Recent PTAS Algorithms on the Euclidean TSP

by

Leonardo Zambito

Submitted as a project for CSE 4080,
Fall 2006

1 Introduction

The Traveling Salesman Problem, or TSP, is an on going study in computer science. The TSP has a long history ranging back to the 1920's [1]. The TSP became popular after it was publicized by a mathematician named Merrill Flood at the RAND corporations in 1948 [9]. The TSP problem is defined as, given a complete graph G with Vertices V and edges E, where each edge $e_{ij} \in E$ has an associated cost c_{ij} incurred when traversing from vertex $i \in V$ to $j \in V$, find the optimal, or cheapest, Hamiltonian cycle of G. The vertices can be considered buildings, landmarks or other geographical locations. Thus, a Hamiltonian cycle of G is also considered a Tour.

There are different variations of the TSP. In the general case, there are no restrictions on the edge costs. Therefore, each edge may have two associated costs, $c_{ij} \in \Re$ and $c_{ji} \in \Re$, that may not necessarily be equal. Thus, G can be a directed graph and the edge costs can be negative. The general TSP is NP-complete. In this report, we will be considering the Euclidean TSP. In the Euclidean TSP, all of the vertices are points in the plane. The plane can be 2-dimensional, as in the xy-plane, or d-dimensional in general. The edge costs are the Euclidean distances between the points. Since we are working in the plane, there are restrictions on the edge costs, and some assumptions can be made, which simplify the problem. First of all, edge costs are non-negative real numbers, and $c_{ij} = c_{ji}$ for all $i, j \in V$. Moreover, the triangle inequality holds, which means that for any three vertices A, B and C, if you wish to go from vertex A to vertex C, it is always cheaper to go from A directly to C rather than passing through B. Despite these added restrictions, the Euclidean TSP is still NP-hard.

For the Euclidean TSP, it is known that there exists a superpolynomial time exact solution to the problem [12]. However, the running time of such an algorithm is not reasonable, even for input graphs containing less than a thousand vertices or points. To give you an idea of how unpractical an exact superpolynomial time solution may be, consider an exact solution of input instance being 15,112 German cities. The benchmarking of this TSP instance was conducted by TSBLIB [10] in 2001. The algorithm to solve this instance was run on a network of 110 workstations. The total execution time spent was equivalent to almost 30 years on a single 500MHz processor [12].

2 The Approximation Approach to NP-hard Problem

Since we have not been able to find an efficient exact solution to the Euclidean TSP, and since one probably does not exist, alternate approaches have been considered. A popular approach has been to find approximation solution algorithms to solve the problem. Approximating optimal solutions has become a popular technique in tackling NP-hard problems. The argument is, if we cannot find feasible exact solutions, then at least we can find efficient approximation solutions that can be used in practice.

To clarify, an approximation algorithm is one in which, for some fixed constant a, solves a problem by returning a solution that costs no more than a times the cost of an optimal solution. This solution is found in polynomial time. An NP-hard problem is considered approximable if there exists a polynomial time approximation NP-hard algorithm to that problem. The Euclidean TSP is approximable.

To expand on this approximation approach, let us consider a more useful class of approximation solutions knows as approximation schemes. In an approximation scheme,

the constant factor value c is not fixed. The inputs to an approximation scheme consist of a data set of size n and a constant value $c > 0$. A $(1 + c)$-approximation algorithm is an algorithm that returns a result whose cost is at most $(1 + c)$ times the cost of an optimal solution. We consider two approximation schemes. The first is a Polynomial Time Approximation Scheme, or PTAS, and the second is a Fully Polynomial Approximation Scheme, or FPTAS, where FPTAS $\subset$ PTAS. The only distinction between these two approximation schemes lies in the running times. The running time of a PTAS must be polynomial in the input size, but may or may not depend on c. Whereas, the running time of a FPTAS must be polynomial in both the input size and in $1/c$.

It has been proven that there does not exist a PTAS for the general case of the TSP [2]. Moreover, it was show my Garey and Johnston [6] that many NP-complete problems do not have FPTAS. However, there does exist polynomial time approximation schemes for the Euclidean TSP.

3 PTAS Algorithms on the Euclidean TSP

The polynomial time approximation schemes for the Euclidean TSP are useful in practice. Of course, there is a direct relation between the running time of such approximation schemes and the accuracy of the solution. Since one is able to specify an upper-bound on the optimality ratio of the approximate solution to the optimal solution, one can tailor the algorithm to produce a solution that is suitable to his/her needs. For instance, in some applications the accuracy of the solution is much more important than the running time to find that solution, and vice versa.

In this section, we will look at the available PTAS for the Euclidean TSP. The first solution we will look at is Arora's PTAS. Arora was the first person to find a PTAS

for the Euclidean TSP. The second solution we will look at is Mitchell's PTAS. Mitchell independently found a PTAS for the Euclidean TSP only months after Arora. Finally, we will look at a PTAS for the weighted planar graph version of the TSP, which is a special case of the Euclidean TSP. This solution was found by Grigni, Karger, Klein and Woloszyn [7]. All of the solutions listed in this report were discovered during the late 1990's.

3.1 Arora's PTAS for the Euclidean TSP

Sanjeev Arora discovered a PTAS for the Euclidean TSP in the year 1996 [3]. The approximation ratio of Arora's algorithm in $\Re^2$ is $(1 + 1/c)$, for any given $c > 1$ [3]. The main idea of Arora's solution is to recursively partition the plane, and then use dynamic programming to find a tour that crosses each line of the partition at most $O(c)$ times. When Arora first presented this PTAS, it had a proven running time of $n^{O(1/c)}$ time [5]. About a year later, he was able to improve the running time of his algorithm to $O(n(\log n)^{O(c)})$ in $\Re^2$. Moreover, in general, the algorithm can be applied to the Euclidean TSP in $\Re^d$, where a $(1 + 1/c)$-approximate solution will be found in $O(n(\log n)^{(O(\sqrt{d}c))^{d-1}})$, which is nearly linear time for any fixed c, d [4].

As mentioned, Arora'a PTAS performs a recursive geometric partitioning of the plane such that a $(1 + 1/c)$-approximation tour crosses each line of a partition no more than $O(c)$ times [3]. The portion of the plane that is considered in the algorithm is only the bounding box, which is the smallest rectangular portion of the plane that encompasses all of the nodes in the graph. The partitioning is determined by using some randomized variant of the *quadtree*. A *quadtree* is a tree data structure that is often used to

recursively partition a plane [13]. Each node in the *quadtree* can have up to four children, where the node represents a partition and its children represent the quadrants or regions of that partition [13]. In Arora's application, the partitions are always squares, where each partition is recursively divided into four equal sections [3]. Arora refers to this partitioning of the bounding box as *dissection*. Arora [3] also mentions that the idea of partitioning a plane, in order to find a more efficient solution to NP-hard problems in $\Re^2$, has already been done. It had previously been used to find an exact solution to the 2-demensional Euclidean TSP in $2^{O(\sqrt{n})}$ time [3]. Arora took advantage of this useful approach in order to develop his PTAS for the Euclidean TSP.

For each line in any given partition, the algorithm tries to guess where the tour will cross it, and in what order those crossings occur (more detail on these crossings is provided in the explanation of Arora's Structure Theorem below). It then recurses on both sides of the line. Arora [3] notes that there are O(n log n) different regions in a given partition containing n nodes. Furthermore, the accuracy of the guessing depends on *c*, so that it spends, at most, $(\log n)^{O(c)}$ on each region [3]. Thus, the total time is $n \cdot (\log n)^{O(c)}$ [3]. Experimentations have shown that the best known achievement, in terms of tour optimality thus far, is when $c \approx 10$ [3]. However Arora believes this can be improved.

Arora's PTAS algorithm is based on a theorem, which he calls the 'Structure Theorem'. We will state the theorem here, and then provide a brief explanation of it.

Structure Theorem: "Let the minimum nonzero internode distance in a TSP instance be 8 and let L be the size of its bounding box. Let shifts $0 \leq a, b \leq L$ be picked randomly.

Then with probability at least 1/2, there is a salesman path of cost at most (1 + 1/*c*)-OPT that is (m, r)-light with respect to the dissection with shift (a, b), where m = O(c log L) and r = O(c)." [3]

This theorem basically states that the crossings over the partition lines happen over a small amount of pre-specified points. These points are called *portals*. This is made possible by allowing the virtual *salesman* to stray from a straight-line path between two nodes, forcing the *salesman* to pass through a *portal* in order to exit a square. Each square has four *portals*, one at each corner, and an additional *m* equally spaced portals on each side of the square [3]. Arora [3] then defines a *salesman path*, which is any path that visits each node in the 2-demensional plane once, but can pass through any *portal* more than once.

Let us now define an (*a*, *b*)-shift. The values of *a* and *b* are integer values between 0 and L, where L is the size of the bounding box. To perform an (*a*, *b*)-shift means to modify all of the *x* and *y* coordinates of the partition lines in the following manner: The new *x* values are determined by evaluating the equation, (*a* + *x*) mod L, for each old *x* value [3]. The new *y* values are determined by evaluating the equation, (*b* + *y*) mod L, for each old *y* value [3]. Wraparound of the square occurs where applicable. That is, when squares fall outside of the bounding box due to shifting, they wraparound to the opposite side of the bounding box.

Finally, the last thing that you will need to know in order to understand the Structure Theorem is the definition of an (*m*, *r*)-light *salesman path*. The *salesman path* is considered (*m*, *r*)-light, taking into account the (*a*, *b*)-shifting of the *dissection*, if the

path crosses only crosses through *portals*, and does not cross a side of each square more than *r* times [3].

We can now list, and briefly describe, the three steps of Arora's PTAS.

Step 1: Make the Input Instance Well Rounded.

In this step, the input instance needs to be modified to be made *well-rounded* [3], which means that the instance must be prepared to reflect the Structure Theorem. All of the coordinates must be of integral value, and the resulting values must meet the following criteria: Nonzero distance values need to be greater than or equal to eight units. Lastly, the maximum distance value must be O(n). This *well rounding* modification of the input instance is done in linear time [3].

Step 2: Construct a Shifted Quadtree

The dissection is performed. Random values for *a* and *b* are chosen, and the bounding box is partitioned until each square has a size ≤ 1, which means that each square will have at most one node. The shifted quadtree is created during this dissection. Since L = O(n), the depth of the quadtree is O(log n), and it can easily be built in $O(n \log^2 n)$ time [3].

Step 3: Solve by Dynamic Programming.

In this final step, dynamic programming is used to the find the optimal (*m*, *r*)-light *salesman path* [3] using the quadtree that was constructed in step 2.

For more details on the solution of Arora's PTAS for the Euclidean TSP, the reader is encouraged to refer to S. Arora [3].

3.2 Mitchell's PTAS for the Euclidean TSP

Joseph S. B. Mitchell independently discovered a PTAS for the Euclidean TSP softly after Arora. Mitchell's algorithm is very much similar to Arora's PTAS in that it first partitions the plane, and then uses dynamic programming to reach a solution.
To partition the plane, Mitchell's algorithm exploits the concept of an "*m*-guillotine subdivision" [8].

Mitchell defines an *m*-guillotine subdivision as being vary similar to a polygonal subdivision. Partitions or subdivisions are made via "cuts", where a single "cut" is basically a straight line that divides a partition into two new partitions. A given partition is said to be *m*-guillotine if there exists a "cut" that intersect with existing subdivision "cuts", creating only O(*m*) connected components, and the two new partitions are also *m*-guillotine [8].

Mitchell then submits a theorem that is the basis for his PTAS. The theorem states that any polygonal subdivision can be converted into an *m*-guillotine subdivision [8]. This can be accomplished by adding *c*/*m* times more edges in the new subdivision then in the original subdivision [8]. Using this theorem, and applying dynamic programming to obtain optimal results from a collection of *m*-guillotine subdivisions,

Mitchell is able to produce a $(1 + 2\sqrt{2}/m)$-approximation algorithm for the Euclidean TSP that runs in $O(n^{20m+5})$ time [8]. The value of *m* is any non-zero positive integer specified as part of the input to the algorithm [8].

Unfortunately, the details of Mitchell's PTAS are much more complex than that of Arora's PTAS. For this reason, the specifics of Mitchell's method are not included within this report. For an in-depth explanation of Mitchell's PTAS, please refer to J. S. B Mitchell [8].

3.3 Grigni, Karger, Klein and Woloszyn's PTAS for the Weighted Planar TSP

Grigni, Karger, Klein and Woloszyn [5, 6] were able to produce a PTAS for a weighted planar graph containing edge weights or costs. This PTAS is included in this report because a weighted planar graph is a special case of the Euclidean TSP. A planar graph can be drawn on the plane, just as the Euclidean TSP. The difference in a planar graph is that edges in the graph are not allowed to cross each other.

Grigni, Karger, Klein and Woloszyn [5] present a PTAS for the weighted-planar-graph TSP that, given a graph G and some constant value $c > 0$, produces a $(1 + c)$-approximation salesman tour [5]. The algorithm that accomplishes this approximation runs in $n^{O(1/c^2)}$ [5]. The algorithm proceeds by first simplifying the input graph G into a spanner subgraph G′. The graph G′ has the same set of vertices as in G, except that the distance between any two vertices in G′ can be up to $(1 + c)$ times the distance between the corresponding two vertices in G [7]. Moreover, the sum of the costs of all the edges in G′ must be $O(1/c)$ times the cost of a minimum spanning tree of G [7].

The spanner subgraph G′ is then hierarchically partitioned. The hierarchical orderings of the partitions are maintained within a tree data structure, such as a quadtree. Grigni, Karger, Klein and Woloszyn refer to this partitioning process as a hierarchical decomposition of G′, and the resulting tree is denoted as a decomposition tree [7]. The proof of the PTAS's $(1 + c)$ approximation ratio is dependent on this decomposition tree [7]. While performing the decomposition, edges in G′ are contracted in such a manner that assures costs are not modified too much as to rending the approximation solution infeasible. G′ is now considered a contacted graph.

Dynamic programming is then used to extract a tour from G′. Dynamic programming methods allow Grigni, Karger, Klein and Woloszyn to find, in polynomial time, the optimal tour T′ in G′ [7]. Note that the tour T′ of the contracted graph G′ is a sub-optimal tour of the original graph G. Finally, the optimal tour T′ of G′ is then translated into a valid tour T of G. Tour T will have a cost that is marginally greater than the cost of tour T′, which is due to the translation. However, Grigni, Karger, Klein and Woloszyn [5, 7] were able to prove that the tour T of G, which is returned by their PTAS algorithm, has a cost no more than $(1 + c)$ times an optimal tour of G.

References

[1] D. AppleGate, R. Bixby, V. Chvatal and W. Cook. *On the Solution of the Traveling Salesman Problems*. Documenta Mathematica – Extra Volume ICM, chapter 3, pp. 645-656, 1998.

[2] S. Arora. *The Approximability of NP-hard Problems.* Princeton University, 1998.

[3] S. Arora. *Polynomial Time Approximation Schemes for Euclidean Traveling Saleman and Other Geometric Problems.* Princeton University, 1996.

[4] S. Arora. *Nearly Linear Time Approximation Schemes for Euclidean TSP and other Geometric Problems.* Princeton University, 1997.

[5] S. Arora, M. Grigni, D. Karger, P. Klein and A. Woloszyn. *A Polynomial-Time Approximation Scheme for Weighted Planar Graph TSP*. Society for Industrial and Applied Mathematics, 1998.

[6] M. Garey and D. Johnson. *"Strong" NPcompleteness results: motivation, examples and implications*. Journal of the ACM, chapter 25, pp. 499-508, 1978.

[7] M. Grigni, D. Karger, P. Klein and A. Woloszyn. *A Approximation Scheme for Planar Graph TSP*. In Proc. IEEE Symposium on Foundatins of Computer Science, pp. 640-645, 1995

[8] J. S. B. Mitchell. *Guillotine subdivisions approximate polygonal subdivisions_ Part II - A simple polynomial-time approximation scheme for geometric k-MST, TSP, and related problems.* State University of New York, Stony Brook, 1996.

[9] E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan and D. B. Shmoys. *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization*. John Wiley & Sons, 1985.

[10] TSBLIB. http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/

[11] Wikipedia, the free encyclopedia - *Polynomial-time approximation scheme.* Retrieved October 27, 2006, from http://en.wikipedia.org/wiki/Polynomial_time_approximation_scheme

[12] Wikipedia, the free encyclopedia – *Traveling salesman problem, Euclidean TSP*. Retrieved October 12, 2006, from http://en.wikipedia.org/wiki/Traveling_salesman_problem#Euclidean_TSP

[13] Wikipedia, the free encyclopedia – *Quadtree*. Retrieved October 12, 2006, from http://en.wikipedia.org/wiki/Quadtree