2001

The traveling salesman problem improving K-opt via edge cut equivalence sets

Eric Holland

THE TRAVELING SALESMAN PROBLEM
IMPROVING K-OPT VIA EDGE CUT EQUIVALENCE SETS

A Thesis
Presented to the
Faculty of
California State University,
San Bernardino

In Partial Fulfillment
of the Requirements for the Degree
Master of Science
in
Computer Science

---

by
Eric Holland
June 2001

THE TRAVELING SALESMAN PROBLEM

IMPROVING K-OPT VIA EDGE CUT EQUIVALENCE SETS

A Thesis

Presented to the

Faculty of

California State University,

San Bernardino

---

by

Eric Holland

June 2001

Approved by:

[redacted]

Owen Murphy 4-16-2001

Date

[redacted]

Yasha Karant

[redacted]

Kay Zemoudah

ABSTRACT

The traveling salesman problem (TSP) involves finding a minimum cost Hamiltonian circuit in a weighted graph. Many heuristic approaches have been taken to try to efficiently solve this problem which is known to be NP-complete. Such heuristics include general-purpose techniques like simulated annealing, genetic algorithms, and ant colony systems. Many papers have been written on the application of these techniques to the TSP. While these techniques differ in their approaches to solving combinatorial problems, most of the successful implementations share a common underlying heuristic specific to the TSP. Namely, the general purpose algorithms will employ an underlying edge swapping heuristic to improve performance and tour quality. These edge-swapping techniques, known as K-opt where K is the number of edges being swapped, have proven to be the workhorse of most TSP implementations. In this paper, we introduce a new technique to improve upon K-Opt by utilizing edge cut equivalence sets. These sets allow for exhaustive K-Opt to be applied to more K-Sets of edges without exhaustively applying the heuristic to all possible tour reconstructions.

TABLE OF CONTENTS

CHAPTER ONE: INTRODUCTION

This paper is divided into four chapters. Chapter one is the introduction. Chapter two covers background information on the traveling salesman problem (TSP) and discusses existing techniques used to solve the problem. It includes an introduction to some of the most successful techniques for solving the TSP, which are based on edge swapping techniques known as K-Opt. Chapter two also stresses the significance of K-Opt as a local optimizer used in many general heuristic approaches to the TSP. These approaches include prevailing techniques such as genetic algorithms, simulated annealing, and ant colonies. This background material is necessary in order to introduce a new technique for solving the TSP. The new technique is based on a special partitioning of all possible K-Opt tour reconstructions into a group of sets. We call these new sets edge cut equivalence sets. Edge cut equivalence sets and their impact on the TSP are the main focus of this thesis. Chapter three discusses edge cut equivalence sets in detail and illustrates how they can be used to create a more efficient implementation of K-Opt by reducing the number of solutions that must be analyzed. Practical methods for including the new technique in existing TSP implementations are discussed as well. Finally, chapter four is a brief exposition on some of the important references used in developing the

new methods introduced in this paper. This chapter is intended to be of assistance to future researchers looking into related topics.

CHAPTER TWO

Traveling Salesman Problem and Existing Techniques

The traveling salesman problem (TSP) has become a classic in the field of combinatorial optimization. Attracting computer scientists and mathematicians, the problem involves finding a minimum cost Hamiltonian cycle in a weighted graph. The problem is quite easy to state both in laymen's terms and mathematically. In laymen's terms, the TSP involves a salesman who desires to start at his home city, visit each city in his sales area exactly once, and then return home - all at a minimum cost. Mathematically, we seek to find a permutation Π of the set {1,2,3...,n} that minimizes the quantity

$$C(\pi) = \sum_{i=1}^{n-1} d_{\pi(i),\pi(i+1)} + d_{\pi(n),\pi(1)}$$

where d_{ij} denotes the distance from city i to city j [14]. There are many variations on the TSP such as whether or not the distances d_{ij} and d_{ji} are equivalent. If the distances are equivalent, the TSP problem is called symmetrical otherwise it is called asymmetrical. While many other variations exist, the most prevalent form of the TSP is the symmetric TSP on a complete graph. Unless otherwise mentioned, this paper will always assume a symmetric instance on a complete graph. It is important

to note that no further assumptions are made. This distinction is important as some TSP algorithms are made more efficient by assuming certain properties such as problem instances that obey the two dimensional Euclidean distance formulas. This special case is sometimes called the metric TSP and special algorithms exist for finding its solution. We make no such assumptions about the problem instances to be solved and maintain the ability to solve instances that do not comply with the metric TSP.

The ease of stating the TSP, combined with the fact that it is extremely difficult to solve efficiently has led to its wide popularity. It is important to note that it is not difficult to solve if we don't worry about efficiency. In fact, an algorithm can be implemented in a few hundred lines of code. The critical point is that it is extremely difficult to solve efficiently. In fact, the TSP has been proven to be NP-complete [11] adding to its popularity in research. Heuristic approaches, which do not guarantee an optimal solution but efficiently find solutions of high quality, dominate the literature. This is due to the dramatic exponential growth of the solution space of the TSP, which makes it difficult to guarantee an optimal solution in an acceptable amount of time.

To better understand the difficulty involved in solving the TSP, consider the growth of its solution space. Given a problem instance of N cities, we will be given the first city in which our tour starts. For the second city, (N-1) choices remain. For the third city, we have (N-2) and so on. This leads us to a growth rate of the TSP solution space of:

1*(N-1)*(N-2)*(N-3)...*3*2*1

Which, of course, is simply (N-1)!. To be precise, we should note that half of these solutions could be ignored since they will represent the same cycles in reverse order. In other words, a tour from A-B-C-D is the same as a tour from A-D-C-B, only the directions have been reversed. Asymptotically, however, we can simply use (N-1)! as the TSP solution space growth rate. To get a practical feel for how fast the factorial function grows, consider a few factorial values:

10! ≈ 3.6 Million (We can relate to this)

100! ≈ 10^{155} (This is so large, we cannot easily relate to it)

1000! ≈ 10^{2500} (1 followed by 2,500 zeros. This is enormous it's difficult to even find an example of something so huge)

Even with these numbers in mind, some people fail to get an appreciation for how difficult it is to solve a problem with such a growth rate. To drive the point home, assume that some creative researcher was able to solve the 1,000 case from above covering the 10^{2500} possibilities with some clever algorithm. Let's say this researcher's algorithm was able to solve the problem in 1 day of running time. While this may seem to be a significant breakthrough, in truth, we would have gained very little, for consider what would happen if we added only 3 more cities to the problem and required the researcher to solve an instance of 1003 cities.

1000 cities (Solved in 1 day)

1001 cities (How long would this take? It's a factorial growth rate so we can conjecture that it would take approximately 1000 times as long. That's 1000 days or approximately 3 years and we've only added one city.)

1002 cities (Again, this would be approximately 1000 times more difficult than the previous case which required one year. Having added only 2 cities, we have gone from 1 day to 3,000 years or 3 millenium)

1003 cities (Continuing the pattern, we have 3,000 * 1,000 or 3,000,000 million years. Back to the drawing board for the previously happy researcher)

Having established the difficulty of the problem, we will now look at prevailing approaches to implementing its solutions.

Prevailing Techniques, the Importance of Edge Swapping, and a New Technique

Since the TSP is so difficult to solve for the case in which we can guarantee an optimal solution, the focus of most research has been on heuristic approaches, which do not guarantee an optimal solution but produce good solutions efficiently. Many heuristic approaches have been developed to attack the TSP problem along these lines. Well-known general-purpose techniques like simulated annealing, genetic algorithms, and ant colony systems have been heavily researched. Numerous papers have been written on each of the mentioned heuristics in solving the TSP. While these general heuristics differ greatly, a detailed review reveals a strong underlying commonality in most of the implementations reporting good results. This commonality is the inclusion of an underlying edge swapping heuristic to improve performance and tour quality. These edge-swapping techniques, known

as K-opt where K is the number of edges being swapped, have proven to be the workhorse of most TSP implementations. In fact, it is has not been well established as to whether any of these general heuristics offer any advantage over any others. Undoubtedly, however, were the edge swapping to be removed from the implementations, the performance and effectiveness of the implementations would likely suffer greatly. Figure 1 illustrates the importance of edge swapping techniques.

One such example of how general heuristics for the TSP have come to rely on edge swapping can be found in [14] in which the author clearly states the importance of a local search operator in genetic algorithms. In establishing the importance of incorporating some type of local search operator, the author in [14] states that "The results published in the literature indicate that it is necessary to combine some of these methods in order to arrive at high quality solutions". These other methods or local search operators, predominately, involve some type of edge swapping. In fact in [14], the author augments his genetic algorithm by utilizing a form of edge swapping called Lin-Kernighan which will be discussed shortly.

As an additional example of the reliance upon edge swapping, consider [16] in which ants are used as agents to find improved tours to the TSP. Ant systems work by

applying varying levels of pheromone to the tour that an ant follows based upon the positional results of the tour that the ant follows. In order to make these ant colony algorithms competitive, edges swapping is commonly used as a local search technique. In [16], for example, 2-opt and 3-opt are utilized as the local search operators.

The practical importance of edge swapping is further supported by the well known Lin-Kernighan (LK) algorithm which is widely accepted as the most efficient algorithm know to date for the TSP. The LK algorithm is in fact a clever implementation of edge swapping. It differs from standard K-Opt in that the K is variable and is determined dynamically during execution [12]. Invented in 1971, it's long standing as the king of TSP algorithms is testament to the power of edge swapping techniques in solving TSP problems efficiently.

The Critical Role of Edge Swapping Techniques

Now that we the importance of K-Opt edge swapping to the TSP has been established, we have chosen to focus on the details of K-Opt rather than introducing some other general heuristic. It is likely that any such heuristic that we may devise would benefit from K-Opt. So we choose to focus on the techniques which are used most prevalently in order to increase the impact of our

findings. With this in mind, we introduce a new technique to improve upon K-Opt.

Within any edge-swapping algorithm whether 2-opt, 3-opt, or LK, after the edges have been cut, there will be some number of ways to reconstruct the tour using replacement sets of edges. The new technique, which we introduce in chapter three, attempts to reduce the number of solutions that must be analyzed during program execution. The basis of this technique is in a new type of set that we introduce and call an edge cut equivalence set. These sets allow for exhaustive K-Opt to be applied to more K-Sets of edges without exhaustively applying the heuristic to all possible tour reconstructions. We describe the new technique in chapter three. In addition, we include empirical results demonstrating the effectiveness of the new technique. Finally, we establish future directions of research related to this topic and propose frameworks for utilizing the technique in a complete algorithm for solving the TSP. We begin now by completing our background information on existing techniques before proceeding to chapter three.

K-Opt Edge Swapping

We now review the existing K-Opt techniques. By covering 2-Opt, and 3-Opt, and giving some examples, we

prepare the way for chapter three, which introduces our new technique for improving K-Opt.

The goal of any edge-swapping algorithm is to take an existing tour and improve on it by swapping some number of edges. Popular implementations are 2-Opt and 3-Opt [10]. The basic concept of edge swapping can be seen best by examining 2-Opt, which is the simplest of all edge swapping algorithms.

2-Opt Edge Swapping

As seen in figure 1, a 2-opt move simply involves cutting 2 edges from the existing tour and replacing them with 2 new edges. Figure 2 illustrates a 2-opt move. In this case, the change in tour cost can be computed by subtracting the cost of the edges that have been cut and adding the cost of the new edges. The improvement then, is given by:

$$(A:B)+(C:D)-(A:C)-(B:D)$$

Where (A:B) denotes the edge from A to B. 2-Opt implementations generally compare all possible pairs of edges until a positive gain is achieved. Note that there are N*(N-1) or approximately N^2 such pairings. When all pairs of edges have been checked for possible improvements, we say that the tour is 2-optimal.

A tour is 2-Optimal if there is no better tour which can be constructed by swapping out any 2 edges from the existing tour.

That is, there are no two edges that can be removed from the tour so that a reconstruction will produce a better tour.

Adding to the popularity of 2-Opt, is the ease in which it can be implemented. A small code section showing the ease of implementation of 2-Opt is shown as figure 3. 2-Opt has proven to be a very effective algorithm for efficiently producing tours of good quality [10], [16], [1]. It is highly susceptible, however, to local optima, which can prevent it from finding tours of better quality. As can easily be inferred, any tour in which more than 2 edges must be swapped to produce a better solution is a local optima for 2 opt. To try to minimize the effects of local optima, the obvious next step in the process of swapping edges is to proceed from 2-Opt to 3-Opt and even higher. The general case then is K-Opt in which K edges are swapped. The higher the value of K, the less likely we will encounter a local optima. The extreme case is K=N which is tantamount to exhaustive search in which the optimum is guaranteed. Having stated the

specific case for 2-optimality above, we now state the general case.

> A tour is K-Optimal if there is no better tour which can be constructed by swapping out any K edges from the existing tour.

It is conjectured that for relatively small K, say K=10, a K-Optimal tour is highly likely to be the optimal tour in most cases. Unfortunately, as K increases, the complexity of an implementation increases greatly and it's efficiency decreases drastically. As mentioned, for 2-opt, there are N*(N-1) or approximately N^2 pairings of edges that must be checked. In the general case, for K, there are approximately N^K sets of K edges that must be checked. In addition to the increase in the number of edge sets that must be checked, the number of possible tour reconstructions also increases. In 2-Opt, there is only one way to reconstruct the tour. The number of possible reconstructions increases exponentially in terms of K. Due to the large number of K-sets and the large number of reconstructions, higher order K opt has not been heavily researched.

Increasing K - An Example from 3-Opt

To see the effects on tour reconstructions when increasing K, we now explain 3-Opt and show that the number of possible tour reconstructions begins to increase with K. Of course the underlying principle of 3-Opt is to cut 3 edges and then replace them with 3 new edges. In 2-Opt there was only 1 way to reconnect the tour. With 3-Opt, however, the case is more complicated as shown in figure 4.

As can be seen, in figure 4, with 3-Opt there are at least 4 ways to reconstruct the tour. In fact, figure 2 shows only the tour reconstructions, which do not use any of the original edges in the edge cut set. If we allow the edges in the edge cut set to be used in the reconstruction, there would have been 7 ways to reconstruct the tour. This does not count the case that uses all of the edges in the edge cut set, as this would simply reproduce the original tour. In addition to this increase in the number of reconstructions, we now have N^3 sets of 3 edges that can be cut. These facts lead the implementation to become much more cumbersome and the efficiency of the algorithm is drastically reduced. Even 3-Opt, however, is very manageable and easy to visualize. But when we move into higher order K-Opt, the situation becomes less manageable. As shown in table 1, the number

of possible tour reconstructions increases exponentially as a function of K.

This exponential growth rate makes higher order K-Opt impractical. As an example, Complete 8-Opt would require approximately N^8*645,119 tours to be inspected. While these higher order K-Opt algorithms lead to better tour quality, the improvements come at a high cost in extremely long run times. Our objective then is to reduce the number of solutions within K-Opt that must be analyzed. By leveraging an observed characteristic of the tour reconstructions we have devised a technique which can greatly reduce the number of solutions needed to be explored. We now introduce the technique by discussing edge cut equivalence sets, which are central to the process.

CHAPTER THREE

Improving K-Opt with Edge Cut Equivalence Sets

We have established the importance of K-Opt as related to the TSP and have provided background information on K-Opt techniques. We now move forward into our new research which seeks to improve upon K-Opt by reducing the number of solutions which must be analyzed. Chapter three begins by introducing the concept of edge cut equivalence sets which are central to the new technique. It continues by illustrating how edge cut equivalence sets can be used in an algorithm to search for improved tours. An algorithm is established for the entire process and some empirical results from an implementation are included to show proof of concept.

Edge Cut Equivalence Sets

In order to make high order K-Opt more practical, we seek to reduce the number of solutions required to be explored. To meet this objective, we now introduce the concept of edge cut equivalence sets. Given a set of reconstructions, say S, for an instance of K-Opt, we can partition set S into a group of non-intersecting subsets which we call edge cut equivalence sets. These sets are defined from the perspective of a single cut within the K-Opt instance. If we identify one of the edges being cut

as A:B then we may define an edge cut equivalence set EC as

EC(A,B,C,D) = the set of all solutions x in S such that A:C is in x and B:D is in x.

Stated simply, an edge cut equivalence set with respect to edge cut A:B is the subset of all possible tour reconstructions in which vertices A and B are connected to the same vertices. That is, we do not care about the remaining shape of the reconstruction, but the shape of the reconstruction with respect to A:B is the same for all members of the edge cut equivalence set. Figure 5 helps to illustrate and explain the properties of edge cut equivalence sets further. Before discussing the properties and advantages to partitioning the set of reconstructions into edge cut equivalence sets, we first give an example using 5-Opt to help clarify the concept of edge cut equivalence sets.

A 5-Opt Edge Cut Equivalence Set Example

The best way to gain an understanding of edge cut equivalence sets is to see an example using some value for K. If we choose K=5 (5 opt), there are a total of 383 possible tour reconstructions. The process of partitioning these 383 solutions into edge cut equivalence sets begins by selecting one of the 5 edge

cuts as the reference point. From our discussion above, this is the edge cut from A to B. With A and B defined, we group all of the 383 reconstructions into sets based upon the vertices which A and B are connected to. In the case of 5 opt, this results in 43 edge cut equivalence sets. To illustrate this, figure 6 shows diagrams of one of the 43 resulting edge cut equivalence sets.

As shown, in figure 6, there are 8 ways to reconstruct the tour such that vertex A is connected to vertex C and vertex B is connected to vertex D. There are no other ways to reconstruct the tour so that edges (A:C) and (B:D) are included. In a similar fashion, the remaining 42 edge cut equivalence sets can be constructed for 5 opt. The key concept to note is that the only portion of the tour reconstruction that decides which edge cut equivalence set the solution falls into is which vertices A and B are connected to. The remaining shape of the reconstruction is irrelevant to the partitioning process. In fact, this is clearly shown in the pervious diagram as the shape of the 8 members of the edge cut set shown differ greatly.

Using Edge Cut Equivalence Sets to Minimize the Number of Reconstructions Analyzed

Now that the properties of edge cut equivalence sets have been defined, we begin to discuss how they can be used to minimize the number of tour reconstructions that

we must analyze for a K opt algorithm. As we have established, an edge cut equivalence set EC(A,B,C,D) contains all tour reconstructions which include edges (A:C) and (B:D). We will now establish that by selecting a different pair of vertices for A and B, say A' and B' respectively, we can guarantee that no better solutions exist within the edge cut equivalence set by inspecting only 1 tour reconstruction. We call this move, where we select new values for A and B a slide move.

The Slide Move

Given an edge cut equivalence set EC(A,B,C,D), a slide move finds a new position for the reference edge cut being considered. This implies that a slide move applied to edge cut set EC(A,B,C,D) will result in some new edge cut set EC(A',B',C,D). This can be seen by noting that all solutions in EC(A,B,C,D) will have all of the same edges as solutions in EC(A',B',C,D) with the following excpetions:

- Edge A:B was not part of the original solution but is part of the new solution.
- Edge A':B' was part of the original solution but is not part of the new solution.
- Edges A:C and B:D were part of the original solution but are replaced by edges A':C and B':D in the new solution.

Pictorially, the slide move is depicted as figure 7.

The reason that we introduce the slide move is that the move alters all members of the edge cut equivalence set by a constant value. To see this, consider the effects of a slide from A:B to A':B' within the edge cut set EC(A,B,C,D). We have established that all members of EC(A,B,C,D) will include edges (A:C) and (B:D). By sliding A:B to A':B', we create a new edge cut set EC(A',B',C,D). All members of this new set EC(A',B',C,D) will contain edges (A':C) and (B':D). Note that the only difference between any member T of EC(A,B,C,D) and T' of EC(A',B',C,D) is the removal of edges (A:C), (B:D), (A',B') and the addition of edges (A':C), (B':D), and (A,B). So the effect of the slide move from A:B to A':B' on all members of the edge cut equivalence set is:

(A':C)+(B':D)+(A':B') - (A:C)-(B:D)-(A,B)

We can now use this information to our advantage by reducing the number of reconstructions that we need to analyze. The following rule, which we refer to as the Slide Move Theorem, is the fundamental property that we will exploit in our new technique:

Slide Move Theorem:

Let P be the best tour reconstruction of edge cut set EC(A,B,C,D). Then after a slide move, P must

still be the best possible tour reconstruction in EC(A',B',C,D).

Proof:

Let P be the best tour reconstruction of EC(A,B,C,D).

Let R be any other tour reconstruction in EC(A,B,C,D).
Since, P is the best tour reconstruction of EC(A,B,C,D), we know that, C(P) <= C(R). (Where C(P) denotes the tour cost of P)

Let P' be the solution P after the slide move and let R' be the solution R after a slide move to EC(A',B',C,D)

We seek to show that C(P')<=C(R').

After executing the slide move to EC(A',B',C,D), P and R will be adjusted by some constant, say V. This follows based on the properties of an edge cut equivalence set. In particular,

$$V=(A':C)+(B':D)+(A:B)-(A:C)-(B:D)-(A':B')$$

So, we have C(P')=C(P)+V, and C(R')=C(R)+V.

Finally, since C(P)<=C(R), we know that C(P')<=C(R').

In effect, P which is the best solution within the edge cut equivalence set, acts as a lower bound within the edge cut equivalence set and we are guaranteed that we can execute any number of slide moves and P will still be the best solution in the set.

To see the benefits of this, consider table 2 which shows the number of possible tour reconstructions and the number of edge cut equivalence sets for different values of K. From this table, we can see that 8 opt, for example, has 645,119 possible tour reconstructions and only 157 edge cut equivalence sets. If we take an instance of the TSP and apply 8-opt, we can partition all of the 645,119 reconstructions into the 157 edge cut equivalence sets. Now, if we execute a slide move, we create an entirely new set of 645,119 tour reconstructions. If we know the best reconstruction within each of the 157 edge cut equivalence sets, however, we need only inspect these 157 best tours in order to guarantee that we have the best solution from among the 645,119 possibilities.

When we execute a slide move, we need only check one solution from each of the edge cut equivalence sets in order to cover all possible tour reconstructions. Based on table 2, we can compute the number of solutions that are actually covered every time we analyze a tour reconstruction as:

(number of possible reconstructions) / (number of edge cut equivalence sets).

The augmented table showing this for various values of K is shown as table 3.

To further appreciate the improvement, we can consider the improvement as a percentage. To do this, we take the number of reconstructions that must be analyzed, subtract the number of solutions that must be analyzed with the new method and divide by the original number of solutions that must be analyzed. For example, with 6 opt we get

(3839-73)/3839 = 98.099 % improvement

Of course the percent improvement increases dramatically as we increase K. the full table showing the percent improvement is given as table 4.

Empirical Results Using 8-Opt

To demonstrate the potential of this technique and to establish that it works in practice, we have implemented the method and applied it to 8 opt. The algorithm implemented works as follows:

-Create an initial solution at random.

-For some number of iterations

Select a random set of 8 edges to be cut

Explore all 645,119 solutions and mark the best of each edge cut equivalence set

Slide a pre-selected edge-cut to as many positions as possible. Tracking the best solution of each edge cut set

-End For

Table 5 shows empirical data gathered for some fairly large instances from the TSPLIB. As expected, the implementation using the edge cut equivalence sets returns tours of higher quality while requiring very modest increases in run time. Also, as expected, the benefits of using the equivalence set method are increased when the problem size grows.

This is due to the increase in the number of slide moves executed.

This data merely confirms the established fact that the edge cut equivalence sets can be used to find solutions to K-Opt without using exhaustive reconstruction within the K-Opt algorithm. The intent at this time is not to create a complete TSP solver, but rather to clearly show the effectiveness of the new technique.

The Implementation

The current implementation consists of 2 parts. The first part is a program, which produces a data file containing all possible tour reconstruction patterns. This program is the source for all of the cardinal values given in the paper for number of possible tour reconstructions and number of edge cut equivalence sets for a particular value of K. The second part of the implementation is the program that reads in the problems instance and applies the new technique to it using the reconstructions provided by the first program. Source code for both modules is included with some narrative about the implementation.

Future Directions

While the current implementation establishes that the technique is effective in finding solutions without

having to exhaustively check all possibilities, it is not intended to be a general-purpose TSP solver. As a next step, the technique needs to be encapsulated in a general algorithm for solving the TSP so that direct comparisons can be made against other algorithms. Two main approaches are possible. First, an exhaustive K-opt implementation can be explored in which all possible K sets of edges are checked. To implement this, it will be important to try to maximize the number of slide moves that can be made so that the smallest number of solutions needs to be analyzed. A second approach would involve some heuristic, which seeks to find suitable edges to be cut rather than to exhaustively try all possible K sets. One such possibility is an alpha nearness function, which has been shown to be effective in selecting edges that are good candidates for removal. Many of the general-purpose techniques stated in the introduction such as genetics, ant colonies, and simulated annealing may be used effectively to find edges to cut also.

Issues for an Exhaustive K-Opt Approach

As mentioned, it has been conjectured that a K-optimal tour is likely to be globally optimal for a relatively small K. Also, as mentioned, the unfortunate realization is that existing techniques to produce a guaranteed K optimal tour for K of any substantial size,

say 10, are currently not practical for large problem instances. One use of the new technique discussed in this paper, however, would be to try to implement an exhaustive K-Opt algorithm efficiently. Again, this would be done for a value of K such that an optimal solution would be likely. The key to this approach lies in maximizing the number of slide moves and minimizing the number of transitional moves where a new set of edges are selected as the edge cut set. This is an obvious objective since it is the slide move that yields efficient results allowing a very large number of solutions to be dismissed by analyzing very few possible solutions.

An exhaustive K-Opt algorithm must consider all possible sets of K edges as candidates to be cut. Of course there are N choose K such sets. The simplest approach, and in fact the one that is most often used in practice, is to use a lexicographical ordering to generate all possible K sets of edges to be cut. The question that arises , then is how many slide moves versus transition moves are required by this approach. Further, the goal is to produce an efficient algorithm that will produce a maximum number of slide moves and minimize the number of transitional moves. These are non trivial problems and require additional research for acceptable answers. It is hoped that an algorithm that

maximizes the number of slide moves can be combined with the technique discussed in this paper to provide for a highly effective TSP solver.

Issues for a Heuristic Based Approach

A heuristic approach would involve some method of selecting the candidate sets of edges to be cut. Fortunately, many existing algorithms can be easily adapted or even used as is with the edge cut equivalence set method. A perfect example of this is the genetic family of algorithms. At some point within its operation, a genetic algorithm seeks to takes two existing solutions and combine them in some way to produce a new, hopefully improved, solution. This generally involves a heuristic that selects many of the edges from the parents and passes them on to the new child. Regardless of the actual heuristic used, this will often result in a child with some number of missing edges. That is to say the heuristic will generally not create a complete solution. Rather, it will create a partial solution using the edges of the parents. Some *repair* algorithm must then be applied to this partial solution in order to fill in the missing edges and produce a complete solution. This, of course, is the perfect opportunity to utilize the edge cut equivalence set technique introduced in this paper. Since the genetic crossover heuristics for the TSP often

yield partial solutions, the new technique can be used to fill in the missing edges. This is consistent with the general goal of the genetic algorithm which is to capitalize on the strengths of the parents while attempting to find some improvement in the child.

Other general heuristics can also be adapted to be used with the new technique. In ant colony algorithms for the TSP, for example, a number of ants are distributed throughout the graph and each ant searches for a solution. After each iteration, the best ants deposit pheromone to their edges making them attractive to ants in future iterations. While similar to genetics in general theme, ant systems differ in their implementation. At no time, does the algorithm produce a partial solution. During each iteration, the ants each produce a complete solution making an additional step necessary to utilize the edge cut equivalence set technique. In practice, however, an extra step is often taken with ant systems as those implementations that report good results utilize a local optimizer in the form of 2 opt, 3 opt or LK. In likewise fashion, the edge cut equivalence set technique can be used in combination with an ant colony algorithm for the TSP. The local optimizer is simply replaced with an edge cut equivalence set routine. Thus, the general heuristic approach should offer excellent prospects for future research since some

existing heuristics can be used almost as is while others can be used with minor adaptations as discussed.

CHAPTER FOUR

Reference Notes

This chapter gives a brief exposition of the key references as they relate to this topic. This should prove useful information for guiding future research related to the new techniques discussed in this paper. By annotating the material which led to the creation of the edge cut equivalence set method, we hope to focus future research on the references that are most applicable for meeting the research objective.

General Combinatorial Algorithms

As a general introduction to combinatorial algorithms, [11] "Combinatorial Algorithms", gives much useful information. Its division of the subject into three main sections generation, enumeration and search provides for an excellent framework for studying the subject. While it's direct focus on the TSP is limited, it does briefly discuss the two-opt edge swapping technique which is fundamental to the techniques in this paper. In addition, its overall discussion of combinatorial search provides useful background information for pursuing topics such as the TSP.

For mathematical concepts related to computing and combinatorial algorithms, [9] "Concrete Math", is recommended. This text provides an excellent mathematical

foundation for working with combinatorial algorithms. Of particular interest and directly related to future directions involved with this paper, are the chapters related to recurrence relations. As mentioned, finding closed forms for the growth rates of the edge cut equivalence sets would be very useful. The techniques in this reference could prove useful in finding such closed forms.

The Traveling Salesman Problem

[2] "Finding Tours in the TSP" and [10] "The Traveling Salesman Problem: A Case Study in Local Optimization" are two excellent references for the TSP and prevailing solutions. Both papers discuss edge swapping and general heuristics and add support to the viewpoint presented in this paper that some type of edge swapping as a local optimizer is a key factor in making general heuristics perform well. In particular, [2] gives an excellent discussion of Lin-Kernighan. The LK algorithm is beautifully simple when properly presented and [2] offers perhaps the clearest explanation of the algorithm. Theoretical analysis is rarely provided with research on the TSP. In [10], however, a brief discussion of theoretical complexity of the TSP is given. Also provided is a good discussion of the very important Held-

Karp lower bound which is often used as a performance key when the optimal tour length is unknown.

As a source of additional references on the TSP, [13], the TSPBib provides an up to date categorized collection of TSP related references. The TSPBib proved invaluable as a time saver in searching for reference material related to particular areas of interest during the completion of this thesis.

Edge Swapping

Edge swapping is discussed heavily in most of the references but [10], provides the most comprehensive coverage. In addition to giving a solid explanation of 2-opt and 3-opt, they are discussed in sufficient detail to familiarize the reader with their inner workings. Theoretical and experimental results are provided as well as a discussion on techniques for improving 2-opt and 3-opt performance. A discussion is provided of techniques that reduce the number of comparisons required by using nearest neighbor techniques and so called-don't look bits. This is of particular interest since it bears the same goal as edge cut equivalence sets. Its approach however is quite different as it does not guarantee to consider all possible tour reconstructions, as is the case with edge cut equivalence sets.

APPENDIX: TABLES AND FIGURES

Figure 1: The Importance of Edge Swapping

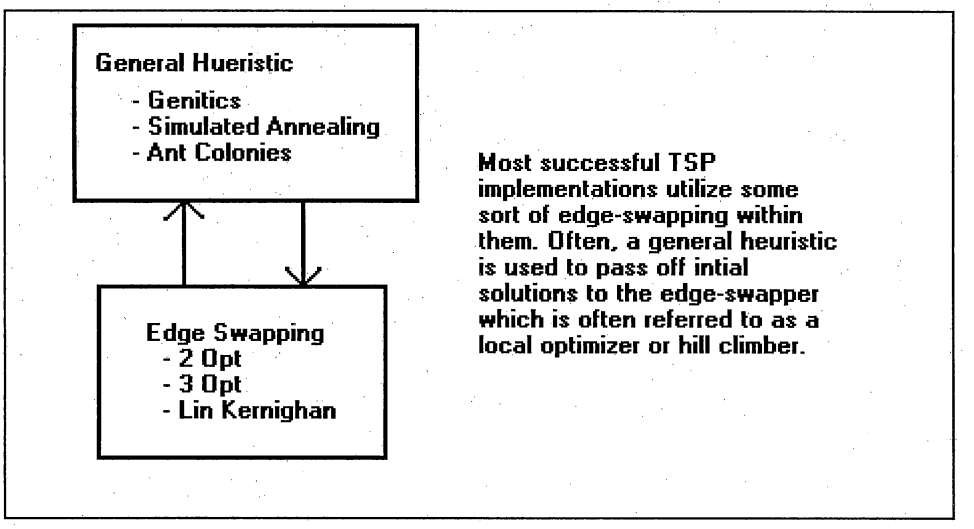


Figure 2: A Simple 2-Opt Move

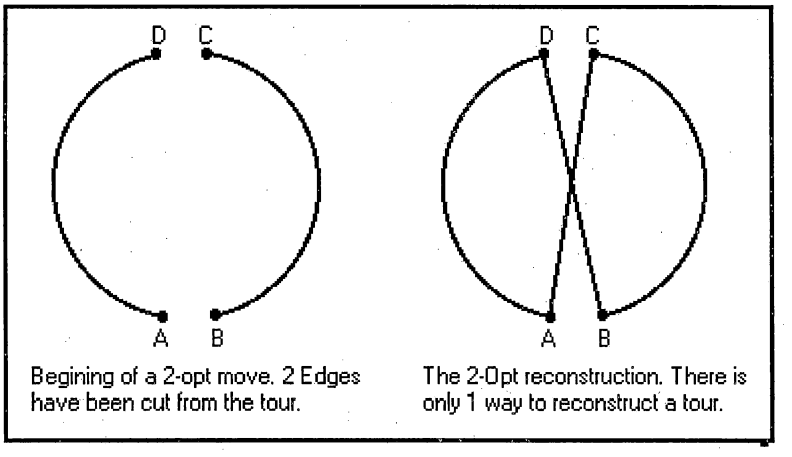
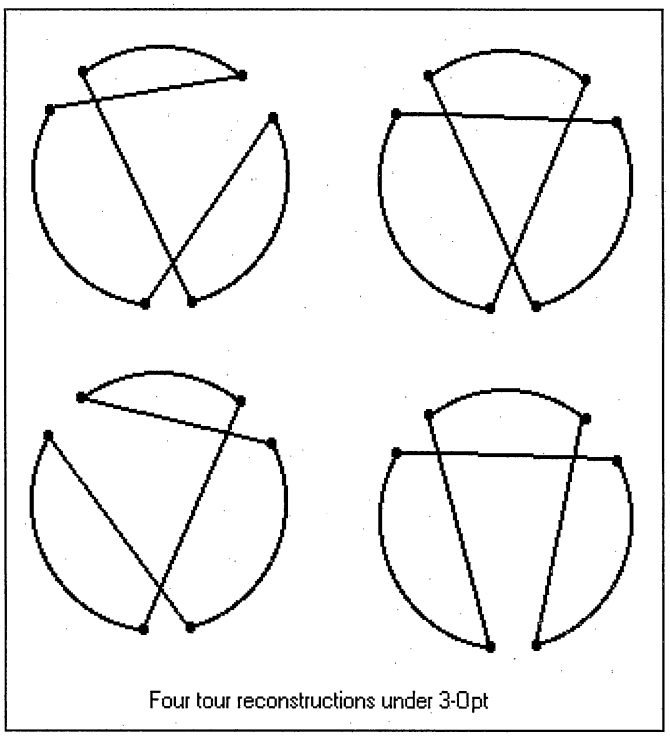


Figure 3: 2-Opt C++ Code

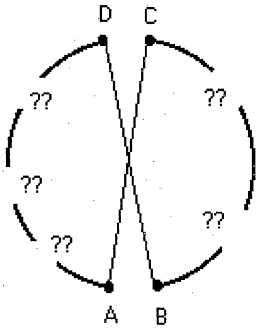
```
bool CTSPSolution::TwoOpt()
{
  int iTemp;
  bool bImproved=false;
  int x1,x2,y1,y2; //these are the positions
  int a1,a2,b1,b2; //these are the vertices
restart:
  int iGain=1;      //set to 1 to get started
  while (iGain>0)
  {
    for (x1=0;x1<iDimension-2;x1++)
    {
      y1=x1+1;
      a1=aiSolution[x1];
      b1=aiSolution[y1];
      for (x2=y1+1;x2<iDimension-1;x2++)
      {
        y2=x2+1;
        a2=aiSolution[x2];
        b2=aiSolution[y2];
        iGain=*(tspData->aiEdgeWeight+iDimension*a1+b1)+
            *(tspData->aiEdgeWeight + iDimension*a2+b2)
          -*(tspData->aiEdgeWeight + iDimension*a1+a2)
          -*(tspData->aiEdgeWeight + iDimension*b1+b2);

        if (iGain>0)
        {
          //we have a gain, make the move and start over
          //to see if there are any new  2-opt moves
         //possible
          //to make the move, we reverse the tour from y1
          //x2
          bImproved=true;
          while (y1<x2)
          {
            iTemp=aiSolution[y1];
            aiSolution[y1]=aiSolution[x2];
            aiSolution[x2]=iTemp;
            x2--;
            y1++;
          }
          iCost-=iGain;
          goto restart;
        }
      }
    }
  }
  return bImproved;
```

Figure 4: 3-Opt Tour Reconstruction



Four tour reconstructions under 3-Opt

Figure 5: An Edge Cut Equivalence Set



An edge cut equivalence set EC(A,B,C,D) is the subset of all reconstrctions such that edge A:C and edge B:D are both in the reconstruction. We do not care about the reconnections in the remaining part of the reconstruction. In this example, we have 7 edges being cut. There 46,079 ways to reconstruct the tour. We partition these 46,079 solutions into edge cut equivalance sets with repect to A:B by creating an edge cut equivalence set for all possible values of C and D. In the case of 7 cuts, this will result in 111 edge cut equivalence sets.

Figure 6: A 5-Opt Edge Cut Equivalence Set

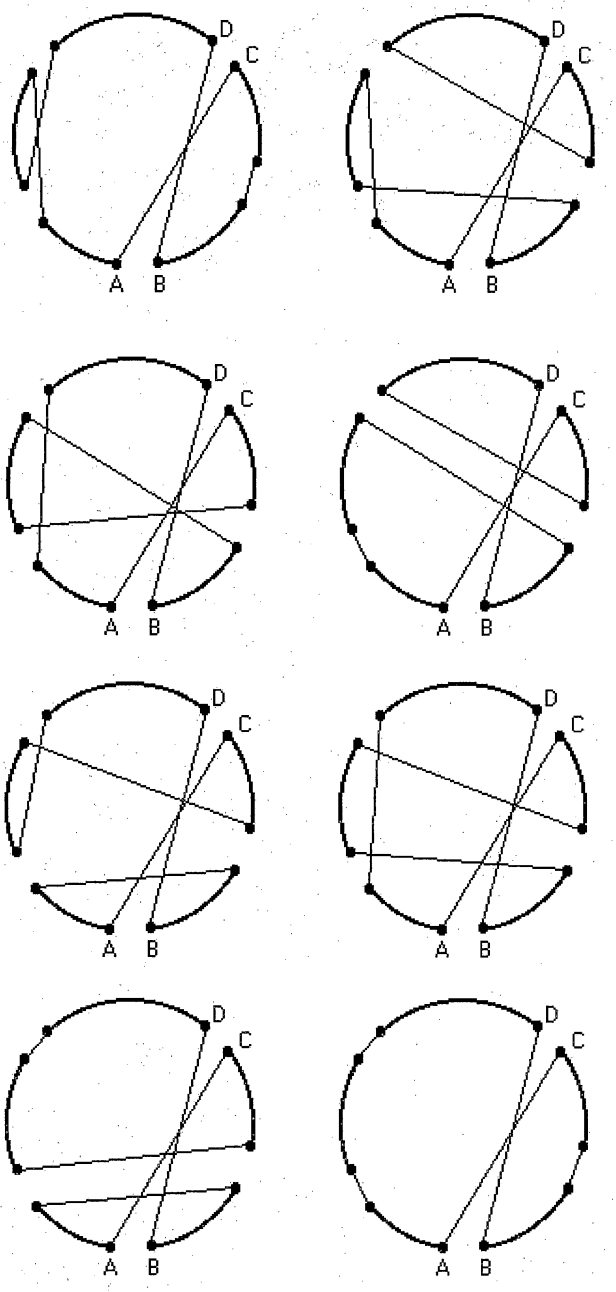


Figure 7: The Slide Move

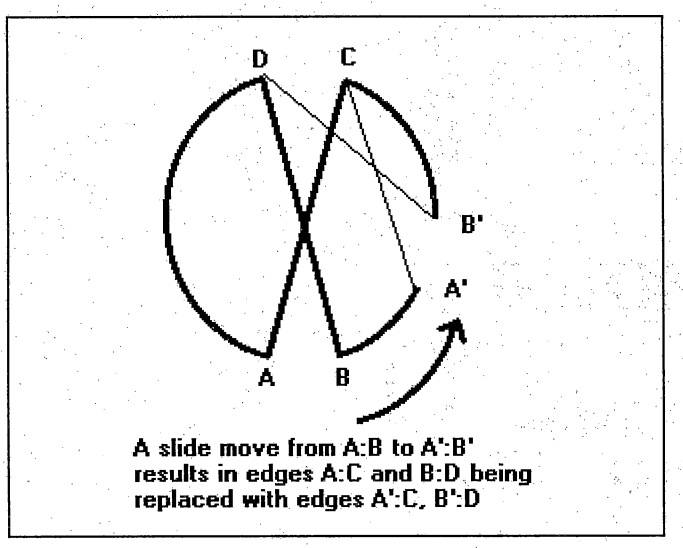


Table 1: Number of K-Opt Reconstructions

K - Number of Edges Cut	Number of Reconstructions Possible
3	7
4	47
5	383
6	3,839
7	46,079
8	645,119
9	10,321,919
10	185,794,559

Table 2: Number of K-Opt Edge Cut Equivalence Sets

K - Number of Edges Cut	Number of Reconstructions Possible	Number of Edge Cut Equivalence Sets
3	7	7
4	47	21
5	383	43
6	3,839	73
7	46,079	111
8	645,119	157
9	10,321,919	211
10	185,794,559	273

Table 3: Number of Solutions Per Analysis

K - Number of Edges Cut	Number of Reconstructions Possible	Number of Edge Cut Equivalence Sets	Number of Solutions Per each Analysis
3	7	7	1
4	47	21	2
5	383	43	5
6	3,839	73	53
7	46,079	111	415
8	645,119	157	4,109
9	10,321,919	211	48,919
10	185,794,559	273	680,567

Table 4: Percent Improvement Using Edge Cut Sets

K - Number of Edges Cut	Number of Reconstructions Possible	Number of Edge Cut Equivalence Sets	Number of Solutions Per each Analysis	Percent Improvement
3	7	7	1	0.000
4	47	21	2	55.319
5	383	43	5	88.773
6	3,839	73	53	98.099
7	46,079	111	415	99.759
8	645,119	157	4,109	99.976
9	10,321,919	211	48,919	99.998
10	185,794,559	273	680,567	99.999

Table 5: Empirical Results

Problem Name/ Number of Nodes	Optimum	Without Sets	With Sets
d198	15,780	19,252 (346 seconds)	17,728 (353 seconds) (194 Improvements by Sets)
d657	48,912	149,831 (346 seconds)	107,635 (359 seconds) (483 improvements by sets)
d1291	50,801	383,704 (349 seconds)	307,418 (370 seconds) 524 via sets
d1655	62,128	549,857 (350 seconds)	455,987 (375 seconds) 507 via sets

REFERENCES

[1] E.H.L. Aarts, J.K. Lenstra: "Local Search in Combinatorial Optimization (Wiley-Interscience Series in Discrete Mathematics and Optimization) , June 1997

[2] D. Applegate, R. Bixby, V. Chvatal, W. Cook, "Finding Tours in The TSP", Center for Research in Parallel Computing (1999).

[3] B. Bixby, G. Reinelt, "The TSPLib", A Library of TSP Problems, http://nhse.cs.rice.edu/softlib/catalog/tsplib.html

[4] B. Bullnheimer, R Hartl., C. Straub: "A New Rank Based Version of the Ant System - A Computational Study", *Central European Journal for Operations Research and Economics,* 7(1):25-38, 1999.

[5] A Colorni., M.Dorigo, F.Maffioli, V. Maniezzo, G. Righini, M. Trubian (1996). Heuristics from Nature for Hard Combinatorial Problems. *International Transactions in Operational Research,* 3(1):1-21.

[6] M., Dorigo, V. Maniezzo and , A. Colorni: "Ant Systems: Optimization by a Colony of Cooperating Agents", IEEE Transactions on Systems, Man, and Cybernetics 26 (1), 1996, 29-41

[7] M. Dorigo, , L. Gambardella: "Ant Colony System: A Cooperative Learning Approach to the Traveling Salesman Problem", IEEE Transactions on Evolutionary Computation, Vol.1 No. 1, 1997

[8] B. Freisleben and P. Merz, "A Genetic Local Search Algorithm for Solving the Symmetric and Asymmetric Traveling Salesman Problem", *Proceedings of the 1996 IEEE International Conference on Evolutionary Computation,* Nagoya, Japan, pp. 616-621, 1996.

[9] Graham, Knuth, Patashnik, "Concrete Math", Addison-Wesley (1989).

[10] D. Johnson, L. McGeoch, "The Traveling Salesman Problem: A Case Study in Local Optimization", To be a chapter in the book Local Search in Combinatorial Optimization, edited by EHL Aarts and JK Lenstra.

[11] D. Kreher, D. Stinson, "Combinatorial Algorithms", CRC Press (1999).

[12] S. Lin and B. W. Kernighan, "An Effective Heuristic Algorithm for the Traveling Salesman Problem", Operations Research 21 (1973) 498-516.

[13] P. Macato, "TSPBib", Comprehensive TSP Bibliography, http://www.densis.fee.unicamp.br/~moscato/TSPBIB_home.html

[14] P. Merz and B. Freisleben, "Genetic Local Search for the TSP: New Results" ,1997 IEEE International Conference on Evolutionary Computation, IEEE Press, pp. 159-164, Indianapolis, USA, 1997.

[15] G. Reinalt, "TSPLib-A traveling salesman problem library", ORSA. J. Comput. 3(1991), 376-384.

[16] Stützle T. and M. Dorigo (1999). "ACO Algorithms for the Traveling Salesman Problem", In K. Miettinen, M. Makela, P. Neittaanmaki, J. Periaux, editors, *Evolutionary Algorithms in Engineering and Computer Science*, Wiley, 1999.