Order Number 1352353

Dynamic point relocation: An enhancement for traveling salesman problem initial tour construction procedures

Gale, Andrew Dent, M.S.

The University of Arizona, 1993

U·M·I
300 N. Zeeb Rd.
Ann Arbor, MI 48106

DYNAMIC POINT RELOCATION : AN ENHANCEMENT FOR TRAVELING SALESMAN PROBLEM INITIAL TOUR CONSTRUCTION PROCEDURES

by

Andrew Dent Gale

A Thesis Submitted to the Faculty of the

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

In Partial Fulfillment of the Requirements
For the Degree of

MASTER OF SCIENCE
WITH A MAJOR IN ELECTRICAL ENGINEERING

In the Graduate College

THE UNIVERSITY OF ARIZONA

1 9 9 3

STATEMENT BY AUTHOR

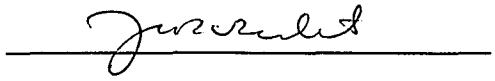
This thesis has been submitted in partial fulfillment of requirements for an advanced degree at the University of Arizona and is deposited in the University Library to be made available to borrowers under rules of the Library.

Brief quotations from this thesis are allowable without special permission, provided that accurate acknowledgment of source is made. Requests for permission for extended quotation from or reproduction of this manuscript in whole or in part may be granted by the copyright holder.

SIGNED: 

APPROVAL BY THESIS DIRECTOR

This thesis has been approved on the date shown below:



Jerzy W. Rozenblit
Associate Professor of Electrical
and Computer Engineering

5/6/93

Date

ACKNOWLEDGEMENTS

I would like to thank my wife, Amy Gale, for the tremendous amount of support she has given me throughout both my undergraduate and graduate educations at the University of Arizona.

I would also like to thank each member of my advisory committee. Each provided a unique aspect to the research, Dr. Sampath Kannan provided the initial direction and made many important suggestions during the research. Dr. Jo Dale Carothers, who understands the design process so well, was very perceptive in regard to the ups and downs of the research and provided insight into NP completeness and graph theory. Dr. Jerzy Rozenblit helped in so many ways, above all he gave me confidence in my abilities.

Dr. David Johnson of AT&T Bell Laboratories should be thanked for his suggestions and for providing the TSPLIB as well as many articles on the traveling salesman problem used in this research.

I also thank Terrence Scott of IBM who has been a great inspiration to me.

TABLE OF CONTENTS

TABLE OF CONTENTS - Continued

LIST OF FIGURES

LIST OF FIGURES - Continued

LIST OF TABLES

ABSTRACT

Traditional solving techniques for the traveling salesman problem are carried out in two phases. First an initial tour is constructed. Then this tour is improved using some form of optimization. Heuristics which are used to construct initial tours ordinarily are incapable of performing any global optimization. The procedure introduced in this research applies a simple optimization technique which minimally increases the runtime of the initial tour construction procedure. At the same time it greatly improves the quality of the tour which is constructed. It does this by globally searching the tour after each insertion for points which require repositioning and performing point relocations where necessary.

CHAPTER 1

INTRODUCTION

1.1 PROBLEM STATEMENT: THE TRAVELING SALESMAN PROBLEM

The traveling salesman problem (TSP) can be stated as follows: given a list of cities a salesman must visit each city in the list once and finish at the same city from which he started such that the total distance traveled is minimized.

More formally a problem is composed of a domain containing instances of the problem and a question that can be asked about any of the instances [Garey & Johnson 1979]. The traveling salesman problem described in this format follows:

TSP
Instance : a finite set of cities C = <c(1), c(2), ... c(n)>, and for each pair of cities c(i), c(j) a distance d(c(i) c(j)) $\in Z+$.
Question : which cyclic permutation Π of the cities minimizes the quantity

$$\sum_{i=1}^{n-1} d(c\pi(i)\ c\pi(i+1)) + d(c\pi(n)\ c\pi(1))$$

This quantity is called tour length.

The Euclidean or geometric traveling salesman problem is stated as the more general TSP with the added constraint that the distance between any two cities must be linear. The cities can be treated as points in the plane. Formally stated

the ETSP requires that each distance be stated as follows: $d(c\pi(i)\ c\pi(j)) = \sqrt{(x\pi(i)-x\pi(j))^2 + (y\pi(i)-y\pi(j))^2}$ where $x\pi(i)$ and $y\pi(i)$ are the x, y coordinates of cities $c\pi(i)$ and $c\pi(j)$.

The TSP is an NP 'hard' combinatorial optimization problem. Thus no known algorithm exists which can solve the TSP in polynomial time. The number of permutations which must be compared to find an exact solution to the TSP is $(n-1)!/2$. This function grows exponentially with the size of the problem. For this reason finding optimal solutions to even relatively small instances of the TSP (100 cities or less) requires an unreasonable amount of time. It would require ten to the sixty-two comparisons to ensure an optimal solution to a 50 city instance.

Practical TSP solving procedures seek to find near optimal solutions in a reasonable amount of time. The most popular such method is to use some form of local optimization. The goal of this research is to find some way to enhance existing procedures of this kind without appreciably increasing runtime. Tests were performed using an inexpensive, in terms of runtime, 'optimize as you go' heuristic.

If this previously untested approach can be proved to be useful on the traveling salesman problem the application of this methodology could be expected to produce similar results when applied to other NP 'hard' optimization problems.

1.2 BACKGROUND

Interest in the traveling salesman problem was stimulated by Merrill Flood in 1937 while attempting to find optimal school bus routing schedules [Dantzig, Fulkerson and Johnson 1954]. It had been discussed informally by mathematicians prior to that time and its earliest roots go back to Euler [Lawler et al 1985]. The problem Euler posed in 1759 was known as the knight's tour problem. A chessboard is modeled as a graph in which two vertices are adjacent if a knight can legally move from one square to the other. The object is to find a hamiltonian circuit in this graph.

The TSP was formulated as a linear programming problem by Dantzig, Fulkerson, and Johnson [Dantzig, Fulkerson and Johnson 1954]. A cutting plane approach was employed to obtain the solution to a 49 city instance of the TSP. This investigation is cited as the origin of Branch & Bound [Lawler et al 1985]. The problem was defined using six constraints which limit the degree of every vertex, the formation of subtours and impose binary conditions on the variables. Even for small TSP instances this formulation required the relaxation of various constraints and the introduction of specialized algorithms to reduce the runtime of the procedure. The runtime of this procedure was still exponential with respect to n, the number of cities [Laporte 1992].

Extensive experimentation using Branch & Bound on the TSP

has been done since this early formulation. It was found that the number of subtour formation constraints could be reduced at the expense of extra variables [Miller, Tucker and Zemlin 1960]. This reduced the runtime of the algorithm.

The best procedures of this type are based on assignment problem relaxation. By relaxing the constraint that no subtours can exist within the overall tour the TSP as it was formulated by DF&J the TSP becomes an assignment problem that can be solved in $O(n^3)$ time [Laporte 1992].

Carpenato and Toth report solving randomly generated 240 city traveling salesman problems to optimality in less than one minute on a CDC 6000 using this approach [Carpenato & Toth 1980]. Balas and Christofides employed LaGrangean relaxation. Using this approach they report solving 325 city instances to optimality in less than one minute on a CDC 7600 [Balas & Christofides 1981]. Recently, Miller and Pekny have solved instances containing up to 3000 cities to optimality using a Branch & Bound algorithm with assignment problem relaxation [Miller & Pekny 1991].

These solving procedures all seek to solve the TSP to optimality. For many practical TSPs where the problem size may be much larger than those discussed up this time this approach may be too slow.

For practical purposes the traveling salesman problem is treated as an optimization problem in which an efficient approximation algorithm is employed to generate near optimal

solutions in relatively fast runtime.

The earliest approximation algorithms were composed of two fundamental steps. First create a complete tour. Then optimize to improve this initial solution. This is still the most popular approach and generally it produces the best results.

Initial tour construction consists of the choice of a starting point (a subset of the set of cities) followed by the insertion or addition of the remaining cities into the subtour until a tour containing all cities to formed. The first such procedures closely resembled either Kruskal's or Prim's algorithms for generating minimum spanning trees [Kruskal 1956] [Prim 1957]. For example the nearest merger algorithm starts with n subtours and successively merges them to produce a complete tour much the same way Kruskal's algorithm merges fragments to generate a minimum spanning tree. The nearest addition algorithm starts with a single point and successively adds the nearest point until a tour is generated. This process resembles Prim's algorithm.

More sophisticated heuristics for selecting and adding or inserting cities into the subtour have been developed which do not appreciably increase time and space needs when compared to these early tour construction procedures. Experimentation has also been done on various subtour types and subtour creation procedures. These are discussed in detail in section 1.4.

To obtain a tour closer to optimal after constructing a

complete tour some method of edge or vertex swapping must be employed. The first such procedure 'inversion' was used by Flood in 1956 [Bentley 1990a]. Inversion has become known as '2 opt', a frequently used local optimization technique. A '2 opt' swap consists of removing two edges from the tour and interchanging one of the vertices of each edge such that a tour is maintained.

figure 1. A '2 opt' swap

'2 opt' has been implemented on the traveling salesman problem using many different criterium to select which edges on which the swapping operation should be performed. A large assortment of heuristic methods have also been used to speed up the edge swapping operation. For example Steiglitz and Weiner use a fixed radius nearest neighbor search to limit the number of possible swapping operations [Steiglitz & Weiner 1968].

The 'r opt' algorithm was introduced by Lin in 1965. It consists of removing r edges and reconnecting them in all possible ways. If any tour is shorter than the previous tour it replaces the initial tour [Lin 1965]. r is held constant during the entire optimization procedure.

The famous Lin-Kerninghan edge swapping procedure is an 'r opt' algorithm in which r is varied as the algorithm progresses. The procedure is difficult to implement but produces solutions close to optimal [Lin & Kerninghan 1973]. A randomized iterated Lin-Kerninghan optimization procedure was introduced by Johnson. This procedure produces better tours than the original L-K algorithm [Johnson 1990].

Some of these traditional algorithms have been implemented using sophisticated data structures such as K-d trees [Bentley 1990] and splay trees, two-level trees and segment trees [Fredman, Johnson, McGeoch & Ostheimer 1993]. Bentley reports that the runtime of some operations such as nearest neighbor searching and fixed-radius searching can be reduced from O(log N) to constant expected time using such data structures.

Other approaches to the traveling salesman problem include the elastic band algorithm [Durbin & Willshaw 1987], tabu search [Glover 1989], neural nets [Hopfield & Tank] and threshold accepting [Dueck 1988]. These approaches are all slower than single run Lin-Kerninghan and do not in general produce better solutions [Johnson 1990]. Johnson has performed a comparison between the iterative Lin-Kerninghan algorithm and Simulated Annealing. The conclusions of this report are forthcoming [Johnson, Aragon, McGeoch & Schevon in preparation]. Simulated annealing found tours which were 0.4% shorter on the average than Lin-Kerninghan when both were run

on 1000-city random geometric instances. Runtime for the Simulated Annealing algorithm was 200 times longer than Lin-Kerninghan. Johnson suggests that a more valid tour quality comparison would result if the Lin-Kerninghan procedure were compared after multiple runs encompassing the same amount of time over which the Annealing algorithm had been run. Lin-Kerninghan does not always produce the same tour for a given problem thus such a comparison is feasible [Johnson 1990].

The Lin-Kerninghan procedure does not attempt any optimization during the tour construction phase. Insertions are executed without any real concern for global optimality. For this reason there exists the possibility that portions of the initial tour may be so misplaced that their correct repositioning during the optimization phase would be impossible.

The procedure introduced here, Dynamic Point Relocation, performs optimization during initial tour construction. In doing so the possibility of placing portions of the tour in an irrevocably suboptimal position is diminished. The runtime of this device is small in comparison to the amount of improvement in initial tour quality due to its use.

1.3 MOTIVATION

The traveling salesman problem is interesting both from a mathematical standpoint, for its engineering applications, and as a scheduling or sequencing problem.

Perhaps the most straightforward application of the TSP is as a vehicle routing problem. A vehicle must visit a given number of locations and return to the starting point such that the total distance is minimized. This problem can be constrained in various ways to create a wide variety of vehicle routing type problems. Such constraints include, among others: time dependencies, the number of vehicles, and the number of 'depots ' to which the vehicles must return [Laporte 1992].

Some engineering applications include x-ray crystallography, printed circuit board drilling, and VLSI fabrication.

In the field of x-ray crystallography it is necessary to perform numerous measurements by positioning a detector at specific locations on the crystals. The order in which these measurements are performed is a solution to the traveling salesman problem. As many as 14,000 measurements have been performed on a single crystal [Bland & Shallcross 1989].

The punching of holes into printed circuits may be modeled as Euclidean TSP (ETSP). The problem is to minimize the horizontal distance traveled by the punching

device [Reinelt 1992]. Printed circuit boards as large as 17,000 have been modeled as ETSPs [Litke 1984].

Traveling Salesman problem instances as large as 1.2 million have been reported in VLSI fabrication [Korte 1988]. In comparison TSP instances of over 100 cities were considered large in the 1970's. This reinforces the need for optimization algorithms that are both fast and produce high quality tours.

The TSP is also interesting from a purely mathematical standpoint as an NP complete decision problem. The decision problem does not seek to find the actual minimum distance tour. It attempts to verify in polynomial time whether any given 'guessed' tour length can to verified as being the minimum or not.

The theory of NP completeness dates back to 1971 [Cook 1971]. It is believed that problems classified as being NP complete cannot be solved by any algorithm in polynomial time. Over 300 decision problems have been classified as being NP complete [Garey & Johnson 1979]. These problems have the interesting attribute that any NP complete problem can be mapped into any other NP complete problem in polynomial time. Thus if one NP complete problem were solved in polynomial time all others would also be polynomial time solvable. Hence the motivation to find new solving techniques for the TSP and problems like it.

1.4 TOUR CONSTRUCTION PROCEDURES

Approximation algorithms which employ some form of local optimization are among the best traveling salesman problem solving procedures in terms of both run time and the quality of the tours they produce. Initial tour construction procedures produce a complete tour which is then improved using some form of local optimization. It follows logically that the closer to optimal the initial tour is the better the optimization algorithm will perform [Bentley 1990a].

The initial tour construction procedure is itself a two step process. First a subtour is created. Then the cities which are not part of the subtour are added to the subtour using some heuristic. These subprocedures are discussed separately.

1.4.1 The Value of the Convex Hull as a subtour

A variety of subtour types have been employed during the initial tour construction process for the TSP. Some of these types are listed below:

a selected single point,
an arbitrary point,
the two points which are closest to one and other,
the two points which are farthest from one and other,
the smallest triangle,
the largest triangle,
x, y, minimum and maximum,
the set of maxima,
the convex hull,
an approximate convex hull.

These subtours have several common features. They are

all optimal tours for the points they contain and the worst case runtime required to produce any of the above subtours is faster than the worst case runtime required by even the simplest procedures which add additional points to produce a complete tour.

The last six subtours in the preceding list require a Euclidean version of the TSP. They lead to better overall complete tours because of this. Of these the convex hull will lead to a better quality tour in a shorter amount of time than the others virtually every time it is used. Overall runtime is improved due to the fact that the convex hull is larger than other subtours thus fewer points will be added or inserted by the slower tour completion heuristic. The fact that the convex hull is larger than the other subtours and that it is optimal for the points it contains accounts for the improvement in tour quality. It has also been observed that the ordering of points contained in the convex hull is the same order in which those points will occur in the optimal solution [Flood 1956].

The following table demonstrates the importance of using the convex hull as a subtour [Lawler 1985]. The five problems in the table are 100 point ETSP instances to which the optimal solution has been verified [Crowder & Padberg 1980]. These problem instances were formulated by [Krolak, Felts & Marble 1971]. The comparison is between the Cheapest Insertion procedure using a single point as a subtour versus the

cheapest insertion procedure using the convex hull as a subtour. Each algorithm is evaluated in terms of percent excess above the optimal solution.

problem number	optimal solution	% excess Cheapest Insertion	% excess Convex Hull Cheapest Insertion
24	21,282	12.1	8.3
25	22,141	11.7	5.0
26	20,749	20.8	4.3
27	21,294	13.0	2.0
28	22,068	12.2	3.6

Table 1: Cheapest Insertion with and without the convex hull.

The average improvement due to the convex hull is about 8 percent in terms of percent excess. This is remarkable in light of the fact that the average percent excess was 14.9 percent without the convex hull. The tours produced using the convex hull as a subtour are over 50 percent closer to optimal. The cheapest insertion procedure requires $O(n^2)$ runtime while the convex hull procedure requires O(nlogn) so runtime is speeded up as well.

1.4.2 Heuristics that Construct Complete Tours

Some of the most popular tour construction heuristics are listed below along their respective worst case runtimes and a brief explanation of how each creates a complete tour. The addition and insertion procedures require a subtour or a fragment as a starting point.

The distinction between an addition procedure and an

insertion procedure is an insertion procedure maintains a subtour during the tour construction process while an addition procedure maintains a tour fragment. Generally insertion procedures outperform addition procedures if the same metric is used to select each point to be added or inserted. This is largely due to the fact that in an addition procedure it is not until the last point is added that a tour is formed. There is no guarantee that the two ends of the tour fragment prior to this addition will be anywhere near each other. The last addition is likely to be very costly.

Hundreds of heuristics have been used to construct initial tours for the traveling salesman problem. Some of the better known procedure are listed below.

Nearest Neighbor Addition(NN). Begin from an arbitrary city. Until all cities have been added visit the unvisited city closest to the most recently added city. Complete the tour by connecting the last city added to the first city. Runtime is $O(n^2)$.

Nearest Insertion(NI). Begin with an initial subtour of the two cities closest to one and other. Until all cities have been inserted into the tour find the city not in the tour which is closest to its nearest neighbor in the current tour. Insert it between the two cities so the cost in minimized. Runtime is $O(n^2)$.

Cheapest Insertion(CI). Same as Nearest Insertion except the insert the city which minimizes d(c(i),c(k)) + (c(j),c(k)) - d(c(i),c(j)) instead of inserting the nearest cities so as to minimize the increase in cost. c(i) and c(j) are in the tour and c(k) is not. Runtime is $O(n^2)$.

Farthest Insertion(FI). Begin with an initial subtour of the two cities whose distance is maximum. Until all cities are inserted into the tour find the city with maximum distance to its nearest neighbor in the tour and insert it the in the same way as nearest insertion. Runtime is $O(n^2)$.

Convex Hull Insertion(CHI). Form the convex hull around the cities. Until all cities are inserted, for each city not yet in the tour find the two cities which will produce the cheapest insertion (as above), insert the city c(k) between c(i) and c(j) for which d(c(i),c(k)) + d(c(j),c(k)) / d(c(i),c(j)) is minimized. Runtime is $O(n^2)$. [Stewart 1971].

Greatest Angle Insertion(GA). Form the convex hull around the cities. Until all cities are inserted find the city not yet in the subtour and the two cities in the subtour which form the greatest angle. Insert the city in the tour between the two in the tour. Runtime is $O(n^2)$. [Norback & Love 1977].

Convex Hull, Cheapest Insertion, Greatest Angle Procedure(CCA). Use the convex hull as a subtour. Use cheapest insertion to select a set of triplets for all points not yet in the tour which obey the cheapest insertion cost criteria. Insert the point which forms the greatest angle with the other two points in the triplet. Runtime is $O(n^2)$. [Lawler et al 1985].

Cheapest Insertion Convex Hull Procedure(CICH). Use the convex hull as a subtour. Execute cheapest insertion to form a complete tour [Lawler 1985].

Greedy Algorithm. List the distances in nondecreasing order. Go through the list adding edges whenever doing so will not create a cycle with less than n cities or create a vertex whose degree is greater than 2. Runtime is $O(n^2 \log n)$

Double Minimum Spanning Tree. Construct a graph which consists of two copies of a minimum spanning tree for all cities. This graph must have an euler cycle. Construct one. Derive a tour by traversing the cycle and taking cities in the order which they are encountered. Runtime is $O(n^2)$.

Christofides(CH). Construct a graph consisting of a copy of a minimum spanning tree and a minimum length matching of the vertices of odd degree in that tree. Proceed the same way as Double Minimum Spanning Tree above. Runtime is $O(n^3)$. [Christofides 1977].

Spacefilling Curve. Visit the cities in the order they appear along a spacefilling curve for the unit square. Runtime is O(nlogn) [Platzman 1989].

Karp's Partitioning Algorithm. Partition the cities recursively using horizontal and vertical cuts through median cities until less than B cities remain. Use dynamic programming to optimally solve the subproblems. Patch the solutions together using the shared medians between set. Runtime is O(n for fixed B) [Karp 1977].

The following table provides a reference as to how well some of the above tour construction procedures perform [Johnson 1990].

procedure	10runs 100 cities	5 runs 1000 cities	5runs 10000 cities
Karp	49.2	56.1	76.0
Double MST	36.9	39.0	39.5
Spacefill	25.4	31.2	34.2
Nearest Insertion	23.0	26.5	26.5
Greedy	22.1	16.9	16.2
Farthest Insertion	8.3	12.5	13.3
Christofides' heuristic	8.8	9.9	-

Table 2 : Average % Excess Above Help-Karp Lower Bound

The Held-Karp lower bound is a statistical estimate of optimal tour length for points randomly distributed in the unit square.

A second table presented below allows for comparison between the two best procedures above in terms of tour

quality, farthest insertion and Christofides' heuristic, with some of the better insertion procedures introduced earlier [Lawler 1985]. The problems 25 - 28 are the same 100 point ETSP instances, to which optimal solutions are available, which were used to illustrate the importance of the convex hull in table 1.

	problem number				
Heuristic	24	25	26	27	28
NN	16.67	16.88	13.35	16.51	13.27
NI	18.69	17.81	22.96	14.44	20.33
FI	5.14	6.97	3.17	1.99	7.42
AI	4.46	3.50	3.28	2.90	5.03
CI	12.07	11.67	20.83	12.97	12.16
CICH	8.29	5.00	4.31	2.04	3.60
CH	7.53	3.93	5.36	14.44	8.51
CHI	3.64	2.52	2.54	2.35	3.45
CCA	1.84	1.35	2.29	3.03	4.54

NN = Nearest Neighbor
NI = Nearest Insertion
FI = Farthest Insertion
AI = Arbirary Insertion
CI = Cheapest Insertion
CH = Christofides' Heuristic
CHI = Convex Hull Insertion
CCA = Cheapest Insertion, Convex Hull, Greatest Angle

Table 3 : Percentage above optimal solution.

On the average the convex hull insertion (CHI) and convex hull cheapest insertion greatest angle(CCA) procedures produce the best tours for problem instances up to 1000 points. Christofides' heuristic is competitive with CHI and CCA for problems of between 1000 and 10000 points. Christofides' heuristic outperforms all other initial tour

construction procedures for problems larger than 10000. Unfortunately Christofides' heuristic is so slow, $O(n^3)$, it is seldom used on large problems where it performs so well.

The three heuristics which were used to evaluate the tour construction procedure introduced during this research, DPR, were the cheapest insertion convex hull procedure (CICH), the convex hull insertion procedure (CHI) and the convex hull cheapest insertion greatest angle procedure (CCA). It was hoped that DPR could improve even the best initial tour construction procedures without appreciably increasing their runtime.

1.5 DYNAMIC POINT RELOCATION

It was stated in the previous section that the convex hull is an optimal subtour of the points it contains and that the order in which the points occur in the convex hull is the same order in which they will occur in the optimal tour. for this reason the deviation from optimality of any tour construction procedure that use the convex hull as a subtour must occur during the insertion procedure (it would not make sense to talk about the convex hull in conjunction with an addition procedure).

As a point is inserted a segment of the existing subtour is broken and two new segments are formed in its place. This has the effect of pulling the tour towards the new point. After a series of insertions the tour may be pulled in such a way so as to make various parts of the tour suboptimal. At any point after the first insertion the potential need to reposition points exists because of this. None of the previously mentioned insertion procedures attempt maintain the condition of optimality during tour construction.

Dynamic point relocation allows any previously inserted point to be moved after each new point is inserted into the subtour. All points are checked to see if moving them from their existing place in the current tour into one of the two newly created segments would reduce the overall cost of the current tour. This is done by reversing the cheapest

insertion criteria for each segment and then applying the cheapest insertion criteria to the three points involved in the latest insertion. Note that two new segments are created by each insertion both must be checked. Hence three points are involved in an insertion. An Example follows:

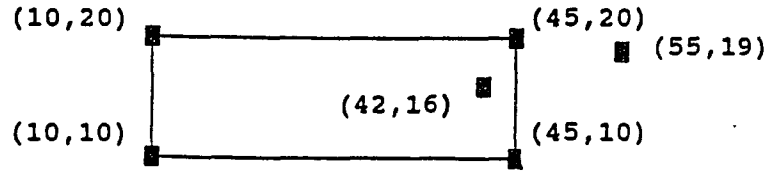


figure 2. A subtour with two cities to be inserted

Cities A, B, C, and D are presently in the subtour and their ordering is optimal. Almost all insertion heuristics will insert E between B and C. This leaves the costly insertion of F to be done. Most insertion procedures will then insert F between E and C to produce the following tour.

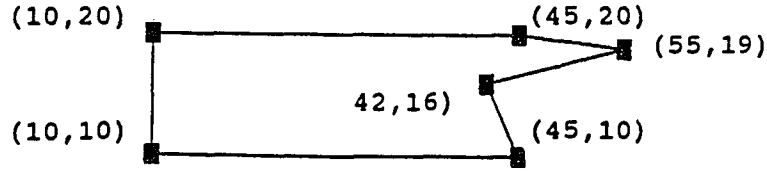


figure 3. Complete tour constructed by most insertion procedures.

It is evident that this is not the optimal tour for these seven points. The insertion procedure in placing points in such a way so as to ensure local optimality fails to create a tour that is globally optimal. Dynamic point relocation will

compute the costs of various other tour configurations and retain the shortest tour, below, which here case is optimal.

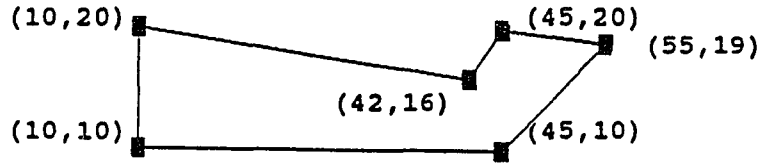


figure 4. Optimal tour produced using DPR in addition to the insertion procedure.

The actual lengths of each of these tours is below:

tourlength w/o DPR 35+10+35+6.71+13.34+10.05 = 110.1
tourlength with DPR 100+35+13.45+10.05+5+32.25 = 105.7

This amounts to a 4.1% reduction in overall tour length due to DPR.

1.6 TOUR QUALITY EVALUATION METHODS

There are three traditional ways of evaluating traveling salesman problem heuristics. They are worst case analysis, probabilistic analysis and empirical analysis. All three have certain merits.

Worst case analysis is a standard evaluation techniques for any algorithm particularly in terms of its runtime. It is a valuable measure as worst case runtime is not machine dependent and so procedures can be compared easily. Worst case analysis is perhaps not the best measure of an algorithm's merit in terms of tour quality. Many algorithms guarantee a solution not worse than twice the optimal solution. In this respect they are equal. In practice though one may perform far better than the others.

Probabilistic analysis is a truer measure of how well a heuristic performs in general. For the Euclidean TSP it is predictable that the optimal tour length will be close to a certain constant which is a function of the number of cities if the points are distributed randomly in the unit square. This lower bound on tour length for randomly generated TSP instances is the Held-Karp lower bound [Held-Karp 1971]. If a procedure is run on enough random instances its average performance with respect to this lower bound will indicate how well it performs in general.

Empirical analysis consists of running the procedure on well known instances so that its performance can be compared

with other procedures which have been run on the same instances. Empirical analysis allows the problem designer to include bizarre instances which may more fully test a procedure. It also provides an exact comparison between various procedures both in terms of runtime and tour quality.

For this research a traveling salesman library (TSPLIB) of instances was obtained from the Computer and Information Technology Institute at Rice University [Reinelt 1991]. This library provided a wide range of test problems in various sizes which exhibit a variety of properties. Most of the comparisons done between the procedures tested in this research were made based on empirically obtained data for this reason.

CHAPTER 2

CONVEX HULL CONSTRUCTION PROCEDURE

2.1 GRAHAM'S SCAN

The convex hull of a set of points P is the smallest convex set that includes all the points in P. If the points were nails in a board the convex hull could be obtained by wrapping a rubber band around the outermost nails so that all nails were interior to the boundary created by the rubberband. The points touching the rubberband would represent the convex hull.

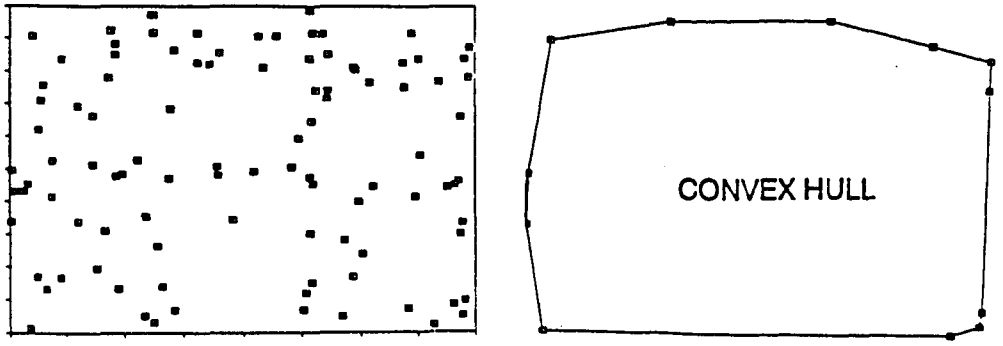


figure 6. the convex hull of a set of points

The convex hull can be used as a subtour only for Euclidean traveling salesman problem instances as the distances between the cities must be linear. This is one of the reasons it is possible to get better solutions to the Euclidean traveling salesman problem than for less constrained traveling salesman problems.

The importance of the convex hull as an subtour was discussed in the introduction. The three tour construction procedures which were implemented for this research use the convex hull as a subtour. This speeds up the overall runtime and improves the quality of each of the procedures.

Graham's scan was used as the convex hull generating procedure [Graham 1972]. It has O(nlogn) worst case runtime. Jarvis' march is another convex hull producing procedure which has O(hn) worst case runtime. Where h is the number of points in the convex hull. In most cases h is very close to logn so two procedures perform at about the same speed. Graham's scan is somewhat easier to implement.

Graham's scan can be implemented in several different ways. The following methodology is due to [Preparata 1985]:

```
FIND the point P with the minimum y value, if there is
     more than one take the leftmost
SORT the other points by minimum polar angle with respect
     to P
PUSH the first three points onto the stack.
FOR i = 3 to n (n is the total number of points)
     DOWHILE ( the angle formed by point NEXT-TO-TOP,
          TOP, and P(i (not in the stack)) makes a
          nonleft turn)
               POP(TOP)
     PUSH( P(i))
RETURN the convex hull as the stack
```

figure 6: Pseudocode for Graham's scan.

2.1.2 Graham's Scan Implementation

Each point's x and y values are contained in arrays. The y minimum is found by making a single pass through the y coordinate array. In the event of a tie the leftmost point is

retained as the y minimum. A pointer to the y minimum's x and y coordinate arrays is pushed onto the stack, which will at the completion of the subtour construction routine contain a representation of the convex hull. This representation is an array of pointers into the x and y coordinate arrays ordered in the way which the points occur in the convex hull.

The rest of the points are sorted by polar angle from smallest to largest with respect to the y minimum which has been pushed onto the convex hull stack. A separate array is created into which the relative polar angles of the points are placed with respect to the y minimum.

All points will be in the first or second quadrant of the Cartesian 2-dimensional plane thus the sine function can be used to compute the polar angles easily because it is positive in both the first and second quadrants.

Trigonometric functions are expensive in terms of runtime. For this reason the relative angles were computed using the slopes of the lines defined by each point with respect to the y minimum rather than the sine function.

Several problems arise if the above approach is taken. First the slope of a vertical line is infinite. Second for points in the second quadrant the slopes are negative and the larger the angle is the smaller the slope becomes. The points will come out backwards if they are in the second quadrant. If the sign is merely reversed these points will become confused with the points in the first quadrant.

To rectify these two problems points which define a line of infinite slope with the y minimum are assigned a value of 1000, the slopes of points in the first quadrant are computed as usual, and for points in the second quadrant the reciprocal of the slope is subtracted from infinity (1000). This way points in the first quadrant create slopes between 0 and 1000, points in the second quadrant create slopes greater than 1000, and points in which the slope is infinite with respect to the y minimum are assigned a slope of 1000 as below.

One additional problem presents itself regarding the ordering of polar angles. If two points define the same polar angle with respect to the y minimum it is necessary that the scan visit the one closest to the y minimum first. The reason for this will become apparent when the scanning phase of the procedure is explained in more detail. In order to ensure that the points are so ordered the lines or vectors defined by the y minimum of each other point in the set must first be sorted by magnitude with respect to the y minimum. This ensures that in the event of three or more collinear points the first ones to be visited by the scan will be those closer to the y minimum. This sorting is accomplished without actually going through the points an additional time by adding or subtracting a fraction of each point's y coordinate from the slope depending on which quadrant the point is in. For example in the case of a point that defines an infinite slope the y value is divided by 100 and subtracted from 100. In this way points

which define equal slopes are sorted by magnitude as well as slope and will be visited in the desired order during the scanning phase of Graham's scan.

It was mentioned that expensive trigonometric operations are avoided by this approach to relative angle computation. The constant term of the sorting procedure has been halved as well by avoided the need to sort by magnitude as well as by angle. These runtime reductions are important as the sorting aspect of Graham's scan is the overriding expense in terms of runtime.

In this respect, Two of the better sorting procedures were considered for this research, Quicksort, and Heapsort. Quicksort is usually 1.5 to 2 times faster than Heapsort on the average yet it uses a great deal more memory. Heapsort is a in-place sort so the memory demands are O(n). For this reason Heapsort was used rather than Quicksort.

Three arrays are passed to the heapsort function. The one that contains the criteria by which the points are sorted is the relative polar angle array. The other two are the x and y coordinate arrays and the elements of these two arrays are merely moved according the sorting of the angle array. The fact that the order of the elements of three arrays must be altered during the sorting process rather than just one was the overriding criteria in placing the space needs slightly ahead of the time needs in this case.

The scanning phase of Graham's scan consists of popping

points off the stack until a non-left turn is produced by the top two points in the stack and the next point not in the stack. A non-left turn is determined by treating the segments between points as vectors and taking the cross product of point triplets. The middle point in the triplet is the top of the stack. The other two are the next to top of the stack point and the next point in the sorted angle list not yet in the stack.

If the cross product of a triplet is zero the points are collinear. Had the points not been ordered by magnitude as well as by angle in cases where two points angles were equal it would be impossible to determine in what order the points occur in such a situation. The possibility of including points in the convex hull which do not belong exists.

Finally after points are popped off the stack which form a non-left turn the next point from the list of unscanned points is pushed onto the stack. This sequence of steps is repeated until all points have been pushed onto the stack and checked. The points which are left in the stack are the convex hull, returned in the order are visited by a traveling salesman tour.

The following figures illustrate many of the ideas discussed above. The figures contain the various stages of development of a convex hull for a small point set.

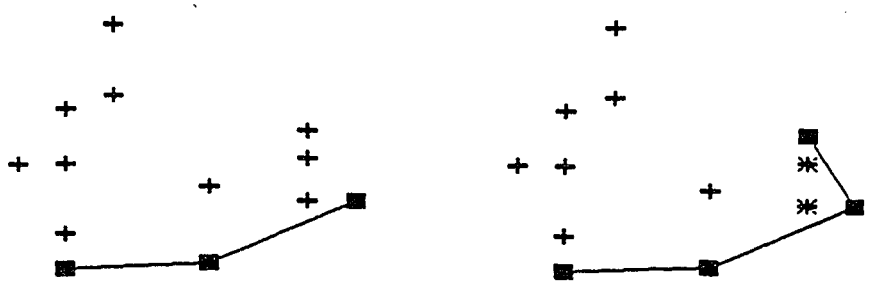


figure 7. The point set with the first three points pushed onto the convex hull stack. The convex hull is indicated by the filled squares.

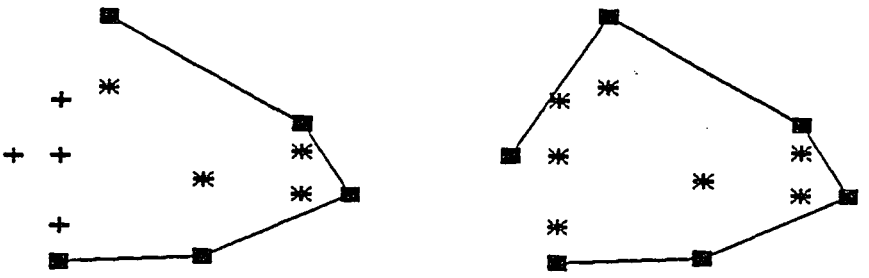


figure 8. During each iteration a point is pushed onto the stack. Then points are popped off the stack until a nonleft turn is formed. A popped off point is indicated with an asterisk.

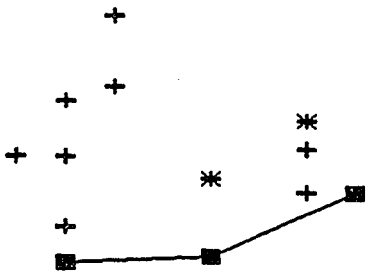


figure 9. Points colinear with the y minimum, indicated with an asterisk, present a problem as the order they are visited is unknown unless they are sorted by magnitude.

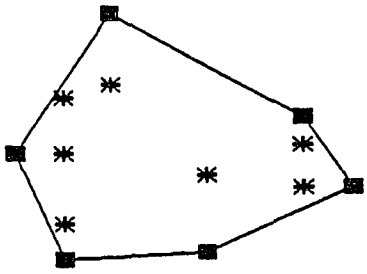


figure 10. The convex hull.

2.1.2 Runtime and Memory Requirements for Graham's Scan

The space needs of creating a convex hull are two additional arrays of size n where n is the number of cities in the TSP instance. One of these arrays is used to stored a sorted list of angles. The other is used to store the convex hull. There is the potential to use more space and this can be done so as to reduce the runtime of the convex hull creation procedure. As the overriding expense in terms of time is not in the creation of the convex hull in some cases where a tradeoff is necessary it seems sensible to choose in favor of reducing space requirements rather than runtime.

Graham's scan is an O(nlogn) algorithm in terms of worstcase runtime. This is easily verified by going through the pseudocode.

The y minimum can be found in constant time.

The points can be sorted by polar angle in O(nlogn) time using a fast sorting procedure.

Lastly, the single scan around the ordered points which

eliminates the interior points from the hull requires O(kn) runtime, where k is a constant which is dependent on the metric used to scan through the points.

Proof that the single scan required O(n) runtime follows: An angle test can be performed in a constant number of operations. After each test the scan either advances to the next point in the sorted list or it removes a point from the stack. Since there are only n-2 points, (P is always in the convex hull as is the first point pushed onto the stack), the scan cannot advance more than n-2 times. A maximum of n-3 points can be popped off the stack as the smallest convex hull requires three points. Thus the scanning aspect of the scan requires O(n) runtime.

The entire procedure requires O(nlogn) runtime in the worst case. This is the time required to sort the points. The implementation used in this research has attempted to minimize runtime except in the case where memory requirement were the overriding factor.

CHAPTER 3

INSERTION HEURISTICS THAT CREATE COMPLETE TOURS

3.1 WORST CASE PERFORMANCE CHARACTERISTICS

Insertion algorithms, when used in conjunction with the convex hull subtour, produce the highest quality initial tours on the average for problem sizes of 10000 cities or less[Bentley 1990a]. Several procedures were presented in the previous chapter which have faster runtime, such as the Spacefilling curve algorithm, but the sacrifice in terms of tour quality is very large. It was also mentioned that Christofides' Heuristic has a better worst case performance guarantee in terms of tour quality and for large TSP instances (over 10000 cities) produces better tours on the average. But its runtime of $O(n^3)$ makes its use impractical in the area where it would be perform best, on large instances.

Any insertion type heuristic can be implemented in $O(n^2)$ worst case runtime. The variance between different insertion procedures is in the complexity of the heuristic used to select and insert points into the subtour. The more complicated an algorithm is mathematically the longer will be the average runtime and possibly its space requirements as well. At the same time if the procedure is to have some merit it will produce higher quality tours in proportion to the added cost in terms of runtime and space requirements.

The initial tour construction algorithms used in this investigation are: the cheapest insertion, convex hull procedure (CICH), the convex hull insertion procedure (CHI), and the convex hull, cheapest insertion, greatest angle procedure. These procedures provide a good cross section of the overall set of insertion procedures in terms of average runtime and average tour quality.

The underlying structure of each of the three algorithms is very similar. This facilitated their implementation and comparison.

Performance data was readily available on each of these procedures with respect to tour quality on TSP problem instances in the TSPLIB. This made verification of the correctness of each of the three heuristics possible.

3.2 GENERALIZED INSERTION PROCEDURE STRUCTURE

Insertion procedures follow the same general format. A subtour is constructed using some heuristic. The remaining points are inserted into the subtour until a complete tour on all points is formed. The adding of points into the subtour requires *selection* of the point not yet in the current subtour followed by its *insertion* into the subtour between two points according to some heuristic measure.

CICH uses the simplest *selection* and *insertion* criteria. It is The fastest of the three on the average. It also produces the poorest tours of the three on the average and requires less memory allocation than do the other two.

CHI is in between the other two in terms of average runtime and tour quality. Its space needs are the same as CCA. CCA produces the best tours on the average and requires the longest average runtime.

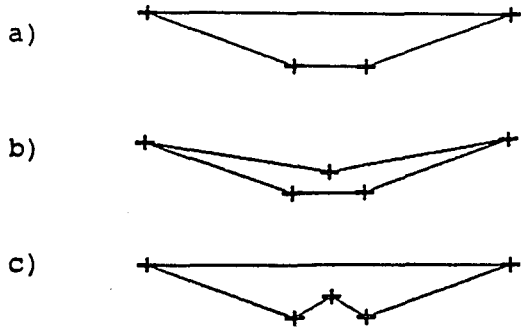


figure 11. A single insert comparison of CICH, CHI, CCA. (a) the initial subtour, (b) greatest angle insertion, (c) cheapest insertion. Convex Hull insertion is generally (a) or (b).

3.3 CHEAPEST INSERTION CONVEX HULL PROCEDURE (CICH)

The Cheapest Insertion convex hull procedure uses simple and straightforward metrics for *selection* and *insertion*. *Select* the point whose insertion into the subtour causes the minimum increase in cost. *Insert* it between the two points such that the overall cost increase is minimized. Pseudocode for such CICH is as follows:

```
build the convex hull as a subtour

FOR i = 1 to n
   mincost  = infinity
   FOR j = 1 to n
      IF (j = n) then jnext = 1
         ELSE jnext = j + 1
      IF (pt[i] is not in the tour)
         IF ((dis[pt[i],pt[j]] + dis[pt[[i],pt[j]] -
              dis[pt[i],pt[jnext]]) < mincost)
selection   THEN MINCOST = (dis[pt[i],pt[j]] +
                            dis[pt[[i],pt[j]] -
                            dis[pt[i],pt[jnext]]) <
                            mincost)
            SAVE(i, j, jnext,-> pnew, prev, pnext)

insertion   INSERT (pnew, prev, pnext)
```

figure 12. Pseudocode for CICH

3.3.1 CICH Implementation

The convex hull is created by the subtour creation procedure as described in chapter 2. It is stored as an array of pointers to the x, y coordinates of each point it contains. Furthermore an array of flags is used to indicate which points from the entire list of points have been inserted into the subtour.

In the case presented in the previous section, kroA100,

the convex hull contains 12 points of the 100 total points. One array points to the position of each of these points in the x, y coordinate array. This array contains the pointers to the x and y coordinates of the points in the convex hull. The other array flags the position in the tour of each of these points from its position in the x, y coordinate arrays. These arrays point to each other.

During each pass through the two FOR loops the remaining points are compared using the *selection* and *insertion* criteria described above. When the program terminates all the flags are set in the flag array to point to each coordinate's position in the tour and all the elements of the pointer array contain pointers to valid x, y coordinates. The tour is output indirectly through these pointers and the overall tour length is computed using the Euclidean distance metric between each two consecutive points plus the distance between the last point and the first.

The actual distances between points are computed only once and stored in a matrix. In fact only the upper half of this matrix is computed. Since the distance matrix is symmetric the computation from any city a to city b is the same as the distance from city b to city a. Storing these computations increases the overall storage needs of the procedure but reduces the average runtime of the *selection* and *insertion* procedure. Due to the number of distance

calculations that would be required without this storage the tradeoff is well worthwhile.

The indices of each distance in the distance matrix can be accessed using the same pointers which are utilized to access x, y coordinates. The points physically do not change positions during the tour construction process. Pointers are used to access their x, y values, their position in the tour, and the distance between them.

The points are physically moved when they are sorted by polar angle during the convex hull subtour procedure. Thus the distance computations are not performed until after the subtour is constructed.

3.3.2 Runtime and Memory Requirements for CICH

The worst case runtime of CICH is $O(n^2)$ this is evident by a nested FOR loop. It does not require any additional memory above what the convex hull procedure requires. In fact it uses less space because Graham's scan requires an extra array to hold points sorted by polar angle whereas no such need exists in CICH. The same array which holds the pointers to the convex hull holds the pointers to the complete tour when CICH is finished running.

CICH is computationally simple as well. Each cost comparison requires only two additions and a compare to the present best values. Average case runtime is tabulated in the Appendix to facilitate comparison of the three heuristics.

3.4 CONVEX HULL INSERTION PROCEDURE (CHI)

The Convex Hull Insertion Procedure uses the same subtour and the same *selection* criteria along with an additional measure to determine where the *insertion* of each point should occur. For each point not in the subtour the two points in the tour which require the cheapest insertion are retained along with that point as a triplet during the *selection* process. Of these triplets the one *inserted* is the one in which the ratio of the old cost divided by the new cost is minimum. Pseudocode for (CHI) follows:

```
build the convex hull as a subtour

FOR i = 1 to n
   mincost  = infinity
   FOR j = 1 to n
      IF (j = n) then jnext = 1
         ELSE jnext = j + 1
      IF (pt[i] is not in the tour)
         IF ((dis[pt[i],pt[j]] + dis[pt[[i],pt[j]] -
              dis[pt[i],pt[jnext]]) < mincost)
selection   THEN MINCOST = (dis[pt[i],pt[j]] +
                            dis[pt[[i],pt[j]] -
                            dis[pt[i],pt[jnext]])
            SAVE(i,j,jnext,-> pnew[i], prev[i], pnext[i])
insertion   MINRATIO = infinity
            FOR j = 1 to n
               IF (pnew[j] is not in the tour)
                  IF (j = n) then jnext = 1
                     ELSE jnext = j + 1
                  IF ((dis[pnew[j],prev[j]] +
                       dis[pnew[[j],pnext[j]]
                            / dis[prev[j],pnext[j]]) <
                                 MINRATIO)
                  THEN MINRATIO = ((dis[pnew[j],prev[j]]+
                                   dis[pnew[[j],pnext[j]])
                                  /dis[prev[j],pnext[j]])
                        INSERT (pnew, prev, pnext)
```

figure 13. Pseudocode for CHI

The only difference between CHI and CICH is that the *selection* process of CHI requires repeating the outer FOR loop one additional time for each insertion.

3.4.1 CHI Implementation

The implementation of CHI is identical to CICH as far as the subtour and selection processes are concerned with one exception. During the selection process of CICH only a single point not into the tour is retained along with is potential position in the tour. During the selection process of CHI all such triplets are retained.

These three points are retained using a single array (P) of pointers and the fact that the points in the tour between which the non-tour point belongs are next to each other in the array of pointers holding the tour. The position of the non-tour point in the x, y coordinate arrays and thus the distance matrix is marked by its position in P while the value in P points to the first element in the tour where this point would best fit. The second element in the tour is implicitly next to the first.

The insertion process then requires one additional pass through the outer FOR loop per insertion to compare the ratios described above.

3.4.2 CHI Runtime and Memory Requirements

The Convex Hull Insertion procedure is still an $O(n^2)$ algorithm but the constant is larger. In terms of worstcase

runtime it is doubled by the additional pass through the outer FOR loop. 2n additions, n multiplies and n arithmetic compare are performed each time this outer FOR loop is executed. The effect of these additional operations can be seen in the average runtime which is tabulated in chapter 5.

The space needs of CHI are increased by a factor of n where n is the number of cities in the problem instance. Due to the need to retain a triplet for each city not yet inserted into the tour. This is not significant when it is considered that sorted polar angle array from Graham's scan could be reused if desired for this purpose.

It is notable that the implementation of the tour as an array of pointers makes it possible to access all three points in these triplets using a single array. Each point not in the tour is implicitly pointed to by its position in the array. The array then holds the value of the first point in the tour which is a member of the triplet. The second point in the tour necessarily follows the first one as the two must be adjacent. For example:

```
dptr[j] = i;
```

The x, y coordinates of the point not in the tour are indicated as (x[j], y[j]). The two adjacent points which complete the triplet are (x[dptr[j]], y[dptr[j]]) and (x[dptr[j] + 1], y[dptr[j] + 1). The distances between any of these three points are accessed in a similar manner using the distance matrix.

3.5 CONVEX HULL, CHEAPEST INSERTION, GREATEST ANGLE PROCEDURE (CCA)

The Convex Hull, Cheapest Insertion, Greatest Angle Procedure uses the same subtour and selection processes as both CICH and CHI. It requires the same additional FOR loop that the CHI procedure requires over CICH to perform the insertion of each point. In the case of CCA the point chosen for insertion into the subtour is the one that forms the greatest angle with the two other points in the subtour selected with it in the selection process. The angle is measured with the point not in the tour between the two points already in the tour and this middle point is inserted between the other two. Only the insertion pseudocode is presented as the subtour and selection procedures are identical to CHI:

```
insertion   MAXANGLE = 0
            FOR j = 1 to n
               IF (pnew[j] is not in the tour)
                  IF (j = n) then jnext = 1
                     ELSE jnext = j + 1
                  IF ((angle(prev[j]], pnew[j],
                             pnext[j]]) > MAXANGLE)
                THEN MAXANGLE =(angle(prev[j],pnew[j]],
                                      pnext[j]])
               INSERT (pnew, prev, pnext)
```

figure 14. Pseudocode for CCA

3.5.1 Implementation of CCA

It was mentioned that CCA is identical to CHI in all but the insertion process. After running through the subtour construction procedure and the selection process an array of

points not yet in the tour is indexed by the locations of each elements position in the x, y coordinate array and the distance matrix. The value in this array points to the first tour element to which the cheapest insertion cost criteria is obeyed. The second element is implicit as it is adjacent to the first. The point which is inserted is the one in which the largest angle formed by the triplet. As with the convex hull subtour creation procedure, Graham's scan, the use of trigonometric functions is avoided during the greatest angle comparison heuristic. Angles are compared using the difference in the slopes of the 2 lines created by the three points in question much the same way this was done in Graham's scan.

3.5.2 CCA Runtime and Memory Requirements

In terms of worst case runtime CCA is the same as CHI and CICH. In practice it is the slowest of the three procedures. This is due to the fact that it is mathematically more intensive than the other two, trigonometric functions are expensive in terms of runtime. When compared to CICH, the fastest of the three procedures, 5n additional additions, 4n additional divides and n additional arithmetic compares are performed during execution of the outer FOR loop.

Space needs are identical to CHI and also CICH if the sorted polar array is used to store the pointers to the triplets as with CHI.

3.6 COMPARISON OF CICH, CHI, AND CCA

A complete analysis of the results of running CICH, CHI, and CCA on various tours taken from the TSPLIB is presented in chapter 5. The following example, also from the TSPLIB, is used to illustrate the differences between the selection heuristics of these three initial tour construction procedures. The instance on which the three procedures are compared is a 100 city problem to which the optimal solution is available.

The convex hull is constructed in the same way for each procedure for the following 100 city tour.

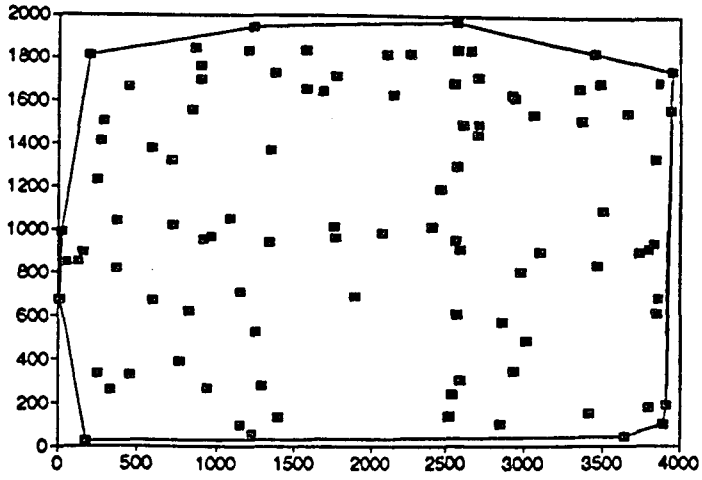


figure 15. convex hull: KROA100

The convex hull contains 12 points. This leaves 88 points to be inserted into the subtour. The following six illustrations show the state of each subtour after 50% of these points have been inserted and then 100% of the points.

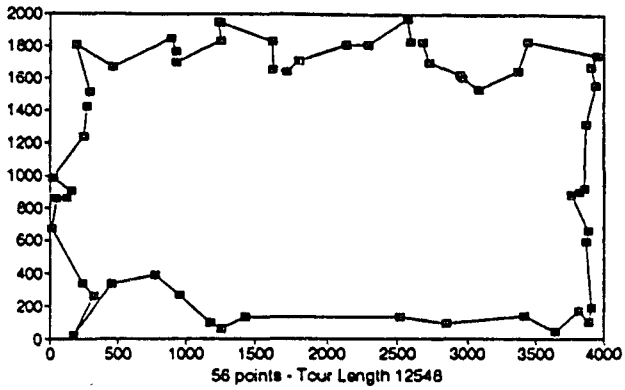


figure 16. CICH 50% of all points inserted, 56 of 100.

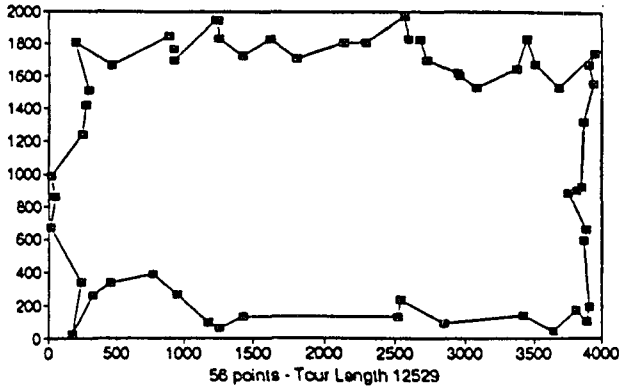


figure 17. CHI 50% of all points inserted.

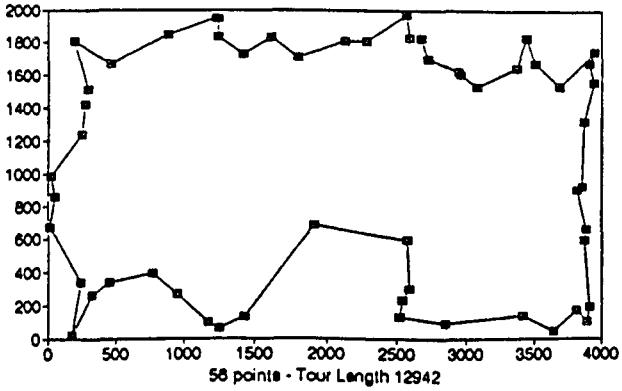


figure 18. CCA 50% of all points inserted.

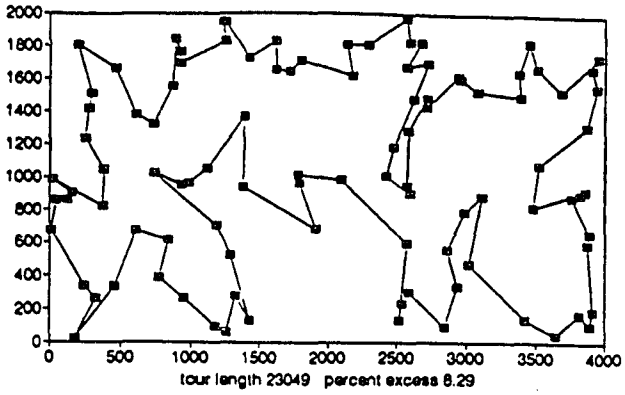


figure 19. CICH complete tour.

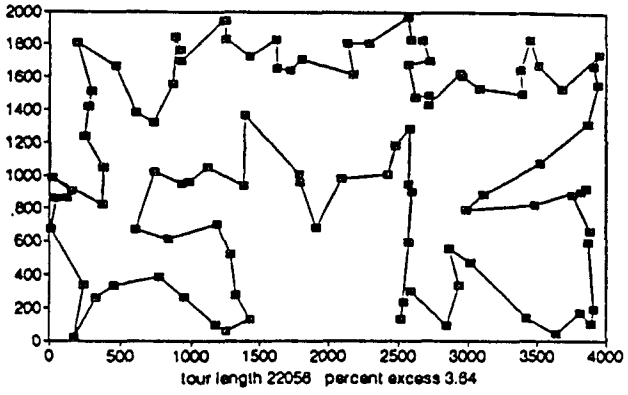


figure 20. CHI complete tour.

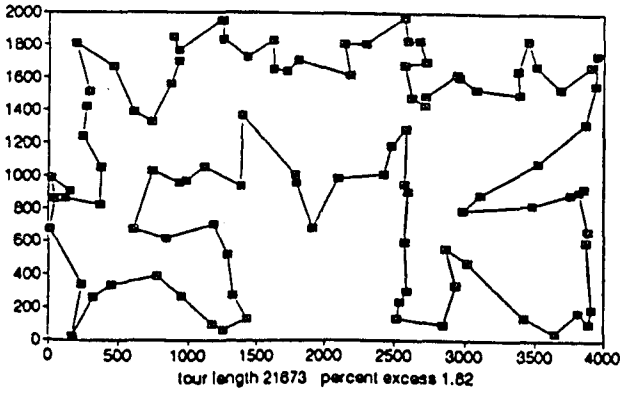


figure 21. CCA complete tour.

The optimal tour is shown below.

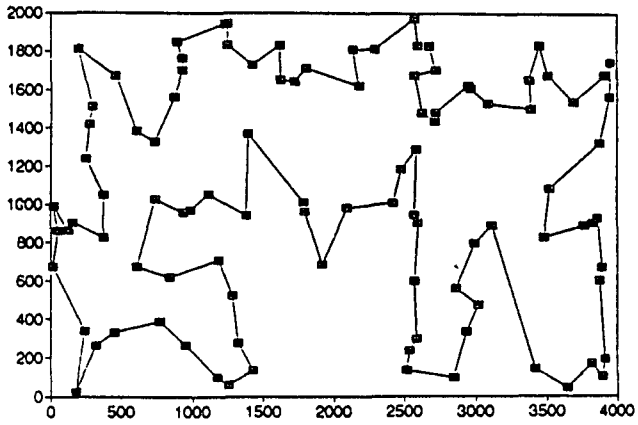


figure 22. Optimal complete tour kroA100 [Lawler].

In the first three figures the 44 of 88 points have been inserted. The tour length of each tour is printed at the bottom of each figure. It is interesting to note how close to optimal each of these tours is for the points it contains. The optimal solution is not available to verify how close each procedure really is to optimal.

In figures 19, 20, and 21 the complete tours are shown along with the percentage excess above the optimal tour which is shown in figure 22. The outcome in terms of tour quality is as expected CCA generates the best tour, CHI is second followed by CICH. Runtime is inversely proportional to tour quality. The three complete tours in this case are almost identical.

CHAPTER 4

DYNAMIC POINT RELOCATION

4.1 DYNAMIC POINT RELOCATION DESCRIPTION

Any insertion procedure can be implemented using Dynamic Point Relocation (DPR) as an enhancement. Dynamic Point Relocation improves tour quality in virtually every instance on which it is used. The increased cost in terms of runtime and space requirements due to DPR is minimal.

DPR is executed during the insertion procedure. The insertion of a point into the subtour eliminates one segment of the tour and replaces it by two new segments, each of which intersects the newly inserted point. The possibility at this point exists that the removal of a previously inserted point and its reinsertion into one of the two new segments may reduce the overall current subtour length.

It was stated in chapter two that the convex hull is an optimal subtour of the points it contains and that the points in the convex hull will occur in that same order in the optimal complete tour. It follows that deviation from optimality occurs during the insertion procedure.

All points which must be inserted into the subtour after the formation of the convex hull are interior to the convex hull. Thus during the insertion process the tour grows inward

or away from the tour boundary defined by the convex hull. After a series of insertions previously distant fragments of the tour may become close to each other and the possibility that points from one fragment may need to be moved to another fragment in order to maintain a condition of optimality in the subtour.

The interesting point is that the heuristics used by insertion algorithms find local optimums. But their ability to ensure a global optimum does not exist. This is the reason an optimization algorithm is employed after execution of the tour construction procedure, to attempt to improve the tour.

Dynamic Point Relocation uses an 'optimize as you go' approach to the TSP. DPR is an optimization type algorithm that executes during the tour construction procedure as opposed to after it as is normally done. The belief is that the longer an optimal tour is maintained the better the eventual complete tour will be. The fact that the convex hull leads to better overall tours than other subtours and that it is the largest optimal subtour lends testimony to this approach

Dynamic Point Relocation is executed after each insertion. Two segments are created by each insertion, one between the newly inserted point and each adjacent point in the tour. If a previously inserted point can be removed from its present position and included in either of these two new segments such that the overall current tour cost is reduced

the point is relocated to this new position.

Cost reduction is determined by simulating the removal of a given point followed by it insertion into both of the new segments. Three configurations exist for comparison: the original one just after the new point has been inserted, that in which the point has been relocated into the first segment, and that in which the point has been relocated into the second segment. The cheapest of the three configurations is maintained. This is illustrated below.

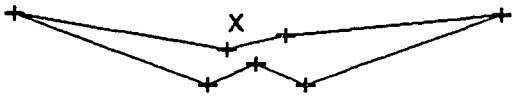


figure 23. original configuration.

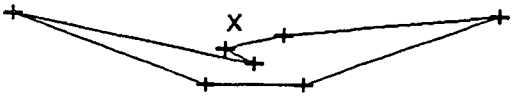


figure 24 relocation within first segment.

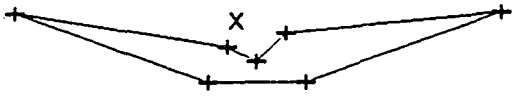


figure 25 relocation within second segment.

The point x has just been inserted. Three of the possible configurations are shown. In this case the third configuration is the shortest tour so this configuration is retained or the point in question will relocated as shown.

DPR continues until all points in the current subtour have been checked in this way.

Pseudocode for DPR follows, the subtour construction and selection processes are omitted as they are not affected:

```
INSERT (p(new), between p(prev) and p(next))

DPR  FOR j = 1 to n
       onext = j + 1
       IF j EQUALS n
       THEN onext = 1
       oprev = j - 1
       IF j = 1
          THEN oprev = n
       IF (COST(p(j), p(new), p(prev)) <
           COST(p(oprev), p(j), p(onext)))
          THEN
             REMOVE(p(j))
             INSERT(p(j) between p(new), p(prev))
       IF (COST(p(j), p(new), p(next)) <
           COST(p(oprev), p(j), p(onext)))
          THEN
             REMOVE(p(j))
             INSERT(p(j) between p(new), p(next))
```

Here p(new), p(prev) and p(next) are the points involved in the latest insertion. P(j) is any previously inserted point. P(oprev) and p(onext) are the points that are adjacent to p(j), one lying on each side of it.

figure 26. Pseudocode for DPR

Essentially what happens is the cheapest insertion heuristic is executed in reverse to simulate the points removal. Then it is executed twice in the normal fashion to simulate insertion of the point into each to the two new segments.

4.1.1 DPR Implementation

Implementation of the subtour creation, selection and insertion processes were discussed in the last two chapters. The tour is stored as an array of pointers to the x, y coordinates of each point which are themselves n element arrays, where n is the number of cities in the problem instance. When a point is inserted its place in the tour is marked by its index in the pointer array. Its x and y coordinates are pointed to by the value in the pointer. Distances between points are also accessed through these same pointers. An array of flags is also maintained in which the roles of the indices and values above pointer array are reversed. The array values point to each tour elements position in the tour and the indices mark the points x and y coordinates in the x, y arrays.

Checking each previously inserted point requires going through the array of pointers once after every insertion and checking to see if any of the points need to be relocated. If a point requires relocation it is removed and then reinserted.

Relocation of a point requires removing it from its place in the array of pointers and reinserting it so that the index of its new location marks its new position in the tour. It also requires changing the value in the flag array to correspond to the points new location in the tour. The use of arrays complicates this process because many points may need

to be moved in either or both of these arrays due to a single relocation.

Care must be taken during both the removal aspect and the replacement aspect to maintain the correct tour. The removal and/or the replacement of a point may involve a the part of the tour that wraps around from the last point to the first. This situation is more complicated than the original insertion. The tour size is increased by one at each insertion. If the newly inserted point happens to be the last point in the tour one of the new segments created due to the insertion involves the first point in the tour. The first point in the tour now advanced two places from the previous end of tour point. The same condition can occur in reverse if the new point is inserted as the first point in the array. Thus a situation can exist where two of the points are wrapped around the end of the tour in one way or the other. Such a situation does not exist in the original insertion process.

4.1.2 Runtime and Memory requirements for DPR

Dynamic point relocation does not require any additional memory allocation. It does increase in the constant term of the worst case runtime as it is executed after each insertion as the additional FOR loop needed to execute Dynamic Point Relocation is part of the insertion procedure which is in the outer FOR loop of the entire tour construction procedure.

The mathematics required to determine whether a point is

to be relocated is minimal. A total of six additions are required to compute the three possible tour configurations and two arithmetic compares determine which of the three is the minimum cost configuration. The effect of these additional operations on CPU usage is discussed in detail in the chapter 5 and tabulated for CICH, CHI and CCA at the end of Appendix 5.

4.2 AN ILLUSTRATION OF DYNAMIC POINT RELOCATION

The difference between CICH, CHI, and CCA was demonstrated using the instance kroA100 in the previous chapter. The same problem instance is used here to illustrate the value of Dynamic Point Relocation.

The following six figures offer a visual means of comparison between three initial tour construction methods, CICH, CHI, CCA, with and without DPR. First with 50% of the tour completed and then with 100% completed. The optimal solution is presented for comparison on page 55.

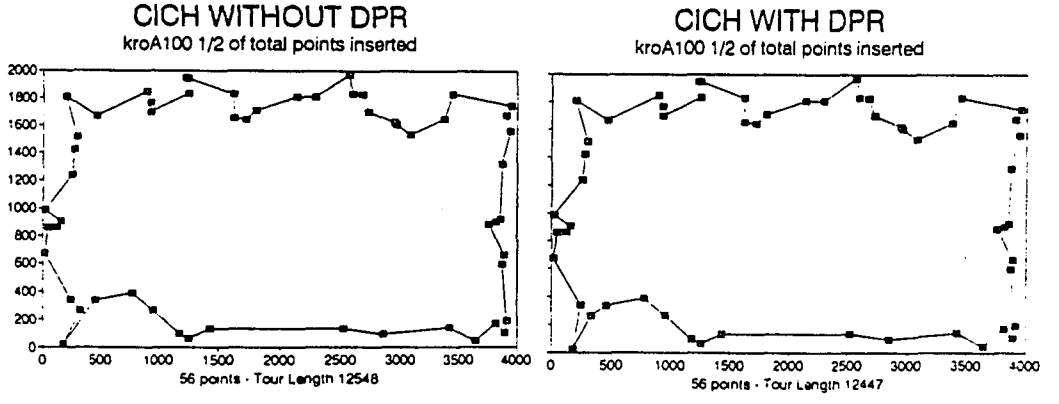


figure 27.CICH 50% of tour completed.

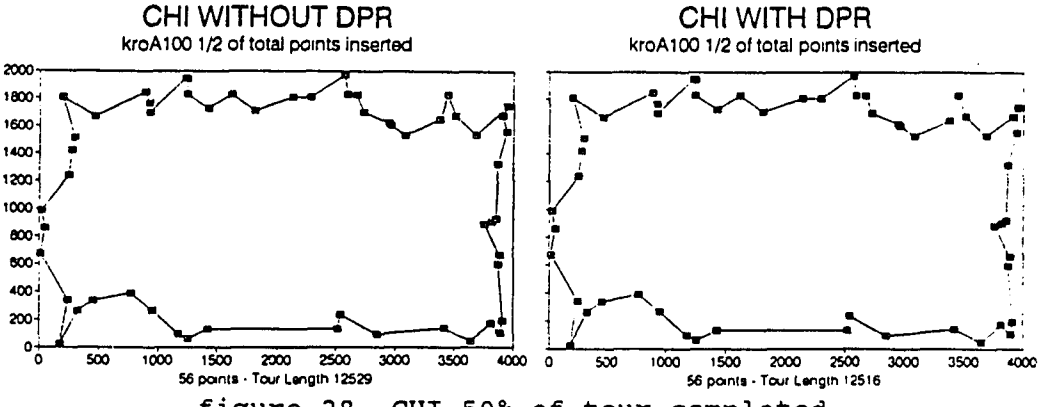


figure 28. CHI 50% of tour completed.

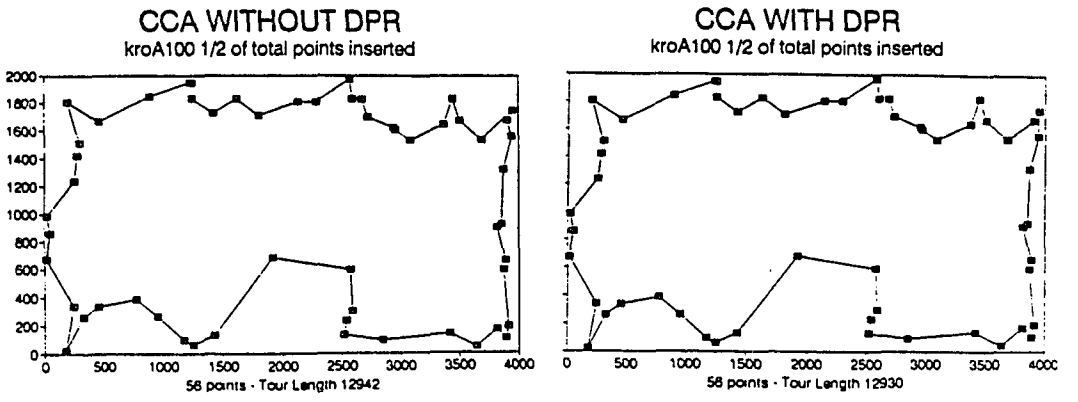


figure 29. CCA 50% of tour completed.

In each of the above cases only a single point has been relocated as half of the total insertions have been made.

The complete tours are presented below along with their respective percentages of excess compared to the optimal tour.

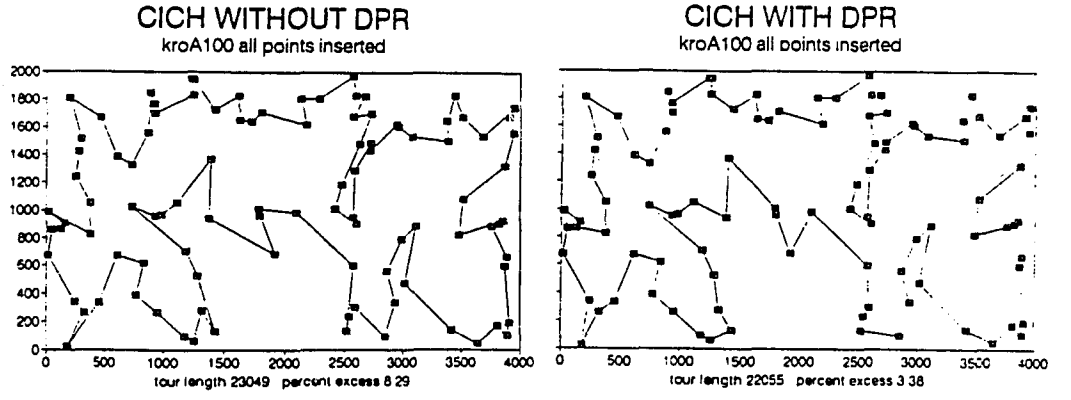


figure 30. CICH complete tour, 11 points moved.

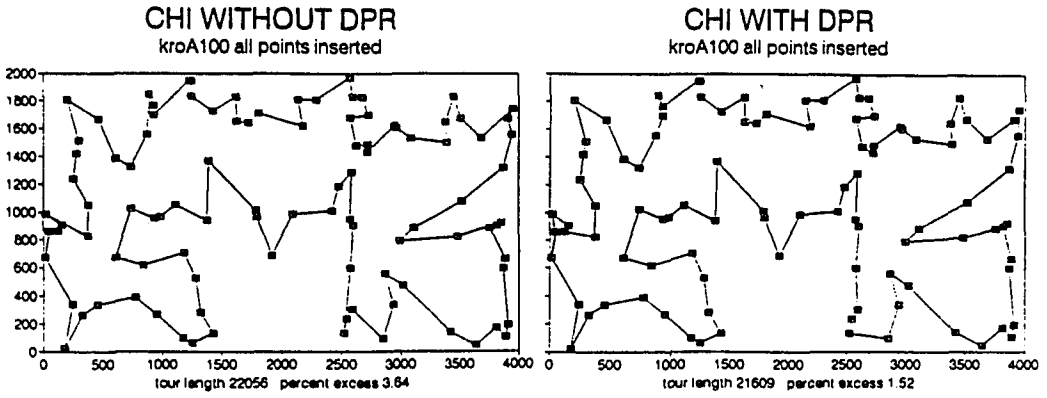


figure 31. CHI complete tour, 8 points moved

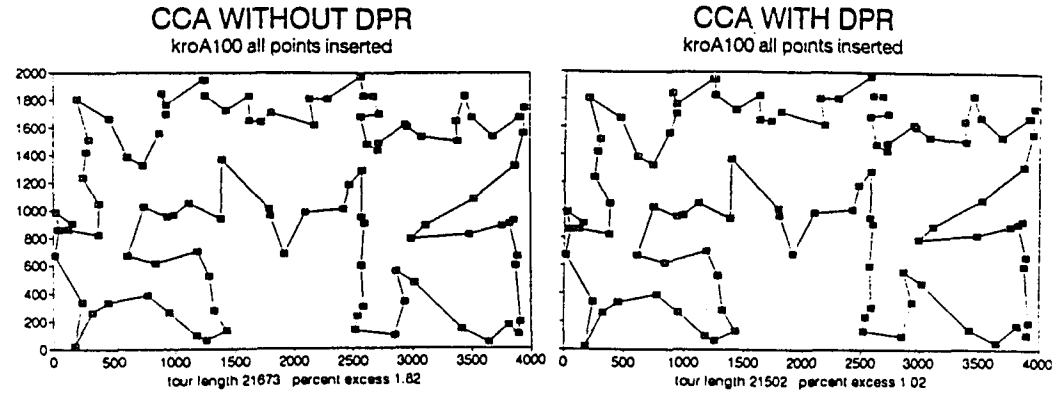


figure 32. CCA complete tour, 3 points moved

The number of points relocated varied with each procedure. Typically the better the initial tour construction procedure the fewer the number of points that will be relocated. This is the pattern in the above example, where in CICH 11 points are relocated, in CHI 8 points are relocated and in CCA just three points are moved. The percentage improvement follows the same pattern. These ideas are discussed in detail in the next chapter.

CHAPTER 5

RESULTS

5.1 EVALUATION CRITERIA

Each of the three procedures used in this investigation, the Cheapest Insertion, Convex Hull Procedure, the Convex Hull Insertion Procedure, and the Cheapest Insertion, Convex Hull, Greatest Angle Procedure was implemented in c programming language both without Dynamic Point Replacement and again with DPR. This was done so that a direct comparison could be made which would show the value of Dynamic Point relocation.

The procedures were evaluated empirically in terms of runtime and in terms of tour quality. CPU usage was used to evaluate the procedures in terms of runtime. Worstcase runtime is also discussed. Overall tour length was used for the comparison of tour quality. These are standard metrics for comparison in cases where an empirical approach is taken to the evaluation process.

There are several reasons why empirical analysis was used rather than probabilistic analysis or worst case analysis. These were discussed in detail in section 1.6.

Each procedure was tested on 37 problem instances taken from the TSPLIB at Rice University once without DPR and then again with the enhancement [Reinelt 1992]. Problem sizes

range from 51 points to 1000 points. Optimal solutions were available to twelve of the problems.

The results of this testing are tabulated in the Appendix. An analysis of tour quality for all problem instances is presented in the first three tables, one for each procedure, followed by a separate analysis for tours to which optimal solutions were available. These results are tabulated in tables 4 through 6. Table 7 contains the CPU usage for each procedure both with DPR and without it.

The first three tables, tables A1-A3, list each instance. The problem size is the numeric part of the problem name. Column 1 contains the tour length, 'TL', for each unenhanced procedure. Column 2 contains the tour length for each procedure with DPR. The difference, 'DIFF', in tour length is listed in column 3. The percentage improvement is listed in column 4 as '% TL IMP' and the number of relocated is displayed in column 5 as 'POINT MOVED'. % TL IMP is explained in detail later.

As was mentioned tables A4 through A6 contain calculations for the tours to which the optimal solution was available. In this case column 1 is the percentage excess above optimal, '% EX', for the unenhanced procedure, column is the percentage excess for the enhanced version, and column 4 is the percentage improvement.

In tables A4 - A6 the percentage improvement is computed as:

$$\% \text{ improvement} = \frac{\% \text{ EXCESS}(x) - \% \text{ EXCESS}(y) \quad * 100}{\% \text{ EXCESS}(x)}$$

where x is the procedure without DPR and y is the procedure with DPR (enhanced).

The denominator term could just have easily been % EXCESS(y). If this were the case the amount of improvement due to DPR would appear to be even more dramatic. For example, 40.8% improvement is stated for eil51. Had we divided by the smaller number, the % excess without the enhancement, the improvement would be stated as 69.2%.

In the case of computation of tour improvement when the optimal tour length is available it is makes more sense to divide by the tour length due to the unenhanced version of the tour construction procedure. The argument follows:

Suppose the unenhanced version of the procedure produced a tour which was 5% longer than optimal and the enhanced version produced a tour that was optimal. If the denominator of the above fraction were the % excess for the enhanced version it would be 0 and the computation of % improvement is erroneous. If the tour length of the unenhanced version were put into the denominator the percent improvement would be 100%, which is correct.

The percentage improvement in tour length due to DPR which is computed all problems regardless as to whether the

optimal solution is available, is done using the same equation as above tourlength is substituted for % excess. Once again the denominator term is with respect to the unenhanced version of the procedure. This performance criteria follows:

$$\% \text{ tour improvement} = \frac{TL(x) - TL(y) * 100}{TL(x)}$$

where x is the procedure without DPR and y is the procedure with DPR (enhanced), TL is tour length , and % tour improvement is used to avoid confusion between this calculation and the one used if the solution was available.

The final table in Appendix A, Table A7, displays the CPU usage calculations. The actual calculation was taken by recording the value returned by the 'time' function before the selection and insertion procedure was executed and then doing the same afterwards. The difference between the two is the CPU usage for the selection and insertion processes. These measurements and calculations are performed for the unenhanced and enhanced versions of each of the three procedures tested. It follows that the difference between CPU usage (enhanced) and CPU usage (unenhanced) is the CPU usage of the enhancement, dynamic point relocation.

5.2 VERIFICATION OF CORRECT IMPLEMENTATION OF PROCEDURES

The correctness of each unenhanced procedure, CICH, CHI, CCA, was verified using eleven instances from the TSPLIB. Performance data with respect to tour length for each of these three procedures was available in [Lawler et al 1985] for these eleven instances. The output generated by the procedures implemented for this research was identical to that found in the literature. This accounts for 1623 points, or 100% of those checked, being placed correctly. It wa assumed that CICH, CHI, and CCA were performing correctly in all cases because of this fact.

Furthermore the convex hull was checked by hand for correctness for all 37 problem instances investigated. This extra care was taken with respect to the convex hull because of the difficulties in implementing a convex hull creation procedure which were discussed earlier. As with the selection/insertion procedures the convex hull creation procedure, Graham's scan, produced a correct convex hull 100% of the time.

Many randomly selected instances were rerun to insure the consistency of the CPU usage measurement.

5.3 PERFORMANCE EVALUATION OF DPR

The average performance of CICH, CHI and CCA was discussed in the introduction with respect to each other and also initial tour construction procedures in general. Of the available tour construction procedures CCA produces the shortest tours on the average for TSP instances of 1000 points or less [Lawler et al 1985]. It has been used repeatedly as a standard for comparison. It is competitive with Christofides' heuristic for instances up to 10000 cities in terms of tour quality measurement. It is also an order of magnitude faster in the worst case. CHI is also an excellent initial tour construction procedure in terms of the quality tours it creates and it is slightly faster than CCA. CICH is faster still and produces reasonably good tours in comparison to most other procedure.

Improvement of CCA, and perhaps CHI as well, in terms of tour quality, without a substantial compromise in terms of runtime, would result in an initial tour construction procedure which produces the best initial tours on the average and remains competitive in terms of runtime.

From Appendix A, the average improvement with respect to percent excess on tours to which the solutions were available was 24.3% for CICH, 25.5% for CHI, and 26.4% for CCA. In comparison the increase is CPU time due to DPR is small. The actual percentage excess is reduced from 5.40% to 4.09% for

CICH, from 4.04% to 3.01% for CHI and from 3.52% to 2.59% for CCA. In only two cases from the above set of 36 comparisons did DPR fail to shorten the tour. These averages are significant in light of the fact that each of the three procedures comes so close to producing the optimal tour on the average without the enhancement. The average improvements for the tours with optimal solutions are summarized below:

HEURISTIC	%EX. WO/DPR	%EX. W/DPR	% IMPROVEMENT
CICH	5.40	4.09	24.3
CHI	4.04	3.01	25.5
CCA	3.52	2.59	26.4

Table 4. Improvement for problems with solutions

Average increase in CPU time for this same set of tours is very small and it is consistent for each individual procedure in terms of problem size. For the five 100 city tours from the above set the average increase in CPU time is 0.07 seconds for CICH, 0.08 for CHI, and 0.18 for CCA. For CICH the average runtime for these five tours is 1.67 seconds. The increase due to DPR amounts to about 4% of the total runtime. For CHI the average runtime for the five tours is 1.84 seconds and the increase due to DPR is 4.1% of total runtime. For CCA .18 seconds on the average is 9.6% of the total average runtime of 1.87 seconds. In comparison to the improvement in tour quality that results form the use of DPR the sacrifice in terms of runtime is almost insignificant.

The percentage improvement in actual tour length, as

discussed in the previous section, is presented in Tables A4-A6. Data is present for all 37 instances on which the three procedures were run, including the tours with solutions. The average improvement due to DPR for CICH is 1.81%, for CHI 1.41%, and for CCA 1.49%.

For the tours with optimal solutions for CICH the average improvement in tour length is 1.36% or 0.45% less than the entire set of all 37 instances. For the tours with optimal solutions CHI the improvement in tour length is 0.91% or 0.50% less than the entire set. For CCA, DPR accounts for as improvement of 0.87% for tours with optimal solutions. This is 0.62% less than the entire set of all tours.

The average problem size for the optimal solution set is 102 cities. The average problem size for the entire set of problems is 258 cities. This may explain why the improvement is slightly larger for the entire set in comparison to the set of instances to which the optimal solutions were available. The results from above discussion are tabulated in the following table.

HEURISTIC	%TL IMP 12	%TL IMP ALL	DIFFERENCE
CICH	1.36	1.81	0.55
CHI	0.91	1.41	0.50
CCA	0.87	1.49	0.62

Table 5. Where %TL IMP 12 is the percentage improvement in tourlength for the tours with optimal solutions. %TL IMP is the percentage improvement for all tours.

The following graph shows the percentage improvement in tour length for each procedure due to DPR with respect to problem size. For problems in the range used for this research there does not appear to be any definite change in the amount of tour length percent improvement due to dynamic point relocation. If anything the performance appears to be more consistent for the larger problems, problems over 300 cities. This is probably due to the fact that more points are moved for the larger problems or that in a small problem a single point relocation has a greater impact on the overall tour length than a single point relocation does on the tour length of a larger problem.

Seven heuristics are listed in table 2 on page 17. The performance of each of these procedures deteriorates as the problems get larger. For problems in the range used in this research, 51-1000 cities, the farthest insertion procedure went from 8.3% to 12.5% above the Held-Karp lower bound. Farthest insertion has comparable performance to CHI in terms of the quality of the tours it constructs.

The fact that the three procedures with DPR used in this research did not show any loss of tour improvement is significant for this reason.

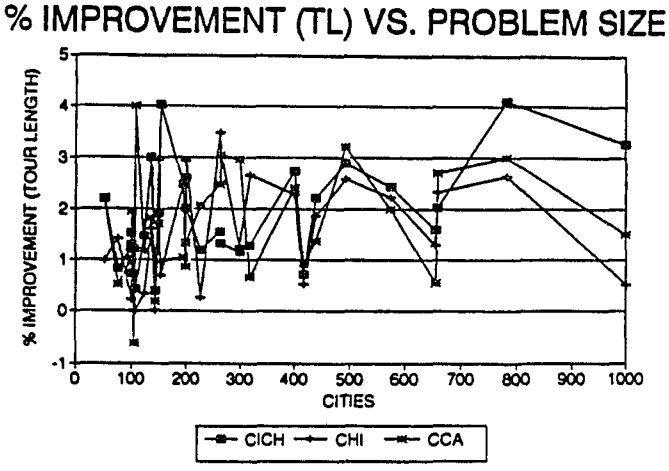


figure 33. % improvement versus problem size

A reasonable explanation for the fact that problem size does not diminish the power of DPR is the fact that as the problem size grows the number of point relocations per problem increases proportionally. This is evident from the graph below. For problems in the range of 51-1000 points the number of relocated points appears to increase linearly as a function of problem size.

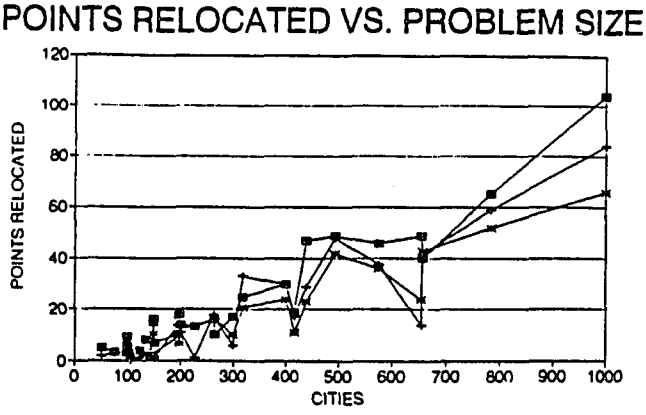


figure 34. % points relocated versus problem size

CPU usage increases linearly with problem size for DPR. This is illustrated by the graph below. This is the expected relationship between CPU time and problem size as DPR increases the constant term of the insertion procedure.

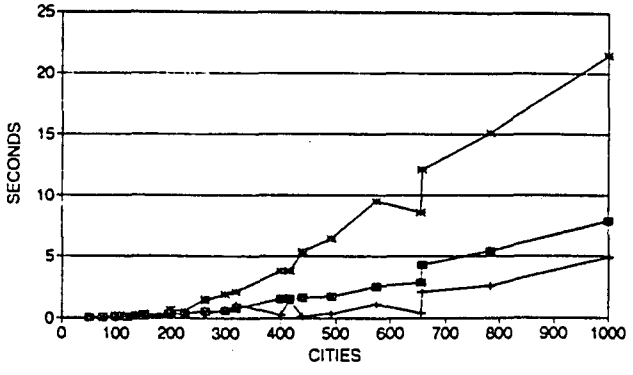


figure 35. CPU usage vs. problem size

It is interesting to note that as problem size increases the percentage of the overall runtime consumed by DPR decreases as problem size increases. For example for the 100 point instances the average percentage increase in CPU time was 4%, 4.1% and 9.6% for CICH, CHI and CCA respectively. For the 1000 city problem these numbers are reduced to .5%, .3% and 1.2% for the same procedures. % CPU usage due to DPR with respect to the overall CPU usage to run the entire procedure has been reduced by a factor of 10 as the problem size has been increased by the same factor. The following graph illustrates the relationship between % CPU usage for DPR with respect to the overall CPU usage of the entire selection and

insertion procedure.

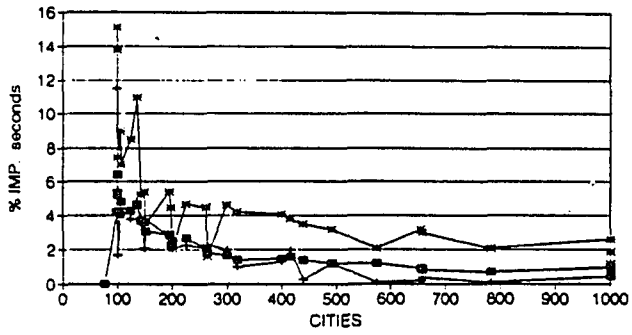


figure 36. % CPU usage DPR/entire proc. * 100

CHAPTER 6

CONCLUSIONS

6.1 PERFORMANCE EVALUATION

The Traveling salesman problem is an NP 'hard' optimization problem thus no known algorithm has been discovered which can find exact solutions to TSP instances in polynomial time. The large number of NP 'hard' problems as well as the wide range of applications of such problems has provided tremendous motivation to develop new solving techniques for this type of problem.

Practical TSP solving procedures require a compromise between solution quality and algorithm runtime. Perhaps the most successful approach in this respect has been to create a traveling salesman initial tour and then employ some form of local optimization to attempt to improve this initial tour.

This approach suffers from the fact that during the initial tour construction phase insertions are made without any concern for global optimality. This may result in a series of poorly inserted points, where each subsequent insertion builds on the suboptimality of the previous insertion. In such a case the local optimization procedure may be faced with an initial tour that is so far from optimal that it will not be able to find the optimal solution in any reasonable amount of time.

The procedure introduced in this document, Dynamic Point Relocation, combines the initial tour construction phase with the optimization phase. This procedure seeks to maintain a subtour as near to optimal as possible during initial tour construction and thus avoid the pitfalls of the above mentioned two phase methods.

In terms of both tour quality and runtime insertion type initial tour construction procedures are among the best available. Dynamic Point Relocation can be attached to any such procedure.

Three insertion procedures were used in this research to evaluate the effect of DPR on algorithm runtime and tour quality. The convex hull cheapest insertion procedure (CICH) is among the fastest of the insertion type procedures. The convex hull, cheapest insertion, greatest angle procedure is among the best such procedures in terms of the quality of the tour it constructs. The convex hull insertion procedure is between these two procedures both in terms of runtime and tour quality. The convex hull was used as a subtour because it leads to better overall tours virtually every time it is used.

CICH, CHI, and CCA were used in the evaluation of DPR as they provided the most difficult environment to improve upon on the one hand with respect runtime (CICH), on the other in terms of tour quality CCA) and lastly in terms of a tradeoff between the two (CHI).

Each procedure was implemented in c in one case with DPR

and in the other case without the enhancement. The procedures were evaluated empirically on 37 ETSP problems taken from TSPLIB at Rice University. This provided a basis to verify the correctness of each implementation and also a way to evaluate the importance of DPR exactly in comparison to many other procedures which have been tested on the same problem instances.

The added runtime due to DPR is minimal in comparison to the runtime of the initial tour construction procedure as a whole. This is true even for fast initial tour construction procedures such as Cheapest Insertion, Convex Hull. In terms of worst case analysis DPR increases only the constant term.

The effect of DPR was evaluated empirically in terms of runtime as well. This evaluation revealed that the percentage of the insertion procedure due to DPR is inversely proportional to the problem size for $O(n^2)$ initial tour construction procedures. Even on small instances the overall increase in runtime is almost insignificant. For example the average increase in runtime for the six 100 city problems in the set was less than 5% for CICH and CHI and less than 10% for CCA. For the 1000 city instance this percentage was reduced by a factor of ten.

In contrast to its effect on runtime DPR's effect on tour quality is substantial. Each of the three insertion procedures were improved by approximately 25% in terms percent excess for the 12 optimal tours they were tested on.

For CICH the improvement was 24.3%, for CHI 25.5% and for CCA the improvement was 26.4%. This is substantial as each of these procedures produced nearly optimal tours without DPR. The average percentage excess for the 12 tours was 5.4% for CICH, 4.04% for CHI and 3.52% for CCA.

An evaluation of each procedure was also performed in terms of the percentage improvement in tour length due to the use of DPR. It was discovered that for problems between 51 and 1000 points the power of the enhancement remains roughly constant. It does not diminish as the problem sizes are increased. This is particularly important because most initial tour construction procedures do show a reduced ability to construct quality tours as problem size increases.

The performance evaluation of each procedure on the problems with solutions may be extended to the entire set of 37 TSP instances. This extrapolation of performance is possible because of the consistency demonstrated by each procedure on all problems. Separate analysis of the 12 optimal tour problems in terms of percent improvement in tour length revealed that DPR performed at the same level on these 12 problems as it did on all 37 problems in the set.

Dynamic Point Replacement improved the best available traveling salesman tour insertion procedure, CCA, by over 25% on the average. It did not appreciably increase the runtime of CCA or of two similar procedures which makes sacrifices in tour quality in favor of decreased runtime. One of these two

procedures CICH is among the fastest available.

The fundamental difference between insertion procedures which have been enhanced by Dynamic Point Relocation and those that have not is that the optimization phase of tour construction is combined with the initial tour construction phase. This research has demonstrated the benefits of taking such an approach to the traveling salesman problem. It is likely that this same approach to any NP 'hard' optimization problem will produce similar results.

6.2 PROPOSED ENHANCEMENTS

6.2.1 Superhull

The procedure introduced in this research attempts to maintain an optimal or near optimal tour as long as possible during the tour construction process. The convex hull is optimal for the points it contains and these points are ordered in the same way in which they will be ordered in the optimal tour. What's more the convex hull creation procedure has faster runtime than the insertion procedure which follows it to create a complete tour.

If a larger set of points could be used as a initial subtour which retained all the desirable properties of the convex hull the overall initial tour construction procedure would be speeded up and very likely an improvement in tour quality would result.

Graham's scan could easily be altered to perform as described above. Points which form a non-left turn as they are visited during the scan are popped off the convex hull stack by Graham's scan. This could be expanded to include points which come close to forming a non-left in comparison to the distance between the points. This would insure that the same properties exhibited by the convex hull are retained but very likely the set of points in this superhull would be larger than the set of points in the convex hull.

6.2.2 Dynamic Fragment Relocation

During the insertion phase of initial tour construction previously inserted points which are out of place can be moved after each insertion by DPR. Though this improves the tour construction procedures ability to construct a quality tour the results indicate that it does not insure that an optimal tour is maintained. In fact the closest tour to optimal constructed in this research was a 100 point problem in which the tour produced was 1.02% from optimal.

Dynamic fragment relocation would relocate tour segments or fragments after each insertion which were not in the proper place. The negative effect of this would again be increased runtime.

Some initial attempts to implement dynamic segment relocation (DSR) were attempted. But the enhancement has not been perfected at this time. A weaker than what is possible version of DSR was attached to CCA with DPR already attached and run on the 12 problems with optimal solutions. Two of the tours were improved, one of them significantly. The runtime increase appears to be about the same as the increase that results from DPR. Interestingly while the number of points moved by DPR is relatively small in comparison to problem size the number of segment moved by DSR is quite large.

No conclusions can be drawn about the usefulness of DSR at this point but it would seem likely that some form of DSR would increase the power tour construction procedures.

6.2.3 Greatest Angle as point relocation heuristic

The implementation of DPR used in this research used the cheapest insertion heuristic to determine which points to relocate. Of the three procedures used to test DPR the cheapest insertion heuristic is the fastest but also the weakest in that the tour produced from its use are the lowest quality. It seems very likely that use of an angle comparison, which is the strongest of the three heuristics, would produce better tours.

6.2.4 Software Improvements

Arrays were used to store the tour, the x and y coordinates and the distance matrix. Experiments have been done using various several different types of tree data structures which speed up runtime in various ways and facilitate the implementation of useful operations such as fixed radius searching and nearest neighbor searching.

Dynamic fragment relocation would require wrapping around the ends of the tour arrays to more than one point. If FDR were to be implemented it would require that some other data structure be used to store the various tour components.

APPENDIX

TABLE A1: CICH-CICHDPR TOUR LENGTH COMPARISON

INSTANCE	CICH TL	CICHDPR TL	DIFF	% TL IMP	POINTS MOVED
eil51	454	444	10	2.20	5
eil76	579	575	4	0.69	6
eil101	678	673	5	0.73	9
kroA100	23049	22005	1044	4.52	11
kroB100	23247	23036	211	0.90	4
kroC100	21632	21513	119	0.55	8
kroD100	21711	21646	65	0.29	1
kroE100	22870	22827	43	0.18	4
rd100	8465	8232	233	2.75	9
lin105	14913	14730	183	1.22	3
lin318	46904	46301	603	1.28	25
pr76	114808	113673	135	0.98	1
pr107	45730	45533	197	0.43	1
pr124	62193	61273	920	1.47	4
pr136	102696	99615	3081	3.00	8
pr144	60625	60378	247	0.40	2
pr152	79952	76716	3236	4.04	7
pr226	83664	82667	997	1.19	13
pr264	53416	52702	714	1.33	10
pr299	52896	52276	620	1.17	17
pr439	120679	117980	2699	2.23	23
rat99	1283	1267	16	1.24	3
rat195	2568	2504	64	2.49	10
rat575	7694	7505	189	2.45	46
rat783	10211	9792	419	4.10	65
kroA150	28814	28138	676	2.34	14
kroB150	27476	27061	415	1.51	18
kroA200	31792	31079	713	2.24	14
kroB200	32123	31566	557	1.73	14
gil262	2680	2638	42	1.56	16
d198	17045	16603	442	2.59	18
d493	39013	37878	1135	2.90	47
d657	56263	55112	1151	2.04	40
fl417	12703	12611	92	0.72	19
p654	37089	36495	595	1.60	49
rd400	17146	16675	471	2.74	30
dsj1000	21901538	21183212	718326	3.27	104
AVG IMPROVEMENT				1.81 %	

CICH = Cheapest Insertion Convex Hull
CICHDPR = Cheapest Insertion Convex Hull With DPR
TL = tourlength DIFF = actual change in length
% TL IMP = % improvement in tour length due to DPR
POINTS MOVED = relocated points due to DPR

TABLE A2: CHI-CHIDPR TOUR LENGTH COMPARISON

INSTANCE	CHI TL	CHIDPR TL	DIFF	% TL IMP	POINTS MOVED
eil51	444	440	4	0.99	2
eil76	577	572	5	0.87	2
eil101	665	656	9	1.35	5
kroA100	22056	21609	447	2.03	8
kroB100	22700	22676	23	0.11	4
kroC100	21276	21110	176	0.83	5
kroD100	21794	21794	65	0.30	1
kroE100	22830	21729	30	0.13	3
rd100	8327	8253	75	0.89	6
lin105	15207	15093	114	0.75	3
lin318	47055	45805	1250	2.66	33
pr76	112480	110264	2216	1.97	4
pr107	49295	49295	0	0.00	0
pr124	61627	61418	209	0.34	1
pr136	102964	101307	1657	1.61	2
pr144	60684	60683	1	0.02	1
pr152	76111	75580	531	0.70	1
pr226	83602	83385	217	0.26	1
pr264	54887	53211	1676	3.05	16
pr299	50623	49978	645	1.27	6
pr439	118550	116329	2221	1.87	29
rat99	1259	1256	3	0.24	2
rat195	2556	2492	64	2.50	14
rat575	7427	7261	166	2.24	38
rat783	9689	9432	257	2.65	59
kroA150	28564	27289	1275	4.46	11
kroB150	27132	26733	399	1.47	16
kroA200	30944	30594	349	1.13	12
kroB200	31486	30920	566	1.80	10
gil262	2669	2576	93	3.48	18
d198	16280	16134	146	0.90	7
d493	37522	36550	972	2.59	48
d657	53485	52233	1252	2.34	42
fl417	12086	12022	84	0.53	17
p654	35786	35322	464	1.30	14
rd400	16837	16452	385	2.29	30
dsj1000	20247896	20141810	106086	0.52	84
AVG IMPROVEMENT				1.41 %	

CHI = Convex Hull Insertion Procedure
CHIDPR = Convex Hull With DPR
TL = tourlength DIFF = actual change in length
% TL IMP = % improvement in tour length due to DPR
POINTS MOVED = relocated points due to DPR

TABLE A3: CCA-CCADPR TOUR LENGTH COMPARISON

INSTANCE	CCA TL	CCADPR TL	DIFF	% TL IMP	POINTS MOVED
eil51	453	443	10	2.21	5
eil76	580	577	3	0.52	3
eil101	670	657	13	1.94	6
kroA100	21673	21502	171	0.79	3
kroB100	22440	22410	30	0.13	4
kroC100	21225	20983	242	1.14	6
kroD100	21939	21877	62	0.28	1
kroE100	23071	22722	349	1.51	5
rd100	8357	8213	144	1.72	3
lin105	14724	14815	-91	-0.62	4
lin318	45031	44728	303	0.67	21
pr76	110670	110438	232	0.21	3
pr107	46768	44896	1872	4.00	6
pr124	61406	60682	724	1.18	3
pr136	101360	99517	1843	1.82	2
pr144	61418	61298	120	0.20	1
pr152	77600	76845	755	0.97	3
pr226	84106	82369	1737	2.07	13
pr264	54887	53211	1676	3.05	16
pr299	50695	49187	1508	2.97	10
pr439	114753	113173	1580	1.38	23
rat99	1276	1275	1	0.08	3
rat195	2494	2468	26	1.04	9
rat575	7453	7304	149	2.00	36
rat783	9627	9335	292	3.03	52
kroA150	27628	27126	502	1.82	11
kroB150	27208	26768	440	1.62	9
kroA200	30576	30412	164	0.54	13
kroB200	30945	30292	566	2.11	12
gil262	2571	2507	64	2.49	16
d198	16232	16093	139	0.87	7
d493	37696	36483	1213	3.22	42
d657	53033	51591	1442	2.72	43
fl417	12327	12212	115	0.93	11
p654	35882	35680	202	0.56	24
rd400	16519	16120	399	2.42	24
dsj1000	20132502	19828792	303710	1.51	66
AVG IMPROVEMENT				1.49 %	

CCA = Cheapest Insertion Convex Hull Greatest Angle

CCADPR = CCA With DPR
TL = tourlength DIFF = actual change in length
% TL IMP = % improvement in tour length due to DPR
POINTS MOVED = relocated points due to DPR

TABLE A4: CICH-CICHDPR % EXCESS COMPARISON

INSTANCE	CICH % EX	CICHDPR % EX	DIFF	% EX IMP	POINTS MOVED
eil51	5.65	3.34	2.31	40.8	5
eil76	6.15	5.42	0.73	11.9	6
eil101	5.71	5.34	0.37	6.5	9
kroA100	8.29	3.38	4.91	59.2	11
kroB100	5.00	4.54	0.46	9.2	4
kroC100	4.25	3.67	0.58	13.6	8
kroD100	1.95	1.65	0.20	10.2	1
kroE100	4.06	3.63	0.43	10.6	4
rd100	7.01	4.07	2.94	41.9	9
lin105	3.68	2.41	1.27	34.5	3
lin318	6.93	5.55	1.38	19.9	25
pr76	6.15	5.96	0.21	0.3	1
AVERAGES	5.40	4.09	1.31	24.3%	8

CICH = Cheapest Insertion Convex Hull Procedure
CICHDPR = Cheapest Insertion Convex Hull with DPR
% EX = percent excess above optimal
DIFF = actual change in % EX due to DPR
% EX IMP = the % improvement in excess due to DPR
POINTS MOVED = relocated points due to DPR

TABLE A5: CHI-CHIDPR % EXCESS COMPARISON

INSTANCE	CHI % EX	CHIDPR % EX	DIFF	% EX IMP	POINTS MOVED
eil51	3.36	2.45	0.91	27.1	2
eil76	5.74	4.80	0.94	16.3	2
eil101	3.48	2.21	1.27	36.4	5
kroA100	3.64	1.52	2.12	58.2	8
kroB100	2.52	2.42	0.10	4.0	4
kroC100	2.53	1.68	0.85	33.6	5
kroD100	2.35	2.04	0.31	13.1	1
kroE100	3.45	3.32	0.13	3.7	3
rd100	5.26	4.32	0.94	17.9	6
lin105	5.73	4.93	0.80	14.0	3
lin318	6.51	4.42	2.09	32.1	33
pr76	3.99	1.95	2.04	51.1	4
AVERAGES	4.04	3.01	1.03	25.5%	7

CHI = Convex Hull Insertion Procedure
CHIDPR = Convex Hull Insertion with DPR
% EX = percent excess above optimal
DIFF = actual change in % EX due to DPR
% EX IMP = the % improvement in excess due to DPR
POINTS MOVED = relocated points due to DPR

TABLE A6: CCA-CCADPR % EXCESS COMPARISON

INSTANCE	CCA % EX	CCADPR % EX	DIFF	% EX IMP	POINTS MOVED
eil51	5.34	3.06	2.28	42.6	5
eil76	6.44	5.75	0.69	10.7	3
eil101	4.33	2.34	1.99	46.0	6
kroA100	1.82	1.02	0.80	44.0	3
kroB100	1.35	1.21	0.14	10.4	4
kroC100	2.28	1.12	1.16	50.9	6
kroD100	3.03	2.74	0.29	9.6	1
kroE100	4.54	2.96	1.58	34.8	5
rd100	4.38	3.82	0.56	12.8	3
lin105	2.37	3.00	*0.63	-26.5	4
lin318	4.03	1.97	2.06	51.1	21
pr76	2.32	2.10	0.22	9.5	3
AVERAGES	3.52	2.59	0.93	26.4%	6

CCA = Cheapest Insertion Convex Hull Greatest Angle
CCADPR = CCA with DPR
% EX = percent excess above optimal
DIFF = actual change in % EX due to DPR
% EX IMP = the % improvement in excess due to DPR
POINTS MOVED = relocated points due to DPR

TABLE A7: CICH, CHI, AND CCA CPU TIMES IN SECONDS

INSTANCE	CICH	CICHDPR	CHI	CHIDPR	CCCA	CCADPR
eil51	0.23	0.24	0.26	0.29	0.28	0.33
eil76	0.71	0.77	0.82	0.87	0.87	1.01
eil101	1.65	1.73	1.83	1.91	1.94	2.16
kroA100	1.64	1.69	1.82	1.84	1.86	2.09
kroB100	1.59	1.68	1.74	1.84	1.86	1.98
kroC100	1.61	1.68	1.78	1.87	1.90	2.09
kroD100	1.59	1.67	1.73	1.82	1.84	2.05
kroE100	1.59	1.66	1.75	1.83	1.87	2.03
rd100	1.63	1.67	1.77	1.85	1.89	2.16
lin105	1.78	1.86	2.01	2.09	2.14	2.34
lin318	49.57	50.33	52.85	53.93	53.33	55.42
pr76	0.75	0.79	0.83	0.87	0.90	1.03
pr107	1.67	1.75	1.80	1.89	1.87	1.99
pr124	2.33	2.42	2.48	2.58	2.55	2.69
pr136	3.74	3.87	4.07	4.21	4.24	4.48
pr144	4.29	4.43	4.50	4.68	4.72	4.89
pr152	5.31	5.47	5.72	5.88	5.95	6.23
pr226	13.84	14.14	14.26	14.59	14.43	14.96
pr264	28.35	28.83	30.39	30.69	30.99	32.34
pr299	41.03	41.62	43.69	44.42	44.25	46.13
pr439	130.36	131.91	141.28	141.41	140.82	146.17
rat99	1.52	1.61	1.78	1.81	1.88	2.03
rat195	11.45	11.72	12.84	13.05	13.07	13.77
rat575	291.80	294.28	318.19	319.12	311.39	320.93
rat783	754.44	744.62	789.20	791.73	781.98	797.05
kroA150	5.26	5.42	5.72	5.89	6.04	6.39
kroB150	5.32	5.46	5.88	6.03	6.14	6.49
kroA200	12.48	12.86	13.69	14.04	14.23	14.88
kroB200	12.36	12.69	13.26	13.65	13.61	14.28
gil262	27.89	28.41	30.61	31.23	31.15	32.67
d198	11.93	12.23	13.16	13.47	13.35	14.02
d493	184.32	186.02	199.68	200.06	197.56	204.02
d657	436.34	440.64	465.73	467.83	460.44	472.57
fl417	106.39	107.93	113.64	114.06	112.94	116.73
p654	390.73	393.57	409.46	410.10	400.24	408.83
rd400	98.55	100.12	106.56	106.83	105.70	109.51
dsj1000	1560.88	1568.75	1646.08	1651.01	1650.22	1659.94

REFERENCES

Balas E. & Christofides N. [1981]. "A restricted lagrangean approach to the traveling salesman problem", Math. Prog. 21, 19-46.

Bland R.G. & Shallcross D.F. [1989]. "Large traveling salesman problems arising from experiments in X-ray crystallography: A preliminary report on computation", Oper. Res. Letts. 8, 125-128.

Bentley J.L. [1990a]. "Fast algorithms for geometric traveling salesman problems", Computing Science Technical Report No. 151, AT&T Bell Labs. Murray Hill N.J., 35-74.

Bentley J.L. [1990b]. "K-d trees for semidynamic point sets", Computing Science Technical Report No. 151, AT&T Bell Labs. Murray Hill N.J., 17-34.

Bentley J.L. [1990c]. "Experiments on traveling salesman heuristics", Computing Science Technical Report No. 151, AT&T Bell Labs. Murray Hill N.J., 3-16.

Carpenato G. & Toth P.[1980]. "Some new branching and bounding criteria for the asymmetric travelling salesman problem", Management Science 26, 736-743.

Christofides N. [1977]. "Worst case analysis of a new heuristic for the traveling salesman problem", Report 388, Graduate College of Industrial Administration.

Cook S.A. [1971]. "On the complexity of theorem proving procedures", Proc. 3rd Ann. ACM Symp. Theory and Comp., 150-151. Carnegie-Mellon University, Pitts. Pa.

Crowder H. & Padberg M.W.[1980]. "Solving large-scale traveling problems to optimality", Man. Sci. 26, 495-509.

Dantzig G. B., Fulkerson D.R. & Johnson, S.M. [1954]. "Solution of a large scale traveling-salesman problem. Oper. Res. 2, 393-410.

Dueck G. [1989]. "Threshold Accepting: A general purpose optimization algorithm appearing superior to simulated annealing", TR88.10.011, Heidelberg Scientific Center, IBM Germany.

Durbin R. & willshaw D. [1987]. "An analogue approach to the travelling salesman problem using the elastic net method", Nature 326, 689-691.

Flood M.M. [1956]. "The traveling salesman problem", Oper. Res. 4, 61-75.

Fredman M.L., Johnson D.S., McGeoch L.A. & Ostheimer G. [1993]. "Data structures for traveling salesmen", Submitted to 1993 SODA conference.

Garey , M. R. & Johnson, D. S. [1979]. Computers and Intractability: A Guide to the Theory of Np-completeness, Freeman, Sanfrancisco.

Graham R.L. [1972]. "An efficient algorithm for determining the convex hull of a finite polar set", Info. Proc. Lett. 1, 132-133.

Glover F. [1989]. "Tabu search - Part 1", ORSA J. Computing, 1, 190-206.

Held M. & Karp R.M. [1971]. "The traveling salesman problem and minimum spanning trees: Part 2", Math. Prog. 1, 6-25.

Hopfield J.J. & Tank D.W. [1985]. "Neural Computation of decisions in optimization problems", Biol. Cybern. 52, 141-152.

Johnson D. S. [1990]. "Local optimization and the traveling salesman problem", Proc.Seventeenth Colloquium on Automata Languages and Programming, Springer-Verlag, 446-461.

Johnson D.S., Aragon, McGeoch & Schevon [in preparation]. Received via private communication.

Karp R.M. [1977]. "Probabilistic Analysis of Partitioning Algorithms for the traveling salesman problem in the plane", Math. Oper. Res. 2, 209-224.

Korte B. [1988]. "Applications of combinatorial optimization", talk at the 13th Int. Math. Prog. Symp., Tokyo.

Krolak P.D., Felts W. & Marble G. [1971], "A man machine approach toward solving the traveling salesman problem", Comm. ACM 14 327-334.

Kruskal, Jr. J.B. [1956]. "On the shortest spanning subtree of a graph and the traveling salesman problem", Proc. Amer. Math. Soc. 7, 48-50.

Laporte G.[1992]. "The traveling salesman problem: An overview of exact and approximate algorithms", Euro. Jour. Oper. Res. 59, 231-247.

Lawler E. L. et al [1985]. The Traveling Salesman Problem, John Wiley & Sons, Chichester.

Lin S. [1965]. "Computer solutions of the traveling salesman problem", Bell System Computer Journal 44, 2245-2269.

Lin S. & Kerninghan B.W. [1973]. "An effective heuristic algorithm for the traveling salesman problem", Oper. Res. 21, 498-516.

Litke J.D. [1984]. "An improved solution to the traveling salesman problem with thousands of nodes", Comm. ACM 27 , 1227-1236.

Miller D.L. & Pekny J.F. [1991]. "Exact solution of large assymmetric traveling salesman problems", Science 251, 754-761.

Miller, Tucker & Zemlin [1960]. "Integer programming formulations and thraveling salesman problem", Jour. ACM, 7, 326-329.

Norback J.P. & Love R.F. [1971], "Geometirc approaches to solving the traveling salesman problem", Man. Sci. 23, 1208-1223.

Platzman L.K. & Bartholdi III, J.J. [1989], "Spacefilling Curves and the planar traveling salesman problem", J. Assoc. Comp. Mach. 36, 719-737.

Preparata & Shamos [1985]. Computational Geometry An Introduction, Springer-Verlag, NY, NY.

Prim R.C. [1957]. "Shortest connection networks and some generalizations", Bell Systems Tech. J. 36 1389-1401.

Reinelt G. [1991]. "TSPLIB - A traveling salesman problem library", ORSA J. Comp. Vol.3 No.4, fall, 376-384.

Reinelt G. [1992]. "Fast heuristics for large geometric traveling salesman problems", ORSA J. Comp. 4, 206-217.

Stewart, jr. W.R. [1971] "A computationally efficient heuristic for the traveling salesman problem", Proc. 13th Ann. S.E. TIMS 75-85.

Stieglitz K. & Weiner P. [1968]. "Some improved algorithms for computer solutions of the traveling salesman problem", Proc. 6th Ann. Allerton Conf. on Cir. The., 814-821.