Interpreting Expert Knowledge For Computerized Question Generation

Cody Zilverberg
St. John's University
Box 1560
Collegeville, MN 56321
cjzilverberg@csbsju.edu

Introduction

It's the night before his organic chemistry examination, but John isn't ready for the test. Sure, he studied all week for the test, but he still doesn't have a firm grasp of all the material and he knows his professors wouldn't be happy with him if he woke them up now, at 1:30 A.M. So, he sits down at his PC and launches his browser, guiding it to the "Question Generator" homepage where he logs on and selects the subject "Organic Chemistry." He cycles through a series of questions, some of which he answers correctly. When he gets one wrong, a tutorial pops up and explains his error before giving him the correct answer. When he reaches the end of the questions, he restarts the generator from the beginning, and it provides him with a second series of randomly formulated questions. John continues this process until well into the morning, when he finally falls asleep, his hand frozen in a cup-like shape over his mouse.

In organic chemistry class the next day, John and his classmates sit down before their computers and prepare for their exam. Stretching his cramped fingers, , John logs on to the Question Generator homepage like he did the night before and selects "Organic Chemistry." This time, however, he chooses to enter "evaluation mode." Like the "practice mode" he used the previous night, all of the questions in this mode are generated randomly so that he will never get the same question twice, although all of the questions will still refer to the same subject areas. The difference between the modes is that in evaluation mode the answers selected by John and his classmates will be recorded and scored by the computer. Later, professor Johnstone will be able to examine his students' responses and identify what misconceptions his students have. John ponders the whole process, until he is interrupted by a voice coming from the front of the room, "Alright everybody. You may begin."

In fact, John could have used the application for days and never seen the same question twice, because the question generator he used randomly generates its questions from thousands of possibilities. Though the application in the story does not yet exist as such, a prototype of it is in development at St. John's University and the College of St. Benedict. When completed it will cover a broad range of subject areas across many disciplines, and it will also be flexible enough to allow professors the ability to add their own questions from any genre. The questions themselves will be dynamically generated from templates, each of which refers to graphics that are also dynamically generated. A final component of the application will provide feedback to the student and record data on student responses which an instructor will be able to use for evaluation and for identifying misconceptions among his/her students.

The process itself, though, yields several benefits. First, creating traditional paper-based exams requires a large amount of effort on the part of professors, because developing good questions that will accurately test a student's knowledge as well as broaden it is a difficult and time-consuming task. Every time an instructor generates a well-worded question about a particular concept he must ask questions like, "What idea is the question testing?", "How is the correct answer calculated?", "What are the incorrect answer choices?", and "Does the question refer to a graph?" In contrast, when using our application, those issues must be addressed only once. We attempt to capture the rules a professor uses when constructing a question, and then we use those rules to create a question template. Creating the question template will automate the process of question generation by providing thousands of unique possibilities for each question. Our goal is to create a dynamic online homework/exam system that through its versatility will cover a variety of subjects, be educational for students, and productive for educators.

Limitations of Past Electronic Homework Applications

The idea of using computers as an aid in teaching and examination is not a new one. It is has been around for years in one form or another, but recent technological advances have allowed the development of more advanced applications. Also, as each new homework program hits the market, its strengths and weaknesses can be incorporated into or thrown out of future applications. Many of these previously existing homework applications can be categorized in one of the ways listed below.

First among them are those that use a question bank. In this type of an implementation a professor designs a fixed number of questions that cover any number of subjects or concepts. Then, the questions are hard-coded into a computer. When this type of program is run, it selects a question from among those pre-programmed into its question bank and after the student answers, it moves on to the next question. If the question is well designed, it can be very useful the first time the student uses the program. However, this type of application is very limited because after a student uses the program several times, the question bank is exhausted and cannot provide any further unique questions. If such a program is used for evaluation purposes, a student could even cycle through the questions until he/she has learned all of the answers (without reading any of the questions) and get a perfect score. At this point it clearly loses its value as an evaluator and as an educational tool.

A second type of program addresses some of the above issues by inserting variables into each question. These questions require more effort than static questions, but variables allow the number of unique questions to rise rapidly. Unfortunately, this is often not enough to overcome the problem. Many times such questions are also referred to as "plug-and-chug" questions, consisting of question text but no graphics. To obtain the correct answer, you take numbers out of the question and "plug" them into the correct equation to arrive at the answer. Methods such as this work, but in many cases they are only teaching how to use an equation; they fail to teach the actual concept behind the question.

The Solution

What really sets our application apart, from an educational point of view, is our combined use of variable-based text with dynamic images. Each question generated by a template refers to one or more image specific to the template's concept area. Rather than using the algorithmic "plug-and-chug," method, students must combine information from the graphics with information from the text in order to arrive at the correct answer. By understanding the meaning behind the graphics, the students come to better realize the problem that a question represents in reality. The result of this process is that students have a better grasp of the problems' underlying concepts.

As mentioned earlier, we use question templates to expand the number of possible questions our application can ask. Each template has certain primary components: static question text, variable question text, equations, and links to appropriate graphics. Static question text forms the core of any question, and all of the variables in the question must make grammatical sense in relation to the static text. The variable text comes in several varieties. The first type changes the actual wording of a question, while retaining focus on the same concept area. The second type of variable changes which graphic or graphics the question refers to, and the final type is a number value that is increased or decreased within a specific range of values. The combination of these kinds of variables in a question template allows for many variations, but when you also consider that the graphics a question refers to are dynamic, the possible questions that can be generated from a template rises into the thousands. As a result, the potential for anybody to receive the same question twice is next to nothing, though every student's question will still have the same underlying concept. The many possible questions within a single concept area provided by a template and its graphics increase the ability of the application to educate and at the same time reduce the potential for cheating.

Implementation

As technology advances, it has an impact on the methods used to implement any complex software designs. In the past, this meant homework programs were limited to residing on a single computer or floppy drive, complicating the process of making the software available to its users by requiring multiple installations or copies. In addition, data collection was also accomplished using floppies, a cumbersome method which requires that someone collect the floppies from all students and download their data. Those problems began disappearing, however, as the network became more common and the World Wide Web gained popularity. The decision to make our application available from the WWW came early, in response to the desire to make the program available to as many as possible. The Web allows any student at a networked computer anywhere in the world to access the application at any time of the day.

Making a complex program available on the World Wide Web was impossible until very recently. HTML was inadequate for developing the sort of application that we envisioned. However, with the advent of Java the situation has changed. Java is a cross-platform object-oriented programming language. Being platform independent means that a developer need write and/or compile the program only once to have it work on Windows, Unix, etc. because it is not compiled into native machine code. Instead, when the program is run, an interpreter works

behind the scenes to make the application function as it should. When deploying a project on the web, where potential users could be on one of many different systems, platform independence is vital if you wish to reach as many as possible. Unfortunately, certain drawbacks do exist, and they come in the form of different virtual machines. Java has suffered as a result of the browser wars. Both Netscape Navigator and Internet Explorer have their own virtual machines that can perform very differently when running the same program. There are ways around this problem, but they are costly in time and effort.

Java also provides an improvement in performance over CGI scripts. CGI scripts must continually communicate with the host. This communication takes time and can cause impatient users to lose interest. Java applets, however, are loaded when a user loads the web page where they reside. Then, they are run on the local system, greatly improving performance. It is true Java has not achieved the same level of performance as platform-dependent languages such as C++, but with the appearance of JIT compilers, Java has experienced an impressive boost in performance. As technology continues to move forward, we should see further improvements in performance and browser compatibility.

Expert Knowledge

In the past, "expert systems" have been developed to address many issues, including that of medical diagnosis. To solve the problem of medical diagnosis, a scientist may formulate the question, "How does a doctor go about making a diagnosis?" Setting out to create a program that would teach and examine, we were faced with a similar kind of question, "Using a computer, how does one simulate the process an instructor goes through to create a meaningful question?" As it turns out, this is a difficult process that involves many complex issues and a large investment of time.

A typical medical expert system would be fed a vast amount of knowledge from doctors in many fields about symptoms of particular illnesses. Then, when a patient reported his/her symptoms, the program would search its collected knowledge to find a matching illness for the symptoms. Similarly, our task was to capture the rules used by a professor when he/she develops questions, and then break them down into question templates like those described earlier.

We started the process by identifying two concept areas within chemistry, concentration and calorimetry, that we would focus our attention on. Breaking these concept areas down further, we began creating individual questions. The first step to this was generating the graphics that would be referred to by the question templates. These needed to be dynamic, as mentioned earlier, but they also needed to fit within certain constraints that would allow them to be used as broadly as possible.

Our calorimetry questions required heating curves (Figure 1). On the first attempt to create the curves, we modeled them after realistic substances found in nature. The computer would randomly generate numbers within specific ranges of values for a substance's melting point, vaporization point, etc. Then, referring to those values as a base, we calculated the points on the curve using a combination of random numbers and equations. The graphs of the resulting substances were very realistic. Unfortunately, they were also very impractical for use in exam or

homework questions. It turned out that the proportions of the graphs made them difficult to read, as well as difficult to shrink to a size that could be easily viewed on a computer monitor. We decided to sacrifice some of their realistic look for something that would be more helpful in asking questions. This was easily done by changing some equations to reproportion the curves. By tweaking the curves from time to time, we finally achieved a satisfactory graphic that we used throughout the first stage of development. When we began expanding the question base, however, we encountered a new problem that will be addressed later in this paper.

After the calorimetry and concentration graphics were completed, we moved on to compose questions, beginning with approximately half a dozen in each subject area. The text of each question was broken up into two main parts: those parts that were variables, and those that were not. The wording of each question must be such that different combinations of dynamic parts with static parts will still make sense grammatically and conceptually. A simple example of this is in the following, where the possible variables are inside brackets.

Which substance has the [lowest, highest] [boiling, melting] point?

Following the choice of the actual text of a multiple-choice question, one must generate the correct answer as well as a set of incorrect choices. The correct answer is easily found by combining elements of the question with essential data gleaned from the accompanying graphics and plugging them into an equation. It may appear that generating incorrect answers would be even more simple, but that is not so. If one simply generated random numbers or text that were in no way related to what the question was dealing with, the correct answer would often be much too obvious. If we are truly interested in evaluating a student's knowledge, we do not want him/her to answer a question correctly because there is only one answer that even remotely makes sense. Instead, we would rather have a student get the correct answer only when the student understood what was being tested. And, if a student does not grasp the concept, it is useful to know where he/she ran into the problem. However, if the incorrect answers are meaningless, and a student selected one as an answer, it would tell nothing other than the student got the question wrong. For these reasons, we chose to use distracters.

"Distracters" is the term used to describe answers that are incorrect but still have meaning. Our multiple-choice questions use five possible answers, so we generate four distracters every time we ask a question. Each distracter is calculated using a variation of the correct equation. For example, if the correct answer to a question could be found by using the equation (A-B)/2, two possible distracters would be (A-B)*2 and (A+B)/2. Both involve mistakes that a student could easily make while calculating the answer. If so, he/she would likely choose that answer and get the question wrong. It may sound as if we use distracters solely to make exams harder or to trick student, but this isn't the case. Their value comes into play when a professor analyzes which distracters his/her students used. If, for example, 50% of the students chose the distracter whose equation was (A-B)*2, the professor understands that this is a widely held misconception among the students. He/she can now go about correcting the problem in future instruction.

Concerns Within Template Creation

In our current prototype, each question's template exists as a Java class with certain standard variables and functions that return essential data. When the program asks a particular question, it starts by creating an object of that question's class. The constructor chooses the variables and sets the question text. The difficult part begins when one attempts to calculate the correct answer.

As it turns out, certain questions refer to all of the graphics that are generated for that question, others refer to only some of them, and still others refer to only one of the graphics. For those that refer to some of the graphics, but not all, an algorithm must be developed that will choose certain graphics from those already generated. In most cases, any subset of the graphics will be adequate. However, in certain questions a much more complicated algorithm must be developed. In these cases, a particular relationship must exist between two or more of the beakers. For example, take the following question, which refers to a set of graphical Solutions like those in Figure 2:

> Solution #4 is half as concentrated as which Solution?

When we ask this question about Solution #4, there *must* be some Solution among the other randomly generated Solutions that is half as concentrated as #4. If not, there is no correct answer to this question. An inefficient solution to this problem would be to continue generating new Solutions until, by chance, we came up with a pair that had the proper relationship. Rather than take this route, we created rules that determined how the Solutions were selected for each question. If a particular question required that Solution A be twice as concentrated as Solution B, the generation rules guaranteed that this relationship would exist every time the Solutions were generated.

Even after the correct graphics and answers could be chosen for each question, we were confronted with more difficulties. It happened that sometimes a distracter would produce the correct answer, as in the following. Consider my previous example of (A-B)/2, where A and B are points on a graph:

A=6 B=2

Then the correct answer is (6-2)/2=2. But, let's say that distracter #1 is defined by the following:

(C-D)/2

where C and D are points on the same graph, but C and D are not the correct points to use. It could be the case that:

C=10 D=6

Then the distracter would produce the answer (10-6)/2=2. Of course, 2 was also the *correct* answer! It became apparent that not every combination of graphics and distracters could be

used. Development of distracters, then, requires careful consideration of possible missteps that could lead to correct answers.

In the same way, if Solution A need be twice as concentrated as some Solution B, there must be one and only one Solution B for which this property holds. Still another potential pitfall exists when the correct answer to a question can be found by using two different methods. We discovered this problem within one of our calorimetry questions that referred to a set of warming curves. Abandoning the realistic aspects of the curves returned to haunt us when the question had two possible correct answers, but it only recognized one of them as correct. The question itself used only one of two possible methods of obtaining the correct answer. That would not be a problem if the curves did indeed represent naturally occurring substances because both methods would produce the same correct answer. However, because our curves were out of proportion in certain aspects, the answer became ambiguous, and we were forced to shelf the question until we could rework the curves.

Using a Database for Analysis

We are currently working on integrating a database with our prototype, the benefits of which will be many. First, the database will provide instructors with a very powerful analysis tool. Under our current implementation, there is no way to record a student's response, but once connected to a database, the application will become much more versatile. We chose to use the freeware database PostgreSQL for a number of reasons. Most importantly, it supports Java and the JDBC. The JDBC is a standard data access interface that provides the ability to make a connection from a Java application or applet to a relational database such as PostgreSQL. The database also runs on our Unix server and fit our needs for power and as mentioned above, the database is freely available for download off of the net. A large set of documentation is available at the PostgreSQL website detailing how to install the database and set it up for use with Java. All of these made it an affordable and more than adequate option.

Each student that uses the system will have a record of their performance stored in the database. Each student will also be listed in the database as a member of a course they are currently taking. When an instructor wishes to assign questions from the application, he/she will log on and tell the application which questions should be given to the members of a particular course and how long those members have to complete the assignment. When a student starts the application, he/she will begin by logging on to the system. The application will search its database to find the assignments currently registered for that student and give the student the option of beginning those assignments. After the student makes his/her choice, he/she will begin the examination. For every question the student answers, the correct answer, the rules that were used to generate the distracters, and the student's chosen answer are recorded in the database.

Any time after the assignment has been given, the instructor can check on the students' progress by looking at the data in the database. This data, of course, can be organized in many different ways. For example, graphs can be easily created that detail the frequency certain distracters are chosen. This informs the instructor which misconceptions exist and also allows him/her to fine-tune questions by eliminating distracters that are never chosen and replacing them with others that students may be more likely to choose. Through use of a database, all of this is done

automatically, giving it an advantage over programs where each student's score is recorded on his/her own diskette, because such a system requires that somebody compile all of the data before it can be analyzed.

Putting Questions into the Database

Creating dynamic questions by coding them is one thing, but putting that same information into a database is quite another. In the first process, for instance, every time a new question is added, an expert must communicate to a programmer his/her specifications for the new question. Then, the programmer creates a new class for that question and the whole program is recompiled. If the programmer made a mistake, or there was poor communication between the programmer and the expert, the question's code must be modified, the program is recompiled, and the question is again evaluated to see if it is correct. Obviously, this is not the most desirable situation because it is not time efficient and it requires that a programmer and compiler be available.

On the other hand, the overhead for the second process is high, but when completed, expanding the depth and breadth of the questions becomes much more efficient. Database design for such a system involves complicated relationships and requires careful planning. One purpose of using a database to hold question data is to make the program as flexible as possible in regard to content. A question concerning any subject area, not just chemistry, should be able to be plugged into the database and still make sense. For this reason, the database must be designed with some very general rules that can handle different ways of formulating questions, answers, and distracters. For example, many questions that we implemented within calorimetry and concentration used text for answers rather than calculated numbers. In these cases the database must still be able to calculate which are the correct answers using equations, but the results of those equations will be words.

A particular question, when put into the database, is described by two attributes: its subject area and its type (multiple choice, true/false, etc.). From there, a question is broken down into dynamic and static parts, much like what was done when putting a question into code. The process of breaking questions down led to the creation of several tables, the first of which contained static question text. The design of this table was rather simple; each string in the table referred to a particular position within the structure of a particular question. Position 1 was held by the string that represented everything before the first variable in the question, position 2 held the string containing everything before the second variable, and so on. Designing tables for the variables was significantly more complicated, however.

To begin with, it involved two tables rather than just one. The first table describes certain variable groups, which are at a position (like those positions held by question strings) within a particular question. Each variable group can then be represented in the question as either a number within a range of values, or as one of several choices that reside in a second table. The second table contains all of the variable possibilities for that position in the question, as well as the frequency with which they should appear. When the Java applet is run, it selects a variable based on a random number that the applet generates. When a variable has been selected for every position in the question, the applet then puts the pieces together to form the question text.

Of course, every variable that is chosen must also be recorded so that the program can use it later to find the correct answer and distracters.

Determining how to put algorithms that choose which distracters to use, algorithms that calculate what those distracters would be, and algorithms that calculate correct answers was the most difficult part of the entire process. We made the decision that this would be best accomplished by using a set of rules. The rules would exist as text within tables of the database. When a question was constructed, the program would load from the database the rules that apply to the question and then interpret their meaning. The rules take the form of simple logic equivalence statements, such as "If Variable2 Then Rule B Else "Solution #1", which would check the truth of Variable2 and then decide whether to load Rule B or return the string "Solution #1." If it loaded B, the program would then proceed to check the Rule, which may be an equation, a string, or a formulation similar to that of the example. Designing the set of usable rules, of course, is another process requiring careful planning. It should be made simple enough for a non-programmer to use, but powerful enough and flexible enough to handle complex and diverse questions. Non-programmers will be aided in the future after we have created a GUI (Graphical User Interface) for question development, which is one of our long-range goals.

Conclusion

Technology is working its way further into our education system every day, and with new advances in technology come new opportunities for learning. The application that we have developed thus far is an example of combining the latest ideas in teaching with the newest technologies to keep the learning process a fresh and exciting experience. Our combined use of graphics and text brings students to a new level of understanding not achievable by using algorithmic questions. Being dynamic, our application also has tremendous potential as both educator and as evaluator. Computer generated, evaluated, and scored questions take a huge burden off of educators, allowing them to focus their energy where it is most needed.

The application we aspire to build will bring the learning experience to a new level, but it also has significance from a computing perspective. We are attempting to model the human problem-solving process, a very difficult task. If we are successful in designing a system that can accurately simulate the question generation methodology through use of a database, we will be able to apply our experience to other problems in computer science.

Figure 1

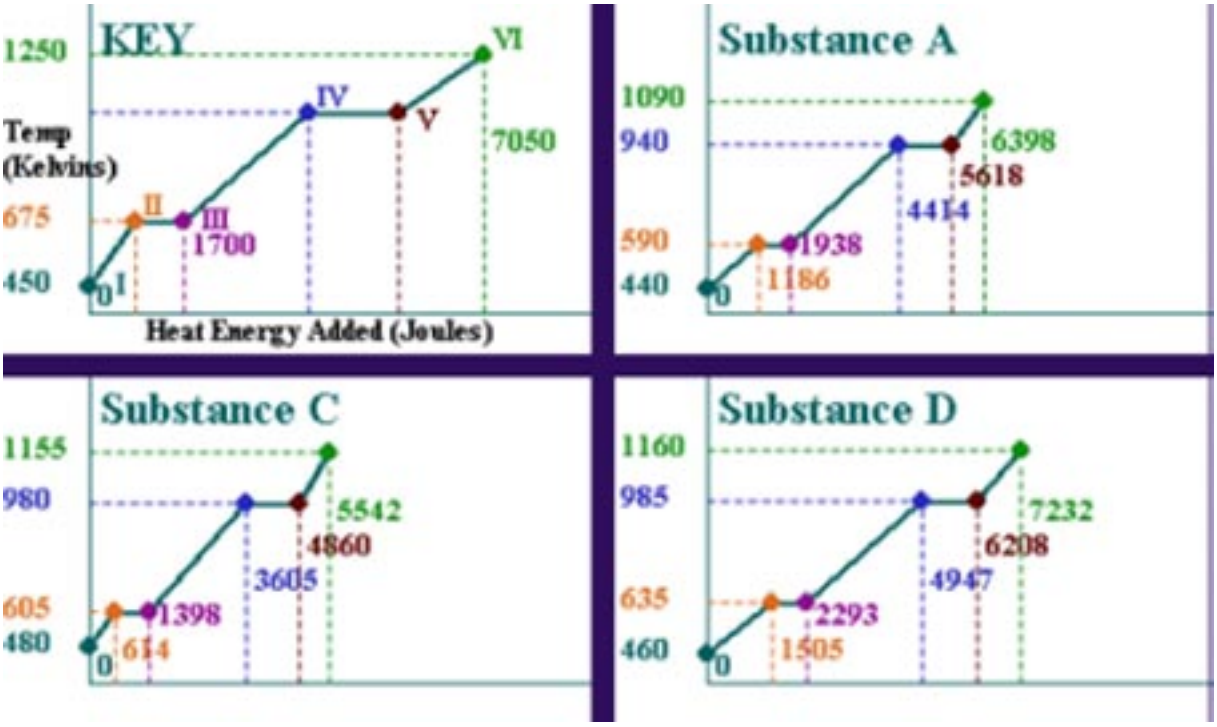
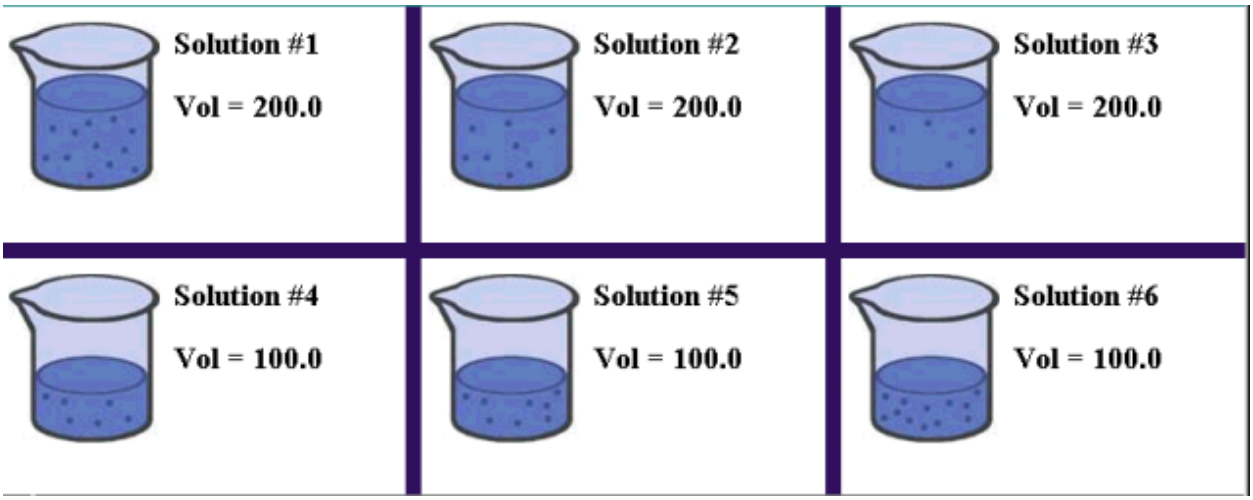

KEY
1250
Temp
(Kelvins)
675
450
0
I
II
III
1700
IV
V
VI
7050
Heat Energy Added (Joules)
Substance A
1090
940
590
440
0
1186
1938
4414
5618
6398
Substance C
1155
980
605
480
0
614
1398
3605
4860
5542
Substance D
1160
985
635
460
0
1505
2293
4947
6208
7232

Figure 2


Solution #1
Vol = 200.0
Solution #2
Vol = 200.0
Solution #3
Vol = 200.0
Solution #4
Vol = 100.0
Solution #5
Vol = 100.0
Solution #6
Vol = 100.0