

The Traveling Salesman Problem: A Comprehensive Survey

by

Leonardo Zambito

Submitted as a project for CSE 4080,
Fall 2006

1 Introduction

The Traveling Salesman Problem (TSP) is widely studied in Computer Science. There are many publications on the TSP, ranging back to at least the late 1940's. The TSP has held the interests of computer scientists and mathematicians because, even after about half a decade of research, the problem has not been completely solved. The TSP falls in a distinguished category of hard problems. The TSP can be applied to solve many practical problems within our daily lives. Thus, a solution to the TSP would be very beneficial. When we talk about a solution to the TSP, we are talking about a polynomial time solution to the general TSP, which will be explained further in chapter two.

This report aims to survey the TSP in a broad fashion. This survey will cover the TSP from its debut into computer science until present, including examples of practical applications of the problem. In chapter two, you will be given a definition of the general TSP and the history of the problem from its roots in mathematics and computer science up to present time. Chapter three provides an overview on the categories of NP-hard and NP-complete problems. This chapter then discusses why the TSP is NP-complete. Chapter four presents some real life uses of a solution to the TSP. The purpose of this chapter is to emphasize the usefulness of a solution to the general TSP. Chapter five attempts to shed some light on the NP-hard nature of the TSP by presenting and explaining some known polynomial time solutions to variations of the TSP. In chapter six, we explore variations of the TSP that are more specific to real life instances of the problem, but are still considered NP-hard. In chapter seven, we consider approximation algorithmic solutions to variations of the TSP mentioned in chapter six. Finally, we end the survey by presenting some insights on the TSP by exploring attempts to achieve an

exact efficient solution to the problem. In addition, we will look at the options being considered in future research to work towards a feasible solution.

2 The Problem – Definition and History

The TSP is stated as, given a complete graph, G , with a set of vertices, V , a set of edges, E , and a cost, c_{ij} , associated with each edge in E . The value c_{ij} is the cost incurred when traversing from vertex $i \in V$ to vertex $j \in V$. Given this information, a solution to the TSP must return the cheapest Hamiltonian cycle of G . A Hamiltonian cycle is a cycle that visits each node in a graph exactly once. This is referred to as a tour in TSP terms.

The essence of the traveling salesman problem is evident within many practical applications in real life. From a mail delivery person trying to figure out the most optimal route that will cover all of his/her daily stops, to a network architect trying to design the most efficient ring topology that will connect hundreds of computers. In all of these instances, the cost or distance between each location, whether it be a city, building or node in a network, is known. With this information, the fundamental goal is to find the optimal tour. That is, to determine an order in which each location should be visited such that each location is visited only once, and the total distance traveled, or cost incurred, is minimal. In the general TSP, there are no restrictions on the distance/cost values.

So, how and when did the traveling salesman problem first emerge within Mathematics and Computer Science studies? According to Lawler, Lenstra, Rinnooy Kan & Shmoys [15], no one really knows. The origins range back to the 1920's, when a mathematician by the name of Karl Menger brought it to the attention of his colleagues in Vienna [1]. The problem then worked its way into Princeton's mathematical community during the 1930's [1]. Then, in the 1940's, mathematician Merrill Meeks Flood,

publicized the name, TSP, within the mathematical community at mass [15]. It was the year 1948 that Flood publicized the traveling salesman problem by presenting it at the RAND Corporation [15]; according to Flood “when I was struggling with the problem in connecting with a school-bus routing study in New Jersey” (Flood, 1956). The RAND Corporation is a non-profit organization that is the focus of intellectual research and development within the United States [27]. In its early days, RAND provided research and analysis to the United States armed forces, but then expanded to provide such services for the government and other organizations [27]. The TSP soon became very popular. This popularity was probably attributed to a few factors, one of which is the prestige of the RAND Corporation. Another factor is the connection between the TSP problem and the rising combinatorial problems within linear programming. Finally, its title is definitely a factor, which demonstrates relevance towards many tasks evident within people’s daily lives.

The TSP demonstrates all the aspects of combinatorial optimization. During the 1950’s, Linear Programming was becoming a vital force in computing solutions to combinatorial optimization problems. This was due to the funding provided by the U.S. Air Force in the interest of obtaining optimal solutions to combinatorial transportation problems. As mentioned, this is one of the reasons why the TSP was in the interest of RAND. Attempts to solve the TSP were futile until the mid-1950’s when Dantzig, Fulkerson, and Johnson [9] presented a method for solving the TSP. They showed the effectiveness of their method by solving a 49-city instance [9]. However, it became evident, as early as the mid 1960’s, that the general instance of the TSP could not be solved in polynomial time using Linear Programming techniques. In fact, it was

conjectured that the TSP, and problems alike, posed such computational complexity that any programmable efforts to solve such problems would grow superpolynomially with the problem size. These categories of problems became known as NP-hard, which will be discussed in more detail in chapter three. There has been a lot of progress in dealing with NP-hard problem such as the TSP. Solutions have been found to instances of the TSP with limited input sizes, some of which are mentioned in chapter eight. Polynomial time solutions have been found for special cases of the TSP. Some of these special cases are covered in chapter five. Researchers have even resorted to finding polynomial time approximation algorithms for NP-hard variations of the TSP, as discussed in chapters six and seven. However, until this very day, an efficient solution to the general case TSP, or even to any of its NP-hard variations, has not been found.

3 NP-Completeness and the TSP

In this chapter, you will be given a brief overview of NP-hardness and NP-completeness, including definitions, as well as an explanation on why the traveling salesman problem is NP-complete in the general case, and is NP-hard even in many special cases.

In order to understand NP-completeness, you must first understand the theory that is used to classify all computational problems and the algorithms used to solve them. That is the theory of Computational Complexity [22]. Recall that an algorithm is a set of step-by-step instructions that, when executed in the order specified, will solve a certain problem. Problems are basically classified as being within one of two categories. A problem is considered 'easy' if it can be solved by an algorithm that runs in polynomial time. On the other hand, a problem is considered 'hard' if it cannot be solved in

polynomial time. Of course, these two classifications or definitions are not very elegant, but they are the basis of the computational complexity theory of problems that was developed.

The computational complexity theory divides problems into two classes. A complexity class is a set of all problems that can be solved in some certain amount of time, and that uses a certain amount of computing resources. An important note is that any problem that has been proven to be ‘hard’, in the sense that there exists only an exponential or superexponential solution to the problem, or that no algorithmic solution to the problem exists, does not fall within a complexity class. The reason for this is because it is not known how much time it will take to solve such a ‘hard’ problem. Even with the super computers that exist today, it may take years to reach a solution, and the computer’s resources will probably run out by that time. In essence, algorithms for problems that are proven to be ‘hard’ are not practical and are very unpredictable.

Formally, the computational complexity theory is restricted to only decision problems [15]. A decision problem is one in which the solution to the problem is either a ‘yes’ or ‘no’ answer. As constricting as it may sound to limit ourselves only to decision problems, in reality all problems can be reformulated as decision problems. To see this, let us briefly consider how one would go about reformulating the TSP into a decision problem.

The basic idea used to transform the TSP into a decision problem is to have two algorithms. One will be known as TSP_SOLUTION. This algorithm takes as input a graph, G , with a set of vertices, V , a set of edges, E , and a cost, c_{ij} , associated with each edge in E . TSP_SOLUTION then calls a helper algorithm known as TSP_DECISION.

TSP_DECISION takes as input the same graph, G , and a numeric bound value, B . TSP_DECISION returns ‘yes’ if there is a tour that exists in G , whose cost is less than B , and returns ‘no’ otherwise. TSP_SOLUTION must run in polynomial time, not taking into account the running time of TSP_DECISION. Therefore, TSP_SOLUTION calls TSP_DECISION a polynomial amount of times. TSP_SOLUTION finds the optimal tour by first using TSP_DECISION to determine the cost, C , of an optimal tour, if one exists. Once this cost is known, TSP_SOLUTION determines which edges are not traversed in some optimal tour. This is done by going through each edge, one at a time, setting the cost for that edge to infinity, and then using TSP_DECISION to determine if a tour with cost C still exists. If so, then that edge is removed. In the end, only the edges in an optimal tour will be present. Thus, the running time of TSP_SOLUTION will be polynomial in running time if and only if TSP_DECISION runs in polynomial time.

This brings us to the formal definitions of the two classifications of decision problems defined by the computational complexity theory. The first classification is the P class. If a problem can be solved by a deterministic algorithm in Polynomial time, then it is considered to be part of the P class of problems [21]. For a problem to be solved in polynomial time means that a correct ‘yes’ or ‘no’ answer will be returned in polynomial time based on the input size. The second classification of problems is the NP class, which stands for the Non-deterministic Polynomial [21]. An important relation between P and NP is that $P \subseteq NP$, which is because a deterministic algorithm is just a special case of a non-deterministic algorithm [15]. The NP classification of problems is a very important concept that is not as easily defined as the P class. The NP class is defined by three equivalent formulations.

The first formulation is the succinct property [15]. This property states that any ‘yes’ result can be certified in polynomial time [15]. In other words, a ‘yes’ instance to the problem contains a ‘succinct certificate’, which means that a certification or check program can be run on the ‘yes’ result and confirm, in polynomial time, that the result is in fact correct [15].

The second formulation involves the non-deterministic construct [15]. This formulation holds when a non-deterministic algorithm is used to solve the problem at hand. You can classify an algorithm as non-deterministic if the algorithm contains an instruction of the following form [15].

go to both label 1, label 2

In this case, the algorithm launches two paralleled streams of computation. Consider now the case when ‘label x’, for all $x > 0$, executes an instruction of this form. You will end up with an exponential explosion of paralleled streams, or branches, of computation. This could continue until all of the available computing resources are depleted, in which case a solution would not be reached. However, such a non-deterministic algorithm could have a polynomial bound. In this case, an input to the algorithm would be a ‘Yes’ instance if any one of the branching paths returns ‘Yes’ within a polynomial number of steps. Otherwise, the input would be a ‘No’ instance.

Finally, the third formulation is one which was studied by Dantzig and it involves Integer Programming [15]. For more detail on Integer Programming, please refer to G. B. Dantzig [10, 11]. A problem would fall under this formulation if that problem could be reduced to Integer Programming in Polynomial Time [15].

These three formulations are equivalent to each other. Together, they define the class of NP decision problems. As formally stated, if A is a decision problem, then A has a succinct certificate property, $A \in \text{NP}$ and A is transferable to Integer Programming in polynomial time [15]. Since $P \subseteq \text{NP}$, then if $A \in P$, then $A \in \text{NP}$. A problem B is considered to be NP-hard if for all problems $B' \in \text{NP}$, you can show that there exists a polynomial time many-one reduction to B [26].

So, where does NP-completeness come into the picture? Let us say that we were able to show that every NP-hard problem can be reduced to some other NP-hard problem B . That is, any NP-hard problem $X \in \text{NP}$ can be transformed to B in polynomial time so that a solution to B would yield a solution to X . In this case, B is considered to be NP-complete. The interesting characteristic of NP-complete problems is that if you can prove that an NP-complete problem $B \in P$, then you have shown that $P = \text{NP}$, which means that every NP-complete problem can be solved in polynomial time in the worst case. On the other hand, if you can prove that an NP-complete problem $B \notin P$, then you have shown that $P \neq \text{NP}$, which means that none of the NP-complete problems can be solved in polynomial time in the worst case. It thus stands that the million dollar question is, does $P = \text{NP}$? Literally, “The Clay Mathematics Institute has offered a USD 1,000,000 prize for a correct solution” [22]. Currently, no one has been able to prove that $P = \text{NP}$ or that $P \neq \text{NP}$. However, it is highly speculated that $P \neq \text{NP}$. For many years, researchers have tried to find ways of solving NP-hard problems in polynomial time, including the TSP. Therefore, if you can show that a problem is NP-hard, then you know that it is extremely unlikely that you will find a polynomial time solution to the problem.

It is without a doubt that the TSP's NP-completeness has attributed greatly to its popularity.

So, what is it about the TSP that makes it NP-complete? Moreover, how does one go about proving that a problem, such as the TSP, is NP-complete? To answer both of these questions in a simplistic manner, because a detailed proof and explanation is far beyond the scope of this survey, we will use the popular reduction method. We will take a problem, call it H, that has already been proven to be NP-complete. Then, we will show that you can solve H by reducing it to the TSP. Once we have shown this, we can say that if TSP is solvable in polynomial time, then we can solve H in polynomial time. At which point we reach a contradiction. We then conclude that TSP cannot be solved in polynomial time because it has already been proven that H cannot be solved in polynomial time. Moreover, TSP must be NP-complete because H is NP-complete.

In proving that TSP is NP-complete, we will choose H to be the popular Hamiltonian Cycle problem, which is known to be NP-complete. At a close look, it is evident that the Hamiltonian Cycle problem is a special case of the TSP. Simply modify the input graph to the TSP algorithm by setting the costs of all the existing edges to be some fixed finite constant. Then, if any tour is found, that tour is a Hamiltonian Cycle. Thus, the TSP must be NP-complete because it has been proven that the Hamiltonian Cycle problem is NP-complete.

4 Applications – Real Life Uses of the Problem

There are many practical real life uses of the TSP. Of course, the most common of which are transportation routing problems. In this chapter, some of the most popular

applications will be mentioned. In addition, this chapter will touch upon some generic instances of the TSP that can easily be adapted for use within many real life problems.

The most popular application of the TSP that comes to mind is finding a route that a salesman would take in order to visit every geographical location in a specified list such that minimum total distance is traveled. Consider the case of a salesman traveling from door to door in a certain sub-division of homes. Would it not be very convenient if the salesperson could obtain a list of all the homes in that subdivision specified in the most optimal order to visit. How about a salesperson that needs to visit hundreds of cities spread throughout an entire country? Or, what about a musical band going on tour throughout the world? Knowing the optimal tour that will visit each city once could potentially save the traveler days or even weeks of traveling time. Consider a postal delivery person who goes to work in the morning with a truck full of parcels to deliver. In what order should those parcels be delivered to minimize the total distance traveled?

For all the instances mentioned so far, the nodes in the graph would correspond to the geographical locations, and the distances would be metric values based on the lengths of the roads connecting the locations. Presently, most of the world, including cities, buildings and land marks, has already been mapped electronically in a plane. To solve these instances mentioned, one would only need to specify the desired locations to be visited, and then let a TSP solution algorithm do the rest.

There are other important practical uses of the TSP. Consider some of the machines in an assembly line. There are machines whose sole purposes are to drill various holes in a certain piece of material. The material may be a circuit board, the frame of a vehicle, or even a piece of wood to be used building a book shelf. The drill is

repositioned by motors that slide along tracks such that the drill could move to any position within a certain area. It will take a certain amount of time to reposition the drill depending on the distance that drill needs to move. A solution to the TSP could be used to find the optimal order in which the holes should be drilled. In the case of an assembly line, saving several seconds to complete the process for each work-piece means producing much more work-pieces by the end of the day. To solve this problem using the TSP application, simply let the vertices represent the locations that the holes need to be drilled and let the edges be the distances between them.

Another application that a solution to the TSP can be applied to is electronic or mechanical connection placement. Consider the wiring of a circuit board, or the electrical wiring within a large building, or even the plumbing layout within a building. In many of these cases, the connections need to be laid out such that the components are all connected in a cycle. In the case of a circuit board, the connections are the wires and the components are the transistors, resistors, etc. When talking about the electrical setup of a building, the connections are the wires and the components are the switches, plugs, light fixtures, etc. Finally, for the plumbing layout of a building, the connections are the pipes and the components are the faucets and water taps. In all of these cases, you will be trying to find a shortest Hamiltonian path in order to save material and to optimize flow by reducing the length of the cycle. Connecting circuits or wiring electronic components so that the current has to travel as minimum a distance as possible will ultimately increase efficiency and overall performance.

The final application that we will look at is the multiple salesmen model. Consider the task of having to visit a set of cities, where each city has to be visited

exactly once by any of the m salesmen on your staff. If you hire salesman j , then you must pay salesman j a fixed cost of d_j . Each of the m salesmen must complete a subtour, and the combination of all the subtours that each of the m salesmen embark on must result in each city in set being visited exactly once. The salesmen start and end their subtours at the same home base location. The object is to determine which salesmen to hire and what the subtours should be in order to minimize the total distance traveled and the cost of hiring the salesmen. This problem can be modeled as the original TSP by adding $(m - 1)$ additional vertices, denoted $-1, \dots, -(m - 1)$ to the input graph. Now, the m salesmen, numbered 0 to $(m - 1)$, are represented by the source vertex and the new $(m - 1)$ vertices, where the source vertex represents salesman 0 . Then, edges are added to connect these $(m - 1)$ new vertices to the rest of the vertices in the original graph. The costs of the newly added edges are determined by adding the costs d_j , for $0 \leq j \leq (m - 1)$, to the cost of existing edges in the graph. For more details on transforming a m salesmen graph into a TSP graph, please refer to page 24 of Lawler, Lenstra, Rinnooy Kan and Shmoys [15].

5 Polynomial Time Solvable Variations

This chapter talks about some of the special cases of the TSP for which there exist efficient polynomial time solutions. These well-solved special cases are divided into two categories. In the first category are cases in which restrictions are put on the matrix that contains the costs of the edges in the graph, but there are no restrictions on the structure of the graph. In the second category are cases in which restrictions are put on the structure of the graph, but there are no restrictions on the edge costs. To clarify, a cost matrix of a graph G refers to an $n \times n$ matrix, where c_{ij} , for $1 \leq i \leq n$ and $1 \leq j \leq n$, refers

to the cost incurred when traversing the edge in G that connects vertices i and j . Some of the most popular cases of the TSP that fall in these categories will be listed here.

The first variation we will look at is known as the Small TSP. In this variation, restrictions are put on the cost matrix. A cost matrix C is considered small if there exists n -dimensional vectors a and b , where $a_1 \leq a_2 \leq \dots \leq a_n$ and $b_1 \leq b_2 \leq \dots \leq b_n$. This instance can be effectively solved in polynomial time because there is a small amount of different values to consider when looking for the most inexpensive tour. A detailed solution to this instance is available in E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan and D. B. Shmoys [15].

The Upper Triangular Matrices version of the TSP can also be solved in polynomial time. A matrix C is considered upper triangular if $i \geq j$ implies $c_{ij} = 0$. Thus, more than half of the matrix is filled with 0's, and the non-zero values form a right angle triangle whose right angle is at the top right corner of the matrix. Please refer to E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan and D. B. Shmoys [15] for the details of a polynomial time solution to this instance of the TSP.

S. Kaushal [14] talks about the bitonic TSP. In this instance, the vertex set is $V = \{0, 1, 2, \dots, n-1\}$, for some $n \geq 1$, and the distances between the vertices are Euclidean. A tour $t = (0, i_1, i_2, \dots, i_k, n-1, i_{k+1}, i_{k+2}, \dots, i_{n-2})$ of V is considered bitonic if and only if $0 < i_1 < i_2 < \dots < i_k < n-1$ and $n-1 > i_{k+1} > i_{k+2} > \dots > i_{n-2} > 0$. This tour can be found very efficiently. One simply starts at vertex 0 and then visits the rest of the vertices based on their x-coordinate value. The solution is obtained via a dynamic programming algorithm [14]. Please refer to S. Kaushal [14] for a detailed solution.

Finally, we consider the Chinese Postman Problem, which is essentially the same as the TSP, but with one important distinction. The goal in this problem is to find the shortest closed walk, containing all of the edges in the graph at least once. In a closed walk, vertices can be visited more than once, and hence edges can be traversed more than once. This problem's complexity is known to be polynomial for both the directed and undirected graph versions of it [19]. The Chinese Postman Problem has many applications, but unfortunately it has not been given nearly as much attention as the TSP [19]. The Chinese Postman Problem, like the TSP, is easily stated, but not easily solved. The polynomial time solutions to the Chinese Postman Problem is rather complicated [19].

6 NP-hard Variations

This chapter talks about some of the popular variations of the TSP problem that are more useful in real life applications. Restrictions are put on the general TSP, either to make it easier to solve, or simply because such restrictions allow the problem to reflect certain applications. For instance, a restriction that says all of the edge costs must be non-negative values may be placed on the general TSP. This restriction is necessary when the input instance involves physical locations of buildings and the edge costs represent the distances between the different buildings. It is obvious that distance values cannot be negative.

Bellow, we will list variations of the general TSP in which restrictions are placed. However, despite these restrictions, the variations listed here are still considered NP-hard. The input for each of these variations is a graph G that contains a set of vertices V and a set of edges E connecting the vertices in V .

In the symmetric TSP variation, all of the edge costs are symmetric. This means that, for all nodes in the graph, the cost incurred, when moving from node a to node b , is the same as the cost incurred when moving from node b to node a . On the other hand, the asymmetric TSP does not have such constraints. The general TSP is considered asymmetric. The input to the asymmetric TSP would be a directed graph.

In the Metric TSP, all of the edge costs are symmetric and they satisfy the triangle inequality. The triangle inequality property means that for any three nodes a , b and c , the cost of going from node a directly to node c is always cheaper than going from node a to node c by passing through node b . In addition, the nodes are points in some space. The edge costs are determined by calculating the metric distance between the points.

Finally, in the Euclidean TSP, all of the nodes lie in the plane, which means it is symmetric and the triangle inequality holds. The cost of each edge e , connecting nodes a and b , is defined by the Euclidean distance between the nodes a and b . In general, the plane can be d -dimensional, where $d > 1$.

7 Approximate Solutions to NP-hard Variations

Unfortunately, many of the real life applications of the Traveling Salesman Problem involve the NP-hard variations. As previously mentioned, an efficient exact solution to any of these NP-hard variations has not been found. Moreover, the exact solutions that we do have are unpractical for most uses even with today's super computers. However, there is some light at the end of the tunnel. Efficient approximation algorithms have been developed, which can be very useful in practice. The approximation technique is a popular approach in tackling NP-hard problems. The idea is that, although it may be near impossible to find an efficient exact solution to NP-

hard problems, we can try to find efficient approximation solutions; a rather successful approach. An approximation algorithm is one in which, for some fixed constant c , solves a problem by returning a solution that is within a factor c of the optimal solution. The running time is required to be polynomial, otherwise the algorithm is intractable on large instances. If there exists a polynomial time approximation algorithm for an NP-hard problem N , then N is considered to be approximable. Note that it is possible to show that an NP-hard problem is not approximable [20].

The polynomial running times of approximation algorithms can be quite costly, for instance, $O(n^{1000})$. However, there are more useful classes of approximation algorithm known as approximation schemes. A Polynomial Time Approximation Scheme (PTAS) is an algorithm that takes as input an instance of an optimization problem and a value c . It then returns a solution to that optimization problem that is within a c factor of the optimal solution, where $\epsilon = (c-1)$ is the relative error in the approximation. The running time of a PTAS must be polynomial in the input size for any fixed value of c , but may or may not be polynomial in $1/(c-1)$. A subset of PTAS is the Fully Polynomial Time Approximation Scheme (FPTAS). FPTAS algorithms behave identical to PTAS algorithm except for a restriction that is added to the running time requirements. For FPTAS algorithms, the running time must be polynomial in the input size and in $1/(c-1)$.

It was recently proved by S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy [6] that if $P \neq NP$, then the Metric TSP does not have a PTAS. However, we have found constant approximation algorithms for the metric TSP. The best of which is Christofides' algorithm, done in CSE4101, with an approximation ratio of $3/2$ [7]. There is another algorithm, known as the Held-Karp heuristic [3], which has been conjectured

to have an approximation ratio of $4/3$. However, because this approximation ratio has not been proven to be a worst-case upper bound, Christofides' $3/2$ remains the best known upper-bound approximation solution to the metric TSP [3].

A very simple and popular approximation algorithm for the Metric TSP is the Double-MST [8, CSE4101]. In this algorithm, you begin by finding the minimum spanning tree, or MST, of the graph. You then list the vertices based on the order they are visited in a preorder walk of the MST. This ordered list is the tour that is returned. It has been proven that this algorithm returns a tour whose cost is no more than twice the optimal tour.

Although a PTAS algorithm has not been found for the metric TSP, the same cannot be said for the Euclidean TSP. After Christofides' discovery, many researchers attempted to find a better approximation algorithm or a PTAS for the Euclidean TSP. Eventually S. Arora succeeded in discovering a PTAS for the Euclidean TSP around the year 1996 [3]. Arora's algorithm computes a tour with approximation ratio $(1 + c)$, for any given $c > 0$. The algorithm recursively partitions the plane and then uses dynamic programming to find a tour that crosses each line of the partition at most $O(c)$ times. The running time of such an algorithm in a 2-dimensional plane is $O(n(\log n)^{O(c)})$. The algorithm also works in for a d -dimensional plane, with a running time of $O(n(\log n)^{O(\sqrt{dc})^{d-1}})$. Please refer to S. Arora [4] for details on the solution to this PTAS.

J. S. B. Mitchell [17] also discovered a PTAS for the Euclidean TSP around the same time that Arora did. Mitchell's algorithm uses an " m -guillotine subdivision" concept, which is a polygonal subdivision with one added property [17]. In a polygonal

subdivision, a line, also referred to as a cut, is drawn through a subdivisions S , thus splitting S into two new subdivisions S_1 and S_2 . The added property is that the cut's intersection with the edges in S consists of $O(m)$ connected components, and the new subdivisions, S_1 and S_2 , are also m -guillotine" [17]. Mitchell's PTAS algorithm returns a tour whose cost is within a factor of $(1 + 2\sqrt{2}/m)$ from an optimal tour cost [17]. The algorithm accomplishes this in $O(n^{20m+5})$ time [17]. Refer to J. S. B. Mitchell [17] for more details on this PTAS solution.

S. Arora, M. Grigni, D. Karger, P. Klein and A. Wolszyn [5] present a PTAS for a planar graph containing edge weights or costs. (A planar graph is a graph that does not contain any edge crossings, and can be drawn on the plane.) Thus, a weighted planar graph version of the TSP is a special case of the Euclidean TSP. The algorithm produces a tour that is within a factor $(1 + c)$, for some specified constant $c > 0$, of the optimal tour. The running time for this approximation algorithm is $n^{O(1/c^2)}$. The algorithm begins by finding a spanner of the input graph, with edge weights preserved. Then, it obtains a step-by-step decomposition of the input graph. Each step partitions each of the current components into at least two pieces, where each piece contains a constant factor of the vertices in its parent piece. Please refer to S. Arora, M. Grigni, D. Karger, P. Klein and A. Wolszyn [5] for the details on this PTAS algorithms.

8 Working Towards an Exact Solution

One of the ultimate goals in computer science is to find computationally feasible exact solutions to all the known NP-Hard problems; a goal that may never be reached.

Feasible exact solutions for the TSP have been found, but there are restrictions on the input sizes.

An exact solution was found for a 318-City problem [Crowder & Padberg, 1980]. The basic idea in achieving this solution involves three phases. In the first phase, a true lower bound on the optimal tour is found. In the second phase, the result in the first phase is used to eliminate about ninety-seven percent of all the possible tours. Thus, only about three percent of the possible tours need to be considered. In the third phase, the reduced problem is solved by brute force. This solution has been implemented and used in practice. Experimental results by AppleGate, Bixby, Chvatal and Cook [1] showed that running this algorithm, implemented in the C programming language and executed on a 400MHz machine, would produce a result in 24.6 seconds of running time.

Other exact solutions have been found. As mention in [1], a 120-city problem by Grötschel [1980], a 532-city problem by Padberg and Rinaldi [1987], a 666-city problem by Grötschel and Holland [1991], a 1,002-city problem and a 2,392-city problem by Padberg and Rinaldi [1991]. However, none of the algorithms that provide an exact solution for input instances of over a thousand cities are practical for everyday use. Even with today's super computers, the execution time of such exact solution algorithms for TSPs involving thousands of cities could take days. What does this say for the circuit board wiring applications of the TSP, where the total amount of components are in the tens of thousands, or even millions?

Computer hardware researchers have been making astonishing progress in manufacturing evermore powerful computing chips. Moore's Law [25], which states that the number of transistors that can fit on a chip will double after every 18 months, has held

ground since 1965. This basically means that computing power has doubled every 18 months since then. Thus, we have been able to solve larger instances of NP-hard problems, but algorithm complexity has still remained exponential. Moreover, it is highly speculated that this trend will come to an end because there is a limit to the miniaturization of transistors. Presently, the sizes of transistors are approaching the size of atoms. With the speeds of computer processors rounding the 5GHz mark, and talks about an exponential increase in speeds of up to 100GHz [25], one might consider the possibility of us exceeding any further need of computational performance. However, this is not the case. Although computing speeds may increase exponentially, they are, and will continue to be, surpassed by the exponential increase in algorithmic complexity as problem sizes continue to grow. Moore's law may continue to hold true for another decade or so, but different methods of computing are being researched. The most notable one in the context of this survey is Quantum computing.

In a quantum computer, particles are used to represent data by manipulating their quantum properties. In a transistor based computer, otherwise known as a classical computer, data is represented in bits. Quantum computers use qubits to represent data. The major distinction between the two is that an n bit register is always in a definite state defined by a combination of n 0s and 1s, whereas an n qubit register can be in a superposition of all the 2^n different states [28]. This means that to record the data in an n qubit register would require 2^n complex numbers [28]. This phenomenon of the quantum computer, in relation to complexity theory, is that it has been shown that it may be possible to solve NP-complete problems in polynomial time. The key is to first figure out how to design a quantum computer with nonlinear operators [28]. If this can be done,

then we end up with a powerful method of computing, referred to as Quantum Parallelism (David Deutsch, 1985). To see this, consider a non-deterministic algorithm N that contains a command such as ‘**go to both** label1, label2’. If the input size to N is n , the ‘**go to both** label1, label2’ command will typically be called n times. Thus, there will be 2^n paralleled streams of computation, and therefore 2^n different possible results. A Quantum Parallel computer could perform these computations in a paralleled fashion, thus performing the necessary calculations in polynomial time [16, 28]. Each of these 2^n different possible solutions can be represented within a single n qubit register. Unfortunately, a detailed discussion on the theory of quantum computing is far beyond the scope of this survey. For more details on Quantum Computing you are encouraged to read S. Lloyd [16].

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] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy. *Proof verification and intractability of approximation problems*. IEEE Symposium on Foundations of Computer Science, 1992.

- [7] N. Christofides. *Worst-case analysis of a new heuristic for the traveling salesman problem*. Academic Press, New York, 1976.
- [8] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms: Second Edition*. McGraw-Hill, Toronto, 2001.
- [9] G.B. Dantzig, R. Fulkerson, and S.M. Johnson. *Solution of a large-scale traveling salesman problem*. *Operations Research* 2, pp. 393-41, 1954.
- [10] G. B. Dantzig. *Linear Programming*. Department of Management Science and Engineering, Stanford University, Stanford, California, 1991.
- [11] G. B. Dantzig. *Linear Programming and Extensions*. Princeton University Press, Princeton, NJ, 1963.
- [12] K. Elbassioni, A. V. Fishkin, N. H. Mustafa, and R. Sitters. *Approximation Algorithms for Euclidean Group TSP*. Max-Planck-Institut für Informatik, Saarbrücken, Germany, 2005.
- [13] M. M. Flood. *The traveling-salesman problem*. *Oper. Res.* 4, 61-75. 1956.
- [14] S. Kaushal. *Empirical Analysis of Algorithms for the Traveling Sales Problem*. Project for CMSC 641, 1996.
- [15] 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.
- [16] S. Lloyd. *Quantum-Mechanical Computers*. *Scientific American: The Solid-State Century*, 1997.
- [17] 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.
- [18] G. Reinelt. *The Traveling Salesman Computational Solutions for TSP Applications*. Springer Berlin / Heidelberg, 1994.
- [19] H. Thimbleby. *The directed Chinese Postman Problem*. University College London Interaction Centre, London, 2000.
- [20] Wikipedia, the free encyclopedia – *Approximation algorithm*. Retrieved October 4, 2006, from http://en.wikipedia.org/wiki/Approximation_algorithm
- [21] Wikipedia, the free encyclopedia – *Complexity classes P and NP*. Retrieved October 4, 2006, from http://en.wikipedia.org/wiki/Complexity_classes_P_and_NP

[22] Wikipedia, the free encyclopedia – *Computational complexity theory*. Retrieved October 12, 2006, from http://en.wikipedia.org/wiki/Computational_complexity_theory

[23] Wikipedia, the free encyclopedia – *Hamiltonian path problem*. Retrieved October 7, 2006, from http://en.wikipedia.org/wiki/Hamiltonian_path_problem

[24] Wikipedia, the free encyclopedia – *Merrill Flood*. Retrieved November 10, 2006, from http://en.wikipedia.org/wiki/Merrill_Flood

[25] Wikipedia, the free encyclopedia - *Moore's law*. Retrieved November 21, 2006, from http://en.wikipedia.org/wiki/Moore's_law

[26] Wikipedia, the free encyclopedia - *NP-hard*. Retrieved October 4, 2006, from <http://en.wikipedia.org/wiki/NP-hard>

[27] Wikipedia, the free encyclopedia - *RAND*. Retrieved November 10, 2006, from <http://en.wikipedia.org/wiki/RAND>

[28] Wikipedia, the free encyclopedia - *Quantum Computer*. Retrieved October 21, 2006, from http://en.wikipedia.org/wiki/Quantum_computer