

**Development of New Hueristics for the
Euclidean Traveling Salesman Problem**

**By Thurman W. Tunnell
and Lenwood Heath**

TR 89-30

**DEVELOPMENT OF NEW HEURISTICS
FOR THE EUCLIDEAN TRAVELING SALESMAN PROBLEM**

by

Thurman W. Tunnell

Project submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of
Master of Science
in
Computer Science in Applications

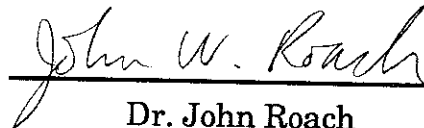
APPROVED:



Dr. Lenwood Heath



Dr. Donald Allison



Dr. John Roach

September 1989

Blacksburg, Virginia

DEVELOPMENT OF NEW HEURISTICS
FOR THE EUCLIDEAN TRAVELING SALESMAN PROBLEM

by

Thurman W. Tunnell

Lenwood Heath, Computer Science

(ABSTRACT)

Many heuristics have been developed to approximate optimal tours for the Euclidean Traveling Salesman Problem (ETSP). While much progress has been made, there are few quick heuristics which consistently produce tours within 4% of the optimal solution.

This project examines a few of the well known heuristics and introduces two improvements, *MaxDiff* and *Checks*. Most algorithms, during tour construction, add a city to the subtour because the city best satisfies some criterion. *MaxDiff*, applied to an algorithm, ranks a city according to its effect (based on the algorithm's criterion) if it is not added to the subtour.

The checks evaluate the subtour during tour construction. After each city is added to the subtour, the subtour is examined to detect inefficiencies in the subtour. If a possible improvement is detected, then a change is made in the tour. Although checks require some time, the goal is to improve the tour with as little cost as possible.

The tests were performed on five 100 city problems and five 500 city problems. The checks consistently decreased the tour length with a 20% to 90% increase in time. *MaxDiff* was particularly successful in the 500 city problems; all three heuristics to which *MaxDiff* was applied resulted in a decrease in tour length and a decrease in time for all five problems.

ACKNOWLEDGEMENTS

I would like to thank Dr. Donald Allison and Dr. John Roach for serving on my committee and for their support and helpful suggestions. I owe a special thanks to my advisor, Dr. Lenwood Heath, whose advice has been invaluable throughout the development of this project.

I would also like to thank Jannae Tunnell and Ed Wilson for their comments on my first drafts, and for letting me use their computer during all hours of the night. I thank my mother, Lane Tunnell, for her emotional and financial support. I also thank Matt Zukoski and many other computer science students at Virginia Tech who have given me invaluable Macintosh programming advice.

During the early stages of this project, the encouragement, patience, and love of Christiane Jung inspired many of my ideas, including MaxDiff and Checks. It is to her I dedicate this project.

TABLE OF CONTENTS

1.0 Introduction	1
2.0 Existing Heuristics for the ETSP	5
2.1 Terms and Definitions	6
2.1.1 The Convex Hull	7
2.2 Nearest Neighbor	8
2.3 Nearest Insertion	10
2.3 Cheapest Insertion	11
2.5 Convex Hull (CH) Cheapest Insertion	12
2.6 Stewart's Algorithm	13
2.7 Simulated Annealing	14
3.0 MaxDiff	15
4.0 Checks	21
4.1 Check1	22
4.2 Check2	24
4.3 Check3	26
4.4 Check4	30
4.5 Check5	32
4.6 Check6	35

5.0 Results and Analysis	39
5.1 Analysis of Checks	46
5.2 Analysis of MaxDiff	48
5.3 Analysis of Computation Times	49
6.0 Conclusion	56
6.1 Further Research	59
List of References	61
Appendix	63

1.0 INTRODUCTION

The Traveling Salesman Problem (TSP) is a well known and well studied problem in the area of combinatorial optimization [Lawler, Lenstra, Rinnooy Kan, and Shmoys, 1985]. The description of the problem is easily stated :

A traveling salesman wants to visit n cities, each city once, and then return home to his starting city. The problem is to find the shortest route. The distance between each pair of cities is given.

This research concentrates on a special case of the TSP, the Euclidean Traveling Salesman Problem (ETSP) where the distance between two cities is calculated from the locations (e.g. the x and y coordinates) of the cities. Thus, the ETSP is also restricted to two dimensional Euclidean space, whereas the TSP is not.

Although the ETSP is very simple to state, finding an optimal tour is difficult. The ETSP is a known NP-hard problem [Garey and Johnson, 1979]; therefore, it has no efficient algorithm unless $P = NP$. Presently, the only way to be assured that the shortest route has been found is to try all possible tours, which is very expensive. For example, when n is only 20, a 100 mip (million instructions per second) machine would literally take centuries to find all possible tours.

Since it is usually impractical to find the best tour, many heuristics have been developed to approximate the optimal tour lengths. The methods have been quite diverse, each with some successes and failures. In this project, a few of these methods are explored, and new heuristics are developed by modifying some of these existing algorithms.

The distinction between tour construction procedures and tour improvement procedures should be noted. Tour construction heuristics build a tour from a set of points. Tour improvement procedures modify an already constructed tour to obtain, hopefully, a better tour. Thus, one method builds a tour and the other improves it. In this paper, only tour construction heuristics are discussed.

This project proposes two ideas, each applicable to many existing tour construction heuristics. The first idea is **MaxDiff**, a heuristic that can be used in conjunction with various algorithms including the tour construction heuristics Cheapest Insertion, Convex Hull (CH) Cheapest Insertion, and Stewart's Convex Hull Insertion procedure. MaxDiff is not a complete heuristic by itself but is rather a concept which we can apply to a variety of existing TSP algorithms.

The second idea is the use of **Checks**. Checks are motivated by common problems with the tours constructed by convex hull procedures. There are six checks presented here, where each one attempts to detect a particular problem and resolve it. Checks are made during tour construction after each city is added to the subtour. None of the checks used in this project increase the overall time complexity of the algorithms to which they are applied.

Extensive testing was performed on a suite of five 100 city problems and a suite of five 500 city problems. The results of using MaxDiff and the checks have been encouraging. One of the more successful algorithms used in ETSP tour construction is the *convex hull insertion procedure* developed by W.R. Stewart [Golden and Stewart, 1985]. Our research has produced many heuristics which perform better than Stewart's algorithm. Although many of these heuristics require more computing time, none of them on the average require double the time of Stewart's algorithm.

MaxDiff performed especially well on the 500 city problems. MaxDiff produced shorter tours, and in less time, than the original algorithms to which MaxDiff was applied. In the 100 city problems, the success of MaxDiff is less certain, but some good results were obtained.

Every check added to an existing heuristic resulted in either the same tour length or in a shorter tour. The checks were particularly successful when used in combination with MaxDiff, or with other checks. The most successful checks were also the most expensive.

This research justifies further research in checks and MaxDiff. If an approximate tour is needed for a large set of cities using Stewart's algorithm or the CH (convex hull) cheapest insertion algorithm, then we can definitely recommend that MaxDiff be applied since a shorter tour and less time is very likely. If distance is more important than time, then checks should also be used. Recommendations of particular heuristics are discussed later.

The existing heuristics used in this research are each separately defined and discussed in section 2.0. Following, MaxDiff and its

applications are discussed in section 3.0. In section 4.0, each check is defined separately, with examples demonstrating their usefulness. The test results are reported and analyzed in section 5.0, and conclusions about MaxDiff and checks are then drawn in section 6.0. The best tours found for each test case in this study are listed in the appendix.

2.0 EXISTING HEURISTICS FOR THE ETSP

Many heuristics have been developed for the ETSP with a variety of trade offs between the quality of a tour and time efficiency. In this paper, a few fast heuristics that produce good tours are considered. The nearest neighbor and nearest insertion algorithms are included because they are very fast and certain conclusions can be drawn about the optimal tour length from the length of the tours obtained from these algorithms. The cheapest insertion algorithm is used in combination with MaxDiff. The CH cheapest insertion procedure and Stewart's algorithm are used with MaxDiff and the checks. Stewart's algorithm is also included because it is one of the most successful quick heuristics [Golden, Bodin, Doyle & Stewart, 1980]. In addition the simulated annealing heuristic is used. Although it is much slower than the other heuristics in this paper, it usually produces very good tours. Simulated annealing is especially useful in analyzing the results of the 500 city problems, because the optimal solutions are not known, and because it provides a good comparison measure.

These algorithms (other than simulated annealing) have basic characteristics common to all of them. Tour construction begins with an initial subtour, which could be, for example, the convex hull (explained below), an edge, or just a single city. Two lists are maintained during tour

construction; one list T contains all cities which compose the subtour, and the second list NT contains all the cities not yet placed in the tour (Noga, 1984). Based on some heuristic, one city in NT is chosen to be inserted into T ; this step continues until all cities are in T . In other words, cities in NT are added to the subtour, one at a time, until all cities in NT have been added and the tour is thus complete. Each of the heuristics begins with only a set of coordinates representing the location of the cities.

2.1 Terms and Definitions

Before describing each heuristic, certain terms need to be defined. Every algorithm has an *insertion criterion* and a *selection criterion* which determine where and what should be added next to the tour. The selection criterion decides which city will be added next to the tour. The insertion criterion determines where the city will be added. The city which satisfies the selection criterion the best is also sometimes said to *fit in* the best. In some of the heuristics (e.g. cheapest insertion, nearest neighbor), these criteria are the same; but many of the algorithms (e.g. nearest insertion, Stewart's, MaxDiff) have two separate criteria which determine where and what should be added.

Two common measures used in this paper are *dist* and *cost*.

$\text{dist}(a,b)$ = the Euclidean distance between city a and b .

$\text{cost}(a,b,c) = \text{dist}(a,b) + \text{dist}(b,c) - \text{dist}(a,c)$.

Cost is a very intuitive measure for the ETSP. Assuming (a,c) is an edge in the subtour and b is in NT , then $\text{cost}(a,b,c)$ is the distance lost in the subtour if b is added to T between edge (a,c) .

Most of the algorithms in this study add a city k to the subtour by inserting k between two adjacent cities in T , i and j . These letters i , k , and j are used consistently in this paper in this context. That is, k represents the last city added to the subtour between edge (i,j) .

Two functions commonly used are A and B . Assuming c is a city in T , then $A(c)$ is the city in T after c , and $B(c)$ is the city before c .

2.1.1 The Convex Hull

Most of the algorithms used in combination with MaxDiff and checks use the convex hull as the initial subtour. The convex hull is the shortest perimeter simple polygon which contains a set of points in a plane [Noga, 1985]. The convex hull can be easily visualized by stretching a rubber band around all points on a graph; the points which the rubber band touches are the vertices of the convex hull.

There are various advantages of using the convex hull as the initial subtour. According to one survey of TSP construction procedures, heuristics which do not employ the convex hull as the initial tour are "hard pressed" to find a TSP tour which is much better than 5% to 7% above the optimal solution [Golden et al, 1980]. One explanation for this is that it has been proven that for any set of cities, the original order of the cities which compose the convex hull remains the same for the optimal tour [Eilon, Watson, and Christofides, 1971]. For this paper, the method used to find the

convex hull is the Graham algorithm [1972]. Two ETSP heuristics that start with the convex hull are used in this project as a basis for applying our new techniques. These two heuristics are Stewart's [1977] algorithm and the Convex Hull Cheapest Insertion Procedure [Golden and Stewart, 1985].

2.2 Nearest Neighbor

The nearest neighbor algorithm :

```

choose an arbitrary city as the initial subtour;
while NT is not empty do begin
    {selection and insertion step}
    find the city closest to the last city added and add this city to the
    subtour;
end;
connect the last city to the first city;

```

The nearest neighbor algorithm [Bellmore and Nemhauser, 1968], [Golden, et al, 1979] is one of the simplest and quickest heuristics for the ETSP. It is also one of the least effective methods at finding a short tour. The nearest neighbor algorithm is of some value because it has been shown that its worst case behavior is :

$$\frac{\text{length of nearest neighbor tour}}{\text{length of optimal tour}} \leq 1/2 [\lg(n)] + 1/2$$

where n is the number of cities [Golden, et al, 1979].

There are other heuristics that have significantly better performance guarantees. The minimum spanning tree algorithm ($O(n^2)$) produces a

tour no more than twice the length of an optimal tour [Johnson and Papadimitriou, 1985]; and Christofides' algorithm ($O(n^3)$) produces a tour length that is always less than $3/2$ times the length of the optimal solution [Johnson and Papadimitriou, 1985].

The relative success of the nearest neighbor heuristic is often dependent on the city chosen as the starting node. In our tests, the nearest neighbor algorithm was run three times, starting with a random initial city each time. The best result (shortest tour) of the three runs is the solution reported.

The nearest neighbor procedure is computationally one of the quickest ETSP algorithms. After each city k is added to the subtour, all cities not yet in the subtour must be evaluated to see which city is the closest to k . Since there are $n-1$ cities added to the tour (after the initial random city is selected) and there are $(n - |\text{subtour}|)$ scans through the cities not yet in the tour after each city is added, the number of computations is proportional to the sum of $(n - |\text{subtour}|)$ as $|\text{subtour}|$ goes from 1 to $n-1$. Thus, the nearest neighbor algorithm requires $O(n^2)$ computations.

2.3 Nearest Insertion

The nearest insertion algorithm :

```

choose an arbitrary city p as the initial subtour;
find the city q closest to p and form the subtour p-q-p;
while NT is not empty do begin
  {selection step}
  find the city k in NT closest to any city in the subtour;
  {insertion step}
  insert city k in between adjacent cities (i,j) such that
  cost(i,k,j) is minimal;
end;

```

This procedure is a little more intricate than the nearest neighbor heuristic but has the same time complexity. After each city k is added to the subtour, each city p not in the subtour must be checked to see if k is closer to p than the city previously closest to p (stored in memory). This check is only one comparison, and thus each check is done in constant time. While each NT city p is checked, it can also be determined if p is the closest city to any subtour city. Thus, to find the NT city which is closest to any T city, only one pass through the NT cities is needed. To insert a city, one pass through T is needed. There is a total of n cities in the two lists NT and T, and $n-1$ cities must be inserted, resulting in a time complexity of $O(n^2)$.

Nearest insertion guarantees a tour length which is less than or equal to twice the length of the optimal tour [Johnson and Papadimitriou, 1985].

2.4 Cheapest Insertion

The cheapest insertion algorithm :

```

choose an arbitrary city p as the initial subtour;
find the city q closest to p and form the subtour p-q-p;
while NT is not empty do begin
  {selection step and insertion step}
  Minimize cost(i,k,j) for all adjacent cities (i,j) in T and k in NT;
  Insert city k between i and j;
end;

```

The usual programming steps of this procedure are to find the next city k to be inserted, and then update the NT cities. Of these two steps, the updating is the more costly. Updating is needed to determine which T edge each NT city fits in between the best. If p is in NT and its minimum edge was the (i,j) which k was just inserted between, then all edges in T must be checked to find a new minimum edge for p . In the worst possible case, all cities in NT would have (i,j) as their minimum edge, leading to $O(n^3)$ [Noga, 1984]. In most cases though, p 's minimum edge is not (i,j) , and therefore only the new edges (i,k) and (k,j) must be evaluated to update the minimum edge for p . Golden et. al. (1979) state the average time complexity is $O(n^2 \lg n)$.

2.5 Convex Hull (CH) Cheapest Insertion

The CH cheapest insertion algorithm is the same as the cheapest insertion procedure with the exception that the convex hull is the initial tour. This heuristic is especially significant in this paper because MaxDiff and many of the checks were conceived of with the CH cheapest insertion algorithm in mind.

The CH cheapest insertion algorithm :

```

T := the convex hull;
while NT is not empty do begin
  {insertion and selection step}
  Minimize cost(i,k,j) for all adjacent cities (i,j) in T and k in NT;
  Insert city k in between i and j;
end;

```

The computational complexity of this heuristic is identical to that of the cheapest insertion algorithm. For this research, the convex hull was calculated by the Graham algorithm [Graham, 1972] which has a worst case time complexity of $O(n \lg n)$ [Noga, 1984]. If most of the cities are on the convex hull, then the complexity is reduced; but in the average case, the complexity of CH cheapest insertion is $O(n^2 \lg n)$.

2.6 Stewart's Algorithm

Stewart's algorithm was originally called the *Convex hull insertion procedure* [Golden and Stewart, 1985], but because this name describes many of the existing heuristics for the ETSP, this paper refers to this algorithm simply as Stewart's algorithm. This heuristic is one of the most successful quick algorithms. In one study [Golden, et al, 1980] of various quick algorithms, Stewart's algorithm, on the average, had the shortest tours.

Stewart's algorithm :

```

T := the convex hull
while NT is not empty do begin
  for each city k in NT do begin {insertion step}
    Find (i,j) in T that minimizes cost(i,k,j);
    k.ratio := [dist(i,k) + dist(k,j)] / dist(i,j);
  end;
  select the k* in NT which minimizes k.ratio;
  Insert the selected city k* in between i and j;
end; {while}

```

This procedure is computationally the same as CH cheapest insertion with the exception that Stewart's algorithm must also calculate a ratio for each city. This doesn't effect the overall time complexity; the number of computations is $O(n^2 \lg n)$ [Golden, et al, 1980].

2.7 Simulated Annealing

We use the simulated annealing heuristic only to find short tours to compare to our results. For more information and implementation details, see Kirkpatrick, Gelatt, and Vecchi [1984], Skiscim and Golden [1983], and Cerny [1985].

Simulated annealing is analogous to the statistical mechanics process called annealing where a low energy state of a compound can be reached by heating the compound and then slowly lowering the temperature. The lowest energy state is analogous to the optimal solution in the TSP. Simulated annealing allows increases in the tour length with the hope that the increase will avoid a local minimum [Golden and Stewart, 1985].

Simulated annealing is computationally very expensive because it examines many tours to find a good solution.

3.0 MAXDIFF

Most of the quick algorithms used in the Euclidean Traveling Salesman Problem are based on a "greedy" approach. That is, based on some local selection criterion, the next city added to the subtour is the city which best satisfies the criterion. For example, in Stewart's algorithm, the next city to be added is the city which minimizes $(\text{dist}(i,k) + \text{dist}(k,j)) / \text{dist}(i,j)$.

MaxDiff can be thought of as a "non-greedy" approach. The basic idea of MaxDiff is to evaluate a city not yet placed in the subtour according to its effect if it is inserted or added at a place in the subtour other than the place which would best satisfy the selection criterion.

MaxDiff is not an algorithm for ETSP but is rather a method for modifying existing ETSP algorithms. Generalizing, though, the basic idea of MaxDiff is as follows :

```
Find the initial subtour; { e.g. the convex hull }
While NT is not empty do begin
  Find the two "best" places for city k in NT to be inserted in the
  subtour according to the algorithm's insertion criterion;
  k.1 := best place;
  k.2 := second best place;
end;
k* := the city k in NT which Maximizes the Difference between k.1
and k.2 according to the selection criterion;
Insert city k* in the subtour according to the selection criterion;
```

Applying MaxDiff to Stewart's algorithm :

```

T := the convex hull
while NT is not empty do begin
  For each city k in NT do begin
    {insertion step}
    find adjacent cities i1, j1 and adjacent cities i2,j2 in T such that
    cost(i1,k,j1) < cost(i2,k,j2) < cost(im,k,jm), where im,jm are all
    adjacent cities in T except i1, j1 and i2, j2;
    k.ratio1 := [dist(i1,k) + dist(k,j1)] / dist(i1,j1);
    k.ratio2 := [dist(i2,k) + dist(k,j2)] / dist(i2,j2);
    k.i1 := i1;
    k.j1 := j1;
  end;
  k* := the k which maximizes k.ratio2 - k.ratio1; {selection step}
  Insert k* in between k*.i1 and k*.j1;
end; {while}

```

Applying MaxDiff to the CH Cheapest Insertion algorithm :

```

T := the convex hull
while NT is not empty do begin
  For each city k in NT do begin
    {insertion step}
    find adjacent cities i1, j1 and adjacent cities i2,j2 in T such that
    cost(i1,k,j1) < cost(i2,k,j2) < cost(im,k,jm), where im,jm are all
    adjacent cities in T except i1, j1 and i2, j2;
    k.cost1 := cost(i1,k,j1);
    k.cost2 := cost(i2,k,j2);
    k.i1 := i1;
    k.j1 := j1;
  end;
  k* := the k which maximizes k.cost2 - k.cost1; {selection step}
  Insert city k* in between k*.i1 and k*.j1;
end; {while}

```

To clarify how MaxDiff works, and why one would want to employ it, consider a simple example. Figure 3.1 illustrates part of a subtour. In this example, we assume that all cities and edges not shown are inconsequential. Figure 3.1(a) shows the initial state. If the cheapest insertion method or Stewart's algorithm is used, cities e and f will be inserted first, and in that order (Figure 3.1(b)). Finally, city g will be added

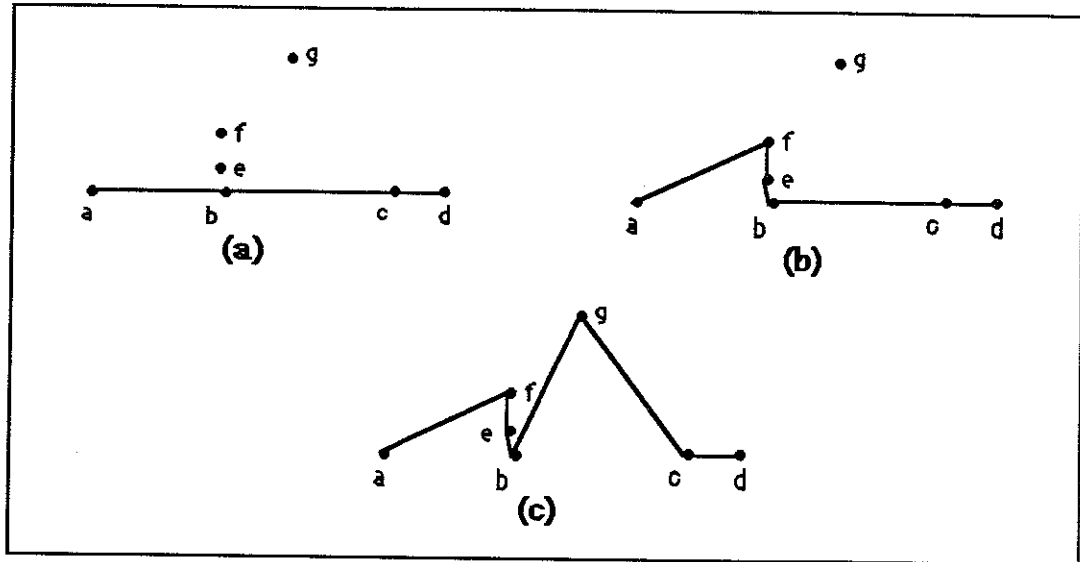


Figure 3.1. One problem with standard insertion heuristics.

(Figure 3.1(c)) between cities b and c according to the insertion criterion. As one can see, this is not the shortest tour possible.

If city g were inserted first, before e and f, then e and f would be inserted between g and b according to the insertion criterion, and a shorter tour would result. This is what MaxDiff does. Cities e and f fit in best between edge (a,b) and second best between edge (b,c). City g fits in best between edge (b,c) and edge (a,b), in that order. As can be seen, cities e and f fit in between b and c almost as well as they do in between a and b; but on the other hand, g fits in much better between b and c than it does between a and b. In other words, city g maximizes the difference between the two places which best satisfy the selection criterion for g. Thus, city g is inserted first (Figure 3.2(b)) and afterward cities e and f are inserted between b and g (Figure 3.2(c)).

MaxDiff is intuitively appealing. If a city k in NT can fit in between two distinct edges almost equally well, then there is little reason to add k now. Only when there is one edge with which k fits in well, should k be added to the subtour.

The basis of MaxDiff can possibly be seen more clearly from the stand point of the cities in NT. Imagine each city in NT competing with each other over which city will be added next to the subtour. Assume city p in NT fits in very well with edge1, and almost equally well with edge2 in the subtour. City q, on the other hand, fits in edge3 well, but fits in all other edges very poorly. It is the hypothesis of MaxDiff that q will have to be added eventually between edge3, and that if q is added now, then tour

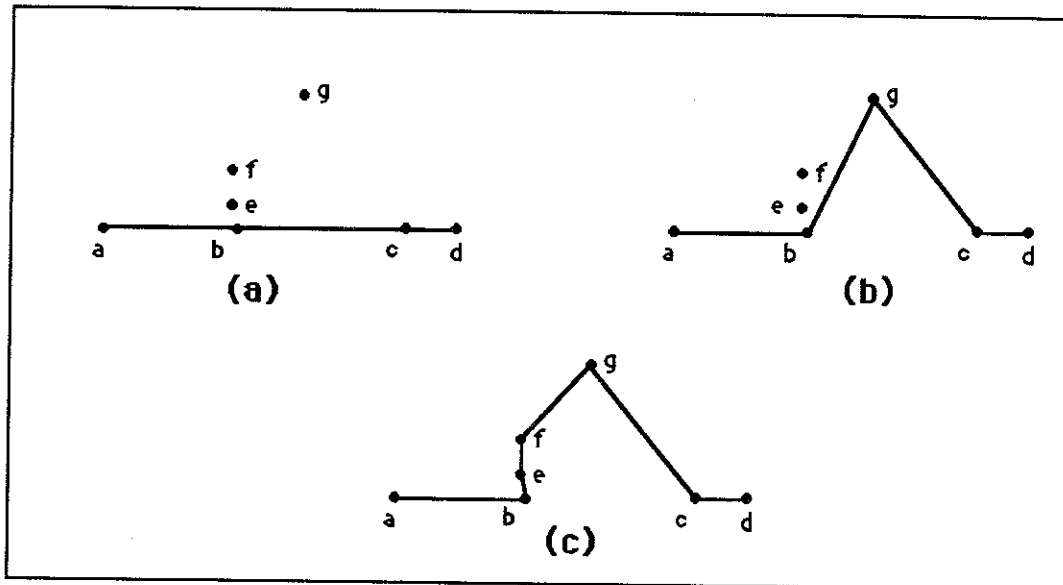


Figure 3.2. MaxDiff solution inserts city g first, which results in an optimal subtour.

construction is enhanced because the edges created by inserting q are used in the construction process.

In this project, MaxDiff is used only in combination with algorithms that determine the next city k to be added according to how well k fits in between two adjacent cities (an edge) in the tour. However, the concept of MaxDiff may be used in conjunction with other algorithms.

As can be seen by comparing the above MaxDiff algorithms with the original algorithms, the time complexities remain the same. MaxDiff only adds a few computations; that is, instead of having a pointer for every city in NT to the best insertion locations, MaxDiff requires two pointers. The most costly part of MaxDiff is the updating of NT after every city k is added to T . When k is added between i and j , the chances are doubled that city p in NT is pointing to edge (i,j) , thus increasing the update time. This issue was discussed previously in the description of the cheapest insertion algorithm.

4.0 CHECKS

Checks are heuristics that determine whether small changes made in the tour reduce the present length of the tour. Checks are made during tour construction, as opposed to tour improvement heuristics that make improvements on a completed tour. Checks are made after each city, other than a city in the initial subtour, is added to the tour. Typically, the number of checks made is equal to the total number of cities minus the number of cities in the initial tour. An exception occurs with *check1* because cities can be removed from T (discussed below), increasing the number of times a city is added to the tour, and thus increasing the number of checks.

Although checks obviously increase the computing time of the ETSP, none of the checks described in this paper actually increases the asymptotic time complexity of the algorithms with which they are used in this project. Below, the complexities of each check are described individually.

It should be noted that there are other checks which could easily be developed but are not covered by this project. The goal of this part of the project is to determine whether further research of checks might be promising.

4.1 Check1

Check1 has the greatest time complexity of the checks but is also one of the more successful checks. After each city k is added between two cities i and j , check1 searches for a city p in the tour such that p fits in better between i and k or k and j than between $A(p)$ and $B(p)$.

The algorithm is :

```

After  $k$  is inserted in  $T$  between cities  $i$  and  $j$  do
  for each city  $p$  in  $T$  (besides  $i, k$ , and  $j$ ) do
    if ( $\text{cost}(i, p, k) < \text{cost}(B(p), p, A(p))$ ) or
      ( $\text{cost}(k, p, j) < \text{cost}(B(p), p, A(p))$ ) then
      remove  $p$  from  $T$ .

```

The appeal of check1 is that cities in the tour can be removed (and reinserted later) if the city fits in better with a new edge than with its present position in the tour. For example, Figure 4.1 shows a situation when check1 would succeed. Figure 4.1(a) is the tour after city k is added to the tour. Check1 examines all cities in the tour; when city p is evaluated, the condition in the above algorithm succeeds. That is, $\text{cost}(i, p, k)$ is less than $\text{cost}(B(p), p, A(p))$; and thus, p is removed, as seen in Figure 4.1(b).

The time complexity of all executions of check1 is $O(n^2)$, assuming no more than n total cities are removed. A question of termination arises here since a city is removed and must be added later. If a city p were removed from T every time a city k was added to T , then the program would terminate. Although no proof is given here, by observation, it appears that

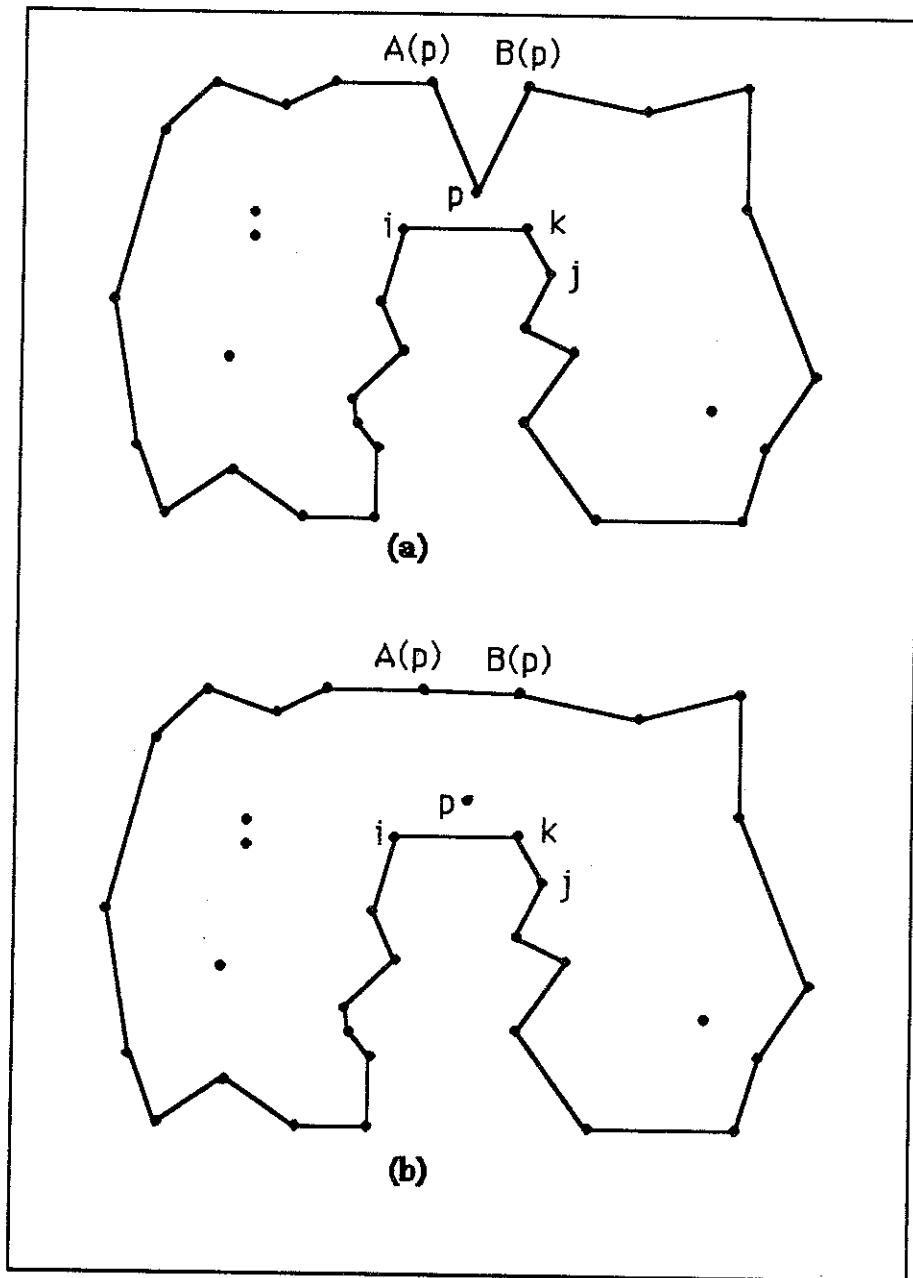


Figure 4.1. Check1 removes city p from subtour.

check1 rarely removes a city more than 5% of the time. Assuming this is true, the computing time of the algorithm (used with check1) is increased but the overall time complexity remains the same.

4.2 Check2

In check1, efficiency suffers whenever a city is removed from the subtour. This means that for every successful check in check1, a city is added twice to the tour, which obviously increases the computing time of the tour construction. Check2 differs from check1 only in that, instead of removing a city from the subtour, the city is put back in the tour between either i and k or k and j . Figure 4.2 illustrates the previous example before (Figure 4.2(a)) and after (Figure 4.2(b)) check2.

The algorithm of check2 :

```

After each  $k$  is inserted in the tour between cities  $i$  and  $j$  do
  for each city  $p$  in the tour (besides  $i, k$ , and  $j$ ) do
    if ( $\text{cost}(i, p, k) < \text{cost}(B(p), p, A(p))$ ) then begin
      remove  $p$  from  $T$ ;
      insert  $p$  between  $i$  and  $k$ ;
    end
    else if ( $\text{cost}(k, p, j) < \text{cost}(B(p), p, A(p))$ ) then begin
      remove  $p$  from  $T$ ;
      insert  $p$  between  $k$  and  $j$ ;
    end;

```

The time complexity of check2 is also $O(n^2)$. Check2 is a little more efficient than check1, because city p is repositioned in T , not added to NT .

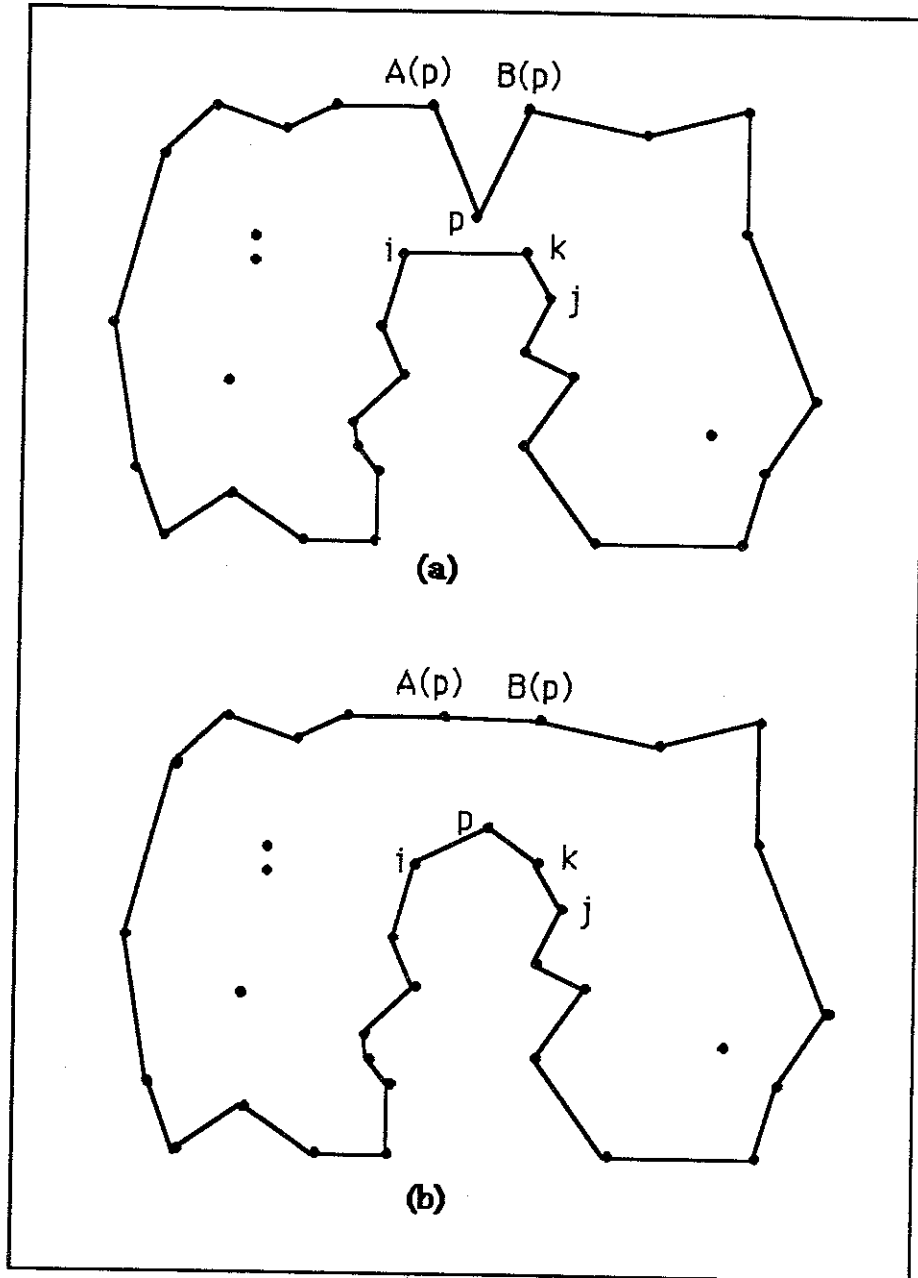


Figure 4.2. Check2 reinserts city p .

4.3 Check3

Because of the nature of most convex hull algorithms (e.g. cheapest cost, Stewart's), certain problems repeatedly arise. One common problem results when, during tour construction, part of the tour forms an hourglass shape. Figure 4.3(b) is an example of this shape. Figure 4.3(a) and 4.3(b) shows a tour as it is being constructed, to illustrate how this problem arises. The purpose of check3 is to find this type of situation and correct it. The result of using check3 on the problem in Figure 4.3(b) is shown in 4.3(c), which can easily be seen to be shorter than 4.3(b).

There are various ways that check3 could be programmed. The algorithm for check3 used in this project is very simple, but can miss some hourglass problems as described above. Two terms used in this check are *present cost* and *local*. The present cost of i is defined by the cost of i between the two cities adjacent to i in the tour; that is, the present cost of i equals $\text{cost}(A(i),i,B(i))$. Local means that only a certain number of edges are checked. Letting k be the last city inserted (between i and j), the basic approach of check3 is to see if there is any edge (p,q) local to i,k,j , where $\text{cost}(p,i,q)$ is less than the present cost of i or $\text{cost}(p,j,q)$ is less than the present cost of j .

The reason for only checking local edges is to avoid checking all edges in the tour. For this project, 12 edges were checked, 6 on each side of i and j respectively. This restriction is one way in which check3 could miss

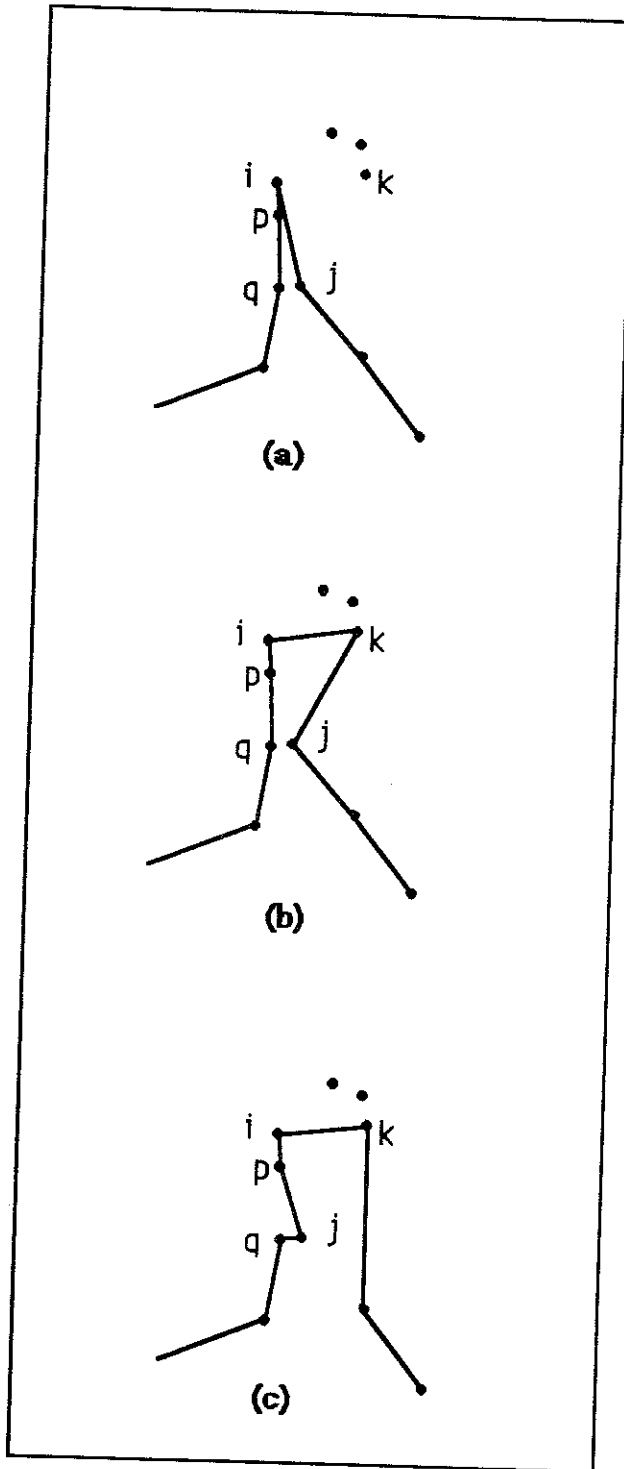


Figure 4.3. Check3 detects the hour glass shape in (b).

possible improvements. For example, in Figure 4.3, if there were seven cities between i and p , the edge (p,q) would have never been evaluated.

In more detail, the algorithm is :

```

After each  $k$  is inserted in  $T$  between  $i$  and  $j$  do begin
  for each of the six edges  $(p,q)$  in  $T$  before  $i$  do
    if  $\text{cost}(p,j,q) < \text{cost}(k,j, A(j))$  then begin
       $p^* := p;$ 
       $q^* := q;$ 
       $\text{found}_j := \text{true};$ 
    end;
  if not  $\text{found}_j$  then
    for the six edges  $(p,q)$  in the tour after  $j$  do
      if  $\text{cost}(p,i,q) < \text{cost}(B(i), i, k)$  then begin
         $p^* := p;$ 
         $q^* := q;$ 
         $\text{found}_i := \text{true};$ 
      end;
    end;
  end;
  if  $\text{found}_j$  then begin
    remove  $j$  from between  $k$  and  $A(j)$ ;
    insert  $j$  between  $p^*$  and  $q^*$ ;
  end
  else if  $\text{found}_i$  then begin
    remove  $i$  from between  $B(i)$  and  $k$ ;
    insert  $i$  between  $p^*$  and  $q^*$ ;
  end;

```

The problem with check3 is with respect to the restriction of evaluating only 12 edges. This does not seem to be a major concern, because by a study of different sets of random cities, one can observe that looking at six edges on either side of i or j is adequate for most cases. As the sets of cities increase, it is likely that the number of edges checked locally should also increase; although, by observation, most possible improvements of the nature of check3 appear to be detectable at an early stage.

Another and more serious problem with check3 lies in the fact that only one city, (i.e. i or j) is examined to be repositioned in the tour. In the example in Figure 4.3, if there were an additional city very close to j , the algorithm above would not detect the obvious improvement. If the algorithm checked for two cities to be repositioned in addition to checking for just one city, the improvement would then be detected; but the problem still remains because there could be any number of cities very close to j (or i) which would all have to be repositioned in the tour for an improvement (see Figure 4.4).

Check3 does 12 checks after every city is added to the subtour. Therefore, the number of computations required is $O(n)$.

4.4 Check4

Check4 is a very simple and quick check. As stated earlier, certain tour construction inefficiencies occur because of the nature of most convex hull algorithms. One situation which is easily improved is illustrated in Figure 4.5(b). Check4 examines two cities in the tour, the city before i and the city after j . If improvement is possible, check4 will reposition either one or both of these cities. Figure 4.5 illustrates an example where city $A(j)$ is repositioned for tour length improvement.

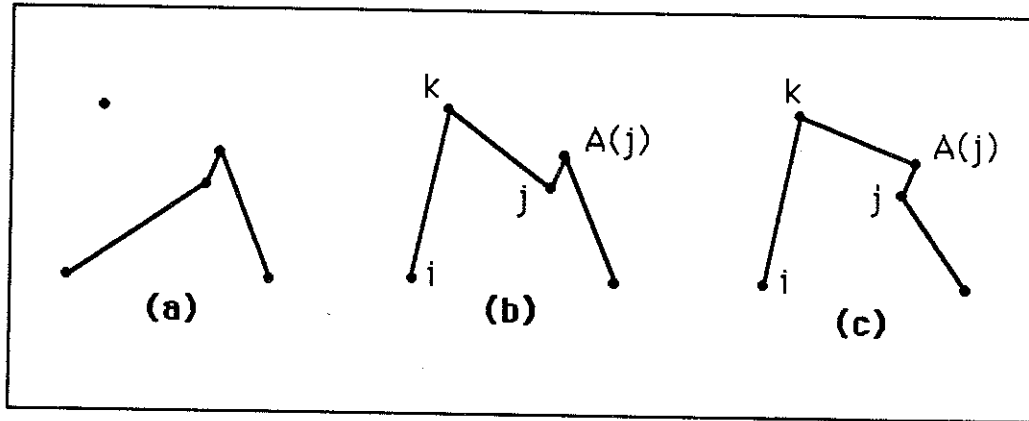


Figure 4.5. Check4 detects the problem in (b) and changes the subtour to (c).

The algorithm for check4 :

```

After each k is inserted in T between i and j do begin
  if cost(i, B(i), k) < present_cost(B(i)) then
    place B(i) in T between i and k;
  if cost(k, A(j), j) < present_cost(A(j)) then
    place A(j) in T between k and j;
end;

```

All cases similar to the example in Figure 4.5 are not detected by check4 for improvement. The weakness of check4 is analogous to the last problem mentioned for check3. That is, only one city (i.e. A(j) or B(i)) is evaluated to be repositioned in the tour. Thus, in Figure 4.5, if there were an additional city very close to A(j), the possible improvement would not be detected.

Check4 is computed in a constant amount of time and is called after every city is added to the subtour, resulting in a time complexity of $O(n)$.

4.5 Check5

Check5 detects the same possible improvements that check4 does and detects some of the situations that check4 misses as discussed above. The algorithm of check5 is more complicated and involved than the previously mentioned checks and, therefore, a longer explanation is needed.

Before discussing check5, the weakness of check4 discussed in the previous section should be further detailed so that the reason for check5 is clear. Figure 4.6(a) illustrates a situation where improvement is possible but is not detected by check4, because there is more than one city between i and B(c).

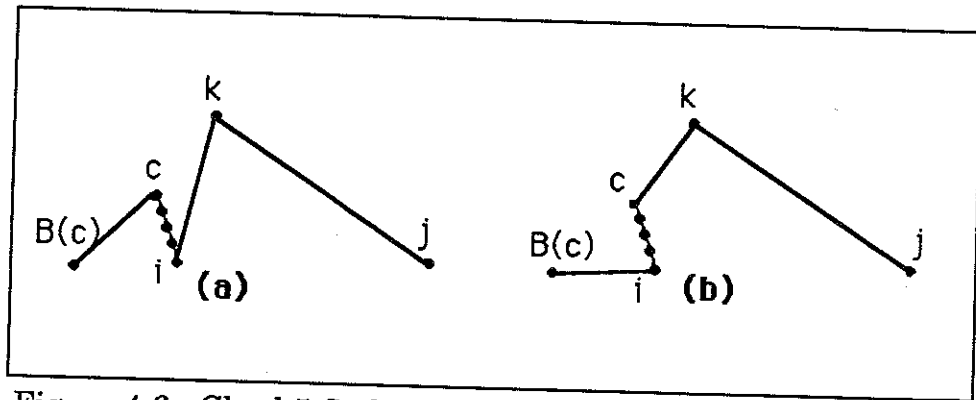


Figure 4.6. Check5 finds city c and improves the tour.

To improve on part of a tour such as in Figure 4.6(a), the city B(c) must be detected. One way to locate B(c) is to evaluate a certain number of cities on each side of i and j. There are two problems with this approach. One is that the number of cities to check is unsure; for example, in Figure 4.6, there could be a very large number of cities between i and c. The other problem is that excessive checking is done because, in most cases (i.e. after every city is added to the tour), no improvement is possible.

Check5 alleviates both of these problems but does not detect all cases similar to the example in Figure 4.6. This check is different from the previous checks in that some of the information used is stored information attained earlier as the tour was being built. When each city k is added to the tour between i and j, the $\text{cost}(i,k,j)$ is stored in association with k (e.g. usually the cost is stored in a record representing city k). This cost to add k is referenced by $k.\text{oldcost}$.

The algorithm of check5 :

```

After each k is inserted in the tour between i and j do begin
  c := i;
  while cost(i,B(c),k) < c.oldcost do
    c := B(c);

  if (c <> i) and (dist(B(c),i)+dist(c,k) < dist(B(c),c)+dist(i,k) then
    begin
      insert all edges from city i to city c between B(c) and k such that
      the new order in the tour is ... B(c), i, ... c, k...
    end;

  { now check for the edges after j }
  c := j;
  while cost(j,A(c),k) < c.oldcost do
    c := A(c);

  if (c <> i) and (dist(k,c)+dist(j,A(c)) < dist(A(c),c)+dist(k,j) then
    begin
      insert all edges from city j to city c between A(c) and k such that
      the new order in the tour is ... k, c, ... j, A(c)...
    end;
  end;

```

In the worst case, the 'while' loops terminate when a convex hull city is encountered since convex hull points have no 'oldcost'. This situation leads to a time complexity $O(n^2)$. In most cases, however, one can observe that the while statement will fail immediately. We speculate that check5 is usually done in constant time resulting in a time complexity of $O(n)$.

4.6 Check6

Like check5, check6 is more complicated than the previous checks, and the algorithm also uses previously stored information. Check6 looks for

situations for improvement similar to the nature of check1 and check2 with the exception that check6 looks at many cities instead of only one city at a time to be moved in the tour. For example, in Figure 4.7(a), the obvious improvement is to insert all cities between and including p and q in between i and k.

The difficulty is in determining the existence and location of cities p and q. One approach is to try every combination of adjacent cities in T. This, of course, would result in a very costly algorithm. Check6, however, discovers most possible improvements of this nature in a much quicker time by employing an edge list which lists all edges that once existed but no longer do. For example, when k is inserted between cities i and j, the now missing edge (i,j) is added to the edge list. Thus, when the last city is added to the tour, the number of edges in the edge list will equal [total number of cities - number of cities in the initial tour].

This edge list is used to find p and q. It is hypothesized that in most cases, if there is a group of cities between cities c1 and c2 that fit in between two other adjacent cities in the tour better, then (c1,c2) was once an edge; and therefore edge (c1,c2) would be in the edge list. In the example in Figure 4.7, using CH cheapest cost or Stewart's algorithm, it is easily seen that c1 and c2 must have formed an edge earlier during tour construction. This hypothesis is not proven here, but is only supported by the observation of many examples.

The edge list contains specific information for each edge: the two cities which make the edge (e.g. c1,c2), the cost of the edge (explained below in the algorithm), and a pointer to the next edge in the list.

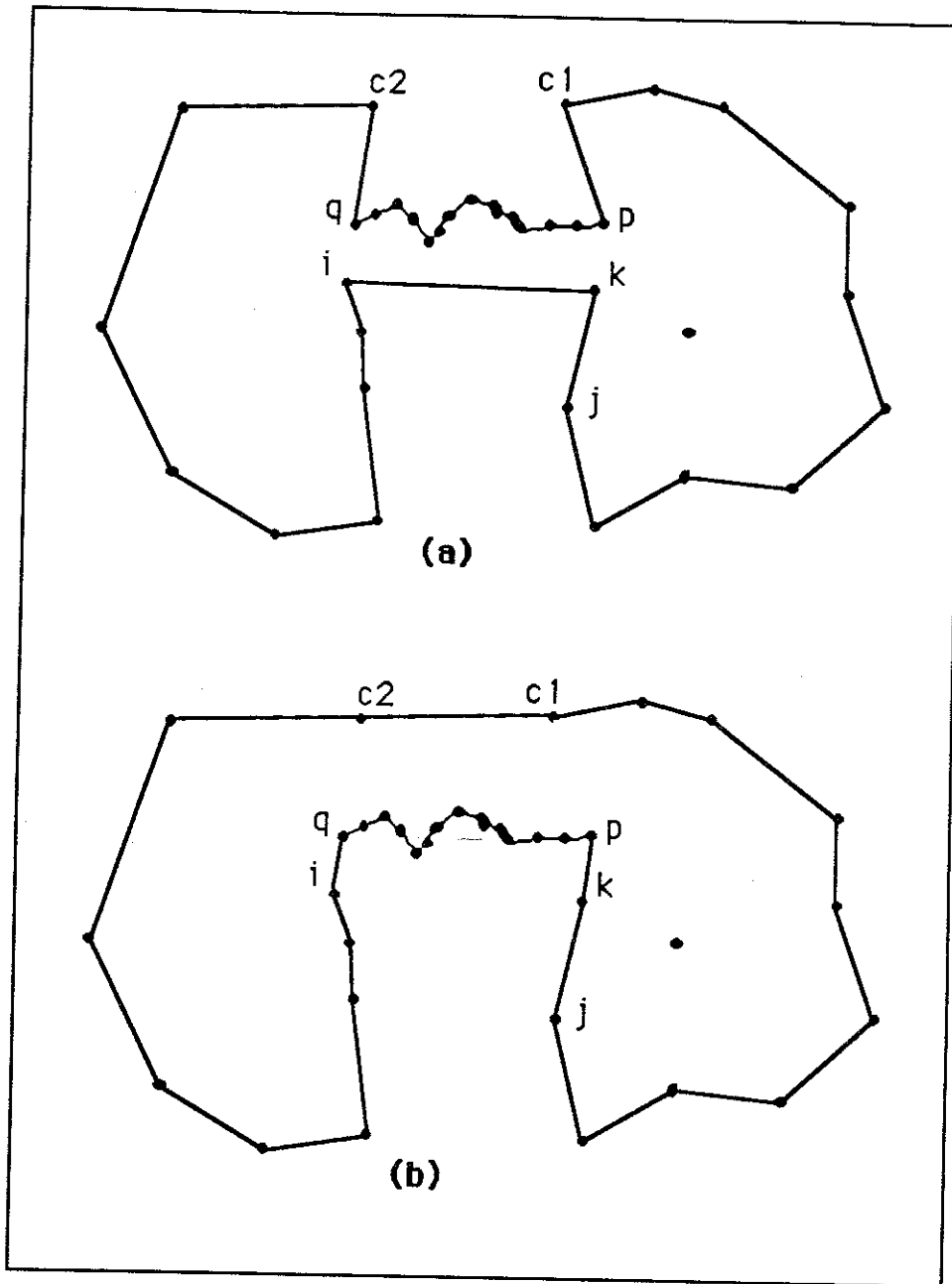


Figure 4.7. Check6 detects this improvement by employing an edge list, in which $(c1, c2)$ belongs.

In check6, we are looking for improvement only when i, k, j are not in the sequence $c1, p, \dots, q, c2$. Otherwise, check6 can find a p, q (as described above) and make a change in the tour which does not improve the tour length. The function *NotInSameTourPart* in the algorithm below checks to see if i, k, j are in the above sequence.

The algorithm for check6 :

```

After each  $k$  is inserted in the subtour between  $i$  and  $j$  do begin
  for each edge  $(c1,c2)$  in the edge list do begin
     $p := A(c1);$ 
     $q := B(c2);$ 
     $edge.cost := dist(c1,p) + dist(c2,q) -$ 
       $dist(c1,c2);$ 
    {check for improvement between edge  $(i,j)$ }
     $distance := dist(i,q) + dist(k,p) - dist(i,k);$ 

    if  $distance < edge.cost$  then begin
      if NotInSameTourPart then
        if this is the best improvement found so far then
          save edge;
    end
    else begin {now check for improvement between edge  $(k,j)$ }
       $distance := dist(k,q) + dist(j,p) - dist(k,j);$ 
      if  $distance < edge.cost$  then begin
        if NotInSameTourPart then
          if this is the best improvement found so far
            then save edge;
      end; {if}
    end; {else begin}
  end; {for each edge  $(c1,c2)$ ...}
end; {After each  $k$  ...}

if an improvement was found then begin
  reposition cities  $p$  to  $q$  in between  $i$  and  $k$  [or  $k$  and  $j$ ];
  remove saved edge from edge list;
  insert edge  $(i,k)$  [or edge  $(k,j)$ ] into the edge list;
end;
add edge  $(i,j)$  to the edge list;

```

To search through the edge list after every city is added to the subtour requires $O(n^2)$ computations.

5.0 RESULTS and ANALYSIS

All heuristics (except simulated annealing) were coded in Turbo Pascal and tested on the Macintosh SE/30. Five 100 city problems and five 500 city problems were the test data. The 100 city problems, reported as problems 24 to 28, were first presented by Krolak, Felts, and Marble [1971]. Optimal tours for these problems were proven by Crowder & Padberg [1980]. (There is a discrepancy for problem 25 concerning the optimal solution. The simulated annealing heuristic found a tour which is slightly shorter than the optimal tour reported by Crowder and Padberg.) This set of test problems has been the test data for numerous articles [Golden et al, 1980; Golden and Stewart, 1985; Norback & Love, 1977].

The 500 city problems were created especially for this project. A scale of 0 to 4000 for the x-axis and 0 to 2000 for the y-axis was used in keeping with the boundary of the 100 city problems. The five 500 city problems are reported as large1 - large5. The random number generator used was the RandomX function in Macintosh's Turbo Pascal; the algorithm for RandomX is:

$$\text{NewX} = (75 * \text{OldX}) \bmod (2^{31} - 1),$$

which is, according to a recent article, a good random number generator [Park & Miller, 1988].

The five principal algorithms used were nearest neighbor, nearest insertion, cheapest insertion, CH cheapest insertion, and Stewart's algorithm. Nearest neighbor and nearest insertion, both very quick algorithms, were used as a basis for comparison. Cheapest insertion is easily converted to a MaxDiff algorithm, but because of its nature, does not fit the style of most of the checks. For this reason, cheapest insertion was only combined with MaxDiff. All three of these algorithms reported the best of 3 runs, each run with a randomly generated starting point. The two convex hull algorithms, CH cheapest insertion and Stewart's algorithm, were used in conjunction with MaxDiff and all of the checks.

Certain combinations of the checks were also tried. The combinations reported were chosen by some pre-testing and according to which combinations appeared intuitively promising.

When checks are combined, certain programming problems arise. For example, assume a combination of checks 3, 5, and 6 are used. If check3 is successful and makes an improvement in the tour, then it has also consequently changed the ordering of i, k, j (where k is the last city inserted between tour edge (i, j)). Therefore, check5 and check6 can no longer look at both edges (i, k) and (k, j) because at least one of these edges no longer exists. In order to simplify the program, if one check is successful, then no more checks are attempted for that particular i, k, j .

The results are shown in Tables 5.1 - 5.4. Table 5.1 shows the solutions to the 100 city problems. The algorithms, which at least once, resulted in the best tour (the best tours are highlighted) of the tested problems were: Stewart's algorithm with check1 and with checks 3 and 6 combined; CH

Table 5.1 Solutions to the 100 city problems.

Algorithm	prob 24	prob 25	prob 26	prob 27	prob 28
Optimal	21282	22141	20749	21294	22068
Simulated Annealing	21285	22139	20770	21294	22163
Nearest Neighbor	26800	25997	24154	27820	26909
Nearest Insertion	25405	26874	25890	25007	26722
Cheapest Insertion	24419	25522	25262	24996	25361
+ MaxDiff	21527	22650	20820	21751	22290
CH Cheapest Insertion	23050	23247	21632	21712	22870
+ Check1	21877	23147	21526	21646	22827
+ Check2	22124	23147	21609	21646	22827
+ Check3	22286	22794	21278	21664	22611
+ Check4	22389	23114	21667	21712	22787
+ Check5	22131	23114	21526	21657	22837
+ Check6	21634	23037	21526	21646	22827
+ Checks 1 & 3	21836	22716	21128	21598	22768
+ Checks 3 & 6	21580	22716	21176	21609	22768
+ Checks 3, 5, & 6	21528	22716	21132	21598	22768
+ MaxDiff	21579	23049	20922	22395	22680
+ MaxDiff + Check1	21579	22437	20922	21898	22550
+ MaxDiff + Checks 1 & 3	21579	22437	21021	21886	22493
Stewart's Algorithm	22055	22700	21275	21794	22830
+ Check1	21481	22676	21016	21729	22809
+ Check2	21727	22676	21100	21729	22809
+ Check3	21848	22526	21023	21794	22528
+ Check4	21957	22689	21224	21794	22780
+ Check5	21589	22689	21014	21739	22830
+ Check6	21520	22576	21271	21729	22780
+ Checks 1 & 3	21481	22513	20923	21728	22528
+ Checks 3 & 6	21605	22395	20923	21728	22519
+ Checks 3, 5, & 6	21605	22395	20923	21739	22519
+ MaxDiff	22657	23178	21233	22205	23556
+ MaxDiff + Check1	21798	23098	20871	21919	22845
+ MaxDiff + Checks 1 & 3	21701	23192	20871	22109	22330

Table 5.2 Solutions as a percentage over the optimal solution.

Algorithm	prob24	prob25	prob26	prob27	prob28	avg
Optimal	21282	22141	20749	21294	22068	
Simulated Annealing	0.01%	-0.01%	0.10%	0.00%	0.43%	0.11%
Nearest Neighbor	25.93%	17.42%	16.41%	30.65%	21.94%	22.47%
Nearest Insertion	19.37%	21.38%	24.78%	17.44%	21.09%	20.81%
Cheapest Insertion	14.74%	15.27%	21.75%	17.39%	14.92%	16.81%
+ MaxDiff	1.15%	2.30%	0.34%	2.15%	1.01%	1.39%
CH Cheapest Insertion	8.31%	5.00%	4.26%	1.96%	3.63%	4.63%
+ Check1	2.80%	4.54%	3.74%	1.65%	3.44%	3.24%
+ Check2	3.96%	4.54%	4.14%	1.65%	3.44%	3.55%
+ Check3	4.72%	2.95%	2.55%	1.74%	2.46%	2.88%
+ Check4	5.20%	4.39%	4.42%	1.96%	3.26%	3.85%
+ Check5	3.99%	4.39%	3.74%	1.70%	3.48%	3.46%
+ Check6	1.65%	4.05%	3.74%	1.65%	3.44%	2.91%
+ Checks 1 & 3	2.60%	2.60%	1.83%	1.43%	3.17%	2.33%
+ Checks 3 & 6	1.40%	2.60%	2.06%	1.48%	3.17%	2.14%
+ Checks 3, 5, & 6	1.16%	2.60%	1.85%	1.43%	3.17%	2.04%
+ MaxDiff	1.40%	4.10%	0.83%	5.17%	2.77%	2.85%
+ MaxDiff + Check1	1.40%	1.34%	0.83%	2.84%	2.18%	1.72%
+ MaxDiff + Checks 1 & 3	1.40%	1.34%	1.31%	2.78%	1.93%	1.75%
Stewart's Algorithm	3.63%	2.52%	2.54%	2.35%	3.45%	2.90%
+ Check1	0.94%	2.42%	1.29%	2.04%	3.36%	2.01%
+ Check2	2.09%	2.42%	1.69%	2.04%	3.36%	2.32%
+ Check3	2.66%	1.74%	1.32%	2.35%	2.08%	2.03%
+ Check4	3.17%	2.48%	2.29%	2.35%	3.23%	2.70%
+ Check5	1.44%	2.48%	1.28%	2.09%	3.45%	2.15%
+ Check6	1.12%	1.96%	2.52%	2.04%	3.23%	2.17%
+ Checks 1 & 3	0.94%	1.68%	0.84%	2.04%	2.08%	1.52%
+ Checks 3 & 6	1.52%	1.15%	0.84%	2.04%	2.04%	1.52%
+ Checks 3, 5, & 6	1.52%	1.15%	0.84%	2.09%	2.04%	1.53%
+ MaxDiff	6.46%	4.68%	2.33%	4.28%	6.74%	4.90%
+ MaxDiff + Check1	2.42%	4.32%	0.59%	2.94%	3.52%	2.76%
+ MaxDiff + Checks 1 & 3	1.97%	4.75%	0.59%	3.83%	1.19%	2.46%

cheapest insertion with the combination check1 and check3 and the combination check3, check5, and check6; and Cheapest insertion with MaxDiff. The only ETSP heuristic to produce the best tour twice was MaxDiff applied to cheapest insertion. With the exception of simulated annealing, there was no one heuristic that performed exceptionally well for all five problems.

Table 5.2 gives the result of each heuristic for problems 24-28 as a percentage over the optimal. Cheapest Insertion with MaxDiff performed the best with an average of 1.39% over the optimal solutions; and Stewart's algorithm with checks 1 and 3 and checks 3 and 6, performed almost equally well with 1.52% over the optimal. Even though MaxDiff applied to cheapest insertion resulted in the shortest tours for the 100 city problems, these results are not guaranteed since for every starting point, a different tour could develop. Thus to guarantee the best tour produced by cheapest insertion plus MaxDiff, a problem must be executed n times, each time with a unique starting point.

Table 5.3 and 5.4 show the results of all five 500 city problems. Simulated annealing produced the best tours and its results were used as an approximation of the optimal solution. Although most of the ETSP heuristics using the convex hull as the initial tour produced tours within 3-7% above the optimal (best known tour length), none of the heuristics performed as well as they did with the 100 city problems. The methods tested which performed the best were Stewart's algorithm with MaxDiff and check1, and CH cheapest insertion with MaxDiff, check1, and check3. Stewart's algorithm with MaxDiff plus check1 had the best average

Table 5.3 Solutions to the 500 city problems.

Algorithm	large1	large2	large3	large4	large5
Best Known - S. Annealing	49253	47992	46412	48003	48080
Nearest Neighbor	61023	54897	58466	58741	57894
Nearest Insertion	59895	59788	57727	58620	58440
Cheapest Insertion	56462	57097	55630	56646	56898
+ MaxDiff	51681	50315	48868	49650	49335
CH Cheapest Insertion	55110	53185	52976	53934	54229
+ Check1	53717	52403	50751	51880	52863
+ Check2	53786	52720	51151	51878	53100
+ Check3	53144	51092	49967	50065	51426
+ Check4	54184	52665	51339	52298	53492
+ Check5	54070	52323	51094	52225	53204
+ Check6	53648	52392	51015	51843	52097
+ Checks 1 & 3	52356	51043	50050	49998	50413
+ Checks 3 & 6	52225	51411	50006	49715	51024
+ Checks 3, 5, & 6	52225	51411	50007	49747	51024
+ MaxDiff	51829	49569	48802	49693	50825
+ MaxDiff + Check1	51343	49382	48627	48839	50809
+ MaxDiff + Checks 1 & 3	51194	49259	48582	48768	50389
Stewart's Algorithm	53674	51712	49549	51208	51719
+ Check1	52063	49867	48366	50234	49830
+ Check2	52585	50184	48781	50337	50220
+ Check3	51951	50299	48794	50106	50188
+ Check4	52466	50894	49044	50767	50638
+ Check5	52676	50607	48852	50709	50576
+ Check6	52154	49502	48202	50140	49571
+ Checks 1 & 3	51547	49801	48358	49363	49697
+ Checks 3 & 6	51751	49408	47957	49719	50535
+ Checks 3, 5, & 6	51665	49352	47820	49368	50535
+ MaxDiff	52793	50920	49509	50335	49972
+ MaxDiff + Check1	50806	49264	48537	49498	49091
+ MaxDiff + Checks 1 & 3	50843	49527	48628	49775	49701

Table 5.4 Solutions as a percentage over the best known tour.

Algorithm	large1	large2	large3	large4	large5	avg
Best Known - S. Annealing	49253	47992	46412	48003	48080	
Nearest Neighbor	23.90%	14.39%	25.97%	22.37%	20.41%	21.41%
Nearest Insertion	21.61%	24.58%	24.38%	22.12%	21.55%	22.85%
Cheapest Insertion	14.64%	18.97%	19.86%	18.01%	18.34%	17.96%
+ MaxDiff	4.93%	4.84%	5.29%	3.43%	2.61%	4.22%
CH Cheapest Insertion	11.89%	10.82%	14.14%	12.36%	12.79%	12.40%
+ Check1	9.06%	9.19%	9.35%	8.08%	9.95%	9.13%
+ Check2	9.20%	9.85%	10.21%	8.07%	10.44%	9.56%
+ Check3	7.90%	6.46%	7.66%	4.30%	6.96%	6.65%
+ Check4	10.01%	9.74%	10.62%	8.95%	11.26%	10.11%
+ Check5	9.78%	9.02%	10.09%	8.80%	10.66%	9.67%
+ Check6	8.92%	9.17%	9.92%	8.00%	8.35%	8.87%
+ Checks 1 & 3	6.30%	6.36%	7.84%	4.16%	4.85%	5.90%
+ Checks 3 & 6	6.03%	7.12%	7.74%	3.57%	6.12%	6.12%
+ Checks 3, 5, & 6	6.03%	7.12%	7.75%	3.63%	6.12%	6.13%
+ MaxDiff	5.23%	3.29%	5.15%	3.52%	5.71%	4.58%
+ MaxDiff + Check1	4.24%	2.90%	4.77%	1.74%	5.68%	3.87%
+ MaxDiff + Checks 1 & 3	3.94%	2.64%	4.68%	1.59%	4.80%	3.53%
Stewart's Algorithm	8.98%	7.75%	6.76%	6.68%	7.57%	7.55%
+ Check1	5.71%	3.91%	4.21%	4.65%	3.64%	4.42%
+ Check2	6.77%	4.57%	5.10%	4.86%	4.45%	5.15%
+ Check3	5.48%	4.81%	5.13%	4.38%	4.38%	4.84%
+ Check4	6.52%	6.05%	5.67%	5.76%	5.32%	5.86%
+ Check5	6.95%	5.45%	5.26%	5.64%	5.19%	5.70%
+ Check6	5.89%	3.15%	3.86%	4.45%	3.10%	4.09%
+ Checks 1 & 3	4.66%	3.77%	4.19%	2.83%	3.36%	3.76%
+ Checks 3 & 6	5.07%	2.95%	3.33%	3.57%	5.11%	4.01%
+ Checks 3, 5, & 6	4.90%	2.83%	3.03%	2.84%	5.11%	3.74%
+ MaxDiff	7.19%	6.10%	6.67%	4.86%	3.94%	5.75%
+ MaxDiff + Check1	3.15%	2.65%	4.58%	3.11%	2.10%	3.12%
+ MaxDiff + Checks 1 & 3	3.23%	3.20%	4.77%	3.69%	3.37%	3.65%

percentage. It should be observed that the cheapest insertion and CH cheapest insertion algorithms performed especially badly with the larger sets of cities. However, cheapest insertion with MaxDiff is still competitive.

5.1 Analysis of Checks

For all checks tested, the tour lengths either improved or remained the same.

Check1 usually resulted in a noticeable improvement when employed with MaxDiff or any of the existing algorithms. It performed slightly better when used with Stewart's algorithm than with the CH cheapest insertion heuristic.

The algorithm for **check2** is the same as check1 with the exception that a city marked for improvement is reinserted back into the tour instead of removed from the tour (explained in the chapter on checks). The hope is that check2 will result in tour lengths comparable to check1 and also find the tours quicker. Although check2 performed as well as check1 in some of the problems, on the average, check1 resulted in shorter tours.

Check3 performed well with a consistent improvement in tour length with little additional computing time.

Check4 and **check5** were both designed to catch similar problems. Check4 is much simpler but check5 found more possible improvements. On the average, check5 usually performed 0.1 - 0.5% better than check4.

Check6, like checks 1 and 3, consistently produced good tours.

Some combinations of checks are more effective than others, and the success of these combinations depends on the algorithm with which they are used (i.e. Stewart's or CH cheapest insertion). The three combinations used in this project were checks 1 and 3, checks 3 and 6, and checks 3, 5, and 6.

All three combinations were successful to some extent. These combinations produced better results when used with Stewart's algorithm rather than with CH cheapest insertion. The combination of checks 3, 5, and 6 performed better than just the combination of checks 3 and 6 when used with Stewart's algorithm; but only a marginal difference between these two combinations occurred when used with CH cheapest insertion. On the average, all three combinations performed about the same. Combinations of checks 1 and 3, and checks 3, 5, and 6 each produce a tour which is approximately half the percentage over the optimal as is the percentage over the optimal for an algorithm without checks.

The behavior of check3 added to check1 together with MaxDiff is interesting. In problems large1 - large5, there is a noticeable improvement (an average of 3.5% to 3.2% above the optimal) when check3 is added to check1 and MaxDiff, used with the CH cheapest insertion heuristic. On the other hand, the addition of check3 with Stewart's algorithm resulted in worse tour lengths (3.0% to 3.5%). In problems 24 - 28, the reverse is true, with an improvement using Stewart's algorithm and a reduction in tour length with CH cheapest insertion, though the differences in percentages here are not as great as in problems large1 - large5.

5.2 Analysis of MaxDiff

The most noticeable improvement occurs when MaxDiff is applied to the cheapest insertion algorithm; cheapest insertion improved from an average of 16.3% above the optimal to 1.4% above the optimal for problems 24 - 28. Besides this case, MaxDiff performed much better on the average with the large (500 cities) problems than with the 100 city problems. In the 100 city problems, MaxDiff applied to cheapest insertion produced much better solutions than cheapest insertion (without MaxDiff). Besides cheapest insertion, MaxDiff showed little or no improvement as compared to the original algorithm it was being applied to. Only when MaxDiff was used in combination with check1 were good tours consistently found.

In the larger problems, the application of MaxDiff made a marked improvement over the original algorithms (Stewart's, cheapest insertion, and CH cheapest insertion). This improvement increased when the checks were used with MaxDiff, especially check1 and the combination of checks 1 and 3.

MaxDiff doesn't apply to Stewart's algorithm as well as it does to the cheapest cost algorithms; although in the 500 city problems, MaxDiff applied to Stewart's algorithm resulted in a shorter tour and a more efficient algorithm than Stewart's algorithm in all five problems.

5.3 Analysis of Computation Times

Table 5.5 and Table 5.6 list the computing times (in seconds) for all 10 problems. The times marked by an asterisk were estimated. Nearest neighbor and nearest insertion were by far the fastest of the algorithms tested, with nearest neighbor usually taking a little less than a third of the time required by nearest insertion. Nearest neighbor took 6 seconds to find a solution for the 100 city problems and approximately 2 minutes for the 500 city problems.

The rest of the heuristics took a much longer time, with times of one to two minutes for the 100 city problems and times of 32 to 58 minutes for the 500 city problems. Most of the time results are not surprising with the checks consistently increasing the computation time. Although the checks did increase the computation time, on the average the time was never doubled.

Although check2 performed faster than check1 as expected (discussed in the *analysis of checks*), the difference in time was not significant. Thus, when check1 removed a city from the subtour, the cost to reinsert the city was minor. There was also no sign of a termination problem as discussed in section 4.1. The difference in time between check4 and check5 was very minor and as detailed above, check5 consistently produced shorter tours than check4.

An algorithm with MaxDiff applied to it is longer than the same algorithm without MaxDiff, because instead of calculating and updating the one edge which satisfies the insertion criterion the best for each city not

Table 5.5 Time in seconds for the 100 city problems.

Algorithm	prob 24	prob 25	prob 26	prob 27	prob 28
Nearest Neighbor	6	6	6	7	7
Nearest Insertion	19	20	20	20	20
Cheapest Insertion	92	80	73	81	101
+ MaxDiff	57	64	63	63	65
CH Cheapest Insertion	45	50	50	53	49
+ check1	93	89	90	86	84
+ check2	82	87	85	85	80
+ check3	57	63	61	59	57
+ check4	52	56	54	54	50
+ check5	52	57	53	53	50
+ check6	78	80	78	78	74
+ checks 1 & 3	98	97	106*	92	92
+ checks 3 & 6	87	89	90	84	79
+ checks 3, 5, &6	87	90	91	85	79
+ MaxDiff	51	55	52	50	54
+ MaxDiff + check1	85	95	95	87	90
+ MaxDiff + checks 1 & 3	90	99	104	93	97
Stewart's Algorithm	56	58	76	59	49
+ check1	97	98	129	92	83
+ check2	87	95	123	91	82
+ check3	63	69	88	65	56
+ check4	57	64	79	60	50
+ check5	57	65	79	59	49
+ check6	85	88	92	85	74
+ checks 1 & 3	101	104	140	99	89
+ checks 3 & 6	92	96*	101	91	79
+ checks 3, 5, &6	93	97*	102	92	80
+ MaxDiff	74	79	90	78	58
+ MaxDiff + check1	114	114	138	125	101
+ MaxDiff + checks 1 & 3	121	121	143	130	107

Table 5.6 Time in seconds for the 500 city problems.

Algorithm	large1	large2	large3	large4	large5
Nearest Neighbor	121	127	132	137	148
Nearest Insertion	447	453	452	463	474
Cheapest Insertion	5340	3837	4722	4459	4146
+ MaxDiff	1537	1573	1597	1524	1560
CH Cheapest Insertion	1929	2003	1930	1913	1903
+ check1	3105	3145	3234	3048	3140
+ check2	2958	3040	3172	2981	2933
+ check3	2275	2380	2517	2512	2486
+ check4	2120	2221	2317	2273	2095
+ check5	2140	2203	2366	2298	2103
+ check6	2896	2957	3059	2905	2964
+ checks 1 & 3	3277	3413	3421	3405	3479
+ checks 3 & 6	2931	2817	3114	3170	3143
+ checks 3, 5, & 6	2931	2826	3123	3135	3150
+ MaxDiff	1466	1587	1541	1570	1569
+ MaxDiff + check1	2448	3279	2482	2639	2502
+ MaxDiff + checks 1 & 3	2476	2422	2516	2649	2540
Stewart's Algorithm	1906	1954	2147	2127	2260
+ check1	3108	3238	3432	3290	3506
+ check2	3027	3121	3206	3287	3325
+ check3	2212	2275	2512	2508	2599
+ check4	2153	2204	2341	2373	2513
+ check5	2180	2238	2355	2390	2527
+ check6	2941	2985	3195	3182	3351
+ checks 1 & 3	3168	3305	3549	3440	3291
+ checks 3 & 6	2804	3031	3057	3262	2883
+ checks 3, 5, & 6	2822	3038	3061	3273	2890
+ MaxDiff	1566	1652	1665	1619	2199
+ MaxDiff + check1	2708	2762	2849	2771	3540
+ MaxDiff + checks 1 & 3	2679	2777	2841	2779	3123

in the subtour, MaxDiff requires that two edges must be maintained. However in this study, MaxDiff applied to a particular algorithm actually often decreased the computing time. In the 500 city problems where MaxDiff was applied to cheapest insertion, CH cheapest insertion, and Stewart's algorithm, the time was reduced in all cases.

The reason for the decreased time is that MaxDiff has a tendency to reduce the complexity of updating NT (the list of cities not yet in the subtour). As discussed in the existing algorithms section, the worst case performance can arise during the updating of NT. A brief review is given here: After every city k is inserted into T (the list of cities in the subtour) between cities i and j , each city p in NT must determine the edge with which p fits in the best. If p previously fit in edge (i,j) the best, then p must look at all edges in the subtour; otherwise p needs to look only at edges (i,k) and (k,j) . The first of these two cases leads to the worse case time complexity.

Figure 5.1 and 5.2 illustrates tour construction by an algorithm not using and using MaxDiff, respectively. In Figure 5.1(a) and 5.2(a), there are 11 cities for which edge (i,j) is the edge where the city fits in the best. After the first city is inserted, all remaining 10 cities must look at all edges in the subtour because each one of the 10 cities pointed to the edge which was just lost. In the non-MaxDiff algorithm, this process continues where the remaining cities must look at all possible edges. In the MaxDiff algorithm, when the second city is added (Figure 5.2(c)), only four cities must look at all edges in the subtour, while the other 5 cities only need to examine the two new edges (e.g. (i,k) and (k,j)). When the third city is added (Figure

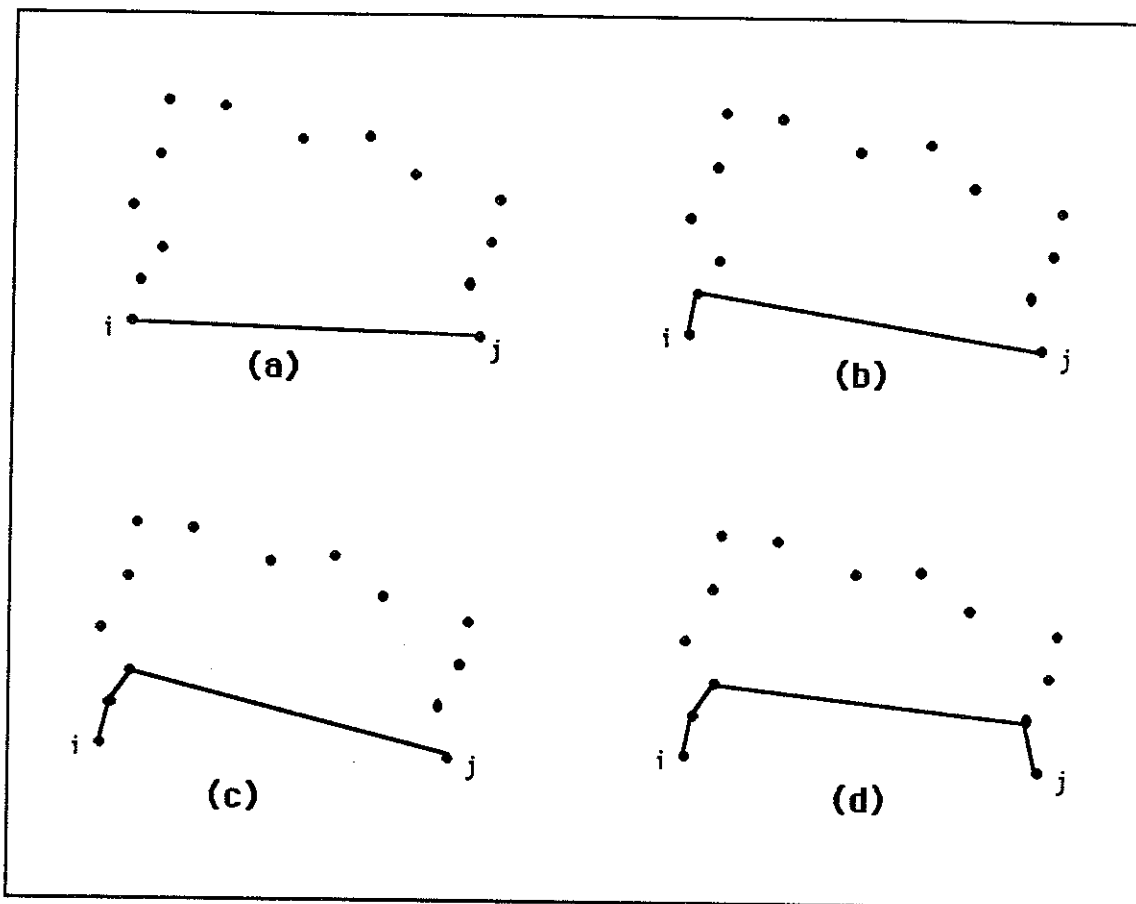


Figure 5.1. Worst case complexity when updating NT.

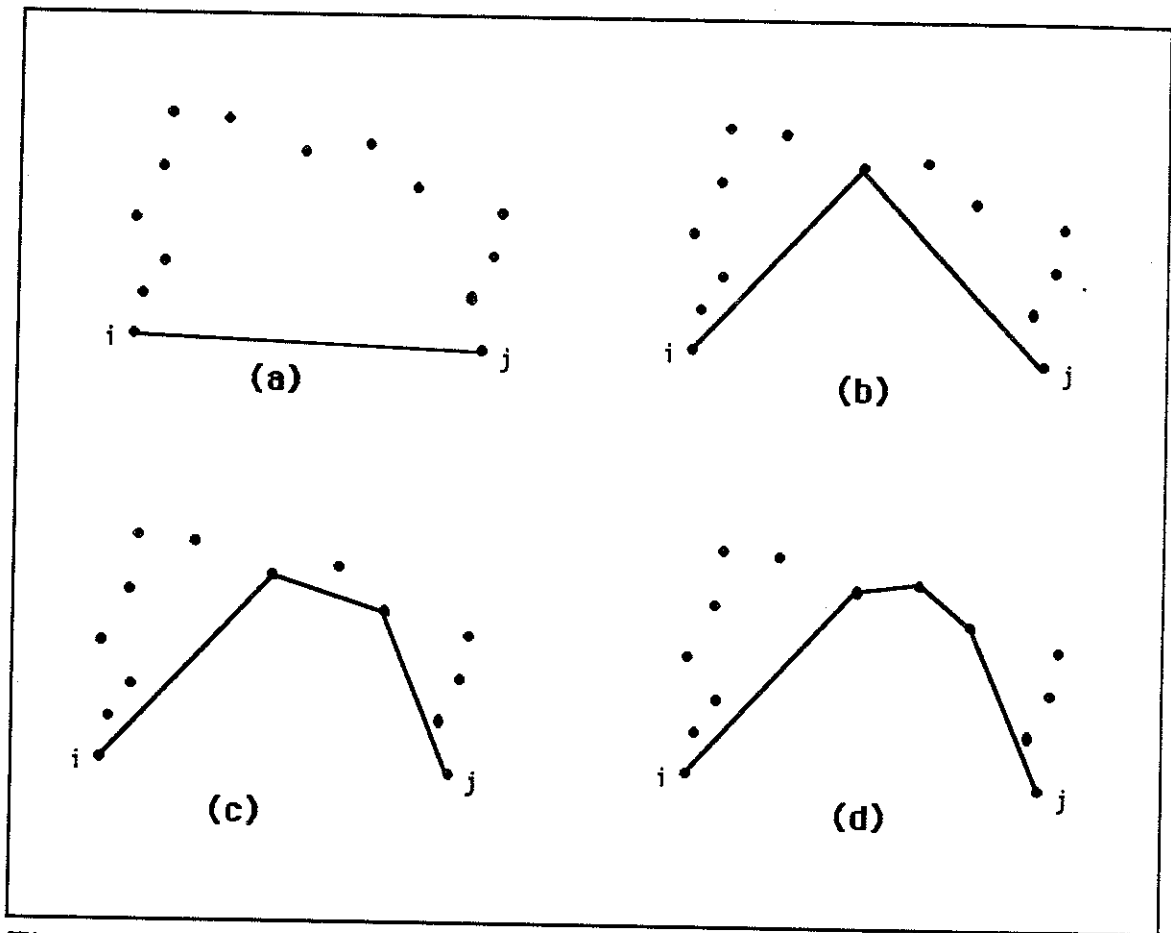


Figure 5.2. MaxDiff solution reduces complexity of updating NT.

5.2(d)), all eight cities will only look at the two edges added last. Thus after 3 cities are added to the subtour in these examples, the MaxDiff inspects all cities in T 15 times ($10 + 5 + 0$) while the non-MaxDiff algorithm does 27 times ($10 + 9 + 8$).

6.0 CONCLUSION

Certain conclusions can be drawn from the results. Nearest neighbor and nearest insertion should only be used when a lower bound on the optimal tour is desired. Although these algorithms are fast, neither one of them produces good solutions.

Cheapest insertion should probably never be used as an ETSP heuristic. The computation time was similar to that of the other $O(n^2 \lg n)$ algorithms studied in this research, but the tour lengths were much worse. If cheapest insertion is to be used, it should definitely be used with MaxDiff applied.

With and without checks, the Stewart algorithm overall performed better than the convex hull cheapest insertion algorithm. Only in one case (problem 27) did CH cheapest insertion, without checks, produce a very good tour. Thus CH cheapest insertion alone is not recommended as a good tour heuristic. CH cheapest insertion becomes a valuable heuristic when MaxDiff is applied to it and checks 1 and 3 are added. Although check1 and check3 used in combination can add 25% to 75% to the computing time, on the average the additional time results in a 2% to 7% shorter tour.

Table 6.1 lists the most successful algorithms tested in this research (the best solutions are highlighted). The first column lists the average

Table 6.1 Summary of the best heuristics.

ALGORITHMS	DISTANCES		TIME		
	% over	avg absolute	Relative values		% of orig
	optimal	value	large 1-5	24 -28	
Cheapest Insertion	17.39%	5.85	0.0013	0.0712	100.00%
+ MaxDiff	2.80%	69.59	0.0164	1.8223	54.58%
CH Cheapest Insertion	8.52%	17.46	0.0042	0.5321	100.00%
+ MaxDiff	3.72%	38.63	0.0148	1.0419	93.15%
+ MaxDiff + check1	2.79%	50.15	0.0115	0.7230	163.94%
+ MaxDiff + checks 1 & 3	2.64%	47.88	0.0113	0.6606	174.28%
Stewart	5.22%	24.52	0.0095	0.6013	100.00%
+ checks 1 & 3	2.64%	51.96	0.0086	0.7035	166.08%
+ checks 3 & 6	2.76%	50.15	0.0087	0.7927	150.37%
+ checks 3, 5 & 6	2.64%	51.25	0.0094	0.7816	151.46%
+ MaxDiff	5.33%	20.92	0.0104	0.3022	105.50%
+ MaxDiff + check1	2.94%	46.79	0.0116	0.4698	170.45%

percentage over the optimal for all ten problems. The next column contains the average *absolute values*. The absolute value is the inverse of the percentage over the optimal (although note that the algorithm with the best absolute value is not the same as the algorithm with the lowest percentage over the optimal). The *relative value*, in the next two columns, is the absolute value divided by the time it took to compute the tour. The relative value describes the quality of the tour as the length per computing time. The average computing time of a modified algorithm (i.e. with checks or MaxDiff) as a percentage of the original algorithm is also listed.

The shortest tours were found by Stewart's algorithm using a combination of checks 3, 5, and 6, checks 1 and 3, and CH cheapest insertion with MaxDiff applied and checks 1 and 3 added. The average increases in time for these heuristics were between 50% and 75%, which is reasonable for the better solution.

Some of the checks were more successful (i.e. improved tour construction more) than others. Check1 and check6 were the most costly and the most successful. Check3 and check5 detected some improvements in the tour that check1 and check6 did not, and did this very quickly. Check2 and check4 were not as successful. Check2 was slightly faster than check1, but did not detect many of the improvements that check1 did. Similarly, check4 was slightly quicker than check5, but also missed many of the improvements that check5 detected.

It is recommended that either check1 or check6 should be used when an increase in time is not critical. Check3 and check5 should be used with these checks since they are efficient and usually result in a shorter tour.

The algorithms that produced the shortest tours were discussed above. Table 6.1 also indicates that MaxDiff applied to cheapest insertion is worth discussion, rating the best in four of the five categories. As mentioned before, the tables report the best of three runs for cheapest insertion. MaxDiff applied to cheapest insertion is not recommended unless a number of runs are made.

In summary, if only one algorithm is to be used, then Stewart's algorithm with checks 3, 5, and 6 should be used since it is the quickest of the three best heuristics. It is recommended though, that at least two algorithms are used, two of them being Stewart's with checks 3, 5, and 6, and MaxDiff applied to CH cheapest insertion with checks 1 and 3. One reason for choosing these two algorithms is that when a modified Stewart algorithm produced a bad tour, a modified CH cheapest insertion algorithm did well; and when CH cheapest insertion performed poorly, Stewart's algorithm usually produced a good tour. If a heuristic is needed which often produces a tour within 4% above the optimal, then MaxDiff applied to CH cheapest insertion is the best choice because of its speed. This heuristic was the fastest (along with cheapest insertion plus MaxDiff) of all of the algorithms tested in this research.

6.1 Further Research

One goal of this project was to determine if further research in MaxDiff and the use of checks is warranted. MaxDiff can be applied to more

algorithms than the ones listed here. It is not always easy to determine which algorithms are compatible with MaxDiff and then how to apply MaxDiff to the algorithm; but we feel that applying MaxDiff to an algorithm is worth the effort .

The checks in this research could be easily improved. For example, in section 5.0, it was explained that if one check were successful for any particular i, k, j , then no other checks would be attempted. This is because most of the checks modify the edges (i,k) or (k,j) when successful, and thus the edges (i,k) and (k,j) might no longer exist for other checks to examine. More possible improvements would probably be detected if after any check c has changed the tour, then all other checks and c itself, are each called using the new edges just created (instead of using the edges (i,k) and (k,j) as usual).

Another possible check is to modify check6 so that the cities to be repositioned in the subtour are instead removed from the subtour as is done in check1. As stated above, check1 performed consistently better than check2 because it removed the cities instead of repositioning them.

Only some of a variety of possible checks are covered in this research. There are many more checks which could be developed. A good check detects many improvements while adding very little computation time to the original algorithm. Through continued research in this area, hopefully, good checks will develop and thus improve the performance of existing algorithms.

List Of References

- Bellmore, M. and G.L. Nemhauser (1968). The traveling salesman problem: a survey. Oper. Res. 16, 538-558.
- Cerny, V. (1985). Thermodynamical approach to the traveling salesman problem: an efficient simulation algorithm. Journal of Optimization Theory and Applications 45, 41-51.
- Crowder, H., and M.W. Padberg (1980). Solving large-scale symmetric travelling salesman problems to optimality. Management Sci. 26, 495-509.
- Eilon, S., Watson, C.D.T., and Christofides, N (1971). Distribution Management, Griffin, London.
- Golden, B.L., L.D. Bodin, T. Doyle, and W. Stewart, Jr (1980). Approximate traveling salesman problems. Oper. Res. 28: 694 - 711.
- Golden, B.L., and W. Stewart Jr. (1985). "Empirical analysis of heuristics." In The Traveling Salesman Problem. Edited by E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, and D.B. Shmoys. John Wiley and Sons Ltd, Great Britain.
- Garey, M.R. and D.S. Johnson (1979). Computers and Intractability. A Guide to the Theory of NP-Completeness, W.H. Freeman and Company, New York.
- Graham, R.L. 1972. An efficient algorithm for determining the convex hull of a finite planar set. Info. Proc. Lett 1, no. 1, 132-133.
- Johnson, D.S. and C.H. Papadimitriou (1985). "Performance guarantees for heuristics." In The Traveling Salesman Problem. Edited by E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, and D.B. Shmoys. John Wiley and Sons Ltd, Great Britain.
- Kirkpatrick S., C.D. Gelatt Jr., and M.P. Vecchi (1983). Optimization by simulated annealing: quantitative studies. Journal of Statistical Physics 34, 671-680.

- Krolak, P.D., W. Felts, and G. Marble (1971). A man-machine approach toward solving the traveling salesman problem. Comm. ACM 14, 327-334.
- Lawler E.L., J.K. Lenstra, A.H.G. Rinnooy Kan, and D.B. Shmoys (Eds.) (1985). The Traveling Salesman Problem. John Wiley and Sons Ltd, Great Britain.
- Noga, M.T. (1984). "Fast geometric algorithms.", Ph. D. dissertation, Virginia Polytechnic Institute and State University, Blacksburg, Virginia.
- Norback, J.P., and R.F. Love (1977). Geometric approaches to solving the traveling salesman problem. Management Sci. 23, 1208-1223.
- Park, S.K., and K.W. Miller (1988). Random number generators: good ones are hard to find. Comm. ACM 31, 1192-1201.
- Skiscim, C.C. and B.L. Golden (1983). Optimization by simulated annealing: a preliminary computational study for the TSP. Proceedings of the 1983 Winter Simulation Conference, 523-535.
- Stewart, W.R. Jr. (1977). A computationally efficient heuristic for the traveling salesman problem. Proc. 13th Annual Meeting of S.E. TMS, 75-85.

Appendix

The following lists are the test data used in this project. The coordinates given are actually the tours of the most successful algorithm (not including simulated annealing) for each problem. The last tour listed is of problem 25 produced by simulated annealing, which resulted in a better tour length than the optimal length reported by Crowder and Padberg [1980].

problem 24	3447	1830
length=21480.6	3510	1671
tour found by	3683	1533
Stewart's	3911	1673
algorithm +	3955	1743
check1 and also	3950	1558
+ checks 3,5,&6.	3874	1318
178 24	3520	1079
241 341	3113	885
19 674	2991	792
53 857	3479	821
22 987	3756	882
123 862	3822	899
161 906	3854	923
376 825	3888	666
378 1048	3875	598
252 1240	3913	192
274 1420	3893	102
298 1513	3815	169
198 1810	3640	43
463 1670	3416	143
611 1384	3022	474
738 1325	2863	558
872 1559	2936	337
928 1700	2848	96
929 1766	2519	135
890 1846	2542	236
1234 1946	2588	302
1247 1945	2573	599
1251 1832	2599	901
1424 1728	2574	946
1621 1830	2586	1286
1625 1651	2484	1183
1724 1642	2421	1007
1807 1711	2097	981
2178 1619	1917	687
2139 1806	1795	962
2290 1810	1787	1009
2573 1969	1393	1368
2597 1830	1380	939
2678 1825	1115	1052
2728 1698	984	965
2576 1676	938	955
2628 1479	742	1025
2716 1432	611	673
2721 1482	839	620
2945 1622	1187	706
2961 1605	1286	525
3085 1528	1323	280
3384 1498	1429	134
3373 1646	1256	61

1178	100	380	478
953	268	468	319
776	392	347	252
457	334	387	190
327	265	61	81
		171	514
		298	615
		399	850
		749	920
Problem 25.		556	1056
length=22394.5		376	1018
Tour found by		193	1210
Stewart's		71	1323
algorithm +		177	1390
checks 3 & 6.		3	1817
2630	20	563	1513
2614	195	627	1261
2372	127	839	1355
2503	352	782	1462
2310	635	731	1741
2330	741	706	1925
2830	775	962	1895
2801	695	1182	1853
2800	653	1090	1652
2929	485	1423	1322
2938	543	1490	1123
3084	748	1526	1612
3084	774	1697	1924
3370	791	1794	1589
3438	901	1729	1498
3133	1143	2132	1432
3220	1454	2191	1579
3140	1401	2426	1851
3058	1276	2408	1747
2698	1221	2489	1520
2639	1239	2741	1583
2642	1269	2937	1568
2312	1270	3114	1629
2030	1186	3245	1828
2009	1163	3317	1966
2000	1110	3453	1998
1782	995	3417	1808
1829	812	3507	1851
1612	328	3515	1892
1538	224	3611	1968
1517	266	3782	1865
1286	550	3834	1827
1213	910	3675	1522
896	705	3858	1472
844	520	3904	1444
694	552	3876	1165
422	542		

3918	1088	1533	1780	2576	189	1009	1001
3896	742	1357	1905	2781	478	1021	962
3938	516	1327	1893	2990	214	997	942
3829	513	1362	1526	3099	173	981	848
3684	445	1183	1391	3124	408	1179	969
3821	147	1544	863	3249	378	1264	1090
3595	111	1307	964	3297	491	1393	859
3292	152	1027	1041	3278	799	1677	1238
3162	367	826	1226	3174	1064	1699	1294
3123	217	737	1285	3213	1085	1768	1578
3060	155	693	1383	3394	1028	1623	1723
3017	108	901	1552	3564	676	1632	1742
		705	1812	3806	746	1646	1817
		554	1825	3939	640	1787	1902
		457	1607	3835	963	1994	1852
		323	1714	3646	1018	2028	1736
		43	1957	3704	1082	2050	1833
		22	1617	3635	1174	2214	1977
		138	1610	3729	1188	2374	1944
		185	1542			2221	1578
		482	1337			2356	1568
		234	1118			2834	1512
		86	1065			3007	1524
		192	1004			2927	1777
		219	898			3220	1945
		396	828			3248	1906
		242	584			3373	1902
		99	536			3786	1862
		40	462			3805	1619
		14	454			3918	1217
		29	6			3535	1112
		213	220			3332	1049
		721	186			2740	1101
		805	272			2901	920
		812	351			2982	949
		913	317			3023	871
		960	303			3060	781
		1058	372			2944	632
		1031	428			2993	624
		1000	457			3452	637
		834	629			3600	459
		781	671			3599	514
		779	777			3642	699
		868	731			3868	697
		1097	643			3935	540
		1410	307			3946	459
		1774	107			3766	154
		1779	90			3538	125
		1868	197			3503	301
		2049	417			3062	329
		2221	291			2995	264

problem 26
length=20820.4
Tour found by
Cheapest
Insertion +
MaxDiff.

problem 27
length=21598.1
Tour found by
CH cheapest
insertion +
checks 1&3
and also +
checks 3,5,&6.

844 520
 694 552
 422 542
 380 478
 468 319
 347 252
 387 190
 61 81
 171 514
 298 615
 399 850
 376 1018
 193 1210
 71 1323
 177 1390
 3 1817
 563 1513
 731 1741
 706 1925
 962 1895
 1182 1853
 1090 1652
 782 1462
 839 1355
 627 1261
 556 1056
 749 920
 1213 910
 1490 1123
 1423 1322
 1526 1612
 1697 1924
 1794 1589
 1729 1498
 2132 1432
 2191 1579
 2426 1851
 2408 1747
 2489 1520
 2741 1583
 2937 1568

largel	1503	847	958	297
length=50805.6	1497	799	1057	229
Tour found by	1423	837	1160	214
Stewart's +	1392	846	1177	113
MaxDiff +	1294	853	1173	68
check1.	1350	940	964	65
1685 5	1163	1005	965	181
1601 139	1139	946	794	201
1413 81	1092	990	801	165
1347 119	1068	1017	695	112
1320 128	1122	1105	679	22
1333 195	1140	1117	666	119
1434 184	1101	1173	604	209
1395 342	1076	1196	585	220
1344 515	1037	1253	612	274
1369 575	915	1163	792	416
1465 296	904	1064	837	535
1539 239	769	1239	771	589
1588 302	718	1121	680	523
1552 405	762	1033	538	519
1537 593	625	964	527	556
1620 645	537	1016	363	617
1661 652	543	1043	223	622
1684 576	540	1054	146	545
1746 706	515	1079	217	503
1759 719	499	1028	164	466
1628 738	427	880	119	414
1608 735	450	864	268	340
1626 881	384	734	318	386
1673 876	513	772	453	451
1699 975	569	772	491	332
1678 1004	663	755	376	206
1692 1014	693	870	445	127
1661 1074	776	776	231	102
1692 1078	835	689	64	26
1782 1083	901	826	109	285
1912 1155	899	849	109	328
1915 1321	861	913	36	381
1830 1293	998	902	56	480
1793 1271	1138	898	6	525
1761 1267	1230	821	78	644
1798 1323	1195	715	161	747
1765 1435	1004	619	234	815
1625 1480	1170	543	20	856
1592 1351	1176	475	2	979
1578 1277	1191	449	170	1037
1461 1187	1142	374	223	930
1442 1194	1082	390	369	939
1360 1289	1016	423	361	972
1328 1210	987	459	328	997
1538 1016	939	335	314	1195

283	1333	1157	1438	2489	1764	3818	1626
221	1353	1314	1532	2535	1735	3841	1591
188	1302	1232	1652	2569	1805	3938	1444
25	1227	1228	1688	2681	1857	3904	1446
104	1362	1201	1703	2756	1755	3720	1404
47	1421	1170	1798	2714	1711	3667	1489
186	1433	1318	1840	2705	1703	3589	1433
160	1596	1397	1845	2679	1659	3660	1397
179	1612	1369	1909	2641	1657	3643	1274
250	1663	1366	1946	2586	1591	3619	1217
249	1687	1407	1996	2874	1457	3597	1137
256	1730	1457	1941	2908	1487	3423	1150
157	1775	1504	1921	2928	1542	3399	1023
157	1816	1504	1821	2934	1343	3456	1035
159	1819	1671	2000	3101	1289	3467	984
65	1795	1651	1818	3135	1314	3507	869
67	1882	1715	1693	3249	1187	3524	858
293	1946	1803	1712	3305	1131	3584	747
335	1930	1768	1567	3303	1288	3639	750
334	1859	1838	1468	3420	1478	3697	691
511	1969	1860	1422	3472	1465	3735	704
536	1866	1970	1510	3440	1491	3762	688
536	1771	2113	1476	3349	1618	3829	792
587	1858	2052	1317	3304	1649	3822	900
693	1861	2028	1256	3281	1666	3732	865
923	1976	2006	1214	3264	1664	3685	899
1065	1854	2063	1179	3207	1618	3743	939
1060	1662	2158	1237	3133	1558	3670	962
940	1606	2273	1368	3050	1534	3702	1044
916	1594	2414	1333	3038	1577	3766	1171
861	1645	2473	1436	3137	1673	3825	1152
788	1685	2425	1455	3100	1963	3957	1135
749	1561	2409	1609	3217	1994	3878	996
693	1617	2423	1658	3355	1901	3927	914
677	1658	2271	1480	3305	1818	3982	849
639	1640	2256	1591	3281	1801	3957	829
520	1593	2147	1780	3338	1747	3996	756
494	1563	1908	1726	3362	1780	3878	614
412	1592	1871	1736	3440	1887	3977	533
323	1623	1942	1789	3464	1926	3874	521
316	1555	1889	1839	3463	1995	3873	477
363	1489	1893	1866	3487	1902	3867	451
373	1345	1853	1895	3582	1876	3944	433
459	1279	1850	1987	3550	1789	3961	434
556	1342	1957	1872	3628	1631	3988	302
613	1341	2176	1951	3695	1840	3956	209
624	1472	2300	1904	3841	1982	3967	44
710	1401	2410	1950	3990	1787	3943	36
807	1463	2480	1949	3988	1630	3826	89
878	1398	2517	1954	3911	1690	3874	249
895	1335	2410	1869	3896	1641	3787	405

3696	435	2249	1121	2780	222	large2
3704	456	2276	1061	2833	178	length=49259.0
3679	581	2224	1010	2789	9	Tour found by
3503	527	2160	1081	2621	80	CH cheapest
3398	528	2118	1093	2524	76	insertion +
3321	457	2086	1087	2441	211	MaxDiff +
3178	429	2078	1059	2410	251	checks 1&3.
3258	239	1990	1043	2296	279	2165 3
3281	259	2048	950	2299	197	2175 101
3411	279	2045	829	2384	119	2095 39
3481	248	2151	726	2458	45	2031 16
3546	197	2216	765	2402	19	2030 41
3517	74	2227	605	2272	10	1966 84
3409	145	2307	519	2138	27	1893 124
3398	111	2374	604	2082	60	1850 79
3359	76	2351	626	2078	65	1716 121
3319	28	2371	716	2107	119	1702 142
3313	6	2333	740	2138	149	1664 62
3013	97	2300	764	2079	144	1622 45
3059	272	2357	804	2023	213	1464 301
3082	428	2353	815	1973	188	1596 321
2924	442	2386	844	1915	221	1767 292
2804	514	2447	976	1959	290	1827 362
2855	614	2516	947	2130	307	1814 464
2777	722	2565	1073	2300	400	1771 524
2865	749	2609	1030	2226	422	1872 583
2878	812	2671	962	2063	476	1999 552
2893	818	2703	876	2021	435	2074 464
2933	646	2603	808	1935	418	1942 459
3006	542	2497	810	1900	479	1967 282
3124	658	2517	748	1907	483	2030 275
3200	842	2643	692	1969	538	2059 186
3197	845	2502	633	1969	577	2116 178
3187	905	2496	590	1807	522	2169 267
3106	983	2533	572	1843	392	2195 278
3080	940	2544	538	1710	384	2327 187
2998	896	2399	455	1714	383	2371 252
2864	997	2489	431	1772	361	2422 352
3008	1042	2512	394	1790	324	2272 393
2993	1163	2560	343	1811	314	2254 438
2827	1285	2622	362	1867	303	2198 445
2753	1215	2653	376	1749	187	2182 439
2758	1187	2633	404	1899	136	2179 455
2686	1168	2677	458	1869	51	2228 515
2605	1158	2714	425	1813	33	2230 579
2661	1310	2749	314	1794	43	2145 644
2557	1252	2783	324	1770	30	2265 652
2493	1159	2859	284			2257 669
2452	1187	2866	232			2340 768
2409	1180	2865	213			2354 743
2363	1095	2817	252			2366 613

2382	645	3546	684	2106	1599	984	1452
2441	689	3537	755	2067	1494	1043	1360
2507	568	3577	748	2223	1492	1020	1332
2566	604	3594	929	2265	1454	905	1361
2606	523	3538	906	2200	1436	930	1276
2620	492	3479	1005	2189	1374	917	1123
2568	379	3457	991	2172	1288	1165	1087
2620	413	3373	1081	2275	1283	1284	1031
2645	419	3358	1192	2282	1353	1312	966
2675	512	3378	1188	2382	1337	1290	941
2718	477	3504	1072	2318	1194	1269	909
2757	458	3540	1052	2285	1194	1316	852
2820	480	3650	1037	2278	1164	1560	949
2813	507	3677	1039	2186	1200	1550	916
2779	535	3739	1084	2176	1058	1532	857
2699	564	3795	1090	2073	1066	1528	757
2741	611	3769	1096	2062	1193	1587	795
2921	599	3640	1169	2037	1226	1783	879
2835	636	3581	1141	2006	1267	1976	923
2649	771	3568	1163	1981	1302	2134	884
2657	825	3523	1251	1988	1358	2130	851
2530	900	3563	1300	2027	1345	2082	858
2500	861	3491	1316	2091	1340	2034	839
2430	913	3426	1316	2075	1379	1804	772
2402	979	3485	1502	2025	1424	1823	694
2376	1037	3425	1543	1989	1396	1680	627
2576	1159	3368	1465	1964	1446	1657	615
2597	1243	3344	1432	1960	1523	1549	673
2706	1290	3315	1258	1913	1459	1555	522
2580	998	3219	1266	1907	1333	1520	467
2663	975	3046	1339	1855	1183	1499	496
2689	990	3026	1346	1803	1267	1486	491
2745	1051	2835	1481	1656	1258	1474	529
2810	1049	2724	1409	1558	1174	1355	710
2868	932	2675	1459	1543	1156	1337	742
2895	942	2775	1601	1538	1207	1310	748
3024	881	2864	1676	1426	1211	1289	766
3029	882	2693	1815	1412	1196	1199	777
3007	1115	2660	1769	1377	1258	1167	622
3158	1104	2666	1682	1244	1297	1117	806
3187	1058	2550	1677	1153	1359	1059	890
3153	951	2546	1661	1222	1428	1022	751
3253	972	2525	1547	1278	1542	983	740
3256	914	2456	1497	1161	1622	892	882
3273	902	2361	1622	1166	1588	847	941
3225	840	2304	1602	1109	1546	825	693
3291	877	2301	1669	1109	1527	932	696
3404	909	2258	1719	1077	1515	937	541
3425	830	2056	1702	1057	1464	853	442
3489	740	2093	1641	1053	1449	918	334
3550	667	2097	1600	1016	1466	946	272

1005	375	396	730	776	1576	3236	1698
1133	306	540	644	879	1585	3268	1745
1176	463	504	721	896	1613	3240	1775
1211	437	541	784	1005	1671	3297	1905
1261	392	573	765	1006	1822	3299	1767
1255	334	654	750	1077	1780	3330	1720
1223	241	709	749	1085	1817	3433	1744
1256	181	692	795	1275	1937	3454	1624
1265	139	626	891	1334	1993	3472	1642
1146	27	578	877	1546	1857	3556	1716
1085	13	606	1046	1443	1774	3682	1712
988	118	464	946	1462	1634	3589	1954
963	111	329	1085	1444	1536	3648	1996
945	183	185	1073	1414	1518	3707	1922
905	104	261	899	1441	1388	3781	1973
817	6	127	940	1544	1343	3814	1994
802	75	130	997	1513	1461	3867	1924
664	151	42	1016	1558	1644	3972	1906
727	162	45	1115	1641	1603	3951	1834
761	183	125	1215	1637	1566	3948	1766
861	283	171	1223	1800	1530	3754	1604
789	315	238	1315	1769	1647	3777	1566
641	591	214	1351	1655	1772	3671	1440
581	544	122	1324	1793	1805	3669	1376
608	495	77	1295	1853	1798	3881	1462
631	498	44	1342	1741	1939	3929	1382
630	484	50	1426	1826	1924	3978	1242
623	436	110	1515	1892	1988	3869	1190
626	394	233	1725	1989	1957	3866	1080
559	368	161	1831	1991	1901	3927	1014
530	223	49	1808	2106	1985	3901	882
458	62	88	1869	2149	1940	3909	861
358	183	229	1990	2190	1966	3853	880
315	210	274	1970	2219	1866	3769	907
252	353	324	1915	2277	1847	3768	780
258	228	342	1895	2493	1996	3816	774
341	140	347	1833	2519	1959	3779	722
341	87	421	1872	2569	1938	3806	652
275	98	456	1886	2694	1947	3890	671
279	13	548	1986	2698	1960	3995	636
104	41	912	1887	2773	1994	3982	562
51	285	835	1828	2853	1876	3931	463
13	302	705	1770	2999	1991	3959	336
71	478	667	1717	3102	1974	3937	326
96	674	605	1612	3093	1859	3895	296
245	583	453	1433	3037	1749	3857	208
301	492	570	1394	3006	1733	3857	182
399	468	570	1252	3027	1693	3940	163
409	485	626	1235	3100	1681	3973	89
422	558	657	1448	3240	1525	3947	36
373	600	747	1622	3241	1593	3892	56

130	657	569	1952	907	1696	2149	1321
29	699	497	1898	869	1567	2096	1252
63	769	365	1786	887	1555	2174	1278
20	824	478	1709	834	1474	2255	1317
74	953	408	1643	798	1502	2218	1357
179	867	493	1599	777	1495	2250	1390
208	818	477	1528	735	1552	2322	1438
249	763	504	1480	748	1618	2378	1448
339	789	334	1432	663	1649	2335	1513
380	735	333	1382	629	1717	2202	1507
443	765	341	1387	660	1737	2221	1685
480	602	486	1362	664	1770	2310	1867
607	630	594	1386	695	1810	2388	1868
714	785	548	1306	823	1982	2446	1981
610	759	530	1240	901	1911	2631	1999
568	735	538	1151	943	1914	2687	1832
463	860	666	1212	980	1929	2591	1824
479	869	732	1282	927	1835	2501	1742
544	984	847	1277	1051	1814	2467	1625
463	1047	901	1294	1116	1823	2416	1592
486	1104	908	1292	1117	1750	2449	1565
404	1085	883	1108	1127	1707	2506	1418
393	1090	820	1016	1197	1696	2538	1444
378	1139	892	990	1232	1594	2635	1429
294	1213	915	865	1254	1463	2656	1320
252	1186	927	857	1234	1703	2713	1293
211	1168	915	777	1214	1790	2757	1200
287	1029	944	725	1340	1706	2761	1157
332	999	887	592	1432	1589	2764	1105
342	979	929	534	1496	1671	2844	1330
305	959	939	472	1507	1703	2842	1336
290	932	1165	481	1514	1821	2761	1466
198	1028	1084	590	1427	1813	2715	1487
156	1056	1157	668	1482	1905	2723	1579
119	1082	1208	789	1744	1937	2639	1606
14	1122	1031	861	1864	1983	2577	1607
39	1180	1037	1061	1992	1980	2645	1671
116	1268	1028	1140	2101	1984	2718	1706
133	1287	1165	1138	1956	1806	2735	1671
175	1346	1218	1189	1976	1795	2792	1639
81	1393	1304	1243	2021	1708	2816	1650
21	1414	1127	1241	2001	1684	2827	1763
68	1437	1086	1274	1880	1546	2808	1794
154	1513	1030	1365	1923	1502	2849	1965
72	1588	1086	1411	1984	1513	2955	1974
69	1634	1026	1492	2073	1575	2954	1962
104	1677	955	1469	2069	1526	3137	1878
82	1738	972	1616	2103	1466	3161	1867
21	1784	976	1626	2115	1438	3167	1951
248	1852	976	1647	2136	1418	3194	1911
359	1999	923	1703	2084	1385	3201	1881

3353	1955	2915	1227	3607	476	large4
3382	1993	2921	1143	3672	384	length=48767.5
3403	1873	2931	1119	3711	366	Tour found by
3314	1845	2987	1083	3678	457	CH cheapest
3294	1831	3054	996	3671	518	insertion +
3228	1770	3095	896	3692	551	MaxDiff +
3140	1809	3083	868	3720	680	checks 1&3.
3129	1644	3026	879	3813	729	2417 7
3042	1598	3033	814	3850	698	2368 164
2988	1559	3066	720	3906	757	2421 202
3160	1575	2996	649	3898	801	2456 126
3271	1560	2972	623	3938	813	2503 141
3323	1578	3055	651	3938	754	2501 156
3340	1513	3142	628	3910	587	2553 250
3436	1618	3165	732	3920	532	2430 304
3513	1670	3189	718	3922	491	2296 398
3513	1524	3244	702	3941	428	2331 226
3495	1434	3334	589	3967	230	2250 226
3583	1499	3437	592	3893	157	2108 100
3576	1604	3312	653	3795	222	1974 50
3733	1803	3312	684	3777	94	1865 74
3732	1840	3373	744	3671	24	1995 195
3806	1976	3390	827	3569	27	1924 189
3877	1933	3500	838	3473	4	1785 248
3817	1827	3384	989	3637	123	1776 344
3853	1777	3378	989	3651	125	1738 271
3861	1708	3348	921	3561	155	1687 271
3886	1662	3262	932	3481	309	1639 393
3984	1605	3267	951	3426	366	1610 385
3966	1518	3304	1010	3528	413	1592 218
3955	1507	3281	1030	3467	444	1624 123
3849	1447	3260	1026	3403	470	1518 125
3770	1363	3214	1148	3336	450	1492 74
3775	1353	3299	1268	3288	398	1278 223
3830	1232	3331	1260	3264	366	1299 262
3743	1212	3375	1158	3149	453	1384 235
3734	1276	3348	1114	3113	393	1395 269
3709	1331	3441	1097	2967	389	1370 312
3549	1353	3461	1115	2990	363	1342 304
3584	1290	3487	1075	3066	338	1349 440
3601	1259	3505	1010	3301	255	1359 565
3547	1138	3574	1028	3161	135	1373 589
3436	1355	3565	1107	3141	7	1208 528
3403	1363	3619	1070	3002	72	1092 546
3229	1359	3678	1013	2978	52	1081 549
3197	1322	3652	974	2884	21	1081 624
3148	1285	3665	924			1038 679
3058	1381	3717	836			946 552
3081	1230	3698	831			930 487
3010	1216	3621	778			982 431
2967	1216	3681	686			1024 328

1060	300	501	652	337	1301	1316	1420
1115	232	530	571	439	1292	1283	1340
1162	241	528	548	550	1301	1253	1215
1171	247	521	482	576	1341	1165	1324
1216	78	633	609	611	1350	1163	1330
1199	88	841	524	612	1415	1061	1278
1170	74	784	564	615	1460	1050	1307
1015	14	725	644	689	1610	970	1341
1008	142	657	717	607	1609	951	1303
989	177	755	741	522	1725	972	1206
910	250	753	823	599	1711	1087	1166
943	317	790	910	630	1778	1128	1071
701	342	836	1087	628	1818	1266	1056
748	299	891	1191	632	1894	1252	915
803	246	740	1296	803	1986	1176	858
813	136	710	1206	880	1786	1165	835
870	54	748	1138	914	1758	1040	841
746	77	707	1006	995	1760	1118	783
708	53	601	1004	869	1698	1155	704
662	67	556	855	894	1489	1185	676
709	121	515	818	1015	1515	1392	777
658	120	490	735	1199	1583	1443	890
556	208	268	851	1204	1613	1487	780
498	200	225	874	1189	1712	1497	772
544	297	425	978	1354	1689	1599	756
392	289	342	1074	1269	1753	1626	652
405	228	223	1126	1209	1905	1658	630
329	99	129	1055	1259	1906	1713	875
356	69	18	1218	1397	1993	1846	857
305	53	16	1309	1451	1957	1882	585
248	8	57	1368	1436	1934	1839	552
169	35	102	1287	1527	1894	1831	487
159	132	150	1302	1679	1821	1840	419
67	167	180	1404	1646	1736	2007	452
138	202	251	1498	1739	1613	2079	382
157	302	90	1609	1623	1581	2071	410
62	324	191	1711	1631	1559	2078	514
148	384	139	1818	1694	1346	2096	579
59	560	76	1823	1698	1241	2137	602
15	560	232	1947	1674	1197	2118	641
34	688	271	1835	1611	1075	2169	625
90	663	279	1765	1634	1048	2201	573
252	629	366	1800	1669	1031	2356	589
207	557	442	1813	1627	979	2431	511
204	514	372	1743	1506	1023	2502	649
287	307	369	1743	1563	1172	2591	476
363	424	371	1686	1574	1310	2650	569
369	515	465	1622	1561	1318	2672	638
381	587	372	1554	1549	1316	2684	676
348	616	354	1520	1481	1585	2688	758
425	667	378	1428	1431	1488	2683	762

2690	821	2003	1921	3654	1491	2660	1319
2622	850	2215	1890	3600	1442	2537	1183
2413	880	2280	1768	3602	1376	2495	1084
2386	836	2273	1761	3597	1372	2536	1024
2355	892	2268	1744	3549	1338	2557	1039
2320	892	2345	1760	3488	1376	2571	1057
2357	920	2395	1848	3498	1453	2643	1048
2300	1033	2520	1879	3425	1535	2778	1049
2250	1097	2470	1766	3330	1582	2812	1073
2319	1128	2565	1773	3307	1658	2835	992
2402	1184	2613	1745	3280	1581	2866	947
2234	1229	2630	1837	3307	1563	2873	847
2199	1212	2612	1856	3211	1513	2921	814
2238	1319	2751	1982	3177	1515	2910	776
2211	1374	2759	2000	3150	1504	2917	715
2120	1351	2841	1925	3109	1309	2782	618
2080	1359	2901	1816	3229	1337	2888	607
2067	1337	2937	1719	3303	1293	3036	575
2130	1277	3064	1782	3434	1288	3060	680
2073	1104	3129	1908	3503	1239	3110	699
2181	1045	3206	1971	3465	1162	3149	641
2182	1023	3261	1862	3419	1181	3214	498
2107	931	3439	1928	3405	1132	3198	389
2157	828	3588	1922	3284	1063	3267	374
2097	710	3605	1870	3279	1042	3324	456
1948	823	3683	1944	3253	1078	3336	486
1975	852	3732	1991	3282	1150	3460	470
1980	916	3851	1959	3220	1200	3513	502
1954	900	3845	1840	3212	1191	3382	584
1855	967	3822	1816	3145	1157	3346	599
1840	981	3606	1789	3007	1128	3271	719
1766	1021	3620	1730	3024	1024	3261	750
1917	1058	3687	1732	2913	1085	3122	765
1904	1115	3755	1690	2930	1172	3102	753
1735	1207	3884	1752	2919	1219	3142	892
1771	1268	3901	1734	2961	1233	3227	934
1830	1289	3949	1620	2979	1306	3227	927
1856	1298	3898	1577	2986	1363	3314	847
1909	1357	3953	1534	3017	1393	3393	808
1961	1414	3932	1397	2982	1437	3443	971
1966	1406	3984	1268	2820	1401	3507	847
2013	1410	3907	1196	2809	1477	3517	857
2031	1434	3855	1203	2871	1521	3536	865
2130	1482	3843	1218	2899	1597	3599	833
1875	1578	3713	1096	2858	1569	3641	889
1865	1632	3641	1043	2753	1599	3694	725
1924	1669	3661	1079	2739	1565	3833	824
1831	1777	3686	1103	2688	1595	3851	945
1790	1936	3716	1221	2426	1563	3894	966
1960	1930	3787	1333	2555	1455	3972	950
1990	1929	3739	1410	2645	1409	3946	913

3933	839	large5	273	550	16	1382
3960	730	length=49091.4	237	464	29	1381
3934	585	Tour found by	314	467	44	1452
3950	551	Stewart's +	366	409	64	1411
3975	487	MaxDiff +	473	524	113	1402
3894	382	check1.	440	553	165	1440
3798	637	1299 5	395	586	139	1470
3718	568	1350 94	377	671	209	1489
3670	521	1330 123	503	635	289	1571
3585	501	1226 169	562	550	145	1600
3589	456	1105 164	676	503	111	1572
3680	436	1049 203	768	592	130	1632
3720	362	1061 22	837	648	108	1680
3781	271	916 66	889	640	110	1764
3891	61	885 9	878	973	67	1881
3714	55	845 91	716	915	202	1953
3713	116	611 52	707	960	239	1883
3525	196	699 178	681	947	296	1790
3517	269	679 264	543	961	422	1672
3378	323	807 319	597	1044	534	1787
3385	207	799 273	800	1057	511	1793
3420	166	829 264	803	1176	452	1955
3370	113	887 293	874	1196	563	1938
3281	58	848 350	853	1266	565	1909
3275	119	942 462	780	1282	604	1875
3226	241	877 430	746	1332	604	1838
3023	152	806 397	725	1214	612	1730
2997	159	715 399	659	1171	609	1700
2933	36	653 447	651	1207	628	1633
2950	199	643 420	647	1267	643	1609
2959	229	630 294	638	1368	642	1568
3008	283	612 300	596	1394	733	1711
3072	317	477 336	614	1441	699	1781
3032	340	540 247	512	1551	730	1971
2928	340	520 130	508	1278	753	1994
3000	410	414 72	475	1233	917	1923
2928	457	342 59	446	1336	985	1971
2857	409	232 81	289	1371	964	1879
2748	337	274 236	226	1290	1032	1891
2745	294	204 338	378	1149	1055	1817
2665	252	183 351	365	1139	1037	1774
2714	209	93 255	253	1079	1046	1733
2860	241	31 232	315	975	976	1778
2801	193	11 299	282	897	930	1789
2714	100	97 390	97	921	883	1789
2646	106	84 505	142	1012	857	1782
2625	13	90 508	71	1008	845	1693
2588	37	76 554	20	1030	793	1661
		77 646	100	1084	932	1534
		73 691	1	1191	789	1413
		125 586	22	1322	885	1365

975	1325	2317	1288	3033	1336	3815	1391
1094	1376	2340	1196	3145	1395	3771	1366
1191	1360	2482	1306	3176	1396	3927	1279
1232	1271	2516	1208	3252	1479	3945	1167
1256	1234	2555	1235	3113	1490	3903	1018
1276	1284	2546	1281	3114	1488	3815	1047
1461	1356	2614	1316	3094	1457	3821	1094
1355	1492	2666	1249	2978	1497	3777	1059
1292	1547	2630	1211	2804	1418	3681	1132
1275	1575	2623	1135	2763	1489	3634	1178
1265	1780	2599	1065	2739	1436	3524	1178
1140	1899	2643	1026	2663	1470	3509	1184
1098	1984	2677	1027	2630	1508	3392	1203
1275	1997	2743	965	2571	1606	3504	1025
1380	1981	2781	1062	2609	1679	3626	999
1482	1934	2833	982	2714	1686	3622	919
1448	1901	2839	911	2685	1773	3538	929
1504	1822	2780	875	2579	1848	3505	857
1668	1873	2793	852	2681	1904	3545	849
1697	1759	2841	862	2749	1952	3635	847
1592	1741	2916	839	2867	1979	3643	764
1603	1703	3035	811	2948	1876	3706	712
1601	1583	3101	854	2810	1809	3701	745
1601	1577	3046	763	2936	1773	3741	822
1521	1472	3061	703	2992	1644	3883	830
1573	1475	3170	728	3158	1591	3944	834
1737	1440	3261	766	3288	1622	3925	737
1816	1412	3283	790	3073	1744	3903	634
1827	1382	3272	681	3081	1834	3985	554
1855	1244	3267	659	3157	1797	3916	502
1863	1331	3285	678	3192	1831	3936	466
1870	1346	3313	683	3167	1941	3895	392
1935	1438	3449	590	3205	1950	3974	203
2020	1447	3431	740	3262	1925	3964	191
2027	1544	3412	797	3307	1900	3809	229
1888	1672	3388	806	3320	1826	3813	186
1870	1674	3324	867	3385	1794	3835	145
1804	1766	3192	921	3546	1732	3742	18
1949	1892	3147	984	3636	1897	3695	92
1967	1921	3231	1024	3804	1980	3652	213
1942	1945	3211	1133	3944	1925	3542	107
2018	1977	3181	1222	3996	1748	3518	115
2172	1879	3161	1217	3854	1726	3426	21
2374	1986	3095	1192	3767	1767	3331	53
2402	1857	3101	1151	3770	1753	3465	156
2175	1652	3014	1039	3748	1634	3457	289
2263	1649	2978	1172	3660	1628	3552	298
2371	1605	2789	1195	3639	1538	3723	448
2305	1525	2807	1201	3733	1545	3588	524
2308	1375	2901	1234	3754	1487	3569	414
2221	1285	3030	1337	3897	1530	3439	371

3319	354	1875	490	1257	928
3260	392	1799	460	1263	1039
3234	275	1764	506	1119	1213
3216	270	1757	578	974	1213
3167	301	1883	586	938	1064
3169	364	1992	692	1004	1065
3174	434	2064	662	1011	1070
3208	571	2107	579	1078	933
3183	602	2246	627	1147	892
3020	600	2208	674	1062	794
2928	534	2170	752	1073	784
2867	483	2247	814	1183	781
2849	476	2294	814	1139	667
2765	583	2453	764	1059	675
2698	593	2500	676	995	618
2689	510	2553	723	1089	476
2737	523	2687	701	1154	413
2800	467	2587	776	1147	379
2872	343	2580	909	1181	336
2875	339	2443	919	1324	361
3033	449	2446	987	1314	492
3089	300	2324	995	1317	523
3043	267	2229	1077	1288	576
2984	238	2168	1092	1338	540
2912	67	2166	1109	1394	524
2781	95	2142	1096	1468	483
2775	79	2213	907	1475	545
2713	41	2205	875	1541	559
2550	157	2202	842	1609	488
2616	320	2070	836	1624	433
2484	300	2084	885	1615	416
2493	270	1962	934	1635	401
2419	245	1971	1006	1615	397
2387	147	2005	1096	1612	395
2301	16	1799	967	1585	347
2178	94	1769	1037	1659	234
2131	123	1737	1184	1625	209
2167	296	1708	1178	1705	124
2285	345	1562	1034	1628	113
2326	311	1455	932	1595	81
2333	392	1479	885	1601	116
2158	420	1554	865	1513	226
2079	468	1657	793	1383	234
2084	331	1755	740	1477	126
2010	181	1576	721	1439	68
2022	91	1498	689	1477	14
1940	99	1367	741	1420	35
1924	124	1387	778		
1749	180	1366	829		
1746	187	1296	867		
1849	311	1292	933		

VITA

T.W. Tunnell was born in Dallas, Texas, January 13, 1961. He graduated from the University of North Texas with a B.A. in Psychology in 1984 and received a B.A. in Computer Science for the University of North Texas in 1986.

T.W. Tunnell