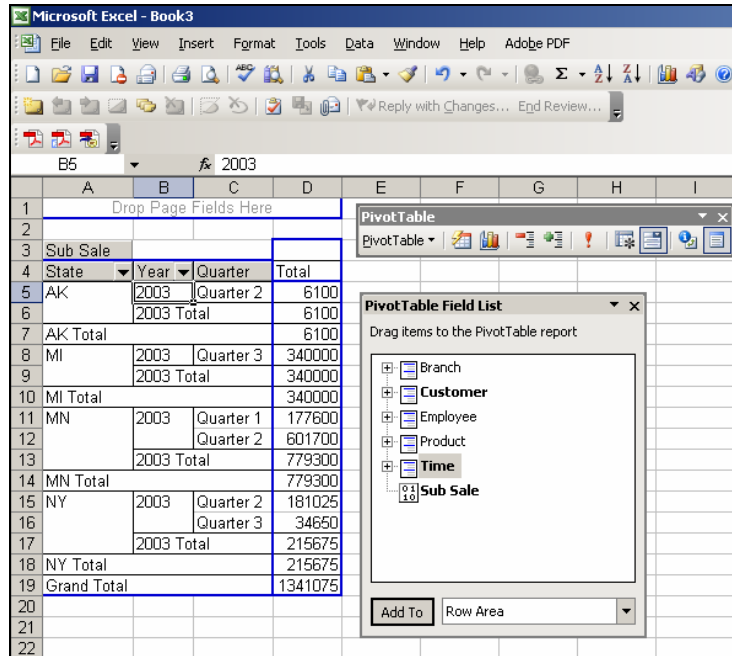


Figure9: Report showing trends in different quarters of year

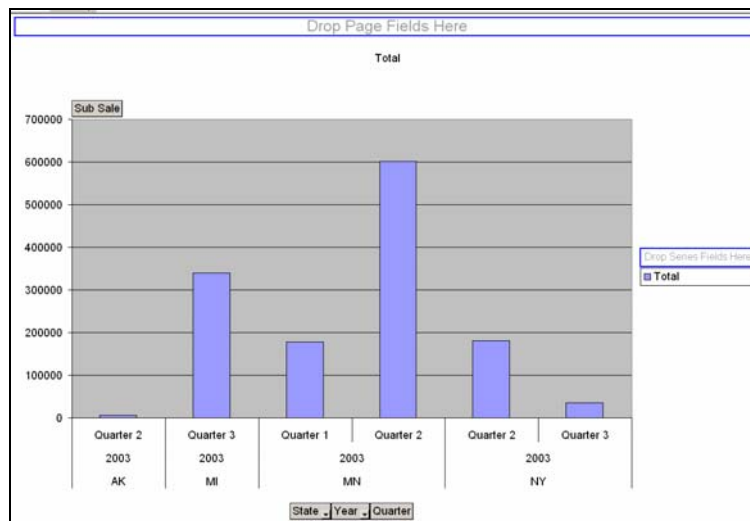


Figure 10: Visualization of trends in different quarters of year

## Conclusion

Based on the literature review a curriculum for a project based data warehousing course using Microsoft Analysis Services was proposed. An implementation of proposed methodology was also presented by developing OES\_DW data warehouse system. However, the implementation did not emphasize on heterogeneous database environment, also no data cleaning / filtering was performed. Future work, a follow up study should focus on heterogeneous environment and data cleaning / filtering tasks should be performed based on the proposed methodology

## References

- [1] Stenmark, Dick, "*Information vs. Knowledge: The Role of Intranets in Knowledge Management*" in *Proceedings of the 35th Hawaii International Conference on System Sciences*, 2002.
- [2] Ponniah, Paulraj, *Data Warehousing Fundamentals*. New York: Wiley, 2001.
- [3] Workman, M. *Expert decision support system use, disuse, and misuse: A study using the theory of planned behavior*. *Computers in Human Behavior* 21: 211-231. 2005.
- [4] S. Chaudhuri and U. Dayal, "*An Overview of Data Warehousing and OLAP Technology*," *SIGMOD Record*, Vol. 26, No. 1, 1997, pp. 65-74.
- [5] Kimball, R., *The Data Warehouse Toolkit*, New York: Wiley, 1996.
- [6] W.H. Inmon and C. Kelley, *Rdb/VMS: Developing the Data Warehouse*, QED Publishing Group, Boston, Massachusetts, 1993.
- [7] Bogdan D. Czejdo, Johann Eder, Tadeusz Morzy, Robert Wrembel: *Design of a Data Warehouse over Object-Oriented and Dynamically Evolving Data Sources*. DEXA Workshop 2001: 128-132.
- [8] Adamson, Christopher, and Michael Venerable, *Data Warehouse Design Solutions*, New York: Wiley, 1998.
- [9] Singh, Harry, *Interactive Data Warehousing*, Upper Saddle River, NJ: Prentice Hall, 1999.
- [10] Microsoft Corporation. *The Microsoft Data Warehousing Strategy*. MSDN Library. Retrieved July 21, 2006,
- [11] E.F. Codd, "*Providing OLAP (on-line analytical processing) to user-analysts: an IT mandate*". Technical report, E.F. Codd and Associates, 1993.



# An Experience in Teaching a Short Summer Robotics Course for High School Students

Andy Lopez  
Computer Science,  
University of Minnesota, Morris  
600 East 4<sup>th</sup> St, Morris, MN 56267  
[alopez@morris.umn.edu](mailto:alopez@morris.umn.edu)

Elena Machkasova  
Computer Science  
University of Minnesota, Morris  
600 East 4<sup>th</sup> St, Morris, MN 56267  
[elenam@morris.umn.edu](mailto:elenam@morris.umn.edu)

## Abstract

In this paper we describe our experience in teaching two sections of a robotics short course to gifted high school students during summer 2007. We used Lego RCX 2.0 and NXT robots and the Lego MINDSTORMS software, Inventions 2.0 for the RCX 2.0 and the NXT 1.1 for the NXT robots. We chose to break the two sections by gender. The students had very diverse backgrounds and levels of interest. Having two instructors share the responsibilities for the class was very helpful. A collaborative group-based approach to learning was used. Most of the students successfully completed their group projects by the end of the second week.

## Introduction

This class was designed and first offered in June 2007 during the 34<sup>th</sup> Annual Henjum Creative Study Institute for high school students. This program is aimed at challenging very capable 12 to 17 year old high school students in the arts, music and technology for two weeks at the University of Minnesota, Morris. The institute was the creation of Dr Arnold Henjum, a retired faculty member in secondary education at the University of Minnesota, Morris.

This past summer the students had the option of taking up to three classes of approximately 2 hours each for the two weeks. The students in our classes ranged in age from 12 to 18 years old. The exact distribution appears in Figure 1. Because of a conjecture [1] that female students may be intimidated by male students in technology classes, we requested that separate sections be created for male and female students. The male section of the class had twelve students and the female section had five students.

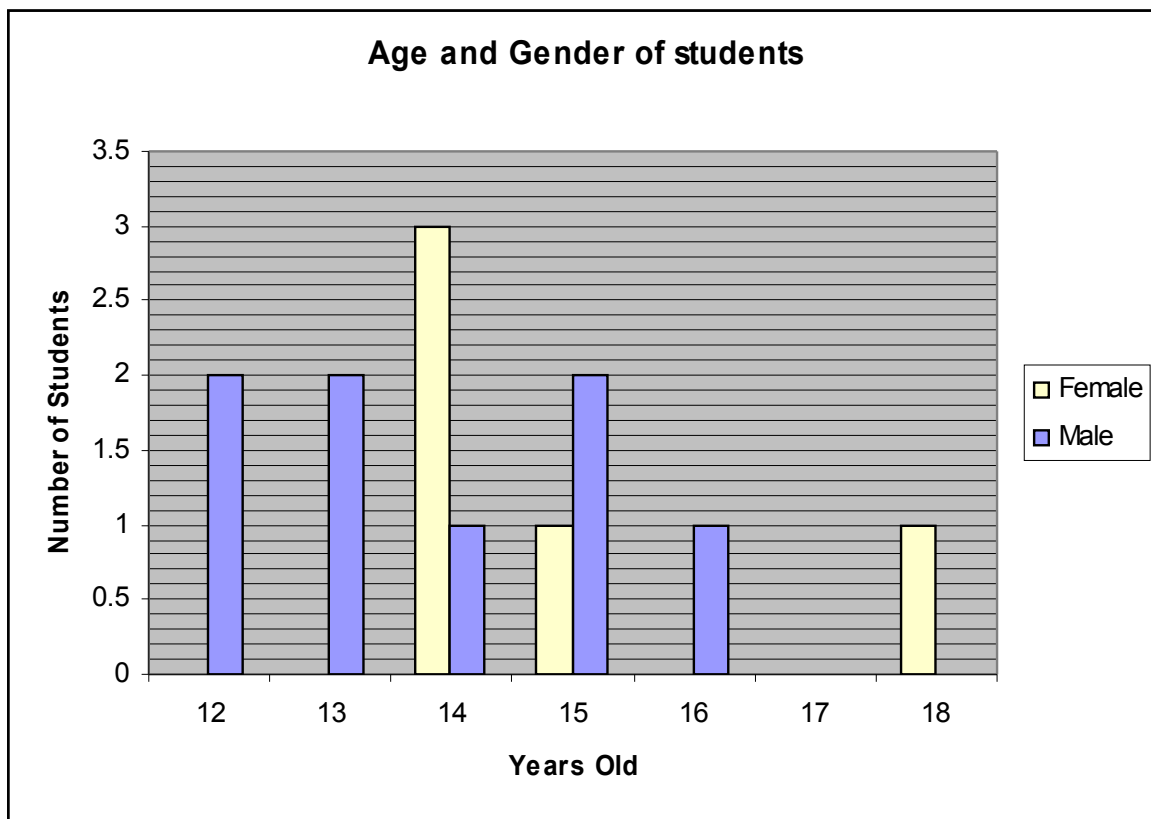


Figure 1 Age and Gender of the Students in the Robotics class Summer 2007

As the chart illustrates, we had over twice as many males students as female students. The age and background difference among the male students was very noticeable. The majority of the younger male students were more interested in building different types of robots than programming them.

Another challenge for our classes was what type of robots to use. The college had a couple of Lego RCX 2.0 that were used in a college class a couple of years earlier and a newer Lego NXT robot that a faculty member had secured with faculty initiation funds. Upon the advice of Elizabeth Jensen who had run a one week robotics camp for the previous three years in Pennsylvania, we decided to adopt the Lego Invention 2.0 software running on RCX 2.0 robots. Another two RCX 2.0 robots were purchased. Sensing that five robots may not be enough for a class of 12 students and due to the lack of affordable RCX 2.0 robots in the market place, we caved in and also purchased one additional Lego NXT robot.

Both the Lego Invention 2.0 software and the Lego Mindstorms NXT software provide drag-and-drop graphical interfaces in which users build programs by placing blocks into a work area, adjusting their parameters (if any) by drop-down menus, and connecting them by drawing “wires” between the blocks. However, the interfaces have substantial differences in the types of available building blocks and the way programs are built [2,3]. We discuss the major differences encountered where applicable.

## Course Material and Structure

### Introductory Steps

More than a half of students in the class were from Morris and knew each other from the high school; the others were from out of town and got acquainted during the introductory events of the Institute. Since students knew each other, we felt that the best way to divide students into teams would be based on their preferences. To accomplish this and to get to know students better, we gave students a questionnaire. In addition to questions about students’ interests and robotics experiences, we gave them the following question about their group partner preferences. The complete questionnaire is given in Appendix A.

*In this class students will be working in groups of 2 or 3. What are your preferences for working with others? You may choose more than one of the options below. We will try to follow your preferences, if possible.*

- *I can work with anyone*
- *I would prefer to work with someone who... (fill in your preferences, such as “is close to me in age”, “is close to me in the level of experience with robotics”, etc.)*
- *I would prefer to work with a person (or people) I already know (please write their names)*

*You may add other comments for your group preferences.*

Most students chose the last option for the answer to the question about working in groups and named 1-4 of others as their preferred group partners. It turned out that most preferences were mutual. We easily divided the male section of 12 into 4 groups of 3 each and the 5-student female section into two groups, of 3 and 2 students respectively.

The female class only used the NXT robots. Students mostly preferred to work with someone close to their own age, except in one case when a 10<sup>th</sup> grade student was “mentoring” two younger students. The team distribution by grades was:

Male section	7, 7, 7	8, 9, 9	6, 7, 10	8, 9, 9
Female section	8, 8, 9	10, 12		

However, one of the male teams of grades 8, 9, and 9 split into two groups since the eighth grader chose to work with his own NXT robot.

### **Lesson structure.**

Given a long class period time and the young age of students, we faced a challenge of keeping the students focused on the material. Therefore, for the first week, we divided lessons into a short (about 20 minutes) “lecture” time followed by exercises based on the lecture material. A lecture covered a particular topic, such as choices (if/else statements) or use of robot sensors. The very first class also involved introductions (each instructor and each student had to say a few words about themselves) and an interactive discussion of robotics – what is it, where it is used, references to robots in books and movies, and similar things. The goal was to get to know students better and also to make them comfortable in the class. At the beginning of the class we also introduced rules for class behavior which included care in working with equipment and robotics parts. For instance, the rules required that students do not remove Lego pieces from their work-stations unless actually using them in a robot and test robots only in specially marked areas of the floor (and never on a table to avoid falls). We included instructions for downloading programs into robots (RCX robots had to be covered with a box during the transmission to avoid interference; NXT use USB ports) and for recharging batteries and saving battery energy. We also listed general classroom rules, such as not leaving the room without permission of an instructor. With a long class time and young audience, we found that clearly defining the rules was to our great benefit: after the first couple of slips, most students followed the rules without even thinking about them which made the class go very smoothly.

The classroom setting included three rows of desks with a projector and a whiteboard in the front that we used for lectures and demonstrations, a desk with extra robotic parts, such as sensors and gears that student could borrow for their project, a work-station for each group with an individual robot set and a computer, and a clear area of the floor covered with cardboard in the back of the classroom which was used for robot testing. As the class progressed, we also added wooden blocks and large heavy books to the testing area which were used as obstacles or barriers for robots (these objects had to be heavy enough so that a robot would not move them when it bumps into them). Sharing a common area for robot testing led to more collaborative behavior as students watching other groups often exchanged ideas and suggestions.

## **Material covered.**

In both groups we covered approximately the same areas and concepts:

- Basics of building robots: connecting moving parts to motors, connecting sensors to ports, following instructions for robot building (especially with NXT) and developing one's own design (especially with RCX).
- Basic robot movements and turns; for NXT – controlling a robotic arm or a “stinger” in the scorpion model.
- Basic robot output devices (sound).
- Use of sensors in robots and the ability to use the input from the sensors to determine the robot's behavior.
- Calibrating sensors based on environment measurements.
- Use of variables in programs.
- Using predefined functions.
- Use of conditionals in programs.
- Loops for repeating an action a given number of times and for continuing an action until a certain condition is met; nesting of loops and conditionals.
- Using a random number generator to create “random” robot behavior.
- Defining functions (blocks in RCX software) to “package” repetitive or long parts of a program.
- Multi-threading – covered partially for both RCX and NXT groups.

The lecture material was somewhat different for the two sections due to differences between the RCX and NXT robot models: RCX has only basic sensors and no good way of detecting sounds or measuring distances. However, it provides a very intuitive and easy to use programming interface with a lot of helpful built-in functions, such as a versatile random number generator, and Lego pieces are very easy to combine to build new kinds of robots. NXT makes it easier to detect sounds and has built-in recordings of music and phrases. It also allows one to program in terms of distances and turns, rather than by the timing of motor movement. However, we found that building with NXT is too rigid (it is very difficult to build a new robot not in the instructions, and even following instructions is often challenging) and we found its programming interface to be quite confusing, even for some tasks that we expected to be simple.

## **Initial Building and Programming Assignments**

Given these discrepancies, we tailored our lectures and exercises (which we called “challenges”) to each of the two groups separately. With the RCX we started by using very basic sensors, such as light sensors and touch sensors, and used them as inputs for loop or if/else blocks. For instance, one of the first challenges was to have the students make a robot turn around when a certain condition has occurred. For the RCX group, the condition was to touch a wall (or an obstacle) with a touch sensor. The NXT group did a similar challenge, but the robot was supposed to turn around when it hears a sound, such as clapping. On the programming side, we spent more time with the RCX group on

programming robot turning. With the NXT robots, loops and using the random number generator turned out to be very challenging. The problem with loops was that the condition was only checked at the end of the loop, so if the loop was supposed to continue until a clapping sound is heard, the robot could miss the sound if it happened when the program was in the middle of the loop's execution. The random number generator in NXT needs to be connected via the right kind of an input in a program block which was quite difficult to figure out. There are different kinds of inputs, such as an integer or a Boolean, which presumably correspond to the respective types in the underlying programming language. While the type system would make sense for college-level students who have experience with a higher-level language, it is very difficult to understand for high-school students and seems out of place in a drag-and-drop interface.

We designed challenges in such a way that they would motivate students to focus on programming rather than just on building robots. Some of the more advanced challenges included: detecting and following a black line, finding an object within a given area using random moves, going around obstacles, and finding an exit in a maze. Not all groups were able to accomplish all of the tasks, but all groups tried and worked on their necessary robot-building and programming skills and achieved different levels of success. For instance, all groups in the male section were able to write a program for following a line in simple cases (slightly curving line), but less than a half of the groups were able to extend the program to follow sharper line turns and to try to find the line once it was lost. Due to issues with NXT construction and programming and less motivation in the female groups, they accomplished less than the male ones in terms of challenges. However, they achieved more success on robots following sound signals and detecting and following an object – tasks that are easier to accomplish with NXT robots.

## **Working on final projects**

Early in the class we announced that during the second week of classes each group will work on a project of their own design. All projects were to be demonstrated to parents and guests during an open hour on the last day of the Institute, and the best projects were to be shown on stage during the final all-Institute presentation. This motivated at least some of the students to pay closer attention to the lecture material and examples since they were trying to estimate which of those would be useful for their projects. In fact, students included extended versions of some of the given challenges as a part of their final projects (for instance, finding an exit in a maze).

As the classes progressed and students started focusing more on the projects, the lectures were no longer needed since students became motivated by a variety of individual project-related goals. At that point the role of instructors became more consultative, although we also occasionally needed to keep students from being distracted by things unrelated to the class. We were also trying to challenge the students to include more features into their projects.

## Final Projects and Observations

The level of difficulty of final projects varied greatly between groups. For instance, we observed that younger students and female students tend to focus more on sounds and appearance, whereas the middle (8-9<sup>th</sup> grade) and older (grades 10-12<sup>th</sup>) males picked more challenging programming or engineering tasks. Since the final projects demonstrate students' level of accomplishments and their interests, we discuss them in detail:

- An RCX robot looking for an exit in a maze. This was the most advanced project from the standpoint of programming. We were very impressed by extensive use of user-defined functions in the program. The students used infrared sensors to measure distances to walls. Unfortunately, despite hours of adjustment attempts, the sensors did not have enough precision to be able to detect which of the two walls (the left or the right) was closer. Therefore the robot relied on precision of turns and moved in a straight line until finding the next wall. Since turns could not be perfectly calculated, the robot would accumulate small mistakes after the first 2-3 turns, and occasionally would hit a wall with its side and get stuck. Despite these mechanical problems, the students did an excellent programming job. Their robot was able to navigate simple mazes successfully more than a half of the time. This work (by two male 9 graders) was exactly the kind of project we were hoping for. For those teaching robotics classes with RCX robots, we recommend investing in a set of high-quality sensors, if you can find them.
- An RCX robot climbing stairs. This group (boys of grades 8, 9, 9) focused on the engineering side of the project. Calibrating the robot's speed when it is climbing stairs turned out to be a challenging programming task. The students modified their design of the robot several times. In particular, after many attempts, the group abandoned the idea of the robot pushing itself up by a long Lego piece at its front and settled for a simpler design. The group overcame many challenges, both in engineering and in group communications, and came up with a very successful project.
- Another advanced project was an NXT robot built by a male 8-grader who used his own NXT kit. He used the scorpion version of the robot and wrote a program that made a robot go around in a walled area that had upside-down plastic cups placed in random positions. The robot was making random moves looking for cups, moving toward one when it was detected, hitting it with its "stinger" and then turning away from it to look for more cups. The program exploited NXT strengths, such as detecting objects. However, programming the robot to make random moves was a substantial challenge. Adjustments also needed to be made with respect to the force with which the robot hits the cups (so that it doesn't actually break them) and to prevent the robot from getting stuck in corners. Overall, it was a very challenging project from the programming standpoint and a very successful one.
- Another male group came up with a project that was not very challenging in a programming sense but showed their creativity. They came up with a pair of robots: a simple random movement robot used for massaging a person's back and a human-looking robot in a hat that was greeting people and waiving its "hand". The group was making slower progress than the more advanced groups in the

challenges and put a lot of effort into trying to keep up. They put their project together in the last couple of days so we feel that they had a good learning experience in the class and also got a chance to show their creativity.

- The group of male students in 7<sup>th</sup> grade encountered several challenges, mostly stemming from the fact that they were not able to adjust their designs based on testing feedback. For instance, their robot was not sturdy enough, and despite multiple tests in which it broke and numerous suggestions from instructors and fellow students to use more robust design, they kept using the problematic one. They programmed the robot to play music which involved a lot of repetitive programming, and did not follow the advice to use functions. The group also had a lot of communication problems, with two group members not very well motivated. The final project for that group was a music-playing robot which worked well but was not very sophisticated. It is possible that our lesson style worked better for older students and we needed to come up with somewhat different approaches for younger ones. Additionally it may be a good idea to have a tryout time for group work (for instance, the first two days) and if the group dynamics seem to be problematic, students may be switched at that point.
- Students in the afternoon (female) section, unfortunately, were not as invested in the projects as those in the male student section. While a part of the reason was the use of more complex NXT robots, another problem was lack of motivation. Building some of the NXT models or even following instructions precisely, is not an easy task. The students were interested in building a human-like figure (one of the possible robot configurations). Unfortunately, the instructions are so unclear that even with substantial participation of both instructors during a two-hour building session the resulting robot failed to walk properly. As a result, the students built a simpler moving robot that reacted to loud sounds by turning. This was not a very challenging project, but, given the circumstances, we were glad to see a finished project. It was entertaining, however, and the audience at the showcase performance liked it.

Overall, the projects let students use more a goal-oriented personalized learning approach which worked for most groups. Students developed problem-solving skills, learned to adjust their work based on real-life tests, worked on their communication skills in groups and in seeking help from instructors, and got a chance to approach the projects creatively. We found that in the all-male section, we successfully developed a collaborative environment that included all students and both instructors. For instance, in every project there were times when someone not in the group, often another student, worked with the group members on a tricky hardware problem or on a programming challenge. We also felt that the ability to be creative and artistic was important for many students and contributed quite a bit to the success of the class. Unfortunately, these strategies brought little success in the all-female section. The reasons for it, as we pointed out above, were not the gender of the students, but the smaller size of the group, the initial lack of enthusiasm for the subject of the class (only one female student chose this class because of her interest in robotics), and the learning curve for building NXT robots that turned out to be steeper than for the RCX model.



## Conclusions

The students seemed to have enjoyed the classes. The evaluations contained statements like 'I liked the part where we got to try to overcome challenges by not only changing the program but the robot as well', 'The robots were pretty nifty' and 'Robots was my favorite' (of three classes that each student took). The students also seemed to be satisfied with the instructors (the authors of this paper) when they said 'They were very good and know what they are talking about' and 'Even though they know a lot more than us, they speak a language we can understand'. When asked whether the instructors were able to answer their questions, one student stated 'Yes, except when I had a question that not even the NXT forum could answer, but we found a different way around it'.

One important question to ask is, 'what did the students learn?'. In terms of learning the material, all students were able to build and program basic robots. All of the students demonstrated working knowledge of the programming concepts, such as simple programming blocks, conditionals and loops. At least half of the students successfully used more sophisticated sensors and more complex programming concepts, such as variables (not necessarily required in a drag-and-drop interface, and thus considered a more advanced concept) and functions.

The purpose of the class, however, was not just to teach robot-building and programming skills. Our more general goal was to teach approaches needed for successful work with modern technologies. The authors would argue that the students learned that constructing and programming robots takes creativity, patience and endurance. At times it was hard to convince the students that there was more than one way to accomplish a task. Toward the end of the two week period, we found the students more willing to go to the board and discuss their solutions prior to coding them. Some of the solutions implemented by the students were very creative, like having the robot play sounds that simulated a particular musical piece or trying to get a robot to 'dance' to a particular tune or having the robot listen for sounds and change course on the basis of the sound received.

We also feel that in the larger, more motivated male class, we successfully created a collaborative learning environment where solutions to problems that a group would run into were found by productive discussion both within the group and with other students and instructors. This taught students important skills in dealing with others in a goal-oriented environment. Unfortunately, the collaborative approach was less successful in a smaller and less motivated female class, although there were many productive discussions in that class as well, mostly within groups or between individual students and instructors.

Another important question would be: did the gender split work? This is a difficult question to answer. The female class accomplished considerably less than the male class. On the other hand there were substantial differences between the two classes, the female class had a higher incident of student absences, a higher proportion of 'home schooled' children and appeared to have less experience with devices like robots. Overall, the authors feel that the female class was a success because they were presented with the task

of building and programming their own robots and had to respond to the challenges themselves which may not have happened in a coed class.

The instructors did face a couple of other challenges. First there was a very significant age and experience difference among the members of the male class. The younger students had more difficulty building and programming their robots and remaining focused on their work. The differences in age and experience also required that the instructors create multiple assignments that were suitable to the age and ability of each subgroup of students.

Another challenge was the difference in hardware and software. Because we were using two substantially different types of hardware and software, it was hard to provide instruction and assignments that would apply equally to all students. On the other hand, having different hardware and software allowed us to demonstrate a principle easier on one of the platforms than the other.

In conclusion, the students learned a reasonable number of principles about robotics and the dynamics of working with technology. The instructors learned how challenging it is to create suitable activities for students of different ages and academic backgrounds, and how satisfying it is to watch students create their own inventions.

## References

- 1) Gender, Perceptions and Reality: Technological Literacy Among First-year Students, Madigan, E; Goodfellow, M.; Stone, J.; ACM SIGCSE Bulletin, Vol 39, No 1, March 2007
- 2) How Lego MINDSTORMS NXT Works, <http://www.ni.com/academic/mindstorms/works.htm>
- 3) Introduction to Lego MINDSTORMS Robotics Invention System 2.0, [http://badlink.com/lego\\_mindstorms/index.htm](http://badlink.com/lego_mindstorms/index.htm)

## Appendix A

Questionnaire administered the first day of class

What is your name?
How do you prefer to be called (if different from your name in the class list)?
What is your home town and the high school?
What grade will you be in next school year?
What are your favorite subjects in school?
What activities do you participate in besides school?

We don't expect you to be familiar with robotics, but if you have any experience, it would be helpful for us to know. Have you done any robotics before? If yes, what system(s) did you use and for how long?

In this class students will be working in groups of 2 or 3. What are your preferences for working with others? You may choose more than one of the options below. We will try to follow your preferences, if possible.

- I can work with anyone
- I would prefer to work with someone who... (fill in your preferences, such as “is close to me in age”, “is close to me in the level of experience with robotics”, etc.)
- I would prefer to work with a person (or people) I already know (please write their names)

You may add other comments for your group preferences.

# Lightweight Software Cost Estimation Model

Izzat Alsmadi  
Department of computer science  
North Dakota state university  
1353 N University Dr  
Fargo, ND 58102  
[izzat.alsmadi@ndsu.edu](mailto:izzat.alsmadi@ndsu.edu)

## Abstract

There are several software cost estimation models available in the software research and industry fields. One of the problems that prevent many companies from using such models for cost, size or effort estimation is the amount of required inputs for those models. In additions, the nature of the required inputs is complex to quantize in numbers or even scales.

The calculated cost is highly depending on those inputs and in many cases such inputs require judgments and not calculations which may put the whole results in jeopardy or questioning.

This paper suggests a lightweight cost estimation model that is intended to be practical and feasible through selecting some parameters or metrics that can be, relatively, easily gathered and that is not human or expert dependent.

## Keywords

Prediction model, software cost estimation, reverse engineering, software metrics, and analogy.

## 1. Background

How can we estimate the amount of resources required to build a software project or a specific release? Do we measure effort depending on previous projects or requirements only? How can we normalize requirements into categories that may require the same amount of resources!?

Using analogy and prediction models, we can compare projects in terms of project resources, complexity, and size. Many data mining problems can be transformed to

prediction models. For example, credit scoring tries to assess the credit risk of a new customer. This can be transformed to a classification problem by creating two classes, good and bad customers (i.e. classify customers to decide whether they should be given credit or not). A classification model can be generated from existing customer data and their credit behavior. Large amount of information is used as input to the model. This classification model can then be used to assign a new potential customer to one of the two classes [2]. Similar to data mining goals, we use the information gathered of several earlier actual data to come up with a single decision whether to allow or reject this person credit application. The predictive validity is the capability of the model to predict the future component behavior from present and past behavior [11].

## 2. Related work

There are several papers, research projects and tools in the field of software predictive models. Those are trying to predict some characteristics of future software projects such as quality, complexity, fault prone modules, or cost estimation.

There are a lot of software cost estimation models (such as COCOMO and COCOTS) that are often used to estimate the cost or effort for a software project. The project managers gather information as input to such models. They may have challenges in trying to define some of the required criteria such as total lines of code, function points, .etc. In COCOMO, Source Lines Of Codes (SLOC) metric is calculated from historical data, or expert opinions. [13].

Gathering information from earlier projects, using software metrics, can be used as an input for those cost estimation models.

Boehm classified several ways to estimate the effort of a software project [12]:

- **Algorithmic Models** use one or more algorithm to calculate the effort to develop the software product with major cost drivers as variables. COCOMO is an example of this.
- **Expert Judgment** is when you ask one or more expert and they give a qualified "guesstimation" of the effort needed. The Delphi Technique is an example of this type of estimating method.
- **Analogy** can be used if you have plenty of data collected from earlier projects. The idea is to relate the actual cost of a completed project, to an estimate of the cost of a similar new project.
- **Parkinson's principle** is about estimating the effort to fill available resources.
- **Price to Win** is used when you estimate to the prize you believe will win the contract or to the deadline you believe is needed to be first on the market.
- **Top-Down** is when you estimate the effort of the entire system and then split it up between the different modules or components.
- **Bottom-Up** is then you estimate the effort of every module or component individually and then sum it up for the total effort.

In analogy, the estimation is based on actual project data. The estimator's past experience and knowledge can be used which is not easy to be quantified. We need to identify the

differences between the completed and the proposed project. However, we have to determine how best to describe projects. The choice of variables must be restricted to information that is available at the point that the prediction required. We also have to determine the similarity and how much certainty can we place in the analogies.

There are several papers presented in using metrics for software classification models. Jiang Yue et al explain software classification models as a way to detect fault prone models in a future software project [6]. They built defect prediction models using requirement metrics from unstructured data.

Taghi et al combined several software quality classification models. Those are predicting modules characteristics (such as fault prone or not) [7]. This is one area of software classification models. In general, classification models should be able to provide some other helpful information for management such as specifying the amount of resources required to build such module.

Other papers discuss software classification models in their structure, hierarchy and function characteristics [8].

Basili et al study measuring the development time or the human effort required for high performance computers [10]. The study introduced a “time to solution” metric in trying to calculate the time required for developers to find the proper solution for a particular problem as well as the amount of computer time required to execute that solution. Measuring the second part can be straightforward given that the first part is provided in a suitable way. Since the human factor is an important one in software projects, developing an accurate prediction model will always be challenged by how accurately we measure the human efforts and abilities. Dekhtyar et al study the effects of software predictive models in predicting human decisions toward certain software artifacts [11].

Function Point (FP) is another method or metric for cost estimation. The functionality of the software is divided into smaller manageable units (i.e. functions). The complexity of those FPs is estimated and then summed up representing general system and project characteristics.

### **3. Goals and approaches**

In order to make our suggested lightweight cost estimation model usable in situations where the available time is not enough to proceed on a rigorous cost estimation process, expert input parameters should not be more than 5 inputs. The mission is not a critical system and we want a model to be a tool and not a goal by itself. This will make it more practical and make the estimation less subjective or largely dependent on human judgments. Depending purely on software metrics is not a proper complete track. On the other hand, having many parameters to be evaluated by the users makes the estimation process easily manipulated.

Both metrics and expert judgments are going to be used as inputs to the model. The inputs to the model are: the requirements and code metrics from earlier projects (such as size and effort), requirement metrics from the current project and up to 5 experience

decisions in the current decisions (optional having similar decisions on earlier projects). The outputs of the model are the size and efforts of the current project.

In estimation by analogy, it is important to calculate the level of similarities between the current project and those in comparison. We have to compromise between being accurate in calculating similarities and not to spend too much time in drawing the similarities. The goal is to be able to make a good judgment of how similar the current project with those earlier ones. Tables and figures may not be very helpful for managers to make decisions. Prediction or classification models use extensive information at the bottom of the pyramid to be able to make micro decisions at the top levels. The judgment of the level of similarities can be largely depending on business or software experts.

Let's take the equation:  $\text{Level of similarity} = 100 \left[ \frac{\text{requirement A}}{\text{requirement B}} + 4 \times \text{experience judgment factor} \right] / 5$ , where A and B represent the current and previous projects respectively. The expert judgment factor is a value between 0 and 1 to indicate the difference between the current project and the earlier one(s).

Requirement metrics, such as function points, can be used to compare the similarity between projects. We should have this information available for the current project as well as those previous ones. If we know the time required to build software and have the list of tasks, features or modules implemented in the project, we can have an approximate picture of the development speed ( given that the project is going to be developed in the same company or environment). As explained earlier, cost estimation is not meant to be accurate (although it should be close enough).

Usually the time given for cost estimation is not long enough to process rigorous cost estimation techniques. Usually in project lifecycle, the further we go into the project, the closer our cost estimation becomes to its actual amount. We don't have to spend long time on something that will eventually be known, adding to that is the amount of the unknowns or the things that will eventually be changed; shrink or enlarge project cost.

Besides the need for expert judgment in the level of similarity between the current project and earlier ones, experts or managers needs to decide the level of knowledge developers team has, the project level of complexity and the environment. In COCOMO II model, there are 17 cost drivers divided into 4 categories: product, computer, personnel, and project. This research goal is to summarize each category drivers into one judged by experts. The original 17 drivers or the new 5 suggested ones are expected to help managers in cost, effort or defect estimation. For example, the 5 cost drivers in the product category are: Required Software Reliability, Database Size, Software Product Complexity, Required Reusability and Documentation match to life-cycle needs.

All of those values are going to be "subjectively" evaluated. This is not much different from having one factor that can be called product complexity (or productivity) evaluated by experts. The goal is not to spend long time trying to evaluate values that will be for most cases individually dependent. Same thing applies to combining all computer categories parameters (i.e. Execution Time Constraint, Main Storage Constraint, and Platform Volatility) into one parameter that can be called computer complexity.

Function points metric represents the list of functionalities in a software project. It is expected that this metric is available for both current and previous projects. The level of similarity can be assist through comparing function points.

## **Uncertainty and Cost estimation**

There are many factors in software projects such as scope, available tools, environment, product complexity, and/or productivity that can not be completely specified or known during the planning stage. It is also difficult to specify the amount of uncertainty in any of those areas. Typically in cost estimation, it is more realistic to specify a range for cost estimation parameters and not a value. Eventually the cost will converge to a value, hopefully, within the pre-specified range. For example, we can say that the expected cost of a certain software project is in the range of \$300,000 to \$500,000, the expected time is between 9 to 12 months. What is the best and most realistic range? We will be 100 % accurate if we set a very wide range ( i.e cost from \$500 to \$500,000,000, or time between 3 to 60 months). We will then loose the point of why we estimate (by having such a wide range).

## **4. Conclusion and future work**

This paper introduces a new approach for software cost estimation that can be practically used in the software industry. In most scenarios, companies do not have enough resources and knowledge to pursue some of the known rigorous cost estimation approaches. Future work will include using some available datasets to evaluate using the suggested lightweight cost estimation track. The information required to process cost estimation should not be complicated and largely subjective. The main goal of cost estimation is to provide a lightweight helpful asset to software projects managers.

Cost estimation is expected to be approximate, not exact or accurate. It is also expected to be a mean or a tool for project managers and not a goal by itself.

## **References**

- [1] Coates, Anthony and Miley Watts. A context model and methodology proposal for UCM. <<http://www.unstandards.org>>. Sep. 2007.
- [2] Chapman, Pete, Julian Clinton, Randy Kerber, Thomas Khabaza, Thomas Reinartz, Colin Shearer, and Rudiger Wirth. CRISP-DM-1.0. Step-by-step- data mining guide. SPSS Inc. <<http://www.crisp-dm.org/CRISPWP-0800.pdf>>. 2000.
- [3] Shepperd, Martin and C. Schofield, “Estimating Software Project Effort Using Analogies”, IEEE Transactions on Software Engineering. 1997.



- [4] Shepperd, Martin and Magne Jørgensen. A systematic review of software development cost estimation studies. IEEE transactions. VOL 33. No.1 2007.
- [5] Muller, Hausi, Jens Jahnke, Dennis Smith, Margaret-Anne Storey, Scott R. Tilley, and Kenny Wong. Reverse Engineering: A Roadmap. <<http://www.cs.ucl.ac.uk/staff/A.Finkelstein/fose/finalmuller.pdf>>.
- [6] Jiang, Yue, Bujan Cukic, and Tim Menzies. Fault Prediction using Early Lifecycle Data. ISSRE 2007.
- [7] Taghi M. Khoshgoftaar, Erik Geleyn, Laurent Nguyen, and Lofton Bullard. Cost-Sensitive Boosting In Software Quality Modeling. Proceedings of the 7th IEEE International Symposium on High Assurance Systems Engineering (HASE'02). 2002.
- [8] Wuyts, Roel and St'ephane Ducasse. Unanticipated Integration of Development Tools using the Classification Model. ESUG Smalltalk Conference. <<http://esug2003.esug.org/academic/5-wuyts.pdf>>. 2003.
- [9] Lanubile, Filippo, and Giuseppe Visaggio. Evaluating Empirical Models for the Detection of High-Risk Components: Some Lessons Learned. <http://www.di.uniba.it/~lanubile/papers/sew95.pdf>. Proc. of the Twentieth Annual Software Engineering Workshop. 1995.
- [10] V. Basili, S. Asgari, J. Carver, L. Hochstein, J. Hollingsworth, F. Shull, and M. Zelkowitz, "[A Pilot Study to Evaluate Development Effort for High Performance Computing](#)," University of Maryland, CS-TR-4588, April 2004.
- [11] Dekhtyar, Alex, Jane Huffman Hayes, and Jody Larsen. Make the Most of Your Time: How Should the Analyst Work with Automated Traceability Tools? PROMISE 2007. <<http://promisedata.org/pdf/mps2007DekhtyarHayesLarsen.pdf>. 2007>.
- [12] Boehm, Barry. Software Engineering Economics, Prentice Hall, 1981.
- [13] Boehm Barry, Chris Abts, A. Winsor Brown, Sunita Chulani, Bradford Clark, Ellis Horowitz, Ray Madachy, Donald Reifer, and Bert Steece. Software cost estimation with COCOMO II. 2000. Prentice hall.

# ISOMER – Augmenting Software Testing Confidence by Automated Comparison with a Lightweight Model

Darren Kulp and Daniel Ernst  
Department of Computer Science  
University of Wisconsin – Eau Claire  
Eau Claire, WI 54702  
{kulpdm,ernstdj}@uwec.edu

## **Abstract**

Non-algorithmic constraints on software development can hinder testing efforts by creating pitfalls for the programmer or by hiding data-dependent errors in complex codes. The ISOMER framework improves testing efficacy and confidence by further automating software component and program testing. Using a familiar expression syntax to define constraints on the random stimulus generated, ISOMER subjects software interfaces to dynamically-generated test cases, adding value over time.

# 1 Introduction

The development of any complex piece of software inevitably uncovers numerous faults. Unfortunately, not all bugs are caught in the time (if any) set aside for debugging; every bug that escapes to production reduces the value of the finished product and increases total costs. Bugs caught early are easier to repair than bugs caught late. Some types of bugs, particularly data-dependent bugs, resist detection, surfacing only in relatively rare cases which may never occur during normal testing.

Our system “ISOMER” helps find data-dependent errors in software programs or components by subjecting their input interfaces to random inputs constrained to thoroughly exercise their input space, and by comparing their output with that generated by lightweight models. Because the input is generated randomly but constrained to valid and “interesting” areas by programmer-supplied rules, the testing framework that results adds value over time; unlike normal “directed” unit tests, which test fixed input and output combinations, our testing system continually produces new tests and improves confidence in correctness and robustness.

In section 2, we will overview previous work and background information that bears on our design and implementation. Section 3 will describe the novel aspects of our design; section 4 will discuss implementation details thereof. Section 5 evaluates the goals of the system in spheres of functionality and performance, and section 6 concludes this document.

## 2 Background

### 2.1 Problem

In some development environments, there can be a significant expenditure of time by the programmer on issues stemming not from the problem to be solved but from constraints imposed by the environment. Heap management, for example, can be tedious and error-prone, but is an integral part of practically every production C / C++ program. Typical test cases may not account for unexpected behavior stemming from such errors, and even if a developer is aware of the possibility of such faults, it is difficult or impossible to design a static test or suite that will detect them. Of course, problems caused by heap management are only a possible manifestation of the more general phenomenon: data-dependent failures.

## 2.2 Use of random stimulus

Random stimulus as a basis for testing has precedent in both hardware design and software development; we mention hardware design specifically because it was the original inspiration for this project [2] and because its use in hardware is more mature and widespread.

### 2.2.1 In hardware design verification

ISOMER's premise of constrained-random inputs and lightweight models is not novel; it exists today in the domain of hardware design verification [2]. A lightweight model of the device or design under test ("DUT") is constructed, to which random inputs, constrained by declarative code to valid and/or "interesting" areas, are fed. The same stimuli are supplied to the real design, and the outputs are compared. Assuming the model perfectly implements the design's designed behavior and that the comparison code is perfect, a mismatch indicates an error in the DUT. The efficacy of the concept lies in the word "lightweight": the model is necessarily easier to construct than the DUT.

### 2.2.2 In software testing

This system is well suited to hardware testing, partly due to the stable, synchronous interfaces of hardware components, and partly due to the great amount of simplification that can be achieved by a model over the real implementation. In hardware design, a significant, sometimes enormous amount of effort is expended not on algorithmic complexity, but on time or space optimizations, since hardware's realization in silicon constrains it in ways unknown to software.

These techniques have not been as commonly applied to software, but solutions do exist under the name of "automatic specification-based testing" [3] [6]. See section 2.3 for more specific information on software precedents.

External constraints in hardware or software design can create their own problems, sometimes the most insidious and difficult to diagnose. In hardware design, external constraints include timing problems and physical implementation size; in software design, they include implementation language and library requirements. For example, when using C or C++, where it is necessary for the programmer to manage memory allocation and where failure to do so properly can result in hard program failure, it can be difficult to debug data-dependent errors, and it is especially vital to ensure that such bugs do not remain hidden, since error recovery at run-time is often poorly supported – segmentation faults are not user-friendly.

In such situations, a lightweight model implemented in a rapid-development language such as Perl or Python can be developed in parallel with its heavyweight equivalent to test correctness. This lightweight model might even act as a prototype, implemented before

the heavyweight version; in cases where this prototype stage already exists, adding the constrained-random-stimulus system represents a significant gain – greatly improved confidence in program correctness – at low cost – small amounts of glue code to harness the test system to the implementations.

## 2.3 Existing tools in the domain

As mentioned previously, tools attacking the problem addressed by ISOMER exist, but generally suffer from fundamental limitations including

- dependence on a particular language or platform
- restriction to synchronous function interfaces

The original inspiration for the ISOMER project was the Synopsys Verification Methodology Manual for SystemVerilog [2], which defines a technology and a methodology for subjecting hardware designs in hardware description languages like Verilog to interesting randomized inputs. In this system, the “model” we refer to is called a “scoreboard”, and includes the logic used to compare actual outputs with expected ones; in our design, these functions are separated.

In the software universe, this technique, automatic specification-based testing, is most preceded in functional environments. The code on which ISOMER is based, LectroTest [6], is itself written in Perl, which is not a specifically functional language, but it is based on and borrows heavily from an implementation called QuickCheck [3], whose website links to implementations in Erlang, Scheme, Common Lisp, Python, and ML, all but one of which are functional languages. ISOMER therefore explores relatively uncharted territory in applying these techniques to alternate programming models.

## 3 Novel design

### 3.1 Goal

Our project facilitates the detection of data-dependent errors by, among other things,

1. more thoroughly exploring the problem input space than otherwise feasible
2. collecting counterexamples / problematic inputs for regression tests
3. providing an extensible framework on which to build additional tools

### 3.1.1 Input space exploration

The fundamental advantage of constrained-random-stimulus (CRS) testing is that the tests it creates are both dynamic and intelligently guided to interesting areas. It is here that it differs from existing testing tools and methods including unit testing (“*xUnit*” for various values of  $x$ ) and fuzzing tools (for example, *fuzz* and *zzuf* [5]). CRS is more dynamic than *xUnit*, and more intelligent than fuzzers. ISOMER, while having important similarities to these tools, does not replace them.

The fundamental difference between ISOMER and a unit-testing system is the presence in ISOMER of a lightweight model and associated input-generating code. Typical unit tests are “directed,” meaning they check for specific, pre-recorded responses generated in response to specific, pre-recorded inputs. ISOMER’s use of a model and uniform, entropic inputs subject to programmer-specified rules potentially greatly increases confidence in program correctness because of the relative *dynamicity* of the tests produced by ISOMER.

ISOMER also differs from “fuzzing” programs, which attempt to produce program crashes by subjecting the program to random input. Fuzzing, however, does not generally produce valid stimuli and does not make use of a reference model, so it can increase confidence only in program robustness in the face of erratic input and not in program correctness. An implementation of our method with no specified constraints (or with constraints that do not represent the valid program input space) is functionally equivalent to a fuzzing program, but such a use does not involve behavior modeling and is therefore unrelated to the project goal. In addition, ISOMER, unlike fuzzers, is not limited to interacting with user-level interfaces (stdin/stdout, for example), but can access APIs directly as well.

### 3.1.2 Counterexample generation

The output produced by ISOMER is of various types; it can replay the exact inputs needed to produce a particular failure, it can be adapted to turn such a replay into a static, “directed” (possibly *xUnit*-style) test, or it can simply collect the random seeds that deterministically led it to a set of failures, which can be regenerated later. The ability of the system to “replay” a failure scenario (to generate a “counterexample” to a property assumed about program behavior) is very useful both for integration with existing test systems and for regression testing.

### 3.1.3 Lightweight modeling

It is worth discussing the meaning of “lightweight” in the context of software development. In hardware design (the original inspiration for this technique), external constraints such as timing and space requirements and concerns about simulating physical characteristics of a design add greatly to the design burden. These non-algorithmic concerns are stripped away

in a lightweight hardware model, resulting in an algorithmically-equivalent but significantly simpler-to-develop design. In software, similar external constraints (e.g., implementation language, frameworks, human-interaction interfaces) exist, but they are often less separable and less obvious than their hardware counterparts. It is well-recognized that some languages and development environments are well-suited to Rapid Application Development (RAD), and it is still true that a large portion of commercial software is not developed with these tools, due in many cases to their immaturity or other stigma. In ISOMER's context, "lightweight" can mean "developed using a RAD tool or methodology instead of one imposed by production constraints." The efficacy of our system depends on the rapidity of development and ease of maintaining the model; a C++ model for a C++ product, for example, is very unlikely to represent an overall gain, unless there are significant algorithmic optimizations that can be avoided in model development.

## **3.2 Limitations**

### **3.2.1 Conceptual limitations**

Despite the advantages to automatic specification-based testing (constrained-random verification), there are important limitations. Perhaps most obvious is the fact that a model must be written; although in some cases the model can be harvested from a prototype created before the production version, more often it must be written from scratch. If the advantages of ISOMER are not clear, a developer or team might be convinced with difficulty to add this to an already-compressed workload. We believe that the long-run advantages of ISOMER far outweigh the initial investment time, but this is a decision that must be made per project.

It is important to note that the constrained-randomly-generated-stimulus approach is not universally applicable; some problem domains do not lend themselves to "lightweight" modeling. For example, constraining random stimuli into a valid MPEG stream may not be reasonably implemented in a lightweight fashion. However, there remains a large class of algorithmic problems to which this approach could be usefully applied.

### **3.2.2 Technical limitations**

ISOMER aims to be a more general solution to the problem presented than the existing tools; by using a language-neutral syntax (unlike the mixed Haskell/Perl syntax of LectroTest [6]) and by using technology-independent, time-proven interfaces like POSIX pipes, ISOMER offers a single system that can be used across multiple projects and which can be learned without previous knowledge of a particular platform. However, this flexibility comes at a cost: a POSIX-pipes-only implementation makes interacting with whole programs easier but increases the effort necessary to interact with more granular components like functions. Language-specific shims can be automatically generated to limit the

impact of this limitation, but clearly an interface cannot be simultaneously entirely generic and perfectly granular.

### 3.3 Usage examples

For example, take a solution engine for the popular Sudoku numbers puzzle, written in C. Such a system might well use a backtracking engine with state objects being created and destroyed thousands or millions of times during a single program invocation. A programmer used to C++ or Java and their use of implicit copy constructors or garbage-collected references, respectively, might mistakenly return a pointer to a state stored on the program stack, for example. If this bug is hidden in an infrequently-used code path that is not well-exercised in the developer's test cases, it may go unnoticed for a long time. Even a simple set of constraints fed to ISOMER and allowed to run nights and weekends on the developer's own machine significantly raise the likelihood of discovering such a flaw by producing a set of inputs that produce deviant output. More carefully-written constraints and dedicated compute nodes can attack even far more complex problems. Another example might comprise a bio-informatics program developed in C++ compared to a rapidly-developed version using Perl and existing tools like BioPerl [1]. In cases like these, where the solution being developed is an innovation on an available tool or process rather than an entirely new one, the cost of developing the model may be negligible: BioPerl programs can't compete with commercial variants in speed, but they may well compete in correctness, which is the only requirement ISOMER makes.

Another field where ISOMER could be useful is the verification of multi-threaded applications when compared to their single-threaded ancestors or to a simpler, single-threaded but equivalent model. With the recent advances in multi-core processor technology, multi-threaded software is becoming necessary more rapidly than it is becoming available. Using ISOMER to compare a new multi-threaded system with a known-good single-threaded version or prototype could capture race conditions or undefined behavior much more quickly than a static testing framework could.

## 4 Implementation

Our implementation so far comprises the following elements:

1. a grammar (Figures 1 and 2) for the constraint system
2. a parser which consumes that grammar and generates an abstract syntax tree
3. a reserialization component (for testing and programmatic modification)
4. a random stimulus generator, producing reproducible sequences of typed data



```

1 <autotree>
2 Top:      Declaration(s?) Constraint(s?) EOF { bless +{
3           declarations => bless($item[1], 'DeclarationList'),
4           constraints => bless($item[2], 'ConstraintList')
5         }, $item[0] }
6 Declaration: Type Dimension(s?) Ident(s /,/) ';' {
7           $thisparser->Extend("VarName: "
8             join ' | ', map '/\b'. $_->_dump.'\b/', @{$item[3]});
9           bless +{ type => $item[Type],
10                  dimensions => $item[2],
11                  idents => $item[3] }, $item[0];
12         }
13 Type:     'int' | 'nybble' | 'byte'
14 Dimension: '[' /[0-9]+/ ']'
15 Constraint: 'constraint' Ident CGroup
16 Evaluable: VarName { $item[1] } | Number { $item[1] }
17 VarName:  <reject> # Starts out empty, gets extended by Declaration
18 Ident:    /\b[A-Z_]\w*\b/i
19 Number:   /\bd+\b/
20 CGroup:   '{' Statement(s ;//) ';' '}' { bless $item[2], $item[0] }
21 Implication { $item[1] } | BExpr { $item[1] }
22 BExpr:    InsideExpr { $item[1] } | CGroup { $item[1] } | IExpr { $item[1] }
23 IExpr:    ISEExpr[$arg[0] || 0] { $item[1] } |
24           IUExpr[$arg[0] || 0] { $item[1] } |
25           IPEExpr { $item[1] }
26 IPEExpr:  '(' IExpr[0] ')'
27 ISEExpr:  <leftop:IAtom[$arg[0]] IBiOp[$arg[0] || 0] IAtom[$arg[0]]> {
28           if (@{ $item[1] } == 1) { $item[1][0] }
29           else { bless $item[1], $item[0] }
30         }
31 IUExpr:   IUnOp IAtom[$arg[0]]
32 IBiOp:    <matchrule:IBiOp$arg[0]>
33 IBiOp8:   <commit> <reject>
34 IBiOp7:   '***'
35 IBiOp6:   '*' | '/' | '%'
36 IBiOp5:   '+' | '-'
37 IBiOp4:   '<<' | '>>'
38 IBiOp3:   '&' | '|' | '^'
39 IBiOp2:   '<=' | '>=' | '<' | '>'
40 IBiOp1:   '==' | '!='
41 IBiOp0:   '&&' | '||'
42 IUnOp:   '~' | '-' | '+'
43 IUnOp:   '!'
44 IAtom:   IUExpr[$arg[0] || 0]
45           { ($arg[0] || 0) < 8 | undef } IExpr[(($arg[0] || 0) + 1) { $item[2] } |
46           Evaluable { $item[1] }
47 InsideExpr: IExpr 'inside' '[' IList ']'
48 IList:     IAtom(s /,/) { bless $item[1], $item[0] }
49 Implication: BExpr '->' BExpr { bless [ @item[1,3] ], $item[0] }
50 EOF:      /\z/

```

Figure 1: Grammar text

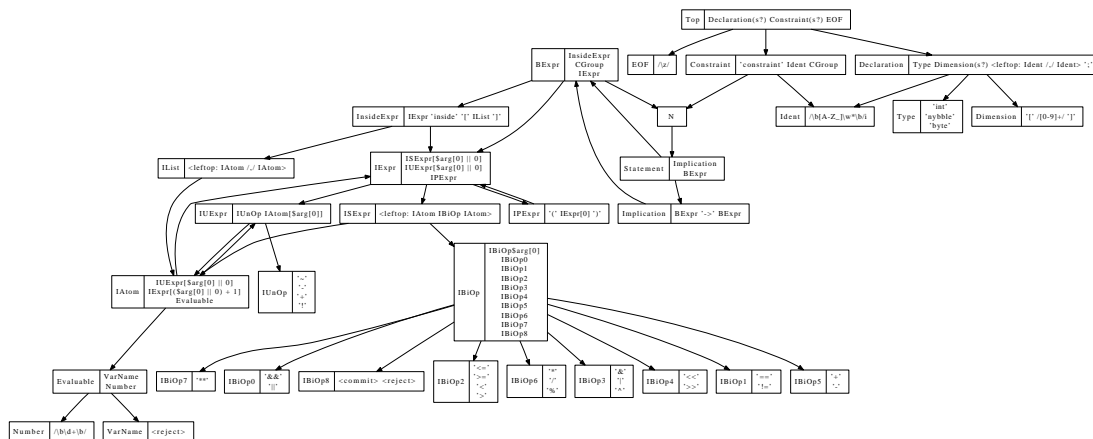


Figure 2: Grammar (graphical depiction)

```

1  int a, b, c, d, e;
2  int[100] large_array, intersecting_array;
3
4  constraint shell_game {
5      l inside [ a, b, c ];
6      a == 1 -> b == c && c == 0;
7      b == 1 -> c == a && a == 0;
8      c == 1 -> a == b && b == 0;
9  }
10
11 constraint complex_constraint {
12     a >= 0 -> {
13         c == 0 || b inside [ 1, 2, 3, 5 ];
14         a inside [ 1, 2, large_array ] -> !a inside [ intersecting_array ];
15     };
16     a < 0 -> ~(b >> 2) & c ^ d == e;
17 }

```

Figure 3: Example constraint file

5. a DUT wrapper, providing a common input/output interface to DUTs using POSIX pipes
6. an output comparator

The grammar is semantically complete but may eventually be extended to differentiate integer and Boolean expressions, which are currently treated in a weakly-typed fashion, as in Perl or C.

The grammar and parser are implemented using the `Parse::RecDescent` parsing module [4] for Perl. This parser-building framework builds, as its name implies, a recursive-descent parser from an LL(k) grammar. The potential inflexibilities of an LL(k) grammar were outweighed by the ease of producing and understanding such a grammar.

An example of a file that conforms to the grammar can be found in Figure 3. This arbitrary set of constraints does not represent the requirements of a typical constraint problem, but it does demonstrate the expressiveness and readability of the grammar. Developers familiar with C-like languages will feel at home with the syntax; the mathematical operations follow the semantics and precedence of the host language (i.e., Perl, and therefore very C-like).

A schematic diagram showing ISOMER’s main components is shown in Figure 4. The top two boxes represent the combination of the grammar in Figure 1 and the parser; the third from the top is a translation layer that converts the expression tree into a format compatible with `Test::LectroTest`. The “Filter Actuator” is the `Test::LectroTest` framework’s `TestRunner`, being supplied entropy on the “left” by its generation framework. The arrows emanating from the actuator represent the parallel POSIX pipes to the concurrently-running model and DUT (“PUT” in the diagram); the comparator is, due to the simple line-oriented interface, simply a patterned string comparison.

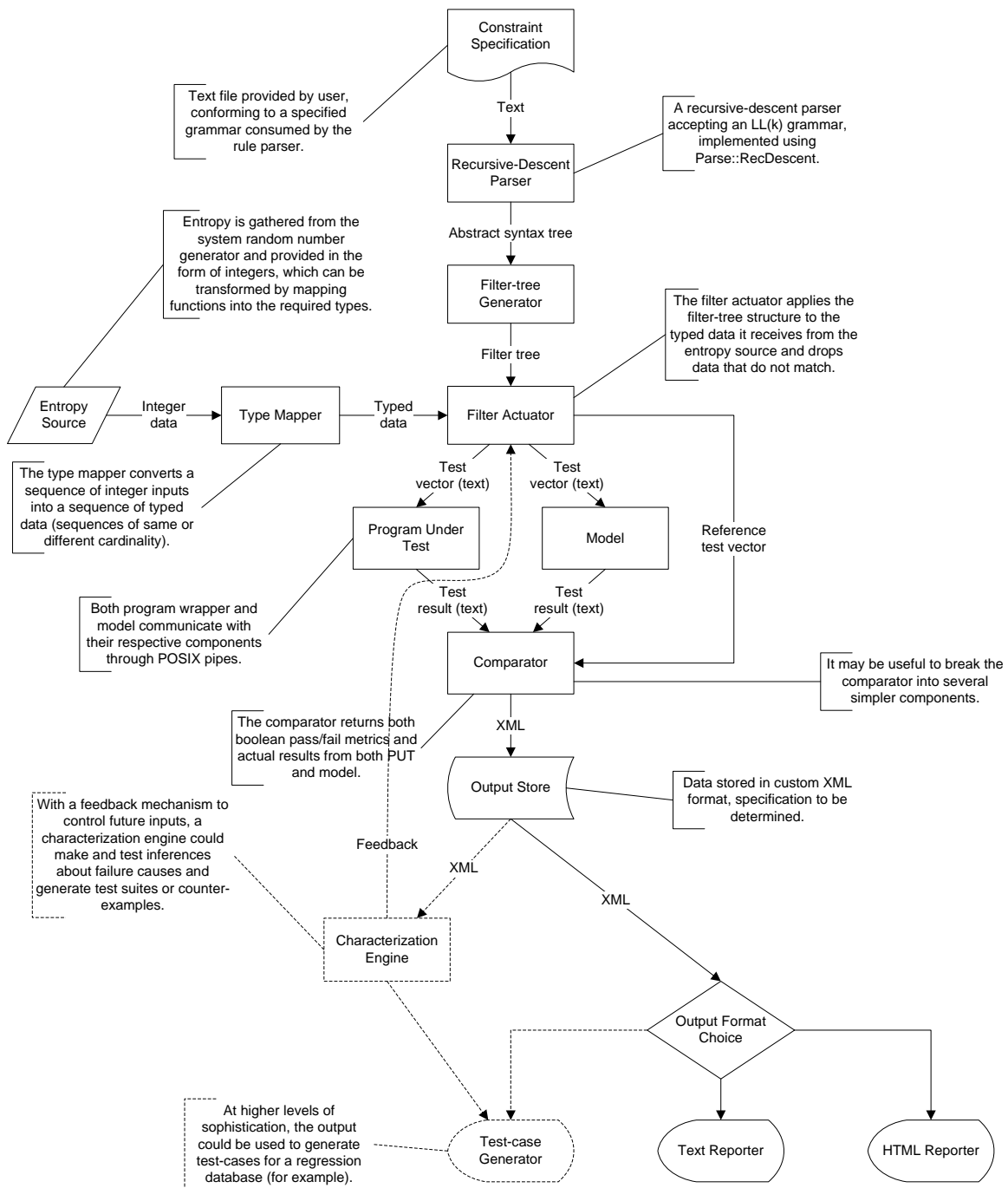


Figure 4: Schematic diagram

## 5 Evaluation

Two important areas in which to evaluate ISOMER are its level of functionality and its performance constraints. We address these in the paragraphs following.

### 5.1 Functionality

The functional evaluation can be divided into two subsets: type support and interface or I/O support. Other functional areas of course exist; these are simply the two spheres where ISOMER differentiates itself most from its predecessors.

#### 5.1.1 Type support

ISOMER currently supports a limited set of types, specifically 32-bit, 8-bit, and 4-bit signed integers. Support in the underlying LectroTest framework exists for more complex types, including strings, lists of other types, and arbitrary structural types through concatenation. In order to support some of these types, ISOMER's grammar will have to be extended; for the present we limit ourselves to integral types for simplicity.

#### 5.1.2 I/O support

As mentioned previously, ISOMER communicates with programs using POSIX pipes. Even under the current system, access to library functions and other program-internal components can be effected using shims. These shims must currently be created individually, but their creation can be automated in the future on a per-language basis to provide easy access to the function-level granularity that other CRS systems offer.

#### 5.1.3 Functionality example

To demonstrate ISOMER's functionality simply, we set up a constraint configuration with a purposely broken version of *bc*, a UNIX command-line calculator. The broken version substitutes the digit 4 for the digit 3 in its input, causing most expressions containing a 3 to fail. The "model" in this case was the normal version of *bc*. Figure 5 shows the log of a brief (5-trial) run of ISOMER on this configuration.

```

1 Seeding with 686968504 at ./harness.pl line 33.
2 First failure at test number 2
3 Stimulus prior to failure:
4 2 + 1
5 1 + 1
6 2 + 1
7 1 + 1
8 2 + 3
9 3 + 1
10 4 + 2
11 Faults:
12 ---
13 - dut_output: 6
14   input: 2 + 3
15   model_output: 5
16 - dut_output: 5
17   input: 3 + 1
18   model_output: 4
19 1..0

```

Figure 5: Example ISOMER run on *bc*

## 5.2 Performance

It was foreseen that ISOMER might create significant run-time overhead, which, if sufficiently severe, could render large-scale use impracticable. The current implementation has not been subjected to significant optimizations; as Donald Knuth said, “Premature optimization is the root of all evil.” At the moment, the filter-based entropy system, which does not benefit from any compilation of constraints to optimized generators, consumes most of the time in a typical run. While a simple design under test (a doubly-linked-list library wrapper) consumed 250 tests in 11ms, the entire run took on average about 9400ms to complete.

Clearly this level of performance is undesirable, but it is important to note that this major gap in performance closes as the DUT slows. For a non-trivial DUT, the time per test might be orders of magnitude slower, while the constraint generation time per test, which is stable and dependent directly on the complexity of the constraints (only negligibly on the number of variables being constrained), and since the generation and consumption of tests occur concurrently, the loss is not so great as it first appears. Naturally, performance issues will be explicitly addressed in future revisions.

## 6 Conclusion

We believe the underlying concept behind constrained-random-stimulus testing to be solidly proven in theory and in real use with hardware, and we feel an exploration of its application to software development will result in long-term improvements to developer productivity and product quality.

## References

- [1] Main Page - BioPerl. <http://www.bioperl.org/>.
- [2] BERGERON, J., CERNY, E., HUNTER, A., AND NIGHTINGALE, A. *Verification Methodology Manual for SystemVerilog*. Springer, 2005.
- [3] CLAESSEN, K., AND HUGHES, J. QuickCheck: An Automatic Testing Tool for Haskell. <http://www.cs.chalmers.se/~rjmh/QuickCheck/>.
- [4] CONWAY, D. Parse-RecDescent-v1.95.1 - search.cpan.org. <http://search.cpan.org/dist/Parse-RecDescent/>.
- [5] HOCEVAR, S. zzuf - multi-purpose fuzzer. <http://sam.zoy.org/zzuf/>.
- [6] MOERTEL, T. Moertel Consulting's Community Projects :: LectroTest. <http://community.moertel.com/ss/space/LectroTest>.

# Detecting Source Code Plagiarism

Joseph Degiovanni and Imad Rahal (advisor)  
Computer Science Department  
College of Saint Benedict Saint John's University  
Collegeville, MN 56321  
[J1degiovann@csbsju.edu](mailto:J1degiovann@csbsju.edu)

## Abstract

The intention of this paper is to act as a tutorial on the subject of the detection of source code plagiarism. The first section is a short introduction to the problem. Then there is a section explaining some of the popular plagiarism detection approaches including Halstead's Software Science Metric, McCabe's Cyclomatic Complexity, Lancaster Word Pairs, and Longest Common Substrings. The third section briefly describes some common plagiarism detection engines that implement the approaches described earlier. Finally, the last section summarizes the discussion and also suggests further research on the topic.

# 1. Introduction

Plagiarism has been defined as the act of fully or partially submitting someone else's written work as one's own. The problem, or plague as described in the literature, is very common in written works especially among university students due to various reasons such as time pressure, lack of understanding of what constitutes plagiarism, and the wealth of digital resources available on the Internet which make "copy/paste" activities almost natural!

There are many types of plagiarism. Some of the types are described by the category of media that is being plagiarized, such as journalistic plagiarism, musical plagiarism, film plagiarism, and even plagiarism of scientific experiments. Another way of classifying plagiarism describes where the original document was retrieved from in relation to the corpus. A corpus is a collection of documents within which plagiarism could likely occur. Plagiarism can be described as intra-corporal or extra-corporal. Intra-corporal plagiarism occurs when both the source and plagiarized documents exist within the corpus, for example, a student copying another student's assignment in the same class. Extra-corporal plagiarism then, not surprisingly, is when the source document exists outside of the corpus, for example, a student turning in a term paper from an online term-paper mill such as [alessays.com](http://alessays.com) or [termpapergenie.com](http://termpapergenie.com). This paper will focus specifically on source code plagiarism which is defined as the act of plagiarizing source code written in programming languages such as Visual Basic, Java, C++, etc..., and how it is being detected.

In any academic department, plagiarism is harmful to a student's education, but it can be particularly harmful within computer science. The reason for this is that programming is a skill that is best (and arguably can only be) learned through experience. Therefore students who plagiarize on programming assignments are usually the ones who do not comprehend what is taught in the classroom.

Source code plagiarism may be a more common practice than one is inclined to think. In a survey given to 110 UK HE computing schools [3], 35 out of 52 instructors who responded to a question regarding "the proportion of students involved in a typical outbreak of source code plagiarism on initial programming courses" reported that they think more than 10% of students would be involved. Four of the 52 respondents reported that they think it is more than half! If one thinks that these figures are exaggerated, here is something to consider: "The evidence of previous UK surveys on student behavior suggests that tutors under-estimate the extent of cheating behavior" [3]. Keeping this information in mind, it is apparent that source code plagiarism is a real problem in schools worldwide that are trying to teach students how to write computer software.

In order to deter students from plagiarizing their programming assignments, some schools have imposed severe punishments for submitting a plagiarized assignment. For example, at the College of St. Benedict/St. John's University in Minnesota, students automatically fail the course and are at risk of expulsion from the school on their second offense. Students who are caught plagiarizing three times are automatically expelled. However, in order for these rules to be effective, they need to be enforced, which brings us to the issue



of detecting plagiarism. A primitive approach to detecting plagiarism is by visual inspection. This works fine as long as the number of documents to be inspected is somewhat small, say 20-30. However, in introductory programming courses, enrollment can be much higher, making visual inspections time-consuming and impractical. This approach can lead to cases such as the one quoted in Clough's article: "Recently, two students handed in exactly the same essay. One stole the essay from a computer used by both. The two copies differed only in the name at the top. The culprit assumed that, given a class of 200 students in which the work is marked by five different tutors, the forgery would go undetected. That was, in fact, a pretty safe assumption since the discovery was entirely accidental" [1]. In such cases, automated systems are necessary in order to effectively detect and thus deter plagiarism.

So how do we build an automated system for detecting plagiarism? The problem with the definition provided in the first sentence of this paper is that it is not concise. Where do we draw the line between plagiarism and coincidence? There needs to be some way to measure how similar two documents are so that we can clearly say that the documents are too similar to happen by pure coincidence.

## **2. Plagiarism Detection Techniques**

In plagiarism detection engines (computer systems for detecting plagiarism) metrics are used to compute the similarity of two documents, usually producing a similarity score. A similarity score is a numeric value in a certain range, like 0-100, that indicates how similar two documents are, where a similarity of 100 means that they are exact copies of each other and a similarity of 0 means that they have absolutely nothing in common.

Many types of metrics have been proposed to measure the similarity among source code submissions. Lancaster and Culwin classify these metrics based on three criteria: 1) the number of submissions that have to be analyzed consecutively to generate metrics, 2) the complexity of the metric, and 3) whether or not the metric applies tokenization to the submissions [2].

The first criterion is divided into two categories. All metrics that have currently been introduced in the literature analyze either one or two submissions at a time. If a metric analyzes just one submission at a time it is called a singular metric. The term singular metric can be deceiving, because it does not mean that the submission is not compared with any other submissions; it just means that the metric is computed with respect to only one submission (e.g. Mean number of characters per line). On the other hand, metrics that analyze pairs of submissions to calculate similarity are called paired metrics. An example of a simple paired metric is the number of words two submissions have in common. Most current plagiarism detection engines today use paired metrics because they are generally thought to be more effective.

The second criterion is also divided into two categories, superficial and structural. Lancaster and Culwin define a superficial metric as a metric that can produce a numerical

representation of a document or set of documents where knowledge of their linguistic properties is not necessary. Similarly, a structural metric is one where knowledge of their linguistic properties *is* necessary. One large advantage that superficial metrics have over structural metrics is that they are language-independent; therefore the metric can be applied to a broader range of documents. This advantage also comes with the drawback that they are easier for a potential plagiarist to fool by replacing some of the original syntax with different structures that produce the same function in the language. For this reason, the majority of plagiarism detection engines in use today use structural metrics. Lancaster and Culwin also mention that distinguishing superficial metrics from structural metrics can be difficult [2].

The last metric classifier is much easier to determine. It is determined by whether or not the metric uses tokenization. Tokenization is the process of replacing commonly used phrases or constructs of a language with a condensed version, or “token”, that carries the same (or very similar) meaning. For example, variable declarations in Visual Basic like:

```
Dim myNumber As Integer
```

Could be replaced by a two character token such as:

VI

Where ‘V’ stands for variable declaration and ‘I’ stands for the variable type, which happens to be Integer in this case. This process of tokenization is particularly useful in detecting plagiarism that has been concealed by one or more of the thirteen techniques that students use to alter source code that they plagiarize. These thirteen techniques have been identified by Whale [10] and they are:

1. changing comments
2. changing formatting
3. changing identifiers
4. changing the order of operands in expressions
5. changing data types
6. replacing expressions with equivalents
7. adding redundant statements
8. changing the order of time-independent statements
9. changing the structure of iteration statements
10. changing the structure of selection statements
11. replacing procedure calls with the procedure body
12. introducing non-structured statements
13. combining original and copied program fragments

For example, technique number three could be attributed to a novice programmer who might just replace the identifier ‘myNumber’ with his own variable name like ‘num’. Tokenizing the code makes non-important and interchangeable features such as identifiers irrelevant when comparing documents.

There is no perfect way to tokenize a submission. Each style has advantages and disadvantages. For example, the example above could be just reduced to the token “V” and the type could be disregarded. This would make the tokenization more resistant to a programmer who replaced the integer type variable with a variable of type single (which can represent fractional values). On the negative side, it would also make non-plagiarized seem more similar to each other, because some of the uniqueness of that specific line of code was lost.

## 2.1. Halstead’s Software Science Metric

In 1977, Maurice Halstead proposed one of the earliest software metrics, named Halstead’s software science metric [4]. Halstead’s software science metric is a singular superficial metric. The way that this metric is implemented in detecting plagiarism is very simple. A system takes a count of the following attributes in each programming submission within a corpus:

- Number of unique operators ( $n_1$ ) (+,-,\*,/, etc...)
- Number of unique operands ( $n_2$ ) (myVariable, name, etc...)
- Total number of operators ( $N_1$ )
- Total number of operands ( $N_2$ )

If two submissions have counts that are exactly the same, they are viewed as very similar and would require further investigation. Other possible implementations using this metric would be to compute the 4-dimensional distance using a formula like Euclidean, Chebychev, or Manhattan (city block) distance. Submissions with near zero distances would then be viewed as suspicious and require further investigation.

A problem with this metric is that for small, simple programs (typical of those assigned to introductory computing classes) the 4-attribute tuples can be very similar or even exactly the same even though this usually doesn't mean that plagiarized has occurred. Also, for programs of different sizes, this metric would be very ineffective at detecting plagiarism if only small portions had been plagiarized because the counts of operands are closely correlated with program length.

## 2.2. McCabe’s Cyclomatic Complexity

M McCabe’s cyclomatic complexity was proposed around the same time that Halstead’s metric was introduced. It directly measures the number of linearly independent paths through a program's source code [7]. Using a graph of the control flow of a program, the cyclomatic complexity is calculated as the number of edges ( $e$ ) minus the number of nodes ( $n$ ) plus two times the number of connected components ( $p$ ):  $V(G) = e - n + 2p$ .

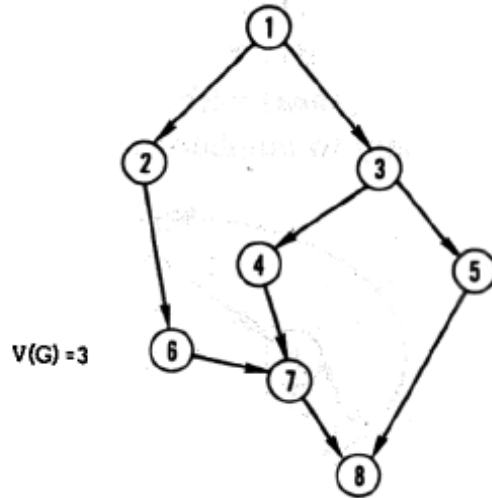


Figure 1: Complexity graph [7]

In the figure above, there are nine edges and eight nodes. Since there is only one component, the number of connected components is equal to one. The nodes represent blocks of code and the edges represent possible paths from the entry point (node 1) to the end point (node 8).

By itself, McCabe's cyclomatic complexity is not a very good metric for detecting plagiarism because there are many control flow graphs with the same complexity number, albeit, the structures could look very different. Also, it suffers from the same problem as Halstead's metric: it is not effective when the corpus consists of a large number of submissions working on the same simple problem, because those submissions are very likely to have the same complexity graph due to the nature of the small assignment. In the end, in order for cyclomatic complexity to be effective it should be coupled with other metrics that will give more information about the compared programs.

### 2.3. Lancaster Word Pairs

This is a paired superficial metric. The idea behind Lancaster word pairs is that if two submissions have a substantial amount of pairs of words that occur consecutively and they are common to both, then the two submissions are very similar. For example, if two submissions have the word pair “green apple” in common, then they are more similar than if they had nothing in common. The more word pairs two submissions have in common, the more similar they are.

An outline of the algorithm is listed below:

Consider two Visual Basic projects, denoted A and B.

For each document, compute a sorted list of all pairs of consecutive words, along with a count of the number of times this pair appears. Denote these lists A and B respectively.

Compute values  $c_1$ ,  $c_2$ ,  $C$ ,  $u_1$ ,  $u_2$  and  $U$ , representing commonality and uniqueness between the projects, where:

$c_1$  is the number of times a pair of words occurs in A, so long as it occurs at least once in B.

$c_2$  is the number of times a pair of words occurs in B, so long as it occurs at least once in A.

$$C = c_1 + c_2.$$

$u_1$  is the number of times a pair of words occurs in a, so long as does not occurs in b.

$u_2$  is the number of times a pair of words occurs in b, so long as does not occurs in a.

$$U = u_1 + u_2$$

Then similarity score for A and B =  $100 * C / (C + U)$ .

[6]

Lancaster and Tetlow implemented and compared this metric with other metrics that have been proven to be effective at detecting collusion in programming assignments, and they found it to be very effective [6]. The results are surprising because it was originally thought that a superficial metric would not be nearly as effective as a structural metric in detecting source code plagiarism. However they also state that there is a major drawback to Lancaster word pairs over other metrics such as tokenized longest substrings because it does not indicate *where* the two submissions are suspected to be plagiarized. One difficult task for tutors would be figuring out why two submissions were flagged by the system, which, on its own, could prove to be a time consuming procedure.

## 2.4. Tokenized longest common substrings

This is a paired, structural, tokenized metric. The first step in using this metric is to tokenize an entire set of documents. The result will be a compressed version of the original documents that still contains the important information that will be used to compute similarity between two documents. The second step is to find the longest common substrings within the tokenized versions of two documents.

When a set of words or tokens exist in two documents in the same order, this is known as a common substring. Consider these two hypothetical tokenized documents:

**SUBVSVIVIFORV+=ENDEND**  
**SUBVIVSWHILEV-=ENDV-VEND**

They both have this substring in common:

SUBVIV=ENDEND

This also happens to be the longest common substring, but any section of that string would also be a common substring as long as the order is preserved. The following is a basic outline of how the tokenized longest common substrings metric can be used to compute a similarity score for two documents:

Consider two Visual Basic projects, denoted A and B.

Produce two further documents, denoted a and b respectively, which are tokenized versions of A and B. Denote the length of these documents, in words and tokens, as  $l_a$  and  $l_b$  respectively.

Compute the longest substring common to both A and B, with a minimum length of five words or tokens. A common substring is defined as a series of words and tokens common to both documents and in the same order, but not necessarily consecutive to one another.

Iteratively repeat this process on the words and tokens in A and B that have not yet been allocated to a substring, until no further allocations are possible.

Denote the total length of these substrings as  $l_c$ .

Then similarity score for A and B =  $200 l_c / (l_a + l_b)$ .

[6]

This metric was originally used for finding similarity in genetic code to help identify genes that are common among different species. It is one that is very commonly used in current plagiarism detection engines because it is resistant to attempts aimed at disguising plagiarism by adding white spaces or redundant code.

## 2.5. Winnowing

Winnowing is an algorithm that makes use of a technology called Fingerprinting [9]. Fingerprinting makes string comparison much more efficient because it reduces strings into representative numerical figures, called “fingerprints”. These fingerprints are derived from strings of a given size,  $k$ , which are called  $k$ -grams.  $K$ -grams are created from a document by finding all consecutive substrings of size  $k$ . Step c in the figure below shows all of the  $k$ -grams, such that  $k$  is equal to 5, from the string in step b. The basic idea is that if two documents share a fingerprint, it is extremely likely that they share a  $k$ -gram (suggesting potential plagiarism). The following example illustrates the winnowing process:

A do run run run, a do run run

(a) Some text.

adorunrunrunadorunrun

(b) The text with irrelevant features removed.

adoru dorun orunr runru unrun nrunr runru  
 unrun nruna runad unado nador adoru dorun  
 orunr runru unrun

(c) The sequence of 5-grams derived from the text.

77 74 42 17 98 50 17 98 8 88 67 39 77 74 42  
 17 98

(d) A hypothetical sequence of hashes of the 5-grams.

(77, 74, 42, 17) (74, 42, 17, 98)  
 (42, 17, 98, 50) (17, 98, 50, 17)  
 (98, 50, 17, 98) (50, 17, 98, 8)  
 (17, 98, 8, 88) (98, 8, 88, 67)  
 (8, 88, 67, 39) (88, 67, 39, 77)  
 (67, 39, 77, 74) (39, 77, 74, 42)  
 (77, 74, 42, 17) (74, 42, 17, 98)

(e) Windows of hashes of length 4.

17 17 8 39 17

(f) Fingerprints selected by winnowing.

[17,3] [17,6] [8,8] [39,11] [17,15]

(g) Fingerprints paired with 0-base positional information.

[9]

Steps a through d in the figure above are typical of most fingerprinting techniques. However, the idea of a window in step (e) is unique to the winnowing algorithm. A window is pretty much the same as a  $k$ -gram except that the length is calculated from a number of input variables and not a direct input value like ' $k$ '. The length of the windows is  $w = t - k + 1$  where  $t$  is the threshold set by a user that is the smallest substring that is

guaranteed to be matched if it exists in both documents. For example, if both documents share a substring of size  $s \geq 9$  and  $t = 9$ , then that substring is guaranteed to be detected as common to both documents after comparing fingerprints. Similarly,  $k$  is the threshold that is guaranteed to ignore substrings that are smaller than  $k$ . So if  $k = 5$ , the substring “the” will be disregarded as noise. Schleimer, Wilkerson, and Aiken [9] prove that  $w$  is the correct length for windows to guarantee those thresholds.

Step (f) uses the hashes in the windows to select appropriate fingerprints, starting with the first window, winnowing selects the smallest hash in the window and moves to the next window. Then, if the selected hash is still the smallest hash in the new window, no new hash is selected for a fingerprint. If the new window does contain a smaller (or equal) hash, that hash is selected as another fingerprint and becomes the new current fingerprint. This process continues until all of the windows have been processed. Finally, step (g) adds positional information to the set of fingerprints. This positional information is useful for visualization tools that point out where the similarity has occurred within documents such as this tool used by JPlag:

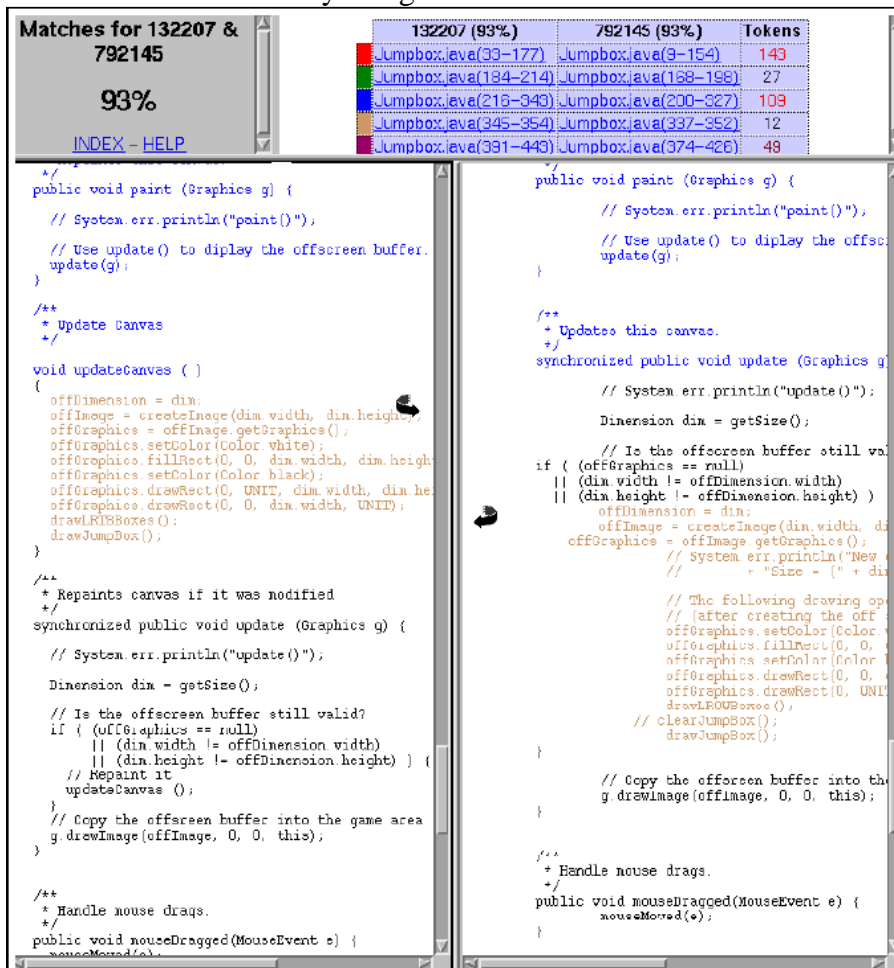


Figure 2: JPlag's visualization tool [8]

## 2.6. Greedy String Tiling



Greedy String Tiling (GST) is considered to be very effective in detecting similarities between strings even if the order has been rearranged. It searches for matching elements of a pattern string, P, within a text string, T. McElory Hoffman uses the following example:

Suppose that

T = TheQuickBrownFoxJumps

And

P<sub>1</sub> = Brown  
P<sub>2</sub> = FoxBrown

Using the Tokenized Longest Common Strings (LCS) approach, the similarity values for P<sub>1</sub> and P<sub>2</sub> are 38.5% and 34.5% respectively. As you can see, P<sub>2</sub> is negatively affected because order matters in LCS. GST corrects this shortcoming by using the idea of a 'tile'. A tile is a 'maximal match' in which every element has been 'marked'. Maximal matches are found by comparing the all the elements in the pattern string to all the elements in the text string. Then, if a match is found, the maximal match can be extended if the next element in P matches the next element in T. It continues to be extended until there is a non-match, end-of-string, or marked element. Maximal matches that have a length less than a given value, the minimum match length, are disregarded as noise; if they are equal to or larger than the minimum match length, then they are deemed as marked and become tiles. Similarity is then computed with the following formula:

$$S = \frac{2 \cdot |\text{tiles}|}{|P| + |T|}$$

where

$$|\text{tiles}| = \sum_{\text{match}(p,t,s) \in \text{tiles}} s.$$

and  $|P| + |T|$  = the number of elements in P added to the number of elements in T [5].

### 3. Plagiarism Detection Engines

Plagiarism detection engines are computer systems that implement one or more metrics in an attempt to detect code plagiarism. Currently, most of those systems are publicly accessible via the Internet. The main application for these engines is for educators to check student submissions for plagiarism.

Lancaster and Culwin have surveyed the known plagiarism detection engines and have provided the following table that classifies the engines by the metrics they use:

Table 4. Classifications of Source Code Engines by Metrics Used.

Name of engine	Singular	Paired	Superficial	Structural	Tokenisation
Big Brother		✓		✓	✓
Cogger		✓		✓	✓
DetectaCopias		✓		✓	✓
Jones	✓		✓		✓
JPlag		✓		✓	✓
MOSS		✓		✓	✓
Saxon		✓		✓	✓
SHERLOCK		✓		✓	✓
SIM		✓		✓	✓
TEAMHANDIN	✓		✓		✓
YAP3		✓		✓	✓

[2]

Here you can see that most engines use a paired structural tokenized metric. It is also worth mentioning that the two most commonly used and recommended engines, MOSS and JPlag, also implement paired structural tokenized metrics. An evaluation of JPlag by Malpohl, Philippsen, and Prechelt show that nearly all plagiarism attempts were discovered, and even caught more than 90% of simulated plagiarisms where the subjects knew that their purpose was to disguise their work for a plagiarism detection engine [8].

## 4. Summary

Even though a lot of research efforts have been invested in detecting source code plagiarism for over 30 years now, there is still a lot of debate about which metrics work the best. Literature on the topic from different academics still seems to disagree with each other with a lot of ambiguity on the terminology used to classify metrics and the engines that implement them. Furthermore, a number of publicly accessible systems have not been described in the literature in an attempt to discourage students from understanding how they work so as not to find workarounds. In addition, there are new programming languages that are being taught in computing schools that do not have any support from the plagiarism detection engines yet. Even older, more commonly used languages such as Visual Basic are not supported by most engines.

More work needs to be done in this area to improve on the current engines or to devise new, more efficient, easy-to-use engines with established performance in order to encourage educators to use them and enhance the academic integrity of computing courses in universities around the world.

## 5. References

- [1] Clough, P. 2000. Plagiarism in natural and programming languages: an overview of current tools and technologies. July 2000. Department of Computer Science, University of Sheffield
- [2] Culwin, F. and Lancaster, T. 2005. Classifications of Plagiarism Detection Engines. ICS (Jan. 5, 2005). <http://www.ics.heacademy.ac.uk/italics/Vol4-2/Plagiarism%20-%20revised%20paper.pdf>
- [3] Culwin, F. MacLeod, A. & Lancaster, T. 2001. Source-code Plagiarism in UK HE Computing Schools, Issues, Attitudes and Tools. Technical Report No. SBU-CISM-01-02. South Bank University.
- [4] Halstead, M.H. Elements of Software Science. Prentice-Hall, Inc., New York, 1977
- [5] Hoffmann M. The Plagiarism Detector Copy-D-Tec. Project 478. Department of Computer Science, University of Stellenbosch. November 2004. Private Bag X1, 7602 Mateieland.
- [6] Lancaster, T. and Tetlow, M. 2005. Does automated anti-plagiarism have to be complex? Evaluating more appropriate software metrics for finding collusion. In Balance, Fidelity, Mobility: Maintaining the Momentum, Proceedings of the 2005 ASCILITE Conference. [[http://www.ascilite.org.au/conferences/brisbane05/blogs/proceedings/42\\_Lancaster.pdf](http://www.ascilite.org.au/conferences/brisbane05/blogs/proceedings/42_Lancaster.pdf)]
- [7] McCabe, T. J. 1976. A Complexity Measure. IEEE Transactions on Software Engineering, Vol. SE-2, no. 4, December 1976
- [8] Prechelt, L., Malpohl, G., and Philippsen, M. 2000. Finding plagiarisms among a set of programs with JPlag. Submitted to Journal of Universal Computer Science; <http://www.ipd.ira.uka.de/jplag>. <http://citeseer.ist.psu.edu/prechelt00finding.html>
- [9] Schleimer, S., Wilkerson, D. S., and Aiken, A. 2003. Winnowing: local algorithms for document fingerprinting. In *Proceedings of the 2003 ACM SIGMOD international Conference on Management of Data* (San Diego, California, June 09 - 12, 2003). SIGMOD '03. ACM Press, New York, NY, 76-85. DOI=<http://doi.acm.org/10.1145/872757.872770>
- [10] Whale, G. 1986. Detection of Plagiarism in Student Programs, "Proceedings of the Ninth Australian Computer Science Conference, Canberra", pp. 231-241

# Parallelizing the Computation of the SPT Statistic

Todd Frederick

Department of Mathematics, Statistics, and Computer Science

St. Olaf College

Northfield, MN 55057

frederit@stolaf.edu

## Abstract

Partition theorist George Andrews defined  $\text{spt}(n)$  as the number of smallest parts of partitions of  $n$ , and proved several congruences for  $\text{spt}$ , patterns that are similar in form to congruences discovered by Ramanujan for the partition function. To aid investigation of these congruences, an algorithms class at St. Olaf developed a method of computing  $\text{spt}(n)$  in quadratic time using linear working memory. We develop a parallel version of this algorithm that runs on a Beowulf cluster, a MIMD parallel computer, with noticeable performance improvements.

# 1 Introduction

## 1.1 Integer Partitions

The set  $\lambda = (\lambda_1, \lambda_2, \dots, \lambda_k)$  is a *partition* of  $n$  if  $\lambda_1 + \lambda_2 + \dots + \lambda_k = n$  and  $\lambda_1 \geq \lambda_2 \geq \dots \geq \lambda_k$ . We say  $\lambda$  has  $k$  parts. The single part  $n$  is the trivial partition of itself.

The partition function  $p(n)$  is the number of distinct partitions of  $n$ . For example, there are 5 partitions of 4.

```

4
3 1
2 2
2 1 1
1 1 1 1

```

Values of  $p(n)$  increase exponentially and are therefore difficult to compute for large values of  $n$ .

## 1.2 Partition Function Congruences

Mathematician Srinivasa Ramanujan discovered a series of patterns in the partition function called congruences, including

$$\begin{aligned}
 p(5k + 4) &\equiv 0 \pmod{5} \\
 p(7k + 5) &\equiv 0 \pmod{7} \\
 p(11k + 6) &\equiv 0 \pmod{11}
 \end{aligned}$$

As an example, the first congruence states that the number of partitions of any integer of the form  $5k + 4$  is divisible by 5.

## 1.3 The Smallest Parts of Partitions

Partition theorist George Andrews defined the statistic  $\text{spt}(n)$  as the number of smallest parts in the partitions of  $n$  [1]. Each partition of  $n$  denoted by  $\lambda$  has a smallest part that occurs  $m_\lambda$  times in  $\lambda$ . The value of  $\text{spt}(n)$  is the sum of  $m_\lambda$  for all  $\lambda$ . For example, we count the smallest parts in the partitions of 4.

```

4           1
3 1        1
2 2        2
2 1 1      2
1 1 1 1    4
          10 = spt(4)

```

The first partition, the trivial partition, has one part, 4 itself. This part 4 is the smallest part in the trivial partition, and it occurs 1 time. A partition may have multiple occurrences of its smallest part. The smallest part of the fourth listed partition is 1, and it occurs 2 times in that partition. When we sum the smallest parts of each partition, we get  $spt(4) = 10$ .

## 1.4 Congruences with SPT

Andrews also proved several congruences for  $spt(n)$ , including

$$\begin{aligned} spt(5k + 4) &\equiv 0 \pmod{5} \\ spt(7k + 5) &\equiv 0 \pmod{7} \\ spt(13k + 6) &\equiv 0 \pmod{13} \end{aligned}$$

Andrews did not find a congruence modulo 11, such as exists for the partition function, motivating a search for new congruences for  $spt(n)$ , starting with conjectures of possible congruences. Searching for possible congruences requires a large set of data in which to search, but the existing generating function-based method of calculating  $spt(n)$  proved prohibitively slow for computing a large set of values of  $spt(n)$ .

## 1.5 Initial Approaches

St. Olaf Mathematics Professor Kristina Garrett posed the challenge of generating a large  $spt$  data set to Professor Olaf Hall-Holt's algorithms class, which included this author. The class investigated different methods of calculating  $spt(n)$  with the goal of significantly increasing the number of values of  $spt(n)$  that could be computed in a fixed amount of time [3][2]. An existing method using a generating function on a popular symbolic calculation system computed hundreds of values in a few minutes. A worthwhile search for possible congruences would require at least thousands or tens of thousands of values.

We first implemented a simple exhaustive search algorithm that listed partitions of  $n$  and counted the number of their smallest parts. This algorithm ran in exponential time, yielding absolutely insufficient performance. We then discovered a recurrence relation for  $spt(n)$  that we developed into a far more efficient algorithm.

# 2 Computing SPT

## 2.1 Serial Algorithm

We generate the set  $\mathbb{S}$  of all  $spt(n_i)$ , where  $1 \leq n_i \leq n_{max}$  for a maximum integer  $n_{max}$ , through a recursive method of counting the partitions of each  $n_i$ . By counting partitions according to their possible smallest parts, we can keep track of the number of each smallest

part for each  $n_i$  and sum these counts to get  $\text{spt}(n_i)$ .

Let  $\mathbb{P}_n$  be the set of possible smallest parts of partitions of  $n$ . The integer  $n$ , the smallest part of the trivial partition of  $n$ , is in  $\mathbb{P}_n$ . If there were some  $p \in \mathbb{P}_n$ , part of a non-trivial partition, such that  $p > \lfloor n/2 \rfloor$ , then the remainder  $n - p$  could not be distributed into a part larger than  $p$ , so  $p$  could not be a smallest part. Thus, we have

$$\mathbb{P}_n = \{p : p = n \text{ or } p \leq \lfloor n/2 \rfloor\}$$

Consider a partition of  $n$  denoted by  $\pi_a$  with a smallest part  $p_a$  and the partition of  $n - p_a$  denoted by  $\pi_b$  with a smallest part  $p_b \geq p_a$  that is equivalent to  $\pi_a$  with a single part  $p_a$  removed. To count the occurrences of  $p_a$  in  $\pi_a$ , we add 1 to the number of occurrences of  $p_b$  in  $\pi_b$ . To count the occurrences of  $p_a$  as a smallest part in all partitions of  $n$ , we add the number of occurrences of  $p_a$  as a smallest part in all partitions of  $n - p_a$  to the number of partitions of  $n - p_a$  that have a smallest part  $\geq p_a$ .

To clarify this recurrence, we present an example. We count the occurrences of 1 as a smallest part of partitions of 5. Any partition of 5 with a smallest part of 1 is a partition of 4, that has an original smallest part  $\geq 1$ , with a 1 joined to the end. Recall the partitions of 4.

```

4
3 1
2 2
2 1 1
1 1 1 1

```

All of these partitions have a smallest part  $\geq 1$ , so adding a 1 to the end of each partition will form all the partitions of 5 with a smallest part of 1.

```

4 1
3 1 1
2 2 1
2 1 1 1
1 1 1 1 1

```

Notice that 1 occurs 7 times as a smallest part of partitions of 4. When we form partitions of 5 that have a smallest part 1, we add an extra 1 to each of the 5 partitions, so there are 12 occurrences of 1 as a smallest part in the partitions of 5.

To compute this recurrence relation, we store  $\beta_{n_i, p_i}$ , the number of occurrences of  $p_i$  as a smallest part in all partitions of  $n_i$ . We also store  $\alpha_{n_i, p_i}$ , the number of partitions of  $n_i$  that have a smallest part that is  $\geq p_i$ . The values of  $\beta_{n_i, p_i}$  are computed by the recurrence

relation as described above. The values of  $\alpha_{n_i, p_i}$  are computed by adding the number of partitions of  $n_i - p_i$  that have a smallest part  $\geq p_i$  to the number of partitions of  $n_i$  that have a smallest part  $> p_i$ , or  $\geq p_i + 1$ . The values of  $\alpha_{n_i, p_i}$  and  $\beta_{n_i, p_i}$  are summarized in Figure 1 and by the following equations

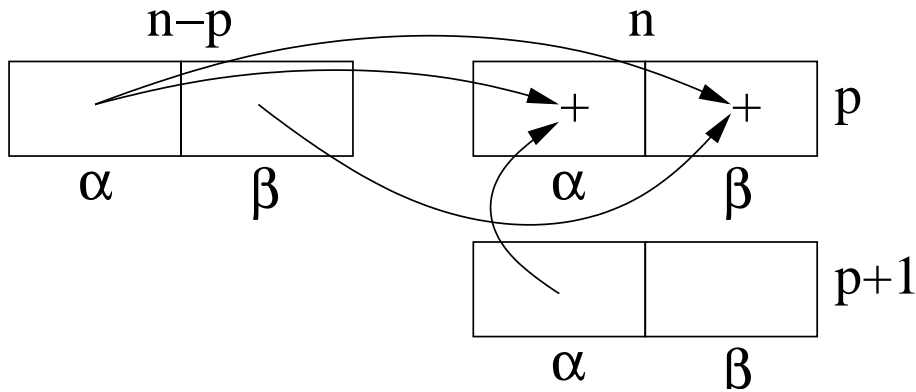


Figure 1: Calculating an  $\alpha, \beta$  pair.

$$\begin{aligned}\alpha_{n_i, p_i} &= \alpha_{n_i - p_i, p_i} + \alpha_{n_i, p_i + 1} \\ \beta_{n_i, p_i} &= \alpha_{n_i - p_i, p_i} + \beta_{n_i - p_i, p_i}\end{aligned}$$

Because  $\beta_{n_i, p_i}$  counts the occurrences of a single part  $p_i$  in the partitions of  $n_i$ , we can define  $spt(n_i)$  in terms of  $\beta_{n_i, p}$ .

$$spt(n_i) = \sum_{p \in \mathbb{P}_{n_i}} \beta_{n_i, p}$$

As there is one trivial partition of  $n_i$ , in which  $n_i$  occurs once,  $\alpha_{n_i, n_i} = 1$  and  $\beta_{n_i, n_i} = 1$ . To compute the elements of  $\mathbb{S}$ , we compute  $\alpha_{n, p}$  and  $\beta_{n, p}$  for all  $n \in \{n : 2p \leq n \leq n_{max}\}$  for all  $p \in \{p : 1 \leq p \leq \lfloor n_{max}/2 \rfloor\}$ . Note that we iterate  $p$  in decreasing order because  $\alpha_{n_i, p_i}$  depends on  $\alpha_{n_i, p_i + 1}$ . These values of  $\alpha$  and  $\beta$  form a triangle when arranged graphically by their coordinates, as shown in Figure 2. Because of the dependencies in the definitions of  $\alpha$  and  $\beta$ , only two horizontal rows of this triangle, the row representing the  $p_i$  being computed and the row for  $p_i + 1$ , must reside in working memory at a given time. As a row  $p$  is completed, each  $\beta_{n_i, p}$  is added to a cumulative value  $\sigma_{n_i}$ . After  $p = 1$  is computed,  $spt(n_i) = \sigma_{n_i}$ .



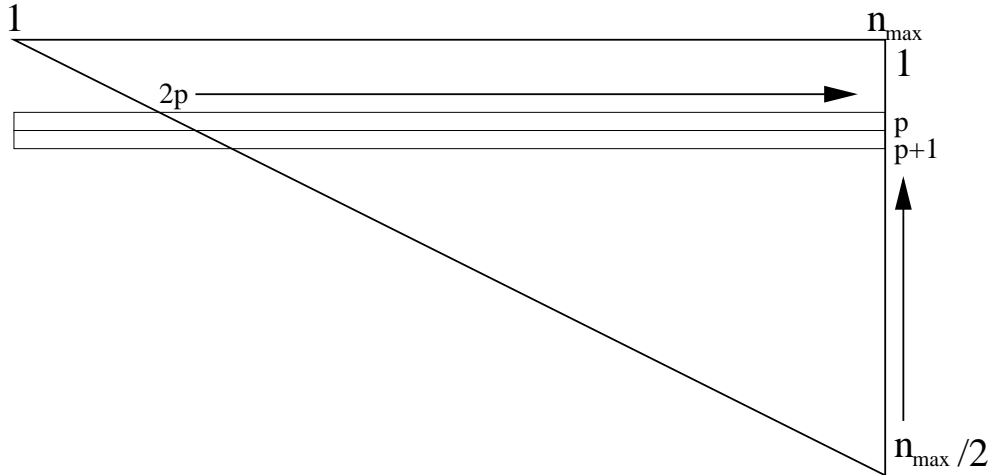


Figure 2: Memory triangle of  $\alpha, \beta$  pairs.

## 2.2 Complexity

The range of parameters for  $\alpha$  and  $\beta$  span two dimensions, where both dimensions are proportional to  $n_{max}$ , so computing  $\mathbb{S}$  takes  $\Theta(n^2)$  time in  $n_{max}$ . Because we are using this spt data to search for congruences with a modulus, we store all values modulo some fixed modulus, so any given value of  $\alpha$  or  $\beta$  occupies a constant amount of working memory. As only the sets of all  $\beta_{n_i,p}$ ,  $\beta_{n_p,p+1}$ , and  $\sigma_{n_i}$  are stored in working memory at a given time for some  $p \in \mathbb{P}$ , and all these sets are proportional to  $n_{max}$ , the amount of working memory used is  $\Theta(n)$  in  $n_{max}$ . This is a vast improvement over an exhaustive search method.

## 3 Parallelization

### 3.1 Motivation

Our algorithm generated the first million values of  $\text{spt}(n)$ , but this method is still limited to a single processor. We investigated ways of overcoming this next barrier. To encourage forays into the world of parallel computing, St. Olaf operates a small Beowulf cluster. This cluster is a collection of off-the-shelf computers connected by a gigabit Ethernet network. Programs written for this system typically use a message passing-based approach to parallelization. Any processor, called a node, may send a message to any other node containing arbitrary data. Each node assigned to run a particular program determines how many other nodes are running the program and its unique index in this group of nodes. As the speed of a gigabit Ethernet network is many orders of magnitude slower than the speed of each machine's internal bus, care must be taken to balance the cost of network communication with the cost of not sharing work among processors.

### 3.2 Parallel Algorithm

An ideal parallelization of our algorithm would evenly distribute the computation of all  $\alpha_{n_i, p_i}$  and  $\beta_{n_i, p_i}$ , represented by the triangle of Figure 2, among some number of nodes  $r > 1$ . The recursive dependencies in the definitions of  $\alpha$  and  $\beta$  make this distribution of work not ideal and non-trivial.

We first consider parallelizing the computation of a single row of Figure 2, that is, a range of  $n_i$  for a given  $p$ . We assume that each node already has all necessary  $\alpha_{n_i, p+1}$  values to compute  $\alpha_{n_i, p}$  in its local memory. To compute  $\alpha_{n_i, p}$  or  $\beta_{n_i, p}$ , a node must have the value of  $\alpha_{n_i-p, p}$  and  $\beta_{n_i-p, p}$ . The first  $p$  values of  $\alpha_{n_i, p}$  and  $\beta_{n_i, p}$  may be computed independent of any other value because they are constant base cases in the recurrence. If a node computes  $\alpha_{n_i, p}$  and  $\beta_{n_i, p}$ , it will have the necessary values to compute  $\alpha_{n_i+p, p}$  and  $\beta_{n_i+p, p}$ . Thus, we may divide the first  $p$  elements of  $\{n : 2p \leq n \leq n_{max}\}$  among  $r$  nodes so that each node computes  $\alpha_{n, p}$  and  $\beta_{n, p}$  for its assigned  $n \in \{n : 2p \leq n \leq n_{max}\}$  and all  $\alpha_{apn, p}$  and  $\beta_{apn, p}$  for each integer  $a$  such that  $apn \leq n_{max}$ . Figure 3 shows the values that a single node might compute on a single row. Each separate grouping of values, separated by a distance  $p$ , is called a *hop*, because a node hops between them without computing anything in between.

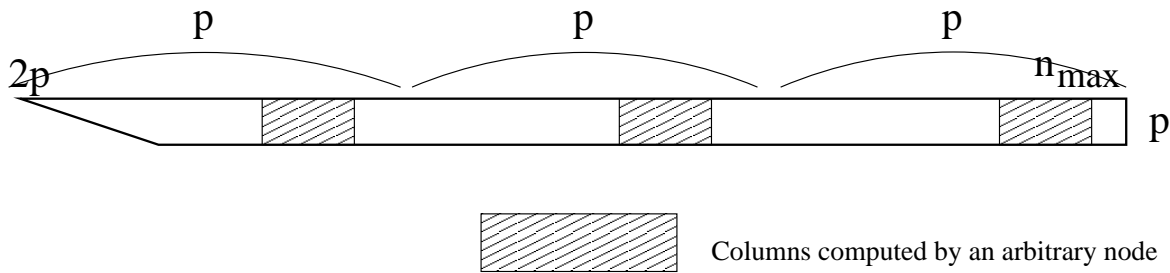


Figure 3: Values of  $\alpha, \beta$  computed by a single node on a single row.

Now we consider parallelizing the next row, for part  $p - 1$ . If the same distribution of values among nodes were used, each node would have the necessary  $\alpha_{n_i, p}$  values to compute  $\alpha_{n_i, p-1}$  in its local memory. However, the first  $p - 1$  elements of  $\{n : 2(p - 1) \leq n \leq n_{max}\}$  are not the first  $p$  elements of  $\{n : 2p \leq n \leq n_{max}\}$ . One solution to this problem would be for all nodes to share their computed values on each row with each other, but this would incur an extremely high communication cost.

We address this problem by synchronizing only a small fraction of rows, a subset  $\mathbb{T}$  of  $\{p : 1 \leq p \leq \lfloor n_{max}/2 \rfloor\}$  such that a constant  $s$  divides  $t \in \mathbb{T}$ . We call  $s$  a *stride length* because nodes stride by  $s$  rows before communicating with each other. If synchronizations are only made at rows in  $\mathbb{T}$ , then nodes will not have certain values in local memory needed to compute their assigned elements of  $\alpha$  and  $\beta$ . We remedy this problem by having each node compute  $\alpha_{n_i, p_i}$  and  $\beta_{n_i, p_i}$  for certain extra values of  $n_i$  for which it does not contribute to  $\sigma_{n_i}$ . At any row  $p_i$ , we assign a node to compute the values  $\alpha_{n_i, p_i}$  and  $\beta_{n_i, p_i}$  that it will be assigned to compute at the next row in  $\mathbb{T}$ . Rows in  $\mathbb{T}$  follow the distribution procedure

described above.

To determine the extra values that a node must compute, consider the hop on a row  $t \in \mathbb{T}$  that is closest to  $n_{max}$ . On row  $t$ , a node computing this hop will have already computed the values the hop depends on. But on the previous row,  $t + 1$ , the node would be missing a single value immediately preceding the previous hop, because of the dependencies in  $\alpha$  and  $\beta$  related to the index of the current row. This missing value would itself depend on another missing value two to the left of the next previous hop, and so on, such that the number of missing values preceding each hop is equal to the number of hops to the end of the row. On previous rows, furthermore, the number of these missing values is multiplied by the number of rows to  $t$ . These missing values must be computed by each node, and they form a triangular *wedge* preceding each hop. Figure 4 shows all the hops and wedges for an arbitrary node.

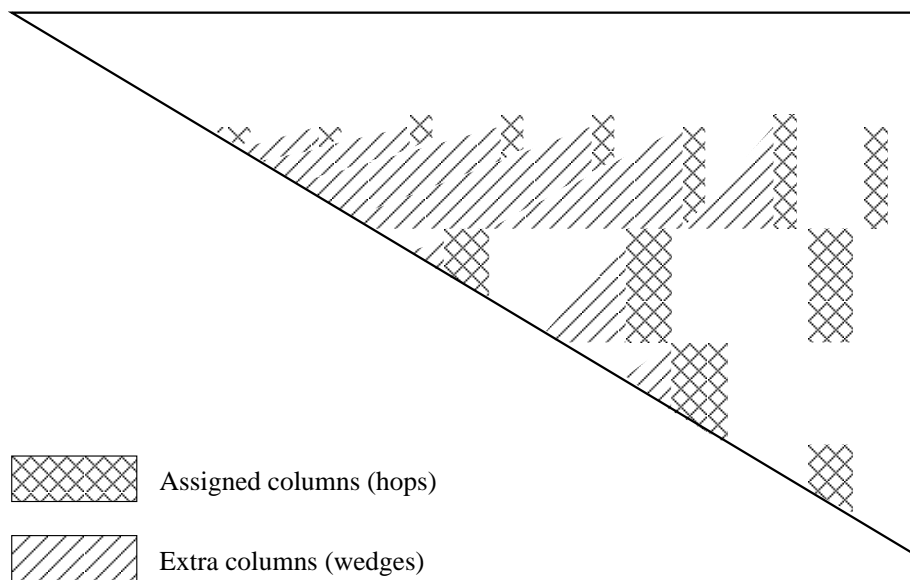


Figure 4: Hops and wedges of a single node.

When the value of  $s$  is decreased, the size of all wedges, wasted computation, decreases, but the communication frequency, and cost, increases. The parameter  $s$  represents the balance between communication cost and computation cost.

### 3.3 Performance Improvements

We empirically selected values for  $s$  and managed to see modest improvements in the wall clock running time of the algorithm over the serial version. For example, the serial algorithm ran on our hardware for  $n_{max} = 100,000$  in about 5 minutes. The parallelization ran in about 3.5 minutes with  $r = 5$  and  $s = 10,000$ . The algorithm's performance fails when nodes reach low values of  $p$ , which have large numbers of hops and therefore large wedges. Even with larger values of  $n_{max}$ , the parallelization never improved over the performance

of the serial algorithm by more than a quarter of the wall clock time. The largest data set feasible on our parallel implementation is only marginally larger than that of the serial implementation.

In the future, we hope to improve the performance of this parallelization by varying the value of  $s$  after each synchronization. We have timed the computation speed and communication speed of our implementation and could use this data to optimize  $s$  as nodes compute rows with high numbers of hops.

## 4 Conclusion

In developing this parallelization, we explored the tradeoff between gained efficiency and sunk communication costs from parallel computing, as applied to a new algorithm. The serial algorithm does not obviously lend itself to parallelization, but a compromise that wastes some computation time can still be made that improves overall performance. In the end, this parallelization was a useful and non-trivial exercise in the synchronization of independent processors. Measurable performance improvements demonstrated the validity of the parallelization approach, and potential minor modifications may reveal the algorithm to be useful for generating larger data sets than its serial counterpart.

## References

- [1] G. E. Andrews, *The number of smallest parts in the partitions of  $n$* , to appear.
- [2] K. Garrett, *New congruences for Andrews' SPT*, project description.
- [3] T. Frederick, M. Krahulec, C. McEachern, *Generating SPT values in quadratic time and linear memory with resulting conjectures*, in preparation.

# Applications of Beowulf Cluster Computing to Problems in Biology

Spencer Debenport  
St. Olaf College  
Class of 2010  
[debenpor@stolaf.edu](mailto:debenpor@stolaf.edu)

Dr. Richard Brown  
Computer Science  
St. Olaf College  
[rab@stolaf.edu](mailto:rab@stolaf.edu)

## Abstract

The field of Biology provides a large number of problems that can be applied to Beowulf cluster computing. Using the two clusters at St. Olaf College, projects including searching through *Tetrahymena thermophila* DNA, modeling nitrogen flow through riparian zones, and creating databases of neurological data were taken on in order to further interdisciplinary research. Over the course of our research we were very successful in laying the groundwork for this research to continue on in future years.

# 1 Introduction

The field of Biology provides many problems and issues which can be applied to Beowulf cluster computing. Genetics, ecological models, databases of neurological data; all of these are rich testing grounds for the applications of large scale cluster computing. Over the past summer I took on three projects in order to further integrate the fields of biology and computer science. Using these projects as examples, it should be clear that there is a large area of research that can be aided using Beowulf cluster computing.

## 2 Beowulf Cluster Computing

Beowulf is a design for high-performance parallel computing using inexpensive personal computer grade computer hardware. The idea behind a Beowulf design is that for a small amount of money, high performance can be obtained using only everyday hardware, software and networking. This way, the entirety of the cluster is replaceable and reproducible at very low costs. Additions to the cluster can be made very easily by simply adding another node to the network.

While there is no specified software to allow programmers to utilize a Beowulf cluster, the most common is the Message Passing Interface (MPI) or Parallel Virtual Machine (PVM). In our research we used MPI to perform communications throughout our cluster. A cluster application typically uses a head or control node which directs the actions of multiple computational nodes. Not every program can be easily coded to run efficiently on a Beowulf cluster. A program must be able to have its computations split evenly among the computational nodes. For example, in calculating the definite integral of a function using the trapezoid rule, the region under the graph of  $f(x)$  can be divided evenly among available nodes (perhaps including the head node) to carry out the calculations quickly.

At St. Olaf college, we have two Beowulf clusters at our disposal, referred to as the development and production clusters. The development cluster is composed of retired machines that we salvaged in order to have a good testing platform for our parallel programs. This allows us to keep our production cluster, composed primarily of server grade Sun machines, free to be used for research runs of finished programs.

## 3 Projects

### 3.1 Modeling Riparian Systems

A riparian zone is defined as the intersection of land and a body of water. Since they act as barriers to erosion and nutrient leakage, riparian zones are quite important junctions. More specifically, riparian zones between farmland and moving bodies of water such as rivers and streams are very important to monitor and maintain, as they allow for the absorption of chemicals in runoff that can be very dangerous for aquatic life.

This project consisted of modeling the flow of nitrogen through one such riparian zone using Beowulf computing. A model, designed by Professor John Schade (Environmental Science) and implemented in a program written by student Tony Waldschmidt at St. Olaf College, outlined all of the equations and computations needed to represent the complete nitrogen flow inside of a riparian plant [1]. This included the initial concentration of nitrogen found in water entering the plant, the uptake needed to perform reactions necessary for the life of the plant, denitrification or chemical reduction of organic nitrogen by the plant, and finally the amount of nitrogen released back into the surrounding water. Since this model used a flowing river as its water source, we can reasonably assume that the nitrogen export of one plant would make up the input value of the next plant downstream. This allowed for an interesting use of the Beowulf cluster, by creating a system of plant models.

Each plant model in this system needed to perform a number of computations in order to determine the amount of nitrogen absorbed and the amount exported. To fully use the capacity of the cluster, I had each node of the cluster act as an individual plant, performing all of the computations needed to determine nitrogen export, and then passing on its export value to the next plant model downstream in the system. This allowed for each node to perform calculations simultaneously, with very little delay waiting for a new input value from another node.

I found that there was a simple linear trend in the number of plants needed to reduce the concentration of nitrogen in field runoff to a safe level for aquatic organisms. This was to be expected using our current model as each plant was using the same equations as all of the others. Currently more work is being done to form models that correspond to specific plants in order for real world applications to be possible.

### **3.2 Patterns in *Tetrahymena thermophila* DNA**

Deoxyribose nucleic acid (DNA) plays the very important role of serving as the template upon which every protein in our body is based upon. DNA consists of a double stranded helix composed of sugars, phosphate groups and nucleotides. It is these nucleotides that serve as the code for creating proteins.

Some details about how cells create proteins from DNA will help in explaining our *Tetrahymena thermophila* application. Each DNA strand has directionality to it, being read in what is known as “the 5' to 3' direction.” These strands exist in opposite parallel, with the 5' end of one strand aligned with the 3' end of the other strand. In forming proteins, only one of the two strands is transcribed into RNA, and eventually used to construct a protein. A start codon is defined as a triplet of nucleotides that signals the start of a protein while a stop codon is a triplet that signals the end of a protein. A gene is considered to be a region of the DNA that codes for a protein, although through the a process known as splicing out introns, instructions for building multiple proteins may be coded using only one gene.

Professor Eric Cole (Biology) believed that he observed a phenomenon when looking at

the DNA sequence for a histone gene in *Tetrahymena thermophila*. He saw that opposite the coding region of this gene, on the reverse direction of the other strand, there seemed to be another coding region beginning with a promoter region and a start codon with no immediate stop codon. This is quite peculiar since only one strand of the DNA in a gene is thought to code for a protein. Even stranger, this opposite region seemed to have a similar sequence to the strand that it was bound to, meaning that the gene was mildly palindromic. Was this apparent “reversible coding region” actually a phenomenon, or was this simply there by chance?

Professor Steven Freedberg (Bioinformatics) wrote a PERL script to generate random genome patterns typically found in *Tetrahymena thermophila* and check the reverse opposite strands of coding regions, to see just how common this event was. I modified this script in order to make dozens of runs on our Beowulf cluster, with each node creating and analyzing a unique genome so that we could harvest large quantities of data in a very short amount of time.

With the amount of data we collected over our two month session, I found that the likelihood of this event occurring by random chance was unfortunately high. This means that it is highly possible that the example observed by Professor Cole is indeed a random event instead of an evolutionary phenomenon. There is still more data to be collected and perhaps some alterations to the PERL script are necessary before we can assuredly say that there is nothing of relevance to be gained from this observation.

### **3.3 Creating a Database of S-Cell Neuron Models**

The S-cell neuron in the leech *Hirudo medicinalis* is considered to be the single cell that controls the organism's learning. This one neuron allows for the leech to react to a stimulus according to past experiences with that stimulus. If this cell is removed from the leech, the organism will no longer 'learn' from experiences with stimuli, but will instead have the same reaction to a stimuli every time. For example, if a leech with it's S-cell intact is touched on the anterior end gently, at first it will pull away quickly, but upon multiple occasions of being poked with no harm done to the leech it will pull away less and less every time. Conversely, if that leech's S-cell is removed, the leech will have the same harsh reaction every time it is poked, even though no harm is being done to it.

Professor Kevin Crisp (Neuroscience) seeks to create a database of computational models for the S-cell, using that neuron's action potential as a partial key, together with graphical information about firing behavior for each model. Once this database has been constructed, Professor Crisp will be able to search it for graphical matches to firing patterns he encounters in his research lab, yielding candidate computational model parameters for the biological neuron he is studying in the lab. By simply dividing the broad range of possible input values by the number of available Beowulf nodes, this database can be calculated and formed in a fraction of the time it would take with a different computational architecture.

This S-cell database project is one of several Beowulf computing initiatives in progress that support Professor Crisp's neuroscience research.



## 4 Conclusion

In just this small sampling of projects, it is easy to see that there are quite a few projects in Biology that can utilize Beowulf cluster computing. The general outlines of each of these projects (genomics, nutrient balance models, and database creation) can be applied to an endless number of questions that need to be answered using only a fraction of the time needed with a different computational architecture. While much work still needs to be done with these projects, they have laid the groundwork for a large quantity of future work.

## References

[1] SCHADE, J. D., AND LEWIS, D. B. Plasticity in resource allocation and nitrogen-use efficiency in riparian vegetation: Implications for nitrogen retention. *Ecosystems* 9 (August 2005), 740–755.

# Contemporary Technologies and Platforms for Electronic and Mobile Commerce System Construction

Wen-Chen Hu

Department of Computer Science  
University of North Dakota  
Grand Forks, ND 58202-9015  
[wenchen@cs.und.edu](mailto:wenchen@cs.und.edu)

Yanjun Zuo

Department of Information  
Systems and Business Education  
University of North Dakota  
Grand Forks, ND 58202-8363  
[yanjun.zuo@und.nodak.edu](mailto:yanjun.zuo@und.nodak.edu)

Lei Chen

Department of Computer Science  
Sam Houston State University  
Huntsville, TX 77341  
[LXC008@shsu.edu](mailto:LXC008@shsu.edu)

Anusha Gopalakrishnan

Department of Computer Science  
University of North Dakota  
Grand Forks, ND 58202-9015  
[anusha.g85@gmail.com](mailto:anusha.g85@gmail.com)

## Abstract

The emergence of wireless and mobile networks has made possible the introduction of electronic commerce to a new application and research subject: mobile commerce. Understanding or constructing a mobile or an electronic commerce system is an arduous task because the system involves a wide variety of disciplines and technologies and the technologies are constantly changing. To facilitate understanding and constructing such a system, this article divides the system into six components: (i) applications, (ii) client computers or devices, (iii) mobile middleware, (iv) wireless networks, (v) wired networks, and (vi) host computers. Elements in these components specifically related to the subject are described in detail and lists of current technologies for component construction are discussed. Another important and complicated issue related to the subject is the mobile and electronic commerce application programming. It includes two types of programming: client-side and server-side programming, which will be introduced too.

# 1 Introduction

With the introduction of the World Wide Web, electronic commerce revolutionized traditional commerce, boosting sales and facilitating exchanges of merchandise and information. The emergence of wireless and mobile networks has now made it possible to extend electronic commerce to a new application and research area: mobile commerce, defined as the exchange or buying and selling of commodities, services, or information on the Internet through the use of mobile handheld devices. In just a few years, mobile commerce has become the hottest new trend in business transactions. The future of mobile commerce is bright, as shown by the following predictions:

- The dramatic growth in demand for smart mobile devices, specifically handhelds, wireless handhelds, and smart cellular phones, through 2007 is shown in Figure 1 (Canalys, 2004, 2005, 2006, & 2008).

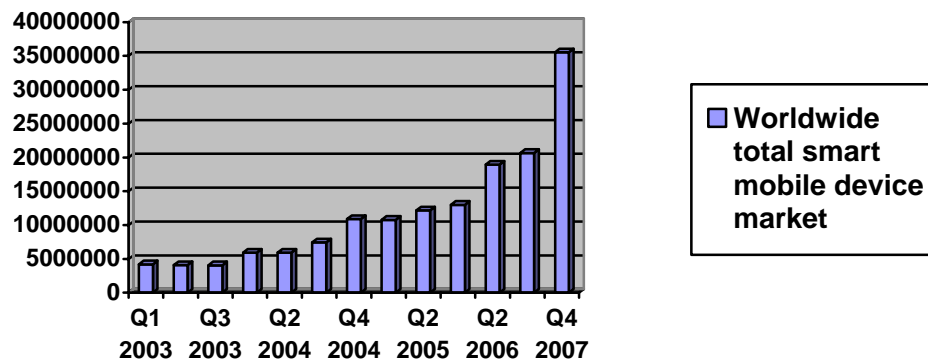


Figure 1: Worldwide total smart mobile device market.

- Cumulative sales of smartphones will reach 1 billion units by the first quarter of 2011 according to IDC, a market research company (Symbian Limited, 2006).
- Estimated worldwide shipments of the following four types of devices in 2007 were
  - o *Cellular phones*: 1.12 billion (Strategy Analytics, 2008),
  - o *Laptops*: 110 million laptops shipped in 2007, 33.8% growth from 2006 (IDC, Corp., 2008),
  - o *PCs*: 160 million desktop computers shipped in 2007, 4.3% growth from 2006 (IDC, Corp., 2008), and
  - o *Smartphones*: 118 million, up 53% from 2006 (Canalys, 2008).

Mobile commerce is an effective and convenient way of enabling consumers to engage in electronic commerce whenever they wish and from wherever they happen to be. Realizing the advantages to be gained from mobile commerce, many retail companies have begun to offer mobile commerce options for their customers to supplement the electronic commerce service they already provide (Yankee Group, 2001). However, a tremendous effort is required to understand and construct a mobile commerce system because it involves such a wide range of disciplines and technologies. To address these difficulties,

this chapter will separate mobile commerce systems into six components: (i) mobile commerce applications, (ii) mobile handheld devices, (iii) mobile middleware, (iv) wireless networks, (v) wired networks, and (vi) host computers. However, since each of these components is sufficiently complex to be a research area in its own right, only the elements specifically related to mobile commerce will be explained in detail. Lists of the technologies used for component construction are given in the first part of this book and other important issues, such as mobile security, are also discussed. Related research on mobile commerce systems can be found in the excellent article by Varshney, Vetter, & Kalakota (2000).

## 2 System Structures

This section illustrates the system structures of electronic and mobile commerce and explains the procedures of mobile commerce transactions. A modular approach will be used to study the systems.

### 2.1 An Electronic Commerce System Structure

Electronic commerce describes the manner in which transactions take place over networks, mostly the Internet. It is the process of electronically buying and selling goods, services, and information. An electronic commerce system is inherently interdisciplinary and there are many different ways to implement it. Figure 2 shows the structure of a traditional electronic commerce system and a typical example of such a system. The system structure includes four components, some of which are at least partly shared by mobile commerce systems: (i) electronic commerce applications, (ii) client computers, (iii) wired networks, and (iv) host computers.

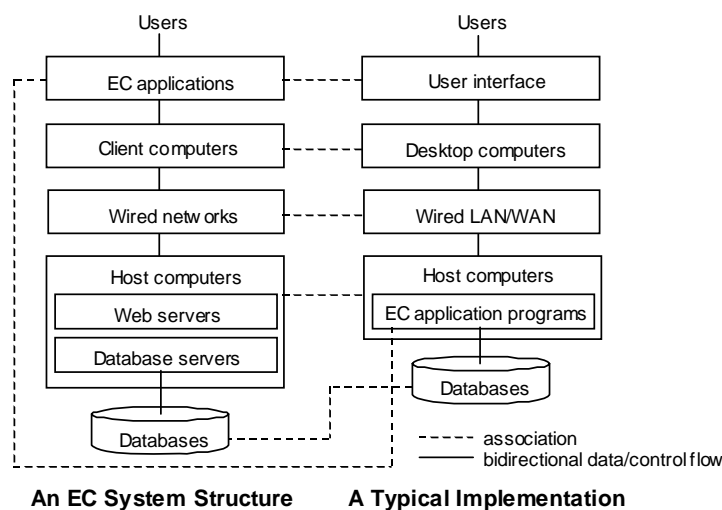


Figure 2: An electronic commerce system structure.

## 2.2 A Mobile Commerce System Structure

Much like the concept of e-commerce, mobile commerce is a type of business conducted 100% electronically through computer networks; m-commerce is a concept of buying and selling goods and services via wireless networks with a mobile device. Compared to an electronic commerce system, a mobile commerce system is much more complicated because components related to mobile computing have to be included. Figure 3 shows the structure of a mobile commerce system and an example of such a system that is currently possible based on the existing technologies (Hu, Lee, & Yeh, 2004). The system structure includes six components: (i) mobile commerce applications, (ii) mobile handheld devices, (iii) mobile middleware, (iv) wireless networks, (v) wired networks, and (vi) host computers. The network infrastructure for mobile commerce systems consists of mobile middleware and wired & wireless networks. The wired networks component has the same structure and implementation as that needed by an electronic commerce system.

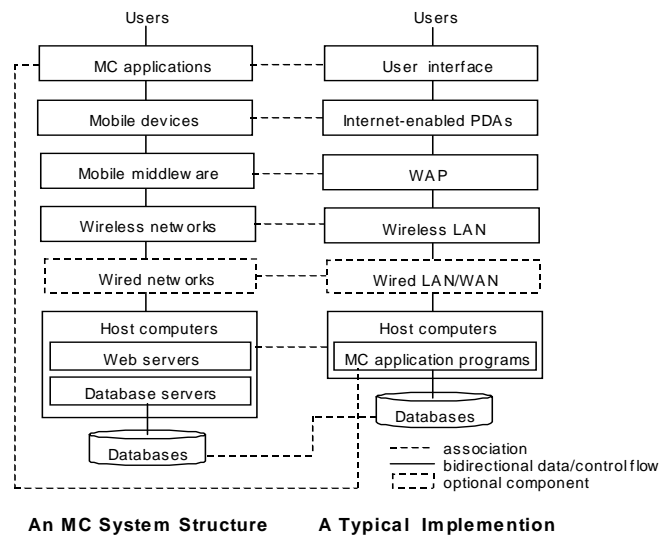


Figure 3: A mobile commerce system structure.

## 3 Applications

The emergence of electronic and mobile commerce creates numerous business opportunities and applications. Electronic commerce, defined as the buying and selling of goods and services and the transfer of funds through digital communications, includes a wide variety of applications, such as auctions, banking, marketplaces and exchanges, recruiting, and retailing, to name but a few. Mobile commerce applications not only cover the electronic commerce applications, but also include new applications, e.g., mobile inventory tracking, which can be performed at any time and from anywhere by using mobile computing technology.

### 3.1 Electronic Commerce Applications

This sub-section discusses some new business models, which were not seen before, created by electronic commerce. Other than the “buy-and-sell” model, the following list gives some other common models created by e-commerce (Turban, et al, 2004):

- *Affiliate marketing*: Affiliate marketing is a marketing method, which allows other Websites to receive a commission by selling your products or services. For the example of Amazon.com’s Associates Program, the associates drive Internet traffic to Amazon through specially formatted links that allow Amazon to track sales and other activities. The partners can receive up to 10% in referral fees on all qualifying revenue made through their links to Amazon’s products and services.
- *Comparing prices*: This method presents a list of services or products based on a consumer’s specifications. mySimon.com is a comparison shopping site for apparel, computers, electronics, jewelry, video games, and more. It gathers prices on millions of products from thousands of stores, so customers can compare products and find the best price before he or she buys.
- *Customization and personalization*: Customization or personalization is to design and creation of content that meets a customer’s specific needs. For example, instead of picking one from few standard models, Dell customers can specify their requirements such as memory sizes and CPU models and Dell will build the systems based on their specifications.
- *Electronic marketplaces and exchanges*: Electronic marketplaces are Internet Websites acting as a meeting point between supply and demand and electronic exchanges are a central marketplace with established rules and regulations where buyers and sellers meet to trade futures and options contracts or securities. Electronic marketplaces and exchanges provide benefits to both buyers and sellers because they are more efficient than traditional ones.
- *Electronic tendering systems*: Tendering is potential suppliers bid competitively for a contract, quoting a price to the buyer. Large buyers usually make their purchases through a tendering (bidding) system, which is more effective and efficient with the help of electronic commerce.
- *Group purchasing*: Large-quantity purchasing usually receives lower prices than small-quantity purchasing does. Electronic commerce allows a group of customers or organizations to place their orders together and negotiate for a better deal.
- *Name your price*: With this model, the product or service prices are set by customers instead of sellers. Priceline.com is the first company applying this method. With Priceline's “Name Your Own Price” hotel reservation service, customers choose the star level of hotel they want, along with the desired neighborhood, dates and price they want to pay. Priceline then works to find a hotel room at the customer's desired price. Customers learn the specific hotel name and location after the purchase is completed.
- *Online auctions*: Traditional auctions usually require bidders to attend the auctions, whose items are limited. Online auctions allow bidders from everywhere to bid products or services provided by various sellers without needing to show up. eBay.com is the world's largest online auction site. It offers an online platform where millions of items are traded each day.

### 3.2 Mobile Commerce Applications

Mobile commerce applications cover almost everything in our daily lives such as traveling and foods. Table 1 lists some major mobile commerce applications along with explanations of one application related to map services (Sadeh, 2002).

Mobile Category	Major Applications	Sponsors	Clients
Advertising	Targeted ads and location-based ads	Business	Travelers
Education	Mobile classrooms and labs	Schools and training centers	Students
Entertainment	Games/images/music/video downloads and on-line gaming	Entertainment industry	All
Health care	Accessing and updating patient records	Hospitals and nursing homes	Patients
Inventory tracking and dispatching	Product tracking and dispatching	Delivery services and transportation	All
Retailing	Paying at vending machines, and checking product prices/information	Retailers	All
Traffic	Global positioning, routing services, toll/parking paying, and traffic advisories	Transportation and auto industry	Drivers
Travel and weather	Reservation services	Airlines, hotels, and travel agencies	Travelers

Table 1: Major mobile commerce applications.

Map services provide various useful functions to mobile users. Some of the functions include:

- *Directions*, which are driving/walking directions from the starting location to destination,
- *Maps*, which include traditional clear maps,
- *Local hangouts and businesses recommendations*, which provide suggestions for restaurant/gas-station/grocery-store/movie-theater, and
- *Satellite imagery*, which includes real images from satellites.

A few mobile map services are available. Google Maps for Mobile (n.d.) lets users find local hangouts and businesses across town or across the country—right from their phones. Figure 4 shows three screenshots from the Google’s map services where

- a clear map of the location with a postal code 58202,
- directions from the postal code 58201 to 58203, and
- a satellite map of (b) and a menu.

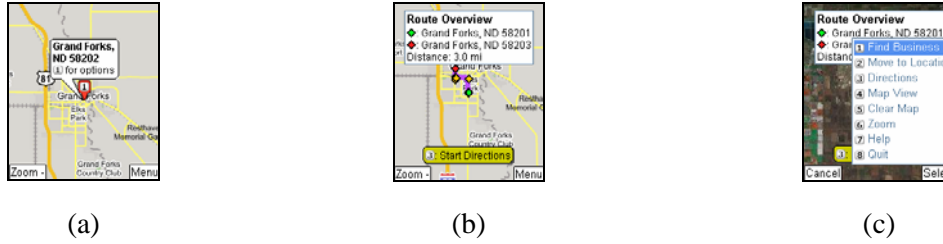


Figure 4: Screenshots of the Google’s map services showing (a) a clear map, (b) directions, and (c) a satellite image of (b) and a menu.

### 4 Client Computers or Devices

Desktop and laptop computers are on the client-side of electronic commerce systems, whereas mobile handheld devices are for mobile commerce systems. An Internet-enabled mobile handheld device is a small general-purpose, programmable, battery-powered computer that is capable of handling the front end of mobile commerce applications and can be operated comfortably while being held in one hand. It is the device via which mobile users interact directly with mobile commerce applications. The differences between these two client machines are given in Table 2. There are other kinds of computers such as tablet computers, which are a special kind of PCs.

	Desktop and Laptop Computers	Mobile Handheld Devices
<i>Browser</i>	Desktop browsers	Microbrowsers
<i>Functions</i>	Full	Limited
<i>Major Input Methods</i>	Keyboards and mice	Stylus and soft keyboards
<i>Major Output Methods</i>	Screens and printers	Screens
<i>Mobility</i>	Low	High
<i>Networking</i>	Wired	Wireless and mobile
<i>Transmission Bandwidth</i>	High	Low
<i>Power Supply</i>	Electrical outlets	Batteries
<i>Screen</i>	Normal	Small
<i>Size</i>	Desktop	Handheld
<i>Weight</i>	Normal	Light

Table 2: Differences between desktop & laptop computers and handheld devices.



## 4.1 Client-Side Programming

Mobile or electronic commerce application programming involves a variety of technologies and languages. It consists of two kinds of programming:

- *Client-side programming*, which is to develop software running on client computers or devices. It is mostly related to web interface construction. Popular languages for web interface construction include CSS, DOM, (X)HTML, JavaScript, WML, WMLScript, XML, XSL(T), etc. Other than web interface construction, client-side programming can be used to build client-side applications such as address and schedule books.
- *Server-side programming*, which is to develop software running on servers. The software normally receives requests from browsers and sends the results from databases/files/programs back to the browsers for display. Popular server-side languages include C/C++, Java, Perl, PHP, etc. Other than web applications, it can be used to implement numerous applications such as instant messaging and telephony.

This sub-section discusses web interface construction. The server-side programming will be covered in the section of Host Computers. Other than building a web system from scratch by using various languages and tools, some common software packages are available for developing web applications easily and quickly. Those packages can be divided into three categories: (i) multimedia editors, (ii) HTML editors, and (iii) integrated development environments (IDEs):

- *Multimedia editors*, which are used to create, edit, and post animation, audio, images, and videos on web pages. Adobe Systems, Inc. provides two popular multimedia editors:
  - *Flash*, which is an authoring environment for creating animation, advertisements, various web-page components, to integrate video into web pages, and more recently, to develop rich Internet applications. Flash Professional is an IDE while Flash Player is a virtual machine used to run, or parse, the Flash files.
  - *Photoshop*, which is image-editing and graphics creation software.
- *HTML editors*, which are used to create static web pages. Three popular HTML editors are
  - *Adobe Dreamweaver*, which is WYSIWYG (What You See Is What You Got) authoring software that allows web developers to generate HTML and JavaScript source code while viewing the site as they work.
  - *Microsoft Expression Web*, which is a design tool to create sophisticated standards-based web sites. It combines both FrontPage and Visual Studio technologies in a new user interface for creating XHTML, CSS, XML, XSLT, and ASP.NET 2.0. Where appropriate, the user interface and features of Expression Web and Visual Studio are identical.
  - *Microsoft SharePoint Designer*, which will enable information workers to develop applications and solutions on top of the SharePoint platform to enable organizational agility, business process automation, and get the value of Microsoft Office applications on the SharePoint platform.

The category of integrated development environments (IDEs) will be covered in the section of Host Computers.

## 5 Mobile Middleware and Wireless Networks

Mobile middleware and wireless networks are for mobile commerce systems only. The mobile middleware is optional, but the system will be greatly simplified with it. A mobile commerce system is already complicated enough. Without mobile middleware, the mobile system becomes even more complicated.

### 5.1 Mobile Middleware

The term middleware refers to the software layer between the operating system and the distributed applications that interact via the networks. The primary mission of a middleware layer is to hide the underlying networked environment's complexity by insulating applications from explicit protocols that handle disjoint memories, data replication, network faults, and parallelism (Geihs, 2001). The major task of mobile middleware is to seamlessly and transparently map Internet contents to mobile handheld devices that support a wide variety of operating systems, markup languages, microbrowsers, and protocols. WAP and i-mode are the two major kinds of mobile middleware:

- *WAP (Wireless Application Protocol)*, which is a secure specification that allows users to access information instantly via mobile handheld devices such as smart phones and PDAs (Open Mobile Alliance Ltd., n.d.). WAP supports most wireless networks including CDPD, CDMA, GSM, PDC, PHS, TDMA, FLEX, ReFLEX, iDEN, TETRA, DECT, DataTAC, and Mobitex. WAP is supported by all operating systems.
- *i-mode*, which is a mobile Internet service that has caused a revolution in both business and private lifestyles in Japan (NTT DoCoMo, Inc., 2007). 46 million subscribers have been attracted to this service since its debut in February 1999 and currently more than 95,000 Internet sites are providing a variety of contents.

Table 3 compares i-mode to WAP.

	<b>WAP</b>	<b>i-mode</b>
<i>Developer</i>	Open Mobile Alliance	NTT DoCoMo
<i>Implementation</i>	A protocol	A complete mobile Internet service
<i>Web Language</i>	WML (Wireless Markup Language)	CHTML (Compact HTML)
<i>Major Technology</i>	WAP Gateway	TCP/IP development
<i>Key Features</i>	Widely adopted and flexible	Highest number of users and easy to use

Table 3: A comparison between the two major types of mobile middleware.

## 5.2 Wireless Networks

Wireless communication capability supports mobility for end users in mobile commerce systems. Wireless LAN, MAN, and WAN are the major components used to provide radio communication channels so that mobile service is possible. The wireless telephone technology includes several generations as follows:

- *0G (1945-1973)*, which refers to mobile radio telephone systems.
- *1G (1980s)*, which is analog cellphone standards including NMT and AMPS.
- *2G (1990s)*, which is digital cellphone standards divided into TDMA-based and CDMA-based standards depending on the type of multiplexing used.
- *2.5G (late 1990s)*, which is implemented a packet switched domain in addition to the circuit switched domain.
- *3G (early 2000s)*, which includes wide-area wireless voice telephony and broadband wireless data, all in a mobile environment.
- *4G (2000s)*, which provides end-to-end IP solution where voice, data and multimedia streaming can be served at higher data rates with anytime-anywhere concept.

A wide variety of technologies and standards for wireless telephones are available. Some of the major ones include:

- *CDMA (Code Division Multiple Access)*, which is based on a spread spectrum method. The method transmits a signal by “spreading” it over a broad range of frequencies.
- *GSM (Global System for Mobile communications)*, which is one of the most popular standards for mobile phones and is specifically developed to provide system compatibility across country boundaries, especially the Europe.
- *IEEE 802.11*, which includes an encryption method, the Wired Equivalent Privacy algorithm. WLAN (Wireless Local Area Network), based on 802.11, allows a mobile user connecting to a local area network (LAN) through a wireless (radio) connection.
- *IEEE 802.16*, which ensures compatibility and interoperability between broadband wireless access equipment. WiMAX (Worldwide Interoperability for Microwave Access), based on 802.16, provides wireless data over long distances, in a variety of different ways, from point to point links to full mobile cellular type access.

Table 4 shows major technologies and standards used in the wireless telephone generations.

	<b>2G</b> (10 Kbps – 40 Kbps)	<b>2.5G</b> (20 Kbps – 171 Kbps)	<b>3G</b> (60 KBps – 54 Mbps)	<b>4G</b> (50 Mbps – 1 Gbps)
<i>CDMA track</i>	IS-95	CDMA 2000	W-CDMA	UMTS Revision 8 (LTE)
<i>GSM track</i>	GSM	GPRS	EDGE	
<i>IEEE 802.11 track</i>			Wi-Fi	
<i>IEEE 802.16 track</i>				WiMAX

Table 4: Wireless telephone technology evolution.

## 5.3 Wired Networks

Wired networks are used to transmit data for electronic and mobile commerce. This component is a requirement for electronic commerce, but not necessary for mobile commerce, though mobile commerce would be greatly benefited by applying wired networks to its data communication because data transmission using wireless networks is more expensive than using wired networks. Among several types of wired networks, three major types are

- *Local Area Network (LAN)*, which spans a relatively small space of only a few square kilometers or less such as an office building. It generally offers a throughput of 10 Mbps or 100 Mbps and is usually based on Ethernet technology.
- *Metropolitan Area Network (MAN)*, which spans a geographical area greater than an LAN but less than a WAN such as few city blocks or a whole city. MAN typically uses wireless infrastructure or optical fiber connections to link its sites and it may connect multiple LANs together.
- *Wide Area Network (WAN)*, which spans a wide geographic area, such as state or country, and uses specialized computers to connect smaller networks, such as LANs. It generally offers a throughput of 1.5 Mbps or more. Two examples of WAN are the Internet, the largest network in the world, and an airline corporation using WAN to connect its offices around the world.

## 6 Host Computers

This component is similar for both electronic and mobile commerce systems because host computers are usually not aware of the differences among the targets, browsers or microbrowsers, they serve. A user request such as checking out or adding items to the shopping cart is actually processed at a host computer, which contains three major kinds of software specifically for electronic or mobile commerce transactions: (i) web servers, (ii) databases and database servers, and (iii) application programs and support software. Figure 5 shows a structure of three-tiered client-server web systems.

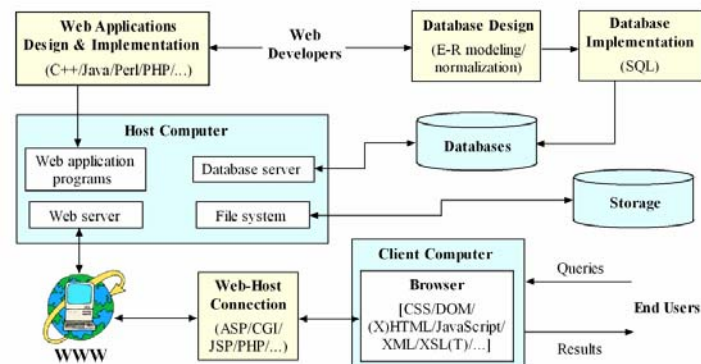


Figure 5: Three-tiered client-server web system structure.

## 6.1 Web Servers

A web server is a server-side application program that runs on a host computer and manages the web pages stored on the web site's databases or files. There are several kinds of web server software including public domain software from NCSA and Apache, and commercial packages from Microsoft, Netscape, and others. Three popular web servers are

- *Apache HTTP servers*, which are a collaborative software development effort aimed at creating a freely-available source code implementation of an HTTP (web) server. They are jointly managed by a group of volunteers located around the world. Since April 1996, Apache has been the most popular HTTP server on the Internet. It was developed in early 1995 based on code and ideas found in the most popular HTTP server of the time, NCSA httpd 1.3 (Apache Software Foundation, n.d.).
- *Microsoft's Internet Information Services (IIS)*, which provide a web application infrastructure for all versions of Windows servers (Microsoft, Corp., n.d). It is the world's second most popular web server after Apache.
- *Sun Java System Web Servers*, which provide organizations with a single deployment platform for web services, JavaServer Pages (JSP), Java Servlet technologies, NSAPI and CGI (Sun Microsystems, Inc., n.d.). They also offer built-in HTTP reverse-proxy capabilities to provide a highly scalable HTTP front-end to application servers or other HTTP origin servers.

## 6.2 Database Servers

A database server manages database access functions, such as locating the actual record being requested or updating the data in databases. Some popular databases include:

- *IBM DB2*: DB2 9 is a hybrid data server with management of both XML and relational data. It includes the following major features:
  - o XML data store,
  - o integration with relational data,
  - o Eclipse-base developer workbench, and
  - o integration with leading application infrastructures like PHP, Java, and .NET
- *Microsoft*: Microsoft provides two kinds of databases: (i) *Access* for desktop computers and (ii) *SQL Server* for the server engines in client-server solutions:
  - o *Access*: The Microsoft Access is a full-featured multi-user relational database management system that designed for the Microsoft Windows operating systems. It makes extensive use of drag-and-drop and visual design for queries, forms, and reports.
  - o *SQL Server*: The SQL Server is a comprehensive database software platform providing enterprise-class data management and integrated business intelligence (BI) tools. The SQL Server data engine lies at the core of this enterprise data management solution.
- *MySQL*: MySQL is an open-source, multithreaded, multi-user SQL relational database management system. It is used in more than 11 million installations ranging from large corporations to specialized embedded applications. Not only is MySQL

the world's most popular open source database, it is a key part of LAMP (Linux, Apache, MySQL, PHP/Perl/Python), a fast growing open source enterprise software stack. More and more companies are using LAMP as an alternative to expensive proprietary software stacks because of its lower cost and freedom from lock-in.

- *Oracle databases*, whose newest version is Oracle10g. The following list shows the Oracle database migration (Oracle, n.d.):
  - o *Oracle7.2*, which is a client-server based relational database management system (RDBMS). The query language is based on SQL.
  - o *Oracle8i*, which is an RDBMS with object capabilities included. Java has been added to the database capabilities.
  - o *Oracle9i*, which features full XML database functionality with the new Oracle XML DB feature, and other improvements.
  - o *Oracle 10g*, which is the first database designed for enterprise grid computing. Grid computing provides an environment in which individual users can access computers, databases, and experimental facilities simply and transparently, without having to consider where those facilities are located.

Other than the server-side database servers, a growing trend is to provide a client-side mobile database or an embedded database to a handheld device with a wide range of data-processing functionality. Some leading embedded-databases are Progress Software databases, Sybase's Anywhere products, and Ardent Software's DataStage (Ortiz, 2000).

### 6.3 Application Programs and Support Software

Application programs and support software are responsible for handling server-side processing. Three generations of programming languages and environments are used for server-side web application development:

1. *1<sup>st</sup> generation*: Traditionally, conventional programming languages such as C/C++ and Java are used for web development.
2. *2<sup>nd</sup> generation*: Dynamic programming languages such as Perl and PHP gradually replace conventional languages for web development. A dynamic language basically enables programs that can change their code and logical structures at runtime, adding variable types, module names, classes, and functions as they are running. These languages frequently are interpreted and generally check typing at runtime.
3. *3<sup>rd</sup> generation*: Recently, web development uses a couple of IDEs (Integrated Development Environments) including: (i) Adobe ColdFusion, (ii) Microsoft ASP.NET, (iii) Microsoft Visual Studio, (iv) NetBeans IDE, (v) Ruby On Rails (ROR), (vi) Sun Java Studio IDE, and (vii) Zend Core.

## 7 Summary

The emerging wireless and mobile networks have extended electronic commerce to another research and application subject: mobile commerce. A mobile or an electronic commerce system involves a range of disciplines and technologies. This level of complexity makes understanding and constructing such a system an arduous task. To

facilitate this process, this article divided a mobile or an electronic commerce system into six components, which can be summarized as follows:

1. *Applications*: Electronic commerce applications are already broad. Mobile commerce applications not only cover those applications, but also include new applications, which can be performed at any time and from anywhere by using mobile computing technology.
2. *Client computers or devices*: Desktop and notebook computers are for electronic commerce and mobile handheld devices, including smart cellular phones and PDAs, are used to perform mobile transactions. Numerous mobile devices are available in the market, but most use one of three major operating systems: Palm OS, Microsoft Windows Mobile, and Symbian OS. At this moment, Symbian OS leads the market, although it faces a serious challenge from Windows Mobile.
3. *Mobile middleware (mobile commerce systems only)*: Mobile middleware is used to facilitate mobile communication. It is not required for mobile commerce systems, but it can greatly reduce the complication of mobile communication. WAP and i-mode are the two major kinds of mobile middleware. WAP is widely adopted and flexible, while i-mode has the highest number of users and is easy to use.
4. *Wireless networks (mobile commerce systems only)*: Wireless communication capability supports mobility for end users in mobile commerce systems. Wireless LAN, MAN, and WAN are major components used to provide radio communication channels so that mobile service is possible. In the WLAN category, the Wi-Fi standard with 11 Mbps throughput dominates the current market. It is expected that standards with much higher transmission speeds, such as IEEE 802.11a and 802.11g, will replace Wi-Fi in the near future.
5. *Wired networks*: This component is a requirement for electronic commerce systems, but not necessary for mobile commerce systems, though mobile commerce systems will be greatly benefited by applying wired networks to its data communication because data transmission using wireless networks is more expensive than using wired networks. Among several types of wired networks, three major types are (i) LAN (Local Area Network), (ii) MAN (Metropolitan Area Network), and (iii) WAN (Wide Area Network) based on the sizes of their covering areas.
6. *Host computers*: Host computers process and store all the information needed for mobile and electronic commerce applications, and most application programs can be found here. They include three major components: (i) web servers, (ii) database servers, and (iii) application programs and support software.

Another important issue about mobile and electronic commerce systems is application programming. Electronic and mobile commerce programming, involving a wide variety of technologies and languages, consists of two kinds of programming:

- *Client-side programming*, which is to develop software running on client computers or devices. It is mostly related to web interface construction. The popular languages for web interface construction include CSS, DOM, (X)HTML, JavaScript, WML, WMLScript, XML, XSL(T), etc.
- *Server-side programming*, which is to develop software running on servers. The software normally receives requests from browsers and sends the results from databases/files/programs back to the browsers for display. The popular server-side languages include C/C++, Java, Perl, PHP, etc.

## References

- Apache Software Foundation. (n.d.). *Apache HTTP Server Project*. Retrieved June 21, 2007, from <http://httpd.apache.org/>
- Canalys. (2004). *Global Smart Phone Shipments Treble in Q3*. Retrieved December 3, 2006, from <http://www.canalys.com/pr/2004/r2004102.pdf>
- Canalys. (2005). *Global Smart Mobile Device Sale Surge Past 10 Million in Quarter*. Retrieved April 25, 2006, from <http://www.canalys.com/pr/2005/r2005041.pdf>
- Canalys. (2006). *Smart Mobile Device Market Growth Remains Steady at 55%*. Retrieved December 3, 2006, from <http://www.canalys.com/pr/2006/r2006071.pdf>
- Canalys. (2008). *Smart Mobile Device Shipments Hit 118 Million in 2007, up 53% on 2006*. Retrieved March 6, 2008, from <http://www.canalys.com/pr/2008/r2008021.pdf>
- Geihs, K. (2001). Middleware challenges ahead. *IEEE computer*, 34(6), 24-31.
- Google. (n.d.). *Google Maps for Mobile*. Retrieved March 12, 2007, from <http://www.google.com/gmm/>
- Hu, W.-C., Lee, C.-w., & Yeh, J.-h. (2004). Mobile commerce systems. In Shi Nansi, editor, *Mobile Commerce Applications*, pages 1-23, Idea Group Publishing.
- IDC, Corp. (2008). *PC Market Rebounds with Strong Demand for Portables, Fueling Hopes for Holiday Sales, According to IDC*. Retrieved January 23, 2008, from <http://www.idc.com/getdoc.jsp?containerId=prUS20995107>
- Microsoft, Corp. (n.d) *Internet Information Services*. Retrieved June 15, 2007, from <http://www.microsoft.com/WindowsServer2003/iis/default.aspx>
- NTT DoCoMo, Inc. (2007). *i-mode*. Retrieved June 12, 2007, from <http://www.nttdocomo.com/services/imode/index.html>
- Open Mobile Alliance Ltd. (n.d.). *WAP Forum*. Retrieved from June 13, 2007, from <http://www.openmobilealliance.org/tech/affiliates/wap/wapindex.html>
- Oracle. (n.d.). *Oracle Databases*. Retrieved May 25, 2007, from <http://www.oracle.com/database/index.html>
- Ortiz, S. (2000). Embedded databases come out of hiding. *IEEE Computer*, 33(3), 16-19.
- Sadeh, N. (2002). *M-commerce: Technologies, services, and business models*, pp. 177-179, New York: John Wiley & Sons.
- Strategy Analytics. (2008). *Nokia Reaches 40% Share as 332 Million Cellphones Ship Worldwide in Q4 2007*. Retrieved February 27, 2008, from <http://www.strategyanalytics.net/default.aspx?mod=ReportAbstractViewer&a0=3755>
- Sun Microsystems, Inc. (n.d.). *Sun Java System Web Server*. Retrieved June 19, 2007, from [http://www.sun.com/software/products/web\\_srvr/home\\_web\\_srvr.xml](http://www.sun.com/software/products/web_srvr/home_web_srvr.xml)
- Symbian Limited. (2006). *Fast Facts*. Retrieved December 10, 2006, from <http://www.symbian.com/about/fastfacts/fastfacts.html>
- Turban, E., King, D., Lee, J., & Viehland, D. (2004). *Electronic Commerce 2004: A Managerial Perspective*. Prentice Hall.
- Varshney, U., Vetter, R. J., & Kalakota, R. (2000). Mobile commerce: A new frontier. *IEEE Computer*, 33(10), 32-38.
- Yankee Group. (2001). *Over 50% of Large U.S. Enterprises Plan to Implement a Wireless/Mobile Solution by 2003*. Retrieved December 10, 2002, from [http://www.yankeegroup.com/public/news\\_releases/news\\_release\\_detail.jsp?ID=PressReleases/news\\_09102002\\_wmec.htm](http://www.yankeegroup.com/public/news_releases/news_release_detail.jsp?ID=PressReleases/news_09102002_wmec.htm)



# A Live View Of The World

Douglas J. Hickok, B.S. Computer Science University of Wisconsin – Platteville Platteville, Wisconsin 53818 hickokd@gmail.com	Dr. Mike Rowe Software Engineering University of Wisconsin – Platteville Platteville, Wisconsin 53818 rowemi@uwplatt.edu
-------------------------------------------------------------------------------------------------------------------------------------------	--------------------------------------------------------------------------------------------------------------------------------------

## **Abstract**

A picture may be worth a thousand words, but none of those words are “now”. Internet enabled webcams offer a sense of “now”, and combining them into a single world view was a tempting idea. With the college Engineering Expo rapidly approaching, the RAD (Rapid Application Development) software process was utilized to create a fully functional, high impact demonstration in a tight timeframe. With some additional work, the project could end up a useful tool for geography teachers and anyone with an interest in seeing the world.

# 1 Introduction

The “Live View of the World” was originally demonstrated at the 2007 University of Wisconsin – Platteville Engineering Expo. It's a web-based program which combines webcams from all around the world into a single map in order to convey what the world looks like at that minute. This paper details the design choices that were taken, the challenges that were faced, the public reaction during the Expo, and how the system can be improved. This project proves that with an ideal software process, a good idea can come to life and capture the interest of others.

# 2 Design and Implementation

The idea to merge webcams with a world map came only two weeks before Engineering Expo, on the last possible day to register a project. With my software engineering background, I knew better than to sit down and start coding whatever came to mind. Good decisions needed to be made in order to complete this project on time.

The obvious application development process that was needed was RAD. [1] RAD, short for Rapid Application Development, is geared towards very fast development with reduced features, usability, and execution speed. It focuses on keeping it simple, but also keeping good quality despite the lack of features.

The high level requirements for the system were as follows:

- The system shall be web based.
- Live webcam images shall be plotted onto a map.
- The live webcams shall not be abused.
- The map should be zoomable.
- The final result shall look uncluttered and presentable.

Making the system web based instantly made it available to anybody who wanted to check it out. It also kept the system cross-platform. Most importantly though, it allowed it to be friendly to the webcam providers (discussed later).

Most people's first reaction would be to implement this in the Google Maps API. However, the result isn't as elegant as you might think. LiveLook [2] and Goccam [3] currently does this, and the result is a cluttered map of pegs. In order to see any of the cameras, the peg must be clicked to open the balloon containing the image. For “A Live View of the World”, the pegs had to go. At the time, the Google Maps API only allowed pegs, making the API a bad choice for a presentable implementation.

Instead, I chose to implement the project in PHP. PHP offered rapid development potential due to it's image manipulation functions and it's ability to easily pull a remote image. Rapid development was aided by the use of XAMPPlite, a free framework that

includes PHP and an Apache web server that runs locally.

The design for the core code was to simply plot a list of clickable images on a map. The list of images was originally intended to be kept in a database for easy maintenance, but the feature resulted in unneeded complexity that made no difference to the intended audience. It was decided that the easiest route was to place them in a comma separated text file along with their GPS coordinates.

Zooming capabilities were needed in order to get a better view of a specific continent or country. The view of the entire world wouldn't be able to show much detail in each region. Also, zooming in on the world map would result in pixilation of the map image itself. True zooming was an overly complex option, so it was decided that fixed maps should be used instead.

The resulting core code was able to generate a webpage with links to various maps, with the currently selected map appearing below it. The map showed a thumbnail sized live image of each webcam centered at the webcam's approximate GPS location. Each webcam was clickable, and clicking it took the user to the full sized original live image. The project was simple and looked superb.

## **3 Maps**

### **3.1 Process**

Due to time constraints, it was decided that the project should have maps for the world and for each large land mass. Maps were created for North America, South America, Europe/Asia, Africa, and Australia. The best webcams from each of those were included in the world map.

The base maps were generated using Google Maps. A small number of screenshots were combined into a single larger map image. To remain compliant with the Terms of Service, the copyright was added to the web page. Nonetheless, this specific use of Google Maps remains questionable, and is further discussed in the Future Work section.

### **3.2 Challenges**

Mapping is surprisingly complicated when trying to do it manually. The Earth is round, and the image must be flat. The specific map projection makes plotting GPS points into a math problem, and plotting those points to a pixel value on the final image requires some scaling and map calibration.

Google Maps is known to use a Mercator map projection. [4] Luckily, functions exist

that can crunch the math and convert the GPS value to a global pixel value. [5] Since none of the maps for the project correspond to the resulting pixel value, the values need to be aligned and scaled. Each map has a manually aligned border associated with it. The pixel values and corresponding GPS values were found for opposite corners of the map image, which is needed for the aligning and scaling functions. Finally, we are able to center the webcam thumbnail image on this calculated point, assuming it is within the bounds of the map. PHP functions take care of making the thumbnail and plotting it on the map image.

## **4 Webcams**

### **4.1 History**

The first webcam went live in 1991, and was pointed at the University of Cambridge Trojan Room coffee pot. [6] The coffee was shared by dozens of people, some of which were inconveniently far away. The live image saved them countless trips to an empty pot.

Since then, cameras for computers have gotten smaller and cheaper, to the point of being included for free when buying a new computer. With free software, a computer's camera can be made available to the Internet. To skip the whole computer part, webcams are available that plug directly into the network such as the Axis webcam. [7] Some even have wireless and allow the viewers to zoom, tilt, and pan the camera remotely.

Because of the cheap hardware and easy setup, people have been putting up webcams wherever they can. One of the largest webcam aggregators is EarthCam. [8] Going there, it's clear that webcams exist just about everywhere and point at just about anything. These webcams are mostly put up by people at home out of interest. The government puts up webcams to monitor traffic or display live views of landmarks. The weather service encourages webcams for live weather views. Companies like to show off their services, such as resorts or car lots. Even most large cruise ships have a webcam on board. Anyone with an interest in showing off something or an interest in viewing something remotely is likely to have a webcam.

### **4.2 Process**

In order to keep a clean interface, manually picking webcams was the best option. Because of this, the number of webcams could start sparse, and become more dense until time was up for Expo. The general process was to pick an area on the map, browse for the best webcams for that area, and find the GPS coordinates based on the IP address or the description of where it was at. If several webcams were located too close, the best

was chosen to keep things uncluttered.

All chosen webcams had to be outdoors for a better “world view”. Indoor cameras such as office cams and fish cams didn't produce the desired effect. For viewing the world, the outdoors seemed more appropriate. Cityscapes and landmarks were ideal, because people could connect with them better.

### **4.3 Challenges**

The major challenge in hunting for webcams is that the Earth is round. Therefore, every image for half of the world is black, making it difficult to judge the quality of the webcam or what it's pointing at. With sleep and classes in the way, webcam hunting took place in the afternoon and evening when all of Europe and Africa were in the dark. I quickly learned to do work on the America's early in the afternoon, and then switch to Australia/Oceania and Asia later in the evening. Working on Europe and Africa required an inversion of my sleep schedule.

Webcams run by the government or by corporations sometimes block direct requests. The image is displayed correctly on their page, but a direct request results in an alternate image saying we shouldn't steal the image. This was often discovered after I had done the work to add it.

Since most webcams are run by typical people at home, they are extremely unreliable. They only work when the technology is running and when the Internet connection is working. Many webcams have a short lifespan, and some had even quit working within a few days prior to Expo.

Some of the newer webcams use streaming video technology instead of an image. This project was limited to using pictures, so the streaming video webcams were unable to be used unless they also had the functionality to provide a picture. The Axis webcams offer both types in parallel.

Image quality is normally questionable with outdoor webcams. The image is often blurry, distorted, blocked, pixilated, or too small. There are relatively few high quality cameras, and those were almost always included in this project. Sometimes though, a bad image is all that's available for an area.

Speed was more of an issue than previously though. A lot of webcams are on very slow Internet connections at remote places of the world, and are getting hit by people all over the world. Most webcam aggregators direct clients directly to the image, which results in every user getting their own copy from the webcam. This is especially hard on the webcam providers. Timeouts or the inability to connect are common, and the images sometimes download much slower than dial-up speeds. PHP doesn't have the ability to

multithread, so image downloads have to run one at a time in order to add them to the base map. If the download fails, the image is skipped. The entire process of downloading all the images is known to take up to five minutes.

To increase the speed and make the program more friendly to the webcam providers, caching has been added. A new image is only retrieved if the current cached image is older than 15 minutes, plus or minus a few minutes. This way, a single image download from the webcam provider can be reused for every client viewing the project at that time. Also, each client view can take advantage of the cached versions and may only need to pull a few new images, which speeds up the time to wait for the page to load.

## 5 Highlights

North America is fairly packed with webcams. The map goes down to Mexico, where there are several beach cams along the coast and over to the tropical islands including Bermuda, the Virgin Islands area, and Hawaii. The central US features live views of the capital building in Washington DC, the St. Louis arch, the Golden Gate bridge, the view from behind the Hollywood letters, the Elvis mansion, the Old Faithful Geyser, Pikes Peak in the Colorado Rockies, and many more. The webcam at the University of Wisconsin – Platteville was an important one to add, since the view was of the new building construction on campus and was right by the Expo. Canada contains a few good quality cams including one in Vancouver and one in Banff National Park. Northern webcams include a few around Whitehorse, Yukon by Alaska and several weather cams around Alaska.

South America has a sparse number of webcams, and almost all are of low quality. There are a couple in Venezuela, Peru, southern Brazil, Argentina, Chili, and Uruguay. A nice island cam exists off the coast of Brazil at Fernando De Noronha. Also, another is approaching Antarctica at the South Georgia Islands. At the North, there are a few nice webcams at the Panama Canal which show the ships traveling through.

To save time, Europe, Asia, and the Middle East were combined into a single map. Europe is packed with webcams, similar to the US. The rest of the area is somewhat sparse. There are only a few webcams in the Middle East, Russia, India, and Taiwan. Korea, and the Middle East likely don't have webcams because of Internet censorship. China likely has some, but are blocked by the country's firewall. [9] Japan has several webcams, but only a few can be displayed in such a small area. Europe has the same problem, since a single webcam image can take up a whole country. The project shows the Eiffel Tower, Loch Ness, Venice, and several large cities and famous beaches.

Africa is perhaps the sparsest map in the project. On the north, there is a webcam in Gibraltar, Morocco, Gambia, and two in Egypt. The main one in Egypt is aimed at the pyramids, and the other is at a beach resort. Towards the south, there's one in Kenya, two

in Tanzania, one in Seychelles, one in Mauritius, one in Namibia, and a few in South Africa. One of them in South Africa is at a popular elephant watering hole. Lastly, there's a lone cam in the middle of Africa in Cameroon. It was set up as part of a project to de-gas lake Nyos in response to the threat of dangerous explosions. [10] The image is updated via satellite twice a day so that the process can be observed remotely.

Australia and Oceania are full of beach cams. Beach cams are popular around most of Australia, since that is where the population mostly is. Sydney area and some of the other big cities have several other cams, including one at a winery, one pointing at the Sydney Opera House, and several cityscapes. New Zealand has numerous webcams. French Polynesia does also, but all seem to be owned by resorts. Webcams are sparse in New Caledonia, Indonesia, Malaysia, Philippines, and Thailand.

The world view takes the best webcams from the other maps, including any famous landmarks or any webcams which seem to best represent the area. A few additional ones are added to the world view in places that are not covered by other maps. A hotel cam in Iceland, and a cityscape in Honningsvåg, far north Norway for example. The webcams near the north pole are difficult to maintain, since they are easily damaged by shifting ice and storms. [11] Believe it or not, but there are also six webcams around Antarctica at various research stations, including one at the south pole.

## 6 Uses

It is often said that people don't know what they want until they see it. This seemed to hold true during the Engineering Expo. Several people thought that it would make a valuable tool for a variety of uses.

Teachers can make use of this project in teaching geography. Landmarks and areas of the world can be taught visually and in relation to where they are on the world map. Students can see the different types of climates in different areas. They can see where the snow is, and watch as the hurricanes approach the coastal areas. They can observe the architecture, and sometimes the people, of different cultures. Using this tool, it's easy to prove to the youngsters that the Earth really is round, because half of the cameras are black as night. If anything, the youngsters who saw this project at Expo seemed to discover a new interest in geography, and a curiosity about other places in the world.

Weather services have already been making use of webcams, but don't seem to have the same sort of interface as this project. Due to the size of Alaska, the FAA has set up a network of weather cams to better predict the weather. [12] It uses a map interface, but forces the user to zoom in to a region and select the camera location before actually seeing the image. By plotting the thumbnails directly on the map, it may be easier to pick out larger patterns, and to zoom in to cameras of interest without having to know their name.

HAM radio operators can even benefit from this project. They are able to speak to people all over the world, but sometimes have no idea what the area is like that they are speaking to, or at least what it's like right now. Taking a peek at the others' area should give them something to talk about.

## **7 Future Work**

Although the RAD software process worked well for getting this project done on time, there are several areas that need improvement.

The first is to either get better maps or switch to the Google API. The Google API currently allows custom icons, but it is unknown if there is a limit of different icons or if it is able to display thumbnails of webcams. The static maps worked for the Expo demo, but it would be very handy to pan and zoom to different locations. The Google API would work very nicely for this, as long as it is able to display thumbnails.

The data would need to be moved to a database for easier querying. Also, the work of requesting the images should be moved to the browser since it can do multiple simultaneous requests. It should still go through a proxy on the web server though, so that it can get all the benefits of a cache.

The last technical improvement would be to keep the philosophy of choosing the best image for a specific location. Webcams should have a rating that is voted on by the users, and the best rated webcam should appear in a location as the user zooms out. No matter what zoom level the user is at, the shown webcams should be the best that are available and should remain uncluttered.

Webcams should be user-contributed, since adding and maintaining webcams can be overwhelming. Also, the system should support moving webcams. Most of the big cruise ships have webcams on board, and a webpage listing their current GPS location. Webcams have also been added to some taxis and semi trucks. It would be interesting to be able to plot these in their current locations on the map.

## **8 Conclusion**

The RAD process worked like a charm to get the Live View of the World looking good for the Engineering Expo. The project brought positive reactions, and awakened curiosity about the rest of the world. With some improvement and user-driven content, the Live View of the World could prove to be a valuable tool.



## References

- [1] Automated Architecture, Inc. "Rapid Application Development," *blueink.biz*. [Online]. Available: <http://www.blueink.biz/RapidApplicationDevelopment.aspx>. [Accessed Mar. 6, 2008].
- [2] LiveLook LLC, "Outdoor Webcams and Live Video". [Online]. Available: <http://www.livlook.com>. [Accessed Mar. 6, 2008].
- [3] Butterfat, LLC, "Watch the unsuspecting," *Goocam*. [Online]. Available: <http://www.butterfat.net/goocam/>. [Accessed Mar. 6, 2008].
- [4] Charlie Savage, "Google Maps Deconstructed". [Online]. Available: <http://cfis.savagexi.com/articles/2006/05/03/google-maps-deconstructed>. [Accessed Mar. 6, 2008].
- [5] Various Authors, "Mercator," *OpenStreetMap*. [Online]. Available: <http://wiki.openstreetmap.org/index.php/Mercator>. [Accessed Mar. 6, 2008].
- [6] Quentin Stafford-Fraser, "The Trojan Room Coffee Pot". [Online]. Available: <http://www.cl.cam.ac.uk/coffee/qsf/coffee.html>. [Accessed Mar. 6, 2008].
- [7] Axis Communications, "Leader in network cameras and other IP networking solutions". [Online]. Available: <http://www.axis.com/>. [Accessed Mar. 6, 2008].
- [8] EarthCam, Inc., "Webcam network". [Online]. Available: <http://www.earthcam.com/>. [Accessed Mar. 6, 2008].
- [9] Ben Elgin and Bruce Einhorn, "The Great Firewall of China," *BusinessWeek*. [Online]. Available: [http://www.businessweek.com/technology/content/jan2006/tc20060112\\_434051.htm](http://www.businessweek.com/technology/content/jan2006/tc20060112_434051.htm). [Accessed Mar. 6, 2008].
- [10] Michel Halbwachs, "Webcam on lake Nyos". [Online]. Available: <http://pagesperso-orange.fr/mhalb/nyos/webcam.htm>. [Accessed Mar. 6, 2008].
- [11] National Oceanic and Atmospheric Administration, "Live from the North Pole". [Online]. Available: [http://www.arctic.noaa.gov/gallery\\_np.html](http://www.arctic.noaa.gov/gallery_np.html). [Accessed Mar. 6, 2008].
- [12] Federal Aviation Administration, "Weather Cams Home". [Online]. Available: <http://akweathercams.faa.gov/>. [Accessed Mar. 6, 2008].

# Computer Supported Collaborative Learning in the Geology Explorer

Otto Borchert, Brian M. Slator, Guy Hokanson, Lisa M. Daniels, John Reber, Dan Reetz, Bernhardt Saini-Eidukat, Donald P. Schwert, Jeff Terpstra  
World Wide Web Instructional Committee  
North Dakota State University  
Fargo, ND 58102-5057  
{Otto.Borchert, Brian.Slator, Guy.Hokanson, Lisa.Daniels, John.Reber,  
Dan.Reetz, Bernhardt.Saini-eidukat, Donald.Schwert,  
Jeff.Terpstra}@ndsu.edu

## Abstract

The Geology Explorer is a multi-user geologic simulation created by North Dakota State University's World Wide Web Instructional Committee (WWWIC). Students use the Geology Explorer software to explore a mythical planet to learn geologic concepts like rock and mineral identification and landform creation.

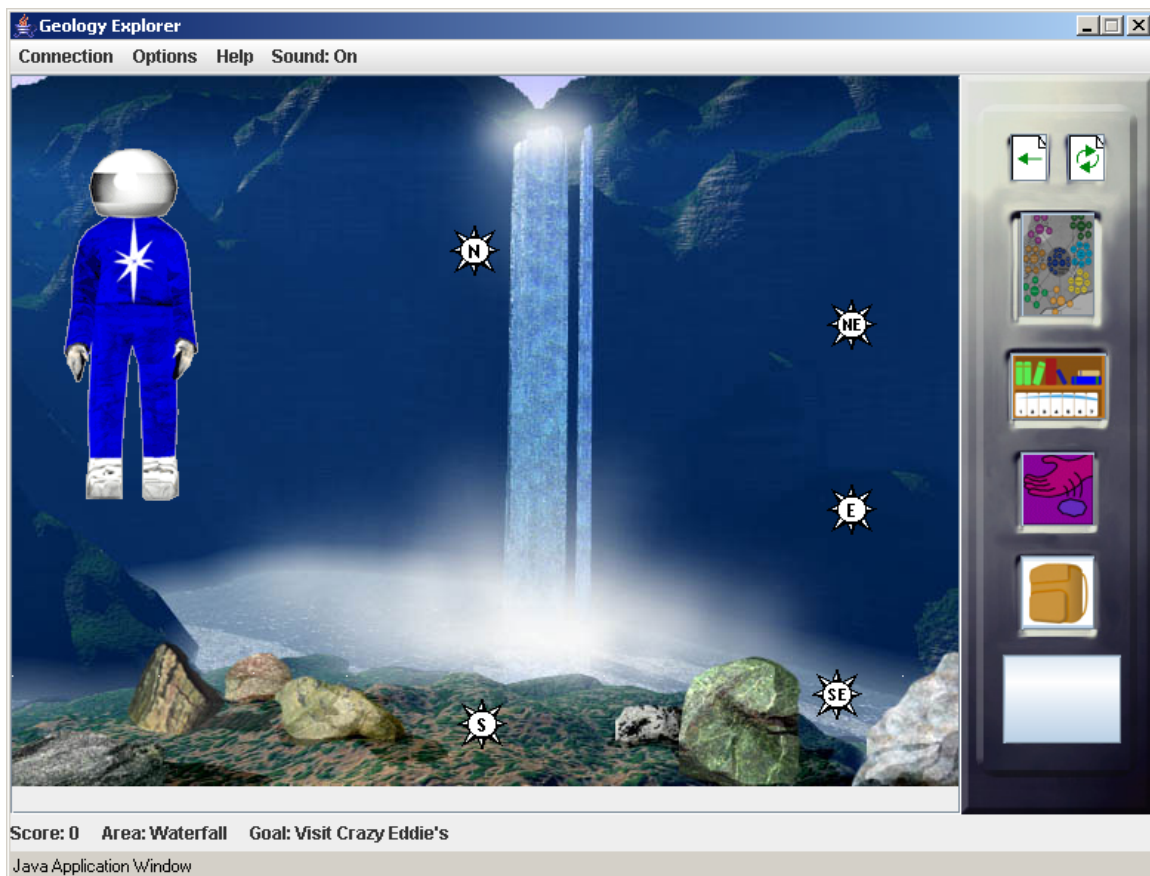
Until recently, cooperation between online explorers was limited to informal groups of students in close physical proximity. Advances in the software have provided a framework for group participation from remote virtual explorers. A recent pilot study provides some initial thoughts and reactions for further research. Further experimentation has shown positive results for use of the software in classroom settings.

This paper will describe the recent developments to the Geology Explorer software, both pedagogically and organizationally, the difference between group and individual goals and how educational theories contributed to their creation, results of the previous experiments and research directions indicated by those results.

## Introduction

The Geology Explorer (<http://oit.ndsu.edu>) is a role-based virtual environment designed to teach students about geology. Immersed in the environment, students learn to think and act as professional geologists, studying rocks, minerals, interesting geologic areas, and landforms. Students begin playing on a mythical planet called Oit, which according to the fictional 'back story' has an orbit directly opposite the sun from Earth.

Players are asked to follow a series of goals that slowly become more and more difficult via a process of scaffolding. Students begin the game by answering a series of questions assessing pre-treatment attitudes towards science and computers, and determining initial geological understanding. Each individual student is then free to explore the planet, purchase instruments, and engage in the first task, rock and mineral identification.

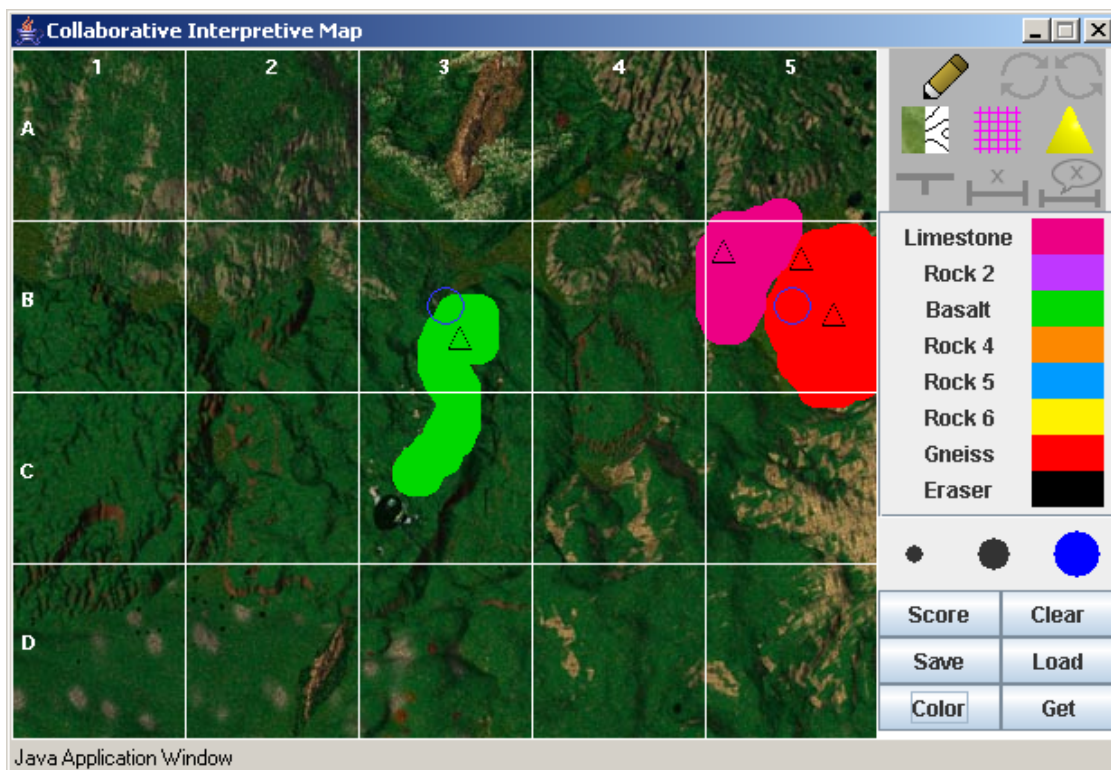


*Figure 1: A Waterfall on Planet Oit, with Rocks below. Navigation is achieved by clicking on the compass points in the scene. To the right are buttons for navigating backwards and reloading the scene, accessing a map of the planet, accessing a 'bookshelf' of reference materials, selecting an instrument or tool, storing samples, and magnifying an object. In this scene a player (upper left) is just floating into view from elsewhere on Planet Oit.*

The first geologic exploration module of the game continues the scaffolding philosophy.

Students are assigned a specific mineral to locate and identify. They are given hints as to the location of the mineral, but these prompts are progressively taken away as the student proceeds through the module. Typically, they are asked to gain 500 points in this module. They receive 100 points for finding a “primary” goal or the specific rock or mineral suggested by the game. They also receive 25 points for identifying any other outcrop on the planet.

To perform these rock and mineral identifications, students travel to virtual landscapes that represent real-life geologic features, some examples include an arroyo, a desert, a cutbank, and a glacier. A waterfall is pictured in Figure 1. Rocks and minerals are highlighted by a white border while the mouse cursor is on them. This interface allows students to perform experiments using tools and instruments "purchased" previously and to eventually identify the outcrops using the scientific method.



*Figure 2. A Cooperative Map. Players 'paint' the regions of the map using colors to represent rocks and minerals. Note a region of Basalt has been painted into the center of the region. Players in distant locations can share the map, and add their own coloring using the "Get" button in the lower right-hand corner of the controls.*

After completing this module, students are directed to a larger area where they are asked to create a geologic map. A geologic map shows the location and distribution of rocks and minerals within a particular geographical area (Skinner, 2004). A sample map is shown in Figure 2. By completing this map, students are able to perform even more geologically interesting tasks later, which currently includes a “true thickness” module, where students are directed to find the shortest path through a particular rock unit in order to run a power cable through a particularly unstable portion of rock.

The game is completed after students finish a battery of post-assessment activities to determine content knowledge acquisition and attitude changes that may have occurred during the experience.

## **Background**

The World Wide Web Instructional Committee (WWWIC) at North Dakota State University is dedicated to the development of multi-user immersive virtual environments for educational purposes. These environments drive students to learn through role-based, inquiry driven simulation. Topics of interest include geology, biology, computer science, anthropology, and economics through the Geology Explorer (Slator, et al., 2006), Virtual Cell (McClellan, et al., 2001), Programming Land MOOseum (Slator, et al., 2004), On-A-Slant Virtual Village (Hokanson, et al., In Press), and Dollar Bay Economic Simulation (Regan and Slator, 2002), respectively.

This paper focuses on the development and dissemination of formal cooperative activities in the Geology Explorer. A number of educational theories were utilized to develop these activities, and experiments performed to determine the effectiveness of the simulations. These experiments have shown that motivation among group members increased significantly as compared to individuals working alone.

## **Advantages of Collaborative Learning**

A number of advantages for collaborative learning are enumerated in the literature and include the following positive attributes for groups: they increase academic skills, make it easier to root out misconceptions, teach collaborative work skills, show that the sum of individual knowledge is greater than the parts, allow their members to develop social skills, strengthen inter-group relations, as well as many other positive benefits.

Of utmost importance to this research is the ability of collaborative groups to increase academic skills. This is cited throughout the literature (Kagan, 1994, Slavin, 1995, Sharan and Sharan, 1978). By harnessing social constructivism, students are able to work together doing hands-on tasks to gain a better understanding of the subject material. Our research has shown significant increases in learning after using the Geology Explorer on an individual basis (McClellan, et. al., 2001). By incorporating group work into previously individualized tasks, it is hoped that this improvement can be maximized.

Collaborative groups also excel at rooting out misconceptions (Brown, Collins, Duguid, 1989). For example, in a collaborative group, each student may get an opportunity to explain how a particular concept works. If one of the students has a response markedly different from others, the effective group can come to a consensus about which opinion is correct. Note, however, that the group must be effective. If groupthink or the polling problem are present, misconceptions can take hold (Koschmann et al., 1996; Janis,

1972)).

The National Association of Colleges and Employers sponsors a survey every year called Job Outlook that polls employers for the traits, skills, and qualities they look for in future employees. Interpersonal skills, communication skills and teamwork skills rate first, third, and sixth respectively in the 2004 survey (NACE, 2004). By giving students the opportunity to work in groups and providing training on how to work effectively in those groups, group members are able to build skills that will be useful throughout their personal and occupational lives. As Brown, Collins, and Duguid (1989) state: “If people are going to learn and work in conjunction with others, they must be given the situated opportunity to develop those skills”.

Feltovich, et al. (1996) focus on how the sum of individual knowledge is greater than the composite parts. When individual students become experts at a particular field, new points of view that are present when working in a group can create a synergy of intellectual endeavors. Each individual provides a different piece of the puzzle by having his own unique point of view, but by working together they can create a masterpiece of greater combined understanding.

Goldman (1996) provides an alternative analysis of how often learning moments occurred within a high school physics class by using a software package designed to help students learn about light and its properties. Despite teacher interaction, students tended to talk quite a bit about non-class related activities. They still completed the work, but in the process of learning about light and reflection, they learned important social skills which are equally necessary later on in life.

One of the advantages of cooperative work is the strengthening of inter-group relations. People in groups tend to relate better to each other. As noted in Slavin (1995), this is quite evident in racially diverse school populations. Well created situations can strengthen relations between students with different racial backgrounds. This is also true for students who are academically challenged. In a competitive environment, students who aren't performing to normal standards are easy targets for emotional abuse and verbal taunting. In a cooperative environment, students depend on each other to do well, and as such, will provide support to those group members who are struggling to understand the material. Cooperative work also increases self esteem, time on task, and altruism. It decreases disruptive activity, creates a more internal locus of control for students and creates an environment where academic endeavors are considered positive, where students enjoy class, school, and other classmates (Slavin, 1995).

Brown, Collins, and Duguid in their seminal paper on Situated Cognition also listed a number of features that pervade collaborative learning. They claim that collaborative learning groups provide for collective problem solving, allow students to perform multiple roles, create opportunities to confront misconceptions and ineffective strategies, and provide collaborative work skills (Brown, Collins, Duguid, 1989).

## **Disadvantages of Collaborative Learning**

Koschmann, et al. (1996) describes one of the primary problems of cooperative work, called the “polling problem”. The polling problem occurs when dominant members of a group suppress the opinions of submissive members, resulting in a decrease in the diversity of ideas. The polling problem also occurs when answers are gathered in a public fashion with more submissive members. If a vast majority of group members agree to begin with, a more submissive member may simply agree with the group, rather than express doubts about the issue. If one group member is more insistent and others are not willing to challenge inconsistencies or incorrect assumptions, the polling problem becomes quite acute.

Janis (1972) defines “groupthink” as a decrease in critical thinking and contrary viewpoints in a highly cohesive group. A group that is cohesive interacts well, has been together for a long time, and the individuals in the group have similar views and values. Highly cohesive groups tend to want to remain together with as little conflict as possible. Groupthink occurs when individuals reinforce the cohesiveness of the group by singling out people with dissenting opinions or views that run against group norms. In this manner, students run the risk of not hearing a correct solution, because it runs against the common beliefs of the group. More recent research by Baron (2005) indicates that there are three main conditions for groupthink to occur. First, one must identify with the group; they must feel they belong to a group having a common purpose. Second, the group must have a common viewpoint or norms guiding its direction. Third, the group must feel that they are unable to complete the task given to them, either because of its complexity or because of the inexperience of the group members.

Slackers in groups result in decreased effectiveness because of a lack of cohesiveness. Slackers can occur in any group for a variety of reasons including: lack of motivation, esteem, and/or understanding. The antithesis of the slacker also causes difficulties. This individual does not feel comfortable allowing other people to have control of his personal goals, and so attempts to do vast majorities of the collaborative effort, resulting in a net group loss (Middlecamp, 1997).

## **Group Composition**

Kagan (1994) describes 4 different methods of assigning students into groups: in random groups, by interest groups, in heterogeneous groups, and in homogeneous language teams. Each of these methods have their advantages and disadvantages.

Placing students into random groups is the easiest to implement, but one of the least effective forms of group composition. An instructor merely has to count students off into groups of the desired size. The primary drawback to this method is the potential lack of diversity within groups. A group with all low performing students would have no one to turn to if they had problems, other than the instructor. A group of only high achieving students would quickly outpace other groups, potentially causing boredom and classroom

disruption.

Students can also be asked to form groups themselves. This will be the option preferred by most students in the class, as they are able to be with friends. However, the disadvantages in the random groups become more pronounced in this situation, as friends are more likely to have similar interests and educational levels resulting in low levels of diversity.

The most effective group composition method is to group students heterogeneously. This is mainly accomplished by giving students a content quiz covering the material to be presented before the lesson. These scores can then be coupled with gender, age, ethnicity, and other demographic data to create groups that are diverse. These groups, when working effectively, have the highest potential for learning, as high achieving students teach the lower, and people with different background are able to share personal perspectives of the topic at hand.

Homogeneity in groups is primarily effective when dealing with classes that are being taught in a student's non-native language. Groups can then be created in which students with similar native language ability are together. The group is still heterogeneous in reference to other traits, but if a language translation issue comes up, they are able to communicate in their native language to learn about the concept being presented.

## **Educational Theories on Collaborative Learning**

There are a number of activities in educational research that have been studied and shown to be quite effective in teaching students in group settings. This paper will focus on the Jigsaw method, Student Teams - Achievement Divisions (STAD), and Group Investigations (GI).

The Jigsaw method, originally developed by Aronson (1978) is used to study a vast body of knowledge by groups of students. The instructor begins by introducing the subject matter to be studied to the class as a whole, engaging student interest and brainstorming how to divide the knowledge into pieces. Each group is assigned a piece of this knowledge to study in-depth. Within the group, each individual student is tasked with gathering information about a portion of the group's task. After gathering this information, the group meets to combine the information into a cognizant, demonstrable whole. Finally, each group presents their knowledge to the class as a whole, so that students learn from each other and with each other.

Student Teams – Achievement Divisions is a method of teaching students that harnesses both collaborative and competitive learning strategies. Students are placed heterogeneously into groups of four individuals each. Each group is then given a lesson to learn, the group members work together to learn the material. After they feel prepared, the students take a quiz, where they are not allowed to help each other. Based on the quiz scores, the group is given a number of points. As students spend more time in their group,



they gain more points. After achieving certain milestones, the group is recognized and rewarded for the work they have been doing (Sharan, 1994).

Group Investigation was created by Sharan and Sharan (1994). In this collaborative technique, students follow the methodology of scientists developing research presentations. First, the class as a whole discusses what topic they want to investigate. They then break up into teams to research portions of the topic that are of interest to them. They plan an investigation, perform research, and give a presentation of their findings to their colleagues in the class. Although similar to the Jigsaw method, students in a GI project get to choose the topics they will study themselves. They also require knowledge and experience with group work beforehand, so that they can work effectively as a research team.

## **Computer Supported Collaborative Learning**

The field of Computer Supported Collaborative Learning provides a rich set of implemented examples of software used to instruct students in groups. Three of the most highly referenced projects are the Computer Supported Intentional Learning Environment (CSILE), the Fifth Dimension Project (5thD), and the Electronic Network for Interaction.

CSILE was created by Scardmalia and Bereiter (1994) to allow students to grow a body of knowledge in a hyperlinked environment. Students in a classroom research a particular body of knowledge and create a class project showing notes, discussions, gathered facts, and experiments performed. This body of knowledge can then be shared with other classes or used by students around the world.

The Fifth Dimension project was originally developed to allow children to learn in a collaborative, community based after school environment. The project has become a template for a number of different locations, each implementing their own version of the Fifth Dimension.

When starting a Fifth Dimension project, a college or university works with a local school to build a community of integral members. At the base of the structure are the children doing the learning via virtual and real-world based activities. The children are taught by graduate and undergraduate students in education, psychology, and other interested disciplines. These students learn about research in their field through experience in the classrooms. At the top of the Fifth Dimension are the college professors that provide initial instruction and guidance for the project and are a source of volunteers for the project. The entire structure is built with the support of the surrounding community which provides space and sustainability (Brown and Cole, 2000).

ENFI was produced at Gallaudet University as a method of teaching hearing-impaired students to write English in a computer-based collaborative environment. The first ENFI network consisted of what would be recognized today as an instant messaging program. The teacher and students each had their own computer running the main ENFI client

program. This program consisted of a window which displayed what was written by class members and a region where individuals were able to type in their own message. This initial ENFI network spread across the country to many different colleges and universities (Bruce, Peyton, Batson, 1993).

## **Group Goals**

When creating group goals, effort must be exerted to minimize the disadvantages of collaborative groups. Kagan (1994) discusses a set of principles for collaborative group learning he calls PIES, which stands for Positive Interdependence, Individual Accountability, Equal Participation and Simultaneous Interaction. These four principles, when followed while creating group goals decreases freeloading and one person doing all the work (described above).

Positive Interdependence refers to the fact that all group members need all other members to succeed to complete the goal. All group members are needed and they each contribute to the overall goal. Each member depends on the others to complete the task at hand.

Individual Accountability means that each group member must complete their part of the task for the group to succeed. On the surface, this seems obvious, however, not only does each person need to complete their portion, but all group members must be able to see the portions completed. Each person needs to feel that they have contributed to the group effort, and they see that by looking at each person's contribution.

Not only do group members have to see that others have completed a portion of the tasks, they need to see that equal portions have been completed by each member. Equal Participation refers to this concept. By seeing that all group members have had an equal share of the work, students feel that the exercise has been fair.

Finally, Simultaneous Interaction refers to the fact that all group members must be able to work on the problem together. Individuals need an opportunity to learn from one another as they progress through the task. By working on a task simultaneously, students are able to work together, teaching and learning the material as presented.

## **Geology Explorer Group Interface**

The Geology Explorer is built on a client-server architecture, with a Java-based client and LambdaMOO (Curtis 1997) based server. Both the client and the server required extensive changes to implement the group functionality. This paper will focus on the client-side, user experience portion of the group interface.

Students can be placed into groups into methods similar to those listed in “Group Composition” above and include: random teams, interest groups, and heterogeneous groups.

Students are placed into random teams via a web-based instructor-only interface. Instructors are able to control all aspects of group formation, permissions, and logging via this interface. Teachers can form a random number of groups or assemble them as they see fit. They are also able to view what the group has done, what they have said, and how many points they have scored. This interface is shown in Figure 3.

**Automatic Group Formation Test -- Group Editor**

**Class-wide Actions and Options**

- [Create New Group](#)
- [Change Class-wide Options](#)
- [Randomly Create Groups](#)

A New Group (#42597)			
Member Name	Change the Leader	Change Member Permissions	Remove this Member
OttoBorchert35	<a href="#">Set As Leader</a>	<a href="#">Change Permissions</a>	<a href="#">Remove</a>
OttoBorchert34	Leader	<a href="#">Change Permissions</a>	<a href="#">Remove</a>
Information and Actions			
Current Goal: Visit Rolling Hills Goal (#1728)			
<a href="#">Add Member</a>	<a href="#">Disband Group</a>	<a href="#">Set Name</a>	<a href="#">Set Goal</a>
		<a href="#">Delete Group</a>	<a href="#">View History</a>
			<a href="#">View Chat Log</a>
			<a href="#">View Notes</a>

Remove Member Test (#42602)			
Member Name	Change the Leader	Change Member Permissions	Remove this Member
OttoBorchert34	Leader	<a href="#">Change Permissions</a>	<a href="#">Remove</a>
OttoBorchert35	<a href="#">Set As Leader</a>	<a href="#">Change Permissions</a>	<a href="#">Remove</a>
Information and Actions			
Current Goal: Nothing (#-1)			
<a href="#">Add Member</a>	<a href="#">Disband Group</a>	<a href="#">Set Name</a>	<a href="#">Set Goal</a>
		<a href="#">Delete Group</a>	<a href="#">View History</a>
			<a href="#">View Chat Log</a>
			<a href="#">View Notes</a>

Otto's Clone Army (#43110)			
Member Name	Change the Leader	Change Member Permissions	Remove this Member
OttoBorchert48	Leader	<a href="#">Change Permissions</a>	<a href="#">Remove</a>
OttoBorchert41	<a href="#">Set As Leader</a>	<a href="#">Change Permissions</a>	<a href="#">Remove</a>
OttoBorchert42	<a href="#">Set As Leader</a>	<a href="#">Change Permissions</a>	<a href="#">Remove</a>
OttoBorchert43	<a href="#">Set As Leader</a>	<a href="#">Change Permissions</a>	<a href="#">Remove</a>
OttoBorchert44	<a href="#">Set As Leader</a>	<a href="#">Change Permissions</a>	<a href="#">Remove</a>
OttoBorchert45	<a href="#">Set As Leader</a>	<a href="#">Change Permissions</a>	<a href="#">Remove</a>
OttoBorchert46	<a href="#">Set As Leader</a>	<a href="#">Change Permissions</a>	<a href="#">Remove</a>
OttoBorchert49	<a href="#">Set As Leader</a>	<a href="#">Change Permissions</a>	<a href="#">Remove</a>
OttoBorchert30	<a href="#">Set As Leader</a>	<a href="#">Change Permissions</a>	<a href="#">Remove</a>
OttoBorchert31	<a href="#">Set As Leader</a>	<a href="#">Change Permissions</a>	<a href="#">Remove</a>
OttoBorchert35	<a href="#">Set As Leader</a>	<a href="#">Change Permissions</a>	<a href="#">Remove</a>
OttoBorchert40	<a href="#">Set As Leader</a>	<a href="#">Change Permissions</a>	<a href="#">Remove</a>
OttoBorchert47	<a href="#">Set As Leader</a>	<a href="#">Change Permissions</a>	<a href="#">Remove</a>
OttoBorchert33	<a href="#">Set As Leader</a>	<a href="#">Change Permissions</a>	<a href="#">Remove</a>
Information and Actions			
Current Goal: Nothing (#-1)			
<a href="#">Add Member</a>	<a href="#">Disband Group</a>	<a href="#">Set Name</a>	<a href="#">Set Goal</a>
		<a href="#">Delete Group</a>	<a href="#">View History</a>
			<a href="#">View Chat Log</a>
			<a href="#">View Notes</a>

Group 1 (#55599)			
Member Name	Change the Leader	Change Member Permissions	Remove this Member
ThisIsAOHara	Leader	<a href="#">Change Permissions</a>	<a href="#">Remove</a>
OttoBorchert34	<a href="#">Set As Leader</a>	<a href="#">Change Permissions</a>	<a href="#">Remove</a>
Information and Actions			
Current Goal: Nothing (#-1)			
<a href="#">Add Member</a>	<a href="#">Disband Group</a>	<a href="#">Set Name</a>	<a href="#">Set Goal</a>
		<a href="#">Delete Group</a>	<a href="#">View History</a>
			<a href="#">View Chat Log</a>
			<a href="#">View Notes</a>

*Figure 3: Instructor WWW Interface*

Students are able to form groups themselves as well. This provides for the “interest groups” method of group composition. Students create a new group, then search for friends by login name or by a series of options, which include whether they are in the same room, in the same class, or have the same goal.

Finally, students can be placed into groups heterogeneously. After completing the pre-assessment portion of the game, students are asked to wait a predetermined amount of time for everyone to finish the pre-assessment. At this deadline, students are placed into groups based on demographic data and pre-assessment scores, with a high achieving student, matched with a lower achieving student.

## Cooperative Goals

The modules in the Geology Explorer have also been re-written to conform to Kagan's PIES principles. In the rock and mineral identification goal, students are asked to score 800 points, with each student needing to contribute at least 300 points to the group effort. The point structure remains the same; Students get 100 points for correctly identifying a primary goal and 25 points for identifying other outcrops.

In the geologic mapping exercise, students work together to identify one outcrop of each type, then work on the geologic map cooperatively. Students take turns drawing on the map using a token interface. While one student has the token, the other is able to view the map being drawn and can chat with other player, working together to complete the map.

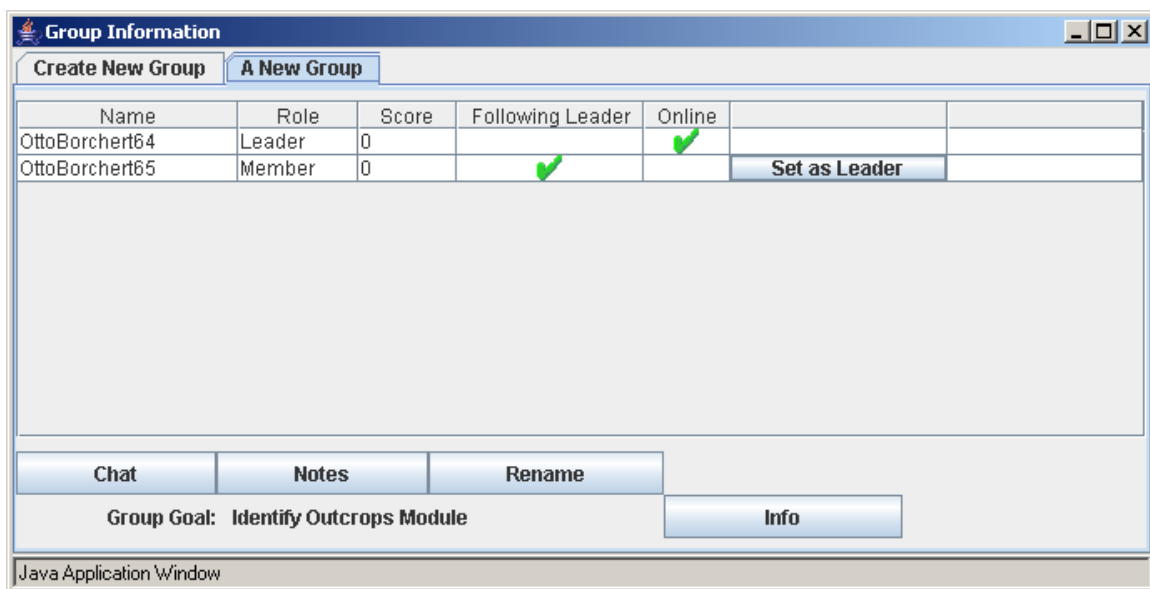


Figure 4: The Main Group Interface

## Experiment

There have been two major experiments testing the efficacy of the Geology Explorer group interface. Both were performed at North Dakota State University as a required (graded) assignment in a freshman level geology course. The first was performed in the Fall of 2004 (Brandt, et al., 2006). This pilot study had two main functions: 1) to act as a testbed for the newly developed interface, so that any errors could be corrected for the full experiment and 2) to determine initial completion rates which were used to measure motivation and 3) examine inter-student conversations, which were measured in words typed. A total of 309 students played the individualized version of the Geology Explorer, while 19 groups of two volunteered to test the group interface.

No formal statistics was performed on the fall 2004 data, but results appeared promising, and the pilot study revealed a number of issues that were present in the technology. In Brandt et al. (2006), it was shown that all of the cooperative players finished the game,

while only 90% finished in the individualized group. Some initial data also pointed to added virtual collaboration between group members.

Students are able interact by double clicking on their avatar to make an announcement to the whole room, or by double clicking on another person's avatar to talk to just that person. This results in a “text balloon” message being sent. Also, group students are able to chat with each other in a chat room dedicated to group interactions. These interactions are listed as “Group Interactions”.

In fall 2006, another experiment was completed to test the efficacy of the learning goals to teach geological concepts (Borchert, 2008). In this experiment, 224 students worked as individuals while 177 worked in teams of two.

There were three extra modules added between 2004 and 2006 which explain the added rows in 1: embedded assessments, the Intro Goal, and the true thickness module. In 2004, assessments were given via a web page interface, while in 2006, these assessments were embedded within the game itself. The Intro Goal gave students an opportunity to learn how to use the interface. Finally, the True Thickness module added some trigonometry and rock orientation exercises to the Geology Explorer. These three exercises were done individually even when players worked as a team.

	Solo Group			Team Group		
	Actual	Exp	Chi <sup>2</sup>	Actual	Exp	Chi <sup>2</sup>
Never Logged In	4 (1.8%)	3.35	0.125	2 (1.1%)	2.65	0.159
Preassessment	3 (1.3%)	2.79	0.015	2 (1.1%)	2.21	0.019
Intro Goal	17 (7.6%)	11.73	2.367	4 (2.3%)	9.27	2.995
Identification Goal	27 (12.1%)	17.32	5.415	4 (2.3%)	13.68	6.853
Interpretive Module Goal	14 (6.2%)	13.41	0.026	10 (5.6%)	10.59	0.033
True Thickness Goal	15 (6.7%)	13.97	0.077	10 (5.6%)	11.03	0.097
Finished the Game	144 (64.3%)	161.44	1.883	145 (81.9%)	127.56	2.383
Total Students	224			177		

Table 1: Experimental Completion Rates – Fall 2006

In the Fall 2006 experiment, a chi-squared test for homogeneity was performed on the completion rates to determine if there was a statistical difference between the control and experimental groups. These results are shown in 1. With DF = 6, the chi-square value was 22.449, p = .001. These results show that the experimental group completion rates were statistically different from the solo completion rates, with a majority of the difference coming from students that stopped at the intro and identification goals and those that finished the game completely.

The more people completing the game should correlate to better understanding of geology, as those students would have been immersed in the game world longer. As expected, more team students completed the exercise than solo players. Team players are more likely to push each other into completing the task, resulting in higher motivation.

Question	Df	Chi <sup>2</sup>	p-Value	Team	Improve	Pre	Post	N
3.1	2	2.13	0.344	Y	7.41	92.59	95.37	108
				N	3.64	96.36	96.36	165
3.2	3	4.54	0.209	Y	5.56	91.67	93.52	108
				N	7.27	88.48	86.06	165
3.3	3	3.16	0.368	Y	12.96	68.52	75.00	108
				N	18.18	68.48	76.97	165
3.4	3	1.07	0.783	Y	10.19	82.41	87.96	108
				N	8.48	86.67	90.30	165
3.5	3	0.90	0.825	Y	14.81	84.26	97.22	108
				N	13.33	84.24	95.76	165
3.6	3	0.23	0.972	Y	1.85	97.22	95.37	108
				N	1.21	97.58	95.15	165
3.7	3	5.03	0.170	Y	8.33	6.48	10.19	108
				N	3.64	4.85	6.67	165
3.8	3	5.60	0.133	Y	29.63	38.89	50.00	108
				N	18.18	46.06	46.06	165
3.9	3	1.20	0.754	Y	8.33	86.11	81.48	108
				N	10.30	85.45	86.06	165
3.10	3	4.93	0.177	Y	9.26	87.96	89.81	108
				N	3.64	95.15	92.12	165

*Table 2: Homogeneity Results for Content Improvement by Team Effect – Fall 2006*

Content knowledge acquisition was also studied in the 2006 experiment with the results being shown in 2. Improve(ment) is the percentage of students that improved from the pre-treatment to the post-treatment. A chi-square test was performed to determine if this improvement was statistically significant between the individual and team groups. Both groups did better on the questions overall, but there was no statistical difference between the groups. This means that whether students played individually or as a group, they were helped by the Geology Explorer interface. Since students were self-selecting in the experiment, future research needs to be done to determine if certain learning styles or behavior patterns emerge as more effective for group learning.

## Acknowledgements

This research was supported by National Science Foundation grants ESI-0454767 and GEO-0608082. Human subject testing was conducted under NDSU IRB Protocol SM-07070 (replacing SM-98026)..

## References

Aronson, E., Stephan, C., Sikes, J., Blaney, N., & Snapp, M. (1978). The Jigsaw

Classroom. Beverly Hills, CA: Sage.

Baron, R. S. (2005). So Right It's Wrong: Groupthink and the Ubiquitous Nature of Polarized Group Decision Making, In M. P. Zanna (Ed.), *Advances in Experimental Social Psychology*, San Diego, CA: Elsevier Academic Press, 219-253.

Borchert, O. (2008, in press). Computer Supported Collaborative Learning in a Geologic Simulation. Master's Thesis. North Dakota State University.

Brandt, L., Borchert, O., Addicott, K., Cosmano, B., Hawley, J., Hokanson, G., Reetz, D., Saini-Eidukat, B., Schwert, D., Slator, B. Tomac, S. (2006). Roles, Culture, and Computer Supported Collaborative Work on Planet Oit, *Journal of Advanced Technology for Learning*, 3(2), 89-98.

Brown, K., & Cole, M. (2000). Socially-shared cognition: System design and the organization of collaborative research, In D. Jonassen and S. Land (Eds.), *Theoretical foundations of learning environments*, Mahwah, NJ: Lawrence Erlbaum, 197-214.

Brown, J. S., Collins, A., & Duguid, P. (1989). Situated Cognition and the Culture of Learning, *Educational Researcher*, Washington, DC: AERA, 18 (1), 32-42.

Bruce, B., Peyton, J., & Batson, T. (1993). Introduction, In P. Bruce and T. Batson (Eds.), *Network-Based Classrooms: Promises and Realities*, Cambridge, UK: CUP, 1-6.

Curtis, P. (1997). *High Wired: On the Design, Use and Theory of Educational MOOs*. University of Michigan.

Feltovich, P., Spiro, R. J., Coulson, R. L., & Feltovich, J. (1996). Collaboration within and among minds: Mastering Complexity, Individually and in Groups, In T. Koschmann (Ed.), *CSCL: Theory and Practice: An Emerging Paradigm*, Mahwah, NJ: Lawrence Erlbaum Associates, Inc., 38.

Goldman, S. V. (1996). Mediating Microworlds, In T. Koschmann (Ed.), *CSCL: Theory and Practice: An Emerging Paradigm*, Mahwah, NJ: Erlbaum Associates, Inc., 57.

Hokanson, G., Borchert, O., Slator, B. M., Terpstra, J., Clark, J. T., Daniels, L. M., Anderson, H. R., Bergstrom, A., Hanson, T. A., Reber, J., Reetz, D., Weis, K. L., White, R., & Williams, L. (In Press, 2008). Studying Native American Culture in an Immersive Virtual Environment. *Proceedings of the IEEE International Conference on Advanced Learning Technology (ICALT-2008)*. IEEE Computer Society Press. Santander, Spain. July 1-5.

Janis, I. L. (1972). *Victims of Groupthink*. Boston, MA: Houghton Mifflin Company, 8-9.

Koschmann, T., Kelson, A. C., Feltovich, P. J., & Barrows, H. S. (1996). Computer Supported PBL, In T. Koschmann (Ed.), *CSCL: Theory and Practice: An Emerging Paradigm*, Mahwah, NJ: Lawrence Erlbaum Associates, Inc., 100.

McClellan, P., Saini-Eidukat, B., Schwert, D., Slator, B., & White, A. (2001). Virtual Worlds in Large Enrollment Science Classes Significantly Improve Authentic Learning, In J.A. Chambers (Ed.), *Selected Papers from the 12th International Conference on College Teaching and Learning*, Jacksonville, FL: Center for the Advancement of Teaching and Learning, 111-118.

Middlecamp, C. (1997). Students Speak out on Collaborative Learning, retrieved August 31, 2007, from <http://www.wcer.wisc.edu/archive/CL1/cl/story/middlecc/TSCMD.htm>.

National Association of Colleges and Employers. (2003). *Job Outlook 2004: Student Version*, retrieved March 11, 2005, from <http://www.jobweb.com/joboutlook/2004outlook/JO04student.pdf>.

Kagan, S. (1994). *Cooperative Learning*, San Juan Capistrano, CA: Kagan Cooperative.

Regan, P. & Slator, B. Case-based Tutoring in Virtual Education Environments, In W. Broll, C. Greenhalgh, & E. F. Churchill (Eds.), *Proceedings of the 4th International Conference on Collaborative Virtual Environments*, Bonn, Germany: ACM, 2-9.

Scardamalia, M., Bereiter, C., & Lamon, M. (1994). The CSILE Project: Trying to Bring the Classroom into World 3, In K. McGilly (Ed.), *Classroom Lessons: Integrating Cognitive Theory and Classroom Practice*, Cambridge, MA: MIT Press, 201-228.

Sharan, S. and Sharan, Y. (1978). *Small Group Teaching*, Englewood Cliffs, NJ: Educational Technology Publications, 10-19.

Sharan, Y., & Sharan, S. (1994). Group Investigation in the Cooperative Classroom, In S. Sharan (Ed.), *Handbook of Cooperative Learning Methods*, Westport, CT: Greenwood Press, 115-136.

Skinner, B., Porter, S., & Park, J. (2004). *Dynamic Earth: An Introduction to Physical Geology: 5th Edition*, Hoboken, NJ: John Wiley and Sons, Inc., 232-233.

Slator, B. M., Hill, C., & Del Val, D. (2004). Teaching Computer Science with Virtual Worlds, *IEEE Transactions on Education*, 47(2), 269-275.

Slator, B. M., Beckwith, R., Brandt, L., Chaput, H., Clark, J. T., Daniels, L. M., Hill, C., McClellan, P., Ograde, J., Saini-Eidukat, B., Schwert, D. P., Vender, B., & White, A. R. (2006). *Electric Worlds in the Classroom: Teaching and Learning with Role-Based Computer Games*, New York, NY: Teachers College Press, 86-93.

Slavin, R. E. (1995). *Cooperative Learning: 2nd Ed*, Boston, MA: Allyn and Bacon, 50-69.



# The Role of Writing Efficient Programs in a Data Structures Course

Chenglie Hu  
Department of Computer Science  
Carroll College  
100 N East Avenue, Waukesha, WI 53186  
chu@cc.edu

## Abstract

How can writing efficient code play an effective role in improving student programming skills and developing professional consciousness of code efficiency in a data structures course? This article attempts to answer this question. First, some basic strategies of code efficiency are overviewed. Some guidelines of improving code efficiency in implementing data structures are then presented with examples. These guidelines are categorized in two levels: method level and data representation level. The article is concluded with a summary and some comments.

# 1. Introduction

In the real world, a programmer has two concerns: code correctness and code efficiency. Anyone would agree that a piece of code must be, first of all, correctly does what it is supposed to do. Yet, the author has observed over the years in his data structures course (or CS2) that students often stop right there once the code has worked, even though writing efficient code is one of the evaluation criteria of their assignments. How is the efficiency of a computer program measured? Probably, the most common measurement is the consumption of CPU time and internal memory. However, the advancement of computer hardware has made the consumption level of a program on hardware resources much less of a concern than it was, say, 20 years ago. As a result, writing efficient programs might have been pushed farther back in programming classes so that learning algorithm analysis with big oh in a data structures class may be more of a thing that the instructor must cover, with often standard or even trivial examples, than something students actually use in their later programming activities.

Although network transport delays, large volume data access, and slow remote component execution can all contribute negatively to the efficiency of a program execution, writing efficient code can make such an impact minimized. In the real-world, people in fact are concerned more about inefficient ways of writing programs today than ever before as the size and complexity increase rapidly in developing modern software while facing high demand for compact devices with very constrained memory capacity and data transmission bandwidth. The author has seen little recent published material that has the same theme intended by this article.

Over the years, the author has faced two problems in his data structures classes. First, by the time students are in a data structures class, most likely, they have already known well the array-based list implementations as object-oriented programming examples such as a list of DVDs or bank accounts are now common in CS1 classes. As a result, the treatment on array-based lists (and, in fact, array-based queues and stacks as well) normally seen in textbooks may simply not provide materials that are challenging and refreshing to the students. Second, students are, most of time, learning how various data structures are implemented rather than how implementations can be done or how students would do if asked to implement a data structure on their own. According to [1], a data structure should be addressed from logical, application, and implementation standpoints. The author believes that students have a better chance to gain programming skills by letting them focus more on the implementation level than on the application level of a data structure, as the former tends to address more programming-in-the-small. Other than iteration verses recursion, textbooks don't usually seriously talk about algorithm comparison until students learn sorting methods. The author has tried to fill this gap by incorporating writing efficient algorithms early on so that students are not only exposed to various programming techniques while learning how to write efficient code, but they also have their own examples to practice on with big oh analysis. This learning process, indeed, is not an easy one for every student. But most students felt, expressed on the course evaluations, that it is a rewarding learning experience.

## 2. Basic Code Efficiency Concerns

Arguably or not, learning how to write efficient code effectively improves one's programming skills. The primary concern of a typical novice programmer is, understandably, whether the code he or she comes up with works. Thus, novices tend to write programs that can often be improved on inefficiency. As summarized in [3] and [4] and extended by the author, the following problems can all cause inefficient programs.

*Redundant operations in a loop* This happens when certain constants (or constant expressions) are repeatedly computed as demonstrated in the following loop:

```
for(int i = 0; i < result.length; i++) results[i] = (a*a*a + b)*i + (a*a + c)*(i + 1);
```

Obviously, the coefficients of  $i$  and  $(i+1)$  should have been computed once for all before the loop is entered.

*Unnecessary referencing* The problem can be demonstrated in the following example of selecting the maximum element in an array:

```
int p = 0;
for(int i = 1; i < arr.length; i++) if (arr[i] > arr[p]) p = i;
max = arr[p];
```

where the referencing to  $a[p]$  could have been saved as in the standard approach that  $max$  holds  $arr[0]$  initially and gets updated once  $arr[i] > max$ .

*Using expensive operations when less expensive ones are available* For example, the following two loops are equivalent. Yet, the second version is a bit more efficient when addition, instead of multiplication, is used to update  $x$ .

```
for(int i = 0; i <= N; i = i + 2){ x = 5*i; y = a*x + b; .... }
x = 0; for(int i = 0; i <= N/2; i ++){ y = a*x + b; x = x + 10; .... }
```

*Using too many weakly guarded statements* Introducing unnecessary guards or conditional statements often makes correctness checks easier. For example, in selecting a maximum amongst three numbers ( $num1$ ,  $num2$ , and  $num3$ ), novices often write:

```
if(num1 >= num2 && num1 >= num3) max = num1;
else if(num2 >= num1 && num2 >= num3) max = num2;
else if(num3 >= num1 && num3 >= num2) max = num3;
```

when the following statement, in fact, suffices:

```
if(num1 >= num2 && num1 >= num3) max = num1;
else if(num2 >= num3) max = num2;
else max = num3;
```

which is, however, less obvious as far as correctness is concerned.

*Inefficiency due to late termination* An example of this is when linear search is applied to a sorted array as in the following code fragment (assuming array `intArr` is in an ascending order):

```
int i = 0; boolean found = false;
while(!found && i < intArr.length){ if(x == intArr[i]){ found = true; break; } else i++; }
```

The code can be written more efficiently as follows to eliminate unnecessary comparisons for an unsuccessful search.

```
int i = 0; boolean found = false;
while(!found && i < intArr.length){ if(x > intArr[i]){ i++; continue; }
found = (x == intArr[i]); break; }
```

*Inefficiency due to extra layers of code* To implement an iterator for a binary tree, as in [1], one may pre-load elements into a queue using one of the tree traversal methods, and then process the node data one element at a time using the methods *getNext* and *next*. Thus, invoking an iterator with *getNext* and *next* adds an additional layer to the code. A more efficient way is to skip the intermediate storage and process the data while traversing.

Some of the basic strategies of writing efficient code, as summarized above, are presented to students very early on in the course, and more are added as the course progresses. Being able to implement efficient data structures, however, requires more than the basic strategies. Further guidelines, whenever deemed appropriate for improving code efficiency, are also introduced early in the course when students are learning basic data structures such as lists, stacks, and queues. These guidelines are presented in section 3.

### 3. Some Guidelines for Developing Efficient Data Structures

For implementing efficient data structures, students need more data handling and processing guidelines beyond the ones mentioned in section 2. Although their applicability goes beyond what is presented, the following guidelines, stated in the subtitles, are presented in conjunction with list data structure implementations. These guidelines are elaborated with or without examples as appropriate. It is also shown that when consciously writing code with efficiency in mind, students are learning big oh analysis in a meaningful way.

#### 3.1 Reusing existing methods may not be a good idea

One of the software design principles is to improve the reusability of software components. Yet, reusing existing methods may introduce operations overhead, especially when reused modules involve costly operations such as loops. Here is an example (see also Chapter 5 Exercises in [1]): Write a method for the sorted linked list class to delete all occurrences of a given item and return the number of copies deleted (assuming that the list allows duplicated items). The method can be conveniently implemented as follows using existing methods *isThere* (searching for a given item) and *delete* (deleting an item from the list):

```
public int deleteCopies(Comparable item){
    int numCopies = 0;
    while( isThere(item) ){
        delete(item);
        numCopies++;
    } return numCopies;
}
```

On average, the method *delete* requires  $O(N)$  comparisons and so does the method *isThere* (as binary search is not normally applicable to linked lists). If there are  $K$  copies of the item to be deleted, the total cost is then roughly  $O(KN)$ . The cost would be  $O(N^2)$  if  $K$  were in the same order of magnitude as  $N$ . However, if instead, one first identifies the immediate predecessor of the first occurrence and the last occurrence of *item* to be deleted, and then do the “rewiring” by skipping all the nodes in between (i.e., all copies of *item*), the average cost for comparison would be reduced to  $O(N)$  (with a single pass of the list) as the following computation shows:  $[1 + 2 + \dots + (N-K)]/(N-K) + K = (N + K + 1)/2 = O(N)$ .

```
public int deleteCopies(Comparable item){
    ListNode start, end, cursor = list;    //list is the list-head reference
    int numCopies = 0;
    while( item.compareTo(cursor.info) != 0 && cursor.next != null ){
        start = cursor;
        cursor = cursor.next;
    }
    while( item.compareTo(cursor.info) == 0 && cursor.next != null ){
        end = cursor; cursor = cursor.next;
        numItems--; numCopies++;
    }
    if(start == null) list = cursor;
    else start.next = end.next;
    return numCopies;
}
```

It’s worth mentioning that students did come up with other strategies, such as saving the items (not to be deleted) in a new list using *insert* method, or saving the items with a new list by “manually” linking the copies of the nodes rather than using the method *insert*. The pros and cons of those variations are then discussed and evaluated in class.

## 3.2 Looking for inexpensive ways to reduce or delete non-determinism

Non-determinism often causes one to apply multiple conditions (or guards) in an if-else statement as demonstrated in the following partial implementation of a method (see also Chapter 3 Exercises in [1]) that trims the list given lower and upper bounds. The actual positions of the lower and upper bounding items are unknown. If these positions remain unknown during the processing of the array, the following loop may seem a reasonable attempt by most students:

```
for(int i = 0; i < numItems; i++){
    if(list[i].compareTo(lower) < 0 || list[i].compareTo(upper) > 0)
        { delete(list[i]); }
}
```

This would result in  $O(N^2)$  count for the number of comparisons due to overhead operations from *delete*. Alternatively, if the opposite of “if” condition holds, one may save *list[i]* to a local array and assign the reference to *list* at the end, which could reduce the number of comparisons to  $O(N)$  with an introduction of additional storage.

However, if instead, one searches for the lower and upper bounding items first so that their positions become known (i.e., the elimination of non-determinism), one may have a better chance to devise a more efficient method. The following code fragment articulates the idea (note that when method *isThere* finds the item, it stores its index in an instance variable *currentPos*):

```
if( isThere(lower) ) lowerPos = currentPos;
if( isThere(upper) ) upperPos = currentPos;
for(int i = lowerPos; i <= upperPos; i++) list[i-lowerPos] = list[i];
numItems -= upperPos - lowerPos + 1;
```

By taking advantage of binary search, the cost for comparison now is further reduced to  $O(\log N)$  with no local array needed.

This example (and, in fact, the implementation of the method *deleteCopies* in subsection 2.1 falls in this category as well) shows the benefit of reducing or eliminating non-deterministic information related to the given parameters, if it can be done inexpensively. As a more general remark, it is often the case that when a data processing involves multiple criteria such as items being duplicated in some form, all within a given range, or all in relation to a given reference, etc., determining the exact range of the data pieces may often be a starting point of more efficient algorithms if the data is sorted in one way or another.

## 3.3 The way how data is stored can make a difference

This guideline is explained, again, with implementations of a sorted list. It is well known that for an array-based implementation, insertion or deletion requires possibly heavy data shifting for leaving or filling a gap. Shifting becomes costly when a large amount of data needs to be frequently inserted and deleted. While the loops of such implementations can hardly be improved, the efficiency can, nonetheless, be altered by changing the way data is stored. The author has attempted in his data structures course three different strategies in doing so with an array-based sorted list: letting the list head float, introducing gaps, and using a 2-dimensional array. The three strategies vary significantly in implementation difficulty, and so does the efficiency impact on list processing. They are outlined as follows.

*Floating list head*                      If the head of the list is the first element of the array, inserting or deleting items near the head is more costly due to the resulting shifts of almost the entire list. A simple idea to alleviate the situation is to make the head of the list float (and so does the tail of the list). In other words, the list would start at index *head*, grow clockwise and wrapped around, and end at index *tail* (which may or may not be greater than *head*). Since the head index can now move in both directions, the most expensive shifts with a fixed head now become among the least expensive ones. In exchange, the implementation becomes a bit trickier (although a good exercise for students) especially for binary search when the range between *head* and *tail* must be mapped to a range between 0 and *list.length-1*, required for the binary search.

*Introducing gap elements* [4]                      “Gaps” are defined to be place holders between items of the list. One possible configuration of the list-holding array might be (*gap, item, gap, item, ..., gap, item*). Consecutive gaps can be filled with the duplicates of the last item before them so that binary search is not affected. To identify the gaps correctly, another array (of the same size) is generally needed, whose elements would indicate whether a corresponding location in the list-holding array is a list item or a gap (either a Boolean or bit value should suffice for such identification). An insertion can be done without shifting if the new item fills an existing gap or with a few shifts if there is a nearby gap. When shifting becomes problematic (using a tolerability threshold or a measure suggested in [4]), one can re-configure the array by either restoring the original configuration or simply introducing more scattered gaps as appropriate. The implementation of such a variation, however, is considerably more challenging than that of floating head approach outlined earlier, yet still manageable by most students.

*Using a 2-dimensional (2D) array*                      The idea of using a 2D-array container for linear data structures comes from the fact that 2D arrays are represented using row objects. In other words, a 2D array is an array of arrays in most object-oriented languages such as Java and C#. Such a representation is especially beneficial when it comes to array resizing. More specifically, an array can be resized either by introducing additional rows or additional columns in a specific row. For the former, only the row references of the old array need to be copied over to the new row reference variables, whereas for the latter, only elements in that particular row being resized need to be copied over after a new row is instantiated. In either case, only a subset (small usually) of the elements is affected when resizing – an operation often associated with insertion.

How may the amount of shifting be reduced using a 2D array? Figure 1 gives an example of a (row-wise) sorted list in a 2D array configuration. If an integer, say 6, were to be inserted, one would either place it at the end of the second row without shifts or the beginning of the third row with a few shifts (and resizing the row if necessary). Similarly, if integer 25 were to be inserted, one would either simply attach it at the end of last row or introduce a new row (i.e., the sixth row) and place it at the beginning.

```

0
2  3  5
7  8
10
12 15 19

```

**Figure 1: A sorted list of integers stored in a 2D array**

How would a sorted list grow in the first place? Different strategies exist. In particular, one may initially fill up the first column with a fixed number of rows. The list then grows horizontally as individual rows are resized as needed. Binary search still works but in a different fashion. For instance, if integer item 8 in Figure 1 is searched for, one may simply perform a binary search in the first column (to find item 7) followed by another binary search in the row that starts at 7. The big oh analysis suggested that such a binary search could be even quicker than its 1D counterpart. In fact, suppose a 2D array has  $Nr$  rows and the average number of elements of a row is  $Nc$ . The size of the list is then  $N = Nr * Nc$ . It can be shown (see [5] by the same author for details) that the average number of comparisons is asymptotically less than  $\log_2 Nr + \log_2 Nc = \log_2 (Nr * Nc) = \log_2 N$ . However, the actual performance showed the opposite – 1D binary search performed slightly better (about 1.7 times faster) in most of the testing cases due possibly to the additional cost of reference arithmetic in the 2D case.

Because of the increased complexity of storage dimension, such an implementation is even more challenging than introducing gaps, yet still within the reach of most CS2 students. Some implementation details are given in [5]. For a particular implementation, as it turned out, insertion or deletion using a 2D array takes only about 20%-25% of the time spent on the same operation using a 1D array in all of the testing cases. The other two approaches performed not as competitively, although both are improvements from the standard implementations.

## 4. Conclusions

It is demonstrated above how emphasizing on writing efficient code can play an effective role in improving student programming skills and consciousness about writing code that performs better. The benefits can be summarized as follows.

The approach raises student's consciousness about writing efficient code significantly. Since the code efficiency is addressed right from the beginning of the course, by the second half of the semester, students are usually able to show significant improvement on



code efficiency when working on their assignments. Many indicated that the one handed in seemed to be the best he or she came up with (although it might not be the case). When the implementation of a new data structure or an algorithm is presented to them, there are always students questioning about whether there are other implementation options that may result in a better performance. For instance, when the standard insertion sort is presented to them, some questioned why we don't take advantage of binary search for the sorted portion of the list. They were then asked to do so and show the result both in theory (with big oh analysis) and practice (by comparing it with standard insertion sort algorithm and some of the  $O(N \log N)$  sorting methods). It turned out that the insertion sort incorporated with binary search has, in fact, a comparable performance with the best  $O(N \log N)$  sorting method (covered in [1]) in practice and is also  $O(N \log N)$  in theory. Students are also asked to compare the performances between the list structures they had implemented and the ones in Java Collections. Greater motivation comes along naturally when students are seeing that their implementations are beating *ArrayList* and, by far, *LinkedList* in performance.

The approach creates a “real-world” environment for learning big oh algorithm analysis. Big oh is learned beyond the level of dealing with standard examples or using purely imaginary “polynomial” times. Big oh becomes a necessary tool for students to address the efficiency of the algorithms they have attempted, although much of the analysis is more or less heuristic rather than relying on solving recurrence equations with mathematical rigor, given the level of CS2.

The approach addresses the importance of the way data is stored and retrieved, which, in turn, reinforces the objectives of a data structures course. Even with the superior support of a class library, writing your own data structures is not a rare event for various reasons in contemporary software development. Using a data structure is relatively easy, but writing one on your own is generally not, and being able to write an efficient one is even harder, which, however, is what a data structures course is all about.

It should be pointed out that there are other areas of program efficiency that could be included in a data structures course. For example, to implement doubly linked list, reference [6] presented a memory efficient approach – defining the node structure using only one pointer (*ptrdiff*) that captures the difference between the pointer to the next node (*next*) and the pointer to the previous node (*previous*) using exclusive OR operator instead of using two pointers as in the conventional implementations. It is interesting to observe that pointer *next* (or *previous*) can then be recovered by XORing *previous* (or *next*) with *ptrdiff*. The saving may seem insignificant, but may be critical for developing software to be installed on compact devices that often have very constrained memory capacity. Such examples may also constitute good discussion topics for highly motivated students.

There is no doubt that the techniques and methodologies mentioned in this article expose students to some of the programming skills that they need to develop, yet may be poorly prepared for in CS1 classes. Because instructors, nowadays, are overwhelmed by the materials they need to cover in CS1 classes so that basic algorithms are increasingly not

sufficiently covered. Although group projects in CS2 can alleviate some of the problems, something essential needs to be addressed in CS1 to improve student code-writing skills.

In closing, as the author believes, the competency of writing efficient codes should in fact be an important quality indicator of our computer science graduates as “the fundamental question underlying all of computing is ‘what can be (efficiently) automated’?” [2].

## References

- [1] Dale,N. Joyce,D. & Weems, C. *Object-Oriented Data Structures Using Java*. Jones and Bartlett Publishers, 2002
- [2] Denning, P. Great Principles of Computing. *Comm. ACM* 32, 1, 9-23, 1989
- [3] Dromey, R.G. *How to Solve it by Computer*. Prentice-Hall, Englewood Cliffs, N.J. 1982
- [4] Gries, D. *The Science of Programming*. Springer-Verlag, New York 1981
- [5] Hu, C. 2D Array Implementation of Sorted Lists, *SIGCSE Bulletin (Inroads)*, Vol. 37, Issue 2, 2005
- [6] Sinha, P. A Memory-Efficient Doubly Linked List. *Linux Journal* 2005(129): 10. 2005

# **Effectively Apply Boundary Value Analysis Method in Students' Programs Testing**

**Syed (Shawon) M. Rahman, Ph. D.**

Assistant Professor, Dept. of Computer Science & Software Engineering

University of Wisconsin - Platteville

1 University Plaza, Platteville, WI 53818, USA

Phone: (608) 342-1625, Fax: (608) 342-1965

Email: [Rahmans@uwplatt.edu](mailto:Rahmans@uwplatt.edu)

## **Abstract**

Boundary Value Analysis (BVA) is a very effective test case design technique and generally, a popular black-box testing method in software testing. The purpose of BVA is to focus the testing effort on error-prone areas by accurately pinpointing the boundary. At those points when input values change from valid to invalid or vice-versa errors are most likely to occur. Unfortunately, most of the textbooks end up presenting BVA by providing a numeric value range; however, in the real world, the computer programs are not restricted to the numeric data ranges. Our experience shows that students encounter difficulty while finding and applying BVA beside a numeric value range. In this paper, we have presented examples how to define test cases and apply BVA effectively for much wider areas and can reveal critical faults and failure in the program.

## Boundary Value Analysis Testing Technique

Boundary value analysis is a test case design technique and it is a very effective method for revealing bugs in computer programs. Boundary Value Analysis is well known as off-by-one errors technique testing, which is nearly all those boundaries where the defects are [1]. In general, there are two types of boundary values: boundary value for continuous range of inputs and boundary value for discontinuous range of inputs. However, a continuous range of inputs considers more suitable kind of input for Boundary Value Analysis. BVA testing method especially are very suitable for function black-box testing and unit testing technique.

Guidelines for BVA are close to those for equivalence partitioning [2]:

1. If an input condition specifies a range bounded by values “a” and “b”, test cases should be produced with values “a” and “b”, just above and just below “a” and “b”, respectively.
2. If an input condition specifies various values, test cases should be produced to exercise the minimum and maximum numbers.
3. Apply guidelines above to output conditions.
4. If internal program data structures have prescribed boundaries, produce test cases to exercise that data structure at its boundary.

### Example of BVA

Boundary values are of special interest in software testing; especially the boundaries of input ranges to a software component are likely to defects. Testing experience shows that test cases that explore boundary conditions have a higher payoff than test cases that do not. Boundary value analysis requires one or more boundary values selected as representative test cases [3].

As an example, a programmer implement the range 1 to 12 at an input, which stands for the month January to December in a date, has in his code a line checking for this range [4]. We can write this condition as follows:

```
if (month > 0 && month < 13)
```

However, a common programming error may check a wrong range e.g. starting the range at 0 by writing or ending at 11,

```
if (month >= 0 && month < 13) or, if (month > 0 && month < 12)
```

Normally, for more complex range checks in a program this may be a problem which is not so easily spotted as in the above simple example. To set up boundary value analysis test cases we need to determine first which boundaries we have at the interface of a software component. This has to be done by applying the equivalence partitioning technique. As we know BVA and equivalence partitioning (EP) are

inevitably linked together. For the example of the month in a date we have the following partitions [3]:

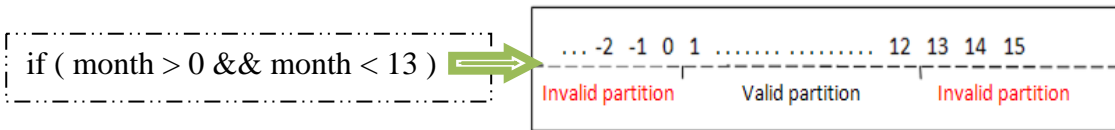


Figure 1: Showing how to determine equivalence class [3].

### Determine Equivalence Classes

Equivalence class is a group of value, which is every value in the group cuts down a number of test cases while it still covers all test cases. Besides reducing the number of test cases, in each test case is expected to cover and uncover the same defect. As we mentioned before, if an input condition specifies a range bounded by values “a” and “b”, test cases should be produced with values “a” and “b”, just above and just below “a” and “b”, respectively. According to figure1, this example presents a set of months in a year, so one set of valid input (valid partition) is {1,2,3,4,5,6,7,8,9,10,11,12} while two sets of invalid inputs (invalid partition) are {...,-2,-1,0} and {13,14,15,...} [3].

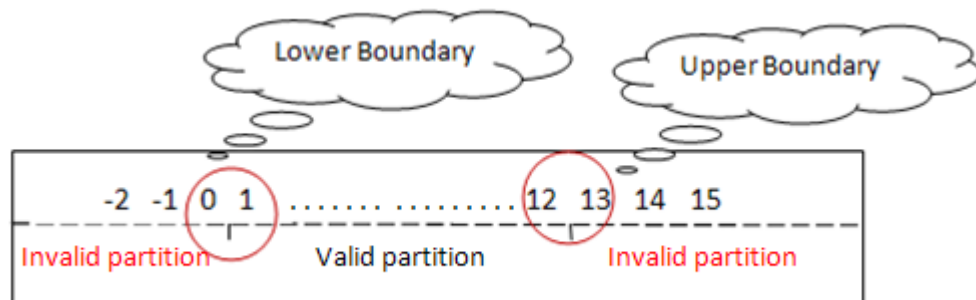


Figure 2: Showing how to determine boundaries [3]

### Determine Boundaries

After identifying the equivalence classes the next step we identify the lower boundary and the upper boundary. According to Figure2 above, the lower boundary would be 0 and 1 and the upper boundary would be 12 and 13.

We would like to mention here that there are two types of boundary values, which are Boundary Values for continuous range (as in Figure 2 and 3) and Boundary Values for discontinuous range (Figure4). According to Figure 4, it is possible that a set of value in boundary values are not continuous as a result, in Figure 4 shows that

1 and 5 are boundary values whereas a set of {0, 2, 3, 4, and 6} considers invalid partition.

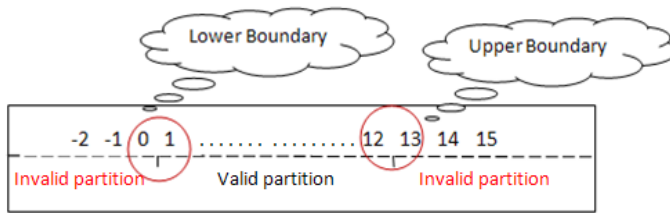


Figure 3: Boundary values for a continuous range of input

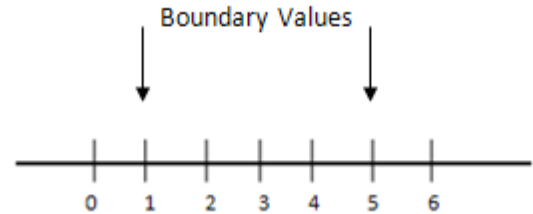


Figure 4: Boundary values for a discontinuous range of input [6]

In summary, the general guidelines for creating test cases in the BVA technique are as follows:

**Test Cases = (n - 1, n, n+1 of lower boundary) + (n - 1, n, n+1 of upper boundary)**

Where, n is the value of the boundary

### Types of Errors BVA Method Can Reveal

Students apply BVA method primarily for functional black-box testing specially for a range of values purposes. Generally, BVA can reveal different kinds of bugs or errors in the program such as [2],

1. incorrect or missing functions,
2. interface errors,
3. errors in data structures or external database access,
4. performance errors, and
5. initialization and termination errors, etc.

In our research, we have studied how we can effectively apply BVA testing methodology for testing students' programs. Not only applying for the numeric data ranges but also many other possible situations where we can apply BVA method successfully and can reveal computer programming bugs that we have discussed later in this paper.

## Define Test Cases by Applying BVA Method

### How to define test cases

We can start writing test cases as early as requirements gathering and requirements analyzing phase. We can create test cases from functional requirements, non-functional requirements, domain requirements, and from user requirements. We can also start writing test cases from customer-accepted requirements documents and later it can be used for customer acceptance testing. Borysowich [5] describes the following steps how to define test cases by applying BVA testing techniques.

1. To determine the tests for the BVA method, first identify valid and invalid input and output conditions for a given function.
2. Identify the tests for situations at each boundary. For example, we can consider the following cases:
  - One value in the  $>$  range
  - One value that is equal to the boundary, and
  - One value in the  $<$  range.
3. Boundary conditions do not need to focus only on values or ranges, but can be identified for many other boundary situations as well, such as end of page, i.e., identify tests for production of output that is one line less than the end of page, exactly to the end of page, and one line over the end of page.
4. The tester needs to identify as many situations as possible. The boundary value analysis testing should not be limited to a range of values. We can apply BVA techniques in several other cases that we have discussed later in this paper.

In this section, we have defined sample test cases for a billing specification of a phone company.

### Function to be tested

For a billing function of a phone company, the following specifications are defined [5]:

1. Requirement1: Generate a bill for accounts with a balance owed  $> 0$
2. Requirement2: Generate a statement for accounts with a balance owed  $< 0$  (credit).
3. Requirement3: Generate a bill for accounts with a balance owed  $> 0$ :
  - Requirement4: Place amounts for which the run date is  $< 30$  days from the date of service in the current total.
  - Requirement5: Place amounts for which the run date is  $=$  or  $> 30$  days, but  $<= 60$  days, from the date of service, in the 30 to 60 days total.
  - Requirement6: Place amounts for which the run date is  $> 60$  days, but  $<= 90$  days, from the date of service, in the 61 to 90 days total.

- Requirement7: Place amounts for which the run date is  $> 90$  days, from the date of service, in the 91 days and over total.
- 4. Requirement8: For accounts with a balance owed  $>$  or  $=$  \$10.00, for which the run date is  $=$  or  $>$  30 days from the date of service, calculate a \$3.00 or 1% late fee, whichever is greater.

### **Input and output conditions**

Identify the input, (i.e., information is supplied to the function) and output, (i.e., information is produced by the function) conditions for the function.

The input conditions are identified as:

1. balance owed,
2. balance owed for late fee.

The output conditions are identified as:

1. age of amounts,
2. age of amounts for late fee,
3. calculation for late fee.

### **Defining tests**

We define test cases by applying boundary value analysis testing techniques for each of the input and output conditions. For example:

**Balance owed:** We can calculate balance owed bill for three input conditions,

1. balance owed  $>$  0,
2. balance owed  $=$  0,
3. balance owed  $<$  0.

**Age of amounts:** We can calculate input conditions for the balance owed  $>$  0 and

4. run date - date of service = 0,
5. run date - date of service = 29,
6. run date - date of service = 30,
7. run date - date of service = 31,
8. run date - date of service = 59,
9. run date - date of service = 60,
10. run date - date of service = 61,
11. run date - date of service = 89,
12. run date - date of service = 90,
13. run date - date of service = 91.



**Balance owed for late fee:** We can calculate input conditions for balance owed for late fees.

the run date - date of service > 30 and

14. balance owed = \$9.99,
15. balance owed = \$10.00,
16. balance owed = \$10.01.

**Age of amount for late fee:** We can calculate output conditions for the balance owed > \$10.00 and,

17. run date - date of service = 29,
18. run date - date of service = 30,
19. run date - date of service = 31,

**Calculation for late fee:** We can calculate output conditions for the balance owed > \$10.00, run date - date of service > 30 and,

20. 1% late fee < \$3.00,
21. 1% late fee = \$3.00,
22. 1% late fee > \$3.00.

## Apply BVA Techniques in Various Scenarios

Boundary value analysis generates test cases that highlight errors better than Equivalence Partitioning (EP). BVA testing techniques is a very effective test case design techniques for students programs testing as well as it is very popular within the professional programmers in software industries. Unfortunately, most of cases, students and other professionals apply BVA method while there is a range of values. This range of values can be either continuous or discontinuous values.

We believe boundary conditions do not need to focus only on a range of values, but can be identified for many other boundary situations as well, such as end of page. We name this kind of boundaries as “invisible boundaries.” The testers or students need to identify as many situations as possible; the following list of common extreme test conditions [5] may help with finding invisible boundaries and applying BVA techniques effectively:

1. zero or negative values,
2. zero or one transaction,
3. empty files,
4. missing files (file name not resolved or access denied),
5. multiple updates of one file,
6. full, empty, or missing tables,

7. widow headings (i.e., headings printed on pages with no details or totals),
8. table entries missing,
9. subscripts out of bounds,
10. sequencing errors,
11. missing or incorrect parameters or message formats,
12. concurrent access of a file,
13. file space overflow.

Finally, we recommend to use equivalence set testing (and error guessing) augment equivalence set testing with boundary value testing. Programmers do not only consider the boundaries caused by the state of the item under test or attributes based on simple data types [8].

## **Steps in BVA Testing Techniques**

The following steps are followed, during boundary value analysis testing, typically perform the following steps (in the following decreasing order of importance) [8]:

1. Identify all objects and data types involved in an interaction.
2. Identify all that have state models.
3. Using the state models and any assertions involving the interaction, identify all relevant boundary values.
4. To identify common failures caused by individual boundaries:
  - a. Create a test case for each boundary value.
  - b. Create two test cases near each boundary value, one on each side of the boundary.
5. Create one test case near the middle of each "volume" bounded by the boundary values.
6. Create a test case simultaneously using as many of the boundary values as is practical to cause rare failures caused by the interaction of multiple boundaries.

## **Limitations of BVA**

BVA testing technique is very simple and straight-forward; however, boundary value testing is typically subject to the following limitations [4, 8]:

1. In order to minimize the number of test cases, BVA testing technique assumes that the boundary values identified from the specification and design are the only boundary values. However, this assumption is based on the understanding that the implementation of the class conforms to its specification and design. If the implementation does not match the design as specified (which it often does not if a defect exists), then additional boundaries will exist due to the defects.

2. Instantiating an object that is either on or near a boundary between regions is often nontrivial.
3. BVA works well when the Program Under Test (PUT) is a “function of several independent variables that represent bounded physical quantities” [9]. When these conditions are met BVA works well but when they are not we can find deficiencies in the results. The reason for this poor performance is that BVA cannot compensate or take into consideration the nature of a function or the dependencies between its variables [9].
4. Not all of values or inputs would be under testing process.
5. BVA does not test values which considers same group separately.
6. BVA does not test all possible inputs and does not test dependencies between combinations of inputs either.

## Conclusions

Boundary value analysis testing technique is a very effective method because testing experience shows that the systems are more likely to fail while data or object values moves from a valid to invalid or vice-versa. BVA can provide a relatively simple and formal testing technique that can be very powerful when used correctly. This method can be applied successfully in the students’ programs testing as well as professionals programs testing. Myers [6] has mentioned in his book about Boundary Value Analysis that if it is practiced correctly, it is one of the most useful test-case-design methods. He has also mentioned that it is often used unsuccessfully as the testers usually see it as so simple they misuse it, or do not use it to its full potential.

In this paper, we have discussed how we can apply BVA method effectively in different situation and provided examples. We have discussed how to determine boundary; how to define test cases; and what are the different steps for BVA testing. We have also discussed how we can apply BVA technique in several other scenarios besides a range of numeric values. We believe that BVA test case design technique is a simple but very effective method for revealing notorious bugs in the program. We can create BVA test cases as many as possible by applying different scenarios and we can reveal bugs in the early stage of the software and ultimately produce higher-quality software in lower cost.

## References

1. Copeland Lee , “Boundary Value Testing, “Practitioner’s Guide to Software Test Design, Norwood, MA: Artech House Publishers, 2004, pp. 39-46.
2. Mark Elshaw, “Software testing techniques”, School of Computing and Technology, University of Sunderland,

<http://www.his.sunderland.ac.uk/~cs0mel/comm83wk5.doc>, web retrieve on March 6, 2008.

3. Myers, Glenford J., *The art of software testing*, Publication info: New York : Wiley, c1979. ISBN: 0471043281 Physical description: xi, 177 p. : ill. ; 24 cm.
4. Wikipedia, "Boundary Value Analysis," [http://en.wikipedia.org/wiki/Boundary\\_value\\_analysis](http://en.wikipedia.org/wiki/Boundary_value_analysis), web retrieve on March 6, 2008.
5. Borysowich, Craig; "Testing via Boundary Value Analysis", Observations from a Tech Architect: Enterprise Implementation Issues & Solutions, 6/23/2007, <http://blogs.ittoolbox.com/eai/implementation/archives/testing-via-boundary-value-analysis-17126>, web retrieve on March 6, 2008.
6. Glenford J. Myers, *The Art of Software Testing*, John Wiley and Sons, Inc. 2004
7. Blake Neate, *Boundary Value Analysis* <http://www.cs.swan.ac.uk/~csmarkus/CS339/dissertations/NeateB.pdf>, web retrieve on March 6, 2008.
8. *Boundary Value Analysis Testing*, <http://www.opfro.org/Components/WorkUnits/Techniques/Testing/BoundaryValueTesting.html>, web retrieve on March 6, 2008.
9. P. Jorgenson, *Software Testing- A Craftsman's Approach*, CRC Press, New York, 1995