AN ANALYSIS OF VARIOUS ASPECTS OF THE TRAVELING SALESMAN PROBLEM

by

Selim G. Akl

A thesis submitted to the Faculty of Graduate Studies and Research
in partial fulfillment of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

School of Computer Science
McGill University
Montreal, Quebec, Canada

March 1978

AN ANALYSIS OF VARIOUS ASPECTS OF THE TSP

ABSTRACT

Various aspects of the Traveling Salesman Problem (TSP) are studied. For the Euclidean TSP an enumerative algorithm is presented that yields the optimal solution to problems which have a small number of cities. This algorithm is utilized to obtain empirical estimates of the expected number of feasible solutions. Approximate solutions to the Euclidean TSP are calculated based on triangulations of points in the plane. When the problem is not necessarily Euclidean, but still symmetric, two heuristic algorithms are described that use graph-theoretic techniques to reach sub-optimal solutions for large problems. One of the two algorithms is modified to work for the asymmetric case. Two topics related to the TSP are also examined in some detail: triangulations and convex hulls. Maximal triangulations are studied in connection with optimal drawings and Hamiltonian graphs. An efficient convex hull algorithm is presented whose asymptotic expected run-time is linearly proportional to the size of the input for uniform distributions on the square.

RESUME

Divers aspects du problème du commis voyageur (PCV) sont étudiés. Pour le PCV Euclidien (PCVE), une méthode d'énumération est présentée qui obtient la solution exacte aux problèmes où le nombre de villes est petit. Cette même technique est exploitée pour calculer des estimés empiriques de l'espérance mathématique du nombre de solutions possibles. Des solutions approximatives au PCVE sont obtenues à partir de triangulations de points dans le plan. Dans le cas de problèmes symétriques, mais pas nécessairement Euclidiens, deux algorithmes heuristiques sont décrits qui utilisent des principes de la théorie des graphes pour arriver à des solutions sous-optimales aux problèmes de grande dimension. L'un des deux algorithmes est modifié et appliqué aux problèmes non-symétriques. Enfin deux sujets reliés au PCV sont examinés en plus de détail: les triangulations et l'enveloppe convexe (EC). La relation entre les triangulations maximales et les dessins optimaux d'une part, et les graphes Hamiltoniens de l'autre est étudiée. Un algorithme efficace est présenté pour l'EC pour lequel l'espérance mathématique du temps d'exécution varie linéairement avec le nombre de points donné pour des distributions uniformes dans le carré.

ACKNOWLEDGEMENTS

Throughout the preparation of this thesis Professor Monroe Newborn provided me with his inspiring guidance and unfailing support. Being able to express in writing my great indebtedness to him is a true privilege.

I also would like to thank Professor Godfried Toussaint who gave me permission to reproduce here a piece of our joint research. Working with him has constantly been to me a pleasure as well as an enriching experience.

Financial support from the National Research Council of Canada is gratefully acknowledged.

Finally, Mrs. Maura Crilly is to be praised for the patience, dexterity and art with which she gave my manuscript the appearance of a thesis.

TABLE OF CONTENTS

ABSTRACT		ii
RESUME		iii
ACKNOWLEDGEMENTS		iv
TABLE OF CONTENTS		v
Chapter One :	Introduction	1
1.1	Terminology	3
1.2	Previous Work	6
Chapter Two :	On the expected number of feasible solutions to the Euclidean TSP	12
2.1	An algorithm for estimating E(CFHC)	13
2.2	An exhaustive algorithm for obtaining the optimal tour	21
2.3	Number of crossing and crossing-free Hamilton cycles	25
2.4	Conclusion	27
Chapter Three :	Approximate triangulations and the Euclidean TSP	28
3.1	An algorithm for the approximate triangulation	29
3.2	Algorithms for the ETSP	34
3.3	Conclusion	48
Chapter Four :	The shortest Hamilton chain of a graph and the TSP	49
4.1	An approximation algorithm for the TSP	49
4.2	The vertex penalty method	51
4.3	A Monte Carlo experiment	57
4.4	An empirical estimate of the expected run-time of TSP3	58
4.5	A Modification of TSP3	58
4.6	Conclusion	59

Chapter Five : Experiments with efficient algorithms for the TSP 60
5.1 Experiments with a sequential tour-building approximation algorithm 60
5.2 A new heuristic algorithm for the symmetric TSP 69
5.3 A new heuristic algorithm for the directed TSP 84
5.4 Conclusion 94

Chapter Six : On local neighborhood search and the TSP 95
6.1 An LNS algorithm for the TSP using biased starts 96
6.2 Testing TSP6 on 'classical' problems 97
6.3 Discussion 99
6.4 Conclusion 104

Chapter Seven : On maximal triangulations, optimal drawings and Hamilton cycles 105
7.1 Maximal triangulations 105
7.2 Maximal triangulations and optimal drawings 110
7.3 Maximal triangulations and Hamilton cycles 111
7.4 An improved lower bound for $\overline{\phi}(n)$ 116
7.5 Conclusion 125

Chapter Eight : On convex hulls 126
8.1 An algorithm for the convex hull in 2-dimensions 126
8.2 An algorithm for the convex hull in d-dimensions 146
8.3 Some convex hull expectations 152
8.4 Conclusion 157

Conclusions 158

References 160

"It appears likely that none of these equivalent problems can be solved in polynomial time. The evidence for this belief stems from our inability to find polynomial-time algorithms for these problems despite the intensive efforts of many workers, and from the theoretical result that a polynomial time algorithm for one of these problems would imply that ... polynomial-time algorithms exist for an unexpectedly wide class of combinatorial search problems."

R. Karp, Algorithms and Complexity, J.F. Traub, Ed., Academic Press, New York, 1976.

"But just as it would have been unfair to argue in the seventeenth century that to place a manmade object into Earth orbit is impossible on the grounds that no one at that time had the slightest idea about how to accomplish it, so it would be wrong today to make impossibility arguments about what computers can do entirely on the grounds of our present ignorance."

J. Weizenbaum, Computer Power and Human Reason, Freeman, San Francisco, 1976.

Chapter One
Introduction

The Traveling Salesman Problem (TSP) has been studied extensively by a wide variety of researchers. For over forty years engineers, combinatorists, graph theoretists, operations researchers, management scientists and many others have been interested in finding efficient techniques for solving it. This is due to the importance of the practical applications in which it arises, as well as its theoretical appeal. Today, more than two hundred (English) publications on the TSP may be found in the literature.

An allegorical way of stating the TSP, which gave it its name, goes as follows:

> "A number of cities are given along with the cost of traveling between each pair. Starting at one city a traveling salesman wishes to visit each of the remaining cities and return to his point of departure. What itinerary should he follow in order to minimize the cost of his trip?"

From a mathematical point of view, the problem is then an easy one: simple to state and trivial to solve. A quite satisfactory solution (for pure mathematicians) is:

> "Enumerate all possible tours and choose the cheapest one."

In practice, the difficulty with this approach (for applied mathematicians that is) is its unfeasability for solving problems that involve a large number of cities. To see this, note that for an n-city problem, if one starts at any city and wishes to visit each of the remaining cities just once, then the number of ways of doing this is $(n-1)!$ With n as small as 20, the number of different round trips is 121,645,100,408,832,000; a very fast digital computer capable of examining each tour in one microsecond would require more than 38 centuries to exhaust all possibilities and hence choose the cheapest tour!

Methods - other than of an enumerative nature - were sought and, for small values of n $(n < 40)$, some algorithms have been quite successful in providing the optimal solution to the problem. For large values of n, however, space along with time puts severe limitations on the size of problems that can be attacked by these methods. For problems involving a large number of cities another approach is adopted; algorithms based on a variety of heuristic techniques can be used to obtain approximate solutions that are satisfactory for practical purposes. The importance of this approach has been put in perspective by current research in complexity theory. Recent work in this field has shown that the TSP is a member of a wide class of combinatorial problems for which no efficient algorithm that guarantees an optimal solution is likely to be discovered.

This thesis provides an analysis of various aspects of the TSP. When the cities to be visited are in the plane, estimates of the expected number of feasible solutions are obtained for small values of n. A statistical analysis is also carried out on approximation algorithms that yield very good near-optimal solutions to the TSP. Finally, a set of other problems, related to the TSP, is studied; this includes the convex hull problem, the triangulation problem and a generalization of Sylvester's problem. The main contributions of this thesis are listed below.

1) Five algorithms - TSP1 to TSP5 - for the TSP that essentially use graph-theoretic techniques are described along with an appraisal of these algorithms as carried out by extensive Monte Carlo experiments. The first two algorithms are for the planar TSP: when the number of cities is small, algorithm TSP1 empirically estimates the expected value of the number of feasible tours in a random map; algorithm TSP2 uses a new tool - triangulations of points in the plane - to obtain approximate solutions to larger problems. The next two algorithms are for the general (i.e. not necessarily planar) symmetric TSP. Algorithm TSP3 is basically a reward-punishment method that yields near-optimal solutions; a bound on the quality of the solution is provided, as well as an empirical estimate of the expected run-time of the algorithm. Algorithm TSP4 is an extremely efficient approximation algorithm that combines the powers (while trying to avoid the shortcomings) of some of the presently best known heuristics that have been used to approach the TSP. The major advantages of TSP4 are its

simplicity and speed as well as the quality of the approximate solution it provides. It is also superior to the other approximation methods in that it can be easily extended to attack the asymmetric TSP. Algorithm TSP5 is an extension of TSP4 to the asymmetric case.

2) A study of maximal triangulations in connection with optimal drawings and Hamilton cycles is carried out. We give a method of placing n points in the plane and joining them by straight-line segments that yields a triangulation with the maximum possible number of edges. Triangulations of this type are shown to be Hamiltonian and an expression for the number of Hamilton cycles they contain is derived. We prove a theorem relating these triangulations and crossing-number-optimal rectilinear drawings of complete graphs. A new lower bound is also presented for the maximum number of crossing-free Hamilton cycles in a rectilinear drawing of a complete graph.

3) An algorithm - CH - is presented for solving the convex hull problem in two dimensions. We use geometrical probability theory to analyze the algorithm. It is shown that for a uniform distribution of the n data points on the unit square the asymptotic expected run-time of CH is O(n) . This is in contrast to all existing algorithms whose expected run-time is at least O(n log n) . Experiments with the algorithm confirm its intuitive and theoretical merits. A second algorithm for the planar problem - CH2 - is also described. Ideas for extending CH2 to higher dimensions are suggested.

In the present chapter we introduce the terminology adopted throughout the thesis and review the various approaches that have been used to address the TSP.

1.1 Terminology

1.1.1 Definitions

A *graph* G is a pair (V, E), where

(1) V is a finite non-empty set $\{v_1, v_2, \ldots, v_n\}$ of *points*, and

(2) E is family of pairs of points of V called *lines*.

A line (v_i, v_j) is said to *connect* v_i and v_j ; if $v_i = v_j$ the line is called a *loop*.

A graph is said to be *simple* if

(1) it contains no loops, and

(2) no more than one line connects two points.

The *complete* graph K_n has every pair of its n points connected by a line.

A graph may be either a *directed* or an *undirected* graph. In an undirected graph each line can be represented by the *unordered* pair (v_i, v_j) and vertex v_i is said to be *adjacent* to v_j . Line (v_i, v_j) is said to be *incident* to its ends v_i and v_j . In a directed graph each line has a definite orientation assigned to it and is represented by the *ordered* pair (v_i, v_j) .

A *subgraph* $G' \subseteq G$ is defined by a subset of the lines of G where G' has the same points as G.

A graph in which a number w_{ij} is associated with every line (v_i, v_j) in the graph is called a *weighted graph* and the number w_{ij} is called the *weight* of line (v_i, v_j).

In an undirected (directed) graph G, an *elementary chain (elementary path)* from point v_i to v_j is a subset of E consisting of a sequence of undirected (directed) lines (v_i, v_k) , $(v_k, v_m), \ldots, (v_s, v_j)$, represented by $(v_i\ v_k\ v_m \ldots v_s\ v_j)$, where points $v_i, v_k, v_m, \ldots, v_s$ and v_j are distinct with the possible exceptions of v_i and v_j . If $v_i = v_j$, then the elementary chain (elementary path) is called a *cycle (circuit)*.

A graph is said to be *connected* if it contains an elementary chain (elementary path) for each pair $\{v_i, v_j\}$ of distinct points.

A *Hamilton* cycle (HC) is a cycle in which every point of the graph appears once and only once. The same definition holds for a Hamilton chain, path and circuit. A graph possessing a Hamilton cycle (circuit) is termed *Hamiltonian*. When the lines are weighted, the *shortest Hamilton cycle* (SHC) is that HC with the minimum sum of line-weights. A *shortest Hamilton circuit* is defined similarly.

The number of lines of subgraph $G' \subseteq G$ incident at point v_k is called the *degree* of point v_k in G' and is represented by $d(v_k, G')$.
In a directed subgraph $G' \subseteq G$, the number of lines from v_k to the other

points in G' is called the *outdegree* of point v_k , $d^o(v_k, G')$, and the number of lines from every point to v_k is called the *indegree* of point v_k , $d^i(v_k , G')$. An *Euler* cycle (EC) of an undirected graph (or an Euler circuit of a directed graph) is a cycle (circuit) such that every line appears on it exactly once. An undirected (directed) graph which has an Euler cycle (circuit) is termed *Eulerian.* An undirected connected graph possesses an Euler cycle if and only if all its points are of even degree. A directed graph G whose underlying unidirected graph is connected possesses an Euler circuit if and only if for all of its points v_k, $d^o(v_k, G) = d^i(v_k, G)$.

A *tree* is a connected graph which contains no cycles. Given an undirected connected graph G , a subgraph $G' \subseteq G$ which is a tree (connecting together all nodes) is called a *spanning tree* (ST). A *directed tree* is either rooted to a point or from a point. A *tree rooted from point v_i* is a tree in which the indegree of v_i is zero and the indegree of each of the other points is at most one. A *tree rooted to point v_i* is a tree in which the outdegree of v_i is zero and the outdegree of the other points is at most one. A *directed spanning tree* (DST) is as its name suggests. When the lines are weighted, a *minimal spanning tree* (MST) is that ST with the minimum sum of line-weights. The *minimal directed spanning tree* (MDST) is defined similarly.

A *matching* (M) of G(V, E) is a subset of E such that no two lines meet at the same point. A *perfect matching* (PM) of G(V, E) is a matching which covers all points of V . When the lines are weighted an *optimal perfect matching* (OPM) is that PM with the minimum sum of line-weights.

1.1.2 Synonyms and Abbreviations

The abbreviations introduced in section 1.1.1 are used throughout the thesis. Note that HC and EC also stand for Hamilton circuit and Euler circuit respectively when there is no place for ambiguity. We also point out that for our purposes, line, edge, link, branch, arc and segment are all equivalent. This also applies to point, vertex and node.

When we talk about the *tour* of a traveling salesman, we mean an HC (or circuit) in a graph each node of which represents a *city*. The words optimal, shortest and cheapest are used interchangeably. Some abbreviations - like SHC, MST or OPM for example - not only denote the corresponding graphs but also their costs when the lines are weighted. By 'the cost of traveling from a to b' we mean 'the weight of line (a, b)' as well as 'the distance between a and b'.

1.1.3 Computer Programs

All computer programs mentioned in this thesis are written in FORTRAN for the IBM 370/158 and all running times reported are for this machine (unless otherwise specified). McGill's time-sharing system MUSIC was used to run all programs. The maximum run-time allowable for a job under that system is 60 CPU seconds.

In analyzing algorithms and programs, we use the notation O(f(n)) to denote a quantity not greater than c*f(n) where c is a positive constant. All logarithms mentioned in this thesis are base 2.

1.2 Previous Work

Algorithms for the TSP are usually classified into two categories: exact algorithms, i.e. those guaranteeing an optimal solution and approximation algorithms that yield a near-optimal solution in a relatively short amount of time.

Among the various approaches used to devise exact algorithms we mention dynamic programming [Beal, Gon, Hel], the branch-and-bound technique [Ag, As, Ea, Gar, Lal, Lit, Pan, Si, Stc], linear programming [Danl, Dan2, Mil, Mo, Mu, Rao], combinatorial programming [Ri, Ross], a generating function method [Ko], and the algorithm of [Gim] for a special form of the TSP.

Approximation algorithms have been obtained by applying the principles of local-neighborhood search [Ad, Ba, Boc, Chr3, Cr, Dac, Linl, Lin2, Lin3, Lin4, Rei, Stel, Ste2, Wei], the minimal spanning tree [Chr2, Chr6, Dil, Di2, Deo, Gibl, Han, He2, He3, Kar3, Kru, 0, Pri, Rosl, Ros2], nearest-neighbors [Beol, Gav], tour-building [Jo, Ka, Net, Ray, Rosl, Ros2, Web], enumeration [Robl, Rob2, Se, Rot], statistics [Go], and man-machine interaction [Kro, Mic].

The shortest-path problem, in connection with the TSP, has received much attention [Bof, Hav, Pi, Sak] and several bounds on the optimal tour have been derived [Be, Chr4, Ei, Fe, Ham, Hel, Ma, Sm, V].

Surveys of the different methods of solution and applications of the TSP are given in [Ac1, Ac2, Bea2, Beo1, Beo2, Bre, Chr1, Chr5, Cof, Con, Ei, Fl, Hen, I, La2, Le1, Le2, Liu, Roy, Sas, Sav3, Wa, Wel]. Finally, the complexity of the TSP and its relation to other combinatorial problems is studied in [Gae1, Gae2, Kar1, Kar2, Kar3, Lew, Papa, Reg, Sah, Sav2, Sav3, Wed].

In the following, we briefly describe some algorithms examplifying the various techniques used to attack the TSP. Our description is based on the excellent survey works in [Beo1] and [Ei]. We denote by $N = \{1, 2, \ldots , n\}$ the set of cities to be visited by the salesman and by c_{xy} the cost of going from x to y.

1.2.1 Exact Algorithms

1.2.1.1 Integer Linear Programming [Dan1]

Let S and $\overline{S}$ be a partition of N, such that

$$S \cap \overline{S} = \emptyset \quad \text{and} \quad S \cup \overline{S} = N .$$

The optimal tour can be obtained by minimizing

$$\sum_{j=2}^{n} \sum_{i=1}^{j-1} c_{ij} \, x_{ij}$$

subject to $x_{ij} = 0,1 \qquad (j = 2,\ldots,n \; ; \; i = 1,\ldots, j-1)$

and the loop constraints

$$\sum_{i\in S} \sum_{j\in \overline{S}} x_{ij} \geq 1$$

for all non-empty partitions $(S, \overline{S})$ where

$$x_{ij} = 1 \text{ if the salesman proceeds from city } i \text{ to city } j$$
$$= 0 \text{ otherwise.}$$

The exponential number of constraints and the fact that integer linear programming algorithms are not always guaranteed to converge rapidly, make this approach the least attractive of the exact methods. Only one large problem (n = 42) is known for having been solved using this method [Beol].

1.2.1.2 Dynamic Programming [Beal]

Let $N' = N/\{1\}$

$= \{2,3,\ldots,n\}$.

We denote by $C(S, x)$, where $S \subset N'$ and $x \epsilon S$, the cost of the shortest chain with endpoints 1 and x that includes all the elements of S . This quantity is determined from

$$C(S, x) = \min_{y \epsilon S/\{x\}} [C(S/\{x\}, y) + c_{yx}] .$$

The above equation can be applied recursively for all subsets S of N' , and for all cities x , beginning with

$$C(\{x\}, x) = c_{1x} \qquad \text{for all } x$$

and terminating with the cost of the optimal tour given by

$$\min_{x \epsilon N'} [C(N', x) + c_{x1}] .$$

This approach is primarily handicapped by the exponential amount of storage required when solving the recursive equations on a computer. Computational results are only available for $n \leq 13$ [Beol, Ei].

1.2.1.3 The Branch-and-Bound Method [Lit]

Branch-and-Bound is an efficient technique for solving constrained optimization problems [Lal]. As mentioned earlier, several algorithms based on a branch-and-bound approach have been proposed for solving the TSP. One of these algorithms [Lit] is now described.

Let T be the set of all feasible tours to an n-city traveling salesman problem with a cost matrix $[c_{ij}]$, in which all diagonal elements are set to infinity. It is possible to reduce the matrix $[c_{ij}]$ as follows: subtract from every entry of each row the smallest element of that row and then subtract from every entry of each column (of the resulting matrix) the smallest element of that column. Let the final matrix be $[c'_{ij}]$. The optimal tour under $[c'_{ij}]$ is the same as the optimal tour under $[c_{ij}]$. Every row and column of $[c'_{ij}]$ now contains at least one zero element. If we could find a tour among the zeros, it would be optimal and in terms of the original matrix the cost would be equal to the amount of the reduction. Let r_o be the reduction. Every tour in T will cost at least r_o. We say that r_o is a lower bound on the tours contained in T. The set T is now partitioned into two mutually exclusive subsets and lower bounds for each are computed. The subset with the smaller lower bound is then partitioned and two more lower bounds are computed. At each stage the subset with the smallest lower bound obtained so far is selected and partitioned again. A subset that contains a single tour whose cost equals the lower bound is eventually obtained: this is the optimal tour.

We now describe the partitioning process. For each zero element c'_{ij} of the reduced matrix, a minimum penalty that would have to be incurred if link (i,j) is not selected is computed from,

$$\rho_{ij} = \min_{k \neq j} c_{ik} + \min_{\ell \neq i} c_{\ell j} \ .$$

Next, that link whose penalty is the largest is chosen. Let this link be (s, q). The total number of tours is hence divided into two subsets: those that include link (s, q) and those that do not. The lower bound of all tours not including (s, q) is thus $r_o + \rho_{sq}$. To compute the lower bound of all tours including (s, q), row s and column q are crossed out and c'_{qs} is set to infinity. If the deflated matrix can be reduced the reduction is denoted by r_1, else $r_1 = 0$. Now, the lower bound sought is $r_o + r_1$.

This method is believed to be the most powerful of the exact algorithms although the time and storage requirements increase exponentially with the size of the problems. Computational experience is reported for $n \leq 40$ [Ei].

1.2.2 Approximation Algorithms

1.2.2.1 Sub-optimal Tour Building

Dynamic programming and the branch-and-bound method just described are known as *exact tour-building algorithms*. There exist also *approximation tour-building algorithms*. Basically, one starts with an arbitrary city, say i, builds a sequence (i, j, k,...) by successively including other cities into the sequence and stops when a tour is obtained. The simplest algorithm chooses the nearest unvisited neighbor city as the next city to be included in the sequence. When no more cities are left the last visited city is connected to the starting city thus yielding a tour [Beol, Ros1, Ros2]. Since the tour reached at termination depends on the starting city, it is possible to produce many final tours by using different starting cities and then select the best of these final tours [Gav]. (Note that this is a property which the majority of approximation algorithms share). It is obvious that the run time of this algorithm is $O(mn^2)$ where m is the number of different final tours obtained.

Another method [Ka, Ray, Ros1, Ros2] inserts cities sequentially into a partial tour such that the cost of the resulting partial tour is the least possible. Essentially, one tries to minimize $c_{ik} + c_{kj} - c_{ij}$, when city k is to be inserted between cities i and j in the partial tour. The process continues until no more cities are to be inserted. Methods vary according to the criteria used for choosing the next city to be added. The run time of the algorithms of [Ka, Ros1, Ros2] is of $O(n^2)$ while that of [Ray] is of $O(n^3)$.

1.2.2.2 <u>Minimum Spanning Tree-based Methods</u>

The basic idea behind all these algorithms is the following: first, the MST is obtained, then a sequence of transformations is applied to it finally yielding an approximate solution to the TSP.

One method reduces the MST into a chain, by means of additions and deletions of lines [Chr2, O]. In [Han, He2, He3] the relationship between the TSP and the MST is explored to derive a very sharp lower bound on the cost of an optimum tour. This method may be used in a branch-and-bound algorithm for the exact solution of a TSP.

A sequential tour-building algorithm described in [Ros1, Ros2] chooses nodes for inclusion in the partial tour in the same order in which they would have entered the MST. The algorithm has a run-time of $O(n^2)$ and the cost of the near optimal solution it obtains is guaranteed to be smaller than twice the cost of the optimal solution.

In [Chr6] a solution is obtained by adding to the branches of the MST those of an OPM of the odd-degree nodes: the resulting graph is used to yield a Hamilton cycle whose cost is no more than $\frac{3}{2}$ times the cost of the optimal tour. The algorithm runs in $O(n^3)$ time.

1.2.2.3 Local Neighborhood Search

Local neighborhood search algorithms

(1) start with an initial tour chosen either randomly or arbitrarily, and

(2) attempt to improve upon it by searching its neighborhood following some specific policy that yields a better tour.

As soon as an improvement is found it is immediately adopted and the above procedure is now applied to the new tour thus obtained. If no further improvement is possible the tour on hand is called a local optimum. The entire procedure can now be repeated as many times as required each time using a different initial tour. The cost of the various local optima is then chosen as the final approximate solution to the TSP.

This idea was first exploited in [Ba]: starting with an arbitrary tour a series of ever shorter tours is obtained where the (m-1)st tour has each of the n groups of m consecutive edges constituting it form a path of minimum length. This tour is said to be "mini per set of n". When m=n-1, the "mini per set of n" tour is obtained and represents an approximate solution to the TSP.

A similar method is described in [Cr]: starting with an arbitrary tour, apply simple transformations (called *inversions*) iteratively to obtain a crossing-free tour. An inversion is the replacement of two edges in a tour by two other edges to form a new tour.

Further developments of the original ideas of [Ba] and [Cr] were provided in [Chr3, Lin1, Lin2, Lin3, Lin4, Rei, Ste1, Ste2] yielding an optimization criterion considered today as one of the best approximation algorithms for the TSP. The technique, known as "k-optimality", was first formally described in [Lin1]. A tour is said to be "k-optimal" if it is "(k-1) - optimal" and if, in addition, no better tour can be obtained by replacing any k edges of the tour with k other edges. Experiments [Ei, Lin3] have demonstrated that 3-optimal tours have a high probability of being optimal. In general, the cost of a 3-optimal tour exceeds that of the optimal tour by a very small amount. Problems of up to 110 cities have been attempted using this method [Lin3].

Chapter Two
On the Expected Number of Feasible Solutions to the Euclidean TSP

In the statement of a classical TSP the only data provided are the costs of traveling between cities. For an n-city problem these are usually represented by an n x n matrix C - the cost matrix. Thus,

$$\begin{aligned} c_{ij} &= \text{cost of traveling from } i \text{ to } j \text{ , for } i \neq j \\ &= \infty \text{ , for } i = j \end{aligned}$$

When the entries of C are Euclidean distances in two dimensions, the TSP is termed Euclidean (ETSP) and has special exploitable properties. Some of these are obvious and they are stated below without proof.

For an ETSP,

(i) the cost matrix is symmetric,

i.e. $c_{ij} = c_{ji}$ for all i and j ;

and the triangular inequality holds,

i.e. $c_{ij} + c_{jk} > c_{ik}$ for all $i \neq j \neq k$;

(ii) every city is visited once and only once in the optimal tour; and

(iii) the optimal tour does not intersect itself [Beol], under the assumption that no three cities are on a straight line.

From (i) and (ii) it follows that the number of different tours is $\frac{(n-1)!}{2}$ for an n-city problem each being a Hamilton cycle. From (iii) it follows that only crossing-free Hamilton cycles (CFHC) need be considered [Beol]. It should be clear that property (ii) is an immediate result of the triangular inequality and is not necessarily true in a general TSP. This property has so much influenced the literature that very often it is stated as a condition of the problem, even for TSP's where the triangular inequality does not hold [Lin3].

In this chapter we present an algorithm for empirically calculating E(CHFC) -

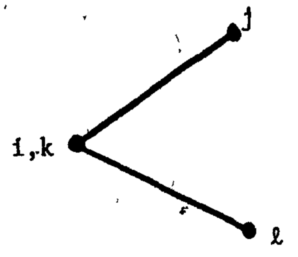
the expected number of feasible solutions in a random TSP. For small values of n, a refinement of the algorithm will serve in finding the solution of an ETSP. This approach will be useful in the appraisal of various techniques examined in later chapters. Finally, a related problem is investigated.

2.1 An algorithm for estimating E(CFHC)

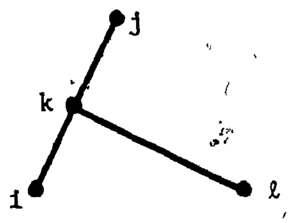
In an ETSP, cities lie in the cartesian plane and the cost of traveling from one to the other is the length of the straight line segment joining them. Usually, we are given a map of the problem, each city being defined by a pair of cartesian coordinates (x, y). This map is a rectilinear drawing of K_n , the complete undirected graph with n nodes, defined by the cities (its nodes) and the segments joining them (its edges). Our program for estimating E(CFHC) is essentially composed of two subroutines. In section 2.1.1 we describe procedure CROX for determining all crossings in a drawing of K_n . Procedure PERMU for generating CFHC's is presented in section 2.1.2. In section 2.1.3 the two procedures are used to estimate the expected number of CFHC's in a random drawing of K_n .

2.1.1 Determining all the crossings in a drawing of K_n

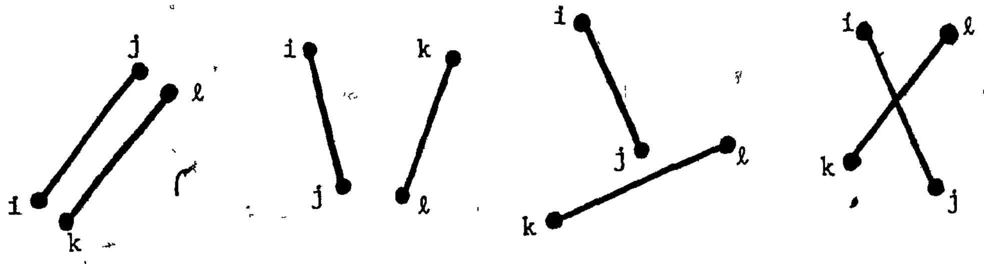
A rectilinear drawing of K_n in the plane is given. For every four different vertices of K_n , say i, j, k and ℓ , it is required to determine whether the straight-line segments [i, j] and [k, ℓ] intersect. We assume that the vertices are given by their cartesian coordinates, that no two of them coincide (Fig. 2.1.a), and that no three of them are collinear (Fig. 2.1.b). The four possible situations are shown in Figure 2.1.c .



(a)



(b)



(c)

Figure 2.1

Two different implementations of procedure CROX are now described.

Method 1

"For every pair of independent segments, find the point of intersection, if any, of the two straight lines determined by the segments and check whether it lies on both segments."

This is done as follows.

First compute the parameters of the two straight lines,

$$a_1 = y_i - y_j \qquad\qquad a_2 = y_k - y_\ell$$

$$b_1 = x_j - x_i \qquad \text{and} \qquad b_2 = x_\ell - x_k$$

$$c_1 = x_i y_j - x_j y_i \qquad\qquad c_2 = x_k y_\ell - x_\ell y_k \;.$$

Then find $d_1 = c_2 b_1 - c_1 b_2$ and $d_2 = a_1 b_2 - a_2 b_1$. If $d_2 = 0$ the segments do not intersect (they are parallel).

Else $x = d_1/d_2$, is the x-coordinate of the point of intersection.

Now, if x lies "between" x_1 and x_2 on the one hand and "between" x_3 and x_4 on the other hand the two segments intersect; this can be determined by at most six comparisons as shown in the partial flow-chart in Figure 2.2

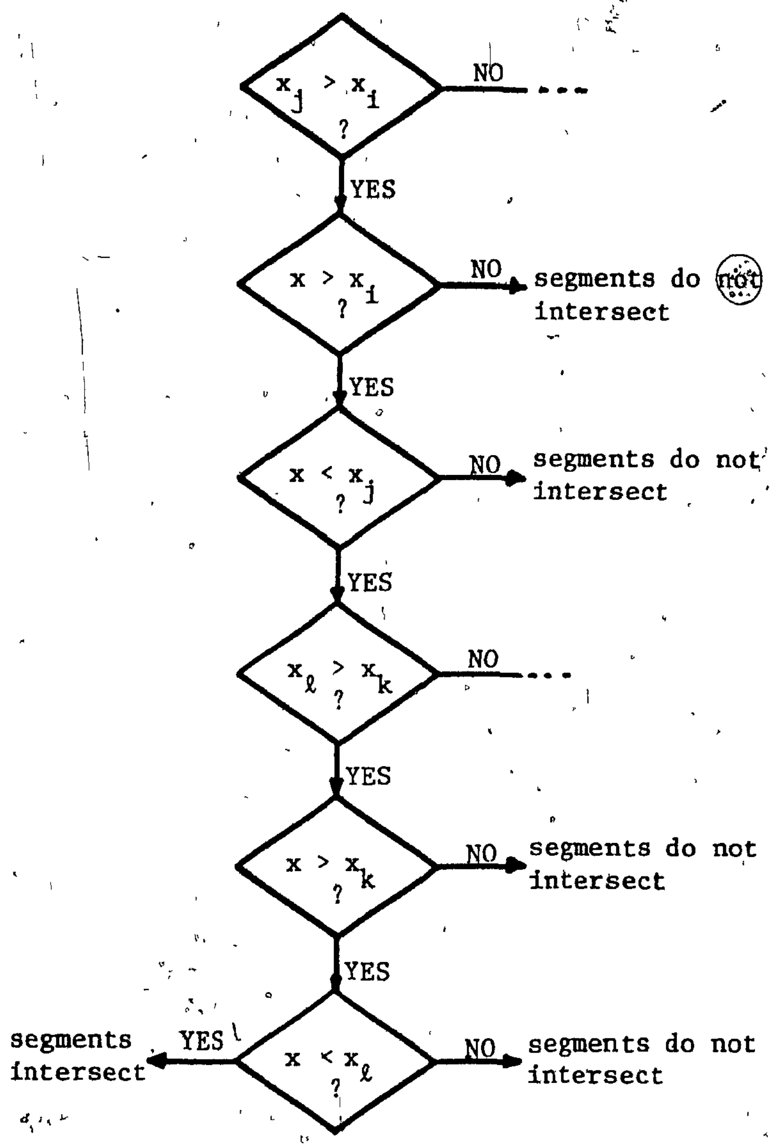


Figure 2.2

The total number of elementary operations required by this method is then 24 as shown in Table 2.1 .

Operation	Number
Subtraction	8
Comparison	7
Multiplication	8
Division	1

Table 2.1

Method 2[Av1]

"For every pair of independent segments check whether the two extreme points of each segment lie on different sides of the straight line determined by the other segment".

This can be done as follows.

First compute

$$a_1 = (y_i - y_k)\ (x_\ell - x_k)$$

$$b_1 = (y_\ell - y_k)\ (x_i - x_k) \text{ and compare } (a_1 : b_1) \ ;$$

$$a_2 = (y_j - y_k)\ (x_\ell - x_k)$$

$$b_2 = (y_\ell - y_k)\ (x_j - x_k) \text{ and compare } (a_2 : b_2) \ ;$$

If the outcomes of the first and second comparisons are different, compute

$$a_3 = (y_k - y_i)\ (x_j - x_i)$$

$$b_3 = (y_j - y_i)\ (x_k - x_i) \text{ and compare } (a_3 : b_3) \ ;$$

$$a_4 = (y_\ell - y_i)\ (x_j - x_i)$$

$$b_4 = (y_j - y_i)\ (x_\ell - x_i) \text{ and compare } (a_4 : b_4) \ .$$

If the outcomes of the third and fourth comparisons are different, the two segments intersect.

The number of comparisons required is at most 4 as shown in the partial flow-chart in Figure 2.3 .

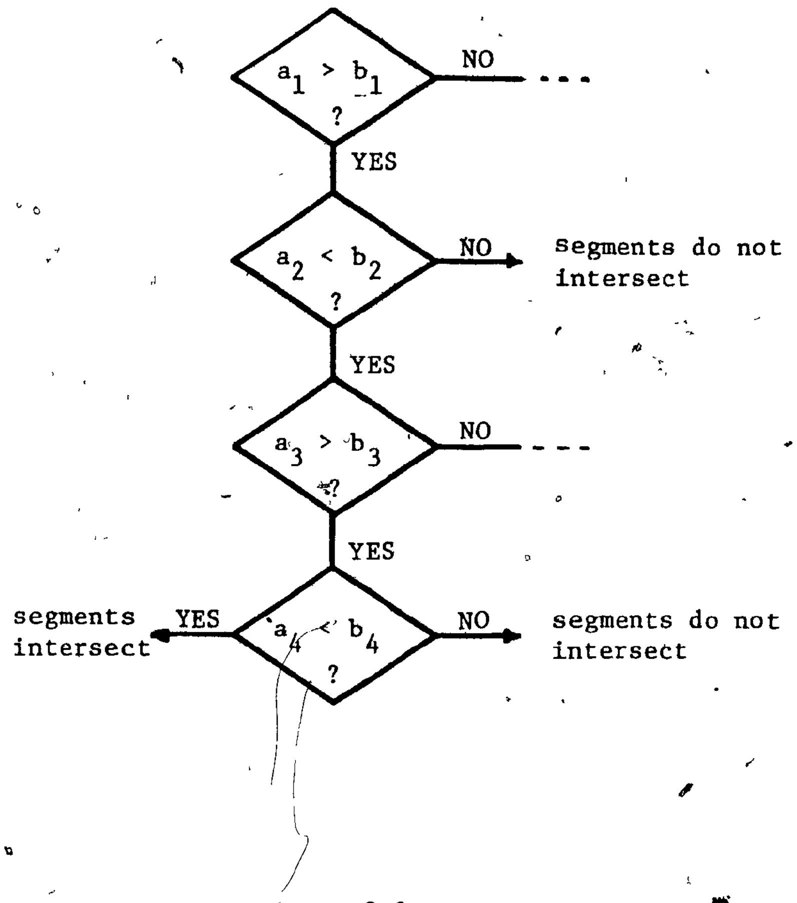


Figure 2.3

The total number of elementary operations required by this method is then 24 as shown in Table 2.2 .

Operation	Number
Subtraction	12
Comparison	4
Multiplication	8

Table 2.2

Although the two implementations require the same number of elementary operations for every pair of segments, the second method is recommended since it involves no divisions.

The output of procedure CROX is stored in a 4-dimensional integer array ICROX.
If segments [i, j] and [k, ℓ] inter~~sect~~ then eight entries of ICROX will be set to 1, namely,

$$\begin{array}{l} \text{ICROX}(i, j, k, \ell) \\ \text{ICROX}(k, \ell, i, j) \\ \text{ICROX}(j, i, k, \ell) \\ \text{ICROX}(k, \ell, j, i) \\ \text{ICROX}(i, j, \ell, k) \\ \text{ICROX}(\ell, k, i, j) \\ \text{ICROX}(j, i, \ell, k) \\ \text{ICROX}(\ell, k, j, i) \ . \end{array}$$

Else the eight entries are set to 0 .

2.1.2 Determining all CFHC in a drawing of K_n

Procedure PERMU upon being given the ICROX array will generate all CFHC's as follows.

"A tour is a permutation of the cities. Starting with a city generate a crossing-free permutation by sequentially adding those edges that introduce new cities and do not intersect previous edges. If the tour is complete generate the next permutation. When all branching possibilities have been exhausted. Stop".

Two 1-dimensional arrays are used in the implementation of the above idea:

PERM : contains the permutation being formed.

NEXT : contains available cities.

The basic algorithm, along with three special procedures, are outlined below. All arrays and simple variables are common to all procedures.

<u>Procedure PERMU</u>

```
Step 1 (Initialization)
    For J = 1 to N  set  NEXT(J) = J + 1 ;
    Set NEXT(N+1) = 1  and  I = 0 .
Step 2 (Add an edge)
    Call procedure ADDELE .
Step 3 (Test length)
    If I ≤ 3 , go to Step 2 ;
    Else continue.
Step 4 (Using array ICROX check for intersections)
    If most recently added edge (PERM(I-1), PERM(I)) intersects
```

```
        with any previous edge, go to Step 8 ;
        Else continue.
    Step 5 (Is permutation complete?)
        If I < N, go to Step 2 ;
        Else continue.
    Step 6 (Check for repetitions)
        If PERM(2) > PERM(N) , go to Step 8 ;
        Else continue.
    Step 7 : (Add last edge, then check for intersections)
        If last edge (PERM(N) , PERM(1)) intersects with any
        previously introduced edge, go to Step 9 ;
        Else output cycle and go to Step 9 .
    Step 8 : (Done?)
        If PERM(2) = N , exit ;
        Else continue.
    Step 9 : (Increment permutation)
        Call procedure INCPER ;
        Go to Step 3 .
Procedure ADDELE
    Step 1 : If NEXT(N+1) = N+1 call procedure INCPER and exit ;
             Else continue.
    Step 2 : Set I = I + 1 , PERM(I) = NEXT(N+1) , NEXT (N+1) =
                NEXT (PERM(I)) and J = 1 .
    Step 3 : If NEXT(J) = NEXT(N+1) , exit ;
             Else set NEXT(J) = NEXT(N+1) , J = J+1
             and repeat Step 3 .
Procedure INCPER
    Step 1 : If NEXT(PERM(I)) = N+1 , go to Step 4 ;
             Else continue.
    Step 2 : Set Q = PERM(I) ;
             Call procedure RESTORE ;
             Set PERM(I) = NEXT(Q) .
    Step 3 : If NEXT(Q) = NEXT(PERM(I)) , exit ;
             Else set NEXT(Q) = NEXT(PERM(I)) , Q = Q+1 and repeat
             Step 3 .
    Step 4 : Call procedure RESTORE ;
             Set I = I-1 and go to Step 1 .

Procedure RESTORE
    Step 1 : Set J = PERM(I) - 1 ;
    Step 2 : If J=0 , go to Step 3 ;
             Else if NEXT(J) < PERM(I) , exit ;
                Else set NEXT(J) = PERM(I) ,
                J=J-1 and repeat Step 2 .
    Step 3 : If NEXT(N+1) < PERM(I) , exit ;
             Else set NEXT(N+1) = PERM(I) and exit .
```

2.1.3 Determining E(CFHC)

At this point it is not difficult to see how procedures CROX and PERMU can be used to obtain the number of CFHC's in a given configuration of cities.

In order to get E(CFHC) in a random map a Monte Carlo experiment is performed. For each value of n from 4 to 10, the following three steps were repeated 1000 times to get the average number of CFHC's as well as the standard deviation, rounded to the nearest integer.

```
    Step 1   Use a uniform random number generator to generate
             coordinates of cities in the unit square.
    Step 2   Use CROX to find all crossings in the drawing generated
             in Step 1.
    Step 3   Use PERMU to determine the number of CFHC's in the
             drawing generated in Step 1 .
```

Table 2.3 shows the results obtained .

n	$n_1=(n-1)!/2$	$n_2=E(CFHC)$	Standard deviation	$\frac{n_1}{n_2} \times 100$
4	3	2	1	66.66
5	12	3	2	25.00
6	60	8	5	13.33
7	360	20	14	5.55
8	2520	54	39	2.14
9	20160	160	118	0.79
10	181440	474	365	0.26

Table 2.3

2.2 An exhaustive algorithm for obtaining the optimal tour

The two procedures described in the previous section can be used to obtain the shortest Hamilton cycle (SHC), in a drawing of K_n by exhaustive enumeration. The algorithm is described below.

Step 1 Given the coordinates of the cities use the Euclidean metric to obtain the intercity distances, i.e. form the cost matrix.

Step 2 Use CROX to find all crossings.

Step 3 Use PERMU to find all CFHC's along with their costs.

Step 4 Choose the SHC.

Although the number of candidate tours is substantially decreased by this method from $(n-1)!/2$, as shown in section 2.1.3, procedures CROX and PERMU , however, are still exponential algorithms.

In section 2.2.1 we describe a refinement which, when introduced in the algorithm above, causes a dramatic reduction in the number of cases investigated. The modified algorithm is used in section 2.2.2 to obtain a new estimate of the expected number of feasible solutions in a random map.

2.2.1 Kink-free cycles need only be considered

A rule of thumb will now be described that we shall use to reduce the number of cycles considered in our exhaustive enumeration of feasible solutions. The idea - although remotely related to the principles used in [Ba] and [Linl] - is entirely novel as far as the ETSP is considered since it is solely based on a geometric consideration and is applied *during* the sequential construction of a tour.

Let a crossing-free chain be composed of (r+2) edges where $r = 1,2,\ldots, n-2$. Further, let (a,b) and (e,d) be the first and last edges of the chain respectively, as shown in Figure 2.4

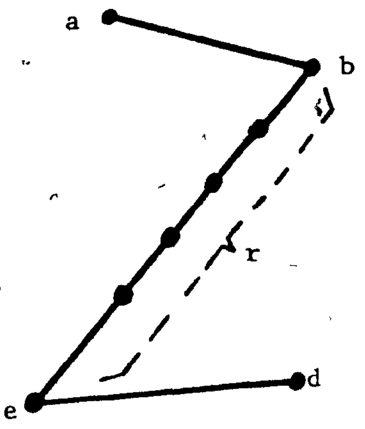


Figure 2.4

We say that this chain has an r-kink if $c_{ab} + c_{ed} > c_{ae} + c_{bd}$.
Now, obviously, a tour which has an r-kink is not to be considered since edges (a,b) and (e,d) can be replaced by (a,e) and (b,d) yielding a cheaper tour. This idea is exploited in procedure PERMU .
"In the process of building a tour, every time an edge is added to a chain of length $\ell-1$, which does not cross any previous edge, a test for an r-kink follows, for $r=1,2,\ldots,\ell-2$. If the new edge creates an r-kink it is disregarded, else it is kept and a new edge is fetched."

Listed below is procedure PERMU1 the new version of PERMU . It shows the tests for r-kinks and how the shortest cycle is determined. At exit from the procedure, a one-dimensional array SHC stores the shortest cycle.

Procedure PERMU1

```
Step 1':
        For J=1 to N set NEXT(J) = J+1 ;
        Set NEXT(N+1) = 1 and I = 0 ;
        Set SCOST = ∞
Step 2:
        Call ADDELE .
Step 3:
        If I ≤ 3 , go to Step 2 ;
        Else continue.
Step 4':
        If new edge creates an intersection, go to Step 8 ;
        Else
           if new edge creates an r-kink, go to Step 8 ;
           Else continue.
Step 5:
        If I < N , go to Step 2 ;
        Else continue.
Step 6:
        If PERM(2) > PERM(N) , go to Step 8 ;
        Else continue.
Step 7':
        If last edge creates an intersection go to Step 9 ;
        Else
           If last edge creates an r-kink go to Step 9 ;
           Else compute COST, the cost of the new cycle ;
           If COST > SCOST go to Step 9 ;
           Else set SKOST = COST and store cycle in SHC ;
           Go to Step 9 .
Step 8:
        If PERM(2) = N , exit ;
        Else continue .
Step 9:
        Call procedure INCPER ;
        Go to Step 3 .
```

The modified algorithm - referred to as TSP1 - solves a 12-city problem in less than 18 secs on the average.
In later chapters we shall make use of TSP1 for purposes of comparison and appraisal of various heuristic techniques for the TSP.

2.2.2 A new estimate of the number of feasible solutions

The modified algorithm is now used - in a manner similar to that described in section 2.1.3 - to obtain a new estimate of the number of feasible solutions for an ETSP. By a feasible solution it should be understood that we mean a crossing-and-kink-free Hamilton cycle (CKFHC). Table 2.4 shows the average number of CKFHC's as well as the standard deviation rounded to the nearest integer, in a random map.

n	E(CKFHG)	St. dev.
4	1	0
5	1	0
6	1	0
7	1	1
8	1	1
9	2	1
10	2	1
11	2	1
12	3	2

Table 2.4

Some typical distributions are shown in Table 2.5 where F denotes the number of CKFHC's and P(F) the probability of F .

n	F	P(F)
6	1	0.853
	2	0.140
	3	0.007
7	1	0.766
	2	0.197
	3	0.032
	4	0.005
8	1	0.647
	2	0.271
	3	0.063
	4	0.013
	5	0.004
	6	0.002

Table 2.5

2.3 Number of crossings and crossing-free Hamilton cycles

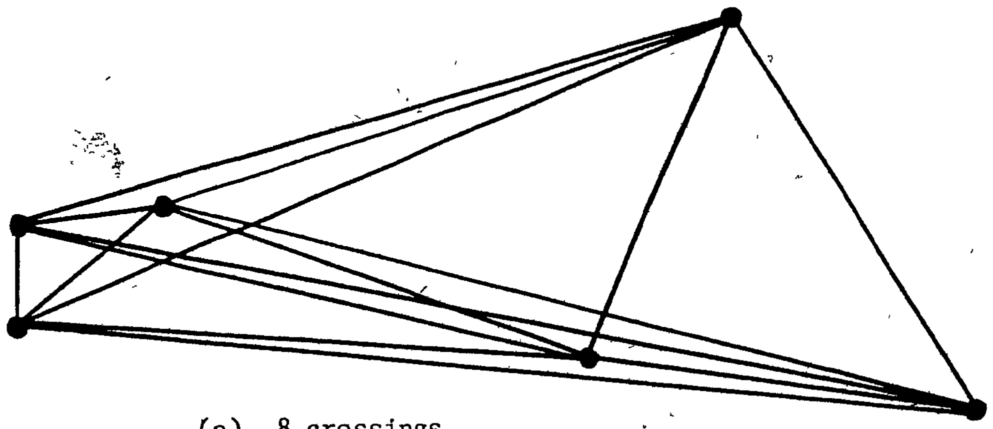
So far we have considered only random drawings of K_n. In [New] a special class of drawings of K_n is studied : the CFHC - optimal drawings. These are (rectilinear and non-rectilinear) drawings of K_n possessing the maximum possible number of CFHC's. In the rectilinear case this number is denoted by $\bar{\phi}(n)$. Table 2.6 (from [New]) shows the known values of $\bar{\phi}(n)$.

n	$\bar{\phi}(n)$
3	1
4	3
5	8
6	29
7	92*
8	339*
9	1228*

*conjectured

Table 2.6

It was observed in [New] that for small n, CFHC - optimal drawings of K_n are also crossing-number-optimal. The latter are drawings possessing the minimum possible number of crossings [Er] . This leads us to ask the following question: is there a correlation between the number of crossings and the number of CFHC's in a drawing of K_n ? In other words, is it true that the more the crossings the fewer the CFHC's and vice versa? Procedures CROX and PERMU are used to answer these questions. It turns out that for the majority of cases, the above statement is true; however this property does not hold in general. The two drawings of K_6 shown in Figure 2.5 illustrate a counter-example. (See Chapter 7 for the derivation of a lower bound on $\bar{\phi}(n)$) .


(a) 8 crossings
12 CFHC's

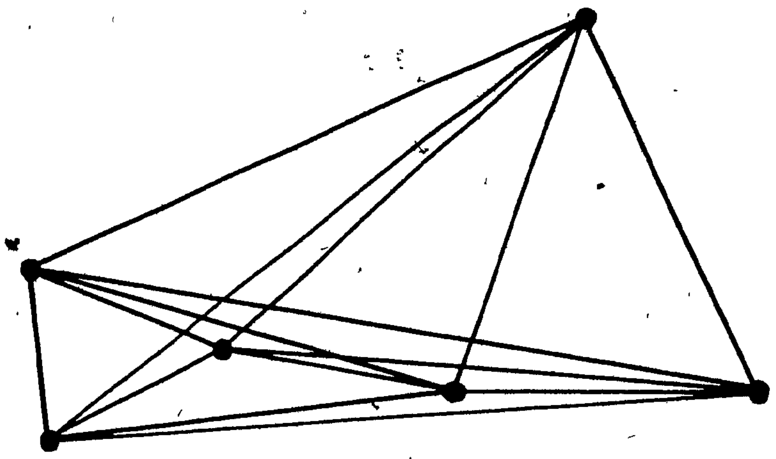

(b) 9 crossings
13 CFHC's

Figure 2.5

2.4 Conclusion

In this chapter we demonstrated how the set of candidate tours for the ETSP could significantly be reduced in size when some simple heuristics are used. In fact, it was experimentally shown how small the subset of feasible solutions actually is. We therefore ask: Will more powerful techniques be developed in the future to search 'intelligently' the astronomical set of candidate tours looking for this subset of feasible solutions? Will these techniques have a running-time that grows polynomially with the size of the input? It is hoped that the above questions will some day be answered affirmatively.

A proof that the ETSP is NP-complete [Aho] was given in [Gae2, Papa]. When a problem is shown to be a member of this class, the general tendency today is to conclude that no efficient algorithm is likely to be discovered for solving it. No one, however, has been able to prove this statement. The present writer believes that we still lack the appropriate mathematical tools to tackle these problems. Once (and if) these tools are discovered and the right representation for each problem chosen, we may be able to devise solution methods with the required degree of efficiency. Without the invention of calculus, it is hard to believe that man would have been able to walk on the moon!

Chapter Three
Approximate Triangulations and the Euclidean TSP

The concept of the minimal spanning tree (MST) has been extensively used [Chr6, Han, He2, He3, Kar3, O, Ros1, Ros2] to obtain an approximate solution to the TSP (See Chapters 4 and 5). It has been shown [Sh2] that for the ETSP an algorithm [Ros1] with running time O(n log n), where n is the number of cities, can be used to get an approximate solution which is no worse than twice the optimal.

Another concept which led to a theorem [Beo1] for the Euclidean TSP is that of the convex hull (CH) of a set of points in the plane, i.e. the smallest convex polygon containing all the points. This theorem states that the order in which the points forming the vertices of the CH appear in the shortest tour, is the same as the order in which they appear in the CH. The theorem follows directly from the obvious geometrical fact that the shortest tour does not intersect itself. Several algorithms have been described for obtaining the CH [Akl, Gr, Ja, Pre, Sh2] and they all have an expected run-time of O(n log n). A detailed treatment of the problem is provided in Chapter 8.

It is not difficult to show that the MST and the CH of a set of points in two dimensional Euclidean space are related to a (little bit more involved) structure that we define shortly: the minimum-weight triangulation (MWT). A triangulation of n points in the plane is the plane graph obtained by joining the points by non-intersecting straight line segments until no edge can be added without creating an intersection [Berm, Sku, Wh]. From this definition it follows directly that:

1) The CH of the set of points is a subset of every triangulation since - by definition - no edge of the CH intersects another edge of the complete graph.
2) Every region of the triangulation interior to the CH is a triangle.

Now, if each edge is assigned a weight (the Euclidean distance), the MWT is a triangulation such that the sum of the weights of its edges is the least possible.

Two algorithms have been published to obtain the MWT [Du, Sh2] and they have both been proven incorrect by counterexample in [L1], where it is also conjectured that the MWT problem is NP-complete.

In this chapter we study approximate (minimum-weight) triangulations (AT) in connection with the ETSP (Note that AT $\geq$ MWT). In particular we show that the AT is a source of potential candidates for the edges of an optimal solution to the ETSP.

3.1 An Algorithm for the Approximate Triangulation

In this section we present an algorithm for obtaining an approximate triangulation (AT). The algorithm will also allow the MST to be derived. Our interest in the MST in this context is due to the following reasons:

1) It was demonstrated in [L1] that the MST is not necessarily a subset of the MWT. Nevertheless, the MST is a subset of the dual of a Voronoi diagram [Sh1] as shown in [Sh2]. This latter was believed for sometime to be the MWT; although it is not so in general, it turns out to be - in most of the cases - a very good AT. It follows that the MST is a subset of some AT.
2) The MST provides a good approximation to the optimal tour in a TSP (see Chapter 5).

From (1) and (2) it is obvious that we will need the MST for comparison purposes when constructing an approximate solution to the TSP based on the AT.

3.1.1 Algorithm AT

```
Step 1.  Sort edges in ascending order of weight. Let
         $E = \{e_1, e_2, \ldots, e_\ell\}$ be the set of sorted edges
         where $e_i < e_j$ if $i < j$ and $\ell = n(n-1)/2$ ; let
         MST = $\emptyset$ ; set i = 1 , LIST = E and j = 0 .
Step 2.  If ($j = n-1$) or ($e_i$ creates a cycle in MST)
         go to Step 3;
         Else set MST = MST $\cup \{e_i\}$ and j = j+1 .
```

```
Step 3.  If $e_i \notin$ LIST go to Step 4;
         Else set AT = AT $\cup \{e_i\}$
         and LIST = LIST/$\{e_k$ : $e_k$ and $e_i$ intersect$\}$
Step 4.  If $i = \ell$ , stop ;
         Else set $i = i+1$ and go to Step 2.
```

3.1.2 Remarks

One should observe that

1). Algorithm AT (without the operations for the MST) resembles an algorithm described in [Du] which was believed to yield the MWT.

2) The MST is basically obtained in algorithm AT by the 'greedy' method of [Krü]. (The name 'greedy' is actually used for a whole class of algorithms [La2]).

3) The complete graph of n points has $\binom{n}{2}$ edges; the AT being a plane graph, the maximum number of edges it can have is 3n-6 , and this happens when its CH is a triangle. (The minimum being 2n-3 and this occurs when the n points form a convex n-gon). The running time of AT can thus be described as shown in Table 3.1 .

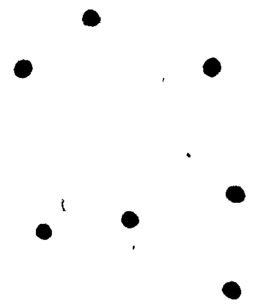
Step	Complexity	Maximum number of times executed
1	$0(n^2 \log n)$	1
2	$0(n)$	$0(n^2)$
3	$0(n^2)$	$0(n)$
4	$0(1)$	$0(n^2)$

Table 3.1

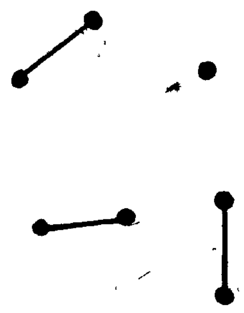
It follows that the computation is dominated by Step 3 and that the overall complexity of algorithm AT is $0(n^3)$.

3.1.3 Detection of cycles

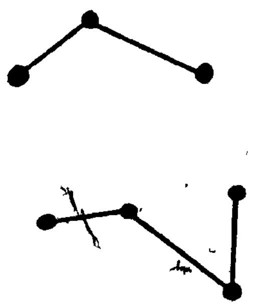
In Step 2 of algorithm AT , when an edge becomes a candidate for the MST the question of whether or not it creates a cycle with the previous tree edges is posed. Here we show how this question can be answered. We first note that in the procedure described above, the MST may be created by growing several subtrees that are ultimately merged to yield the final spanning tree. Figure 3.1 shows an example of this process.



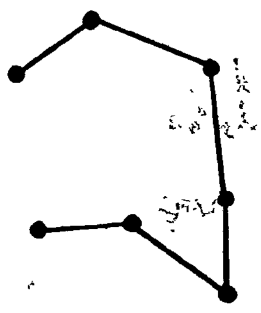
(a) Given seven points in the plane.



(b) The first three closest pairs are connected and three subtrees result.



(c) The next two shortest edges are introduced reducing the number of subtrees to two.



(d) When the last edge is included the two subtrees are merged to yield the MST.

Figure 3.1

In order to detect cycles, we attach two labels to each node i , LAB (i) and MV(i) , as follows:

(1) all nodes belonging to the same subtree have the same label LAB ; initially LAB(i) = i for all i ; and

(2) if a node has not been entered into any subtree its label MV is 0 , else it is 1 ; initially MV(i) = 0 for all i .

Now, when an edge connecting nodes i and j is considered for inclusion into the tree:

If LAB(i) = LAB(j) the edge is rejected since nodes i and j belong to the same subtree (and hence the new edge would create a cycle).

Otherwise the edge is accepted (since it obviously merges two subtrees into one).

When an edge merges two subtrees, the labels are updated as follows:

a) If MV(i) = 0 and MV(j) = 0 , this is the case of two totally new vertices (see Figure 3.2); we arbitrarily set LAB(i) = j then MV(i) = 1 and MV(j) = 1 .

Figure 3.2

b) If MV(i) = 0 and MV(j) = 1 , this is the case of a new vertex, i , that is being connected to a subtree (see Figure 3.3); we set LAB(i) = LAB(j) and MV(i) = 1 .

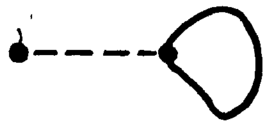


Figure 3.3

c) If MV(i) = 1 and MV(j) = 1 , this is the case of two subtrees merged by the edge connecting i to j (see Figure 3.4); we arbitrarily set all nodes k with LAB(k) = LAB(i) to LAB(k) = LAB(j) .

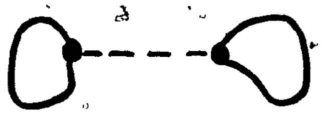
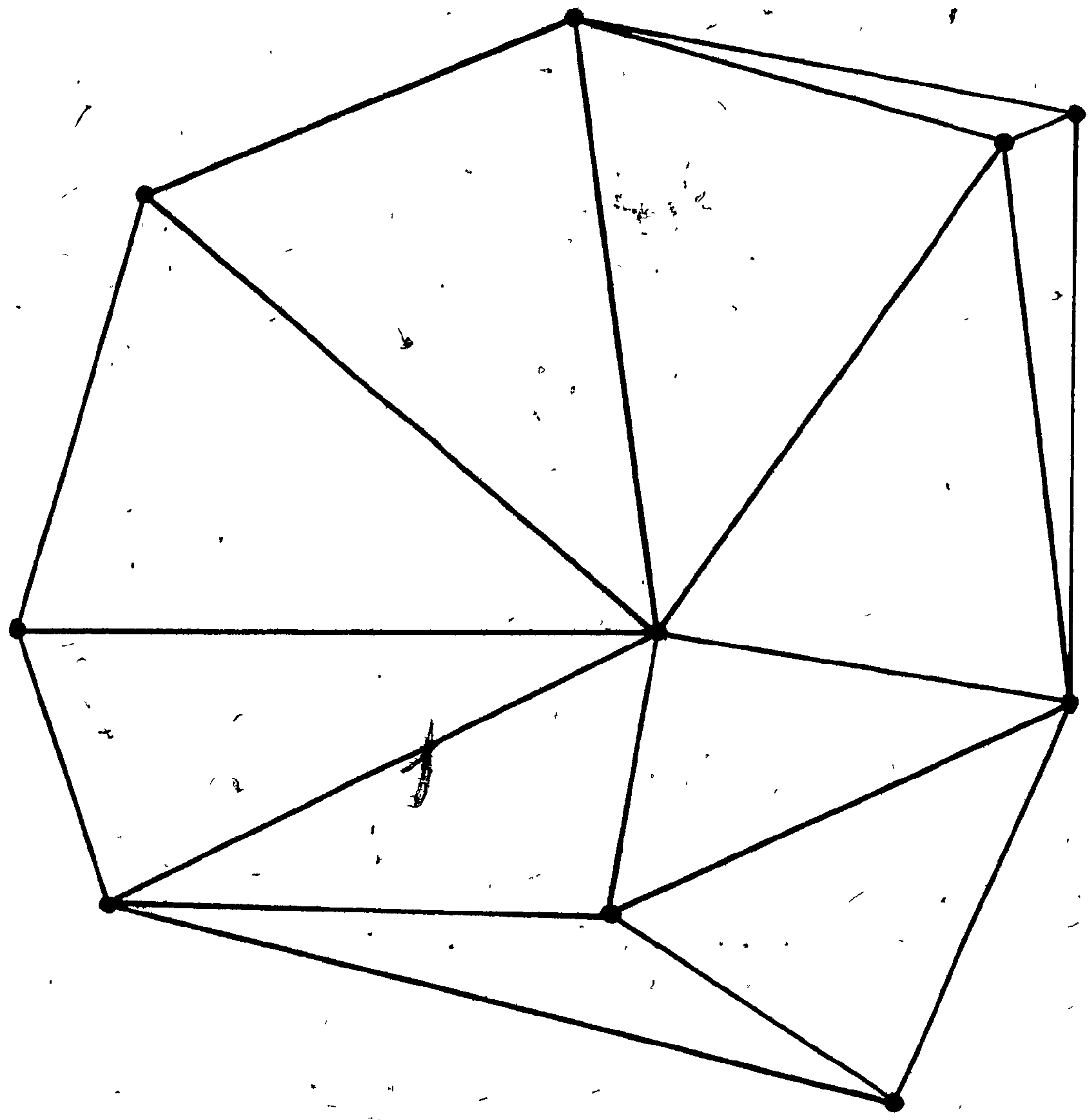


Figure 3.4

(For a complete analysis of this problem see [Aho]).

3.1.4 Example: Figure 3.5 shows a set of points in the plane and AT .



A set of points and AT.

Figure 3.5

3.2 Algorithms for the ETSP

In this section we present three algorithms for the ETSP based on the AT .

3.2.1 Preliminaries

We first relate optimal Hamilton chains to optimal Hamilton cycles in the Euclidean plane. The following propositions are straightforward.

Proposition 3.1

Given the shortest Hamilton chain through a set of points, the shortest Hamilton cycle is not necessarily obtained by adding an edge to the chain.

Proof:

By counterexample, as shown in Figure 3.6 .

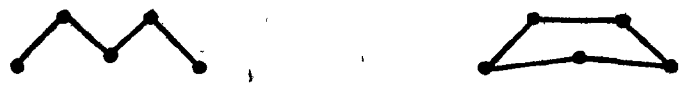


Figure 3.6

Obviously: Shortest Hamilton Chain + Closing Edge ≠ Shortest Hamilton Cycle. Q.E.D.

Proposition 3.2

Given the shortest Hamilton cycle through a set of points, two points p_i and p_j adjacent on the cycle are joined by the shortest Hamilton chain with end-points p_i and p_j .

Proof:

Given the shortest Hamilton cycle, as shown in Figure 3.7,

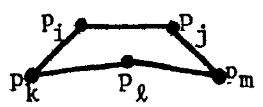


Figure 3.7

we want to prove that $p_i\ p_k\ p_\ell\ p_m\ p_j$ is the shortest Hamilton chain with end points p_i and p_j . Assume this is not the shortest Hamilton chain with end-points p_i and p_j ; this means that there exists a shorter chain, contradicting the optimality of the Hamilton cycle. Hence,

Shortest Hamilton Cycle - edge (p_i , p_j) = Shortest Hamilton Chain with end-points p_i and p_j .

Q.E.D.

Proposition 3.3

Given the shortest Hamilton chain through a set of points; if its end points are p_i and p_j and edge (p_i, p_j) happens to be in the shortest Hamilton cycle, then

Shortest Hamilton Chain + edge (p_i, p_j) = Shortest Hamilton Cycle.

Proof:

Assume the resulting Hamilton cycle is not the shortest.

Since edge (p_i, p_j) is in the shortest Hamilton cycle, its removal from that cycle would yield a Hamilton chain shorter than the initial one, hence a contradiction.

Q.E.D.

We now proceed to define and discuss a few concepts relating to a set of points in the plane and on which the technique is based.

3.2.2 Nearest-Neighbors Algorithms

The "nearest-neighbors" principle was used with relative success to obtain an approximate solution to the TSP. One of the simplest algorithms [Beol] starts with a city, connects it to its nearest neighbor, then connects this last one to its nearest neighbor not yet included and continues in this fashion thus creating a Hamilton chain. The first and last cities are then connected yielding a tour. (This approach is also known as a 'greedy' heuristic [Goo]). An improvement on the method is described in [Gav] and bounds on its general performance are given in [Rosl, Ros2].

Another technique known as the "engineering approach" [Robl, Rob2] enumerates all possible Hamilton cycles in which each node branches only to one of its m nearest neighbors, where m is arbitrary.

In the so-called "linear 2-opting" method [Ste1] an attempt is made to improve on the efficiency of the 2-opting procedure [Lin1] by only considering the closest neighbors of every city.

The approach adopted in this chapter belongs to this family of algorithms. We call two points "neighbors" if they are connected by a line segment in the AT .

3.2.3 An Approximation Algorithm for the ETSP

Proposition 3.2 above suggests that the shortest Hamilton cycle can be obtained by finding the shortest Hamilton chain - between every two points in the given set. However, this is far from being a reasonable approach to the problem, the number of Hamilton chains being equal to n times the number of Hamilton cycles, let alone the fact that getting the shortest chain is by no means easier than getting the shortest cycle. Nevertheless, we shall describe a heuristic technique for obtaining an approximation to the shortest Hamilton cycle that uses a variant of the above idea. Solutions obtained by this method were always optimal and led to an interesting observation regarding the ETSP. We note that, although the procedure has a complexity which grows exponentially with n , it is presented for its theoretical interest. Algorithms based on the same principles, but with lower time requirements, will be presented in section 3.2.4.

3.2.3.1 Algorithm TSP2

<u>Step 1</u> Find an AT .

<u>Step 2</u> Identify points on the CH .

<u>Step 3</u> For every point p_i on the CH do the following:

a) Enumerate all Hamilton chains starting at p_i and composed only of edges of the AT

b) Add to every chain found in (a) an edge connecting the first and last points, thus creating a Hamilton cycle.

<u>Step 4</u> Among all cycles found in Step 3 choose the shortest.

Figure 3.8 shows a set of 12 points in the plane, the corresponding AT and the cycle obtained by TSP2 . We note that this cycle is the SHC .

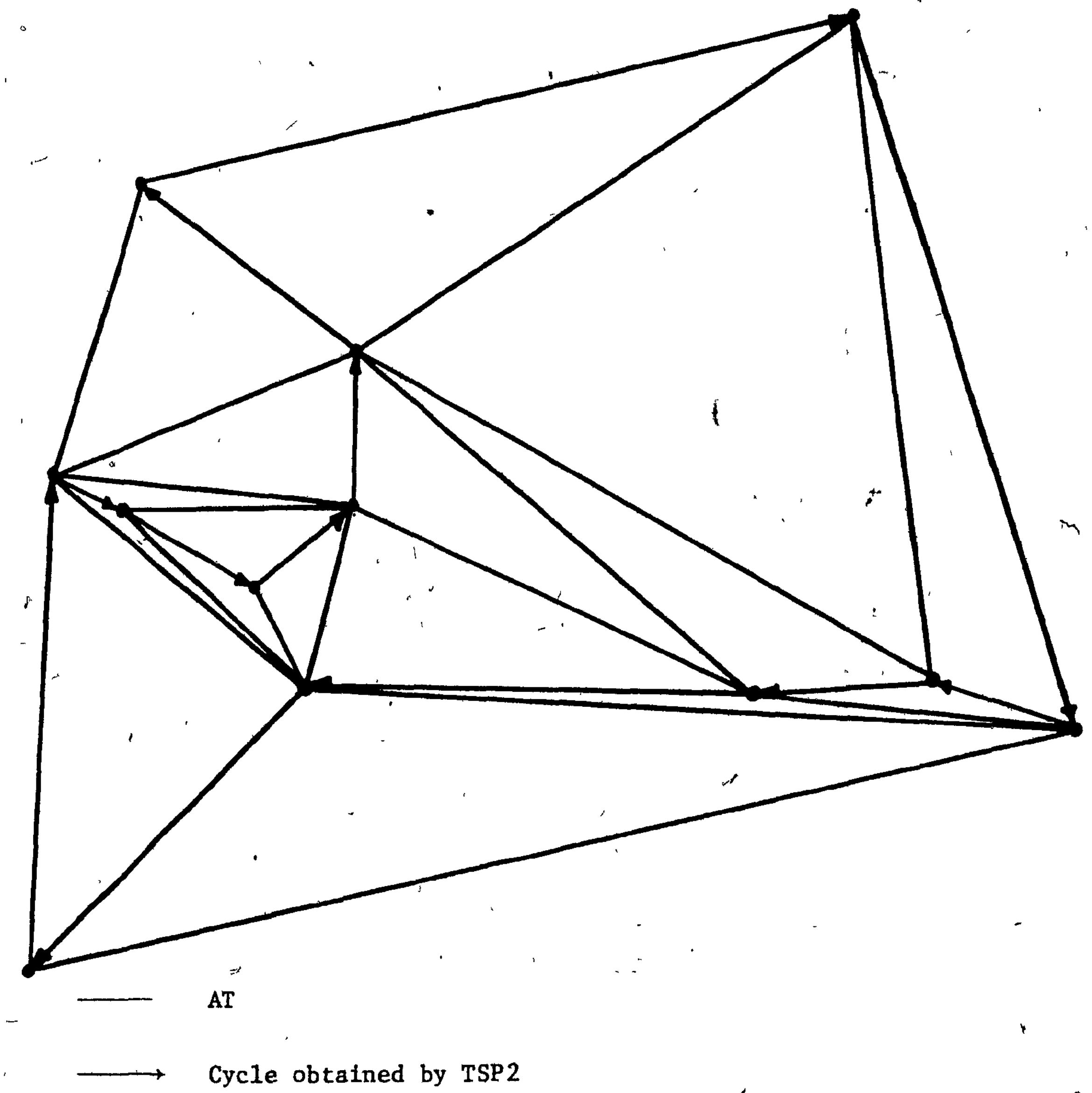


Figure 3.8

3.2.3.2 Analysis of Algorithm TSP2

Let d be a random variable denoting the degree (i.e. number of neighbors) of a node in the AT and taking the value d_i for node i . Also let t be the number of edges in the AT . Hence

$$\sum_{i=1}^{n} d_i = 2t$$

and
$$E(d) = \frac{2t}{n}$$

Now, when node i is connected to node j in the process of constructing a cycle, the number of branching possibilities of node j is - at least - reduced by one since j cannot branch back to i .
Thus, an upper bound on the number of chains for every starting point is given by

$$[E(d) - 1]^{n-1} \quad .$$

In the following theorem we develop an expression for t and show that

$$2n-3 \leq t \leq 3n-6 \quad .$$

Theorem 3.1 Given n points in the plane, the number of edges in a triangulation of the points is given by

$$t = (2n-3) + (n-h)$$

where h is the number of edges on the CH of the points.

Proof:

The maximum value that h can take is n . In that case, the number of edges of the triangulation is given by:

n edges on the convex hull

+

(n-3) edges connecting a node to its (n-3) non - (CH) neighbors.

This is illustrated in Figure 3.9 for n = 8 .

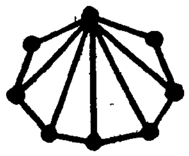


Figure 3.9

We have just proved that 2n-3 is the minimum value t can take.

Since the minimum value of h is 3 , the upper bound of 3n-6 follows. This is the maximum number of edges a plane graph can have. A triangulation with $t = 3n-6$ is called maximal and can be obtained by, for example, placing the n points as $\lfloor n/3 \rfloor$ concentric triangles (see Chapter 7).

To prove that the formula is valid in general, assume it is true for h-1 , i.e.

$$t = (2n - 3) + (n-h+1)$$

If we now increase the number of nodes on the convex hull by one, by pulling an interior point outwards, as shown in Figure 3.10, the number of edges of the triangulation is then reduced by one,

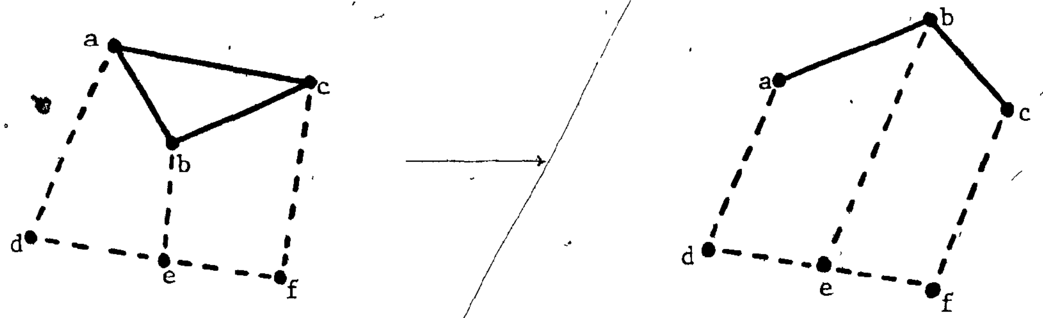


Figure 3.10

i.e. $t = (2n-3) + (n-h)$.

Q.E.D.

The expected values of t and d are given by

$$E(t) = (3n-3) - E(h)$$
$$E(d) = 2E(t)/n = 6 - (\frac{6 + 2E(h)}{n})$$

where E(h) is the expected number of edges on the CH . It follows that an upper bound on the expected running time of the algorithm is given by $E(h) \cdot [E(d) - 1]^{n-1}$.

Table 3.2 shows the values of E(h) - as given in Chapter 8 for a uniform distribution of the points in the unit square - and the corresponding computed E(d) for various n . We note that $\{E(h) \cdot [E(d) - 1]^{n-1}\}$, although an upper bound, is less than $\frac{(n-1)!}{2}$ for $n \geq 10$.

n	E(h)	E(d)
3	3	2.0
4	3	3.0
5	4	3.2
6	4	3.6
7	5	3.7
8	5	4.0
9	5	4.2
10	5	4.4
20	7	5.0
30	8	5.2
40	9	5.4
50	10	5.4
60	10	5.5
70	11	5.6
80	11	5.6
90	11	5.6
100	11	5.7

Table 3.2

To get more insight, we stochastically estimated by a Monte Carlo experiment the actual expected number of Hamilton chains and cycles present in a random triangulation, for various values of n and a uniform distribution of the points in the unit square. These values are listed in Table 3.3 .

n	E(chains)	E(cycles)
4	2	2
5	3	2
6	7	4
7	11	6
8	33	15
9	46	23
10	104	45
11	186	72
12	594	196
13	674	376
14	830	566

Table 3.3

For n = 8 , e.g., algorithm TSP2 will have to search through 5 x 33 = 165 cycles on the average before yielding an answer. Experiments with the algorithm (described in the next section) lead us to conjucture that this answer is almost always optimal.

3.2.3.3 Experiments with Algorithm TSP2

The following experiment was conducted several times for various values of n:

1. A set of points was placed in the unit square using a uniform random number generator.
2. A solution was sought using algorithm TSP2 .
3. The optimal solution was obtained by exhaustive search.

The TSP2 solution was always optimal.

This result led us to ask the following question: "How good is a random solution composed only of edges of the AT as compared to a purely random solution."

For small values of n , i.e. $4 \leq n \leq 12$, the reference point was the optimal tour obtained by exhaustive search. For $25 \leq n \leq 100$, the various tours were compared to 1.102*MST , an approximation of the optimal tour derived in Chapter 5. The results are shown in Tables 3.4

and 3.5

n	Random/Optimal	Random from AT/Optimal
4	1.056	1.004
5	1.175	1.043
6	1.386	1.091
7	1.466	1.114
8	1.636	1.127
9	1.708	1.141
10	1.862	1.163
11	1.866	1.179
12	1.888	1.194

Table 3.4

n	Random/1.102*MST	Random from AT/1.102*MST
25	3.642	1.365
50	4.949	1.422
75	6.035	1.489
100	6.847	1.517

Table 3.5

Each entry in Tables 3.4 and 3.5 is an average over 100 trial graphs randomly generated for every value of n by generating uniformly and independently the two cartesian coordinates of n points in the unit square. As the tables show a random cycle chosen from the AT is definitely superior to a purely random one.

At this point we believe that a question is legitimate: "What if the AT does not contain a Hamilton cycle?" To answer this question we make three remarks:

1) Determining a priori if a graph contains a Hamilton cycle is one of the most difficult unsolved problems in graph theory [Gael]. We shall not attempt to examine this problem here even for the special case of the AT.
2) Most authors [Rob1, Rob2, Se] assume tacitly that their graphs are dense enough (i.e. contain a sufficiently

large number of edges) to possess at least one Hamilton cycle. We shall follow them in this assumption.

3) The algorithm does not look for Hamilton cycles but, rather, for Hamilton chains - a less restricted structure.

The algorithm was tested on several hundreds of graphs. In each case the AT contained numerous Hamilton cycles. (For a further discussion of triangulations, see Chapter 7).

3.2.4 Better Algorithms for the ETSP based on the AT

We saw that algorithm TSP2 was not very efficient with respect to running time. Two variants are suggested in this section that are considerably faster.

3.2.4.1 Algorithm TSP2.1

<u>Step 1</u> Get the AT.

<u>Step 2</u> Enumerate all different Hamilton cycles whose edges belong to the AT.

<u>Step 3</u> Choose the cheapest cycle: this is an approximate solution to the TSP.

This algorithm - which is, for obvious reasons, many times faster than TSP2 - suffers from the following drawbacks:

1) Running time is still handicapped by the exhaustive nature of Step 2;
2) The possibility that an AT may not contain an HC is now present.

Our experience with algorithm TSP2 gave us a very good confidence in the quality of the answer it provided; in contrast TSP2.1 fails to find the optimal solution to a 6-city problem as shown in Figure 3.11 .

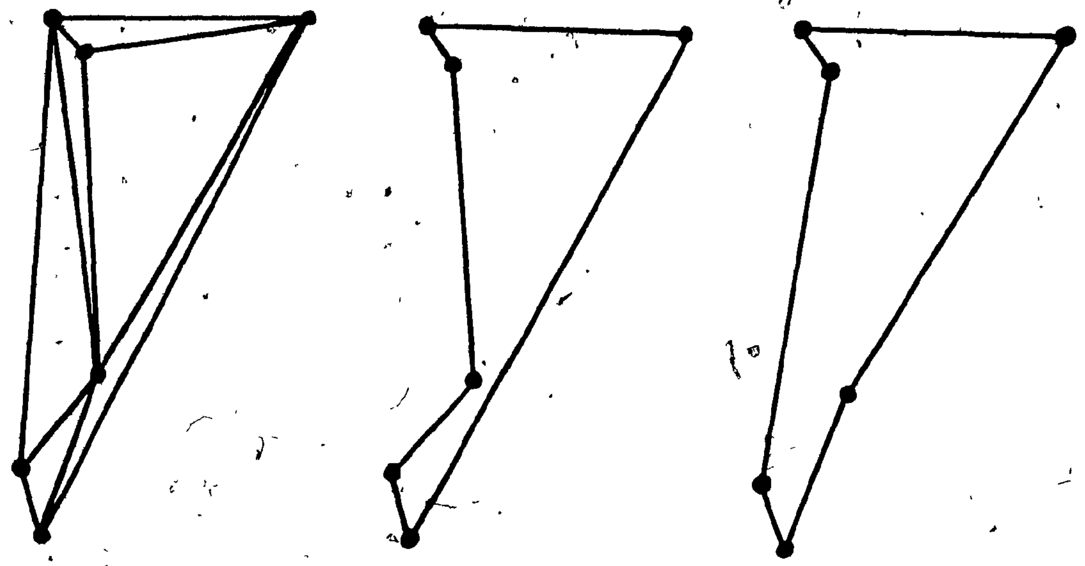


Figure 3.11

We therefore consider another variant of TSP2 .

3.2.4.2 <u>Algorithm TSP2.2</u>

<u>Step 1</u> Get the AT .

<u>Step 2</u> Get the CH .

<u>Step 3</u> Use any heuristic technique to obtain n good TSP tours.

<u>Step 4</u> Within the tours found in Step 3, consider only the set S of edges joining a point on the CH to an interior point. Let $T \subset S$ such that $T = \{t : t \notin AT\}$. If $T = \emptyset$ then $T = S$. Among the edges of T choose the one with the highest frequency of occurrence: let it be (a,b) .

<u>Step 5</u> Enumerate at most n^2 Hamilton chains with end-points a and b and whose edges belong to the AT .

<u>Step 6</u> The shortest chain found in Step 5, plus edge (a,b) is an approximate solution to the TSP .

The two examples of Figures 3.12 and 3.13 illustrate the algorithm. In each example a set of points and their AT are shown along with the Hamilton cycle found by algorithm TSP2.2 . In Figure 3.12, the points were drawn at random from a uniform distribution over the unit square. Figure 3.13 is from [L1]. Edge (a,b) chosen by the algorithm in Step 4 is also shown in each case. Note that in both examples the cycle obtained by TSP2.2 is the SHC.

Before concluding the description of TSP2.2, we remark that the algorithm used in Step 3 should allow us to obtain n different good tours. The nearest-neighbor method [Beol] and the heuristic technique of [Ka] are both $0(n^2)$ and have this property of yielding different tours for different starting cities. Another algorithm which shares this property is the nearest-insertion of [Rosl] which gets an approximate solution to the ETSP in $0(n \log n)$ as shown in [Sh2]. A very good tour could also be obtained by the $0(n \log n)$ algorithm of [Kar3].

3.2.4.3 Analysis of TSP2.2

From Table 3.6 it follows that TSP2.2 is $0(n^3)$.

Step	Complexity
1	$0(n^3)$
2	$0(n \log n)$
3	$0(n^3)$
4	$0(n^2)$
5	$0(n^3)$
6	$0(1)$

Table 3.6

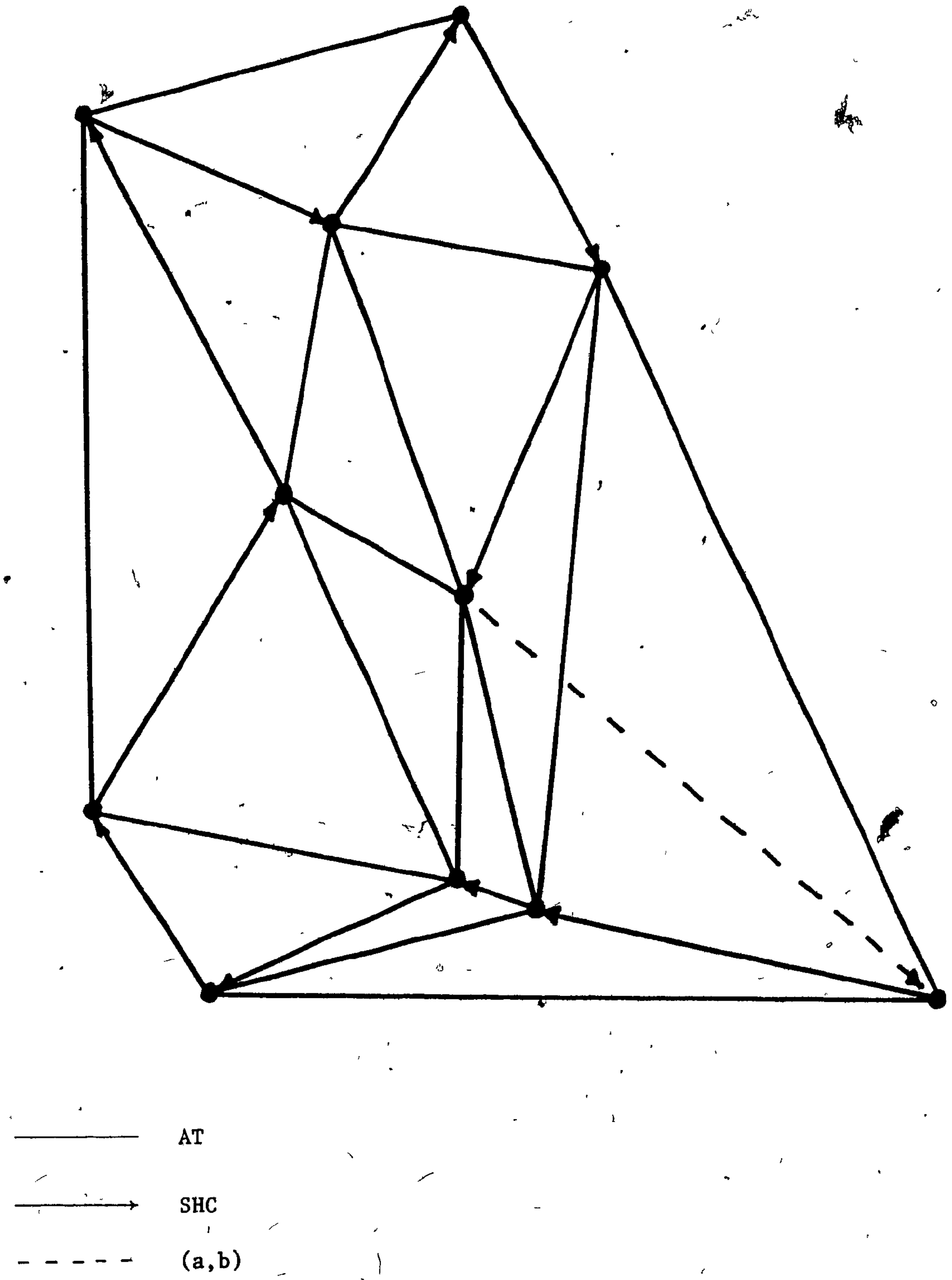


Figure 3.12

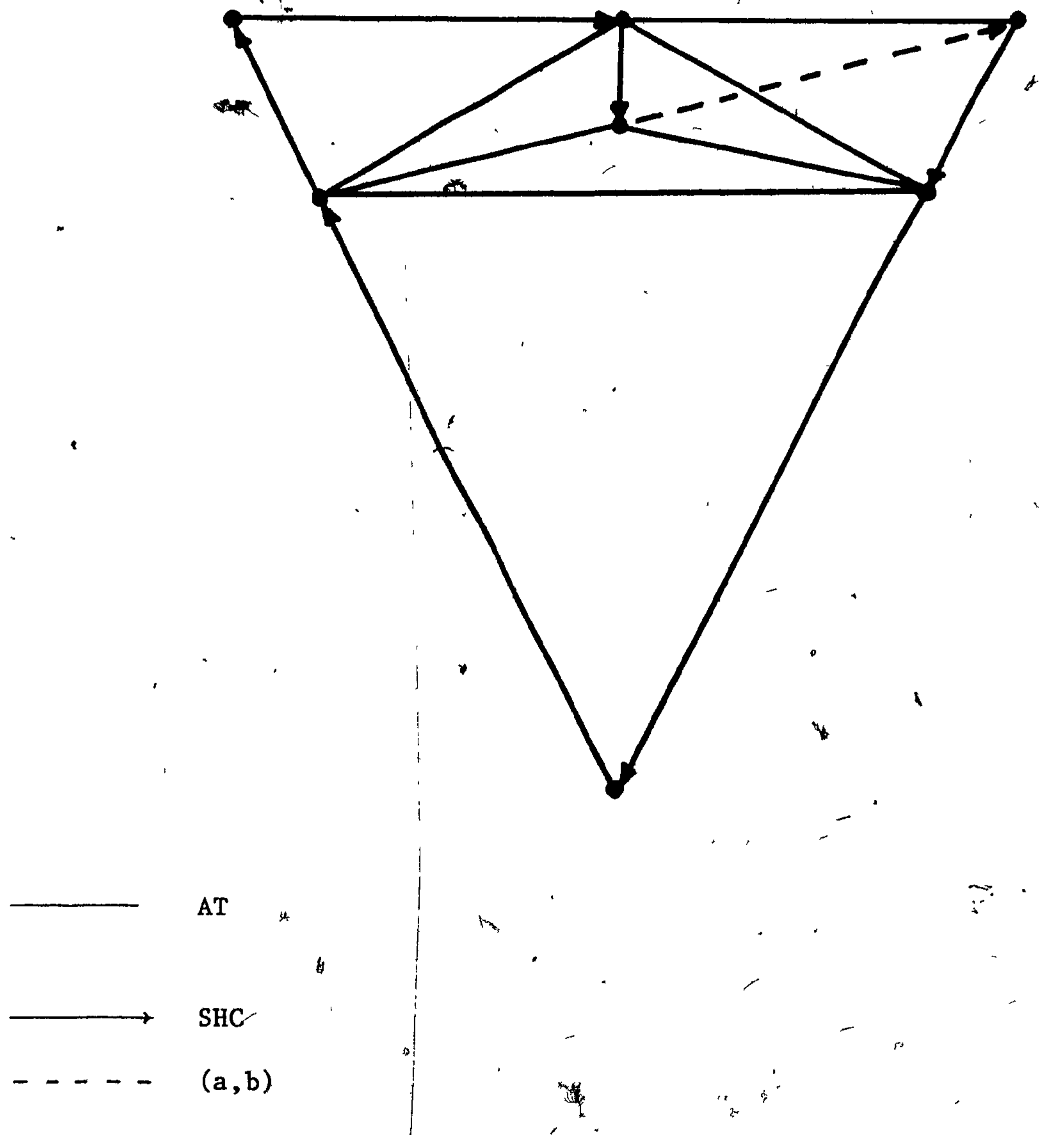


Figure 3.13

3.3 Conclusion

The experiments described in this chapter provide an empirical evidence that triangulations of points in the plane can be used to obtain good approximate solutions to the Euclidean Traveling Salesman Problem. Very little is known, however, about the geometric properties of triangulations. Our experience leads us to believe that triangulations could be a powerful tool in solving combinatorial optimization problems in the plane. It is hoped that the results obtained in this chapter will stimulate an interest in the study of the intrinsic properties of triangulations.

Chapter Four
The Shortest Hamilton Chain of a Graph and the TSP

In Chapter 3 a solution to the ETSP was obtained via enumeration of a class of Hamilton chains. The same approach will be used here to attack the general (i.e. not necessarily Euclidean) symmetric TSP. In the Euclidean case the method relied primarily on a planar structure, the AT; in this chapter the tool used is the MST. Basically, a sequence of transformations is applied to the MST, ultimately yielding the shortest Hamilton chain and hence an approximate solution to the TSP. A bound on the quality of the solution is provided, as well as an empirical estimate of the expected run-time of the algorithm.

4.1 An approximation algorithm for the TSP

The propositions of section 3.2 lead to the following algorithm for getting an approximate solution to the TSP, given a weighted complete graph on n nodes.

Algorithm TSP3

Step 1 Find the shortest Hamilton chain of the graph.

Step 2 Connect the end points of the chain: the resulting Hamilton cycle is thus an approximate solution to the TSP.

Proposition 4.1

When the cost matrix obeys the triangular inequality, the solution obtained by algorithm TSP3 is at most 50% more expensive than the optimal solution.

Proof:

We want to show that $\frac{\text{approximate solution}}{\text{exact solution}} \leq \frac{3}{2}$.

Since,

$$\frac{\text{approximate solution}}{\text{exact solution}} = \frac{\text{shortest Hamilton chain + last edge}}{\text{shortest Hamilton cycle}}$$

$$= \frac{\text{shortest Hamilton chain}}{\text{SHC}} + \frac{\text{last edge}}{\text{SHC}} ,$$

we prove that

$$\frac{\text{shortest Hamilton chain}}{\text{SHC}} \leq 1 \text{ and } \frac{\text{last edge}}{\text{SHC}} \leq \frac{1}{2} .$$

1) Removal of an edge from the SHC yields a Hamilton chain which is at least equal to the shortest Hamilton chain; in other words

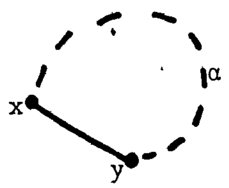
   (SHC - edge) $\geq$ shortest Hamilton chain.

   Thus, SHC $\geq$ shortest Hamilton chain.

2) Let (x, y) be the longest edge in the graph. There are two possibilities:

   a) <u>(x, y) is in the SHC</u>

   Let α be the length of the chain resulting from the exclusion of edge (x, y) from the SHC as shown in Figure 4.1



SHC

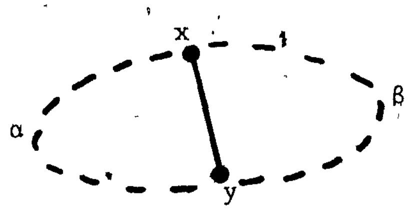
Figure 4.1

From the triangular inequality it follows that $xy \leq \alpha$ and that $2xy \leq \alpha + xy$. Hence,

$$xy \leq \frac{SHC}{2} .$$

   b) <u>(x,y) is not in the SHC</u>

   Let α and β be the lengths of the two chains with end points x and y and whose concatenation yields the SHC as shown in Figure 4.2 .



SHC

Figure 4.2

Again from the triangular inequality, $xy \leq \alpha$, $xy \leq \beta$ and $2xy \leq \alpha + \beta$. Hence,

$$xy \le \frac{SHC}{2}$$

Q.E.D.

The difficulty with algorithm TSP3 lies of course in the fact that finding the shortest Hamilton chain is a hard task in its own right. In fact the complexity of that problem is of the same order as that of the TSP. Nevertheless various techniques for obtaining the shortest Hamilton chain were given in [Chr2, Chr5, He2, He3]. We now describe a heuristic technique called "the vertex penalty method", first given in [Chr2] and then developed in [Chr5].

4.2 The Vertex Penalty Method

The essence of the technique is to force the MST of the set of points to become a Hamilton chain by penalizing vertices with degree other than 2. Let d_i be the degree of vertex x_i in the MST ,

c_{ij} ($\equiv c(i,j)$) the cost of edge (i, j)

and $p(i)$ a penalty imposed on vertex x_i . Then,

Step 1 Get the MST.

Step 2 If the MST is a Hamilton chain, stop.
Else continue

Step 3 For every x_i with $d_i \neq 2$,

$$c_{ij}^{new} = c_{ij}^{old} + p(i)$$

and

$$c_{ji}^{new} = c_{ij}^{new} \text{, for all } j \text{ .}$$

Go to Step 1 with the new cost-matrix .

In [Chr2] it is shown that the Hamilton chain obtained in Step 2 is the shortest Hamilton chain of the graph and in [Chr5] the following penalizing strategy is prescribed for use in Step 3 .

1) When $d_i > 2$

a) Remove from the MST just one of the links (x_i, x_r) incident at

x_i thus creating two subtrees T_1 and T_2 as shown in Figure 4.3.

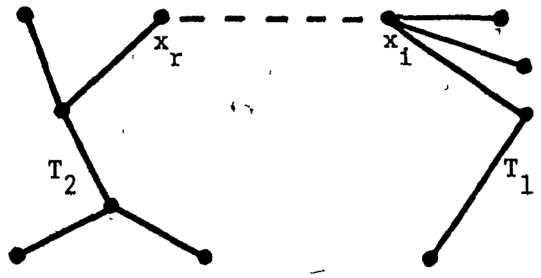


Figure 4.3

b) Find the least cost link joining these two subtrees; i.e. find the link (x_j^r , x_k^r) such that

$$c(x_j^r , x_k^r) = \min_{\substack{x_j \varepsilon T_1 \\ x_k \varepsilon T_2 \\ x_j \neq x_k \neq x_i}} c(x_j , x_k) ,$$

as shown in Figure 4.4 .

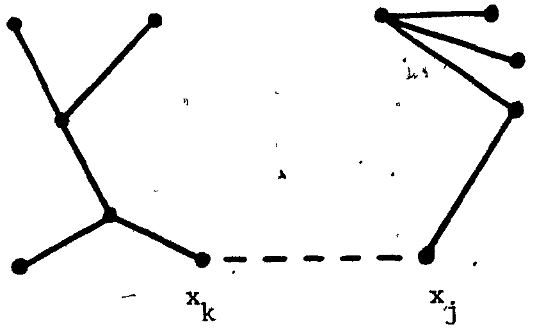


Figure 4.4

Then, $p(i) = \min_{(x_i , x_r) \varepsilon MST} [c(x_j^r , x_k^r) - c(x_i , x_r)]$.

In other words, p(i) is the minimum positive penalty which when applied to x_i alone causes d_i to be reduced by one.

2) <u>When $d_i = 1$</u>

a) Add a link (x_i , x_r) to the tree as shown in Figure 4.5 .

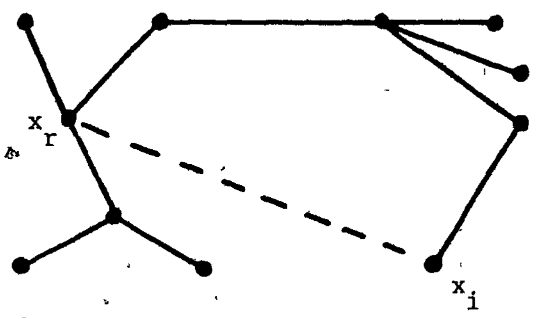


Figure 4.5

b) Let S_{ri} be the set of links in the path from x_r to x_i - excluding the last link incident to x_i : if (x_i , x_r) is added to the tree and any one of the links in S_{ri} is removed, another tree results in which $d_i = 2$.

Then, $p(i) = \min_{x_r} [c(x_i , x_r) - \min_{(x_j,x_k)\varepsilon S_{ri}} \{c(x_j , x_k\}]$.

In other words, $p(i)$ is the maximum negative penalty which when applied to x_i alone causes d_i to become 2 .

We have used the heuristic technique and penalizing strategy just outlined in Step 1 of our approximation algorithm TSP3 . Two minor modifications were however introduced that are described below.

1) In the case of $d_i = 1$, $p(i)$ is, by definition, the maximum negative penalty (i.e. least negative) which when applied to node x_i alone causes d_i to become 2 . It follows that

$$p(i) \underline{\Delta} \max_{x_r} [\max_{(x_j,x_k)\varepsilon S_{ri}} \{c(x_j,x_k)\} - c(x_i,x_r)] .$$

We note that the expression in square brackets:

a) is always negative because $c(x_i,x_r)$ is larger than any edge in the tree (or else it would replace an edge of the tree),

b) is maximized for every S_{ri} by choosing the longest edge in the path to be (x_j, x_k) .

The expression above, therefore, agrees with the definition and is the correct negative penalty for $d_i = 1$. To show the difference with the expression given in [Chr5] we write

$$-p(i) = \max_{x_r}(-[c(x_i, x_r) - \max_{(x_j,x_k)\varepsilon S_{ri}} \{c(x_j, x_k)\}])$$

$$= -\min_{x_r} [c(x_i, x_r) - \max_{(x_j,x_k)\varepsilon S_{ri}} \{c(x_j, x_k)\}] .$$

Note that the minus sign has been introduced and max now replaces min inside the square brackets.

2) The penalties (positive or negative) that we use are given by $(1 + \varepsilon)\ p(i)$ where ε is very small ($\varepsilon = 10^{-4}$, say). To illustrate this point we take the following example where n=4 , the MST is indicated by solid lines and edge costs are as shown in Figure 4.6 .

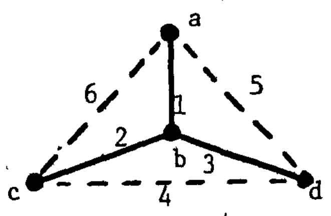
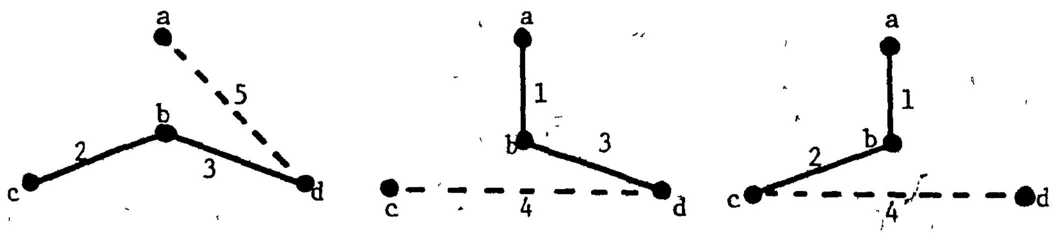


Figure 4.6

a) Positive penalties

Node b is to be penalized. There are three alternatives depicted in Figure 4.7 which show that p(b) = 1 .



ad-ab = 5-1=4 cd-cb = 4-2=2 dc-db = 4-3=1

Figure 4.7

After adding this penalty to the edges incident at b the new edge costs will be as in Figure 4.8 .

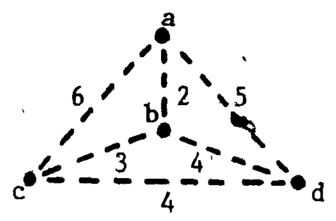


Figure 4.8

One can see that edges (b, d) and (c, d) are equal and the MST may very well be the same as the original one (see Figure 4.9) causing the procedure to never converge.

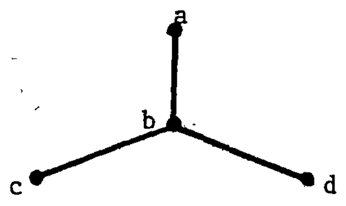


Figure 4.9

If, however, the penalty is taken as (1 + ε) p(b) we get the new edge costs of Figure 4.10 .

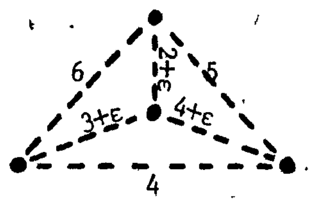


Figure 4.10

The new MST shown in Figure 4.11 is now as required.

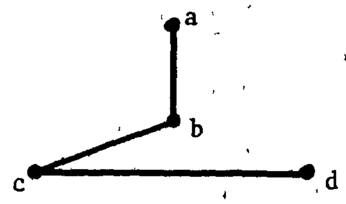
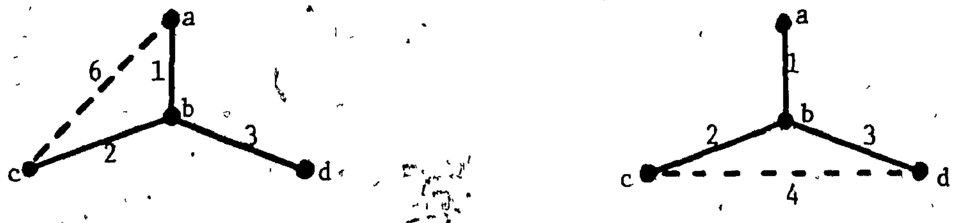


Figure 4.11

b) <u>Negative penalties</u>

Node c is to be penalized. There are two alternatives depicted in Figure 4.12 which show that p(c) = -1 .



cb-ca = 2 -6 = -4 db - dc = 3 -4 = -1

Figure 4.12

Adding this penalty to the edges incident at c yields the new edge costs of Figure 4.13 .

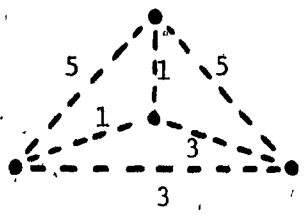


Figure 4.13

Edges (b, d) and (c, d) have the same cost and the new MST could very well be the same as it was originally, see Figure 4.14 .

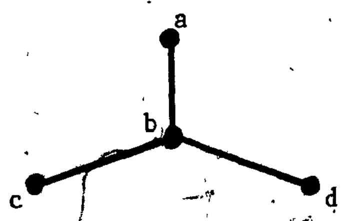


Figure 4.14

If, however, the penalty is taken as $(1 + \varepsilon)\ p(c)$ we get the new edge costs of Figure 4.15 .

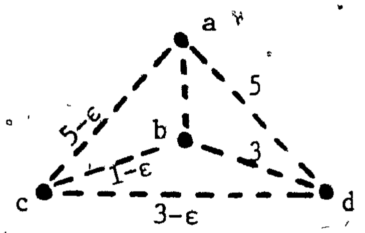


Figure 4.15

and the new MST shown in Figure 4.16 is now as required.

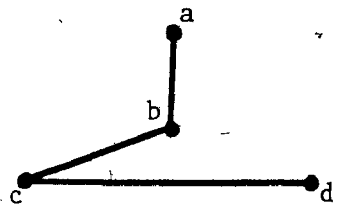


Figure 4.16

When all penalties are applied to a, b, c and d, the MST and edge costs are as shown in Figure 4.17 .

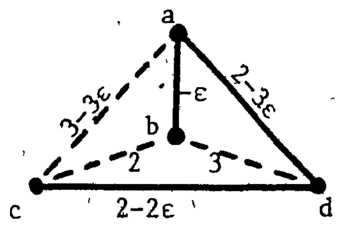


Figure 4.17

4.3 A Monte Carlo Experiment

Random points were generated uniformly in the unit square. In every case the approximate solution to the TSP was compared to the exact solution obtained by exhaustive enumeration. Table 4.1 summarizes our experience with the algorithm. It is worth mentioning that termination was observed in all cases considered. Furthermore the average number of iterations required before convergence was roughly $\lfloor n/4 \rfloor$ (taking only into account cases where initial MST $\neq$ Hamilton chain).

n	Average number of iterations before convergence	Average over 100 runs of TSP3/TSP1
4	1	1.000
5	1	1.005
6	1	1.007
7	1	1.010
8	2	1.014
9	2	1.015
10	2	1.026
11	2	1.058
12	3	1.061

Ratio of answer provided by TSP3 to exact answer provided by TSP1

Table 4.1

4.4 An Empirical Estimate of the Expected Run-Time of TSP3

The cost of getting the MST is $0(n^2)$. Since k iterations are needed for convergence the overall complexity is $0(kn^2)$. As mentioned before the observed k was $\lfloor n/4 \rfloor$; this means that algorithm TSP3 has an (empirically-estimated) expected run-time of $0(\frac{n^3}{4})$ and guarantees - when the triangular inequality holds - an approximate solution to the TSP which is no worse than $\frac{3}{2}$ times the optimal.

4.5 A Modification of TSP3

In a TSP, if an edge has a very high probability of being in the optimal solution, then the following algorithm is suggested by Proposition 3.3

Algorithm TSP3.1

```
Step 1  Use a very fast heuristic technique to obtain a number of
        good tours.
Step 2  Within the tours found in Step 1, choose the edge with the
        highest frequency of occurrence. Let this edge be (a,b) .
Step 3  Get the shortest Hamilton chain with end points a and b .
Step 4  The chain obtained in Step 3 plus edge (a,b) is an approx-
        imate solution to the TSP.
```

This algorithm which is believed to yield better answers in shorter times is suggested for future research.

4.6 Conclusion

An algorithm, based on a modification of the "reward-punishment" method originally proposed in [Chr2], has been tested on the TSP. Theoretically, the algorithm produces an answer no worse than 1½ times the optimal. A Monte Carlo experiment showed the tours obtained to be very close to the exact solutions. An empirical estimate of the expected run-time of the algorithm is $O(n^3/4)$. It would be interesting to find out whether this method is applicable to other combinatorial optimization problems.

Chapter Five
Experiments with Efficient Algorithms for the TSP

Various concepts from graph theory have greatly influenced the work on approximation algorithms for the TSP. These include, the minimal spanning tree (MST), the optimal perfect matching (OPM), the Euler cycle (EC) and the Hamilton cycle (HC). In particular, the contribution of the MST was twofold: it provided good estimates for the optimal tour [Chr4, Ei] as well as efficient algorithms for an approximate solution [Chr6, Ros1].

In this chapter we analyze two algorithms for the general TSP: the first from [Ros1] is a typical sequential tour-building technique (section 5.1); the second is essentially based on the above-mentioned graph-theoretic concepts (section 5.2). In section 5.3 a variant of the second algorithm - that addresses the directed TSP - is investigated.

5.1 Experiments with a sequential tour-building approximation algorithm

In dealing with a class of combinatorial problems - like the TSP - for which all known exact algorithms have a running time that grows exponentially with the size of the input, it is often useful to estimate the expected solution, or put some bounds on it. This estimate (or bound) can serve several purposes:

1) In some distribution management problems it is sometimes necessary to estimate the expected distance that would be involved in supplying customers - when the exact locations of the customers are not known in advance - in order to decide, for example, upon the number and locations of depots.
2) The branch-and-bound approach uses lower bounds to eliminate from further consideration whole parts of the decision tree that would otherwise have to be investigated.
3) Finally, and most important for our purpose, when an approximation algorithm is tested, a lower bound serves as a reference point against which near-optimal solutions are compared.

The present section is concerned with this last application. In section 5.1.1 estimates and bounds for an optimal TSP tour are discussed. Experiments with a sequential tour-building approximation algorithm are described in section 5.1.2 . In section 5.1.3 a possible improvement on the quality of the solution provided by this algorithm for the case of an ETSP is investigated.

5.1.1 Estimates and bounds for an optimal TSP tour

5.1.1.1 An asymptotic estimate of the expected value of the optimal tour

The length of the SHC through n points in a bounded plane region of area A is shown in [Be] to be almost always asymptotically equal to $K\sqrt{nA}$, for large n , where K is a constant of proportionality. This constant - which is independent on the shape of the region - was estimated in [Ham] as being equal to 0.75. The same theoretical and experimental results of [Be] and [Ham] are obtained in [Ei].

5.1.1.2 Lower bounds on the optimal tour

These are described in [Ei] and [Chr4].

i) The sum of the shortest links

In an HC each node has degree two. Let d_{j1} and d_{j2} be the shortest and next shortest edges connected to node j in the complete graph. Thus a lower bound on the SHC is given by

$$B_1 = \frac{1}{2} \sum_{j=1}^{n} (d_{j1} + d_{j2}) \ .$$

ii) The minimal spanning tree

If the longest edge is removed from the SHC we are left with a spanning tree which may or may not be the MST. Thus

$$\text{SHC} - \text{longest edge in SHC} \geq \text{MST} \ .$$

Let d_{j2} denote the distance separating node j from its second neighbor. Then

$$\max_j (d_{j2}) \leq \text{longest edge in SHC} \ .$$

It follows that $B_2 = \text{MST} + \max_j (d_{j2})$ is a lower bound on the SHC.

iii) The assignment problem

The assignment problem (AP) is represented by the following linear program

$$\min \omega = \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} \; x_{ij}$$

subject to $\sum_i x_{ij} = \sum_j x_{ij} = 1$ for all i and j ; and

$$x_{ij} = 0 \text{ or } 1 \; .$$

If we let $c_{ii} = \infty$, for all i , and add the condition that the solution must be a single HC, then the solution to the AP is a solution to the TSP. Noting that the AP (without the extra condition that the tour be a cycle) is much easier to solve than the TSP we get

$$B_3 = \min \omega \; ,$$

another lower bound on the SHC.

We believe that the lower bounds B_1, B_2 and B_3 are of better value than $0.75 \sqrt{An}$ for estimating the SHC when one is testing the performance of an approximation algorithm. This superiority is due to the following reasons:

1) they are problem-dependent; in other words they provide a value of the expected optimal tour which not only depends on the number of cities but also on the structure of a given instance of the TSP;
2) they are not asymptotic estimates and hence can be practically used for small as well as for large values of n ;
3) they neither over - nor underestimate the optimal solution;
4) they are not restricted to the special case where the inter-city distance is a metric.

It should be pointed out that B_1, B_2 and B_3 are listed in the order of increasing quality and computational difficulty. The easiest bound to compute, B_1 , relatively underestimates the solution. A better estimate is provided by B_2 and several efficient algorithms exist [Di1, Di2, Kru, Pri]* to obtain the MST of a complete graph very rapidly. The sharpest bound, B_3 , can also be computed in polynomial time but requires a little bit more effort. Further improvements on B_2 and B_3 are described in [Ei] and [Chr4] to obtain even better bounds at the cost of additional computation time.

An estimate of the expected solution that - in our opinion - combines simplicity and quality will now be derived for the case where the intercity distance is a metric. It has been shown [Gibl] that for a large number n of points in a region of area A , the expected length of the MST is approximately equal to $0.68 \sqrt{nA}$. It follows that, for large n ,

$$\frac{SHC}{MST} = \frac{0.75\sqrt{An}}{0.68\sqrt{An}} = 1.102 \quad .$$

An estimate of the expected value of the SHC is thus given by

$$B_4 = 1.102 * MST \quad .$$

Experiments with this estimate showed that it is tighter than B_1 and B_2 ; further it is easier to compute than B_3 .

5.1.1.3 An upper bound on the optimal tour

In [Fe] it is shown that there always exists a Hamilton chain through n points in a unit square of length less than

$$\sqrt{2n} + 1.75 \quad .$$

It follows that

$$SHC \le \sqrt{2}\,(1 + \sqrt{n}) + 1.75 \quad .$$

5.1.2 Experiments with NEARINSERT

NEARINSERT is an approximation algorithm for the TSP described in [Ros1, Ros2]. When the intercity distance is a metric the solution obtained is always less than twice the optimal.

5.1.2.1 NEARINSERT

Step 1 Start with a subgraph consisting of a single node, say node i .

Step 2 Find a node k , such that c_{ik} is minimal; add this node to the subgraph and construct an HC (for the subgraph) consisting of two occurrences of the edge (i, k) .

Step 3 Given an HC containing a subset of the nodes, find the uncontained node k , closest to any contained node (i.e. find a minimal c_{mj} , such that node m is in the cycle and j is not, and take k = j) .

Step 4 Given k find an edge (i, j) in the HC for the subgraph such that $c_{ik} + c_{kj} - c_{ij}$ is minimal.

Step 5 Given k and the edge (i, j) obtain a new HC by replacing edge (i, j) with edges (i, k) and (k, j) .

Step 6 If there are any remaining nodes not in the HC go to Step 3 ;
Else stop.

5.1.2.2 A bound on the approximate solution

We now give a simple argument to prove that the solution obtained by NEARINSERT is never greater than twice the optimal when the distance measure is a metric. First we note that the nodes enter the cycle in exactly the same order in which they would have entered the MST when the latter is obtained by the algorithm of [Pri].

Since $\text{MST} \le \text{SHC} - \text{longest edge}$

i.e. $\text{MST} < \text{SHC}$,

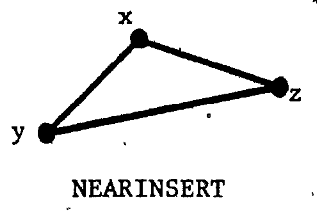
it follows that if we can show

$$\text{NEARINSERT} \le 2\ \text{MST}$$

we would have immediately

$$\text{NEARINSERT} \le 2\ \text{SHC} \quad .$$

A simple inductive argument is now used to demonstrate that NEARINSERT $\le$ 2MST . This is obviously true for n = 3 ; Figure 5.1 shows that



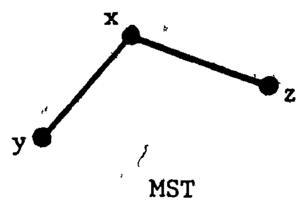
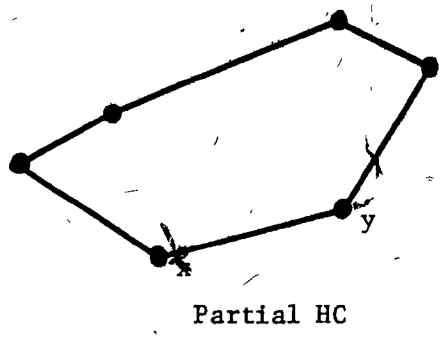


Figure 5.1

$$\text{NEARINSERT} = c_{xy} + c_{xz} + c_{yz} \le c_{xy} + c_{xz} + c_{xy} + c_{xz}$$
$$= 2\,(c_{xy} + c_{xz}) = 2\ \text{MST}\ .$$

Now, assume this is true for a number of nodes m , $3 \le m < n$, i.e. assume a partial HC has been constructed by NEARINSERT which is no more than twice longer than the corresponding MST (see Figure 5.2) .



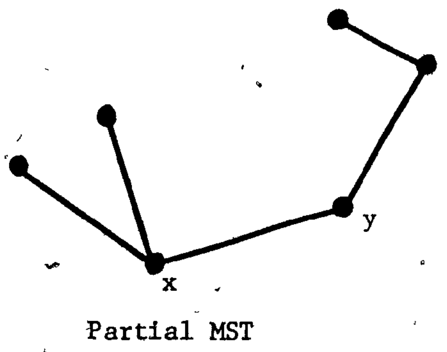
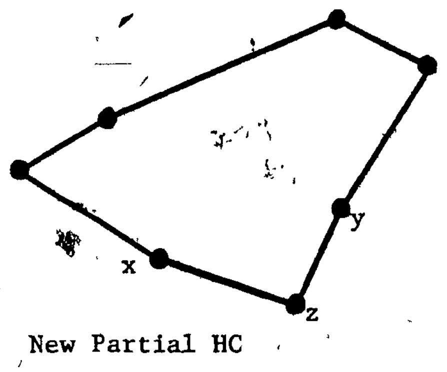


Figure 5.2

An uncontained node z is now inserted adjacent to x in the partial MST as shown in Figure 5.3 .



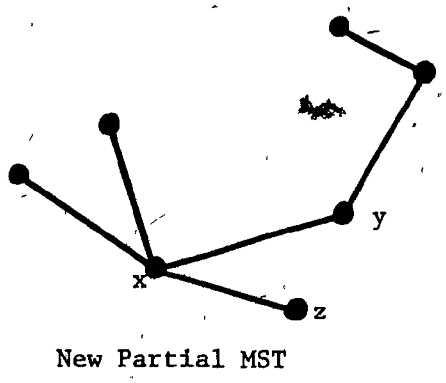


Figure 5.3

The increase in total length will be $c_{xz} + c_{zy} - c_{xy}$ for the HC, and c_{xz} for the MST. Since

$$c_{zy} < c_{xy} + c_{xz}$$

it follows that

$$c_{xz} + c_{zy} - c_{xy} \le 2\, c_{xz} \quad .$$

We have tacitly assumed that z was inserted between x and its neighbor y in the partial HC. If this was not the case, but, instead, z was inserted between u and v (say) then evidently

$$c_{uz} + c_{zv} - c_{uv} \le c_{xz} + c_{zy} - c_{xy} \quad .$$

Hence, in general,

$$\text{New partial HC} \le 2 \text{ New partial MST} \quad .$$

This completes the proof.

5.1.2.3 Experiments and results

Tables 5.1, 5.2 and 5.3 show the results obtained when applying the algorithm to randomly generated problems where the intercity distance is a metric. Every entry is an average over 100 problems.

n	NEARINSERT/B_4
25	1.287
50	1.276
75	1.274
100	1.272

Ratio of the answer provided by NEARINSERT to the estimated answer B_4 for various values of n .

Table 5.1

n	NEARINSERT/TSP1
4	1.0000
5	1.0113
6	1.0276
7	1.0439
8	1.0468
9	1.0511
10	1.0547
11	1.0738
12	1.0874

Ratio of answer provided by NEARINSERT to the exact answer provided by TSP1 for $4 \leq n \leq 12$.

Table 5.2

n	Average Running Time
25	0.033
50	0.130
75	0.275
100	0.485

Solution time in seconds required by NEARINSERT for various values of n

Table 5.3

From Table 5.1 it is seen that the solution obtained by NEARINSERT is about 27% more expensive than the estimate B_4 . In Table 5.2 , the problems are Euclidean in two dimensions and TSP1 refers to the exact solution obtained by the algorithm of Chapter 2. The run-time of the algorithm is approximately equal to $(0.5 \times 10^{-4} \times n^2)$ secs, on the average, see Table 5.3 . Note that it was shown in [Sh2] that, for the ETSP, NEARINSERT can be modified so that it has a run-time proportional to n log n . Various problems that have appeared in the literature were tried using NEARINSERT ; the results are displayed in Table 5.4 .

n	Source	Best known solution	NEARINSERT	% deviation	Run-time in seconds
10	[Ba]	378	399	5.5	0.005
20	[Cr]	246	344	39.8	0.020
25	[Hel]	1711	2003	17.0	0.030
33	[Ka]	10861	12520	15.2	0.050
42	[Ka]	699	834	19.3	0.080
48	[Hel]	11461	13157	14.7	0.110
57	[Ka]	12955	14667	13.2	0.160
100	[Kro]	21282	25785	21.1	0.480

Table 5.4

5.1.2.4 A modification for the ETSP

As Tables 5.1 to 5.4 show, NEARINSERT is quite fast in obtaining a satisfactory solution to the TSP. In the case of an ETSP the quality of the solution can even be improved upon by modifying Step 6 of the algorithm as follows:

```
Step 6'  If there are any remaining nodes not in the HC
         go to Step 3 ;
         Else repeat the following as long as possible :
         "Replace every intersecting pair of edges (a, b)
         and (e, d) by the pair (a, e) and (b, d)."
```

Note that if (a, b) and (e, d) intersect then, obviously,

$$c_{ae} + c_{bd} \leq c_{ab} + c_{ed} ,$$

as it can be seen from Figure 5.4 .

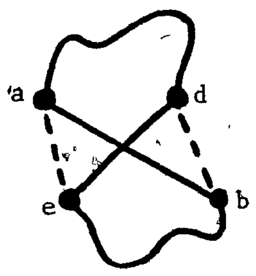


Figure 5.4

When the modified algorithm was tested, the results were rather discouraging as seen in Table 5.5 . The improvement on the tour is extremely small, on the average, almost nonexistent in the majority of cases and does not compensate for the additional time spent looking for intersections.

n	NEARINSERT/B_4
25	1.273
50	1.268
75	1.265
100	1.261

Average over 100 randomly generated ETSP's of the ratio of the answer provided by NEARINSERT (with some intersections removed) to the estimated answer B_4 for various values of n .

Table 5.5

This leads us to believe that the tour obtained by NEARINSERT is in fact very close to optimal and that the worst-case bound of (2 * optimal) is for the majority of cases overly pessimistic.

5.2 A new heuristic algorithm for the symmetric TSP

In this section we describe and empirically analyze a new heuristic algorithm for the TSP. The algorithm which has relatively limited storage and time requirements - yields a very good approximate solution to the problem. It combines the following ideas: k-optimality [Lin1, Lin2, Lin3, Lin4], perfect matching [Chr6], the minimal spanning tree [Chr6, O, Ros1], Euler and Hamilton cycles [Chr6], the first two of which are discussed in section 5.2.1 . A description of the algorithm as well as experimental results are the subject of section 5.2.2 .

5.2.1 2-optimal perfect matching

5.2.1.1 k-optimality

A traveling salesman tour is said to be k-optimal if it is (k-1)-optimal and no k edges can be replaced by k other edges to yield a cheaper tour. This concept - used in the literature under one form or the other - was first formalized and generalized in [Lin1].

The question of the applicability of a heuristic rule to various combinatorial problems is posed in [Wei]. In particular, the problem of determining the versatility of "k-optimality" was given as an example. We shall try to provide a very partial answer to this question by applying k-optimality to the matching problem. We define a 2-optimal perfect matching (TOPM) as a perfect matching no two edges of which can be replaced by two other edges to yield a cheaper matching. Efficient $O(n^3)$ algorithms exist for obtaining the OPM for a graph of n nodes [Gab, La2]. By using 2-optimality we propose to obtain a near-optimal matching in time proportional on the average to n^2. The algorithm to be described is a typical local neighborhood search algorithm (see Chapter 6) and each iteration requires $O(n^2)$ operations.

5.2.1.2 2-optimal perfect matching algorithm

Given a complete weighted graph with n nodes, where n is even.

Algorithm TOPM

```
Step 1  Start with an arbitrary matching $(m_1, m_2, \ldots, m_n)$.
        where $m_i$ is a node and $(m_i, m_{i+1})$, i odd,
        represents an edge of the matching.
Step 2  For $i = 1,3,5,\ldots,n-3$ do
            For $j = i+2, i+4, \ldots, n-1$ do
```

If $c_{m_i m_j} + c_{m_{i+1} m_{j+1}} < c_{m_i m_{i+1}} + c_{m_j m_{j+1}}$

then interchange m_{i+1} and m_j ;

If $c_{m_i m_{j+1}} + c_{m_j m_{i+1}} < c_{m_i m_{i+1}} + c_{m_j m_{j+1}}$

then interchange m_{i+1} and m_{j+1} .

Step 3. If an interchange took place in Step 2, go to Step 2 ; Else the matching on hand is 2-optimal, Stop.

5.2.1.3 Analysis of TOPM

We observe that Step 2 takes time proportional to n^2 . In fact, since $\frac{n}{2}$ is the number of edges in the matching, there are $\binom{n/2}{2}$ ways of choosing 2 edges out of n/2 and each choice gives two alternatives. Every iteration of Step 2 therefore requires $(\frac{n}{2}) * (\frac{n}{2} - 1)$ comparisons.

A very easy and efficient way of appraising the above algorithm is now described. We are mainly interested in evaluating the quality of the solution obtained and the time spent to obtain it. To achieve our first purpose we derive an upper bound for the optimal perfect matching. Given the SHC define two perfect matchings M_1 and M_2 such that,

$$M_1 \subset SHC \quad \text{and} \quad M_2 \subset SHC ;$$

$$M_1 \cap M_2 = \emptyset ;$$

$$M_1 \cup M_2 = SHC .$$

Assume without loss of generality that

$$M_1 \leq M_2 ;$$

hence $$M_1 \leq \frac{1}{2} SHC .$$

Now M_1 may or may not be the OPM ; it follows that

$$OPM \leq \frac{1}{2} SHC .$$

From the upper bound defined in 5.1.1.3 ,

$$\text{SHC} \le \sqrt{2}\,(1 + \sqrt{n}) + 1.75 \ ,$$

we have $$\text{OPM} \le \frac{1}{2}\,[\sqrt{2}(1 + \sqrt{n}) + 1.75] \ .$$

The ratio (TOPM/Upper bound) determines the quality of the solution.

A measure of the running time of the algorithm is the number of times Step 2 is executed. Table 5.6 shows the results of a Monte Carlo simulation where points were generated in the unit square of the Cartesian plane and the Euclidean distance used to represent the cost. Every entry is an average over 100 runs.

n	TOPM/Upper Bound	Number of executions of Step 2
10	0.310	2
20	0.360	3
30	0.378	3
40	0.389	3
50	0.404	4
60	0.407	4
70	0.413	4
80	0.420	4
90	0.424	4
100	0.426	4

Table 5.6

As it can be seen from Table 5.6 , TOPM's are very good approximations of OPM's that can be obtained in a very short amount of time, on the average proportional to n^2 . k-optimality has thus been used successfully for the matching problem to obtain a near-optimal solution at low expense.

5.2.2 Algorithm TSP4

Given a complete graph with n nodes, the following algorithm yields an approximate solution to the TSP. The condition imposed on the input is that the cost matrix be symmetric; the condition on the output is that every city be visited just once.

Step 1. Find the MST.

Step 2. For the complete subgraph formed by the nodes with odd degree in the MST, get a TOPM.

Step 3. Construct an Euler cycle through the Eulerian graph formed by the edges of the MST plus the edges of the TOPM.

Step 4. Starting with each of the nodes appearing in the EC, obtain a new Hamilton cycle. The cheapest of the HC's thus obtained, is chosen as the final answer. Stop.

Although the algorithm is quite straightforward every step deserves a few words of explanation.

Step 1 Any efficient algorithm [Di1, Di2, Pri] can be used to get the MST.

Step 2 The procedure described in section 5.2.1.2 serves to obtain a TOPM.

Step 3 That the graph resulting from the union of the edges of the MST and the TOPM is Eulerian follows from the fact that every node has even degree. A simple back-tracking procedure will generate the Euler cycle.

Step 4 The method of getting a Hamilton cycle from the EC is as follows:
Assume the EC is $(n_1, n_2, \ldots, n_{\ell-1}, n_\ell)$ where the n_i's represent the nodes of the graph and are not necessarily distinct. If two copies of the EC are placed contiguously,

$$n_1, n_2, n_3, \ldots, n_{\ell-1}, n_\ell, n_1, n_2, n_3, \ldots, n_{\ell-1}, n_\ell$$

one can build an HC by: starting at a node, moving to the right and introducing a node in the HC only if it appears for the first time.

The run-time complexity of Steps 1 and 4 is $O(n^2)$. The expected complexity of Step 2 was empirically estimated in section 5.2.1.3 to be

$0(n^2)$. Step 3 requires $0(n)$ operations. Therefore, the overall expected complexity of TSP4 can be regarded as being equal to $0(n^2)$. The memory requirement is also $0(n^2)$ when the cost matrix is stored in core.

5.2.2.1 Illustrative examples

We illustrate the above algorithm by showing its performance on small problems that appeared in the literature. In every example the lower half of the symmetric matrix is shown along with the best known solution to the problem. The step-by-step solution provided by TSP4 follows. Note that in every case the answer agrees with the conjectured optimal one.

Example 5.1 [Deo]

For the cost matrix in Table 5.7 the best known solution to the TSP is (1, 5, 2, 6, 4, 3, 1) , with a cost of 33 units .

2	4				
3	10	12			
4	18	8	4		
5	5	2	18	14	
6	10	6	16	6	16
	1	2	3	4	5

Table 5.7

Step 1. Get the MST as shown in Figure 5.5 .

Step 2. Get a TOPM :

Starting with the perfect matching {(1,2) , (3,5)} , we get the TOPM {(1,3) , (2,5)} .

Step 3. An Euler cycle is obtained in the Eulerian graph formed by the edges of MST $\cup$ TOPM (see Figure 5.6) . The EC is (1, 2, 5, 2, 6, 4, 3, 1) .

Step 4. Get the shortest Hamilton cycle obtainable from the EC

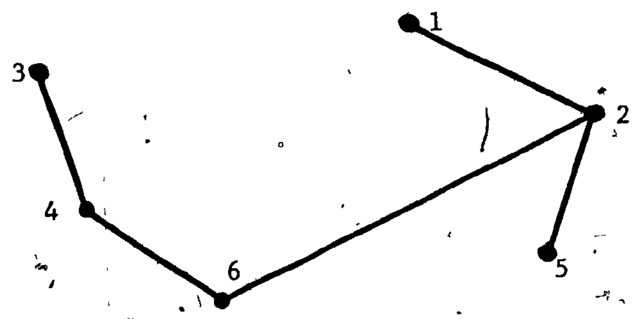


Figure 5.5

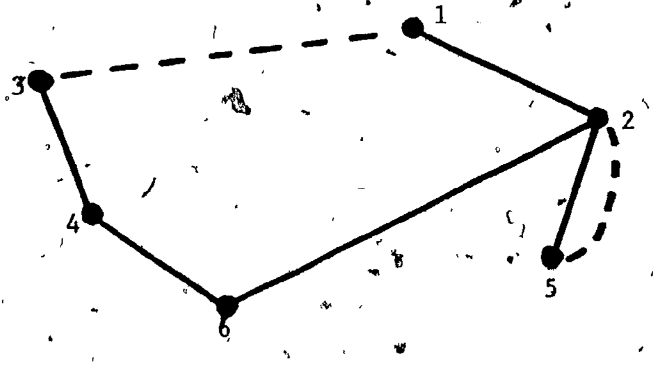


Figure 5.6

(shown by asterisk in Table 5.8) .

HC	Cost
(1,2,5,6,4,3,1)	42
*(5,2,6,4,3,1,5)	33

Table 5.8

<u>Example 5.2 [Danl]</u>

For the cost matrix in Table 5.9 the best known solution to the TSP is (1, 3, 2, 4, 5, 6, 1) , with a cost of 22 units.

2	4				
3	3	2			
4	7	5	5		
5	7	7	6	3	
6	6	7	6	5	3
	1	2	3	4	5

Table 5.9

<u>Step 1</u> Get the MST as shown in Figure 5.7 .

<u>Step 2</u> Get a TOPM :
the only choice is {(1,6)} .

<u>Step 3</u> From the graph in Figure 5.8 get
an Euler cycle:
EC = (1, 3, 2, 4, 5, 6, 1) .

<u>Step 4</u> Get the shortest HC obtainable
from the EC ;
there is only one choice: HC = (1, 3, 2, 4, 5, 6, 1) .

<u>Example 5.3 [Ray]</u>

For the cost matrix in Table 5.10 the best known solution to the TSP is (1, 6, 2, 3, 5, 4, 7, 1) , with a cost of 179 units.

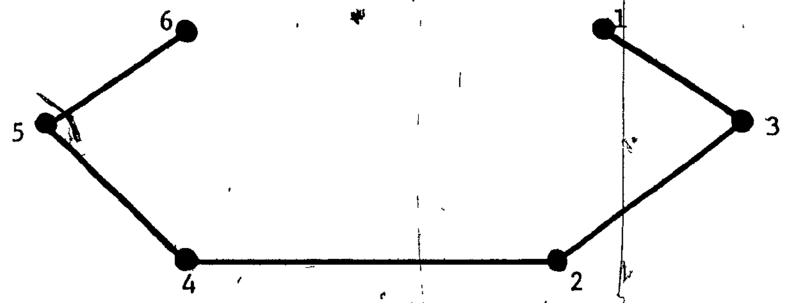


Figure 5.7

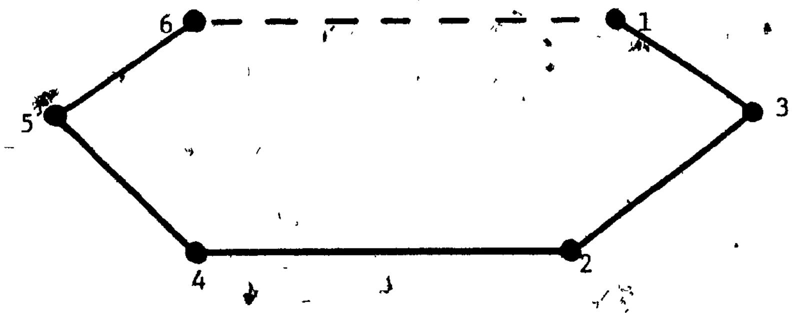


Figure 5.8

2	30					
3	40	29				
4	55	68	47			
5	28	35	22	31		
6	17	20	24	51	20	
7	33	56	46	27	25	36
	1	2	3	4	5	6

Table 5.10

<u>Step 1</u> Get the MST (see Figure 5.9).

<u>Step 2</u> Get a TOPM :

Starting with PM = {(1,2), (3,4), (5,6)}
we get the TOPM = {(1,6), (2,3), (4,5)} .

<u>Step 3</u> From the graph in Figure 5.10 get an Euler cycle:

EC = (1, 6, 2, 3, 5, 4, 7, 5, 6, 1) .

<u>Step 4</u> Get the shortest HC obtainable from the EC (shown by asterisk in Table 5.11) .

HC	Cost
*(1,6,2,3,5,4,7,1)	179
(2,3,5,4,7,6,1,2)	192
(4,7,5,6,1,2,3,4)	195

Table 5.11

5.2.2.2 <u>Experiments</u>

In order to evaluate the performance of TSP4 on larger problems various experiments are performed. First, the TSP's of Table 5.4 are tried. The results are shown in Table 5.12 .

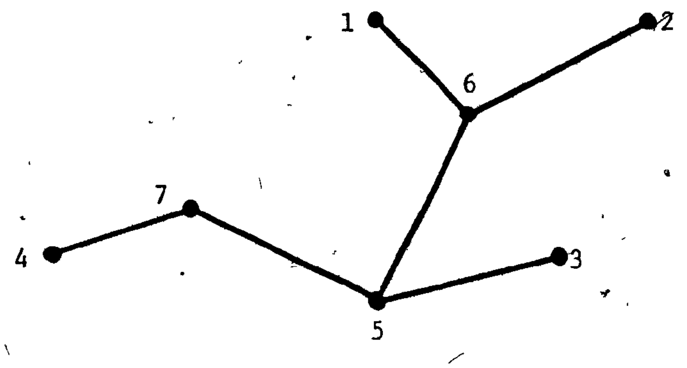


Figure 5.9

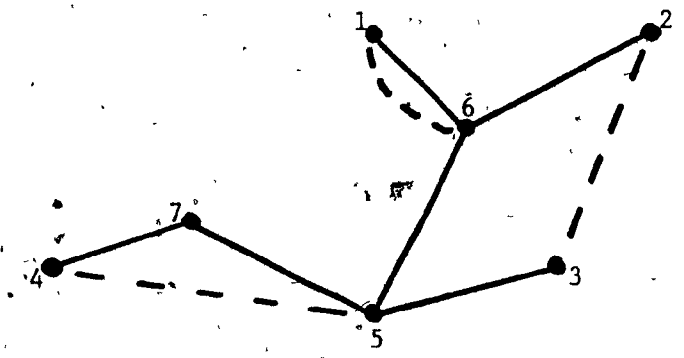


Figure 5.10

n	Source	Cost of best known solution	Cost of solution by TSP4	% deviation	Solution time in seconds
10	[Ba]	378	387	2	0.01
20	[Cr]	246	280	13	0.04
25	[Hel]	1711	1796	4	0.04
33	[Ka]	10861	12078	11	0.11
42	[Ka]	699	752	7	0.15
48	[Hel]	11461	13078	14	0.23
57	[Ka]	12955	14182	9	0.42
100	[Kro]	21282	23789	11	1.86

Table 5.12

Monte Carlo experiments are now described. Random points are generated uniformly in the unit square of the Cartesian plane and the (straight-line) Euclidean distance is used to represent the cost of going from one city to the other. The problems are thus symmetric and - incidently - the triangular inequality holds. Results are shown in Tables 5.13 to 5.15 where

TSP4	=	cost of approximate solution obtained by TSP4 ,
B_4	=	1.102 * MST ,
TSP1	=	cost of exact solution obtained by TSP1 ,
RAND	=	cost of a random solution obtained by a fast algorithm to generate random permutations [Mas] .

Every entry in Tables 5.13 - 5.15 is an average over 100 runs.

n	TSP4/B4
25	1.231
50	1.178
75	1.177
100	1.171

Table 5.13

n	TSP4/TSP1
4	1.000
5	1.010
6	1.025
7	1.018
8	1.037
9	1.027
10	1.027
11	1.030
12	1.033

Table 5.14

n	TSP4/RAND
25	0.338
50	0.238
75	0.195
100	0.171

Table 5.15

We note that

1) For $n \geq 50$ the approximate solution is less than 20% more costly than the lower bound.
2) For $n \leq 12$ the approximate solution is less than 4% more costly than the exact solution.
3) For $n \geq 50$ a random solution is more than 4 times as costly as the approximate solution.

The run-time of TSP4, proportional to n^2, was also experimentally verified as shown in Table 5.16 .

n	n^2	Ratio	Average solution time in seconds for TSP4	Ratio	$\frac{\text{Solution Time}}{n^2}$
25	625		0.085		136×10^{-6}
50	2500	4.	0.349	4.1	139×10^{-6}
75	5625	2.25	0.831	2.3	147×10^{-6}
100	10000	1.77	1.462	1.7	146×10^{-6}

Table 5.16

We observe the following

1) Solution time is equal to kn^2 where $k \approx 14 \times 10^{-5}$.
2) The data structure that achieves this speed - at the expense of storage - is the adjacency matrix form used to store the Eulerian graph. If one uses instead the edge-list structure - to save one space - an increase in computation time of 25% is observed.
3) The run-time can still be improved upon if the Eulerian graph is stored in the (more complex) form of a doubly-linked adjacency list.

5.2.2.3 Comparison with other algorithms

In this section we briefly compare TSP4 with the three other algorithms that have appeared in the literature and use similar ideas. The main differences are summarized in Table 5.17 . The algorithm of [Lin3], presently considered as the most efficient (published) heuristic - regarding the quality of the approximate answer - deserves a few more words, in connection with 2-optimality used in TSP4. We note that:

1) Getting a 2-optimal tour would require much more computation - i.e. the constant c_1 is high in c_1n^2 - than getting a TOPM which requires c_2m^2 , where m is the number of nodes with odd degree in MST (usually $m << n$) , and c_2 is a constant ($c_2 << m$ as shown in Table 5.6) .

Algorithm	Concepts used	Average solution time for a 100-city TSP	Quality of solution	Complexity
[Ros1]	MST	Less than 0.5 sec. for the IBM 370/158	$\frac{\text{NEAREST}}{\text{OPTIMAL}} < 2$. Statistically, $\frac{\text{NEARINSERT}}{B_4} \le 1.3$.	$O(n^2)$
[Lin3]	k-optimality	3-4 minutes on GE 635	Optimum obtained with above 95% confidence.	$n^{2.2}$ (empirical estimate of expected run-time)
[Chr6]	MST OPM EC	More than 100 secs. (estimated for the IBM 370/158)	Approx.Sol./Optimal Sol. ≤ 1.5 .	$O(n^3)$
TSP4	MST TOPM EC HC	Less than 1.5 secs. for the IBM 370/158.	Statistically, $\frac{\text{TSP4}}{B_4} \le 1.2$.	n^2 (empirical estimate of expected run-time)

Table 5.17

2) The algorithm for getting a 2-optimal tour starts with a random tour which may be quite far from optimal; several runs are thus needed to guarantee a high probability of getting a good tour. TSP4 starts with the MST which is a very good approximation to the final solution: in fact most of the edges of the MST are very likely to be in the optimal tour. (This point is studied in more detail in Chapter 6).

3) For large n , it is pointed out in [Lin1] that the probability for a 2-optimal tour to be the optimal solution is very low.

5.3 A new heuristic algorithm for the directed TSP

The majority of published algorithms for the TSP rely primarily on the symmetry of the cost matrix. On the other hand, algorithms for general problems - which do not assume symmetry - behave very badly on symmetric cases [Beo2, Chr4]. In this section we describe an algorithm especially designed for the directed (i.e. non-symmetric) TSP. The algorithm is intimately related to TSP4 and uses the same general concepts.

Before stating the algorithm and reporting on computational experience we give one definition. A minimal directed spanning graph (MDSG) is a subgraph of the complete directed and weighted graph on n nodes which has minimum weight and whose underlying graph is connected and acyclic. This is equivalent to saying that the underlying undirected graph is a MST in which edge (a, b) is such that $c_{ab} = \min(c_{ab}, c_{ba})$ where (a, b) and (b, a) are edges of the complete graph.

It should be noted that the MDSG differs from the MDST in that it

1) is unrooted, and
2) has no restrictions imposed on the in or out-degree of its nodes.

5.3.1 Algorithm TSP5

Given a complete digraph with n nodes and its (non-symmetric) cost matrix, the following algorithm gets a nearly optimal solution to the TSP defined on that digraph.

Step 1. Get the MDSG.

Step 2. Add a 2-optimal set of arcs to the MDSG in order to make the digraph thus obtained Eulerian.

Step 3. Find an Euler circuit in the digraph.

Step 4. Among all Hamilton circuits obtainable from the EC choose the one with minimum cost. Stop.

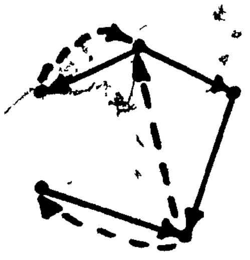
Comments:

Step 1. If we replace every c_{ij} by $\min(c_{ij}, c_{ji})$ in the cost matrix, and apply an MST algorithm on the new matrix we obviously get an MST with the same weight as the

MDSG. It is important to store along with every entry in the cost matrix the direction of the edge having that cost, i.e. $i \rightarrow j$ or $j \rightarrow i$.

Step 2. By adding to the MDSG arcs leaving nodes with out-degree deficiency and entering nodes with in-degree deficiency we get a balanced graph which is also connected and hence Eulerian. This set of arcs can easily be made 2-optimal while keeping the arc directions.

Step 3. Here we note the importance of keeping the arc directions as described in Steps 1 and 2 above. This is clear from the example in Figure 5.11 .


MDSG + Matching arcs

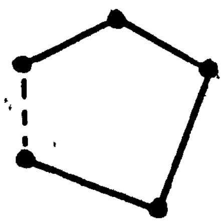

MST + Matching arc

Figure 5.11

The Euler circuits resulting from the two graphs in Figure 5.11 will be quite different!

Step 4. This step is straightforward (see the discussion on Step 4 of TSP4 in section 5.2.2).

Space and time requirements of TSP5 are similar to those of TSP4 as it can be easily observed.

5.3.2 Illustrative examples

As we did for TSP4, we illustrate the operation of TSP5 by trying it on small problems published in the TSP literature. The (non-symmetric) cost-matrix is first given, along with the best-known tour. The step-by-step solution of TSP5 follows.

Example 5.4 [Lit]

For the cost-matrix in Table 5.18 the best known (directed) TSP tour is (1, 4, 3, 5, 6, 2, 1) , with cost 63 units.

1	∞	27	43	16	30	26
2	7	∞	16	1	30	25
3	20	13	∞	35	5	0
4	21	16	25	∞	18	18
5	12	46	27	48	∞	5
6	23	5	5	9	5	∞
	1	2	3	4	5	6

Table 5.18

Step 1. Get the MDSG as shown in Figure 5.12 .

Step 2. Get a 2-optimal set of additional arcs.

Nodes with out-degree deficiency = {1,4,6}

Nodes with in-degree deficiency = {2,3,5}

We start with {(1,2) , (4,3) , (6,5)} and improve by exchanges to get the 2-optimal set {(1,2) , (4,5) , (6,3)} .

Step 3. Get an Euler circuit in the Eulerian graph of Figure 5.13 ;

EC = (1,2,4,5,6,3,6,2,1)

Step 4. Get the shortest Hamilton circuit obtainable from the EC (marked with asterisk in Table 5.19).

HC	Cost
(1,2,4,5,6,3,1)	76
*(4,5,6,3,2,1,4)	64
(3,6,2,1,4,5,3)	73

Table 5.19

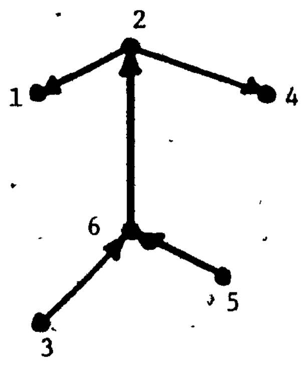


Figure 5.12

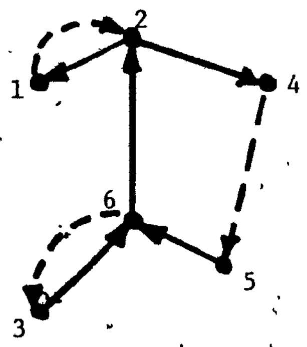


Figure 5.13

Example 5.5 [Beo2]

For the cost matrix of Table 5.20 the best known (directed) TSP tour is (1,2,4,5,6,3,1), with cost 94 units

$$\begin{array}{c|cccccc}
1 & \infty & 7 & 65 & 68 & 34 & 81 \\
2 & 19 & \infty & 22 & 27 & 59 & 29 \\
3 & 14 & 43 & \infty & 62 & 77 & 65 \\
4 & 76 & 53 & 64 & \infty & 6 & 51 \\
5 & 39 & 58 & 38 & 27 & \infty & 13 \\
6 & 46 & 67 & 27 & 11 & 38 & \infty \\
\hline
 & 1 & 2 & 3 & 4 & 5 & 6
\end{array}$$

Table 5.20

Step 1 Get the MDSG (see Figure 5.14) .

Step 2 Get a 2-optimal set of additional arcs.
Starting with {(5,3) , (4,6)} we get the 2-optimal set {(5,6) , (4,3)} .

Step 3 Get an Euler circuit from the graph in Figure 5.15 ;
EC = (1,2,4,5,6,4,3,1)

Step 4 Get the shortest HC obtainable from the EC (marked by asterisk in Table 5.21) .

HC	Cost
*(1,2,4,5,6,3,1)	94
(5,6,4,3,1,2,5)	168

Table 5.21

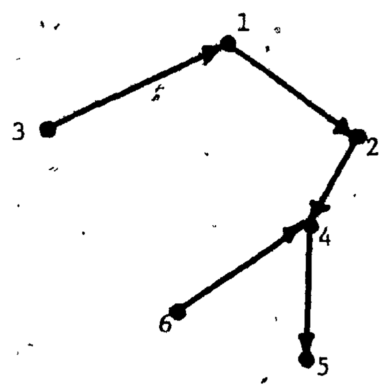


Figure 5.14

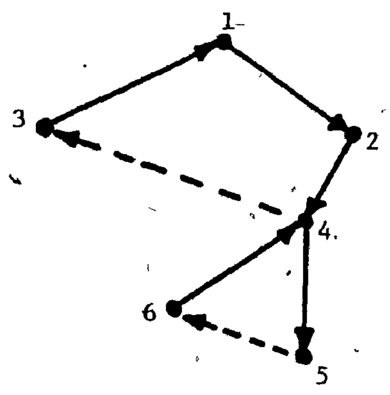


Figure 5.15

Example 5.6 [Lal]

For the cost matrix of Table 5.22 the best known (directed) TSP tour is (1,3,9,4,8,5,10,6,7,2,1) , with cost 146 units.

1	∞	24	18	22	31	19	33	25	30	26
2	15	∞	19	27	26	32	25	31	28	18
3	22	23	∞	23	16	29	27	18	16	27
4	24	31	18	∞	19	13	28	9	19	27
5	23	18	34	20	∞	31	24	15	25	8
6	24	12	17	15	10	∞	11	16	21	31
7	28	15	27	35	19	18	∞	21	21	19
8	13	24	18	13	13	22	25	∞	29	24
9	17	21	18	24	27	24	34	31	∞	18
10	18	19	29	16	23	17	18	31	23	∞
	1	2	3	4	5	6	7	8	9	10

Table 5.22

Step 1 Get the MDSG (see Figure 5.16) .

Step 2 Get a 2-optimal additional set of arcs.
This is given by:
{(1,6) , (2,3) , (5,8) , (5,4) , (7,6) , (9,3) , (10,6)} .

Step 3 Get an Euler circuit in the graph of
Figure 5.17 ; EC =
(1,6,2,3,9,3,5,4,8,5,10,6,7,6,5,8,1)

Step 4 Get from the EC the shortest possible HC.
This is: (1,2,3,9,4,8,5,10,6,7,1) ;
with cost 169 .

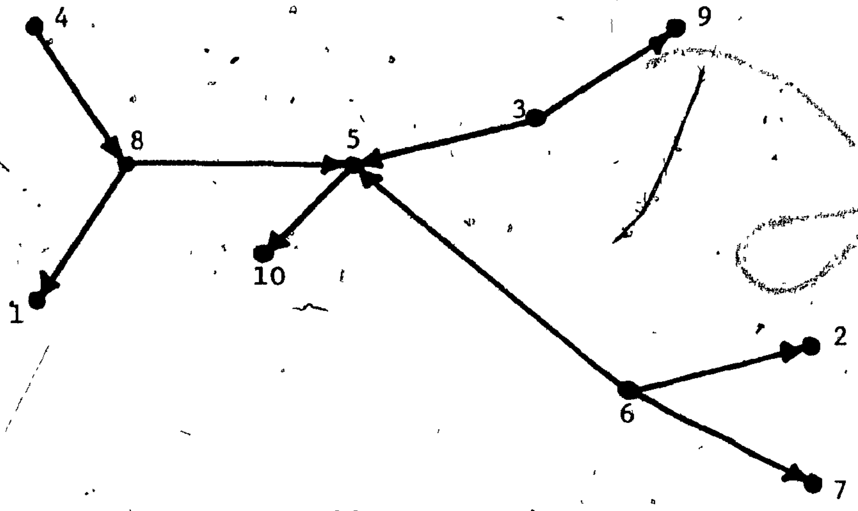


Figure 5.16

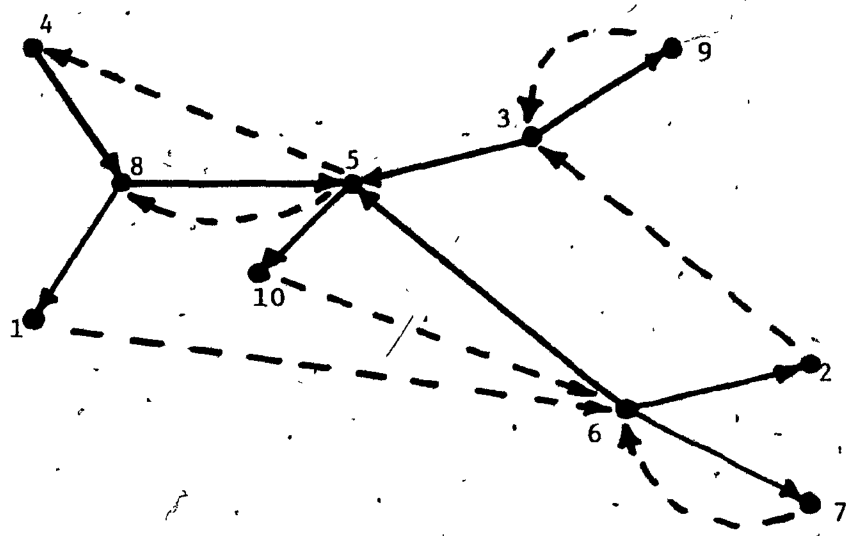


Figure 5.17

5.3.3 Experiments

Very few asymmetric cost-matrices for the TSP appear in the literature, and even these are for trivial values of n [Ac2, Beo2, Con, Lal, Lit, Sas, Wa]. In contrast with the symmetric case it is quite complicated to derive an estimate for the solutions, or put some bounds on it, for general directed TSP's. A variety of lower bounds can be obtained by solving the corresponding assignment problem [Beo2, Chr4]. These usually involve non-trivial computations. Therefore we consider only asymmetric cost-matrices where the triangular inequality holds (every city visited once) and take the MDSG as our lower bound. We describe two ways of getting a random asymmetric cost matrix where the triangular inequality holds.

1) Every entry in the cost-matrix is generated randomly using the appropriate distribution; then a large number K (larger than any cost) is added to every entry.

   It is obvious that,

   if $a + b < c$ then $(a+K) + (b+K) > (c+K)$.

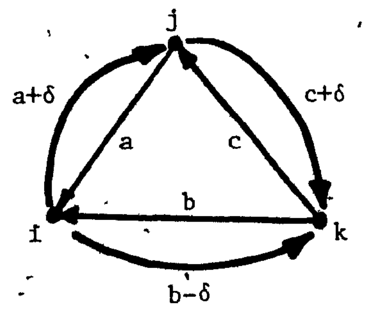
2) x and y coordinates of n points are generated randomly. Then, entry c_{ij} of the cost-matrix is taken as the Euclidean distance between points i and j . Entry c_{ji} is now obtained from

$$c_{ji} = c_{ij} \pm \delta$$

   where δ is an arbitrary small number. This method is illustrated in Figure 5.18 ; note that if in the cost matrix of Figure 5.18(a),

	i	j	k
i	∞	a+δ	b−δ
j	a	∞	c+δ
k	b	c	∞

(a) Cost - matrix



(b) Complete directed graph

Figure 5.18

a+b > c , a+c > b and b+c > a hold, then the six inequalities in the digraph of Figure 5.18(b) also hold.

Table 5.23 summarizes the results of Monte Carlo experiments where:

TSP5 = cost of approximate solution provided by TSP5, and

MDSG = cost of the MDSG.

Every entry is an average of 100 problems with asymmetric matrices and the triangular inequality holding.

n	TSP5/MDSG	Run-time in seconds
25	1.50	0.1
50	1.48	0.7
75	1.45	1.7
100	1.45	3.2

Table 5.23

We notice that the computation time is - on the average - equal to $(32 \times 10^{-5} \times n^2)$ seconds approximately.

Finally we point out that the quality of the answer can be highly improved - at the cost of an increase in the run-time - by getting an optimal bipartite matching in Step 2, using the $O(n^3)$ algorithm for the assignment problem [Bou].

5.4 Conclusion

The main results of this chapter can be summarized as follows:

1) Various - apparently unrelated - methods were combined to yield efficient approximation algorithms for the TSP.

2) Near-optimal solutions to the minimum-weight perfect matching problem have been obtained by applying the concept of k-optimality.

3) The minimal directed spanning graph - a generalization of the minimal spanning tree for directed graphs - was used to address the asymmetric TSP.

Chapter Six
On Local Neighborhood Search and the TSP

A very efficient technique for obtaining approximate solutions to combinatorial optimization problems is known as local neighborhood search (LNS). LNS is defined as follows [Cof]:

"Let S be the set of feasible solutions to a combinatorial optimization problem, then

1) A neighborhood N is defined for the problem if for every $s \in S$ we define

$$N(s) \subset S \text{ as the neighborhood of } s \, .$$

2) An initial solution $s_1 \in S$ is chosen by a technique T .
3) If $C(s_i)$ is the cost of solution s_i , then a policy P is used to search N(s) and choose $s_{i+1} \in N(s_i)$ such that

$$C(s_{i+1}) < C(s_i) \, ,$$

whenever this is possible.

If such an improvement does not exist we say that s_i is locally optimal with respect to N ."

Thus LNS algorithms start with an initial solution and attempt to improve upon it by searching its neighborhood, following some specific rule that yields a better solution. Improvements are adopted as soon as they are found and the above procedure is applied to the new solution thus created. When no further improvement can be made the solution on hand is called a local optimum and represents an approximate solution to the problem. The process may be repeated several times each time starting from a different initial solution. The best of the various local optima is then selected as the final solution.

Algorithms of this type have proved excellent at obtaining near-optimal solutions. A very good example of a successful application of this technique is the principle of k-optimality (see Chapter 5) and its variants described in [Ad, Ba, Chr3, Cr, Lin1, Lin2, Lin3, Lin4, Rei, Ste1, Ste2] for the TSP. Various analyses of LNS algorithms can be found in [Cof, Pap, Sav1, Sav2].

As in [Cof], let us denote this category of algorithms by LNS (T,P,N) where

1) T is the technique by which an initial solution is chosen;
2) P is the policy by which the neighborhood is searched for improvements; and
3) N is the neighborhood searched.

Obviously T, P and N are, in general, problem-dependent. However, the initial solution provided by T falls naturally under one of the classes

1 - Random solutions
2 - Constructed solutions
3 - A combination of (1) and (2).

Now, although P is independent of T, the local optima arrived at in N will heavily rely on the initial starting points.

A very important issue in designing an LNS (T,P,N) algorithm is hence to decide on whether it is preferable to commence with a biased solution or a purely random one. Again the final decision will inevitably be problem-dependent. It is generally believed that for the TSP purely random starting tours work best [Cof]. Given a neighborhood N and a search policy P , for the TSP, the theme of this chapter is an attempt to answer the above question about T. We essentially show that our experience with a new heuristic algorithm that uses LNS contradicts the above assertion about random starting tours. In section 6.1 we describe how biased starting solutions are constructed and used to obtain local optima. Solutions of 'classical' TSP's, obtained by both random and biased starts are compared in section 6.2 . Our conclusions, supported by extensive Monte Carlo runs, are presented in section 6.3 .

6.1 An LNS algorithm for the TSP using biased starts

Algorithm TSP6

<u>Step 1</u>. Get the MST.

<u>Step 2</u>. Get a TOPM of the odd-degree nodes.

<u>Step 3</u>. Get an EC in the graph composed of the edges of the MST and those of the TOPM.

<u>Step 4</u>. Among all HC's obtainable from the EC choose the one with minimum cost.

<u>Step 5</u>. Starting with that cycle get a 2-optimal solution.

We observe that

1) LNS exists in two places in the above algorithm: in Step 2 where a TOPM is obtained and in Step 5 where a 2-optimal HC is used as the final approximate solution to the TSP. As far as the TOPM is concerned we always start with a random perfect matching and make it 2-optimal by the usual procedure. This is due to the fact that we do not know of any efficient way of biasing profitably the starting solution when the distribution of the weights is not known [Av2]. Hence, Step 2 will not be discussed further. In obtaining the 2-optimal HC, however, we start with the best circuit obtainable from the EC (Step 4): this is hence a "biased" starting solution.

2) Steps 1 to 4 are precisely algorithm TSP4. One could therefore phrase the above algorithm simply as follows:
"Get a 2-optimal tour starting with the HC provided by algorithm TSP4".

We propose to compare the approximate solution thus obtained, to a local optimum arrived at starting with a purely random tour.

6.2 <u>Testing TSP6 on 'classical' problems</u>

Table 6.1 shows:

1) the results obtained when applying the new algorithm to the set of famous TSP's of Table 5.4, the cost of the final tour in each case being denoted by TSP6; and

2) the corresponding figures obtained when starting with a random solution and making it 2-optimal, the costs of the final tours being denoted by RAND1 .

n	Source	Cost of best known solution	TSP6	Run-time in secs.	RAND1	Run-time in secs.
10	[Ba]	378	381	0.01	378	0.01
20	[Cr]	246	271	0.05	292	0.12
25	[Hel]	1711	1711	0.05	1711	0.04
33	[Ka]	10861	10861	0.13	11317	0.06
42	[Ka]	699	724	0.29	788	0.12
48	[Hel]	11461	12585	0.35	11828	0.20
57	[Ka]	12955	13747	0.65	14221	0.31
100	[Kro]	21282	22994	2.16	23626	1.73

Table 6.1

Here we should note that TSP6 as it stands leads to *one* solution (in Step 4) which is made 2-optimal (in Step 5); we therefore used *one* starting random solution to obtain the results exhibited in Table 6.1 . However, in order to apply the concept of LNS in its full power, one should start with several -either biased or random - feasible solutions and choose the best local optimum that these lead to. For the purpose of comparison we achieve this by modifying our algorithm as follows:

Algorithm TSP6.1

Steps 1 - 3 : (as before)

Step 4' : Obtain all different HC's obtainable from the EC.

Step 5' : Make every one of these cycles 2-optimal and choose the cheapest resulting cycle.

Before proceeding, we show that an empirical estimate of the expected run-time of TSP6.1 is $O(n^3)$. First note that the MST has n-1 edges. Also, the number of odd degree nodes in the MST being at most n , the maximum number of edges the TOPM can possibly have is n/2 . It follows that the Eulerian graph, resulting from (MST $\cup$ TOPM), has at most $\frac{3n}{2} - 1$ edges. Now, if every node on the EC leads to a different HC, then the maximum possible number of HC's is $\frac{3n}{2} - 1$. It was experimentally observed that each of the $\frac{3n}{2} - 1$ HC's requires on the average $O(n^2)$ steps to be made 2-optimal. It follows that the overall run-time of algorithm TSP6.1 is proportional to n^3 .

Table 6.2 exhibits the results obtained by:

1) Using TSP6.1 to obtain an approximate solution, the cost of which we denote by TSP6.1 .
2) Starting with 2n random solutions, making each one of them 2-optimal and choosing the best solution thus obtained, the cost of which we denote by RANDN .

Note that:

1) The observed run-time for TSP6.1 is approximately $10^{-4} n^3$ seconds, on the average.
2) The number (2n) was chosen since it is considerably larger than the possible number of different HC's obtainable from the EC.

n	Source	Best known solution	TSP6.1	Run-time in secs.	RANDN	Run-time in secs.
10	[Ba]	378	378	0.10	378	0.12
20	[Cr]	246	246	0.80	259	2.25
25	[Hel]	1711	1711	1.43	1711	2.84
33	[Ka]	10861	10861	3.70	10861	7.36
42	[Ka]	699	699	7.80	713	22.05
48	[Hel]	11461	11574	11.30	11769	22.90
57	[Ka]	12955	13300	19.50	13169	51.79
100	[Kro]	21282	22112	110.20	21593	190.62

Table 6.2

6.3 Discussion

Looking at the figures in Tables 6.1 and 6.2, it appears at first glance that in some cases random starts lead to very good results: sometimes better than the biased starts, sometimes even optimal. This observation, however, is misleading. To explain why one should not overestimate the relatively good results obtained by starting with random initial solutions we give the following two reasons:

1) k-optimality can be tested in O(m) comparisons where

$m = \binom{n}{k} (k-1)!.\, 2^{k-1}$ as shown in [Chr3]. This is the complexity of each iteration. We noticed that a large number of starting solutions is needed in general to obtain a good final solution among all local optima. It follows that if this number is of O(n) then the - theoretical - complexity of the algorithm for 2-optimal solutions jumps from n^2 to n^3.

2) In most cases a random solution is further from a local optimum than a biased one; and one particular random solution *may* require several $O(n^2)$ iterations before it becomes 2-optimal. This point is very important for a correct interpretation of the results: in our experiments, when it was noticed that a random solution will require a prohibitive computation time the program was stopped; this happened quite often. Only 2-optimal tours obtained from random ones in less than one minute of CPU time are reported upon in this chapter.

The two points made above lead us to ask the following question: how many cycles would have at worst to be considered by a LNS algorithm - looking for a 2-optimal solution starting from a random one - before the answer is obtained?

We know that all permutations of n objects can be obtained by pair-wise exchanges [Sd]. A tour being a permutation of the cities, it is therefore easy to deduce that one can obtain all (n-1)!/2 different tours of a traveling salesman by starting with a random tour and then exchanging two edges at a time as illustrated in Figure 6.1 for n = 5 . Each of the 12 tours, except the first, differs from the preceeding one by exactly two edges.

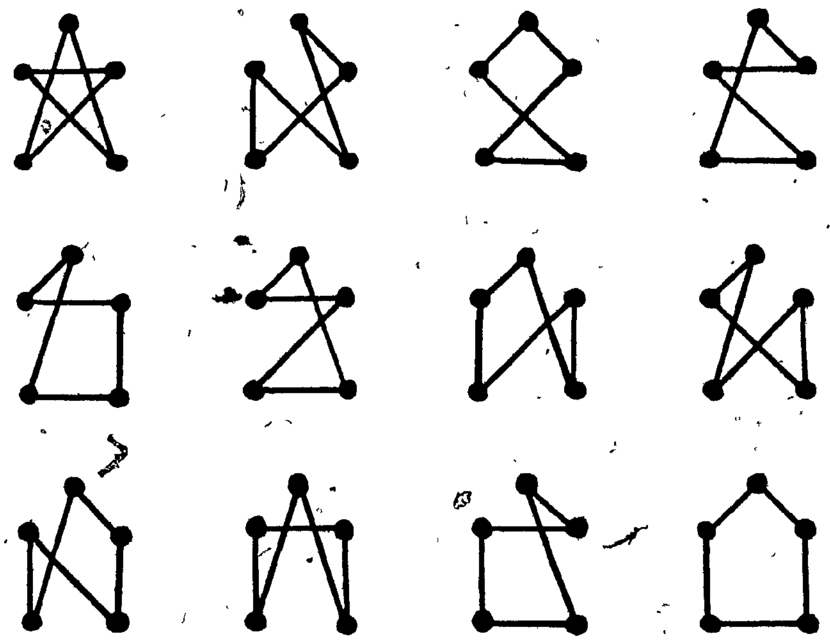


Figure 6.1

Figure 6.2 [Avl] shows a complete weighted graph on 5 nodes for which an LNS algorithm, starting with the random tour (135241), has to consider 7 other tours before the 2-optimal tour (123451) is found.

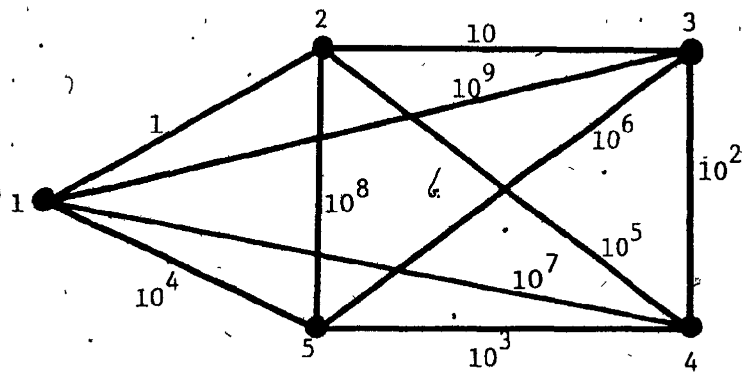


Figure 6.2

The sequence examined is: (135241) , (132541) , (125431) , (125341) , (143251) , (142351) , (124351) , (123451) .

For n = 6 , an LNS algorithm that works on the graph of Figure 6.3 looking for a 2-optimal tour, may have to consider 16 tours if it starts with the random tour (1325641) to finally terminate with (1263451) as shown in Table 6.3 . This is an increase by a factor of 2. Noting that the complete graph of Figure 6.3 is obtained from the one in Figure 6.2 by adding one node, we conjecture an exponential growth of the computation time, as n increases, for the worst case behavior of this LNS algorithm.

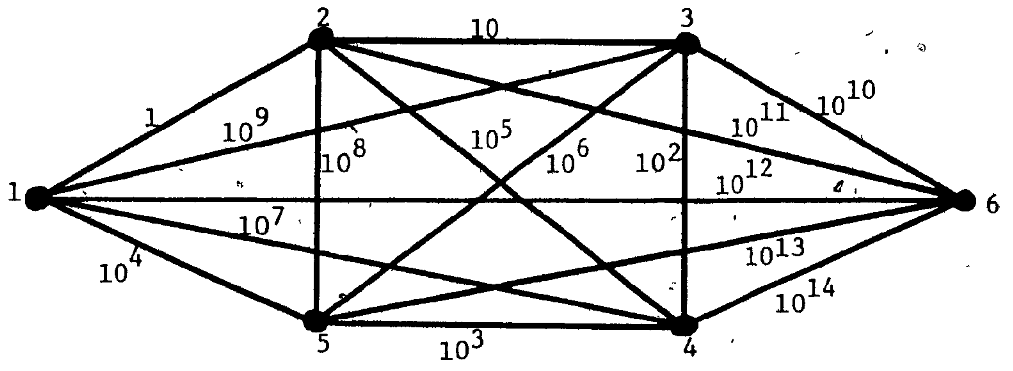


Figure 6.3

Tour	Order in which examined
1325641	1
1325461	2
1352461	3
1246531	4
1324651	5
1326451	6
1462351	7
1453261	8
1452361	9
1254361	10
1245361	11
1542361	12
1362451	13
1426351	14
1263541	15
1263451	16

Table 6.3

6.4 Conclusion

We can conclude from our experience that: in order to get a good balance - on the long run - between quality of solution and amount of computation it is preferable, in solving the TSP by LNS, to use biased starting solutions rather than random ones. Table 6.4 confirms this conclusion. The results shown are averaged over the same 100 problems obtained by generating uniformly n random points in the unit square.

n	TSP6/1.102*MST	Run-time	RAND1/1.102*MST	Run-time
25	1.158	0.12	1.161	0.67
50	1.130	0.50	1.170	3.18
75	1.119	1.12	1.179	9.34
100	1.114	2.00	1.186	19.41

Comparison between quality of solutions TSP6 and RAND1 and corresponding run-time in seconds.

Table 6.4

Chapter Seven
On Maximal Triangulations, Optimal Drawings and Hamilton Cycles

This chapter and the following one investigate a few problems related to the ETSP, and may be considered as a sequel to Chapters 2 and 3. In Chapter 3 we used triangulations to obtain an approximate solution to the ETSP. Other applications in which triangulations arise are the finite element method [B, Br2] and the numerical interpolation of functions of two variables [Dav]. Very little is known, however, about the geometrical properties of triangulations and their relation to other structures.

In this chapter we study a special type of triangulations. In particular, we give a method of placing n points in the plane and joining them by straight-line segments that yields a triangulation with the maximum possible number of edges. Triangulations of this type are shown to be Hamiltonian and an expression for the number of Hamilton cycles they contain is derived. We also conjecture a relation between these triangulations and crossing-number-optimal rectilinear drawings of complete graphs [Er]. Finally, a lower bound is presented for the maximum number of crossing-free Hamilton cycles in a rectilinear drawing of a complete graph. This bound is an improvement over the one that appears in [New].

7.1 Maximal Triangulations

It is known that the maximum number of edges a plane graph can possibly have (i.e. no edge could be added without creating a crossing) is 3n-6 : in that case the plane graph is said to be maximal [Harl]. Also, the 3n-6 upper bound is achieved by a plane graph if every one of its faces is a triangle [Harl]. It follows that a triangulation is maximal only if its convex hull is a triangle.

Theorem 7.1

A triangulation of n points placed in the plane as $\lfloor n/3 \rfloor$ concentric triangles is maximal.

Proof

The theorem is a consequence of the last sentence in the previous paragraph. A counting argument is given below that illustrates the construction. When the n points are placed in $\lfloor n/3 \rfloor$ concentric triangles there will be 0 , 1 or 2 interior points according to whether $n \equiv 0$, 1 or 2 (mod 3) respectively. We examine every case separately.

a) $n \equiv 0 \pmod 3$

The triangulation will contain

n edges = contributed by the n/3 concentric triangles $(= \frac{n}{3} * 3)$

+2n−6 edges = obtained by connecting every triangle − except the innermost − to the next interior one by 6 non-crossing edges $(= \frac{n-3}{3} * 6)$

b) $n \equiv 1 \pmod 3$

The triangulation will contain

n−1 edges = as before $(= \frac{n-1}{3} * 3)$

+2n−8 edges = as before $(= \frac{n-4}{3} * 6)$

+ 3 edges = connecting the innermost triangle to the interior point.

c) $n \equiv 2 \pmod 3$

The triangulation will contain

n−2 edges = as before $(= \frac{n-2}{3} * 3)$

+2n−10 edges = as before $(= \frac{n-5}{3} * 6)$

+ 5 edges = connecting the innermost triangle to the two interior points.

+ 1 edge = connecting the two interior points.

Hence a total of 3n−6 edges in every case.

Q.E.D.

Let us denote by T_n , the maximal triangulation of n points obtained by the construction of Theorem 7.1 . Figures 7.1, 7.2 and 7.3 show the various T_n's for $3 \leq n \leq 9$.

It should be noted that for n ≥ 7 , the construction described in Theorem 7.1 is not the only way of obtaining a maximal triangulation. Figure 7.4 shows three maximal triangulations of 8 points.

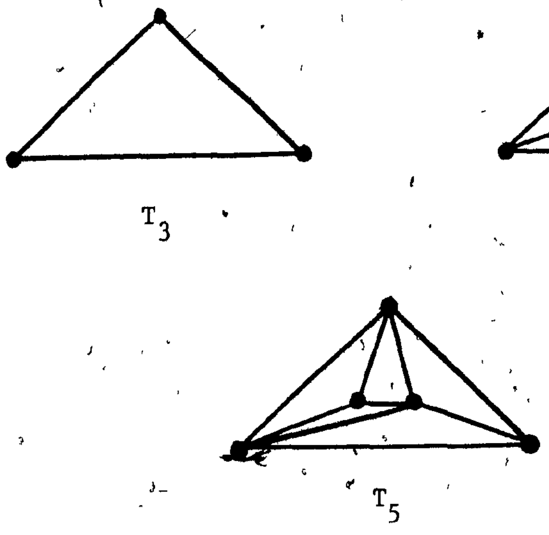


Figure 7.1

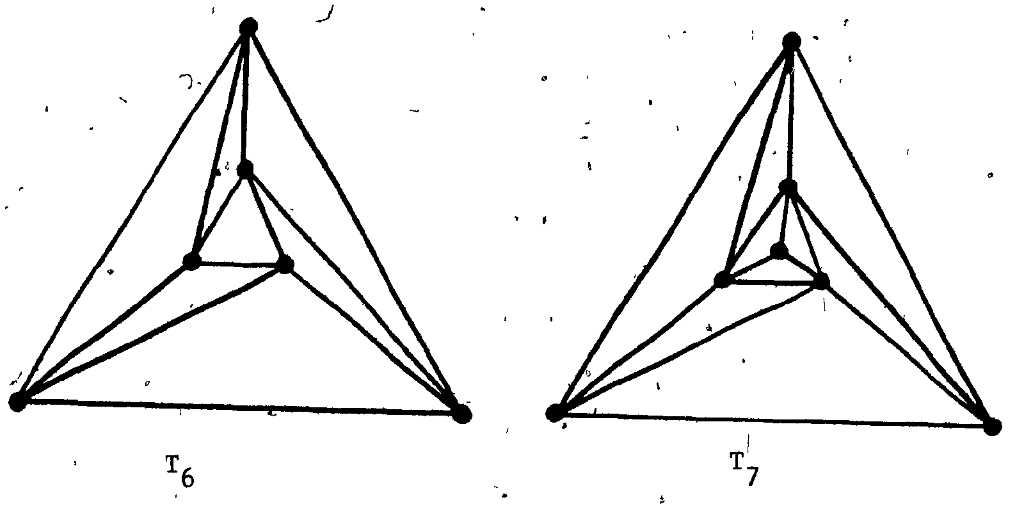


Figure 7.2 (cont'd.)

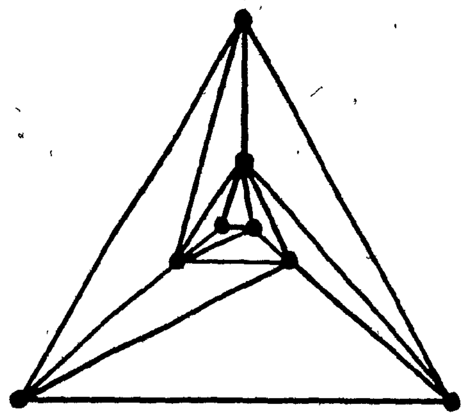

T_8

Figure 7.2

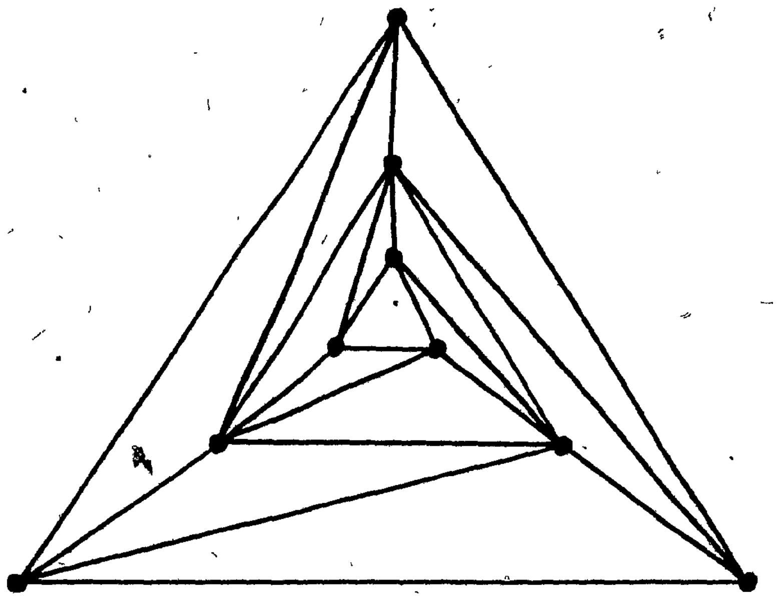
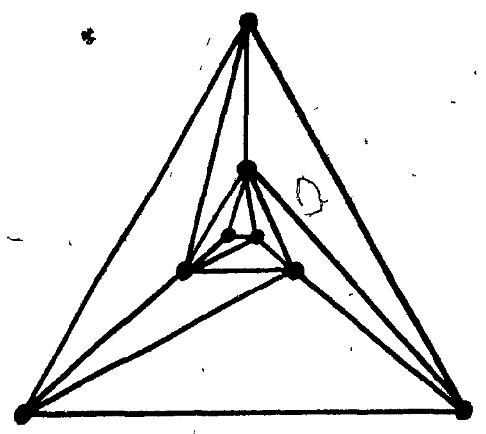
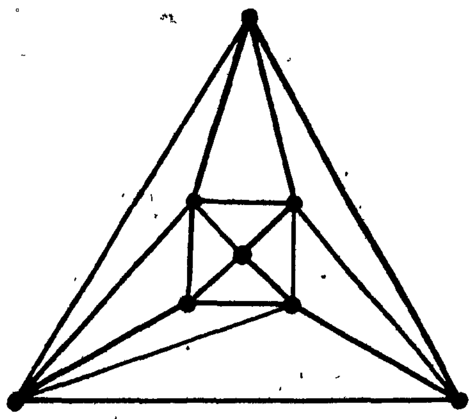

T_9

Figure 7.3





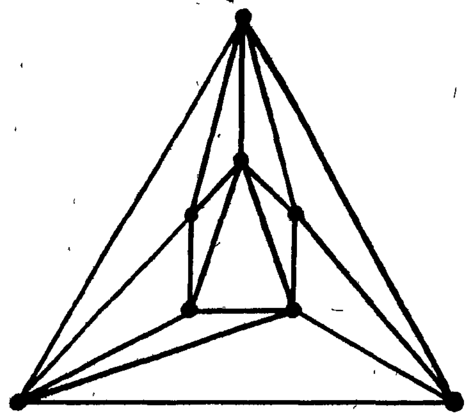


Figure 7.4

7.2 Maximal triangulations and Optimal Drawings

A crossing-number-optimal rectilinear drawing (CNORD) of K_n, the complete graph with n nodes, is a rectilinear drawing of K_n that has the minimum possible number of crossings [Har2]. In spite of the considerable attention optimal drawings have recently been given [Eg, Er, Gu1, Gu2, Har3, Je], CNORD's of K_n are only known for $n \leq 9$. Figure 7.5 shows the CNORD of K_6.

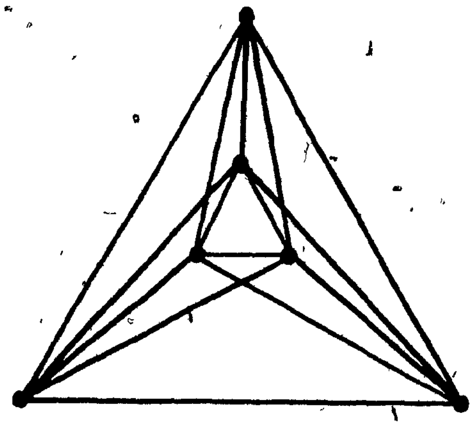


Figure 7.5

Theorem 7.2

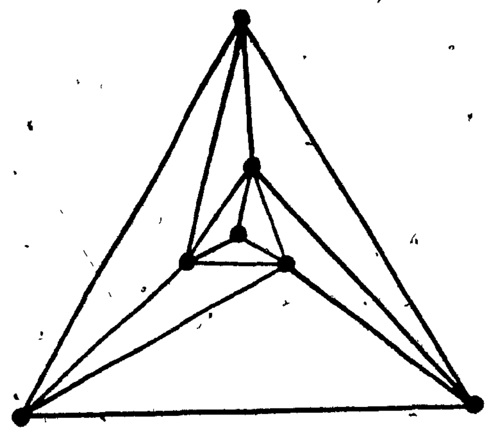
For $3 \leq n \leq 9$ a CNORD of K_n can be obtained from a T_n.

Proof:

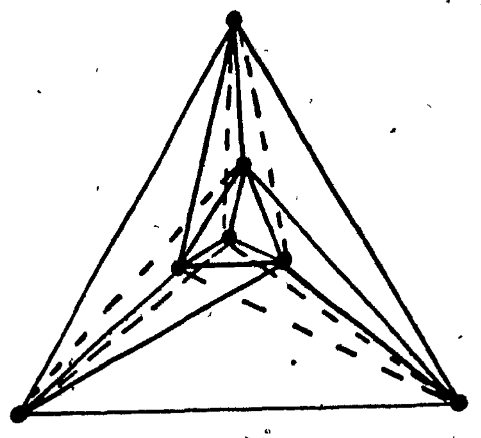
A drawing of K_n can be obtained by adding the missing edges, if any, to a T_n. For $3 \leq n \leq 9$, comparison of the resulting drawings with the known CNORD's of K_n [Har3, Je], shows them to be isomorphic [Gu1].

Q.E.D.

Theorem 7.2 is illustrated in Figure 7.6 for n=7.



T_7



CNORD of K_7 with 9 crossings

Figure 7.6

We believe that Theorem 7.2 could be generalized for all values of n . Loosely speaking, we conjecture that the more the edges in a triangulation of n points, the fewer the crossings in a rectilinear drawing and vice versa.

7.3 Maximal Triangulations and Hamilton Cycles

A graph is called Hamiltonian if it contains a Hamilton cycle [Berg]. We know from [Mn] that not every triangulation is Hamiltonian. Figure 7.7 shows a triangulation that does not contain a Hamilton cycle [Sku].

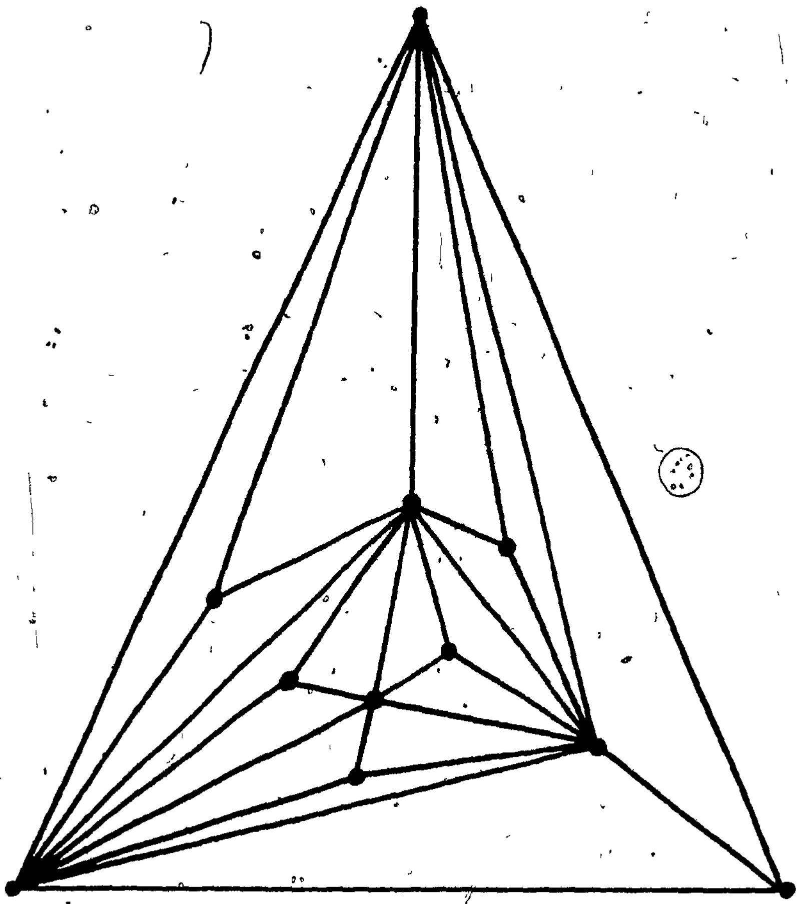


Figure 7.7

On the other hand, is was shown in [Wh] that a triangulation which

a) is maximal

and b) contains no cycle of length 3 apart from the triangles bounding its faces,

is guaranteed to possess a Hamilton cycle. An example is shown in Figure 7.8.

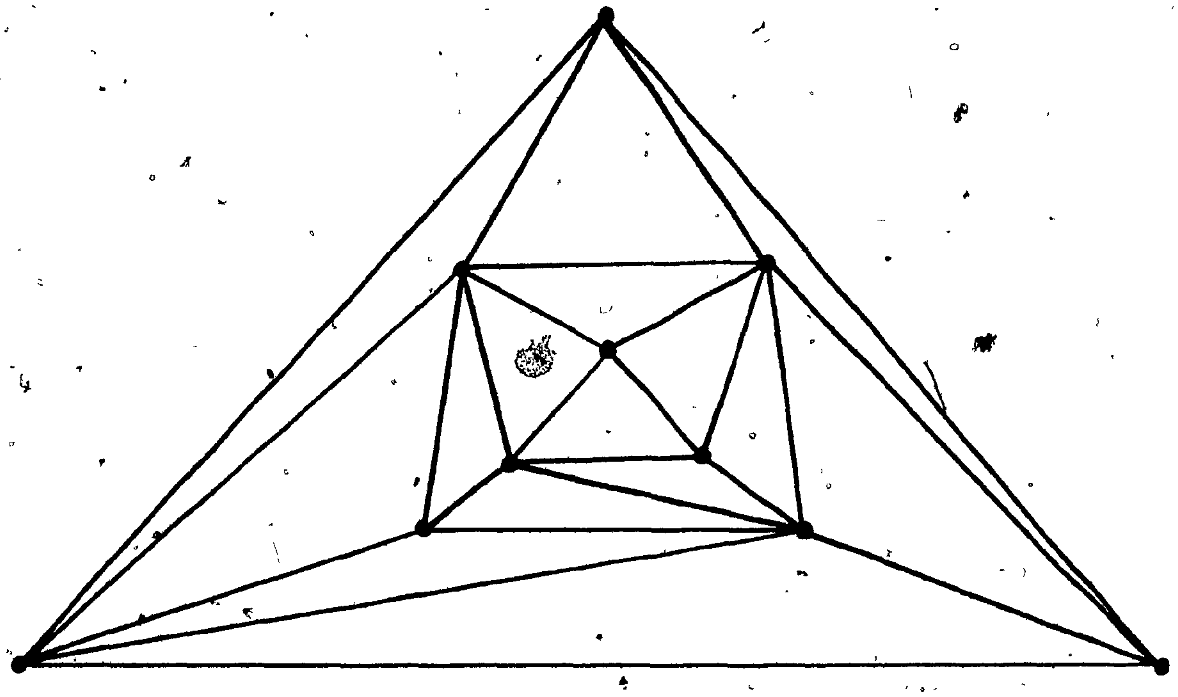


Figure 7.8

Theorem 7.3 [illegible]

Every T_n is Hamiltonian.

Proof:

By induction.

1) Since the innermost triangle of any T_n has either 0 , 1 or 2 interior points we start by showing the theorem true for $n=3$, 4 and 5 .

In Figures 7.9, 7.10 and 7.11 the Hamilton cycle is indicated by directed arcs.

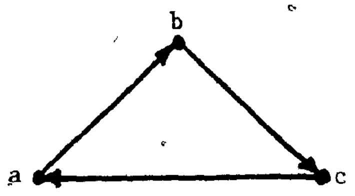


Figure 7.9

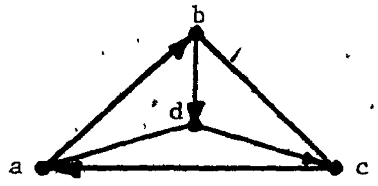


Figure 7.10

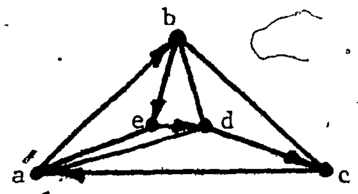
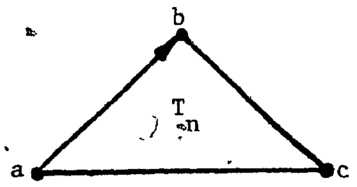


Figure 7.11

2) We now assume that for $n \geq 3$ every T_n contains a Hamilton cycle. Let the outermost triangle of a T_n be given by the vertices a, b and c and assume edge (a,b) is part of the Hamilton cycle, as shown in Figure 7.12.



Symbolic representation of a T_n

Figure 7.12

3) We complete the proof by showing that the theorem is true for $n + 3$.

Following the construction in Theorem 7.1,

(a) three vertices, say p, q and r, are added to those of T_n

and (b) edges (p,q), (q,r), (r,p), (p,a), (p,c), (q,a), (q,b), (r,b) and (r,c) are drawn, yielding T_{n+3} as shown in Figure 7.13.

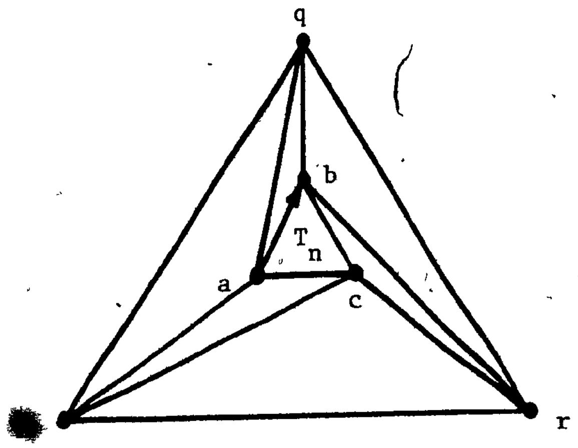


Figure 7.13

In order to obtain a Hamilton cycle in T_{n+3} we replace edge (a,b) in the Hamilton cycle of T_n by the path (a,p,r,q,b).

Q.E.D.

We conclude this section by counting the number of different Hamilton cycles in a T_n. For T_4, T_5 and T_6 this number is 3, 6 and 16 respectively as shown in Figures 7.14, 7.15 and 7.16.

Let t_n^i denote the number of Hamilton cycles in a T_n having i edges on the convex hull of T_n as part of each cycle. Figures 7.14, 7.15 and 7.16 show that the values of t_n^i for $n=4$, 5 and 6, and $i=1$ or 2 are as follows:

$$t_4^1 = 0 \qquad t_4^2 = 3$$
$$t_5^1 = 2 \qquad t_5^2 = 4$$
$$t_6^1 = 6 \qquad t_6^2 = 9$$

Excluding the two special cases $t_3^3 = 1$ and $t_6^0 = 1$, it is clear that $t_n^3 = t_n^0 = 0$, for all n , so that the only types of Hamilton cycles possible are for i=2 or 1 as shown in Figure 7.17 . Hence, given a T_n , with n > 3 , and the initial conditions in (7.1), the number of Hamilton cycles in T_{n+3} will be

$$t_{n+3} = t_{n+3}^1 + t_{n+3}^2$$

where

$$t_{n+3}^1 = 2\ t_n^2 \qquad (7.2)$$
$$t_{n+3}^2 = 6\ t_n^2 + 3\ t_n^1$$

Equations (7.2) are illustrated in Figures 7.18 and 7.19 .

7.4 An Improved Lower Bound For $\bar{\phi}(n)$

In [New] the following problem is posed:
"What is the value of $\bar{\phi}(n)$, the maximum number of crossing-free Hamilton cycles (CFHC) in a rectilinear drawing of K_n ?"

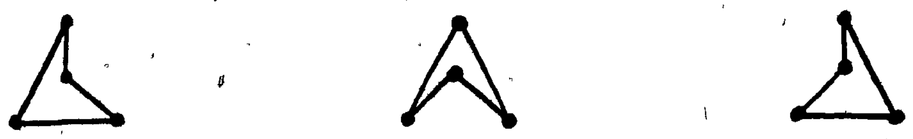


Figure 7.14

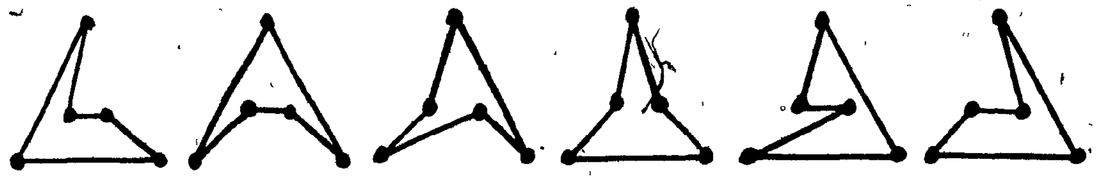


Figure 7.15

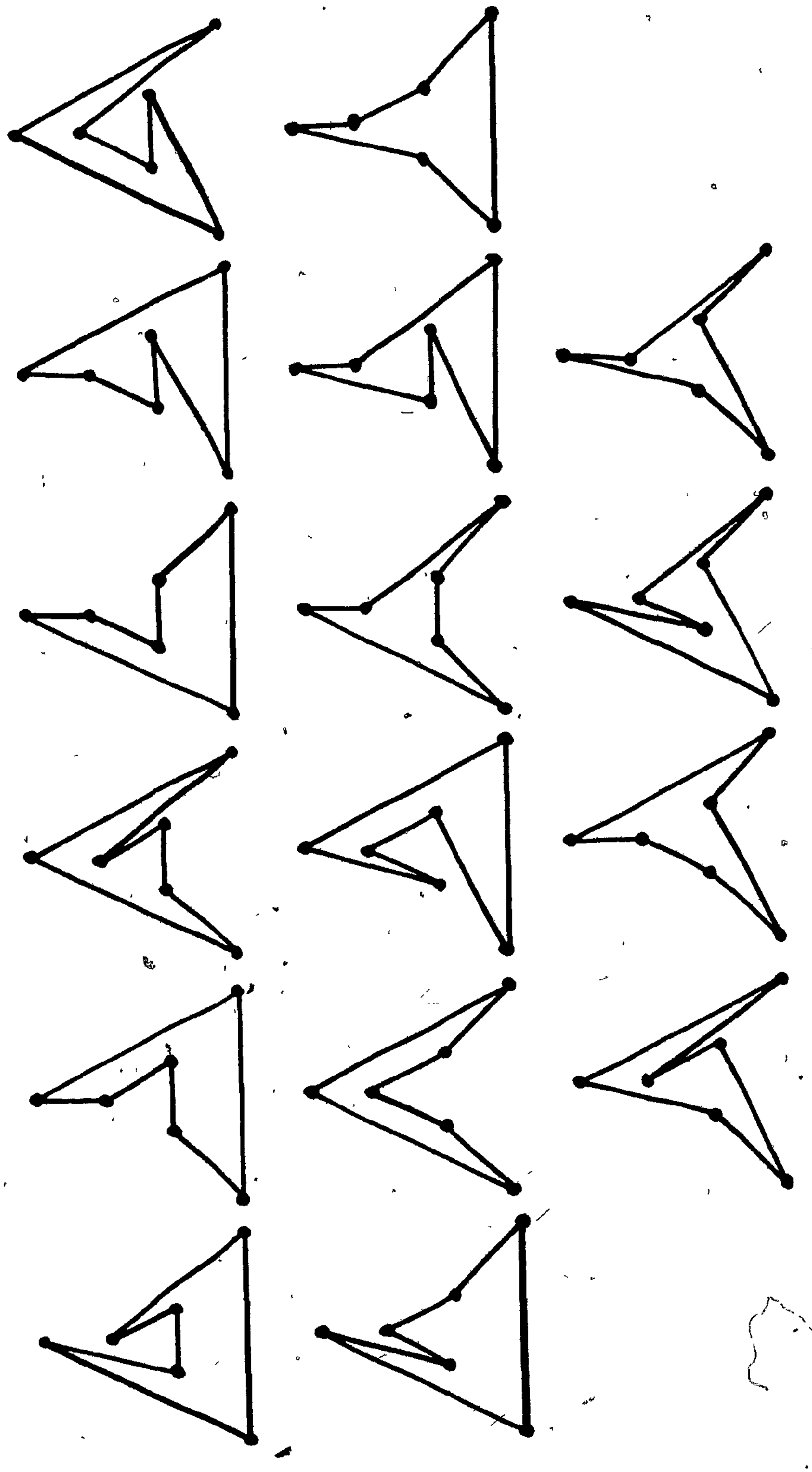
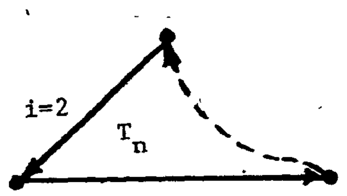


Figure 7.16



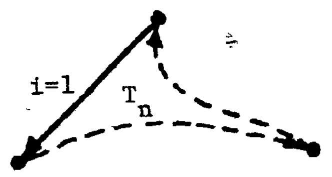
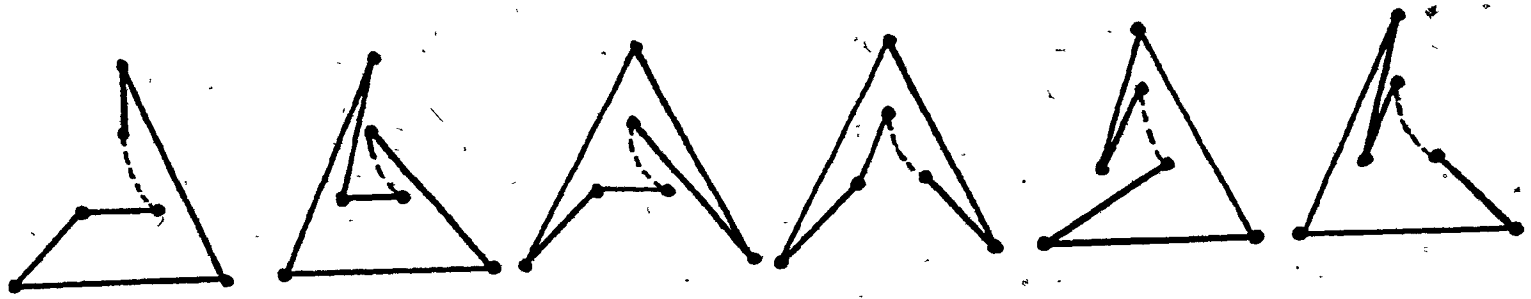
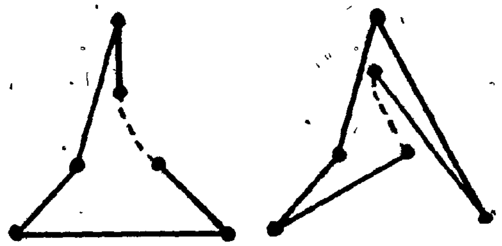


Figure 7.17

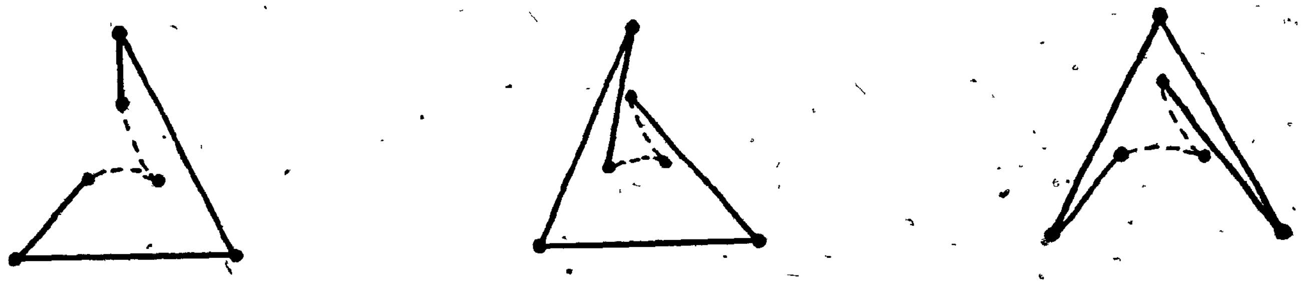


(a) Removal of one edge from the convex hull of a Hamilton cycle in a T_n with i=2 yields 6 Hamilton cycles in T_n+3 with i=2 .



(b) Removal of the two edges from the convex hull of a Hamilton cycle in a T_n with i=2 yields 2 Hamilton cycles in T_n+3 with i=1 .

Figure 7.18



Removal of the only edge from the convex hull of a Hamilton cycle in a T_n with $i=1$ yields 3 Hamilton cycles in T_n+3 with $i=2$.

Figure 7.19

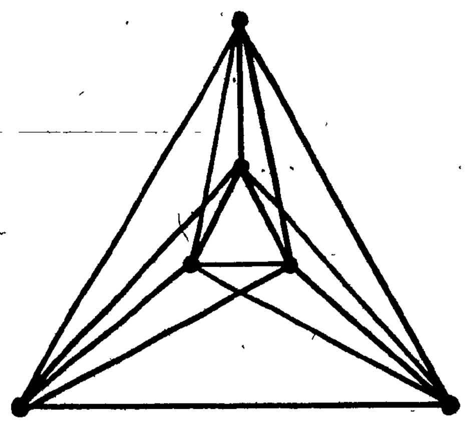
and it is shown that

$$\bar{\phi}(n) \geq \frac{3}{20}\,(10)^{\frac{n}{3}} \tag{7.3}$$

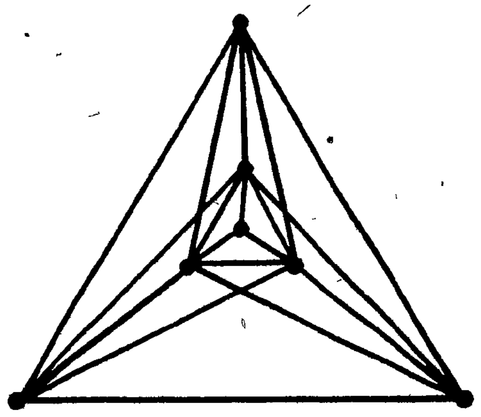
for $n \geq 6$.

Using the technique described in the previous section for counting Hamilton cycles, we obtain a sharper lower bound on $\bar{\phi}(n)$. Essentially, we show that a rectilinear drawing of a subgraph of K_n , denoted by D_n , contains more CFHC's than in (7.3).

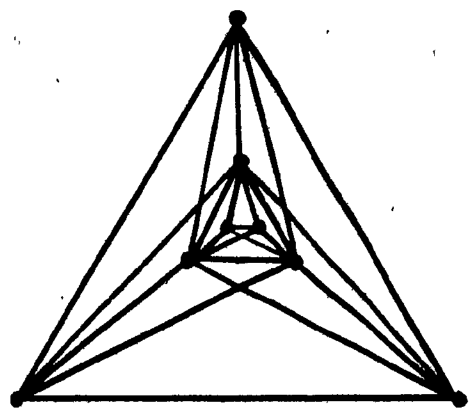
Draw D_n as follows: the n points are placed in $\lfloor n/3 \rfloor$ concentric triangles. The vertices of each triangle are then connected to those of the next interior one by all possible edges as shown in Figure 7.20 .



(a) n≡0 (mod 3)



(b) n≡1 (mod 3)



(c) n≡2 (mod 3)

Figure 7.20

Let us denote by d_n^i the number of CFHC's in a D_n having i edges on the convex hull of D_n as part of each cycle. For D_4 , D_5 and D_6 , and i=1 or 2 we have

$$\begin{aligned} d_4^1 &= 0 \quad & d_4^2 &= 3 \quad & &\text{(as shown in Figure 7.14)} \\ d_5^1 &= 3 \quad & d_5^2 &= 5 \quad & &\text{(as shown in Figures 7.15 and 7.21)} \\ d_6^1 &= 12 \quad & d_6^2 &= 15 \quad & &\text{(as shown in Figures 7.16 and 7.22)} \end{aligned} \qquad (7.4)$$

Excluding the two special cases $d_3^3 = 1$ and $d_6^0 = 2$, it is clear that $d_n^3 = d_n^0 = 0$, for all n , so that the only possible CFHC's in a D_n are when i=2 or 1 , as shown in Figure 7.23 .

Each CFHC in D_n with i=2 leads to 10 and 4 CFHC's in D_{n+3} with i=2 and 1 respectively as shown in Figures 7.18 and 7.24 . Similarly each CFHC in D_n with i=1 leads to 5 CFHC in D_{n+3} with i=2 as shown in Figures 7.19 and 7.25 .

It follows that, for n>6 and the initial conditions in (7.4) , the number of CFHC in a D_n is given by

$$d_n = d_n^1 + d_n^2$$

where

$$\begin{aligned} d_n^1 &= 4 \; d_{n-3}^2 \\ d_n^2 &= 10 \; d_{n=3}^2 + 5 \; d_{n-3}^1 \end{aligned} \qquad (7.5)$$

Solving the recurrence equations (7.5) we get

$$\begin{aligned} d_n &= \frac{9}{2\sqrt{5}} [x_1^{\frac{n-3}{3}} - x_2^{\frac{n-3}{3}}] & & n \equiv 0 \pmod 3 \\ &= \frac{1}{2\sqrt{5}} [(9+3\sqrt{5})x_1^{\frac{n-4}{3}} - (9-3\sqrt{5})x_2^{\frac{n-4}{3}}] & & n \equiv 1 \pmod 3 \qquad (7.6) \\ &= \frac{1}{2\sqrt{5}} [(15+8\sqrt{5})x_1^{\frac{n-5}{3}} - (15-8\sqrt{5})x_2^{\frac{n-5}{3}}] & & n \equiv 2 \pmod 3 \end{aligned}$$

where $x_1 = 5+3\sqrt{5} \simeq 11.7$ and $x_2 = 5-3\sqrt{5} \simeq -1.7$.

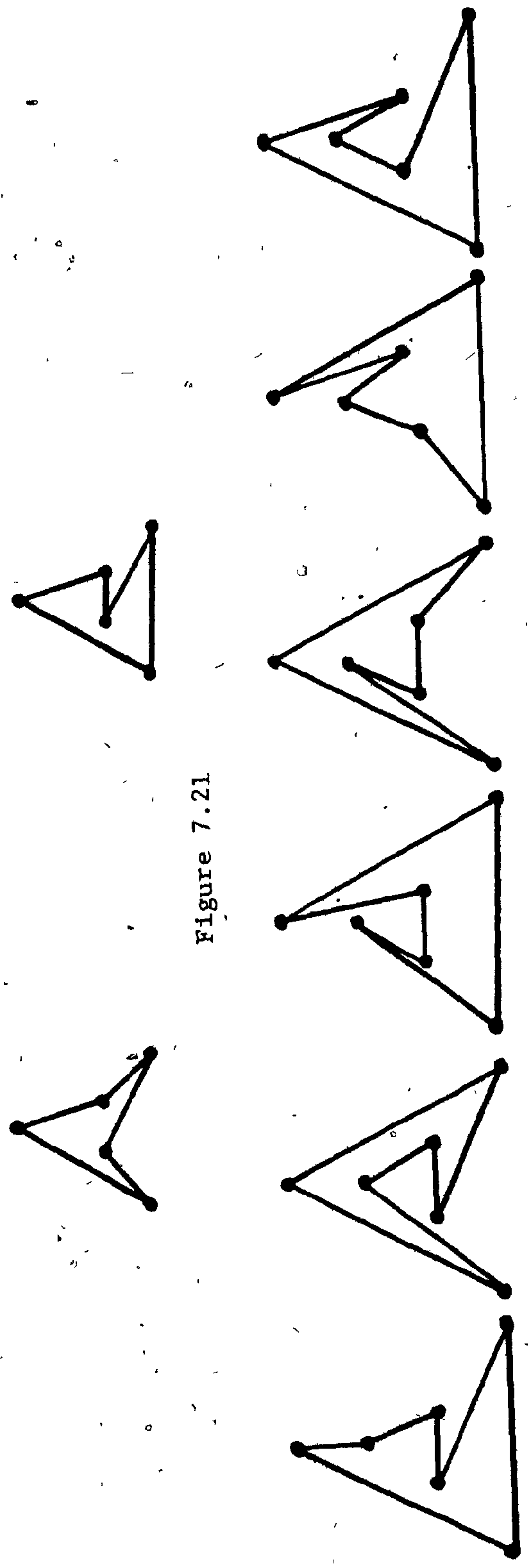


Figure 7.21

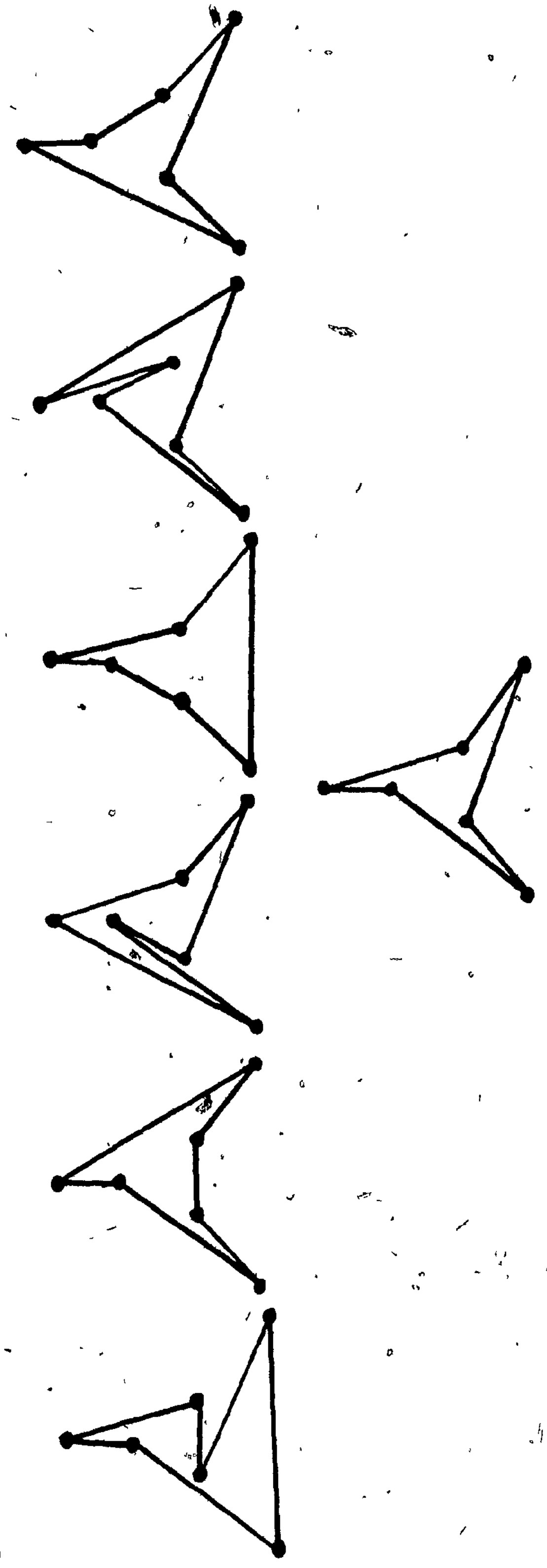


Figure 7.22

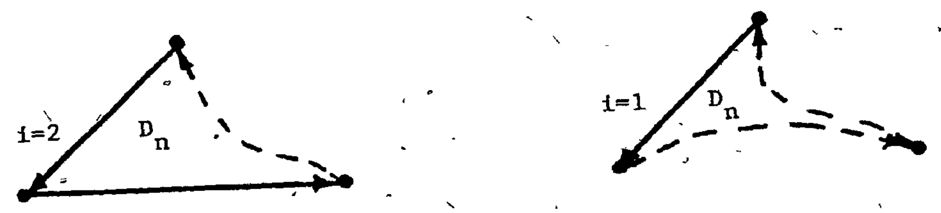
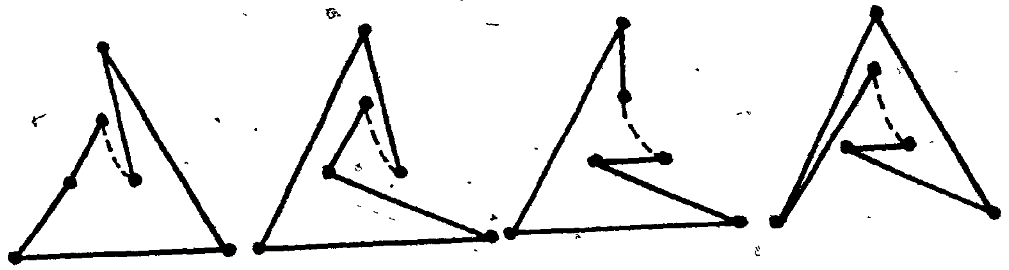
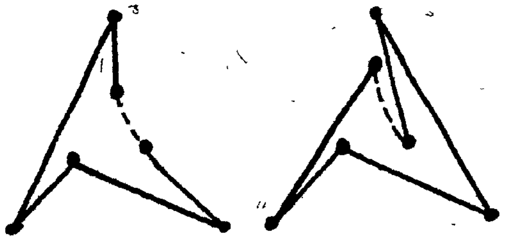


Figure 7.23

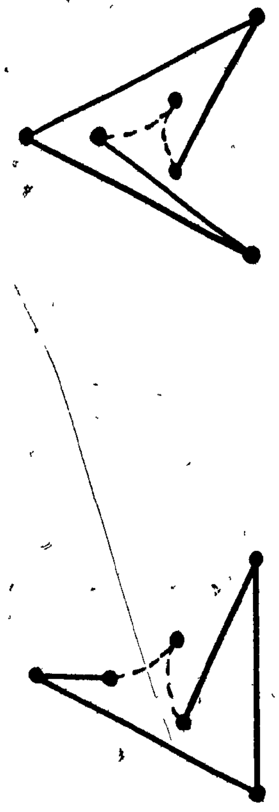


(a) $4\ d_n^2$



(b) $2\ d_n^2$

Figure 7.24



$2\ d_n^1$

Figure 7.25

That $d_n > \frac{3}{20}(10)^{\frac{n}{3}}$ follows directly.

7.5 Conclusion

We have studied a special class of maximal triangulations. Evidence was provided that a relationship exists between this class and rectilinear optimal drawings of complete graphs. Triangulations belonging to this class were shown to always contain a Hamilton cycle and a recurrence formula for the number of such cycles was derived. We finally presented a lower bound on the number of crossing free Hamilton cycles in a rectilinear drawing of a complete graph.

Chapter Eight
On Convex Hulls

Convex hulls (CH) have been extensively studied in the past few years [Ak1, Ak2, Ap, Cha, Di2, Ed, Ef, Fr, Gib1, Gr, Ja, Ku, Pre, Ren, Sh1, Sh2, Sk1, T]. The algorithms of Chapter 3 made use of this concept in connection with approximate triangulations. Other applications include computer graphics, design automation, pattern recognition and operations research.

In this chapter an efficient algorithm for obtaining the CH of a set of points in the plane is presented and theoretically analyzed. For uniform distributions on the square, the algorithm has an expected run-time of O(n). This is in contrast to all other published algorithms whose expected running time is bounded below by O(n log n). Experiments with the algorithm are described that confirm its intuitive and theoretical merits. A generalization to the d-dimensional case is also suggested. Finally, empirical estimates of some convex hull expectations are obtained.

8.1 An Algorithm for the Convex Hull in 2-dimensions

8.1.1 Definitions

Let R^d be the d-dimensional real Euclidean space. A set $K \subset R^d$ is *convex* if and only if for each pair of distinct points a,b $\in$ K the segment with end points a and b is contained in K [Gru]. If $S \subset R^d$ then the *convex hull* of S, denoted by *conv* S, is the intersection of all the convex sets in R^d which contain S [Gru].

The example of Figure 8.1 shows, for d = 2, S and *conv* S.

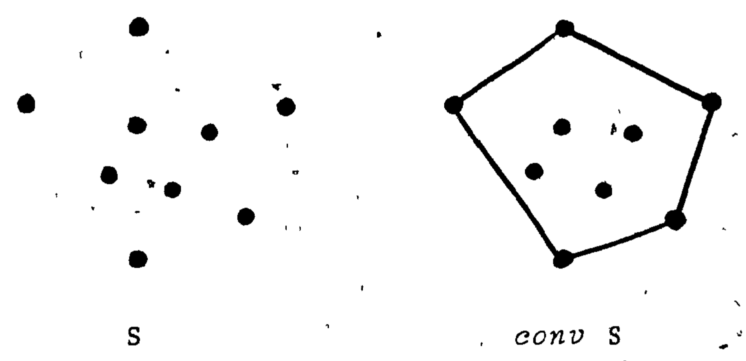


Figure 8.1

The convex hull of a set of points in the plane is, therefore, the smallest convex polygon including all the points.

8.1.2 Previous Work

Several algorithms have been presented for obtaining the convex hull of a set of points. Some of them apply only to the planar [Gr, Ja, Sh2, Ed] or 3-dimensional [Ap, Di2] case, while others are for the general d-dimensional convex hull problem [Cha, Pre]. In the 2-dimensional case, the algorithms of [Gr, Pre] require sorting, while the algorithm of [Sh2] is based on finding the Voronoi diagram [Sh1] : each of these operations has a complexity of O(n log n), where n is the number of points. The algorithms of [Ja, Ed] have a complexity of O(mn), where m is the number of vertices of the CH ; this means that their worst-case behaviour is $O(n^2)$. Also, it was shown in [Ren] that the expected value of m is O(log n) for a uniform distribution of the n points; the algorithms of [Ja, Ed] have therefore an *expected* run-time of O(n log n).

The complexity of the convex hull problem has been established in [Pre, Sh1] as being of O(n log n) for a set of n points.

8.1.3 The Algorithm

8.1.3.1 Basic Ideas

The new algorithm is based on the following simple ideas:

(1) Determining the four extremal points of the set and discarding all points interior to the convex quadrilateral they form.

(2) Breaking the problem into four subproblems determined by the extremal points.

(3) Using the vector cross-product to find the convex path in each problem.

These ideas are now explained in detail. We assume throughout the following discussion that points are given by their cartesian coordinates.

(1) Extremal points:

These are the four points with minimum and maximum X and Y coordinates:. say XMIN, XMAX, YMIN, YMAX, respectively. From Figure 8.2,

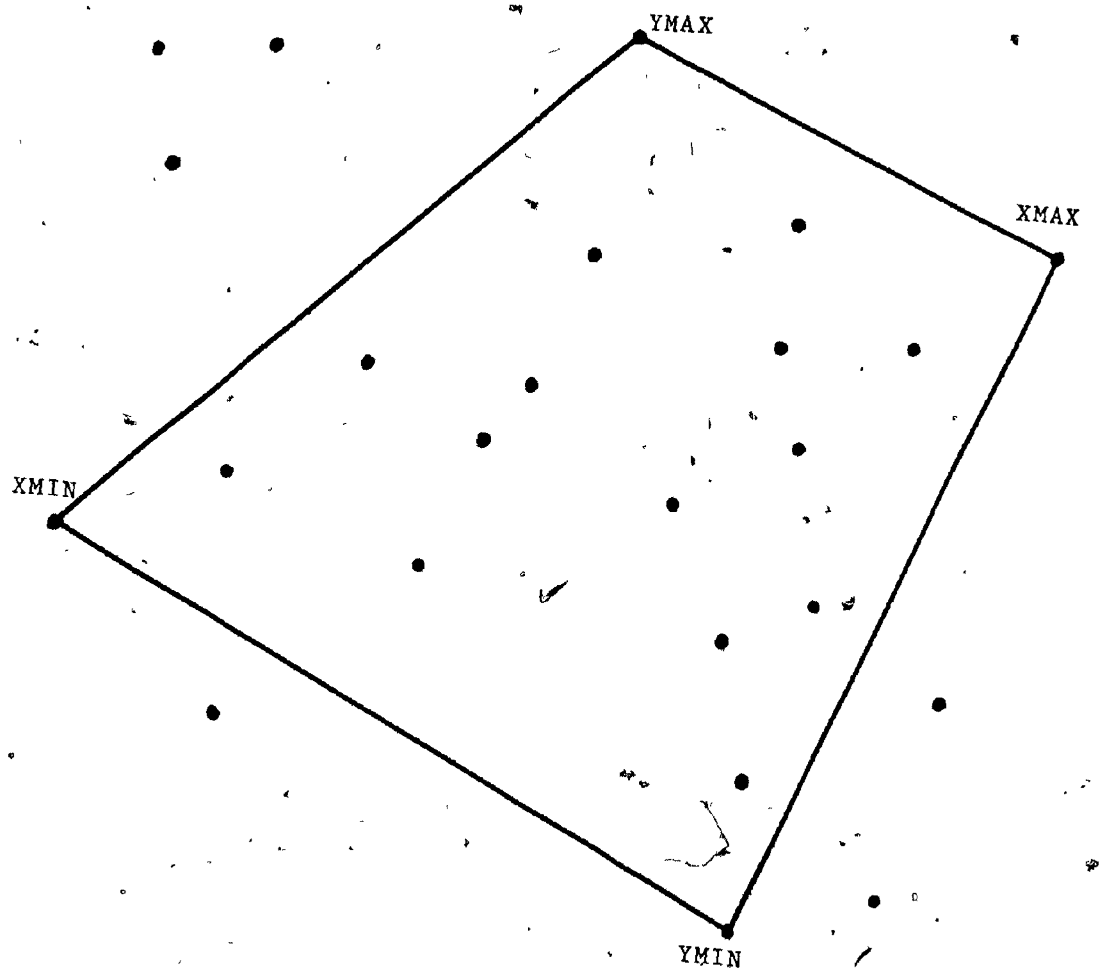


Figure 8.2

two facts are obvious,

(a) The extremal points must belong to the convex hull.

(b) Any point interior to the convex quadrilateral whose corners are the extremal points cannot belong to the convex hull.

It follows that by identifying the extremal points one adds these points to the convex hull and discards all points falling inside the quadrilateral they form; we call this the "throw-away" principle.

(2) Subproblems:

Once the four extremal points have been determined, and some points eventually discarded, one can break the remaining set of points into four regions, as shown in Figure 8.3 . All that remains now is to find a convex "path" from one extremal point to the other in the same region.

(3) Vector cross-product:

While examining the points in one of the regions for inclusion in (or exclusion from) the convex hull, assume that we are advancing along an edge of the quadrilateral such that the region is at our left, as shown in Figure 8.3 . Assume further that we are looking at three consecutive points k, k+1, and k+2 . Obviously if point k+1 is as shown in Figure 8.4(a) it is to be kept temporarily, while it is to be discarded from further consideration if it is as in Figure 8.4(b) .

If a, b and Θ are as shown in Figure 8.4(a) and (b), then the cross-product of the two vectors is given by

$$\begin{aligned} S &= a\ b\ \sin\Theta \\ &= a\ b\ \sin(\alpha_1 + \alpha_2) \\ &= a\cdot b\ [\sin\alpha_1 \cos\alpha_2 + \cos\alpha_1 \sin\alpha_2] \\ &= a\ b\ \left[\frac{y_{k+1} - y_k}{a} \times \frac{x_{k+2} - x_{k+1}}{b} + \frac{x_k - x_{k+1}}{a} \times \frac{y_{k+2} - y_{k+1}}{b}\right] \\ &= (y_{k+1} - y_k)(x_{k+2} - x_{k+1}) + (x_k - x_{k+1})(y_{k+2} - y_{k+1}) . \end{aligned}$$

In Figure 8.4(a), S is positive and in Figure 8.4(b) it is negative. We thus have the following simple rule:

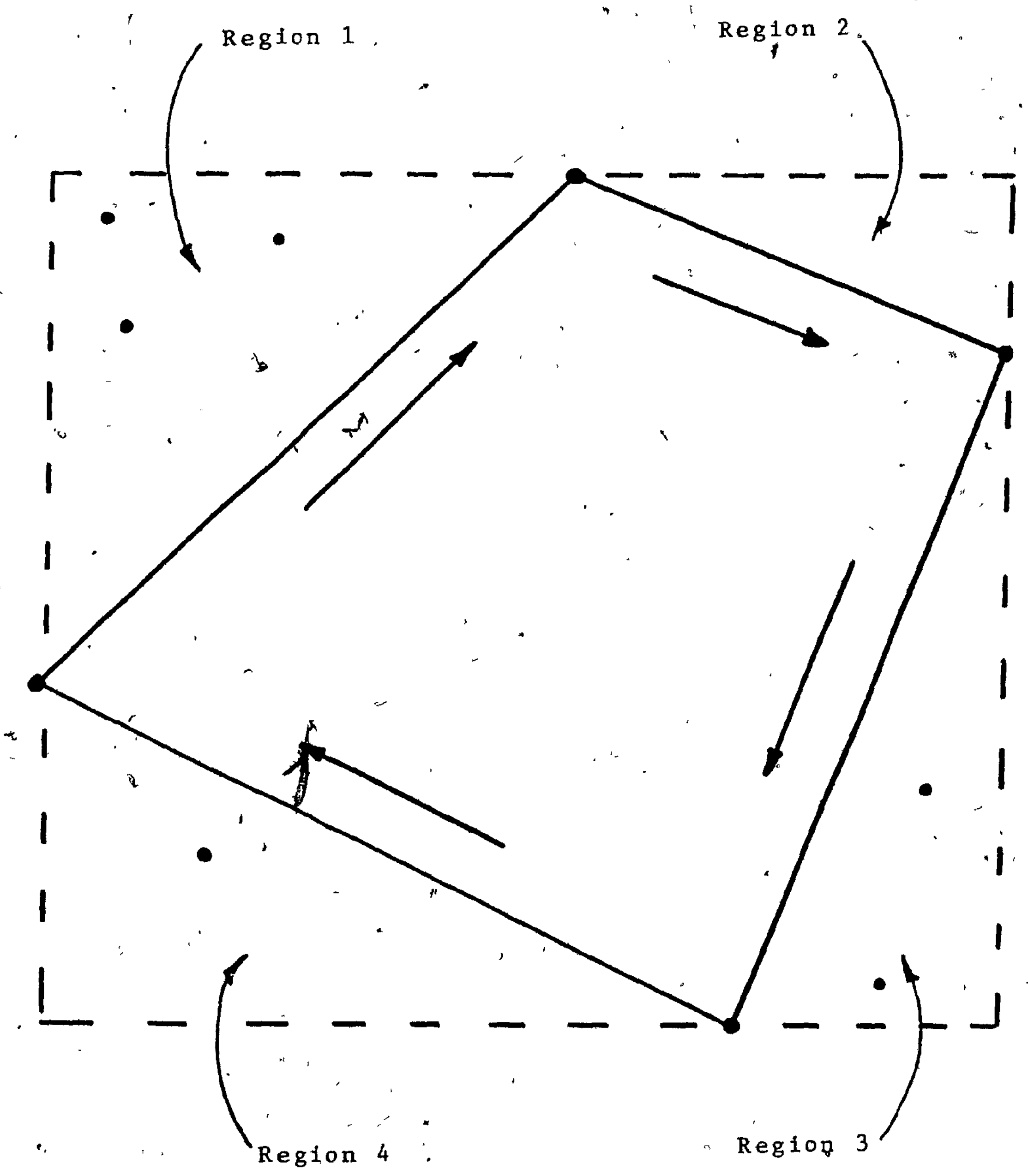


Figure 8.3

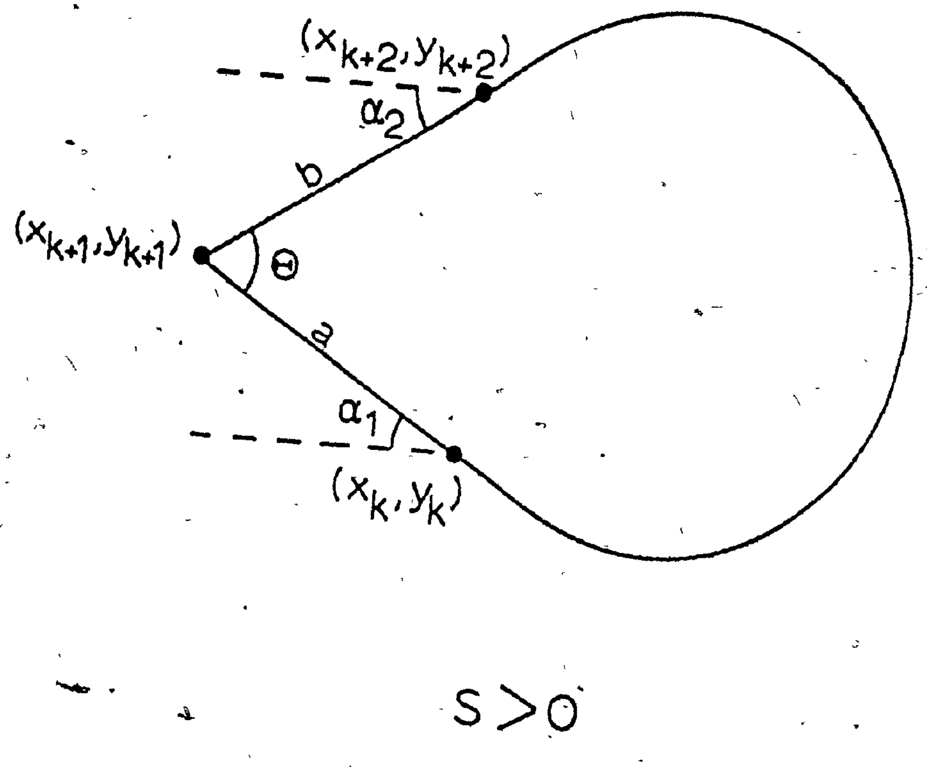


Figure 8.4(a)

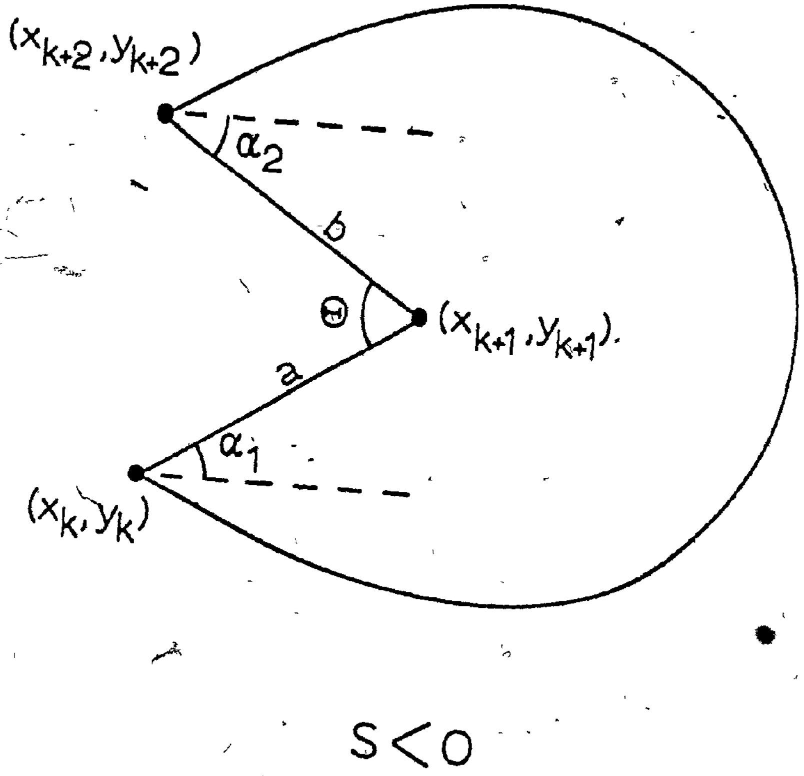
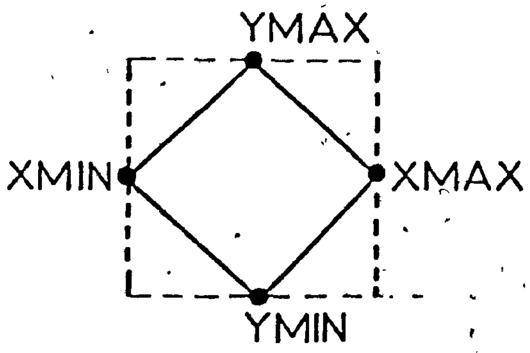
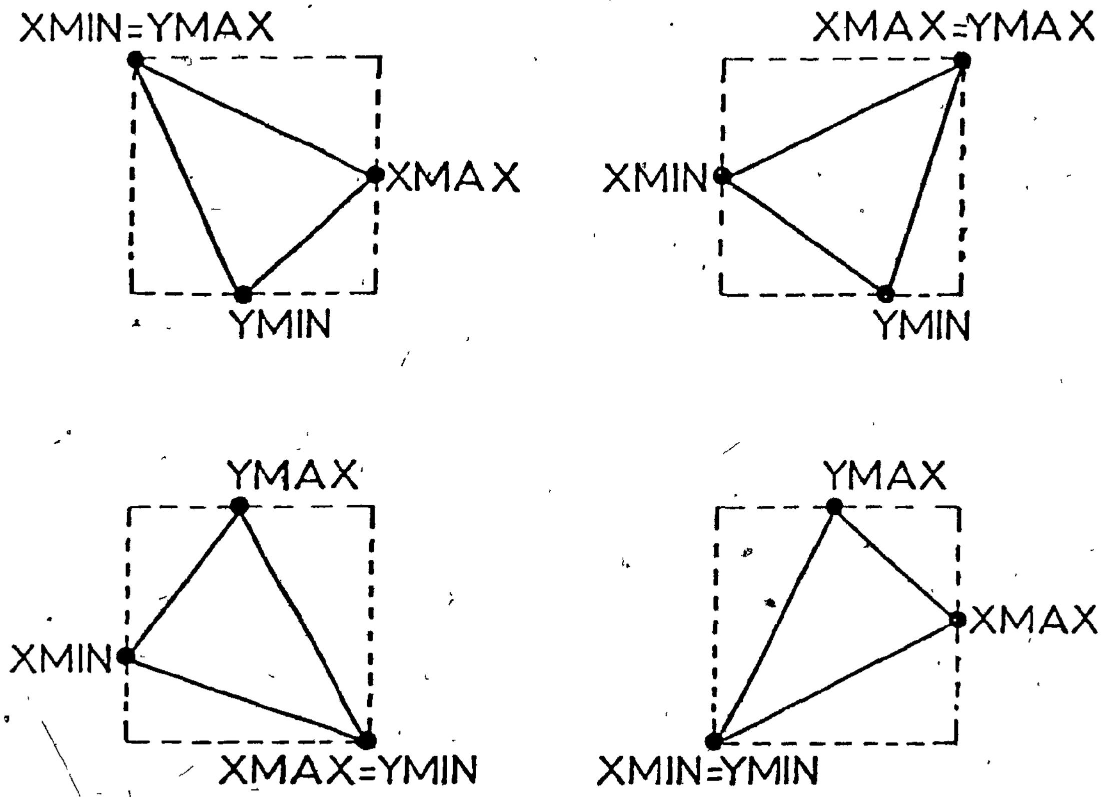


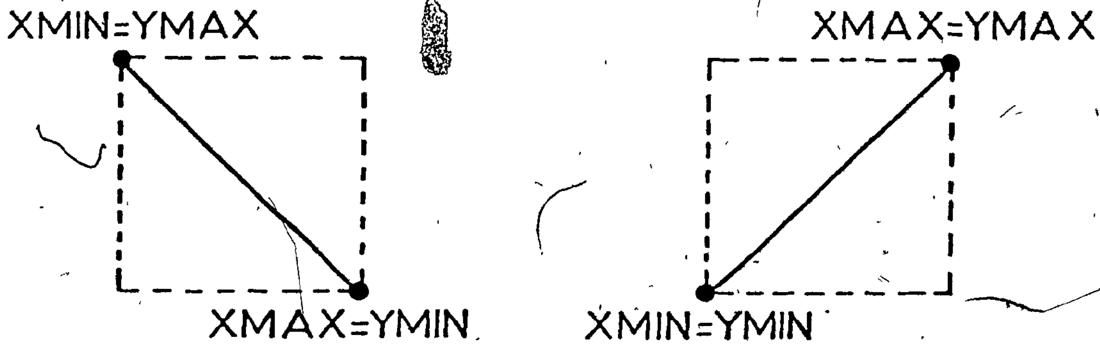
Figure 8.4(b)



(a) 4 regions



(b) 3 regions



(c) 2 regions

Figure 8.5

If $S \geq 0$ keep point (k+1)
else delete point (k+1) .

Before presenting the algorithm we make the following two remarks:

(1) In some cases two extremal points may coincide as shown in Figure 8.5 . The sole effect of these situations is that the number of sub-problems is reduced.

(2) The "throw-away" principle can be applied further to each of the four regions of Figure 8.3 . We illustrate this idea for region 2 as shown in Figure 8.6 .

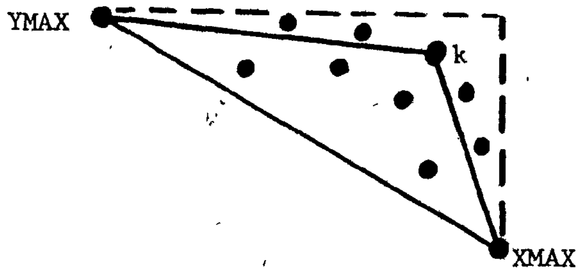


Figure 8.6

Let k be a point inside region 2. It is obvious that any point inside the triangle formed by points YMAX, k and XMAX cannot be a point of the convex hull and should, therefore, be discarded. In order to maximize the number of points thrown away by this method, we choose point k according to the following heuristic:

"among all points inside region 2 choose point (x_k, y_k) such that $x_k + y_k$ is maximum".

8.1.3.2 Algorithm CH

Given n points in the plane by their cartesian coordinates, find their convex hull.

Step 1 Determine the four extremal points and remove from further consideration all points falling inside the quadrilateral they form. The remaining points are distributed among the four external regions thus created.

Step 2 For every one of the four external regions, determined by the two extremal points i and j , find k , the point whose coordinates (x_k , y_k) maximize the quantity

$$m_1 x_k + m_2 y_k$$

where

m_1 = +1 for regions 2 and 3
= -1 for regions 1 and 4 ,

and

m_2 = +1 for regions 1 and 2
= -1 for regions 3 and 4 .

Remove from further consideration all points falling inside the triangle ijk .

Step 3 For every one of the four external regions sort the remaining points on their x-coordinate: in ascending order if in region 1 or 2 and descending order if in region 3 or 4.

Step 4 For every one of the four external regions find the convex path from one extremal point i to the other j using the following rule:

"(1) Starting with i do (a) and (b) below for every three consecutive points k, k+1 and k+2 until j is reached

(a) $S = (y_{k+1} - y_k)\ (x_{k+2} - x_{k+1}) + (x_k - x_{k+1})\ (y_{k+2} - y_{k+1})$

(b) If $S \geq 0$ move one point forward;
Else delete point (k+1) and move one point backward.

(2) If (1) is completed without any deletion, stop; else repeat (1)".

Since at every iteration of Step 4(1) a finite number of points is removed, termination of the algorithm is guaranteed. The remaining points form the convex hull of the original set.

8.1.4 Analysis

Algorithm CH has the following advantages:

(1) Conversion to polar coordinates and computation of angles [Gr], as well as shifting of axes [Ja], all costly operations, are avoided.

(2) The algorithm is conceptually very simple when compared to those in [Ed, Pre, Sh2] and thus has pedagogical significance.

(3) The set of points to be sorted is reduced to a very small subset by the "throw-away" principle.

(4) The convex path in each region is found by the very-easy-to-implement cross-product rule.

(5) Breaking the problem into subproblems makes it easier, and therefore faster, to solve. This is a good illustration of the "divide-and-conquer" concept [Aho].

(6) The worst-case behavior of the algorithm is O(n log n). This happens when all the points lie on their convex hull. For example, if all the points lie on a circular arc (x_1,y_1), $(x_2,y_2),\ldots,$ (x_n,y_n) such that (x_1,y_1) is YMAX and (x_n,y_n) is both XMAX and k, then no points are thrown away in Steps 1 and 2 and therefore the (n-2) points remaining are sorted.

We now present a theoretical analysis of CH. In order to do so we assume that the n points are independent random variables uniformly distributed on the unit square. We use the following notation

Symbol	meaning
XMAX	point with maximum x- coordinate
YMAX	point with maximum y- coordinate
XMIN	point with minimum x- coordinate
YMIN	point with minimum y- coordinate
XYMAX	point with maximum (x+y) value
YXMAX	point with maximum (y-x) value
XYMIN	point with minimum (x+y) value
YXMIN	point with minimum (y-x) value

8.1.4.1 Analysis Algorithm

Since algorithm CH itself is very difficult to analyze due to the conditional nature of Step 2 we define an analysis algorithm which is easier to analyze than CH and which provides an upper bound on CH ; i.e. on every instance, algorithm CH will perform equally well or better than the analysis algorithm.

Algorithm CHA

Step 1 Determine the eight points XMAX, XMIN, YMAX, YMIN, XYMAX, XYMIN, YXMAX, YXMIN and throw-away any point falling inside the polygon they form.

Step 2 Same as Step 3 of CH .

Step 3 Same as Step 4 of CH .

Comments

(1) Note that in Step 1 of the analysis algorithm the 8 points need not be distinct.

(2) The analysis algorithm is slower than CH for two reasons:

(a) It requires at least 2n additions and subtractions in Step 1 to find XYMAX, XYMIN, YXMAX and YXMIN; whereas in Step 2 of CH the analogous points (point k in Figure 8.6) can be found by performing additions and subtractions only for points in the four sub-regions.

(b) The number of points thrown away in Step 1 of CHA is smaller than that in Step 1 and Step 2 of CH as illustrated in Figure 8.7 .

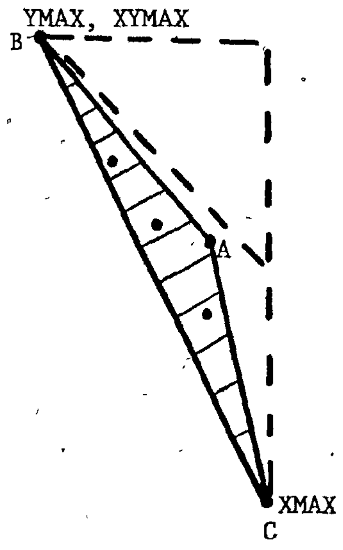


Figure 8.7

In this situation the above algorithm will choose points B and C for YMAX, XYMAX and XMAX, whereas CH , having chosen B and C as YMAX and XMAX, will then choose A as XYMAX and thus throw-away the additional points falling in the shaded triangle.

From the above comments, it follows that an upper bound on the expected running time of CHA will also serve as an upper bound on CH .

In order to determine an upper bound on the expected running time of algorithm CHA we need to know a lower bound on the expected number of points discarded in Step 1, which is proportional to the expected area of the polygon formed by the 8 extreme points.

8.1.4.2 <u>Area of polygon</u>

Let the 8 extreme points determined in Step 1 be denoted by $p_1(x_1,y_1)$, $p_2(x_2,y_2)$,..., $p_8(x_8,y_8)$. The area of the polygon is given by:

$$A = \frac{1}{2} \left| \sum_{i=1}^{8} x_i \, (y_{i+1} - y_{i-1}) \right| \qquad (8.1)$$

where $y_0 = y_8$ and $y_9 = y_1$.

Note that if two or more extreme points coincide (have the same coordinates) this expression would give the area of the corresponding polygon. The expected value of (8.1) is given by

$$E\,\{A\} = \frac{1}{2} E\,\{\left| \sum_{i=1}^{8} x_i \, (y_{i+1} - y_{i-1}) \right|\} \qquad (8.2)$$

Since the absolute value function is concave, it follows from Jensen's inequality that

$$\begin{aligned} E\,\{A\} &\geq \frac{1}{2} \left| E\,\{ \sum_{i=1}^{8} x_i \, (y_{i+1} - y_{i-1})\} \right| \\ &= \frac{1}{2} \left| \sum_{i=1}^{8} E\,\{x_i \, y_{i+1}\} - E\,\{x_i \, y_{i-1}\} \right| \end{aligned} \qquad (8.3)$$

Now $E\{x_i \, y_{i+1}\} = E\{x_i\} \, E\{y_{i+1}\} + \mathrm{cov}\,\{x_i \, y_{i+1}\}$. Since, as $n \to \infty$, $\mathrm{cov}\{x_i \, y_{i+1}\} \to 0$ faster than $1/\log n \to 0$ [Ak2], we have

$$E\{x_i\, y_{i+1}\} = E\{x_i\}\, E\{y_{i+1}\} \quad \text{for large } n$$

and thus

$$E\{A\} \geq \frac{1}{2} \left| \sum_{i=1}^{8} E\{x_i\} (E\{y_{i+1}\} - E\{y_{i-1}\}) \right| \qquad (8.4)$$

Therefore, we can obtain a lower bound on the expected area if we know the expected values of the coordinates of the extreme points, to which we now turn.

Expected values of extreme points

The extreme points fall into two categories: type 1 consisting of XMAX, XMIN, YMAX, YMIN, and type 2 made up of XYMAX, XYMIN, YXMAX, YXMIN.

Expected values of type 1 points

The problem is essentially the following: given n independent random variables $x_1, x_2, \ldots, x_n$ distributed uniformly on [0,1], what is the expected value of

$$V = \max(x_1, x_2, \ldots, x_n) .$$

Consider first the c.d.f. of V

$$\begin{aligned} P[V < a] &= P\,[\max(x_1,x_2,\ldots,x_n) < a] \\ &= P\,[x_1,x_2,\ldots,x_n < a] \end{aligned} \qquad (8.5)$$

Since the x's are independent and identically distributed

$$\begin{aligned} P[V < a] &= P\,[x_1 < a] \cdot P\,[x_2 < a] \ldots P\,[x_n < a] \\ &= P\,[x < a]^n \end{aligned} \qquad (8.6)$$

where we drop the subscript for convenience. Since x is uniformly distributed on [0,1], it follows that $P\,[V < a] = a^n$. By differentiation, the density f(a) of V is given by

$$f(a) = na^{n-1} .$$

Hence $$E\{V\} = \int_0^1 a\, f(a)\, da = \frac{n}{n+1} .$$

By symmetry it follows that the expected value of $\min(x_1,x_2,\ldots,x_n) =$

$1 - \frac{n}{n+1} = \frac{1}{n+1}$.

Every one of the four extreme points has a coordinate that is not extremized and whose expected value is thus equal to 1/2 . It follows that the expected values of the coordinates of the four extreme points of type 1 are as shown in Figure 8.8 .

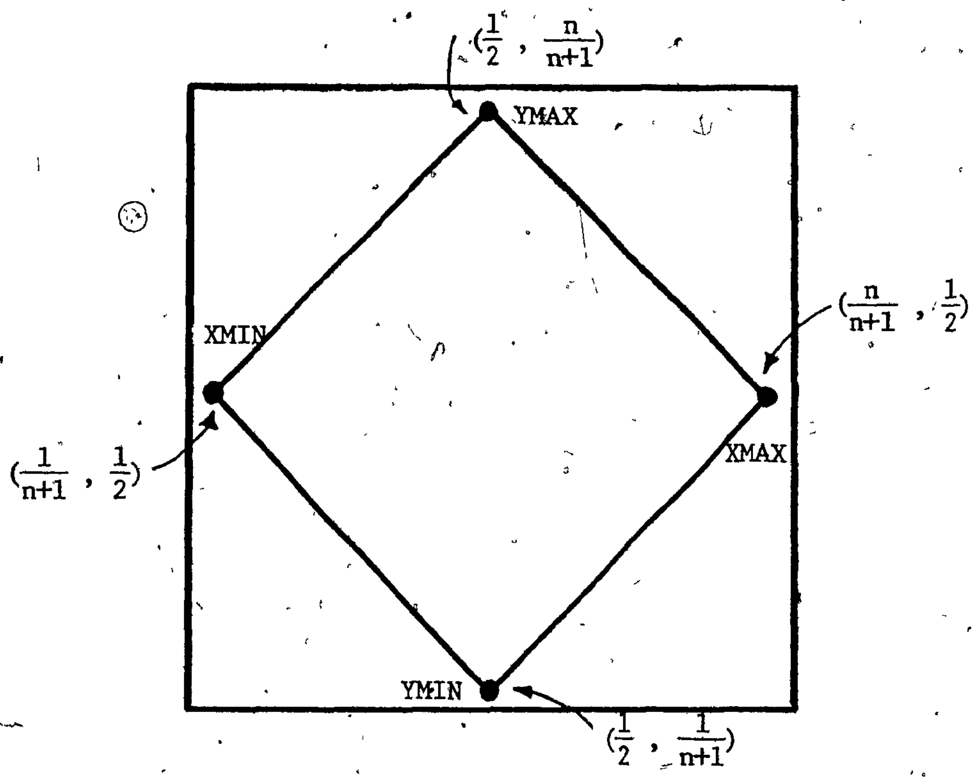

$(\frac{1}{2}, \frac{n}{n+1})$
YMAX
XMIN
$(\frac{n}{n+1}, \frac{1}{2})$
$(\frac{1}{n+1}, \frac{1}{2})$
XMAX
YMIN
$(\frac{1}{2}, \frac{1}{n+1})$

Figure 8.8

Expected values of type 2 points

The problem reduces to the following: Given n independent random variables $Z_1, Z_2, \ldots, Z_n$ such that $Z_i = x_i + y_i$, $i = 1,2,\ldots,n$, and x_i, y_i are independent random variables uniformly distributed on $[0,1]$, what is the expected value of $W = \max(Z_1, Z_2, \ldots, Z_n)$. Consider first the c.d.f. of W .

$$\begin{aligned} P[W < a] &= P\,[\max\,(Z_1, Z_2, \ldots, Z_n) < a] \\ &= P\,[Z_1, Z_2, \ldots, Z_n < a] \\ &= P[Z_1 < a] \,.\, P[Z_2 < a] \;\ldots\; P[Z_n < a] \\ &= P[Z < a]^n \,, \end{aligned}$$

where we drop the subscript for convenience. We therefore need the c.d.f. of Z . Since $Z = x + y$, it follows that Z has a triangular density on [0,2] with corresponding c.d.f. given by

$$P[Z < a] = \begin{cases} 0 & \text{when } a < 0 \\ \frac{1}{2} a^2 & \text{when } 0 \leq a < 1 \\ 1 - \frac{1}{2}(2-a)^2 & \text{when } 1 \leq a \leq 2 \\ 1 & \text{when } a > 2 \end{cases}$$

It follows that the c.d.f. of W is given by

$$P[W < a] = \begin{cases} 0 & \text{when } a < 0 \\ \frac{1}{2^n} a^{2n} & \text{when } 0 \leq a < 1 \\ [1 - \frac{1}{2}(2-a)^2]^n & \text{when } 1 \leq a \leq 2 \\ 1 & \text{when } a > 2 \end{cases}$$

By differentiating we obtain the density g(a) of W

$$g(a) = \begin{cases} 0 & \text{when } a < 0 \\ \frac{2n}{2^n} a^{2n-1} & \text{when } 0 \le a < 1 \\ n[1 - \frac{1}{2}(2-a)^2]^{n-1}(2-a) & \text{when } 1 \le a \le 2 \\ 0 & \text{when } a > 2 \end{cases}$$

Hence the expected value of W is given by

$$E\{W\} = \int_0^2 a\, g(a)\, da$$

Substituting for g(a) we obtain

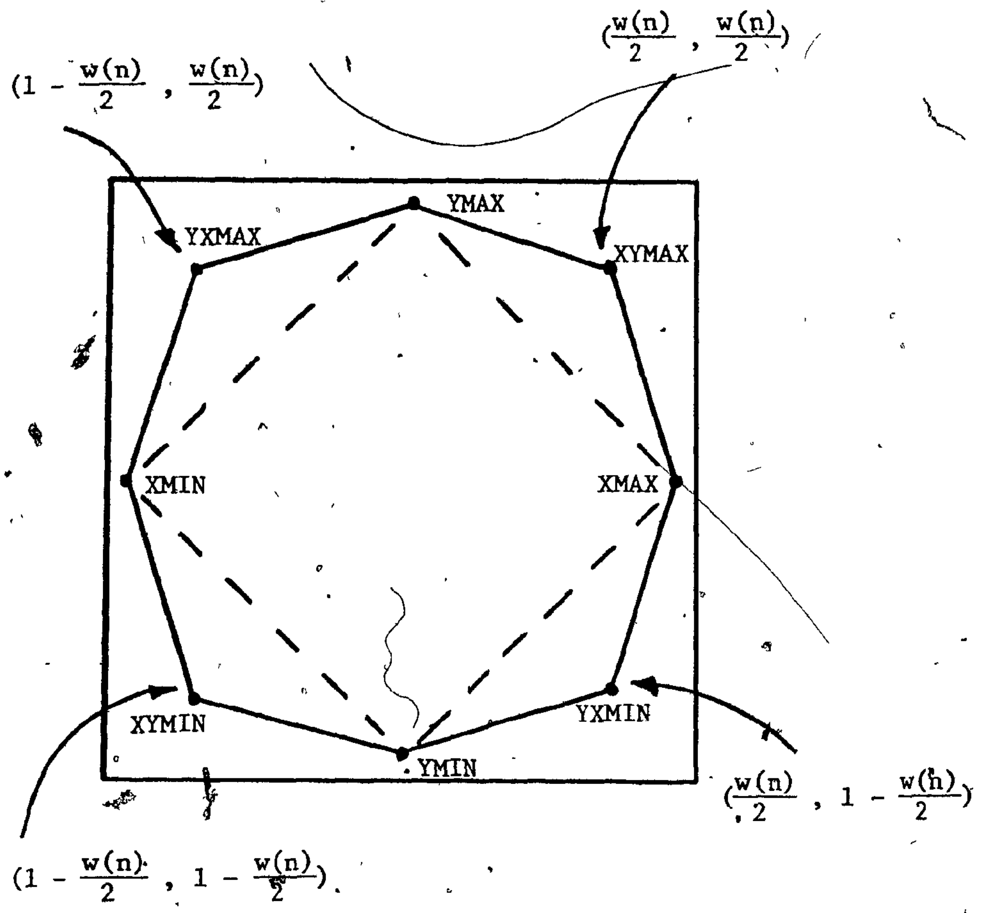
$$E\{W\} = \int_0^1 \frac{2n}{2^n} a^{2n} da + \int_1^2 an[1 - \frac{1}{2}(2-a)^2]^{n-1}(2-a)da$$

$$= \frac{1}{2^n}\left[\frac{2n}{2n+1}\right] + 2 - \frac{1}{2^n} - \frac{\sqrt{2}\,(2n)!!}{(2n+1)!!} \qquad (8.7)$$

Let (8.7) be denoted by w(n) . By symmetry the expected value of min $(Z_1, Z_2, \ldots, Z_n) = 2 - w(n)$. Consider region 2. Let x_{max} and y_{max} denote the x and y coordinates of XYMAX . Then $W = x_{max} + y_{max}$ and $E\{W\} = E\{x_{max}\} + E\{y_{max}\}$. Because x_{max} and y_{max} are identically distributed

$$E\{x_{max}\} = E\{y_{max}\} = \frac{1}{2} E\{W\} = \frac{1}{2} w(n) \qquad (8.8)$$

It follows that the expected values of the coordinates of the four extreme points of type 2 are as shown in Figure 8.9 .



$(1 - \frac{w(n)}{2}, \frac{w(n)}{2})$
$(\frac{w(n)}{2}, \frac{w(n)}{2})$
YXMAX
YMAX
XYMAX
XMIN
XMAX
XYMIN
YXMIN
YMIN
$(\frac{w(n)}{2}, 1 - \frac{w(n)}{2})$
$(1 - \frac{w(n)}{2}, 1 - \frac{w(n)}{2})$

Figure 8.9

The expected values of the four extreme points of type 1 define a quadrilateral Q. Similarly, the expected values of the four extreme points of type 2 define four identical triangles T. Hence (8.4) can be written as

$$E\{A\} = E\{Q\} + 4\,E\{T\} \tag{8.9}$$

Substituting in (8.4) the coordinates of the 4 points in Figure 8.8 yields

$$E\{Q\} \geq \frac{1}{2}\left(\frac{n-1}{n+1}\right)^2 \tag{8.10}$$

Substituting in (8.4) the coordinates of points YMAX, XMAX and XYMAX in Figure 8.9 yields

$$E\{T\} \geq \frac{1}{2}\left[w(n)\left(\frac{n}{n+1}\right) - \frac{w(n)}{2} - \left(\frac{n}{n+1}\right)^2 + \frac{1}{4}\right] \tag{8.11}$$

By substituting (8.10) and (8.11) into (8.9) we obtain the desired lower bound on E $\{A\}$.

$$E\{A\} \geq [w(n) -1]\left(\frac{n-1}{n+1}\right) \tag{8.12}$$

Let n' be the expected number of points discarded in Step 1 of the algorithm. A lower bound on n' is thus given by

$$n' = n\, E\{A'\} \geq n[w(n-1)-1]\left(\frac{n-2}{n}\right)$$

where A' is computed over (n-1) points [Ak2].

Theorem 8.1

For large n , n' ≃ n .

Proof: It suffices to show that $w(n) \to 2$ as $n \to \infty$. From (8.7)

$$\lim_{n\to\infty} w(n) = 2 - \lim_{n\to\infty}\left\{\frac{1}{2^n(2n+1)}\right\} - \sqrt{2}\lim_{n\to\infty}\left\{\frac{(2n)!!}{(2n+1)!!}\right\} \tag{8.13}$$

The first limit on the right hand side of (8.13) is equal to zero. Consider the second limit term. We have

$$\frac{(2n)!!}{(2n+1)!!} = \frac{2\cdot 4\cdot 6\cdot 8 \dots (2n)}{3\cdot 5\cdot 7\cdot 9 \dots (2n+1)} = \prod_{j=1}^{n}\frac{1}{1+\frac{1}{2j}} \tag{8.14}$$

The denominator of (8.14) is given by

$$\begin{aligned}\prod_{j=1}^{n}\left(1+\frac{1}{2j}\right) &> 1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{6} + \dots \\ &= 1 + \frac{1}{2}\left(1 + \frac{1}{2} + \frac{1}{3} + \dots\right) \\ &= 1 + \frac{1}{2} H_n\end{aligned}$$

where H_n is the harmonic series. Since H_n goes to infinity, the second limit goes to zero and $w(n) \to 2$. The remaining part of the proof is straightforward using this result in (8.12) .

Q.E.D.

Remarks

This theorem implies that for most practical situations the majority of points is discarded in Steps 1 and 2 of CH . Furthermore, for large n and uniformly distributed points on the unit square, CH discards almost all points. This is because points such as YMAX in Figure 8.9 converge to $(\frac{1}{2}, 1)$ and XYMAX to (1 , 1) as $n \to \infty$.

8.1.4.3 Asymptotic Expected Complexity

Finding the extreme points and discarding points inside the polygon in Step 1 can always be done in time proportional to n [Bu]. If n* points are discarded in Step 1 then n - n* points are sorted in Step 2. This can be done in time proportional to (n-n*) log (n - n*) . Step 3 can be done in time proportional to (n-n*) . The complexity of this algorithm can thus be written as

$$\begin{aligned} C &= k_1 n + k_2 (n-n^*) \log (n-n^*) \\ &\le k_1 n + k_2 (n-n^*) \log n \qquad (8.15) \end{aligned}$$

where k_1 and k_2 are constants. The expected complexity is thus

$$\begin{aligned} E\{C\} &\le k_1 n + k_2 (n-n') \log n \\ &= k_1 n + k_2 (n-n\, E\{A'\}) \log n \\ &= O(n) + k_2 (1 - E\{A'\})\, n \log n \qquad (8.16) \end{aligned}$$

Theorem 8.2

For uniform distributions on the square, the expected running time of the algorithm is O(n).

Proof: It is required to show that the first term in (8.16) dominates early enough. Theorem 8.1 tells us that (8.12) tends to 1 ; we now demonstrate that this happens sufficiently fast: the convergence of each term in (8.12) to its limit is shown to be faster than the convergence of the logarithmic function. Consider w(n) first:

$$w(n) = 2 - \frac{1}{2^n(2n+1)} - \sqrt{2}\,\frac{(2n)!!}{(2n+1)!!}$$

$$> 2 - \frac{2}{\log n} - \frac{4\sqrt{2}}{\log n} = 2\left(1 - \frac{1+2\sqrt{2}}{\log n}\right)$$

$$\text{since } \frac{1}{2^n(2n+1)} = \frac{1}{2^{n+1}(2n+1)} < \frac{2}{\log n}\ ,$$

$$\text{and } \frac{(2n)!!}{(2n+1)!!} < \frac{1}{1+\frac{1}{2}H_n} < \frac{2}{H_n} < \frac{4}{\log n}\ .$$

Now consider $\frac{n-1}{n+1}$:

$$\frac{n-1}{n+1} = 1 - \frac{2}{n+1} > 1 - \frac{2}{\log n}\ .$$

Q.E.D.

8.1.5 A Monte Carlo Experiment

The expected number of points thrown away in Steps 1 and 2 of CH was estimated empirically by a Monte Carlo experiment where n random points were generated uniformly in the unit square. The results are shown in Table 8.1 for various values of n (every value of n' is an average over 100 runs).

n = total number of points	n' = points thrown away	n'/n
100	88.72	0.8872
200	183.06	0.9153
300	278.97	0.9299
400	376.58	0.9414
500	472.36	0.9447
600	571.44	0.9524
700	668.58	0.9551
800	766.14	0.9576
900	864.67	0.9607
1000	963.60	0.9636

Table 8.1

The average run time in seconds of CH was 0.19 for $n = 1000$. The algorithm of [Ja] required over 4 seconds on the average to perform the same task. Both averages were computed over 100 runs.

8.2 An Algorithm for the Convex Hull in d-dimensions

8.2.1 Basic Idea

Given a convex polyhedron in d-dimensional space. From the simplex method of linear programming we know that for each extreme point of the polyhedron there is an objective function for which that point is an optimal solution. In Figure 8.10, for example, $d = 2$ and only point A maximizes Z .

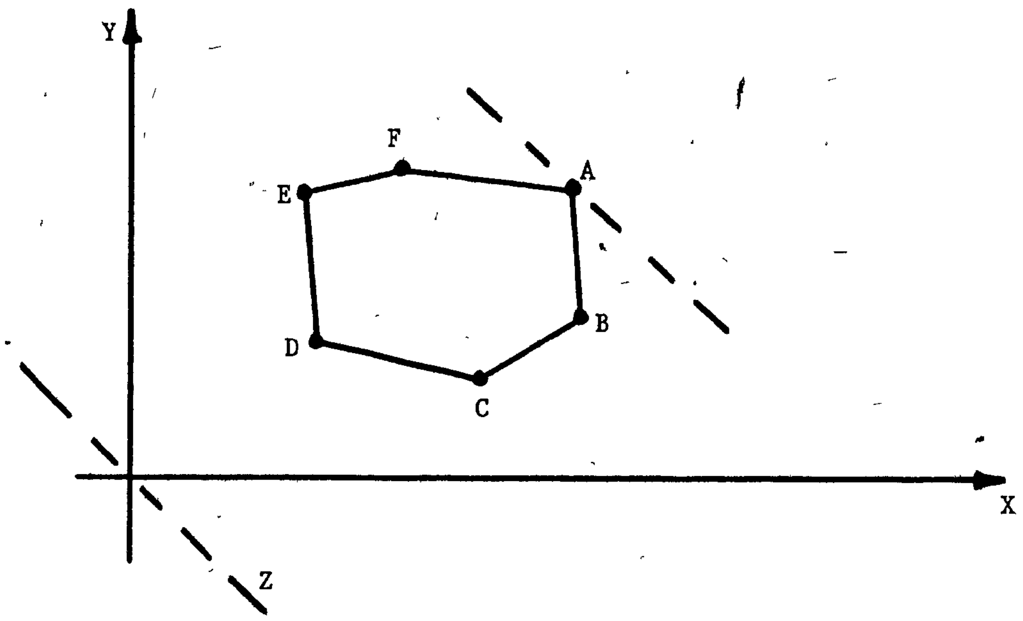


Figure 8.10

The new algorithm we propose in this section relies on the observation that objective functions imply extreme points.

8.2.2 Illustration of basic idea for d=2

Let n points in the plane be given by their cartesian coordinates. It is required to determine the set C of edges forming their CH. We observe that:

(1) The points with maximum and minimum X and Y coordinates are points of the CH.

(2) The points with maximum and minimum (X+Y) and (Y-X) are points of the CH.

(3) No point inside the possibly octagonal polygon formed by the above points is a point of the CH.

(4) A partial CH and up to eight external regions have just been created: in each external region determined by points i and j the *polygonal line* $E_{ij} \subset CH$ is now sought.

The above observations are illustrated in Figure 8.11 .

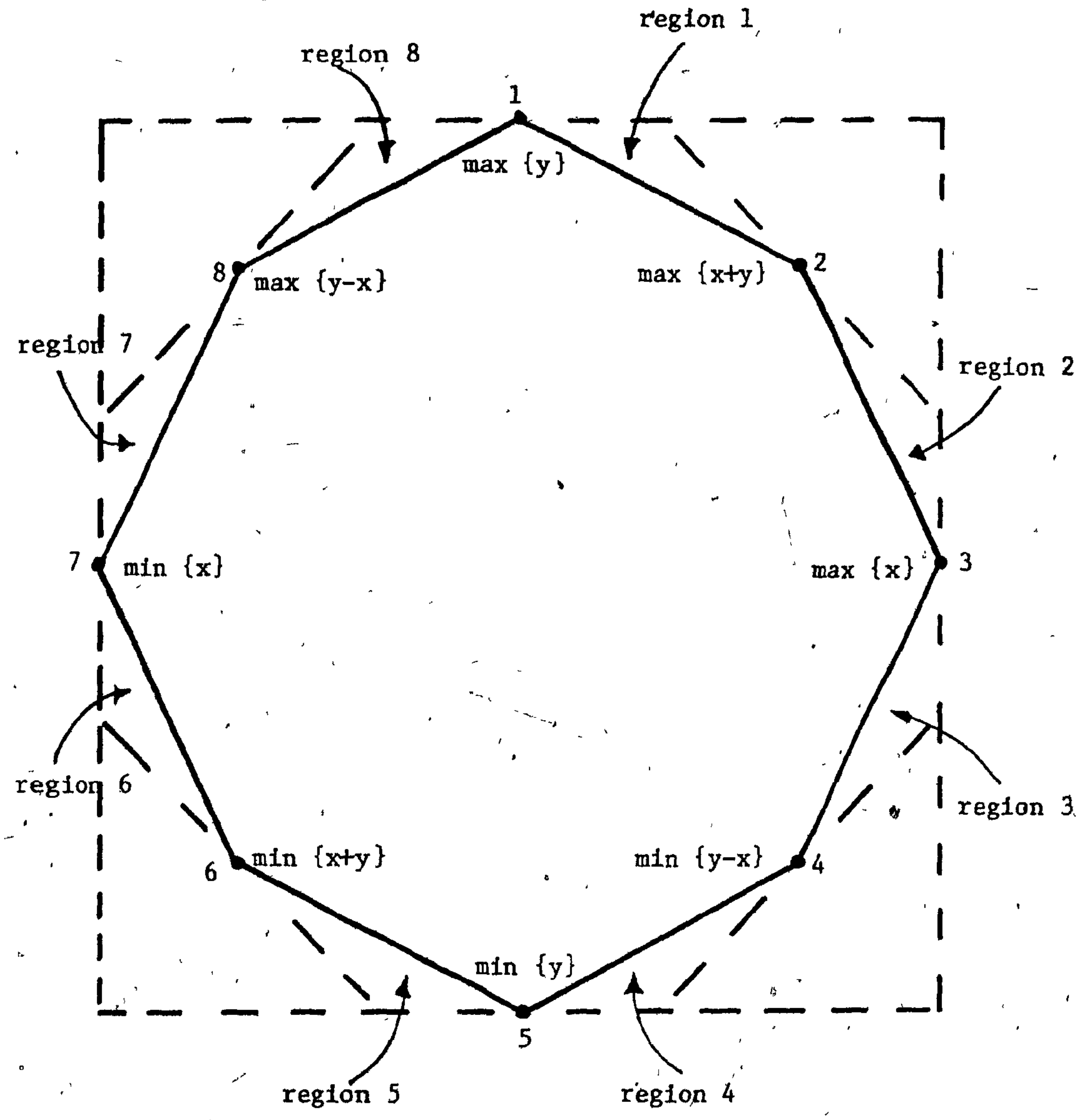

region 1
region 8
1
max {y}
8
max {y-x}
max {x+y}
2
region 7
region 2
7
min {x}
max {x}
3
region 6
region 3
6
min {x+y}
min {y-x}
4
min {y}
5
region 5
region 4

Figure 8.11

Assume a partial CH is known and a set of unexamined points is still left (Figure 8.12) .

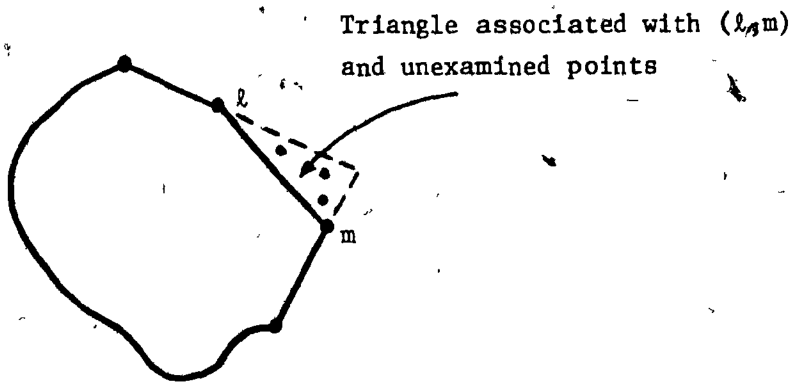


Figure 8.12

A new point can be added to the CH by considering the edge (ℓ,m) , and choosing one of the unexamined points in the triangle associated with (ℓ,m) . Choose the point $p(x_p,y_p)$ whose coordinates maximize the function

$$Z = m_1 x_p |y_\ell - y_m| + m_2 y_p |x_\ell - x_m| \tag{8.17}$$

where

$$\begin{aligned} m_1 &= +1 \text{ for regions } 1, 2, 3, 4 \\ &= -1 \text{ for regions } 5, 6, 7, 8 \end{aligned} ,$$

and

$$\begin{aligned} m_2 &= +1 \text{ for regions } 1, 2, 7, 8 \\ &= -1 \text{ for regions } 3, 4, 5, 6 \end{aligned} .$$

In other words we are choosing p the furthest point in a perpendicular direction from (ℓ,m) . We have

(1) p is a point of the CH.

(2) edges (ℓ,p) and (p,m) replace edge (ℓ,m) in the partial CH.

(3) any point inside the triangle ℓpm need not be considered any more and can thus be "thrown-away".

<u>Algorithm CH2</u>

<u>Step 1:</u> Determine the eight extremal points and remove from further consideration all points falling inside the octagon they form. The remaining points are dis-

tributed among the eight external regions thus created.

Step 2: For every one of the eight external regions determined by the two extremal points i and j we determine E_{ij} as follows:

(a) $E_{ij} = \{(i,j)\}$;

(b) If every point in the region has been either examined or deleted, stop;

(c) For every $(\ell,m) \in E_{ij}$ do the following: if any points are left in the triangle associated with (ℓ,m) , find a point p such that Z is maximized, then

1 - Any point falling inside the triangle ℓpm is deleted, and

2 - E_{ij} is updated as follows

$$E_{ij} = (E_{ij} \cup \{(\ell,p),(p,m)\}) - \{(\ell,m)\} ;$$

(d) Go to (b) ;

Step 3: The set C of edges forming the CH is obtained by merging the E_{ij}'s : $C = E_{12} \cup E_{23} \cup \ldots \cup E_{81}$.

8.2.2.1 Worst-Case Analysis

The computation is dominated by Step 2. In the worst-case the number of comparisons required to determine each of the extreme points is n . If there are n such points, the algorithm will have a complexity of $O(n^2)$ [Avl].

8.2.3 Generalization to d-dimensions - Algorithm CHd

Algorithm CH2 can be easily extended to the general d-dimensional case. Let a point in d-space be given by $(z^1, z^2, \dots, z^d)$. We note that:

1) In Step 1: $(2d+2^d)$ extremal points need to be determined (any point falling inside the polyhedron they form is deleted from further consideration), and 2^d external regions are thus created.

2) In Step 2: a (d-1) hyperplane in d-space is given by d points $(z_1^1, z_1^2, \dots, z_1^d)$, $(z_2^1, z_2^2, \dots, z_2^d), \dots,$

$(z_d^1, z_d^2, \dots, z_d^d)$.

The function to be maximized is

$$Z = \sum_{i=1}^{d} m_i \, z_p^i |\Delta z^i|$$

where $(z_p^1, z_p^2, \dots z_p^d)$ is the new point sought, and

$$\Delta z^i = \begin{vmatrix} (z_2^1-z_1^1) & \dots & (z_2^j-z_1^j) & \dots & (z_2^d-z_1^d) \\ (z_3^1-z_1^1) & \dots & (z_3^j-z_1^j) & \dots & (z_3^d-z_1^d) \\ \vdots & & \vdots & & \vdots \\ (z_d^1-z_1^1) & \dots & (z_d^j-z_1^j) & \dots & (z_d^d-z_1^d) \end{vmatrix} \qquad i \neq j \, .$$

8.3 Some Convex Hull Expectations

In this section we consider the following two problems:

Problem 1: Let S be a convex domain of the plane. The problem associated with the name of Sylvester [Ke, Ki] is to find the probability that four points taken at random inside S form a convex quadrilateral. This problem has been solved in [Del] for various convex domains. The generalization we propose to examine statistically is the following: find the probability p(n) that n points taken at random inside S form a convex n-gon when $n \geq 4$.

Problem 2: Let B be a set of points in the plane (d-space). The convex hull of B is the smallest convex polygon (polyhedron) containing all the points of the set. Integral expressions were given in [Ef, Ren] for the expected number of vertices $E(V_n)$, area $E(A_n)$ and perimeter $E(P_n)$ of the convex hull of n independent and identically distributed random points in two and three dimensions, for n infinite or fixed. We provide a stochastic evaluation of these expectations for the 2-dimensional case.

The two problems are related in an obvious way: by determining the convex hull and counting the number of its vertices, the question of whether the points form a convex n-gon or not can readily be answered.

A Monte Carlo Experiment

In this experiment points were uniformly generated in a 10x10 square and their convex hull identified using algorithm CH . When all n points were included in the convex hull this was counted as a success, thus

$$p'(n) = \frac{\text{number of successes}}{\text{number of trials}} .$$

Also, the number of vertices, area and perimeter of the convex hull were computed, averaged over all trials and rounded to the higher integer. Tables 8.2, 8.3 and 8.4 show the results obtained, the number of trials being 10,000 for $n \leq 10$, and 1000 for $n > 10$.

n	p'(n)
4	0.6867
5	0.3415
6	0.1242
7	0.0335
8	0.0077
9	0.0011
10	0.0006

Table 8.2

n \ V_n	3	4	5	6	7	8	9	10	Average	Standard deviation
4	3133	6867			Every entry represents the number of times - out of 10000 trials - a graph with n nodes has a convex hull of V_n vertices				3.686	0.463
5	1019	5566	3415						4.239	0.621
6	383	3648	4727	1242					4.682	0.735
7	162	2248	4692	2563	335				5.066	0.821
8	67	1319	4103	3448	986	77			5.419	0.889
9	18	902	3397	3799	1633	240	11		5.689	0.942
10	7	557	2684	3940	2219	545	42	6	5.964	0.991

Table 8.3(a)

n \ V_n	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	Aver.	St.dev.
20		3	34	136	260	294	165	83	24	1									7.761	1.352
30			6	37	148	228	255	190	91	32	10	3							8.863	1.519
40			2	14	65	187	214	248	159	77	26	8							9.573	1.571
50			2	10	53	102	218	224	174	121	72	16	8						10.105	1.750
60			1	6	15	73	151	218	205	188	93	38	10	1	1				10.704	1.714
70				7	22	47	112	235	210	183	106	47	26	4	1				10.943	1.792
80					6	32	95	189	183	210	140	87	38	18	2				11.506	1.838
90					8	31	79	151	220	203	157	82	46	16	5	1	1		11.638	1.861
100				1	6	23	54	143	166	209	180	114	66	23	11	2	1	1	12.038	1.958

Every entry represents the number of times - out of 1000 trials - a graph with n nodes has a convex hull of V_n vertices.

Table 8.3(b)

n \ CH	$E(A_n)$	$E(P_n)$
3	8	15
4	15	19
5	22	21
6	27	23
7	32	24
8	37	25
9	40	26
10	43	27
20	62	31
30	72	32
40	77	33
50	81	34
60	82	34
70	84	35
80	86	35
90	87	35
100	88	36

Table 8.4

Note that it was shown in [Del] that $p(4) = 25/36$ when the convex domain is a parallelogram. In our case the domain is a square of area 100.
Since $p(4) = 1 - 4 \frac{E(A_3)}{S}$ it follows that $\lceil E(A_3) \rceil = 8$ which agrees with the experimental value in Table 8.4 .

8.4 Conclusion

An algorithm CH was described for obtaining the convex hull of a set of points in the plane. The algorithm has a worst-case complexity of O(n log n). We proved that the asymptotic expected run-time of CH is linearly proportional to the size of the input when the data points are uniformly distributed in the unit square. It is possible to show that the algorithm has this same asymptotic expected run-time behavior in Gaussian environments.

We also presented a second algorithm CH2 that does not require sorting and is based on the idea of continually throwing away points. A few suggestions were made on how to generalize CH2 to solve the convex hull problem in d dimensions.

Finally, some expectations related to the convex hull of points uniformly distributed in the plane were stochastically estimated.

CONCLUSIONS

In the following we make some general conclusions about the contents of this thesis.

1) In Chapter 2, it was empirically shown for the first time how small the set of feasible solutions to the Euclidean Traveling Salesman Problem could be when one uses some simple geometrical properties to rule out 'bad' tours.

2) Experiments with two new tools for obtaining approximate solutions to the TSP were described in Chapters 3 and 4: triangulations of points in the plane and a reward-punishment method. It is hoped that these techniques will still be refined to yield better answers to the problem and that they will find applications in other combinatorial optimization problems.

3) Powerful heuristics recently proposed for the symmetric TSP were combined in Chapter 5 to yield a very efficient approximation algorithm. The algorithm is easy to program and yields a high quality near optimal solution in a short amount of time. An additional advantage of the algorithm is that it lends itself to be extended for the asymmetric case.

4) In Chapter 6, we concluded from our experience with an algorithm for the TSP that uses local neighborhood search that starting with biased tours is preferable to starting with random ones when computation time is of primary importance.

5) Our study of maximal triangulations in Chapter 7 led to the derivation of a new lower bound on the maximum number of crossing-free Hamilton cycles in a rectilinear drawing of a complete graph.

6) As stated earlier, all published convex hull algorithms have an expected running time bounded below by $O(n \log n)$. This is probably because most algorithms consider all points as possible candidates for

corner points. We observe, however, that when humans are asked to find the convex hull of a set of points they disregard from consideration, presumably using pattern recognition ability, those points falling in the 'center' of the set. Thus one might expect an 'intelligent' algorithm to do the same. In Chapter 8 the heuristic used for discarding points in Steps 1 and 2 of algorithm CH approximates this pattern recognition ability of the human ~~and results~~ in an expected running time of $O(n)$.

References

[Ac1] Ackoff, R.L., Ed., Progress in Operations Research, Vol. 1, Wiley, New York, 1961.

[Ac2] Ackoff, R.L. and Sasieni, M.W., Fundamentals of Operations Research, Wiley, New York, 1968.

[Ad] Adrabinski, A., A Heuristic Algorithm for the Traveling-Salesman Problem, Zastosowania Matematyki, XV, 2, 1976, pp. 223-243.

[Ag] Agin, N., Optimum Seeking with Branch and Bound, Management Science, Vol. 13, No. 4, 1966, pp. B176-B185.

[Aho] Aho, A.V., Hopcroft, J.E. and Ullman, J.D., The Design and Analysis of Computer Algorithms, Addison Wesley, Reading, Massachusetts, 1976.

[Ak1] Akl, S.G. and Toussaint, G.T., A Fast Convex Hull Algorithm, Information Processing Letters, to appear.

[Ak2] Akl, S.G. and Toussaint, G.T., Efficient Convex Hull Algorithms for Pattern Recognition Applications, submitted for publication.

[Ap] Appel, A. and Will, P.M., Determining the Three Dimensional Convex Hull of a Polyhedron, IBM Journal of Research and Development, November 1976, pp. 590-601.

[As] Ashour, S., Vega, J.F., and Parker, R.G., A Heuristic Algorithm for Traveling Salesman Problems, Transportation Research, 6, 1972, pp. 187-195.

[Av1] Avis, D., personal communication.

[Av2] Avis, D., Two Greedy Heuristics for the Weighted Matching Problem, presented at the Ninth Southeastern Conference on Combinatorics, Graph Theory and Computing, Florida Atlantic University, Boca Raton, Florida, 1978.

[B] Babuska, I. and Aziz, A.K., On the Angle Condition in the Finite Element Method, SIAM Journal on Numerical Analysis, Vol. 13, No. 2, 1976, pp. 214-226.

[Ba] Barachet, L.L., Graphic Solution of the Traveling-Salesman Problem, Operations Research, 5, 1957, pp. 841-845.

[Be] Beardwood, J., Halton, J.H. and Hammersley, J.M., The Shortest Path Through Many Points, Proceedings of the Cambridge Philosophical Society, 55, 1959, pp. 299-327.

[Bea1] Bellman, R., Dynamic Programming Treatment of the Traveling Salesman Problem, Journal of the ACM, 9, 1962, pp. 61-63.

[Bea2] Bellman, R., Cooke, K.L. and Lockett, J.A., Algorithms, Graphs, and Computers, Academic Press, New York, 1970.

[Beo1] Bellmore, M. and Nemhauser, G.L., The Traveling Salesman Problem : A Survey, Operations Research, 16, 1968, pp. 538-558.

[Beo2] Bellmore, M. and Malone, J.C., Pathology of Traveling Salesman Subtour-Elimination Algorithms, Operations Research, 19, 1971, pp. 278-307.

[Berg] Berge, C., Graphs and Hypergraphs, North-Holland, London, 1973.

[Berm] Berman, G. and Fryer, K.D., Introduction to Combinatorics, Academic Press, New York, 1972.

[Boc] Bock, F., An Algorithm for Solving "Traveling-Salesman" and Related Network Optimization Problems, Operations Research, 6, 1958, p. 897.

[Bof] Boffey, T.B., A Note on Minimal Length Hamilton Path and Circuit Algorithms, Operational Research Quarterly, 24, 1973, pp. 437-439.

[Bou] Bourgeois, F. and Lasalle, J.C., An Extension of the Munkres Algorithm for the Assignment Problem to Rectangular Matrices, Communications of the ACM, 14, 1971, pp. 802-806.

[Bra] Bramble, J.H. and Zlamal, M., Triangular Elements in the Finite Element Method, Mathematics of Computation, 24, 1970, pp. 809-820.

[Bre] Breuer, M.A., Ed., Design Automation of Digital Systems, Vol. 1, Prentice Hall, Englewood Cliffs, New Jersey, 1972.

[Bu] Burton, W., Representation of Many-Sided Polygons and Polygonal Lines for Rapid Processing, Communications of the ACM, 20, 1977, pp. 166-171.

[Cha] Chand, D.R. and Kapur, S.S., An Algorithm for Convex Polytopes, Journal of the ACM, 17, 1970, pp. 78-86.

[Chr] Christofides, N. and Eilon, S., An Algorithm for the Vehicle-Dispatching Problem, Operational Research Quarterly, 20, 1969, pp. 309-318.

[Chr2] Christofides, N., The Shortest Hamiltonian Chain of a Graph, SIAM Journal on Applied Mathematics, Vol. 19, No. 4, 1970, pp. 689-696.

[Chr3] Christofides, N. and Eilon, S., Algorithms for Large-Scale Travelling Salesman Problems, Operational Research Quarterly, 23, 1972, pp. 511-518.

[Chr4] Christofides, N., Bounds for the Travelling-Salesman Problem, Operations Research, 20, 1972, pp. 1044-1056.

[Chr5] Christofides, N., Graph Theory: An Algorithmic Approach, Academic Press, New York, 1975.

[Chr6] Christofides, N., Worst-Case Analysis of a New Heuristic for the Travelling Salesman Problem, Management Sciences Research Report No. 388, Carnegie Mellon University, February 1976.

[Cof] Coffman, E.G., Ed., Computer and Job-Shop Scheduling Theory, Wiley, New York, 1976.

[Con] Conway, R.W., Maxwell, W.L. and Miller, L.W., Theory of Scheduling, Addison Wesley, Reading, Massachusetts, 1967.

[Cr] Croes, G.A., A Method for Solving Traveling-Salesman Problems, Operations Research, 6, 1958, pp. 791-812.

[Dac] Dacey, M.F., Selection of an Initial Solution for the Traveling-Salesman Problem, Operations Research, 8, 1960, pp. 133-134.

[Dan1] Dantzig, G., Fulkerson, R. and Johnson, S., Solution of a Large-Scale Traveling Salesman Problem, Operations Research, 2, 1954, pp. 393-410.

[Dan2] Dantzig, G., Fulkerson, R. and Johnson, S., On a Linear Programming Combinatorial Approach to the Traveling-Salesman Problem, Operations Research, 7, 1959, pp. 58-66.

[Dav] Davis, J.C. and McCullagh, M.J., Eds., Display and Analysis of Spatial Data, Wiley, New York, 1975, p. 378.

[Del] Deltheil, R., Probabilities Geometriques, Traite du Calcul des Probabilities et de ses Applications, Vol. 2, No. 2, 1926.

[Deo] Deo, M. and Hakimi, S.L., The Shortest Generalized Hamiltonian Tree, Proceedings of the Third Annual Allerton Conference on Circuit and Systems Theory, 1965, pp. 879-888.

[Dil] Dijkstra, E.W., A Note on Two Problems in Connexion with Graphs, Numerische Mathematik, 1, 1959, pp. 269-271.

[Di2] Dijkstra, E.W., A Discipline of Programming, Prentice Hall, Englewood Cliffs, New Jersey, 1976.

[Du] Düppe, R.D. and Gottschalk, H.J., Automatische Interpolation von Isolinien bei willkürlich Stützpunkten, Allgemeine Vermessungsnachrichten, 77, 1970, pp. 423-426.

[Ea] Eastman, W.L., Linear Programming with Pattern Constraints, Ph.D. Dissertation, Harvard, 1958.

[Ed] Eddy, W.F., A New Convex Hull Algorithm for Planar Sets, ACM Transactions on Mathematical Software, Vol. 3, No. 4, pp. 398-403; pp. 411-412.

[Ef] Efron, B., The Convex Hull of a Random Set of Points, Biometrika, 52, 1965, pp. 331-343.

[Eg] Eggleton, R.B., Crossing Numbers of Graphs, Ph.D. Thesis, University of Calgary, 1973.

[Ei] Eilon, S., Watson-Gandy, C.D.T. and Christofides, N., Distribution Management, Griffin, London, 1971.

[Er] Erdös, P. and Guy, R.K., Crossing Number Problems, American Mathematical Monthly, Vol. 80, No. 1, 1973, pp. 52-58.

[Fe] Few, L., The Shortest Path and the Shortest Road Through n Points, Mathematika, 2, 1955, pp. 141-144.

[Fl] Flood, M.M., The Traveling Salesman Problem, Operations Research, 4, 1956, pp. 61-75.

[Fr] Freeman, H. and Shapira, R., Determining the Minimum-Area Encasing Rectangle for an Arbitrary Closed Curve, Communications of the ACM, 18, 1965, pp. 409-413.

[Gab] Gabow, H., Implementation of Algorithms for Maximum Matching on Nonbipartite Graphs, Ph.D. Thesis, Stanford University, 1973.

[Gae1] Garey, M.R., Johnson, D.S. and Tarjan, R.E., The Planar Hamiltonian Circuit Problem Is NP-Complete, SIAM Journal of Computing, Vol. 5, No. 4, pp. 704-714.

[Gae2] Garey, M.R., Graham, R.L. and Johnson, D.S., Some NP-Complete Geometric Problems, Proceedings of the Eighth ACM Symposium on Theory of Computing, 1976, pp. 10-22.

[Gar] Garfinkel, R.S., On Partitioning the Feasible Set in a Branch and Bound Algorithm for the Asymmetric Traveling-Salesman Problem, Operations Research, 21, 1973, pp. 340-343.

[Gav] Gavett, J.W., Three Heuristic Rules for Sequencing Jobs to a Single Production Facility, Management Science, Vol. 11, No. 8, 1965, pp. B166-B176.

[Gib1] Gilbert, E.N., Random Minimal Trees, Journal of SIAM, Vol. 13, No. 2, 1965, pp. 376-387.

[Gib2] Gilbert, E.N. and Pollack, H.O., Steiner Minimal Trees, SIAM Journal on Applied Mathematics, Vol. 16, No. 1, 1968, pp. 1-29.

[Gim] Gilmore, P.C. and Gomory, R.E., Sequencing a One State-Variable Machine : A Solvable Case of the Traveling Salesman Problem, Operations Research, 12, 1964, pp. 655-679.

[Go] Golden, B.L., A Statistical Approach to the Traveling Salesman Problem, Networks, 7, 1977, pp. 209-225.

[Gon] Gonzales, R.H., Solution to the Traveling Salesman Problem by Dynamic Programming on the Hypercube, Tech. Rep. No. 18, O.R. Center, M.I.T., 1962.

[Goo] Goodman, S.E. and Hedetniemi, S.T., Introduction to the Design and Analysis of Algorithms, McGraw Hill, New York, 1977.

[Gr] Graham, R.L., An Efficient Algorithm for Determining the Convex Hull of a Finite Planar Set, Information Processing Letters, 1, 1972, pp. 132-133.

[Gru] Grünbaum, B., Convex Polytopes, Wiley, New York, 1967.

[Gu1] Guy, R.K., Latest Results on Crossing Numbers, Recent Trends in Graph Theory, Springer-Verlag, New York, 1971, pp. 143-156.

[Gu2] Guy, R.K., Crossing Numbers of Graphs, Graph Theory and its Applications, Springer-Verlag, Berlin, 1972, pp. 111-124.

[Ham] Hammersley, J.M. and Handscomb, D.C., Monte Carlo Methods, Methuen, London, 1964.

[Han] Hansen, K.H. and Krarup, J., Improvements of the Held-Karp Algorithm for the Symmetric Traveling Salesman Problem, Mathematical Programming, 7, 1964, pp. 87-96.

[Har1] Harary, F., Graph Theory, Addison Wesley, Reading, Massachusetts, 1972.

[Har2] Harary, F., Ed., Proof Techniques in Graph Theory, Academic Press, New York, 1969.

[Har3] Harary, F. and Hill, A., On the Number of Crossings in a Complete Graph, Proceedings of the Edinburgh Mathematical Society, 2, 13, 1962-63, pp. 333-338.

[Hav] Hardgrave, W.W. and Nemhauser, G.L., On the Relation Between the Traveling-Salesman and the Longest-Path Problems, Operations Research, 10, 1962, pp. 647-657.

[He1] Held, M. and Karp, R.M., A Dynamic Programming Approach to Sequencing Problems, Journal of SIAM, Vol. 10, No. 1, 1962, pp. 196-210.

[He2] Held, M. and Karp, R.M., The Traveling-Salesman Problem and Minimum Spanning Trees, Operations Research, 18, 1970, pp. 1138-1162.

[He3] Held, M. and Karp, R.M., The Traveling-Salesman Problem and Minimum Spanning Trees : Part II, Mathematical Programming, 1, 1971, pp. 6-25.

[Hel] Heller, I., On the Problem of Shortest Path Between Points, I and II, Problems 664t and 665t, Bulletin of the American Mathematical Society, 59, 1953, pp. 551-552.

[Hen] Henley, E. and Williams, R., Graph Theory in Modern Engineering, Academic Press, New York, 1973.

[I] Isaac, A.M. and Turban, E., Some Comments on the Traveling Salesman Problem, Operations Research, 17, 1969, pp. 543-546.

[Ja] Jarvis, R.A., On the Identification of the Convex Hull of a Finite Set of Points in the Plane, Information Processing Letters, 2, 1973, pp. 18-21.

[Je] Jensen, H.F., An Upper Bound for the Rectilinear Crossing Number of the Complete Graph, Journal of Combinatorial Theory, 11, 1971, pp. 212-216.

[Jo] Jones, L., A Note on the Traveling Salesman Problem, SIAM Journal on Applied Mathematics, Vol. 32, No. 1, 1977, pp. 220-222.

[Ka] Karg, R.L. and Thompson, G.L., A Heuristic Approach to Solving Traveling Salesman Problems, Management Science, Vol. 10, No. 2, 1964, pp. 225-248.

[Kar1] Karp, R.M., On the Computational Complexity of Combinatorial Problems, Networks, 5, 1975, pp. 45-68.

[Kar2] Karp, R.M., Reducibility Among Combinatorial Problems, in Complexity of Computer Computations, Miller, R.E. and Thatcher, J.W., Eds., Plenum Press, New York, 1972, pp. 85-104.

[Kar3] Karp, R.M., The Probabilistic Analysis of Some Combinatorial Search Algorithms, in Algorithms and Complexity: New Directions And Recent Results, Traub, J.F., Ed., Academic Press, New York, 1976, pp. 1-19.

[Ke] Kendall, M.G. and Moran, P.A.P., Geometrical Probability, Griffin, London, 1963.

[Ki] Kingman, J.F.C., Random Secants of a Convex Body, Journal of Applied Probability, 6, 1969, pp. 660-672.

[Ko] Kohn, S., Gottlieb, A. and Kohn, M., A Generating Function Approach to the Traveling Salesman Problem, Proceedings of the ACM Annual Conference, 1977, pp. 294-300.

[Kro] Krolak, P., Feits, W. and Marble, G., A Man-Machine Approach Toward Solving the Traveling-Salesman Problem, Communications of the ACM, 14, 1971, pp. 327-334.

[Kru] Kruskal, J.B., On the Shortest Spanning Subtree of a Graph and the Traveling Salesman Problem, Proceedings of the American Mathematical Society, 7, 1956, pp. 48-50.

[Ku] Kung, H.T., Luccio, F. and Preparata, F.P., On Finding the Maxima of a Set of Vectors, Journal of the ACM, 22, 1975, pp. 469-476.

[La1] Lawler, E.L. and Wood, D.E., Branch-and-Bound : A Survey, Operations Research 14, 1966, pp. 699-719.

[La2] Lawler, E.L., Combinatorial Optimization : Networks and Matroids, Holt, Rinehart and Winston, New York, 1976.

[Le1] Lenstra, J.K., Clustering a Data Array and the Traveling-Salesman Problem, Operations Research, 22, 1974, pp. 413-414.

[Le2] Lenstra, J.K. and Rinnooy Kan, A.H.G., Some Simple Applications of the Travelling Salesman Problem, Operational Research Quarterly, 26, 1975, pp. 717-733.

[Lew] Lewis, H.R. and Papadimitrion, C.H., The Efficiency of Algorithms, Scientific American, January 1978, pp. 96-109.

[Lin1] Lin, S., Computer Solutions of the Traveling Salesman Problem, Bell System Technical Journal, Vol. 44, 1965, pp. 2245-2269.

[Lin2] Lin, S., Heuristic Techniques for Solving Large Combinatorial Problems on a Computer, Theoretical Approaches to Non-Numerical Problem Solving, Proceedings of the Fourth Symposium, 1970, pp. 410-418.

[Lin3] Lin, S. and Kernighan, B.W., An Effective Heuristic Algorithm for the Traveling Salesman Problem, Operations Research, 21, 1973, pp. 498-516.

[Lin4] Lin, S., Heuristic Programming as an Aid to Network Design, Networks, 5, 1975, pp. 33-43.

[Lit] Little, J.D.C., Murty, K.G., Sweeney, D.W. and Karel, C., An Algorithm for the Traveling Salesman Problem, Operations Research, 11, 1963, pp. 972-989.

[Liu] Liu, C.L., Introduction to Combinatorial Mathematics, McGraw Hill, New York, 1968.

[Ll] Lloyd, E.L., On Triangulations of a Set of Points in the Plane, M.Sc. Thesis, MIT, 1977.

[Ma] Marks, E.S., A Lower Bound for the Expected Travel Among m Random Points, Annals of Mathematical Statistics, 19, 1948, pp. 419-422.

[Mas] Marsaglia, G., Algorithm for Random Permutation, personal communication, 1976.

[Mic] Michie, D., Ed., Machine Intelligence, Vol. 3, American Elsevier, New York, 1968.

[Mil] Miller, C.E., Integer Programming Formulation of Traveling Salesman Problems, Journal of the ACM, 7, 1960, 326-329.

[Mn] Moon, J.W. and Moser, L., Simple Paths On Polyhedra, Pacific Journal of Mathematics, Vol. XIII, 1963, pp. 629-631.

[Mo] Morton, G. and Land, A.H., A Contribution to the "Travelling-Salesman" Problem, Journal of the Royal Statistical Society(B), 17, 1955, pp. 185-194.

[Mu] Murty, K.G., On the Tours of a Traveling Salesman, SIAM Journal on Control, Vol. 7, No. 1, 1969, pp. 122-131.

[Net] Netter, J.P., An Algorithm to Find Elementary Negative Cost Circuits with a Given Number of Arcs - The Traveling Salesman Problem, Operations Research, 19, 1971, pp. 234-237.

[New] Newborn, M. and Moser, W.O.J., On Optimal Crossing-Free Hamiltonian Circuit Drawings of K_n and Related Problems, submitted for publication.

[O] Obruca, A.K., Spanning Tree Manipulation and the Traveling Salesman Problem, The Computer Journal, 10, 1968, pp. 374-377.

[Pan] Pandit, S.N.N., Some Observations on the Longest Path Problem, Operations Research, 12, 1964, pp. 361-364.

[Pap] Papadimitriou, C.H. and Steiglitz, K., On the Complexity of Local Search for the Traveling Salesman Problem, SIAM Journal on Computing, Vol, 6, No. 1, 1977, pp. 76-83.

[Papa] Papadimitriou, C.H., The Euclidean Traveling Salesman Problem Is NP-Complete, Theoretical Computer Science, 4, 1977, pp. 237-244.

[Pi] Pierce, A.R., Bibliography on Algorithms for Shortest Path, Shortest Spanning Tree, and Related Circuit Routing Problems, Networks, 5, 1975, pp. 129-149.

[Pre] Preparata, F.P. and Hong, S.J., Convex Hulls of Finite Sets of Points in Two and Three Dimensions, Communications of the ACM, 20, 1977, pp. 87-93.

[Pri] Prim, R.C., Shortest Connection Networks and Some Generalizations, Bell System Technical Journal, 36, 1957, 1389-1401.

[Rao] Rao, M.R., Adjencency of the Traveling Salesman Tours and 0-1 Vertices, SIAM Journal on Applied Mathematics, Vol. 30, No. 2, 1976, pp. 191-198.

[Ray] Raymond, T.C., Heuristic Algorithm for the Traveling-Salesman Problem, IBM Journal of Research and Development, July 1969, pp. 400-407.

[Reg] Reingold, E.M., Nievergelt, J. and Deo, N., Combinatorial Algorithms Theory and Practice, Prentice Hall, Englewood Cliffs, New Jersey, 1977.

[Rei] Reiter, S. and Sherman, G., Discrete Optimizing, Journal of SIAM, Vol, 13, No. 3, 1965, pp. 864-889.

[Ren] Rényi, A. and Sulanke, R., Über die konvexe Hülle von n zufällig gewählten Punkten, I and II, Zeitschrift für Wahrscheinlichkeitstheorie 2, 1963, pp. 75-84; 3, 1964, pp. 138-147.

[Ri] Riley, W., Micro-Analysis Applied to the Traveling Salesman Problem, Bulletin of Operations Research, 1960, B-12.

[Robl] Roberts, S.M. and Flores, B., Systematic Generation of Hamiltonian Circuits, Communications of the ACM, 9, 1966, pp. 690-694. (See also Communications of the ACM, 10, 1967, pp. 201-202).

[Rob2] Roberts, S.M. and Flores, B., An Engineering Approach to the Traveling Salesman Problem, Management Science, Vol. 13, No. 3, 1966, pp. 269-288.

[Ros1] Rosenkrantz, D.J., Stearns, R.E. and Lewis, P.M., Approximate Algorithms for the Traveling Salesperson Problem, Proceedings of the IEEE Fifteenth Symposium on Switching and Automata Theory, 1974, pp. 33-42.

[Ros2] Rosenkrantz, D.J., Stearns, R.E. and Lewis, P.M., An Analysis of Several Heuristics for the Traveling Salesman Problem, SIAM Journal on Computing, Vol. 6, No. 3, 1977, pp. 563-581.

[Ross] Rossman, M.J. and Twery, R.J., A Solution to the Travelling Salesman Problem by Combinatorial Programming, Operations Research, 6, 1958, p. 897.

[Rot] Rothkopf, M., The Traveling Salesman Problem : On the Reduction of Certain Large Problems to Smaller Ones, Operations Research, 14, 1966, pp. 532-533.

[Roy] Roy, B., Ed., Combinatorial Programming : Methods and Applications, Reidel, Dordrecht, Holland, 1975.

[Sah] Sahni, S. and Gonzales, T., P-Complete Approximation Problems, Journal of the ACM, Vol. 23, No. 3, pp. 555-565.

[Sak] Saksena, J.P. and Kumar, S., The Routing Problem with 'K' Specified Nodes, Operations Research, 14, 1966, pp. 909-913.

[Sas] Sasieni, M., Yaspan, A. and Friedman, L., Operations Research, Wiley, New York, 1959.

[Sav1] Savage, S., Statistical Indicators of Optimality, Proceedings of the IEEE Fourteenth Symposium on Switching and Automata Theory, 1973, pp. 85-91.

[Sav2] Savage, S., Weiner, P. and Bagghi, A., Neighborhood Search Algorithms for Guaranteeing Optimal Traveling Salesman Tours Must Be Inefficient, Journal of Computer and System Sciences, 12, 1976, pp. 25-35.

[Sav3] Savage, J.E., The Complexity of Computing, Wiley, New York, 1976.

[Sd] Sedgewick, R., Permutation Generation Methods, Computing Surveys, Vol. 9, No. 2, June 1977, pp. 137-164.

[Se] Selby, G.R., The Use of Topological Methods in Computer-Aided Circuit Layout, Ph.D. Thesis, London University, 1970.

[Sh1] Shamos, M.I., Geometric Complexity, Proceedings of the Seventh Annual ACM Symposium on the Theory of Computing, 1975, pp. 224-233.

[Sh2] Shamos, M.I. and Hoey, D., Closest Point Problems, Proceedings of the Sixteenth Annual IEEE Symposium on Foundations of Computer Science, 1975, pp. 151-162.

[Si] Shapiro, D., Algorithms for the Solution of the Optimal Cost Travelling Salesman Problem, Sc.D. Thesis, Washington University, 1966.

[Sk1] Sklansky, J., Measuring Concavity on a Rectangular Mosaic, IEEE Transactions on Computers, Vol. C-21, No. 12, 1972, pp. 1355-1364.

[Sku] Skupien, Z., Locally Hamiltonian Graphs and Kuratowski Theorem, Bulletin de L'Académie Polonaise des Sciences, Série des Sciences Mathématiques, Astronomiques et Physiques, 13, 1965, pp. 615-619.

[Sm] Smith, O.D., Minimal Path Connecting n Points, Problem E880, American Mathematical Monthly, 57, 1950, p. 261.

[Stc] Steckhan, H., A Theorem on Symmetric Traveling Salesman Problems, Operations Research, 18, 1970, pp. 1163-1167.

[Ste1] Steiglitz, [illegible] and Weiner, P., Some Improved Algorithms for Computer Solution of the Traveling Salesman Problem, Proceedings of the Sixth Annual Allerton Conference on Circuit and Systems Theory, 1968, pp. 814-821.

[Ste2] Steiglitz, K., Weiner, P. and Kleitman, D.J., The Design of Minimum-Cost Survivable Networks, IEEE Transactions on Circuit Theory, Vol. CT-16, No. 4, 1969, pp. 455-460.

[T] Toussaint, G.T. and Akl, S.G., Convex Hull Algorithms in Two and More Dimensions, submitted for publication.

[V] Verblunsky, S., On the Shortest Path Through a Number of Points, Proceedings of the American Mathematical Society, 2, 1951, pp. 904-913.

[Wa] Wagner, H.M., Principles of Operations Research, Prentice Hall, Englewood Cliffs, New Jersey, 1969.

[Web] Webb, M.H.J., Some Methods of Producing Approximate Solutions to Travelling Salesman Problems with Hundreds or Thousands of Cities, Operational Research Quarterly, 22, 1971, pp. 49-66.

[Wed] Weide, B., A Survey of Analysis Techniques for Discrete Algorithms, Computing Surveys, Vol. 9, No. 4, pp. 291-313.

[Wei] Weiner, P., Heuristics, Networks, 5, 1975, pp. 101-103.

[Wel] Wells, M.B., Elements of Combinatorial Computing, Pergamon Press, New York, 1971.

[Wh] Whitney, H., A Theorem on Graphs, Annals of Mathematics, Ser. 2, Vol. 32, 1931, pp. 378-390.