Teaching the Nearest Neighbor Search Problem in an Undergraduate Algorithm Analysis Course

Sun B. Chung
Department of Quantitative Methods and
Computer Science
University of St. Thomas
St. Paul, MN 55124
sbchung@stthomas.edu

Abstract

The nearest neighbor search problem is an important problem in computer science that has a wide range of applications including medical image processing, pattern recognition, mobile computing, and retrieval of multimedia objects such as images, text, and videos over the Internet.

The problem deserves more attention than it receives in undergraduate computer science education for the following reasons. First, it has a wide range of applications in areas that are relevant to our daily lives. Second, it has interdisciplinary applications such as medical image processing, so it will provide a good opportunity for students to appreciate the contributions that theoretical computer science can make in practical areas. Third, the exact and approximate algorithms for the problem and its variants will provide students with interesting materials for comparison and analysis.

In this paper, I propose a way to incorporate the nearest neighbor search problem in an "analysis of algorithms" course.

1 Introduction

In computer science, there is an important problem called *nearest neighbor search*. The problem can be described informally with the following example: Place a pin on the map of the U.S. What is the nearest city to the pin? The problem deals with a 2-dimensional space, and it is easy to find a solution for such a low dimension. However, in high dimensional spaces, typically greater than 25 dimensions, the problem becomes very difficult.

The nearest neighbor search problem has a wide range of applications including medical image processing, pattern recognition, mobile computing, and retrieval of multimedia objects such as images, text, and videos [5, 18, 20, 23, 24].

The problem was first posed in the 1960s [17]. Since then, researchers have been working to come up with better solutions for *exact* search in high dimensions, but there has been little success [13]. That is why the nearest neighbor search problem is said to have the "curse of dimensionality" [6, 10, 13, 26]. As is typical in computer science when finding an exact solution is difficult, researchers turned to *approximate* search [1, 2, 3, 8, 10, 13, 15]. In the meantime, different data structures have been proposed for managing data points efficiently [4, 7, 19, 21, 25].

Recently, I helped a Ph.D. candidate at the University of Michigan improve his computer program analyzing 64-dimensional data points of medical images. In one of the subroutines of the program, I introduced an algorithm for the "k-nearest neighbors" problem, a variant of the nearest neighbor search problem. It enabled the program to run drastically more efficiently for up to 16 dimensions [18]. It was a precious opportunity for me to learn about the fascinating properties of the problem and to think about teaching it in an undergraduate computer science course.

In this paper, I propose a way to incorporate the nearest neighbor search problem in an "analysis of algorithms" course. The problem deserves more attention than it receives in undergraduate computer science education for the following reasons. First, it has a wide range of applications in areas that are relevant to our daily lives, such as mobile computing and retrieval of multimedia objects over the Internet. Second, it has interdisciplinary applications such as medical image processing, so it will provide a good opportunity for students to appreciate the contributions that theoretical computer science can make in practical areas. Third, the problem has a number of variants, such as the above-mentioned k-nearest neighbors problem. The exact and approximate algorithms for the problem and its variants will provide interesting material for comparison and analysis.

2 Theoretical Analysis and Hands-on Activities

This section describes a number of activities students can do after they are briefly introduced to the nearest neighbor search problem. Some of the activities are suitable for engaging students in active learning in a process of inquiry. Some are also suitable for group activities.

2.1 Areas of Application

Look for information about the following (possibly on the Internet).

(a) application areas for the nearest neighbor search problem

(b) variants of the problem found in application areas such as the following: *k*-nearest neighbors problem which is to find *k* nearest neighbors of a given point for a small integer *k* [11].

(c) the number of dimensions involved in each application area, such as 64 dimensions for medical image processing

2.2 Understanding the Curse of Dimensionality

The following activities provide opportunities to understand why the problem is easy in low dimensions and difficult in high dimensions.

(a) Draw 16 points on a line that is 10 cm long, such that the distances between the points are roughly equal. Below the line, draw a square, 10 cm wide and 10 cm high, that represents a 2-dimensional space. In the square, distribute 16 points such that the distances between the points are roughly equal. Compare the distances between the points on the line and the distances between the points in the square.

(b) To do the above activity in a systematic manner, write a simple program to find out the average distance to the nearest neighbor in *d* dimensions, where $d = 1, 2, 3, \ldots, k$ for a reasonably large *k*.

  (i) The number line representing [0.0 .. 1.0) can be used for the line. Similarly, the 2-dimensional space can be represented by two such lines for the width and height of a square. In general, in a *k*-dimensional space, the co-ordinates of a point can be represented by a *k*-tuple of random numbers in [0.0 .. 1.0).

  (ii) The distance between two points $X = (x_1, x_2, \ldots, x_k)$ and $Y = (y_1, y_2, \ldots, y_k)$ is determined by the square root of $(x_1 - y_1)^2 + (x_2 - y_2)^2 + \ldots + (x_k - y_k)^2$.

(iii) To find the distance to the nearest neighbor for each and every point, use a brute force algorithm (exhaustive search). That is, compute the distance of each possible pair of points.

(c) Observe the performance of the program in the following respects.

(i) For fixed *d*, try different numbers for the number of points *n* ranging from a few thousand to a million. For different numbers, compare the execution time of the program. Is the growth rate linear, quadratic, or exponential in *n*?

(ii) For fixed *n* and different *d*, compare the execution time of the program. Is the growth rate linear, quadratic, or exponential in *d*?

(iii) In addition to analyzing the execution time, analyze the growth rate of space requirement.

(d) Write up a report on what is observed as the programs are executed.

(e) Nene and Nayar [19] provides a good probabilistic analysis of the expected number of neighbors within a given distance in a *d*-dimensional space. Students who have enough statistics background may be referred to it.

2.3 Comparison of Different Algorithms

2.3.1 Simple Algorithms

Compare some of the simple algorithms that exist.

(a) Write a program for the following.

(i) Implement a brute force algorithm that finds the nearest neighbor for a given point. Alternatively, find all the neighbors of a given point within distance ε. A third alternative is to find all the neighbors of all points within distance ε.

(ii) Implement the algorithm by Friedman *et al.* [12] and find out how much more efficient their algorithm is than the brute-force algorithm.

(iii) Implement the algorithm by Nene and Nayar [19] which claims that their algorithm is more efficient than that of Friedman *et al.*

(b) Compare the performance of the algorithms as follows.

(i) For fixed *d*, try different numbers for the number of points *n* ranging from a few thousand to a million. For different numbers, compare the execution time of the algorithms.

(ii) For fixed *n* and different *d*, compare the execution time of the algorithms. Is the growth rate linear, quadratic, or exponential in *d*? Are the other two algorithms more efficient than the brute force algorithm? Up to how many dimensions is the efficiency of the algorithms noticeable?

(iii) In addition to analyzing the execution time, analyze the growth rate of space requirement of each algorithm.

(c) Write up a report comparing the performance of the algorithms.

2.3.2 Reproduce Experiments

McNames [16] compares the performance of seventeen different algorithms on three types of common benchmark data sets. Reproduce the experiments and verify the performance of the algorithms.

2.4 Visualization

2.4.1 Visualization of the Performance of the Simple Algorithms

To illustrate the performance of the algorithms described in section 2.3.1, write a program that uses a Graphical User Interface (GUI) object. Compare the performance of the three different algorithms with different number of points in different dimensions.

2.4.2 Visualization of the Density of Points

Write a program that illustrates the different densities of points in different dimensions.

(a) It is easy to write a program that uses a GUI object to illustrate the different densities of points in one and two-dimensional spaces. Students who know how to use a 3D object can do the same for the 3-dimensional space.

(b) The program can be written such that it gives an animated illustration as follows. Begin with the 3-dimensional space. First, show points that have been uniformly distributed. Next, collapse the 3-dimensional space onto a 2-dimensional space slowly (by removing the z-coordinate). Finally, collapse the 2-dimensional space onto a line (by removing the y-coordinate).

(c) Even when students don't know 3D graphics, it is possible to simulate in the 1-dimensional space the density of points that becomes sparser as the number of dimensions increases.

(i) On a line that indicates the maximum possible distance between two points, illustrate the expected distance between two points (among *n* points that are uniformly distributed).

(ii) For the 1-dimensional space, the maximum possible distance between two points is 1. For the 2-dimensional space, the maximum possible distance between two points is square root of 2, and so on. In general, the maximum possible distance in a k-dimensional space is square root of k.

(iii) Compare the growth rate of the maximum distance and that of the expected distance between the two points.

(iv) Instead of the expected distance between two points, use the results of the program of section 2.2 and illustrate the average distance to the nearest neighbor.

2.5 Writing Expository Papers

Here are suggested topics for writing short expository papers.

(a) Chen *et al*. claim that their algorithm that constructs a lower bound tree at the preprocessing stage runs about one thousand times faster than the exhaustive search [9]. What is a lower bound tree? What makes it run so fast? Is the algorithm powerful enough to defeat the curse of dimensionality?

(b) It is said that, in certain applications, the dimension may be "in the order of a few hundreds, or thousands" [22]. What kinds of applications require so high a dimension?

(c) When an exact solution that is efficient is difficult to find, it is quite common to try to find an approximate solution that is efficient. Write about an algorithm that solves approximate nearest neighbor search. Is the algorithm efficient? For what kinds of applications is the approximate solution useful?

(d) Are there trade-offs between space requirement and time complexity? That is, is it possible for an algorithm to run faster if it uses more space?

(e) Write about different data structures used by different algorithms, such as the *k*-d tree, R-tree, and vp-tree.

(f) Kleinberg presents two algorithms for nearest neighbor search in high dimensions [14]. Describe and compare the algorithms.

(g) Write about two algorithms one of which improves the performance of the other.

References

[1] Sunil Arya, David M. Mount. Approximate nearest neighbor queries in fixed dimensions. *Proceedings of the fourth annual ACM-SIAM symposium on discrete algorithms*, pp. 271 – 280, 1993.

[2] Sunil Arya, David M. Mount, Nathan S. Netanyahu, Ruth Silverman, Angela Y. Wu. An optimal algorithm for approximate nearest neighbor searching fixed dimensions. *Journal of the ACM*, Volume 45, Issue 6, pp. 891 – 923, 1998.

[3] Sunil Arya, Ho-Yam Addy Fu. Expected-case complexity of approximate nearest neighbor searching. *Proceedings of the eleventh annual ACM-SIAM symposium on discrete algorithms*, pp. 379 – 388, 2000.

[4] Kristin P. Bennett, Usama Fayyad, Dan Geiger. Density-based indexing for approximate nearest-neighbor queries. *Proceedings of the fifth ACM SIGKDD international conference on knowledge discovery and data mining*, pp. 233 – 243, 1999.

[5] Stefan Berchtold, Christian Böhm, Bernhard Braunmüller, Daniel A. Keim, Hans-Peter Kriegel**.** Fast parallel similarity search in multimedia databases. *Proceedings of the 1997 ACM SIGMOD international conference on management of* data, pp. 1 – 12, 1997.

[6] Allan Borodin, Rafail Ostrovsky, Yuval Rabani. Lower bounds for high dimensional nearest neighbor search and related problems. *Proceedings of the thirty-first annual ACM symposium on theory of computing*, pp. 312 – 321, 1999.

[7] Christian Böhm, Stefan Berchtold, Daniel A. Keim. Searching in high-dimensional spaces: index structures for improving the performance of multimedia databases. *ACM computing surveys*, volume 33, issue 3, pp. 322 – 373, 2001.

[8] Timothy M. Chan. Approximate nearest neighbor queries revisited. *Proceedings of the thirteenth annual symposium on computational geometry*, pp. 352 – 358, 1997.

[9] Yong-Sheng Chen, Yi-Ping Hung, Chiou-Shann Fuh. Fast algorithm for nearest neighbor search based on a lower bound tree. *Proceedings of the eighth international conference on computer vision*, 2001.

[10] K. Clarkson. An algorithm for approximate closest-point queries. *SIAM journal on computing*, volume 17, pp. 830 – 847, 1988.

[11] Bin Cui, Beng Chin Ooi, Jianwen Su, Kian-Lee Tan. Contorting high dimensional data for efficient main memory KNN processing. *Proceedings of the 2003 ACM SIGMOD international conference on management of data and symposium on principles of database systems*, pp. 479 – 490, 2003.

[12] Jerome H. Friedman, Forest Baskett, Leonard. J. Shustek. An algorithm for finding nearest neighbors. *IEEE Transactions on computers,* pp. 1000 – 1006, 1975.

[13] Piotr Indyk, Rajeev Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. *Proceedings of the thirtieth annual ACM symposium on theory of computing*, pp. 604 – 713, 1998.

[14] Jon M. Kleinberg. Two algorithms for nearest-neighbor search in high dimensions. *Proceedings of the twenty-ninth annual ACM symposium on theory of computing*, pp. 599 – 608, 1997.

[15] Eyal Kushilevitz, Rafail Ostrovsky, Yuval Rabani. Efficient search for approximate nearest neighbor in high dimensional spaces. *Proceedings of the thirtieth annual ACM symposium on theory of computing*, pp. 614 – 623, 1998.

[16] James McNames. A fast nearest neighbor algorithm based on a principal axis search tree. *IEEE Transactions on pattern analysis and machine intelligence*, pp. 964 – 976, 2001.

[17] M. Minsky, S. Papert. *Perceptrons*, MIT Press, Cambridge, MA, 1969.

[18] Huzefa Neemuchwala, Alfred O. Hero, Paul L. Carson. Image registration using entropic graph-matching criteria. *Thirty-sixth Asilomar conference on signals, systems, and computers*, 2002.

[19] Sameer A. Nene, Shree K. Nayar. A simple algorithm for nearest neighbor search in high dimensions. *IEEE Transactions on pattern analysis and machine intelligence*, volume 17, pp. 989 – 1003, 1997.

[20] Cyrus Shahabi, Mohammad R. Kolahdouzan, Mehdi Sharifzadeh. A road network embedding technique for k-nearest neighbor search in moving object databases. *Proceedings of the tenth ACM international symposium on advances in geographic information systems*, pp. 94 – 100, 2002.

[21] Douglas A. Talbert, Doug Fisher. An empirical analysis of techniques for constructing and searching k-dimensional trees. *Proceedings of the sixth ACM SIGKDD international conference on knowledge discovery and data mining*, pp. 26 – 33, 2000.

[22] Panayiotis Tsaparas. Nearest neighbor search in multidimensional spaces. Depth oral report. Department of Computer Science, University of Toronto, 1999.

[23] Ertem Tuncel, Hakan Ferhatosmanoglu, Kenneth Rose. VQ-index: an index structure for similarity searching in multimedia databases. *Proceedings of the tenth ACM international conference on multimedia*, pp. 543 – 552, 2002.

[24] Arjen P. de Vries, Nikos Mamoulis, Niels Nes, Martin Kersten. Efficient k-NN search on vertically decomposed data. *Proceedings of the 2002 ACM SIGMOD international conference on management of data*, pp. 322 – 333, 2002.

[25] Young C. Wee, Seth Chaiken, Dan E. Willard. Computing geographic nearest neighbors using monotone matrix searching. *Proceedings of the 1990 ACM annual conference on cooperation*, pp. 49 – 55, 1990.

[26] Peter N. Yianilos. Locally lifting the curse of dimensionality for nearest neighbor search. *Proceedings of the eleventh annual ACM-SIAM symposium on discrete algorithms*, pp. 361 – 370, 2000.