Fall 1989

# The directed Steiner problem on graphs: A simulated annealing approach

Lawrence Joseph Osborne

# THE DIRECTED STEINER PROBLEM ON GRAPHS: A SIMULATED ANNEALING APPROACH

BY

LAWRENCE JOSEPH OSBORNE, 1946-

A DISSERTATION

Presented to the Faculty of the Graduate School of the

UNIVERSITY OF MISSOURI - ROLLA

In Partial Fulfillment of the Requirements for the Degree

DOCTOR OF PHILOSOPHY

in

COMPUTER SCIENCE

1989

Billy E Gillett

Billy E. Gillett, Advisor

A K Rigler

A. K. Rigler

[illegible]

V. A. Samaranayake

[illegible]

Ralph W. Wilkerson

[illegible]

George W. Zobrist

## ABSTRACT

The well-known Steiner Problem on Graphs is an NP-complete problem for which there are many heuristic and exact algorithms that are deterministic. In this dissertation a new approach to the directed version of this problem is made by applying the ideas of statistical mechanics through the use of the method of simulated annealing. A version of annealing is developed for the Directed Steiner Problem and compared with one of the best general annealing schemes. Then a comparison is made between simulated annealing and the traditional branch and bound technique. The dual ascent algorithm of Richard T. Wong is used to obtain lower bounds for the branch and bound scheme. It appears that for random graphs with more than approximately 60 vertices the new method is on the average superior in finding near-optimum answers quickly. In fact, for large values of N, where N is the number of vertices, it is possible to obtain answers within a few percent of the optimum in polynomial time.

## ACKNOWLEDGEMENT

Many people have helped make this accomplishment possible. Numerous teachers throughout my undergraduate and graduate career have advised and encouraged me. The departments of Mathematics and Computer Science at UMR provided me with graduate teaching assistantships without which I would have been unable to continue.

My thanks go to Southwest Missouri State University which generously granted me two years leave of absence and a half of a year's salary to finish this work. Bruno Schmidt, Head of the Department of Computer Science at SMSU, has always patiently accomodated my schedule and guided me through administrative procedures.

Without Dr. Bill Gillett, my advisor, I would never have persevered through the times when my research did not seem to be productive. He focused my attention on areas where progress could be made, and he saw results when I saw only obstacles.

I would also like to thank Dr. V. A. Samaranayake in the Department of Statistics for the time he has taken for numerous discussions concerning this research. I am indebted to all the members of my committee for their time and efforts on my behalf.

Finally, I would like to thank my parents. Mr. and Mrs. L. B. Osborne, who have supported and inspired me throughout my education.

# TABLE OF CONTENTS

Page

# LIST OF ILLUSTRATIONS

# LIST OF TABLES

# I. INTRODUCTION

The Steiner problem in its most general form asks for the shortest network of arcs that will interconnect a set of given points in space. It cannot be solved by simply placing arcs between the various points. To solve the problem requires adding new points, called Steiner points, that act as intermediate points between the required nodes in a shortest network. The problem became popular in 1941 when Richard Courant and Herbert E. Robbins [23] included it in their book What is Mathematics. Since that time many mathematicians and computer scientists [13, 18, 22, 24, 44, 75, 107] have studied this problem, but an efficient exact algorithm for random graphs with more than approximately 30 vertices has not been found. Moreover, since it has been proven that the problem is within the class of NP-hard problems [40], it is unlikely that a polynomial time algorithm exists for this problem. Hence, in recent years researchers have concentrated on developing heuristic algorithms that provide a good, but not optimal, solution in polynomial time. [96].

The Steiner problem on graphs, both directed and undirected, was introduced by Hakimi [48] in 1971. It varies from the original Steiner problem in that the Steiner points must be chosen from a finite set of fixed given points. This problem has also been shown to be in the class of NP-hard problems [40]. A number of deterministic heuristic algorithms have been developed that deliver results in polynomial time which are guaranteed to be within twice the value of the optimum. [34, 64, 82, 84, 85, 98, 113]. These algorithms prune minimal spanning trees that contain an optimal Steiner tree.

There are many practical applications for the Steiner problem on graphs. Among these are the construction of telephone, pipeline, and transportation networks

[20, 35, 74, 91], the design of integrated circuits [5, 49, 50, 51, 58, 59, 87], building design [95, 97], routing [67], and Phylogeny [33].

The most effective exact methods for attacking the Steiner problem on graphs have been based on integer programming formulations of the problem [6, 8, 9, 112]. These methods attempt to derive lower bounds by means of Lagrangian relaxation [8, 9, 31] and dual ascent [74, 112]. methods, both of which are products of mathematical duality theory. The algorithm of Wong [112] is designed specifically for the directed Steiner Problem on graphs, but it can be applied to the undirected case by simply replacing each arc by two edges going in opposite directions. The branch and bound procedure in conjunction with the algorithms for finding lower bounds is the usual method employed to find an exact solution. Chapter III discusses both the exact and the heuristic algorithms.

The analysis of combinatorial search problems is usually directed toward the estimation of the worst case, but the information obtained by such an analysis is frequently of little use because the worst case seldom arises. A typical example is the simplex algorithm for linear programming which has a worst-case exponential complexity, but an average behavior of polynomial complexity. Average-case behavior is difficult to determine because of the difficulty in assigning probabilities to the enormous number of possible configurations in optimization problems. Since it is hard to analyze average case behavior, it is also hard to isolate the characteristics of algorithms with good average-case behavior. Hence, few general methods exist for attacking complex combinatorial optimization problems. The most common method is that of branch and bound. It is designed to avoid the worst case of complete enumeration of all possible solutions. While it takes exponential time to go through all configurations in many combinatorial optimization problems [81], it is often possible to make decisions early in the search tree that eliminate a large number of unfruitful possibilities. The effectiveness of branch and bound depends on the quality of the bounds that can be obtained on the solution at specific configurations. If there are numerous points close together in a graph, there are numerous solutions to the Steiner problem that differ by only small amounts. Hence, the branch and bound technique, even with a good lower bound producing algorithm such as the one by Wong, may spend a long time eliminating possiblilites that are in a shallow local minimum anyway.

A different approach is to use methods based on a probabilistic analysis of the average costs of admissible configurations in a optimization problem. These methods, inspired by the use of statistical mechanics in thermodynamics, offer the possiblity of near-optimal solutions in polynomial time for most instances of many NP-hard problems [100]. The physicist deals with complex systems involving many interacting particles that are in equilibrium with the surrounding environment. This requires a statistical rather than a deterministic description of the set of possible internal states of the system. Since it is impossible to perform enough experiments on the microscopic state to directly measure the probability distribution, methods of statistical mechanics are used to make systematic guesses as to the average values of such quantities as energy, number of particles, and volume. In the context of combinatorial optimization, the quantity that is to be estimated is the internal system energy, because it represents the cost function for any admissible state. The lowest energy state is analogous to the configuration that has the lowest cost.

The problem is to determine the probability distribution of the set of configurations from a given value of the average energy or cost. From information theory [92] comes the criterion that the probability distribution must maximize the degree of uncertainty, measured by the statistical entropy, concerning the occurrence of any configuration. This is the least biased estimate of the probability distribution

[11]. The distribution that results from this criterion is the Boltzmann distribution. This distribution contains the information regarding the division of possible states among different energy levels that the system can enter while undergoing a heat bath at a fixed temperature. As the temperature is decreased, the Boltzmann distribution assigns larger and larger probabilities to those states near the minimum energy level. This process of reducing a metal to its ground state is known as annealing.

Kirkpatrick [62] pointed out the similarity between combinatorial optimization and annealing in physics. For a given problem each trial solution corresponds to an internal state, and the evolution of the system is an algorithm which generates a sequence of states following a stationary process consistent with a value of the mean of the cost function. The specific value of the mean of the cost function is controlled by the temperature or the control parameter.

In physics random fluctuations exclude the possiblity of a global minimum of potential energy from ever occurring over a reasonable amount of time, but they result in a state that represents the average of a local minimum state and its neighbors. Furthermore, the system is not allowed to remain in local minima that are not very deep compared with the size of the average fluctuations. The global minimum may not be attainable, but by gradually lowering the temperature, a result very close to the optimum is likely to be reached. Low temperatures must be approached slowly to reduce the probability of getting caught in a shallow local minimum.

For the directed Steiner Problem on graphs and other combinatorial optimization problems a control parameter plays the role of temperature. It is decreased as the optimization process proceeds. Equilibrium of thermodynamic character is replaced by equilibrium in the sense of a Markov process. This method is called simulated annealing. It has proven to be a powerful tool for finding near-optimal solutions for a large number of NP-complete combinatorial optimization problems [105]. In addition, the asymptotic convergence of simulated annealing has been mathematically proven [46, 72, 77]. This has motivated a growing number of theoretical studies on the method [11, 36, 41, 43, 47, 80, 89, 101]. Interest in simulated annealing in a parallel processing environment is also an area of current research [1, 2, 17, 54, 56]. Chapter IV gives an overview of the known results concerning the annealing method.

While the paradigm of statistical cooling is intuitively simple and the theory has a solid mathematical foundation, the actual implementation for a specific application involves choices concerning four different parameters: the initial temperature, the length of the Markov chains, the method of decrementing the control parameter between chains, and the condition for terminating the cooling schedule. Simulated annealing has been applied successfully to a great many problems, but no general agreement exists , not only on what form the parameters should take, but also on how to determine the values of the parameters once the form has been established. A methodology for selecting these parameters for the directed Steiner Problem on graphs is presented in Chapter V. In Chapter VI, a comparison is made between the cooling schedule tailored to the directed Steiner Problem on graphs from Chapter V and the robust algorithm of Aarts and van Laarhoven [4] which determines the decrement of the control parameter and the criterion for termination dynamically during execution.

Chapter VII is devoted to a comparison between the results produced from using the quite different techniques of branch and bound and simulated annealing with respect to the directed Steiner Problem on graphs. Statistics on the accuracy and running times for these methods on 144 random graphs are compiled and analyzed. These random graphs have a variety of vertices, edge densities, and proportions of nodes that must be connected. The conclusions drawn from this comparison and the results of the earlier chapters of this study are presented in Chapter VIII.

# II. PRELIMINARIES

In this part of the dissertation relevant definitions are given in the first section, followed by some common reductions that can be done to simplify the directed Steiner tree problem on graphs.

## A. DEFINITIONS

All graph-theoretic concepts not defined in this section are in agreement with Harary [52], with the understanding that they are in a natural way extended to networks.

An undirected network G = (V,E,c) consists of a finite nonempty set V, of n vertices, labeled 1,2,...,n; a set, E, of m unordered pairs of distinct points of V, and a real-valued cost function $c\colon \to \mathbb{R}$. Each pair $e = (u, v)$ of points in E is an edge of G, and this edge is said to join u and v. Often e = uv is used to denote the edge (u,v) in which case u and v are said to be adjacent.

A directed network G = (N,E,c) consists of a finite nonempty set N of n points, labeled 1,2,...,n; a distinguished node called the root or source ; a set, E, of m ordered pairs of distinct points of N; and a real-valued cost function $c\colon \to \mathbb{R}$. Without loss of generality let 1 be the root node. Any ordered pair of distinct points e = (u,v) in E is called an arc and is denoted uv. The arc uv begins at u and ends at v. Then u is said to be adjacent to v and v is said to be adjacent from u. The outdegree(v) of node v is the number of arcs that begin at v, and the indegree(v) is the number of edges that end at v. The root has indegree 0 and outdegree $\geq 1$.

An alternating sequence of distinct points and arcs $v_0, a_1, v_1, a_2, \ldots, a_n, v_n$ in which each arc $a_i = v_{i-1}v_i$ is called a directed path. The cost of such a path is the sum of the costs of the arcs included in it. The cost of every arc $e = (i,j)$ in G is denoted by $c_{ij}$. If a directed path has identical first and last vertices while the remaining vertices are distinct, then it is called a cycle. A forest is a directed network without cycles.

If there is a path from u to v, then v is said to be reachable from u and the distance, d(u,v), from u to v is the length of any such path of minimum cost. Every node is reachable from the root node.

The minimum cost path between two vertices i and j in G is denoted by *$minpath_G(i,j)$*, and its cost is represented as $d_G(i,j)$. To denote the minimum cost path between a subset W of V and a vertex $i \in V$, we use *$minpath_G(W, i)$*.

A semipath is an alternating sequence $v_0, a_1, v_1, a_2, \ldots, a_n, v_n$ of points and arcs, but each $a_i$ may be either $v_i v_{i-1}$ or $v_{i-1} v_i$.

While a network is either connected or not connected, there are three different forms of connectedness for a directed network. A directed network is strongly connected if every two points are mutually reachable. It is unilaterally connected if for any two points at least one is reachable from the other; and it is weakly connected if every two points are joined by a semipath. A directed network is disconnected if it is not even weakly connected.

A unilaterally connected forest rooted at a given node is called a tree.

The undirected Steiner Network Problem (SNP) can be stated as follows.

Given: n vertices, a set, E, of m edges, and an
edge cost function c: $E \rightarrow R$, and a subset $W \subseteq V$
with w vertices.

Find: A subtree of G, $G_W$, that includes all
the vertices of W and such that the sum of the

weights of the edges in the subtree is a minimum [110].

It is not required that the cost function, c, satisfy the triangle inequality. The Directed Steiner Network Problem (DSP) requires finding a minimum cost tree rooted out of node 1 which spans a special set of points $W \subseteq N$. Node 1 is considered to be a member of W. In the work of this dissertation the edge weights $c_{ij}$ satisfy the condition $c_{ij} \geq 0 \; \forall (i,j) \in E$.

A network $H = (Z, F, c_F)$ is called a subnetwork of $G = (N,E,c)$ if $Z \subseteq N$, $F \subseteq E$, and $c_F(e) = c(e)$ for all $e \in F$. H is said to be a spanning subnetwork if $Z = N$. If Z is a subset of N, then the network induced by Z is the subnetwork of G derived by joining those pairs of nodes in Z which are adjacent in G. Any subset of N can induce a subnetwork of G. The concepts of subnetwork, spanning subnetwork, and a subnetwork induced by another can be defined for directed networks in an analogous manner.

A strong component of a directed graph is a maximal strong subnetwork. The set of all strongly connected components of G represents a partition of the set N of vertices.

As noted in Wong [112] we say that a node $v_1$ dangles from node $v_2$ if there is a directed path from $v_1$ to $v_2$, but not one from $v_2$ to $v_1$. A strongly connected component, $\Gamma$, is also a root component if $\Gamma$ contains a member of $W - \{1\}$, but no member of W dangles from a member of $\Gamma$.

Let $G' = (N, E', c_{E'})$ be a subnetwork of a directed network G. Define C(k) to be the set of points in $G'$ for which there is a directed path in $G'$ from i to k. Then the cut-set of node k is the set of arcs (i,j) that satisfy the following conditions.

1. $(i,j) \in E$.

2. $(i,j) \notin E'$.

3. $j \in C(k)$.

4. $i \notin C(k)$.

Suppose $\{P, \overline{P}\}$ is a partition of the vertices V in G. Then $\{ (i,j) \in E \mid i \in P, \text{ and } j \notin P \}$ is called a cut-set of edges between P and $\overline{P}$.

A network in which every vertex is adjacent to every other vertex is called a complete network. A network which can be embedded in a plane such that no two edges intersect geometrically except at a vertex to which they are both incident is said to be a planar network. The notions of complete and planar can be similarly defined for a directed network.

A network or directed network G is said to satisfy the triangle inequality if G is a complete graph, $c(e) \geq 0,\ \forall\ e \in E$, and $c(i,j) \leq c(i,k) + c(k,j)\ \ \forall$ i,j,k $\in$ N. The DSP does not assume that the triangle inequality is satisfied.

A contraction in an undirected network G = (N,E,c) along edge e = (i,j) is a process such that:

(a). edges incident to j are made incident to i.

(b). vertex j is removed from G, and, if $j \in W$, then i becomes an element of W in the contracted network.

(c). loops incident to i are removed from G, and if i has a pair of parallel edges incident to it, then only the one with the smaller cost is retained.

A contraction of a network G along a set $F \subseteq E$ is a sequential contraction along the edges of F.

## B. SPECIAL CASES AND REDUCTIONS

Consider some special cases of the Directed Steiner problem in networks.

1. If $|W| = 1$, then the minimal Steiner tree is a single vertex.

2. If $|W| = n$, then the SNP reduces to the minimal spanning tree problem for which there exists the well-known algorithm of Chu and Liu [21], and Edmonds [29].

3. $|W| = 2$. In this case the SNP is the shortest path problem. Again there is a well-known polynomial time algorithm for this problem [26].

Assume $G = (N, E, c)$, is connected, $c(e) > 0$ for all $e \in E$, and $2 \leq w \leq n$, where $w = |W|$. Let $T_W$ denote the directed Steiner minimal tree for W in G. Then use the local properties of G to reduce a particular instance of the DSP. The reductions are as follows [28]:

A. Least Cost Test: If $d_G(i,j) < c(i,j)$ for some arc $(i,j) \in E$, then this arc can be deleted. The algorithm of Floyd [32] can be used to implement this test.

B. Nearest Vertex Test: If $k \in W$ and $(i, k) \in E \ni$ $c_{ik} = \min\{c_{jk} | j \in N, j \neq k\}$. and if

$$d_G(1, i) + c_{ik} \leq \min\{c_{jk} | j \in N, j \neq i, k\}, \tag{2.1}$$

then arc $(i, k)$ is in an optimal solution

and nodes k and i can be fused.

C. Degree Zero Test: If $i \in N - W$ with outdegree zero, then

point i can be deleted.

D. Outdegree One Test: Let $i \in N - W \ni$ outdegree(i) $= 1$ and

denote by $(i, l)$ the only arc out of i. Then node i can be removed

when those arcs of minimum cost, $(q, i)$, entering i are

replaced by arcs $(q, l)$ of cost $c_{qi} + c_{il}$.

E. Indegree One Test: If k is one of the special vertices

and arc $(i, k)$ is the only edge entering k,

then this edge is in the solution and points i and k can be fused.

For most networks a large savings in time may be attained by using reductions. After applying these tests on the random graphs studied in this dissertation, the ratio, $\frac{n}{m}$, of nodes to edges varied from 0.18 to 0.30 for graphs where n was between 10 and 60 and the arc density ranged from 0.05 to 0.90.

The algorithms in this dissertation were programmed in Turbo Pascal 4.0 on an IBM PS/2 Model 80 with a mathematics coprocessor.

# III. LITERATURE REVIEW

The Steiner problem in networks is a combinatorial optimization problem that is closely related to the much studied Steiner problem in the Euclidean plane [23, 24, 44, 107]. This latter problem involves finding the shortest path joining a set of points in the Euclidean plane. To achieve this goal it is usually necessary to introduce other points at which the edges intersect. Although there are several exact algorithms for the Euclidean Steiner Problem (ESP) [75, 22, 13, 107] , the problem has been shown to be NP-Complete [38], and, because of the computation time, none of these algorithms is able to solve problem instances of even 100 vertices [107]. A heuristic algorithm that produces good solutions in $O(n \log n)$ time has been given by Smith, Lee, and Liebman [96]. Previously, a heuristic was developed by Chang [18].

If the locations of additional points are limited to a finite set, S, then the shortest network spanning a given set, W, will be a subnetwork of the complete network on $W \bigcup S$. If we omit the requirement that $W \bigcup S$ lies in the Euclidean plane and allow arbitrary edge costs rather than using the Euclidean distance between the vertices, then we obtain the Steiner network problem (SNP).

Karp [61] has proven by using a transformation from the Exact Cover by 3-Sets Problem that the Steiner Network Problem (SNP) is NP-complete. It has also been shown that the problem remains NP-complete even if all edge weights are equal, even if G is a bipartite graph having no edges joining vertices in $V - W$ [40], and Garey and Johnson [39] demonstrated that the problem was NP-complete even if G is a planar graph. Some special classes of graphs have been identified, however, for which the SNP can be solved in polynomial time [102, 103, 108, 109].

A rather obvious, but interesting, generalization of SNP is the Directed Steiner Network Problem (DSP), defined in the following way [48]:

Given: A directed network G = (V,E,c) with n vertices, a subset W ⊆ V with w vertices, a set, E, of m edges rooted out of some node, say $k_0$, of W, and an edge cost function c: E → $R$.

Find: A subtree of G, $G_W$, rooted out of $k_0$ that spans all the vertices of W such that the sum of the weights of the edges in the subtree is a minimum.

Wong [112] developed a method for finding a lower bound for the case when c(i,j) $\geq 0$ $\forall (i,j) \in$ E by means of an integer programming technique called the "dual ascent method." Although this method has never been employed in a heuristic algorithm, Winter [110] has suggested the following heuristic technique for attacking the DSP. Assume vertex 1 is the root.

Step 1: Find the directed network H using the dual ascent algorithm. Let Q represent the vertices in H that are connected by a directed path with vertex 1. W will be contained in Q.

Step 2: Find the minimum spanning tree, $T_Q$, of the subnetwork of G induced by Q.

Step 3: Delete from $T_Q$, one vertex at a time, those leaves that are Steiner vertices. Let $T_W$ be the result. Then $T_W$ is the approximate Steiner minimal tree.

Any undirected Steiner network can be transformed into a directed Steiner network by replacing each undirected arc with 2 directed arcs each having the same cost as the original arc. Since SNP is NP-complete and every SNP can be transformed into a DSP, it follows that DSP is also NP-complete.

## A. EXACT ALGORITHMS

There are a number of algorithms for the undirected SNP, some of which can be adjusted to fit the directed case. Hakimi [48] was the first to formulate the Steiner problems for undirected and directed networks. He showed that the solution of the SNP implied the solution of several covering problems in graphs, and he suggested that one solution of the problem was to enumerate all of the minimum spanning trees of subnetworks of G induced by the subsets of V containing W.

Lawler [66] gave the following modification of Hakimi's enumeration algorithm.

Step 1: Compute the shortest path lengths between all pairs of nodes and substitute these lengths for the arc weights, adding arc lengths where necessary to form a complete graph.

Step 2: For each possible subset of V with $p - 2$ or fewer Steiner points, find the minimal Steiner tree, where p is the number of special points.

Step 3: Choose the least costly tree from Step 2 and replace each arc of the spanning tree with the arcs of the shortest path between the associated endpoints.

The time complexity of the algorithm of Lawler is no worse than $O(p^2 2^{n-p} + n^3)$, where the $n^3$ results from the use of Floyd's algorithm for minimum cost path [32].

Balakrishnan and Patel have given another enumeration algorithm in which the original network is modified as follows [7]:

(i) a new node, call it node 0, is added

(ii) a zero weight edge is added connecting node 0 to node 1

(iii) zero weight edges are added between node 0 and each of the Steiner points.

Let $\overline{G}$ denote the new network, and suppose that $\tau$ represents the set of all spanning trees of $\overline{G}$ containing the edge (0,1). For any T $\in \tau$, let $T_1$ be the subtree of T containing node 1 when edge (0,1) is removed. If $T_1$ spans all of the demand nodes, then $T_1$ is a feasible solution to SNP. Moreover, for any feasible Steiner tree, $T'$, it is easy to show that there is at least one member of $\tau$ with a subtree equal to $T'$. Let T be the spanning tree made by adding to $T'$ the edge (0,1) and edges (0,i), for all Steiner points i not in $T'$. Then T is the shortest spanning tree of $\overline{G}$ with $T'$ for a subtree. T and $T'$ have the same total length.

Thus, the SNP is equivalent to finding the shortest spanning tree of $\overline{G}$ that has edge (0,1). and which has a subtree spanning the demand nodes, including node 1, when (0,1) is removed.

Balakrishnan and Patel [7] suggest the following enumeration procedure for solving the SNP.

(1). Generate spanning trees of $\overline{G}$ containing edge (0,1) in order of increasing weight.

(2). Let $T'$ be the first tree in this sequence with edge (0,1) which contains a subtree spanning the special vertices when (0,1) is deleted.

(3). Call this subtree $T_{least}$. Then $T_{least}$ is the required minimal Steiner tree.

The algorithm of Gabow [37] can be used for generating spanning trees of $\overline{G}$ in order of increasing cost.

Hakimi [48] also suggested another algorithm that is related to a well-known method of Melzak for the Euclidean Steiner problem [75]. This approach is based on the fact that $T_w$ must have at least two W-vertices i and j satisfying one of the conditions:

(a). deg(i) = deg(j) = 1, and the path from i to j contains Steiner vertices only; one and only one of these Steiner vertices has degree greater than 2 in $T_w$.

(b). either deg(i) = 1 or deg(j) = 1, and the path from i to j contain only Steiner vertices of degree 2 or no vertices at all.

If (a) or (b) is satisfied, then i and j are said to be "p-adjacent" in $T_w$ [48]. Hakimi's algorithm determines the minimum cost tree spanning W in G with i and j p-adjacent, or concludes that no such tree could be $T_w$. Since there is no way to know beforehand which of the W-vertices are p-adjacent in $T_w$, it is necessary to consider every combination of i and j, to attempt to construct the minimum cost tree containing the W nodes with i and j p-adjacent, possibly concluding that no such tree can be $T_w$, and to select as $T_w$ the answer that yields the minimum cost. Thus, this algorithm is not very efficient. Hakimi gave no computational results for these solution procedures.

Dreyfus and Wagner [27], gave an exact algorithm based on dynamic programming methods which had a worst case complexity of $O(n3^r + n^2 2^r + n^3)$. The basis of the algorithm is the "optimal decomposition property," which can be stated as follows.

> Let T be any Steiner tree connecting W, the set of special points, and let $p \in W$. If $|W| \geq 3$, then $\exists$ a vertex $q \in N$ and a subset $Z \subseteq W$ such that:
>
> (a). Z is a proper subset of W - $\{p\}$, and $Z \neq \emptyset$.
> (b). $T = T_1 \bigcup T_2 \bigcup T_3$, where $T_1$, $T_2$, and $T_3$ are pairwise disjoint.
> (c). $T_1$ is a Steiner path joining q and p.
> (d). $T_2$ is a Steiner tree connecting $\{q\} \bigcup Z$.
> (e). $T_3$ is a Steiner tree connecting $\{q\} \bigcup (W - Z - \{p\})$.

To implement the property directly, an arbitrary node $p \in W$ would be chosen, followed by a search for the optimal q, which itself requires both the discovery of the optimal subset $Z \subset W$, and that Steiner trees $T_2$ and $T_3$ be found. The entire process would be recursive. Dreyfus and Wagner, however, chose to construct the solution in the following $|W| - 1$ steps [27].

```
1.        Remove a node, say p, from W. Let D = W − {p}.
2.        For each set of 2 vertices in D and any
          node n ∈ N, solve the Steiner problem.
          n can belong to D, or it can even be p.
3.        Use the result of step 2 in order to solve Steiner
          problems joining each set of 3 nodes of D and any
          node n ∈ N.
•
•
•
|W| − 2.  Solve the Steiner problems for each set of
          |W| − 2 nodes of D and any node of N.
|W| − 1.  Solve the Steiner problem connecting D with point p.
```

If $E \subseteq D$, and $n \in N$, then each step in the algorithm above requires two searches: Search 1 finds a node, $q \in N$; Search 2 gets the optimal subset $F \subseteq E$ in the sense that the sum of the costs of the Steiner paths joining $\{q\} \bigcup F$ and $\{q\} \bigcup (E - F)$, plus the cost of the path from n to q is the minimum cost for a tree connecting three such sets. This technique of constructing larger optimal solutions from optimal solutions of all possible smaller problems is a general process used in dynamic programming. Levin [69] also gave a dynamic programming algorithm for solving the SNP that was slightly less efficient than the one of Dreyfus and Wagner. According to Christofides [19], the algorithms of Hakimi and of Dreyfus and Wagner, while better than complete enumeration, are able to handle, in a reasonable amount of computing time, problems with only slightly more than ten special vertices.

Shore, Foulds, and Gibbons [94] presented a branch and bound enumeration scheme which separates in a systematic way all of the feasible solutions (i.e. those trees spanning the special nodes) into smaller partial solutions. Each of these smaller subsets of edges is analyzed using upper and lower bounds to determine whether it is able to contain the optimal feasible solution.

A specific set of edges in included and another specific set of edges is excluded in every partial solution. The set of included edges for a partial solution will comprise a set of connected components. A solution is feasible if it contains only one connected component that spans W. The branching process creates two nodes. One of these nodes represents the addition of an edge for consideration, and the other solution corresponds to the exclusion of the edge from consideration. The former node is always selected first.

Branching at an unfathomed node, q, involves selecting an edge producing two new partial solutions For each edge a penalty for not including it in the set of selected edges is assigned The edge with the largest penalty is chosen for branching This edge is found in the following way

Let the points in W be labelled $w_1, w_2, \ldots, w_p$, and let the points in $V - W$ be $w_{p+1}, w_{p+2}, \ldots, w_n$

1 The penalty vector, $T = \{t_i \mid i = 1, 2, \ldots, p\}$, is computed by these steps

   (a) $c_i = \min_{1 \le j \le n} \{c_{ij}\}$, $k_i$ = the value of j that is associated with $c_i$

   (b) $c^*_i = \min_{\substack{1 \le j \le n \\ j \ne k_i}} \{c_{ij}\}$

   (c) $t_i = c^*_i - c_i$

2 Let $t_h = \max_{1 \le i \le p} \{t_i\}$

Then $\{v_h, v_{k_h}\}$ is the edge chosen for branching The node q becomes the parent of two new nodes Each of these nodes corresponds to a partial solution that is identical to that for q, except that for one node $\{v_h, v_{k_h}\}$ is included in the partial solution, and for the other node $\{v_h, v_{k_h}\}$ is excluded from the partial solution

The cost matrix $C_{ij}$ is temporarily adjusted in order to obtain a bound for each partial solution If the new partial solution resulted from the addition of edge $\{v_x, v_y\}$ to the set of edges removed from consideration, then temporarily let $c_{ij} = c_{ji} = \infty$ If the new partial solution was obtained by augmenting the set of included edges , then the components containing $v_x$ and $v_y$ are joined C is then reduced by one row so that it becomes a square matrix again such that each row and each column correspond to a unique component Let the new cost matrix be denoted by $C'$ The values, $c'_{ij}$ in the new matrix must be defined by

$$c'_{ij} = c_{ij}, \quad if\ i \neq x, y \ \wedge j \neq x, y$$

$$c'_{ix} = c'_{xi} = \min \{c_{xi}, c_{yi}\} \ \ if\ 1 \leq x \leq p,\ i = 1, 2, \ldots, n,\ \ i \neq y$$

$$c'_{iy} = c_{yi} = \min\{c_{yi}, c_{xi}\} \ \ if\ i = 1, 2, \ldots, n,\ \ i \neq x$$

A lower bound is calculated for a node with cost matrix $C$ by means of the following theorem.

**Theorem 1:** Let $z'$ be a minimal solution of the SNP on $G(N, E)$, and suppose

$$a = \sum_{i=1}^{p} \underset{1 \leq j \leq n}{\mathrm{Min}} \{c_{ij}\} \text{ and } b = \left( \sum_{i=1}^{p} \underset{1 \leq j \leq p}{\mathrm{Min}} \{c_{ij}\} \right) - \underset{\substack{1 \leq i \leq p \\ 1 \leq j \leq p}}{\mathrm{Min}} \{c_{ij}\}. \text{ Then}$$

$$z' \geq \mathrm{Min}\ \{a, b\} \tag{3.1}$$

Upper bounds for each node in the branch and bound tree are found by applying the algorithm of Kruskal [65] or that of Prim [83] to the edges in the graph at that node until all the points in W are connected.

Shore, Foulds, and Gibbons [94] compared their algorithm to the first algorithm of Hakimi [48] and to the algorithm of Dreyfus and Wagner [27]. The evidence presented was for 45 random graphs of from 10 to 30 nodes, using ALGOL on a Burroughs B6700. The results indicated that when p is small compared to n, Dreyfus and Wagner's algorithm was superior, when p is close to n/2, the algorithm of Shore, Foulds, and Gibbons performed the best, and when p was close to n, Hakimi's algorithm outperformed the other two. The computational evidence also showed that the algorithm of Shore, Foulds, and Gibbons, unlike the other two algorithms, was characterized by results that fluctuated greatly depending on the distribution of the edge weights. For all three of these algorithms, it often took more than 100 seconds of CPU time to find a solution.

Aneja [6] suggested an approach which treated the SNP as an integer programming formulation of a set covering problem. Let $\{P, \overline{P}\}$, be a partition of N such that $W \cap P \neq \emptyset$ and $W \cap \overline{P} \neq \emptyset$. Let the cut-sets for these partitions be labeled $C_1, C_2, \ldots, C_q$, and suppose the edges of G are labeled $e_1, e_2, \ldots, e_m$. $A = (a_{ij})$ is a matrix where

$$a_{ij} = \begin{cases} 1 & \text{if } e_j \in C_i \\ 0 & \text{otherwise} \end{cases} \tag{3.2}$$

for i = 1,2, ... , q and j = 1,2, ... , m. Then the set covering problem is:

$$\text{Min} \sum_{j=1}^{m} c_j x_j \tag{3.3}$$

such that

$$\sum_{j=1}^{m} a_{ij} x_j \geq 1, \quad i = 1, 2, \ldots, q \tag{3.4}$$

$$x_j \in \{0, 1\}, \quad j = 1, 2, \ldots, m \tag{3.5}$$

Suppose that $X^* = (x^*_1, x^*_2, \ldots, x^*_m)$ is an optimal solution to the set covering problem. Let $T_{X^*}$ be a subnetwork of G whose edge set corresponds to those edges, $e_j \ni x = 1$. Then $T_{X^*}$ is the solution to the Steiner network problem. This is easily shown by the following argument. $T_{X^*}$ has no cycles since $c_{ij} > 0 \ \forall \ i$ and $j$. Also, $T_{X^*}$ has exactly one component connecting all the vertices of W, because otherwise a violation of a constraint corresponding to some cut-set $C_i$ would occur. Lastly, since $T_{X^*}$ is found using equation (3.4) , $T_{X^*}$ yields the tree that spans W with the minimum cost.

Since the number of constraints for the set covering problem grows exponentially with the number of nodes, the algorithm treats the constraints implicitly by means of a "row generation scheme" that is a modification of a known algorithm for the general set-covering problem given by Bellmore and Ratliff [10]. In this algorithm it is necessary to solve the linear programming relaxation of the formulation of the set covering problem given above. Since q is large for the Steiner network problem, the dual simplex method for the linear relaxation is not efficient. Aneja showed how to determine the entering and leaving variables for a current basis without explicitly employing the constraints.

Aneja [6] implemented his algorithm in FORTRAN IV for an IBM 370 computer. He attempted problems with as many as 50 nodes, 60 edges and $|W| = \frac{n}{2}$. Solutions to the linear relaxation form of the set covering problem yielded very good bounds on those problems for which the algorithm was able to terminate. However, out of 80 problems that Aneja attempted, 23 did not terminate within 60 seconds, and, of those 23, 20 did not finish the first iteration. Thus, it seems likely that a method of solving the set covering formulation other than linear relaxation could make this method more practical.

Wong [112] suggested a method for obtaining a tight lower bound on the SNP. He formulated the problem as follows.

$$\text{Min} \sum_{(i,j)\in E} c_{ij}y_{ij}, \qquad (3.6)$$

subject to:

$$\sum_{h\in N} x_{ih}{}^{k} - \sum_{j\in N} x_{ji}{}^{k} = \begin{Bmatrix} 1, & i=1, \\ -1, & i=k, \\ 0, & i\neq 1,k, \end{Bmatrix} k \in W \tag{3.7}$$

$$x_{ij}{}^{k} \leq y_{ij}, \tag{3.8}$$

$$x_{ij}{}^{k} \geq 0, \quad (i,j) \in E, k \in W, \tag{3.9}$$

$$y_{ij} \in \{0, 1\}, \tag{3.10}$$

where $y_{ij}$ indicates whether or not arc (i,j) is contained in the solution, and $x_{ij}{}^{k}$ is the amount of flow between nodes 1 and k on arc (i,j). Equation (3.7) causes one unit of commodity k to be sent from node 1 to node k, and constraint (3.8) permits flow only on an arc that is in the solution. In this representation, corresponding to every edge $(i,j) \in$ E, there are two variables, namely $y_{ij}$ and $y_{ji}$. Thus, this form of the SNP is suitable for defining the Directed Steiner Problem(DSP). The optimal solution to this formulation yields the cost of a directed Steiner minimal tree.

The dual of the linear relaxation of (3.6) - (3.10) is:

$$\text{Max} \sum_{k\in W} (v_k{}^{k} - v_1{}^{k}), \tag{3.11}$$

subject to:

$$v_j{}^{k} - v_i{}^{k} - u_{ij}{}^{k} \leq 0, \; k \in W, \; i,j \in N, \tag{3.12}$$

$$\sum_{k \in W} u_{ij}^{k} \leq c_{ij},\ (i,j) \in E, \tag{3.13}$$

$$u_{ij}^{k} \geq 0. \tag{3.14}$$

Solving the dual problem would provide a lower bound for the problem defined by (3.6) -(3.10). Moreover, Wong proved that the solution to the dual is equal to the solution of the linear relaxation of the set covering algorithm of Aneja [6]. Unlike Aneja, Wong does not attempt to solve (3.11) - (3.14) to optimality. Instead, he offers a dual ascent heuristic for obtaining a near-optimal solution.

```
1. Initialize:

    $v_i^k = 0,\ \ k \in W, i \in N,$
    $u_{ij}^k = 0,\ \ (i,j) \in E,$
    $S(i,j) = c_{ij} - \sum_{k \in W} u_{ij}^k = c_{ij}.$

    Form the auxiliary graph H = $(N, A)$ with A empty.
    Since $A = \emptyset$ all of the vertices are strongly connected components
    and every element of $W - \{1\}$ is a root component.
    Let $C(k)$ denote the set of vertices in H that are connected to k
    by a directed path.

2. Choose a root component R. Stop when there are no
    root components left.

3. Select a node k that is in the intersection of W and R .
    Define $S(i^*, j^*) = \min\{S(i,j) \mid (i,j) \in$ cut set of k$\}$.
    Then
        (a). $\forall\ h \in C(k),\ v_h^k = v_h^k + S(i^*, j^*)$.
        (b). $\forall\ (i,j) \in$ cut set of k, $u_{ij}^k = u_{ij}^k + S(i^*, j^*)$ and
            $S(i,j) = S(i,j) - S(i^*, j^*)$.

4. Update the auxiliary graph by $H = H \bigcup \{(i^*, j^*)\}$.
    Then return to Step 2.
```

In the algorithm all members of W are root components and so all $v_k^k$, $k \in$ W are possible choices for incrementing at Step 2. Each variable $v_h^k$, $h \in C(k)$ is increased in Step 3. $S(i,j)$ represents the slack for the constraint in the dual problem corresponding to arc $(i,j)$. The second part of Step 3 makes the $u_{ij}^k$ variables larger and reduces the slack variables $S(i,j)$ for those arcs (i,j) in the cut set of k. Since $(i^*,j^*) \in$ cut set of k, $S(i^*,j^*) = S(i^*,j^*) - S(i^*,j^*) = 0$. Thus, Step 4 adds an arc $(i^*,j^*)$ to H whose slack variable is zero.

The initial solution in Step 1 for equation (3.11) where all variables are zero is feasible. Wong showed that the modification of the variables in Step 3 maintained the feasiblity of the solution. Since during each iteration of the heuristic, one arc is added to H, the algorithm will terminate after finitely many iterations.

Using an IBM 3033 computer, Wong applied his algorithm to four groups of six very sparse graphs, two groups with 40 nodes and two groups with 60 nodes. Except for two of the graphs the algorithm found the exact answer. With the other two graphs the result deviated from the optimal by less than 1%. The average running times for all four groups was less than 0.6 sec. Hence, a branch and bound algorithm similar to the one used in this dissertation should be able to find optimal solutions very efficiently.

The integer programming technique of Lagrangian relaxation [42, 31, 93,]. has been applied to the undirected Steiner Network Problem by Beasley [8, 9]. He has used this method in a branch and bound algorithm that makes use of three different formulations of the SNP. Each node, $N_i$, in the binary depth first search tree is characterized by having a set $IN_i$ of Steiner vertices included in the feasible solution and a set of $OUT_i$ of excluded Steiner vertices. At each stage in the tree search procedure there is, moreover, a set $F_i \subseteq$ E of edges that has been identified as being in the solution and a set $F_0 \subseteq E$ of edges identified as not belonging to the Steiner minimal tree. Suppose $G_t$ is the subnetwork of G constructed by removing the Steiner points in OUT$_t$, and by applying reductions to the resulting undirected network. In $G_t$ the vertices in IN$_t$ are considered to be elements of W. Let $W_t$ denote the W-vertices in $G_t$. To find an optimal solution, if one exists, at a node in the tree search is equivalent to locating the Steiner minimal tree $T_t$ for $W_t$ in $G_t$.

The upper bound $U_t$ for a tree search node N can be determined by adding the costs of edges identified during the branch and bound procedure as belonging to the Steiner minimal tree for W $\bigcup$ IN$_t$ in G – OUT$_t$ to the cost of a tree spanning $W_t$ in $G_t$ found by any of the heuristic methods described in the next section of this chapter.

Beasley [8, 9] gave three different ways of finding the lower bound, $L_t$, at N$_t$, all of which employed Lagrangian relaxations of the SNP for $W_t$ in $G_t$ defined as 0-1 linear programming problems. These problems are defined here for the initial tree node $G_t$ = G and $W_t$ = W.

Let $W_1$ = W – {1}. Let $y_{ij}$ be a binary variable corresponding to edge (i,j) and suppose $x_{ijk}$ represents the amount of flow between points 1 and k on edge (i,j) in the direction from i to j. Then formulate the SNP as follows:

$$\min \sum_{(i,j)} c_{ij} y_{ij} \tag{3.15}$$

such that

$$y_{ij} \geq x_{ijk} + x_{jik} \qquad \forall (i,j) \in E,\ \forall k \in W_1 \tag{3.16}$$

$$\sum_{(i,j)\in E} x_{1jk} \geq 1 \quad \forall k \in W_1$$
$$\sum_{(h,k)\in E} x_{hkk} \geq 1 \quad \forall k \in W_1 \tag{3.17}$$
$$\sum_{(i,j)\in E} x_{ijk} - \sum_{(h,i)\in E} x_{hik} \geq 0 \quad \forall k \in W_1,\ \forall i \in N\backslash\{1, k\}$$

$$p - 1 \leq \sum_{(i,j)\in E} y_{ij} \leq n - 1 \tag{3.18}$$

$$y_{ij} \in \{0, 1\} \quad \forall (i,j) \in E \tag{3.19}$$

$$x_{ijk} \geq 0 \qquad \forall i,j \ni (i,j) \in E,\ \forall k \in W_1. \tag{3.20}$$

Expressions (3.16) and (3.17) force any solution to yield a connected subnetwork spanning W. Expression (3.18) is redundant, but it strengthens the first two lower bounds, LR1 and LR2, described below.

LR1 is determined by the Lagrangean relaxation of (3.16). Let $s_{ijk}$ be Lagrangean multipliers for (3.16). Then $s_{ijk} \geq 0\ \forall (i,j) \in E, k \in W_1$. LR1 is then defined as:

$$\min \sum_{(i,j)\in E} \left( c_{ij} - \sum_{k\in W_1} s_{ijk} \right) y_{ij} + \sum_{k\in W_1} \sum_{(i,j)\in E} s_{ijk} x_{ijk} \tag{3.21}$$

such that (3.17) - (3.20) are satisfied.

The optimal solution to (3.21) is a lower bound of the cost of the Steiner minimal tree. This expression can be decomposed into a number of separate subproblems. The first subproblem is

$$\min \sum_{(i,j)\in E} \left( c_{ij} - \sum_{k\in W'_1} s_{ijk} \right) y_{ij} \tag{3.22}$$

such that (3.18) and (3.19) hold. Then $y_{ij} = 1$ if and only if $c_{ij} - \sum_{k\in W'_1} s_{ijk} y_{ij}$ is either among the $(p-1)$ smallest coefficients or is negative among the $(n-1)$ smallest coefficients. The remaining subproblems are of the form

$$\min \sum_{k\in W'_1} \sum_{(i,j)\in E} s_{ijk} x_{ijk} \tag{3.23}$$

subject to (3.17) and (3.20). These subproblems can be solved by $|W'_1|$ shortest elementary path calculations locating the least expensive path from node 1 to node $k \in W'_1$ in the graph with cost matrix $(s_{ijk})$. The algorithm of Dijkstra [26] can be used for this purpose. LR1 applies the subgradient optimization algorithm [31, 53] to improve the lower bound found in (3.21).

The second lower bound, LR2, is obtained by employing the Lagrangean relaxation of (3.17). Denote the Lagrange multipliers by $t_{ik}$. The lower bound problem is as follows.

$$\text{Min} \sum_{(i,j)\in E} c_{ij} y_{ij} + \sum_{\substack{i,j\in N \\ (i,j)\in E}} \sum_{k\in W'_1} C_{ijk} x_{ijk} + \sum_{k\in W'_1} (t_{1k} + t_{kk}) \tag{3.24}$$

such that conditions (3.16) and (3.18 - 3.20) are true, where

$$C_{ijk} = \begin{cases} -t_{ik} - t_{jk} & \text{if } i \neq k, j = k \\ -t_{ik} + t_{jk} & \text{if } i \neq k, j \neq 1, k \\ 0 & \text{otherwise} \end{cases}$$

where $t_{ik} \geq 0 \ \forall i \in N$, $k \in W_1$, and (3.16) and (3.18 - 3.20) must be satisfied.

From (3.16) when $y_{ij} = 1$, the optimum contribution from $y_{ij}$ is

$$b_{ij} = c_{ij} + \sum_{k \in W_1} \min\{0, C_{ijk}, C_{jik}\} \tag{3.25}$$

This permits (3.24) to be rewritten as:

$$\min \sum_{(i,j) \in E} b_{ij} y_{ij} + \sum_{i,j \in W_1} (t_{1k} + t_{kk}) \tag{3.26}$$

such that (3.18) - (3.20) are satisfied. The optimal solution to this representation is obtained by setting $y_{ij} = 1$ exactly when $b_{ij}$ is either among the $(p-1)$ smallest coefficients, or it is negative and among the $(n-1)$ smallest coefficients. Again to improve the lower bound given in (3.26) subgradient optimization [31, 53] is applied.

The third Lagrangean relaxation form of the problem is based on a shortest spanning tree problem with additional constraints. The network G is modified by adding a vertex 0 and by connecting that vertex with every node $i \in N - W$ by edges with zero cost. Let $\overline{G}$ denote the modified network. Then find the shortest spanning tree of the resulting graph with the restriction that in the shortest spanning tree any vertex in N − W connected by an edge (0,i) must have degree one. After deleting edge

(0,1) from the result, the component containing W is shown by Beasley to be the desired Steiner minimal tree [9].

Suppose $E_j$ represents edges of $\overline{G}$ incident to node $j \in N$. Then the LR3 problem can be formulated in the following way.

$$\min \sum_{(i,j)\in E} c_{ij} y_{ij} \tag{3.27}$$

such that

$$\{(i,j) \mid y_{ij} = 1\} \text{ is a spanning tree of } \overline{G} \tag{3.28}$$

$$y_{0k} + y_{pq} \leq 1 \quad \forall k \in N - W, \ \forall (p, q) \in E_k \tag{3.29}$$

$$y_{ij} \in \{0, 1\} \ \ \forall (i,j) \in E \tag{3.30}$$

The lower bound is determined by Lagrangean relaxation of (3.29). Let $u_{jpq}$ be the Lagrangean multipliers, where $j \in N - (W \cup \{0\})$, $(p, q) \in E_j$. By properly arranging the coefficients, $c_{ij}$, and the multipliers, the problem LR3 can be written in a form that is equivalent to the unconstrained minimum spanning tree problem, which is easily solved by any of the well-known polynomial time algorithms [65, 83].

As before, the subgradient optimization algorithm [31, 53] is used to improve the answer to LR3.

The quality of the lower bounds obtained from LR3 varies widely depending on which vertex of G is designated node 1. Beasley [9] suggested that node 1 be a vertex of W whose average distance to other W-points is a minimum.

Beasley's algorithms are, at least for sparse networks, the most efficient among the algorithms for SNP. LR1 and LR2 are comparable in their effectiveness with LR1 producing a smaller duality gap (i.e. (optimal value - maximum lower bound at the root)/ (optimal value)) and LR2 finding the optimal solution in less time. In most cases, however, the differences are insignificant [8].

LR3 was implemented on a CRAY X-MP/48 computer with special vector processing capabilities not available on the CDC 7600 which was used to test LR1 and LR2 [8, 9]. However, even taking into account this extra computing power, LR3 outperforms the other two relaxations both in terms of duality gap and running times. Using the CRAY X-MP/48 Beasley was able to solve some networks as large as 1000 nodes and 25000 edges within 20 minutes. The efficiency of LR3 improves as p increases.

Exact algorithms using spanning tree enumeration methods [48, 66], integer programming [6, 94, 8, 9, 112], and dynamic programming [27, 69], have been devised, but they are not computationally efficient as they require a number of computational steps that is either exponential in the total number of nodes or exponential in the size of the set of special nodes that must be spanned. The best of these algorithms appears to be those of Beasley for undirected graphs and Wong for directed graphs. Beasley uses Lagrangian relaxation of a 0-1 integer programming problem to provide a lower bound which can be utilized in a tree search procedure to obtain an exact answer. The algorithm of Wong, has produced results comparable for graphs of up to 60 vertices. This algorithm has not previously been used in a branch and bound scheme like the one used by Beasley. In this dissertation, however, a branch and bound implementation of Wong's algorithm is compared with simulated annealing in the study of the Directed Steiner Problem on graphs.

None of the exact algorithms is very efficient. Most of them cannot handle more than 30 nodes. The algorithms of Beasley and Wong can be incorporated into branch and bound schemes capable of doing much larger problems. The large amount of storage and computation time required to maintain the lists used by these procedures, however, has limited the applications with more than 60 nodes to very sparse networks. It is difficult to describe or compare the performances of the exact algorithms mentioned in this section, because the creators of these techniques offer very limited computational evidence. Moreover, the methods, coded in a variety of programming languages and executed on different computers, have been employed to solve a diverse collection of problem instances. The only certain fact about these algorithms is that they do require exponential time in worst cases.

## B. HEURISTIC ALGORITHMS FOR SNP

The inherent intractability of the SNP has made the creation of reliable polynomial time heuristics that provide nearly optimal solutions to the SNP of great practical importance.

### 1. The Algorithm of Takahashi and Matsuyama (TM).

The first important heuristic was published by Takahashi and Matsuyama [98] in 1979. At each step in the algorithm a tree with a subset $W_k$ of $W$ is constructed, and a new vertex of W is inserted together with a shortest path connecting $W_k$ and the vertex. Let $|W| = w$. Then the algorithm can be stated as follows.

Step 1: Start with subtree $T_1 = (W_1, E_1)$ of G

consisting of a single, arbitrary vertex, i, of W.

Let k = 1, $E_k = \{\ \}$, and $W_k = \{ i \}$.

Step 2: Determine a W-vertex, $i \in W - W_k$ closest to $T_k$.

Construct tree $T_{k+1} = (V_{k+1}, E_{k+1})$ by adding to $T_k$ the minimum cost path joining i to $T_k$. k = k + 1.

Step 3: If k < w then go to Step 2. When k = w, then $T_w$ is the solution. Stop.

We note that this algorithm requires at most $O(wn^2)$ time, because the path from i to $T_k$ can be computed in time complexity $O(n^2)$ by Dijkstra's algorithm [26]. The idea behind this algorithm comes from the following observation of Gilbert and Pollak [44] in their important early paper. Let $H = (W,E')$ be the complete graph on the special vertices, and let the cost of an edge (u,v) in H be $d_G(u,v)$. Thus, the minimum spanning tree in H is approximately equal to the Steiner tree for G. Its worst case cost relative to an optimal tree is less than or equal to $\frac{1}{2}$. Takahashi and Matsuyama show that the result, $T_w$, of their algorithm has a cost $\leq 2(1 - \frac{1}{w})$ times that of an optimal Steiner tree. That is, for all n and w $(2 \leq w \leq n - 1)$

$$\frac{c(T_w)}{c(T_{Steiner})} \leq 2 - \frac{2}{w}. \tag{3.31}$$

If w = n, then $c(T_W) = c(T_{Steiner})$ = the cost of a minimal spanning tree of G.

Furthermore, they prove that for all n and w, $2 \leq w \leq n$, $\exists$ a network G = (V,E,c) and a subset W contained in V, where $|V| = n$, and $|W| = w$, such that Equation 3.31 holds with equality. The proof of this latter assertion is as follows [98]:

Proof:

Let

$V = \{ 1,2, \ldots , w+1 \}$, $E = \{(i,j) | \; i = 1,2, \ldots , w+1, j = 1,2, \ldots , w+1\}$, $W = \{1,2, \ldots , w\}$.

Suppose

$$c(i,j) = \begin{cases} 1 & \text{if } i = 1, 2, \ldots, w, j = w + 1. \\ 2 & \text{if } i = 1, 2, \ldots, w - 1, j = i + 1. \\ 10 & \text{otherwise .} \end{cases}$$

See Figure 1 [98].

The tree, $T_{W'} = (W', \{ (i,i+1) | i = 1,2,..,w,w-1 \})$

results from the algorithm and its cost is 2(w-1). The

desired ratio is found by dividing 2(w-1) by the

optimal length, w, of the Steiner tree.

Rayward-Smith and Clare have suggested an improvement to the TM algorithm through the addition of the following two steps [85].

Step 4: Let $\hat{V}$ denote the vertices of V - W in $T_w$.

Step 5: Find the minimum spanning tree of the

subgraph of G induced by

$W \bigcup \hat{V}$ and prune this tree of all

nonspecial leaves.

2. <u>The Algorithm of Kou, Markowsky, and Berman (KMB).</u>

KMB was developed in 1981 [64]. It has these steps.

Step 1: Construct the complete, undirected graph

$G_1 = (V_1, E_1, d_1)$, from G and W such that

$V_1 = W$ and $\forall (i,j) \in E_1$, $d_1(i,j) = d_G(i,j)$.

Notice that for each edge of $G_1$,

there is a corresponding shortest path in G.

Step 2: Find a minimum spanning tree, $T_1$, of $G_1$.

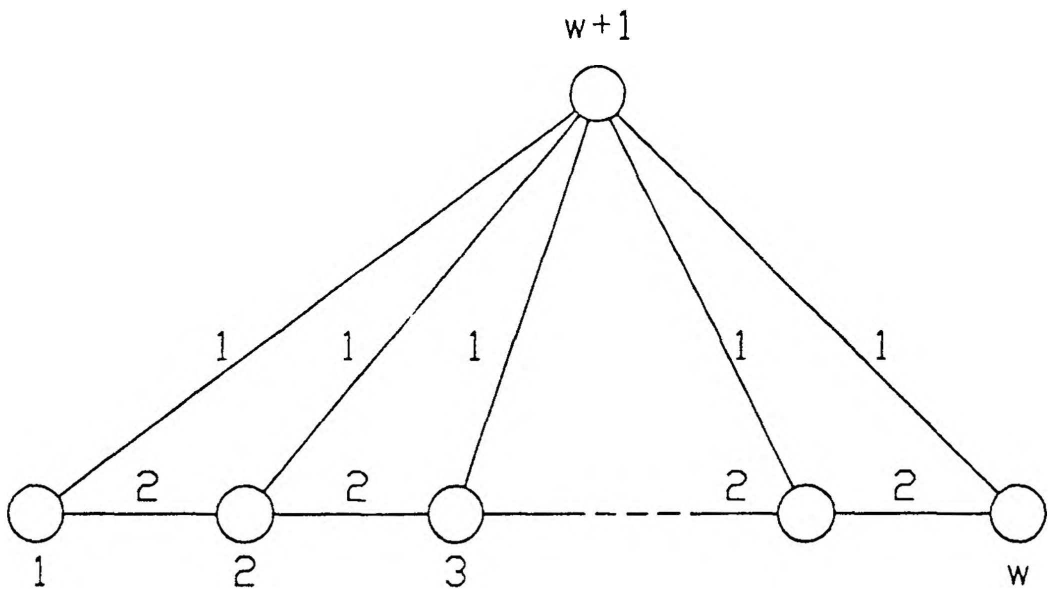

Figure 1. For this tree equality holds in Equation (3.31).

Step 3: Construct the subgraph, $G_s$, of G by substituting a corresponding minimum cost path in G for each edge in $T_1$.

Step 4: Find $T_s$, the spanning tree of $G_s$.

Step 5: Prune from $T_s$ any edges necessary to make all leaves special vertices. Call the result $T_H$. $T_H$ is the desired Steiner minimal tree.

Figure 2 shows an example of this process [64]. In the figure, W = $\{ v_1, v_2, v_3, v_4 \}$. $G_1$ is shown in Figure 2 b, $T_1$ appears in Figure 2 c, $G_s$ is pictured in Figure 2 d, $T_s$ is given in Figure 2 e, and Figure 2 f shows the final Steiner tree, $T_H$, that results from KMB. The answer is exact in this case. This example shows that $T_1$, $G_s$, and $T_s$ may not be unique.

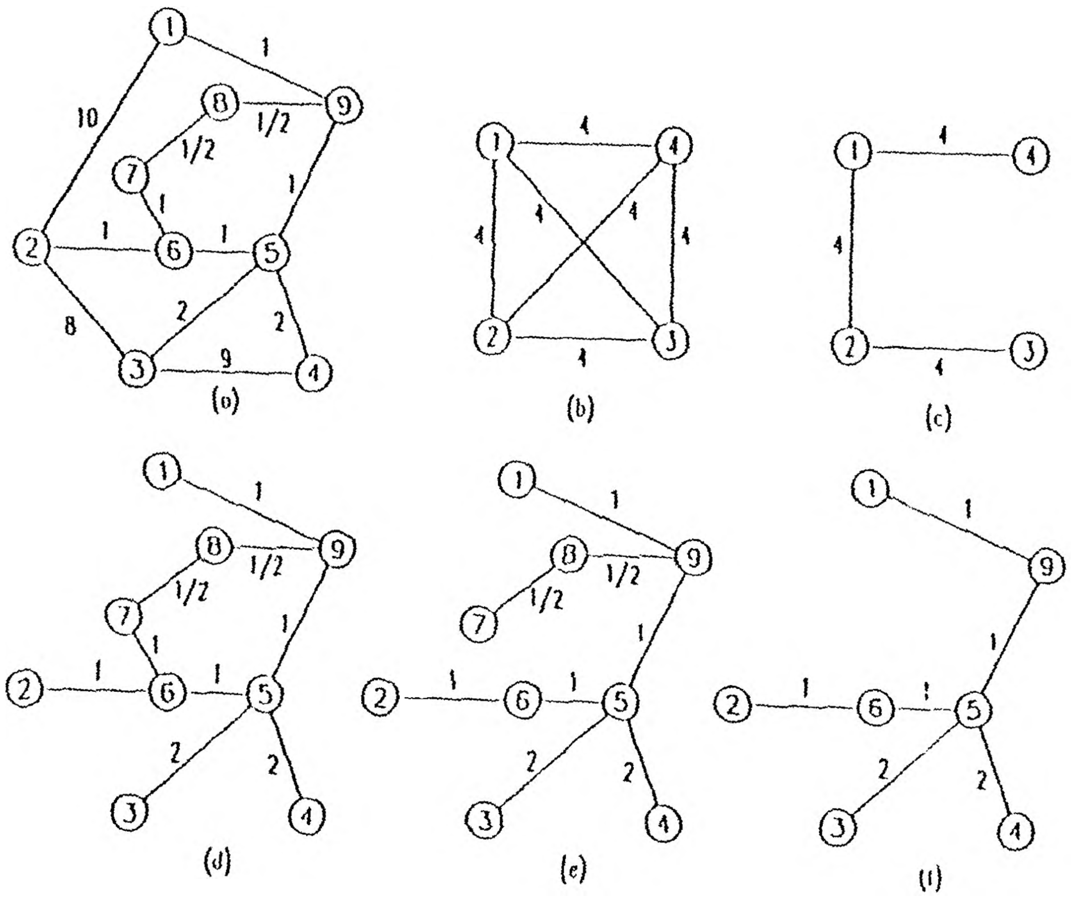

Figure 2. An Example of the KMB Algorithm

As far as computational complexity is concerned, Step 1 can be done in $O(|W||V|^2)$ time by Floyd's algorithm [32], Step 2 can be completed in $O(|W|^2)$ time, with Prim's algorithm [83], Step 3 requires $O(|V|)$ time, Step 4 can be done in $O(|V|^2)$ time, and Step 5 needs $O(|V|)$ time. Thus, Step 1 dominates the computational time, and the heuristic has a worst case complexity of $O(|W||V|^2)$. Kou, Markowsky, and Berman [64] prove that

$$\frac{c(T_H)}{c(T_{Steiner})} \leq 2(1-\frac{1}{\eta}) \leq 2-\frac{2}{w}, \tag{3.32}$$

where $\eta$ equals the total number of leaves in $T_{Steiner}$. Moreover, they show that this upper bound is the best possible. Hence, the heuristics of TM and KMB have the identical worst case behavior. However, the algorithm of TM may at times produce a better result. For instance, in Figure 3a, $T_{TM}$ has cost 15 (if i or j is chosen as the initial W-vertex), while $T_{KMP}$ in Figure 3b, has cost 16 [110]. In Figure 3, the edges selected by the algorithms, and the special vertices, are in double lines. The quality of answer given by $T_{TM}$ depends on the choice of the initial W-vertex.

3. <u>The Algorithm of Wu, Widemayer, and Wong (WWW).</u>

Wu, Widemayer, and Wong [113] have made improvements on the KMB algorithm that make KMB run in O(|E| log|V|) time unless the graph is very dense and most of the vertices are not in W [113]. The central contribution of Wu, Widemayer, and Wong is the idea of a generalized minimum spanning tree, $T_{gen}$, that causes Steps 1 and 2 of KMB to run faster.

$T_{gen}$ and $T_1$ both contain W. They are different in that $T_{gen}$ is constructed directly from the edges of E whereas $T_1$ uses $E_1$. $T_{gen}$ has the following properties:

(a). When $T_{gen}$ is constructed, $G_1$ is not explicitly constructed.

(b). $T_{gen}$ has the same path information as $G_s$.

(c). $T_{gen}$ is a tree. Hence, $T_{gen}$ is equivalent to $T_s$.

(d). No leaves in $T_{gen}$ are Steiner vertices. Thus, $T_{gen}$ is equivalent to $T_H$ in algorithm KMB although there are examples of problem instances where they do not have the same total cost. The reason for a discrepancy is that Step 4 of KMB may make decisions on deleting edges of $G_s$ which produce a different final Steiner tree. This tree may be better or worse than $T_{gen}$.

The total cost of $T_{gen}$ is at most $2(1 - 1/\eta)$ times the cost of a Steiner minimal tree, where $\eta$ is the number of leaves in the optimal Steiner tree. This bound is the same as that for $T_H$.

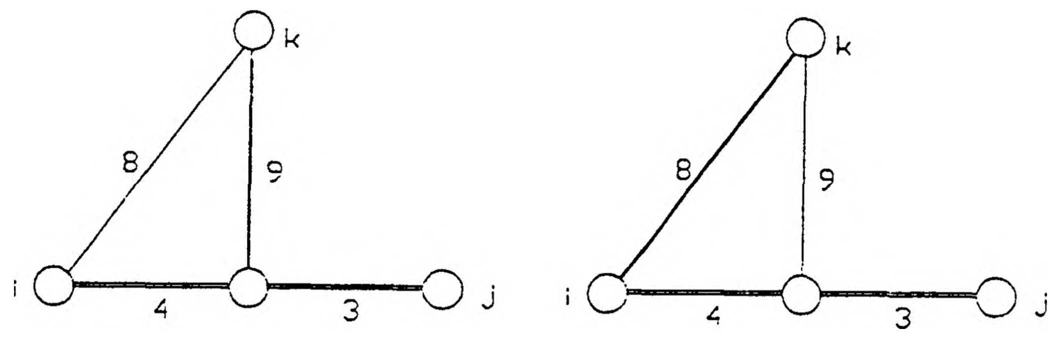

Figure 3. (a) Results from TM (b) Results from KMB

A generalized minimum spanning tree $T_{gen}(V_{gen}, E_{gen}, d_{gen})$ of a given network G = (V,E,c) and a set $W \subseteq V$ is a tree subgraph of G such that

(a). there exists a minimal spanning tree

$T_1(V_2, E_2, d_2)$ of $G_1(V_1, E_1, d_1)$

such that for all edges $(i,j) \in T_1$,

the unique path in

$T_{gen}$ from i to j is of length $d_2(i,j)$

(b). all leaves of $T_{gen}$ are in W.

Consequently, one way to describe $T_{gen}$ is that it is the realization of $T_1$ on G. We now give Algorithm M of Wu, Widemayer, and Wong for finding a generalized minimum spanning tree. For each $v \in V$, let source(v) be a vertex in W which is closest to v, and let length(v) represent the distance from source(v) to v. Treat the nodes in W initially as a forest of $|W|$ separate trees which will eventually be merged into a single tree. The process is similar to the one employed in Kruskal's algorithm. Let Q be a priority queue used to store boundary vertices of paths extended from the trees and possible edges to be used for linking the trees. In the triple (t,d,s), t is a member of V, s is a member of W, and d is the cost of a path from s to t. If t is in W, then (s,t) has the potential of becoming an edge in a generalized minimum spanning tree. Otherwise, s is a possible vertex for source(t). Q is ordered by nondecreasing values of d in the tuples.

<u>Algorithm M of Wu, Widemayer, and Wong</u> [113].

```
Step 1: $\forall\, i \in W$,
    $source(i) \leftarrow i$ and $length(i) \leftarrow 0$.
    $\forall i \in V - W$,
    $source(i) \leftarrow$ undefined and $length(i) \leftarrow \infty$.
Step 2: $\forall\, i \in W$,
        put the triple (r,d,q) into Q, where (q,r) belongs to E
        and $d = d_G(q,r)$. If both q and r belong to W, then only one of
        $(r, d_G(r,q), q)$ and $(q, d_G(q,r), r)$ is placed in Q.
    Step 3: Partition the nodes in W into |W| trees
         such that each tree has exactly 1 vertex.
    Step 4: While all vertices of W are not in a
         single tree, do: Choose a triple (t,d,s)
         with minimum d in Q and remove it from Q.
           Case 1: if source(t) is undefined, then do:
              $source(t) \leftarrow s$
              $length(t) \leftarrow d$.
              $\forall r \ni (t,r) \in E$ and source(r) is undefined, put
              (r,d(t,r) + d,s) into Q.
           Case 2: if source(t) and s are in the same
              set, then take no action.
           Case 3: if source(t) and s are not in the
              same tree, then do:
              subcase (a). $t \in W$. Then merge the 2 trees containing s and t,
               and record an MST edge (which is a path
               in G) between s and t.
              subcase (b). $t \in V - W$. Then put
              (source(t),d + length(t),s) into Q.
```

The authors show that Algorithm M can be easily modified so that it produces an explicit description of the path associated with each edge of the generalized minimum spanning tree. Thus, the modified algorithm calculates $T_{gen}$ directly.

Intuitively, the time bound of $O(|E| \log |V|)$ is possible because when $|W|$ is relatively large compared to $\log |V|$, KMB must perform $|W|$ shortest-path operations, while Algorithm M calculates the shortest paths at the same time it produces $T_{gen}$. Another situation where Algorithm M provides better performance is when G is not very dense. Then $|E|$ is much less than $\frac{|V^2|}{\log |V|}$.

4. <u>The Revised Algorithm of Rayward-Smith (RSR).</u>

V. J. Rayward-Smith [84] developed a heuristic of a different form in 1983 and revised it slightly in 1986 [85]. In this algorithm, at each iteration, T denotes a set of trees which will be subgraphs of the final tree. To begin with, T is made up of the individual vertices of W. A vertex $i \in V$ is chosen using a heuristic function, f, where f(i) is expressed in terms of the cost from i to the various subgraphs of T. After the vertex $i \in V$ is chosen which minimizes f(i), this vertex is used to unite two of the trees in T by joining each of them to i by means of the cheapest route possible. At each iteration of the algorithm, |T| decreases by one until |T| = 1. When T has just one element, then that tree is the near minimal Steiner tree.

Suppose the heuristic function used is

$$f_1(i) = \min \{ d_G(i, V_j) + d_G(i, V_k) \mid V_j, V_k \text{ are vertex sets of 2 elements of T} \}. \quad (3.33)$$

Then any $n \in V$ on the minimum cost path which unites two distinct trees of T will yield the same minimum result from the heuristic function. Therefore, the two trees nearest each other will be joined at each iteration. Actually, the heuristic, f, employed by Rayward-Smith [84] is a little different. Let |T| = k and let $n \in V$. Then

$$f(n) = \min_{1 \leq r \leq k-1} \left\{ \sum_{i=0}^{r} \frac{d_G(n, t_i)}{r} \mid t_0, t_1, \dots, t_r \in T \right\} \quad (3.34)$$

For a given $n \in V$, f can be equivalently defined as follows: To start with, order the elements in T by increasing cost from n. Denote this ordering by $t_{n,0}, t_{n,1}, \ldots, t_{n,k-1}$, where $d_G(n,t_{n,0}) \leq d_G(n,t_{n,1}) \leq \cdots \leq d_G(n,t_{n,k-1})$. Now create a sequence $f_1(n), f_2(n), \ldots, f_{k-1}(n)$ by defining $f_1(n) = d_G(n,t_{n,0}) + d_G(n,t_{n,1})$ and

$$f_r(n) = \frac{\left[ (r-1) f_{r-1}(n) + d_G(n, t_{n,r}) \right]}{r} \text{ for } 1 < r \leq k-1. \tag{3.35}$$

Thus, $f(n) = \min\{f_r(n) \mid 1 \leq r \leq k-1\}$.

Then $f(n) = f_r(n)$, where $f_1(n) > f_2(n) > \cdots > f_r(n)$, and $f_r(n) \leq f_{r+1}(n) \leq \cdots \leq f_k(n)$. Notice that $f_i(n) < f_{i-1}(n)$ if and only if $d_G(n,t_{n,i}) < f_{i-1}(n)$. Therefore, f can be computed by the following pseudocode.

```
i:= 1;
evaluate $f_1(n)$ ;
while if i < k then $d_G(n, t_{n,i}) < f_{i-1}(n)$ else false
   do begin
      evaluate $f_i(n)$ ;
      $i := i + 1$
   end [S4].
```

If i = k or $d_G(n,t_{n,i-1}) \geq f_i(n)$, then the computation stops and $f(n) = f_i(n)$. Intuitively, f(n) measures the least average cost of n from a set of elements of T. We now present the algorithm of Rayward-Smith.

Step 0: (Initialization)

$T := \{(\{p\},\{\ \}) \mid p \in W\}$

For each pair $i,j \in V$ find $d_G(i,j)$ and $minpath_G(i,j)$

(Floyd's algorithm can be used for this).

For each $i \in V$, $t \in T$ find $d_G(i,t)$ and $minpath_G(t,i)$.

Step 1: (Iteration)

For each $i \in V$, evaluate $f(i)$.

Let $m \in V$ such that $f(m) \leq f(i) \ \forall i \in V$. If there is more than one choice of m, then the decision is made according to the following criteria.

Suppose the choice is between m and $m'$,

$$\text{where } f(m) = \sum_{i=0}^{r} \frac{d_G(m, t_{m,i})}{r} \text{ and } f(m') = \sum_{i=0}^{r'} \frac{d_G(m', t_{m',i})}{r'} .$$

Then m is chosen over $m'$ when

(i). m is not a node in the graph $\sum_{i=0}^{r} t_{m,i}$, but $m'$ is a node of $\sum_{i=0}^{r'} t_{m',i}$.

If this does not determine which node has preference, then we try

(ii). if $r < r'$. If neither (i) nor (ii) separates m and $m'$ so that $r = r'$, then

(iii). if r < |T|, then if $f_{r+1}(m) < f_{r-1}(m')$ or if $f_{r-1}(m) = f_{r-1}(m')$, then if $f_{r-2}(m) < f_{r+2}(m')$ ... .

Finally if all three of these tests fail to determine which node has preference, then an arbitrary choice is made.

Step 2: Corresponding to the chosen node m is an ordering of T, $t_{m,0}, t_{m,1}, \ldots , t_{m,k-1}$, where $k = |T|$.

Let $t'$ be the graph $t_{m,0} \bigcup t_{m,1} \bigcup minpath_G(m, t_{m,0}) \bigcup minpath_G(m, t_{m,1})$.

$t'$ will be a tree because it has no cycles.

Step 3: Remove $t_{m,0}$ and $t_{m,1}$ from T and insert $t'$.

Step 4: If $|T| \neq 1$, then for each $i \in V$, find $d_G(i, t')$ and $minpath_G(t', i)$, and go to step 1.

If $|T| = 1$, then the only element of T is $t_{RS}$, the approximation to the Steiner minimal tree.

The initialization step has a time complexity of $O(n^3)$, where |V| = n, because of the use of Floyd's algorithm to find the shortest route between each pair of nodes of V. Evaluating f(n) is easier if a record is kept for each $i \in V$ of the list $t_{i,0}, t_{i,1}, \ldots, t_{i,k-1}$, of elements of T ordered by nondecreasing distance from i. Initially, k = |W|, and, for each $i \in V$, the ordering requires O(|W| log|W|) time. Because |W| < |V|, the first lists can be computed in $O(n^2 \log n)$ time. After the construction of these lists each iteration of the algorithm destroys two of the trees in T and inserts a new one in each ordered list. This is a O(k) procedure for each list, where $k = |T|$. Since there exists n lists and $|T| \leq |W| \leq |V|$, each iteration uses $O(n^2)$ time for updating the lists. After the ordering $t_{i,0}, t_{i,1}, \ldots, t_{i,k-1}$ is completed, evaluating f(i) uses O(k) time for each i. Thus, Step 1 requires $O(n^2)$ time. Step 2 and Step 3 also require $O(n^2)$ time. In Step 4, the evaluation of $d_G(i,t')$ and $minpath_G(i,t')$ necessitate looking at no more than n values for each i. Hence, Step 4 is at worst an $O(n^2)$ process. So each iteration is no worse than $O(n^2)$. There are no more than $|W| - 1$ iterations where $|W| \leq n$. We conclude that the iterative part of the algorithm, like the initialization step, is $O(n^3)$.

Rayward-Smith used Monte Carlo methods to test this algorithm [84]. He used the rectilinear metric on an m x n grid with $2 \leq |S| \leq mn$ of the nodes randomly selected to belong to W. The results were compared with the exact cost as calculated by Dreyfus and Wagner's dynamic programming algorithm [27]. Since the exact results were needed, Rayward-Smith reports that only small cases could be compared. The results, however, for these small examples were encouraging. Out of 100 test cases, the heuristic was wrong only twice and then by an error of about 5 percent in each case.

In 1986, Rayward-Smith and Clare revised the algorithm in the same way that they proposed revising the algorithm of Takahashi and Matsuyama [98]. That is, they suggested using the original algorithm of Rayward-Smith to approximately compute $\hat{V}$, the set of vertices of $V - W$ that are in the Steiner minimal tree. Then they find the the minimum spanning tree of the subgraph induced by $W \cup \hat{V}$, and prune this tree of all leaves that are Steiner vertices.

Rayward-Smith and Clare did an extensive comparison of the computational performances of KMB,TMR and RSR. They began by choosing the eighteen problems tested by Beasley with his exact algorithm [8]. Each of these graphs had between 50 and 100 nodes. For seventeen of the eighteen graphs an exact solution was known, but in all of these graphs the number of edges was less than or equal to 2|V|. A nineteenth problem with a greater density of edges was, therefore, constructed. It had 100 vertices, 500 edges, and costs uniformly distributed in the interval [1,10]. An optimal solution to this problem was not known. Overall, the best algorithm was either TMR or RSR. The results appear in Table I below.

Table I. COMPARISON OF THREE HEURISTIC ALGORITHMS BY RAYWARD-SMITH AND CLARE ON PROBLEMS WITH KNOWN ANSWERS.

| | KMB | TMR | RSR |
|---|---|---|---|
| Average Percentage Error | 3.9 | 1.8 | 1.3 |
| Worst Percentage Error | 14.0 | 7.9 | 5.1 |
| Correct Answer(for 17 Problems) | 4 | 9 | 9 |

Secondly, 1500 random graphs with between 10 and 80 vertices were generated using costs either uniformly distributed in (0,1] or with a normal distribution of mean = .5 and $\sigma = .125$. The three algorithms, KMB, TMR, and RSR, were all tested on these graphs. The RSR algorithm performed the best using either distribution of costs and for all sizes of graphs. Only on 13 occasions out of 1500 did RSR fail to obtain the best solution of the three algorithms. On those occasions the difference of the answer obtained by RSR and the algorithm that did find the best solution was always less than 5% of the best answer. Recently, it has been shown that the worst-case performance of the algorithm of Rayward-Smith is within two times the optimum and that two is the best possible bound in the sense that there are problems for which the algorithm will always do worse than any number less than two times the optimum [104].

5. The Algorithm of Plesnik (PL).

The last heuristic considered in this paper is one by Plesnik [82], which uses a generalized idea of the notion of contraction defined in Section 2. Several new definitions must be introduced before the algorithm can be given.

Let $i \in W$ and $r > 0$. Then let $\{x \in G \mid d_G(x,i) \leq r\}$ be called a neighborhood of i with radius r. The point x may or may not be a vertex in V, it can be any point on the edges of G. The set of all points of all neighborhoods, N(i), $i \in W$, with the same radius, can be partitioned into classes by the following process. Let $x \in N(i)$ and $y \in N(j)$ belong to the same class whenever there is a sequence $N_1, N_2, \ldots, N_k$, of neighborhoods with $N(i) = N_1$ and $N_1 \cap N_2 \neq \emptyset$, $N_2 \cap N_3 \neq \emptyset, \ldots, N_{k-1} \cap N_k \neq \emptyset$, and $N_k = N(j)$.

Denote a class by $C_p$, where $p = \min \{i | i \in C_p \cap W\}$. Classes for the network in Figure 4, where r = 1, are denoted by dotted curves. The figure appears on page 149 of [110].

The contraction of a graph, G, in a set of vertices $W \subseteq V$, with radius r, is a graph, $G_C$, formed as follows.

1. Every class $C_p$ is contracted to a single vertex p,
which is considered
to be a W-vertex on G.
All Steiner vertices that lie
outside of any class are left unchanged by the
contraction.

2. If e = (i,j) is an edge of G such that i and j
belong to different classes, then an edge in the
contraction is generated according to these rules:
   (a). If neither i nor j belongs to any class,
   then e is unchanged in G.
   (b). If $i \in C_p$ and j does not belong to a class, then j is a
   Steiner point and G contains the edge e = (p,j)
   whose cost is given by

$$C_G(p,j) = \left\{ \begin{array}{ll} C_G(i,j) - r & \text{if } i \in W \\ C_G(i,j) & \text{if } i \in V - W \end{array} \right\}.$$

   (c). If $i \in C_p, j \in C_q, p \neq q$, then G has edge e = (p,q)
   whose cost is given by

$$C_G(p,q) = \left\{ \begin{array}{ll} C_G(i,j) - 2r & \text{if } i,j \in W \\ C_G(i,j) & \text{if } i,j \in S \\ C_G(i,j) - r & \text{otherwise} \end{array} \right\}.$$

3. All but the minimum cost edge connecting any pair of vertices in G are deleted (Figure 4c).

Having defined the concept of the contraction of a graph G, we now are in a position to state Plesnik's algorithm [110].

Step 1: Find the minimum cost edge in G that is incident to a W-vertex, and let r be the cost of that edge. Construct neighborhoods $N_i(r), i \in W$, and the associated classes $C_p, p = 1,2, \dots, v$.

Step 2: For each class, $C_p$, let $W_p$ represent the W-vertices of the class. For any vertex, i, in $W_p$, if $|W_p| \neq 1, \exists$ another vertex of $W_p$ on a path whose distance from i is less than or equal to 2r. Hence, using Takahashi and Matsuyama's algorithm we can obtain a tree, $T_p$, spanning $W_p$ with $c(T_p) \leq 2r(|W_p| - 1)$.

Step 3: If v = 1, then $T_1$ is the near minimal Steiner tree and we can return. Otherwise, let $G_C$ be the contraction of G. Let $W_C$ be the W-vertices of $G_C$.

Step 4: Find a tree $T_C$ spanning $W_C$ in $G_C$. This is to be done by recursion.

Step 5: For each $p \in W_C$ substitute $T_p$ for $p = 1, 2, \dots, v$. Reconnect $T_C$ and $T_1, T_2, \dots, T_v$ by adding no more than $\sum_{p \in W_C} (\text{degree}_{G_C}(p))$ edges of length r. This can be done because

the cost of a path from a boundary point

of $C_p$ to a vertex of $W' \cap C_p$

is equal to r by the definition of

contraction. The resulting tree, $T_{PL}$ is the solution. Stop.

Figure 4 may be helpful in understanding this algorithm. Part (b) of the figure illustrates the result of applying all but the last part of the definition of a contraction. Part (c) shows $G_C$, which is obtained from part (b) by deleting all but a single minimum cost edge between any pair of vertices.

This algorithm permits the use of any heuristic in Step 2 for finding shorter trees $T_p$. TM was selected for the work reported in Section 5, because it is easy to implement and very accurate for trees with a small number of vertices.

None of the recursive steps in the algorithm of Plesnik requires more than $O(n^2)$ time, where $|V'| = n$. Also, not more than n recursions are needed. Thus, the worst case complexity is $O(n^3)$.

Plesnik [82] proved that in the worst case, the error ratio satisfies $\frac{c(T_{PL})}{c(T_{Steiner})} \leq 2(1 - \frac{1}{w})$, and that the bound is tight.

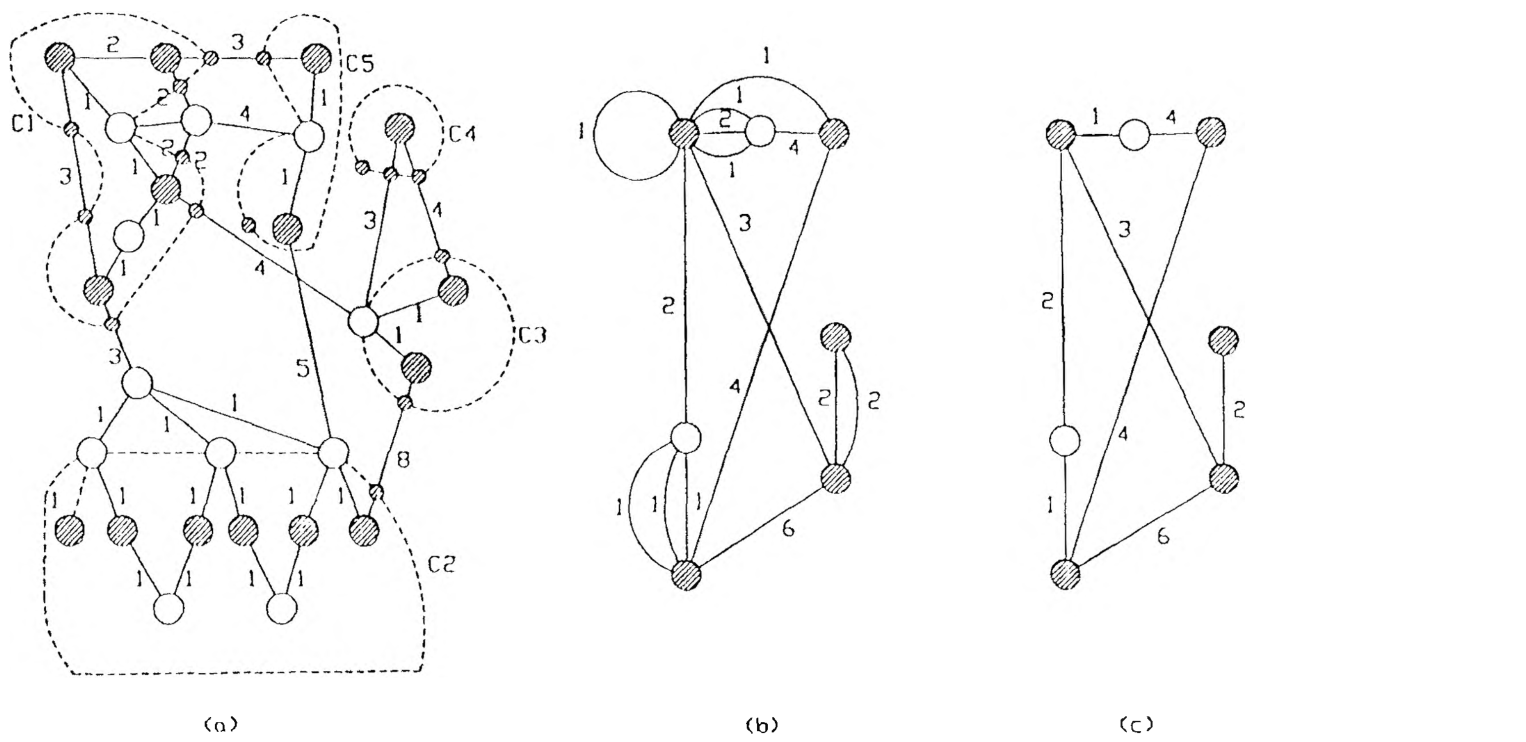

Figure 4. Stages in the Algorithm of Plesnik

# IV. THE METHOD OF SIMULATED ANNEALING

Simulated annealing, as proposed by Kirkpatrick et al. [62], is a popular Monte-Carlo algorithm for finding globally optimum configurations in large NP-complete problems with cost functions containing many local minima at near-stationary points.

## A. INTRODUCTION

For many important practical and theoretical problems, the objective is to find optima of functions of discrete variables. The search for these results, called combinatorial optimization [81], has brought forth many excellent approximation and exact algorithms. The well-known Traveling Salesman Problem (TSP), for example, can now be solved for an instance of up to 318 cities in less than 10 minutes of CPU time on an IBM 370/168 computer [25]. It still remains true, however, that many practical large-scale optimization problems can only be solved approximately on existing computers, a fact which corresponds to the theoretical classification of a large number of these problems, including the TSP and DSP, as NP-complete [40]. Thus, the development of good heuristic algorithms for which there is no certainty that the solution is optimal, but for which there is a high probability of near optimality and for which polynomial time bounds can be given on the computation time, has become necessary.

An instance of a combinatorial optimization problem can be defined in terms of a pair $(\Lambda, C)$, where $\Lambda$ is a finite or countably infinite set of configurations and $C: \Lambda \rightarrow \mathbb{R}$ is a cost function which maps each configuration into a real number. It is assumed, for ease of discussion, that $C$ is defined such that the goal of the optimization is to find the configurations which minimize the value of $C$.

There are two types of approximation algorithms: algorithms tailored to a specific problem, and general methodologies that can be applied to a wide spectrum of combinatorial optimization problems. Simulated annealing is certainly an example of the latter. It is based on Monte Carlo techniques, but it incorporates some aspects of iterative improvement algorithms.

Iterative improvement or local search schemes require the definition of configurations, a cost function, and a method for generating a transition from one configuration to another. The generation prescription formulates neighborhood $\Lambda_i$ for each configuration i, consisting of all configurations reachable from i in a single transition [81]. Iterative improvement works as follows. Generate a sequence of iterates beginning from an initial configuration, each iterate of which is made up of a possible transition from the current configuration to one chosen from the configurations in the neighborhood of the current configuration. If the configuration of the neighbor has a lower cost, then replace the current configuration by its neighbor. Otherwise, another neighbor is selected and the same cost configuration is repeated. The algorithm ends when a configuration has a cost not greater than the cost of any of its neighbors. Disadvantages of iterative improvement include:

(a). the procedures end at a local minimum, and there is no information concerning the deviation of this local minimum from the global minimum.

(b). the particular local minimum found depends on the initial configuration and there are no rules for choosing the initial configuration.

(c). an upper bound for the computation time is frequently an open problem.

One way to avoid these pitfalls is to execute the algorithm for a large number of uniformly distributed initial configurations. This increases the running time, but it finds the global minimum with probability 1 as $n \to \infty$. An example of this method is Lin's 2-opt strategy for the Traveling Salesman Problem [70]. Another approach is to accept, with restrictions, transitions associated with an increase in the cost function. This is what simulated annealing does [62, 16].

In its original form simulated annealing is based on the analogy between combinatorial optimization and the problem in statistical mechanics of determining the lowest-energy ground state of a physical system with many interacting atoms [62, 16]. Simulated annealing is also known as "statistical cooling" [100] and "probabilistic hill climbing" [86]. In physics annealing is a physical process in which a solid is heated to a temperature at which all particles of the solid randomly arrange themselves in the liquid state, followed by a slow cooling, spending a relatively long time near the freezing point. At each temperature T, the solid is allowed to enter thermal equilibrium, a state in which the energy, $E$, at a configuration i is given by the Boltzmann distribution

$$Pr\{\mathrm{E} = E_i\} = \frac{1}{\mathrm{Z(T)}} \cdot \exp\left( -\frac{E_i}{k_b T} \right), \tag{4.1}$$

where Z(T), called the <u>partition function,</u> is a normalization factor equal to $\sum_i \exp(E_i/k_b T)$, $k_b$ is the <u>Boltzmann constant,</u> and $\left( -\frac{E_i}{k_b T} ) \right)$ is known as the <u>Boltzmann factor.</u> The Boltzmann distribution is concentrated on states with low energy as the temperature decreases. Only states with minimum energy have a positive probability of occurrence. If, however, the temperature is lowered too fast, the system does not have time to reach thermal equilibrium for each value of the temperature. The resulting configurations may then have defects in the form of high-energy, metastable, structures rather than having a low energy crystalline lattice form [100]. Intuitively, quickly cooling a solid, or "rapid quenching" [111], is analogous to iterative improvement in combinatorial optimization.

## B. THE SIMULATED ANNEALING ALGORITHM

In 1953, Metropolis et al [76] suggested a procedure for the simulation of the equilibrium states of a solid at a given temperature. The procedure is as follows [111].

```
Metropolis' Procedure

begin
    Select an arbitrary initial configuration $I_0$;
    repeat
        $I_0'$ = a random neighboring configuration of $I_0$;
        $\Delta = E(I_0') - E(I_0)$;
        $Prob = \min(1, e^{-\frac{\Delta}{k_b T}})$;
        if random(0, 1) $\leq Prob$ then $I_0 = I_0'$;
    until false;
end;
```

The Metropolis algorithm was adapted to combinatorial optimization problems by substituting configurations for states in the physical system, by substituting the cost function $C$ for the energy function $E$ of a given state, and by replacing $k_b T$ with a control parameter c, sometimes called the "temperature." If $\Delta \geq 0$, then the probability of acceptance of the perturbed state is $\exp\left(-\frac{\Delta}{c}\right)$, which is known as the Metropolis criterion [100].

Kirkpatrick et al. [62] generalized this technique by introducing a sequence of decreasing values of c, at each of which the Metropolis procedure is executed until the system attains equilibrium. The generic simulated annealing algorithm, given below, is the result [111].

```
Simulated Annealing Algorithm

begin
  $I$ = Initial Solution $I_0$.
  $c$ = Initial Temperature $c_0$.
  while { stopping condition is not satisfied } do
    begin
      while ( equilibrium is not reached ) do
        begin
          $I'$ = a solution of a random neighbor of I .
          $\Delta = C(I') - C(I)$.
          $Prob = \min(1, e^{-\frac{\Delta}{c}})$.
          if random(0, 1) $\leq Prob$ then $I = I'$.
        end.
      update c.
    end.
  print best solution.
end.
```

Simulated annealing can be considered as a robust form of an iterative improvement algorithm. Like the latter, it defines a generation mechanism for moving from one configuration to another. It randomizes this mechanism and permits occasional moves that worsen the current solution in an effort to avoid getting caught in a local optimum. These uphill moves are dependent on the temperature c and become progressively less likely as $c \rightarrow 0$. As with iterative improvement, once a configuration, a cost function, and a generation mechanism are suitably determined, then a solution to the problem can be found. When the value of c is 0, the simulated annealing algorithm corresponds to a version of iterative improvement. Lundy and Mees [72] show that simulated annealing performs better than this version of iterative improvement for many problems. They give an example where the average number of transitions needed for the iterative improvement algorithm to reach a global minimum is $O(n^2)$, while the number required by the simulated annealing method is $O(n)$.

The correspondence between statistical mechanics and combinatorial optimization has led to definitions of measures in combinatorial optimization that are analogous to certain averages in statistical mechanics such as entropy, and specific heat. These are frequently used to control the cooling rate dynamically. Before discussing cooling schedules, however, it is necessary to give a mathematical model of simulated annealing, and to discuss convergence of the algorithm.

## C. MATHEMATICAL MODEL

The generic simulated annealing algorithm is explained in terms of a Markov chain, a sequence of trials where the result of each trial depends only on the result of the preceding one [55]. In simulated annealing a trial corresponds to a transition and the result of a transition depends only on the current configuration.

Let $P_{ij}(k-1, k) = Prob$ ( outcome of $k^{th}$ trial = j | outcome of $(k-1)^{th}$ trial is i ), and suppose $a_i(k) = Prob$ ( outcome of $k^{th}$ trial = i). Then recursively,

$$a_i(k) = \sum_{l \in T} a_l(k-1) P_{li}(k-1, k), \quad k = 1, 2, \dots, \tag{4.2}$$

where T = set of possible outcomes [100]. Let $X(k) =$ result of the $k^{th}$ trial. Then

$$P_{ij}(k-1, k) = Prob\left(X(k) = j \mid X(k-1) = i\right) \tag{4.3}$$

and $a_i(k) = Prob\,(X(k) = i)$. If the conditional probabilities are independent of k, then the Markov chain is said to be homogeneous, otherwise it is nonhomogeneous. The matrix $P(k-1,k)$ is called the transition matrix [4]. It is a function of c. If c is constant, the Markov chain is homogeneous and its transition matrix is given by

$$P_{ij}(c) = \begin{cases} A_{ij}(c)G_{ij}(c) & \text{if } i \neq j \\ 1 - \sum\limits_{l=1,\, l\neq i}^{|\Lambda|} G_{il}(c)A_{il}(c) & \text{if } j = i. \end{cases} \tag{4.4}$$

$G_{ij}(c)$ is the probability of generating configuration j from configuration i, and $A_{ij}(c)$ is the probability of accepting configuration j if it has been generated by configuration i. From the definition it follows that $\forall i,\ \sum_j P_{ij}(c) = 1$. $G(c)$ is usually given by the uniform distribution on the neighborhoods, $\Lambda_i$, and $A(c)$ is defined as the Metropolis criterion.

Two formulations of the model have been developed in the attempt to understand the simulated annealing algorithm [100].

(i). a homogeneous algorithm: each Markov chain is generated for a fixed value of c, and c is decreased between consecutive Markov chains.

(ii). nonhomogeneous algorithm: the value of c is decreased on each iteration.

The statistical cooling algorithm converges asymptotically, that is, $\lim_{k\to\infty} Prob(X(k) \in \Lambda_{i0}) = 1$, where $\Lambda_{i0}$ is the set of globally optimum configurations. Let q represent the vector whose i-th component is the probability that, as the number of iterations approaches infinity, the configuration is i given that initially it was any j. The vector q is called the stationary distribution of a homogeneous Markov chain. After an infinite number of iterations, the stationary distribution is the probability distribution of the configurations. An irreducible and aperiodic Markov chain always has a stationary distribution. Sometimes the word equilibrium is used for the stationary distribution of a homogeneous Markov chain. The homogeneous algorithm converges asymptotically if the following four conditions are satisfied [4, 72, 79, 86].

1. each Markov chain is of infinite length

2. A(c) and G(c) are irreducible and aperiodic.

3. $\lim_{c \to 0} q_i = \begin{cases} |\Lambda_{i0}|^{-1} & \text{if } i \in \Lambda_{i0} \\ 0 & \text{elsewhere} \end{cases},$

   where $q_i$ is the i-th component of the stationary distribution.

   In other words, the vector $q$ converges to the uniform distribution on the set of globally minimal configurations

4. $\lim_{k \to \infty} c_k = 0,$

where $c_k$ is the value of the control parameter during the k-th Markov chain. A Markov chain is <u>irreducible</u> if and only if for every pair of configurations (i,j) there is a positive probability of reaching j from i in a finite number of steps. A chain is <u>aperiodic</u> if and only if for all configurations $i \in \Lambda$, the greatest common divisor of all integers $n \geq 1$, such that $(P^n)_{ii} > 0$ is equal to 1 [55, 100].

For the nonhomogeneous model, let each element of the acceptance matrix be defined by

$$A_{ij}(c_k) = \min\{1, \exp(-(C(j) - C(i))/c)\}, \tag{4.5}$$

and assume that the sequence of control parameters $(c_k)$, $k = 0, 1, 2, \ldots$ has the two properties:

$$\begin{aligned} &\lim_{k \to \infty} c_k = 0; \\ &c_k \geq c_{k+1} \quad k = 0, 1, 2, \ldots \end{aligned} \tag{4.6}$$

Mitra et al. [77] used ergodicity theorems on nonhomogeneous Markov chains to show that under certain conditions on the acceptance matrix $A(c_k)$, the rate of convergence of $\{c_k\}$ must be $\leq \frac{\Gamma}{\log k}$ for some constant $\Gamma$. Gidas and Hajek also gave arguments in favor of this bound [43, 46]. Building on this work, Gelfand and Mitter [41] gave sufficient conditions for asymptotic convergence of the algorithm to any given set of configurations. In 1988, Hajek [47] was able to give necessary and sufficient conditions for convergence to the global minimum by finding an expression for d, where d is a parameter dependent on the structure of the given problem, such that $\Gamma \geq d$. Hajek employed continuous-time nonhomogeneous Markov chains in his analysis. The expression for d gave a stronger bound on $c_k$ than had previous research.

## D. <u>COOLING SCHEDULES</u>

In the previous section it was stated that asymptotic convergence can only be approximated. Convergence to the global minimum with probability 1 in the homogeneous case requires that for each value of the control parameter, $c_k$, an infinite number of transitions must be generated. In the nonhomogeneous case, for each value $c_k$ one transition is generated and $c_k \to 0$ no faster than $O\left(\frac{1}{\log k}\right)$. Hence, any implementation produces an approximate algorithm that is not guaranteed to find the global minimum. Moreover, the convergence results mentioned in the previous section do not give much practical insight when attempting to approximate asymptotic convergence [2, 72]. Usually, simulated annealing is implemented by a sequence of homogeneous Markov chains of finite length at decreasing values of the control parameter. In that case, the following parameters must be specified.

1. the initial control parameter, $c_0$

2. the final control parameter, $c_f$ or some other termination condition

3. the length $l_k$ of the Markov chains

4. a rule for decrementing $c_k$ to find $c_{k+1}$. A choice for these four parameters is called a cooling schedule [100].

Discussion of cooling schedules usually involves the following considerations.

- *Initial value of the control parameter*

  For $c_k \rightarrow \infty$, the stationary distribution of a Markov chain is given by the uniform distribution on the set of configurations $\Lambda$. Initially, quasi-equilibrium can be established by letting $c_0$ be large enough that all transitions are accepted. Then all configurations appear with equal probability, which is associated with the uniform distribution.

- *Final value of the control parameter*

  In defining $c_f$ the objective is to terminate the algorithm if the improvement in cost to be expected by continuing execution of the algorithm is relatively small compared to the computational effort.

- *Length of Markov chains and decrement of the control parameter*

  The length $l_k$ of the $k^{th}$ Markov chain and the rule for decrementing $c_k$ are related by the notion of equilibrium. Intuitively, it is clear that large decreases in $c_k$ will make it necessary to attempt more transitions at $c_{k-1}$ in order to restore equilibrium. Thus, there is a trade-off between large decreases in the control parameter and small lengths of Markov chains. In most cases,

small decrements are chosen to avoid lengthy chains,
but it is also possible to employ long chains and
large decrements in $c_k$.

1. Simple Annealing Schedules. Kirkpatrick [62] developed a simple cooling schedule based on empirical rules rather than on theoretical analysis. The principles upon which this schedule and others similar to it are based are discussed in this section.

- *Initial value of the control parameter*

  The initial value of the control parameter $c_0$ was determined by the process: choose a large value for $c_0$ and perform some iterations. If the acceptance ratio, $\chi$, the number of accepted transitions divided by the number of attempted transitions, is less than a given value, $\chi_0$, then double the value of $c_0$. Continue until $\chi > \chi_0$ [100].

- *Final value of the control parameter*

  A stop criterion was determined either by fixing the number of homogeneous Markov chains that were to be executed by the algorithm [11], or by ending execution whenever consecutive Markov chains had identical last configurations [90].

- *Length of Markov chains*

  The length $l_k$, of the $k^{th}$ Markov chain, was a polynomial function of the problem size. Thus, $l_k$ did not depend on k [11, 15, 90].

- *Decrement of the control parameter*

The decrement in the control parameter was chosen small enough that small Markov chain lengths are sufficient to return to quasi-equilibrium after the decrement. A rule such as

$$c_k = \alpha c_k \quad k = 0, 1, 2, \dots , \tag{4.7}$$

where $\alpha$ is a constant less than, but close to, 1 is frequently applied. $\alpha$ ranges from .5 to .99 [11, 15, 68, 90].

2. <u>More Complicated Annealing Schedules</u>. More complicated cooling schedules have been developed which are based on attempts to quantify the concept of equilibrium. Some of the ideas behind those attempts are discussed next. Quantities enclosed between < and > represent the expected values of those quantities at equilibrium.

- *Initial value of the control parameter*

For $c_0$, White [105] attempts to find an initial control parameter by using the function

$$\omega(C)dC = \frac{1}{|\Lambda|} \, |\{i \in \Lambda \mid C \leq C(i) < C + dC\}| . \tag{4.8}$$

It can be assumed the $\omega(C)$ is Gaussian with mean $\overline{C}$ and standard deviation $\sigma_\infty$ [3, 46]. Then

$$\omega(C) = \exp\left( \frac{-(C-\overline{C})^2}{2\sigma_\infty^{\;2}} \right) \times \frac{1}{\sigma_\infty \sqrt{2\pi}} \tag{4.9}$$

where $\sigma_\infty^2 = \lim_{c \to \infty} < (C(c) - < C(\infty) > )^2 > \;\equiv\; \frac{1}{|\Lambda|} \sum_{i \in \Lambda} (C(i) - < C(\infty) > )^2$.

White [4.8] shows the expected value at equilibrium of $C(c)$, $< C(c) >$, is given by

$$< C(c) > = C_{\mathrm{opt}} + \sqrt{2}\,\sigma_{\infty}\left( y + \frac{\exp(-y^2)}{\sqrt{\pi}\,(1 + \mathrm{erf}(y))} \right), \tag{4.10}$$

where

$$y = \frac{1}{\sqrt{2}}\left( \frac{\overline{C} - C_{opt}}{\sigma_{\infty}} - \frac{\sigma_{\infty}}{c} \right). \tag{4.11}$$

For large values of c, equations (4.10) and (4.11) give

$$< C(c) > \simeq \overline{C} - \frac{\sigma_{\infty}{}^{2}}{c}. \tag{4.12}$$

White [4.8] proposes that $c_0$ be chosen such that
$< C(c) >$ is within one standard deviation of $\overline{C}$.
From equation (4.12) the fact that $\overline{C} - < C(c) > \leq \sigma_{\infty}$
leads to the conclusion that $c_0 \geq \sigma_{\infty}$. Assuming that c is large,

$$\sigma_{\infty}{}^{2} \simeq < C^2(\infty) > - ( < C(c) > )^2, \tag{4.13}$$

which yields

$$c_0 \geq \sqrt{< C^2(\infty) > - ( < C(\infty) > )^2}\,. \tag{4.14}$$

Values for $< C^2(\infty) >$ and $< C(\infty) >$ are found by
tabulating the expected value of the cost function when the algorithm
executes the Markov chain in which all transitions are accepted .

- *Final value of the control parameter*

A bound for $c_f$ is obtained by Lundy and Mees [72] by requiring that, for the final value of the control parameter, once the stationary distribution is reached, the probability for the current configuration to be greater than $\varepsilon$ above the global minimum should be less than $\vartheta$ for some small real number $\vartheta$.
That is, they demand that

$$Prob\{X(k) = i \wedge C(i) > C_{\text{opt}} + \varepsilon \mid c = c_f\} < \vartheta. \tag{4.15}$$

Let $\mathbf{q}(c_k)$ be the vector representing the stationary distribution.
Then the i^th^ component of $\mathbf{q}$ is given by $q_i = \lim_{k \to \infty} Prob\{X(k) = i \mid X(0) = j\}$ for arbitrary j. In the usual formulation of simulated annealing, [100],

$$q_i(c_k) = \frac{\exp(-(C(i) - C_{\text{opt}})/c_k)}{\sum_{j=1}^{\Lambda} \exp(-(C(j) - C_{\text{opt}})/c_k)}. \tag{4.16}$$

Using equation (4.16) Lundy and Mees derive the following bound.

$$\begin{aligned} Prob\{X(k) = i \wedge C(i) > C_{\text{opt}} + \varepsilon\} &\simeq \sum_{i:\, C(i) > C_{\text{opt}} + \varepsilon} q_i(c) \\ &< (|\Lambda| - 1)\exp\left(-\frac{\varepsilon}{c}\right). \end{aligned} \tag{4.17}$$

Combining (4.16 and 4.17) yields

$$c_f \leq \frac{\varepsilon}{\ln(|\Lambda| - 1) - \ln \vartheta}. \tag{4.18}$$

Some other authors [4, 79] determine a final value of the control parameter based on the decrease of the average values of the cost function during execution of the algorithm over a number of Markov chains. Let $\overline{C}(c_k)$ represent the average cost over the $k^{th}$ Markov chain. Then execution halts whenever $\overline{C}(c_k) - C_{opt}$ is small. If $c_k \ll 1$, then

$$\overline{C}(c_k) - C_{\rm opt} \simeq c_k \left( \frac{\partial \overline{C}(c)}{\partial c} \right)_{c=c_k} , \tag{4.19}$$

so that

$$\left( \frac{\partial \overline{C}(c)}{\partial c} \right)_{c=c_f} \times \frac{c_f}{\overline{C}(c_0)} < \varepsilon \tag{4.20}$$

for a small real number $\varepsilon$ [4] or

$$\left( \frac{\partial \overline{C}(c)}{\partial c} \right)_{c=c_f} \times \frac{c_f}{(\overline{C}(c_0) - \overline{C}(c_f))} < \varepsilon \tag{4.21}$$

[79].

Recently Otten and Van Ginneken [80] observed the behavior of the variance of the cost function over homogeneous Markov chains in a number of applications, and they found a critical value existed for the control parameter. Initially, the average cost follows a hyperbolic decline as temperature ( the control parameter ) decreases. This is the behavior to be expected when the density of the costs over the set of configurations is Gaussian [45]. Over a small range of the control parameter, the pattern changes so that the average cost approaches a linear decrease. For high values of the temperature, $c_k$, the standard deviation remains constant at its initial value $\sigma_\infty$. When the average cost is linearly related to the temperature, the standard deviation is proportional to the temperature [80]. The critical value, $T_c$, of the temperature is found by

$$T_c = \sigma_\infty \min\left[ \frac{\sigma(c_k)}{c_k} \right]. \tag{4.22}$$

To actually calculate $T_c$, using this formula, it is necessary to first obtain an estimate for $\sigma_\infty$ by observing the variance during the first few Markov chains executed by the algorithm. Then the values for $\sigma$ for each value of $c_k$ are calculated. An estimate for $T_c$ for each Markov chain is the abscissa of the intersection of the line of $\sigma_\infty$, and the line connecting the origin and $(c_k, \sigma(c_k))$. The slope of the line connecting through the origin should be nearly constant after reaching values of $c_k < T_c$. The point $T_c$ is already close to the global minimum for a process in equilibrium. Otten and van Ginneken recommend halting the algorithm soon after $c_k$ becomes less than $T_c$ [80].

- *Length of Markov chains*

Exact measurement of quasi-equilibrium is not possible because an overwhelming amount of statistical data would have to be collected in order to determine the probability distribution of the configurations. In order to obtain a final configuration approximately equal to the global optimum, it should always be possible to leave any configuration found during the execution of the algorithm [86]. For any configuration i, the expected number of transitions needed to leave i, $< L_i(c) > = (1 - P_{ii}(c))^{-1}$.

However,

$$P_{ii}(c) = 1 - \frac{1}{|\Lambda|} \sum_{j \in \Lambda_i} \min\{1, \exp(-(C(j) - C(i))/c)\}. \qquad (4.23)$$

Thus, evaluation of $P_{ii}$, $\forall i \in \Lambda$, necessitates knowing the values of the cost function for all configurations. Since it is of interest only to know the $\max_{i \in \Lambda} < L_i(c) >$, or, equivalently, $\max_{i \in \Lambda} P_{ii}(c)$, an estimate can be made by approximating $C(j) - C(i)$, $\forall i$ and $j$, by $C(j') - C(i')$, where $i'$ and $j'$ are the configurations with the lowest and highest costs, respectively, found so far during the execution of the algorithm. Hence,

$$\max_{i \in \Lambda} < L_i(c) > \simeq ( \exp( -(C(j') - C(i'))/c))^{-1}. \qquad (4.24)$$

Romeo and Sangiovanni-Vincentelli [86] propose that the Markov chain length be proportional to $\max_{i \in \Lambda} < L_i(c) >$.

- *Decrement of the control parameter*

Decrements, as previously mentioned, are chosen small in order to avoid the need for long chains for re-establishing equilibrium at each new value of the control parameter. Several authors suggest that for consecutive values of the control parameter,

$$\forall i \in \Lambda : \frac{1}{1+\delta} < \frac{q_i(c_k)}{q_i(c_{k+1})} < 1 + \sigma, \ k = 0, 1, 2, \ldots \tag{4.25}$$

for some small $\delta \in \mathbb{R}$, where $q_i(c_k)$
is the i-th component of the equilibrium vector for
the k-th Markov chain [4, 72, 79].
If Expression (4.25) holds, then the probability distribution of
the configurations should rapidly approach a new stationary
distribution after the control parameter is changed
assuming that equilibrium existed at the previous value
of the control parameter. From this starting point the
different authors derive a number of alternative
decrement rules:

1. $$c_{k+1} = c_k \left( \frac{1 + \ln(1+\delta) c_k}{3\sigma(c_k)} \right)^{-1} [4], \tag{4.26}$$

2. $$c_{k+1} = c_k \left( 1 + \frac{\gamma c_k}{U} \right)^{-1}, \tag{4.27}$$

where $U$ is an upper bound on $(C(i) - C_{opt})$ and $\gamma$
is a small real number [72].

3. $$c_{k+1} = c_k - \frac{1}{H_k} \times \frac{c_k^{\ 3}}{(\sigma(c_k))^2}, \tag{4.28}$$

where $H_k = \dfrac{C_{max} + c_k \ln(1+\delta)}{\sigma(c_k)^2 \ln(1+\delta)} c_k$ [79].

Another decrement rule developed by Huang et al. [57]
is based on the average cost of succeeding Markov chains.

Let $\overline{C}(c_k)$ again denote the average cost value for the $k^{th}$ Markov chain, and let the expected cost at equilibrium be denoted by $< C(c) >$. The following equation is satisfied [100].

$$\frac{\partial < C(c) >}{\partial \ln c} = \frac{\sigma^2(c)}{c}. \tag{4.29}$$

Approximating $< C(c_k) >$ by $\overline{C}(c_k)$, in equation (4.29) results in

$$\frac{\overline{C}(c_{k+1}) - \overline{C}(c_k)}{\ln c_{k+1} - \ln c_k} \simeq \frac{\sigma^2(c_k)}{c_{k+1}}. \tag{4.30}$$

Hence,

$$c_{k+1} = c_k \exp\left( \frac{c_k(\overline{C}(c_{k+1}) - \overline{C}(c_k))}{\sigma^2(c_k)} \right). \tag{4.31}$$

To reach equilibrium Huang et al. [57] require that $\overline{C}(c_{k+1}) - \overline{C}(c_k) = -\lambda\sigma(c_k)$ for $\lambda \leq 1$. This yields the decrement rule

$$c_{k+1} = c_k \exp\left( -\frac{\lambda c_k}{\sigma(c_k)} \right). \tag{4.32}$$

## E. PERFORMANCE

There are two common measures of performance of a heuristic algorithm: the quality of the final solution as measured by the difference in cost between the final solution and the global minimum, and the running time of the algorithm. For simulated annealing the results of these measurements depend on both the problem instance and the cooling schedule. With traditional deterministic algorithms a worst-case analysis or average-case analysis of quality and running time for a given probability distribution of problem instances is attempted [81]. Since simulated annealing is itself a probabilistic algorithm, an analysis of it involves not only the probability distribution of problem instances, but also the probability distribution over the set of possible solutions for a given problem instance.

Theoretical analysis of annealing so far has dealt with only worst-case results [4, 41, 43, 72, 77,]. Both Aarts and van Laarhoven and Lundy and Mees [4, 72] showed that, using their respective decrement rules (4.26) and (4.27) and termination criteria (4.21) and (4.18), the number of steps in the value of the control parameter is $O(\ln |\Lambda|)$. Combining this with a fixed length Markov chain of length $|\Lambda_i|$, the size of a neighborhood, provides a worst case expression for the total number of transitions generated by the algorithm:

$$\text{Total transitions} = O(|\Lambda_i| \ln(|\Lambda|)). \tag{4.33}$$

For most combinatorial optimization problems, the neighborhoods can be chosen to have a size that is a polynomial function of the system parameters and $|\Lambda|$ is exponential in the size of the problem input [81]. Hence, equation (4.33) states that the execution of the algorithm takes polynomial time for most problems.

Also, upper bounds can be given for the closeness of the probability distribution of the set of configurations after generation of a finite number of transitions to the uniform probability distribution on the set of optimal configurations [43, 77]. These upper bounds are generally poor. For example, using the bound given by Mitra [77], it is often true that the number of iterations required for good accuracy is greater than the number of configurations. An upper bound on the quality of the final solution is known only for the maximum matching problem which is in the class P [89].

An empirical analysis of simulated annealing, that is, solving many instances of a variety of combinatorial optimization problems with different cooling schedules, can be made on the basis of the numerous problems to which annealing has been applied(see Section F). The findings thus far suggest the following conclusions.

(a). The performance of the algorithm, especially the quality of the solution obtained, depends heavily on the chosen cooling schedule.

(b). With a suitable cooling schedule near-optimal solutions can be found in polynomial running time for numerous diverse combinatorial optimization problems.

(c). Computation times for simulated annealing can be much longer than those for some heuristics tailored to particular applications. For example, CPU times of more than 84 hours on a VAX 11/780 computer working on a large placement problem have been reported. The quality of the solution, however, was 66% better than the one found by the previous algorithm [90].

Another way to evaluate the annealing algorithm is by a probabilistic value analysis in which instances of particular problems are randomly generated and the behavior of the algorithm is analyzed as the problem size goes to infinity. Interest in this kind of analysis arises for two reasons: firstly, by using some of the quantities based on the analogy between combinatorial optimization and statistical mechanics, it is frequently possible to derive mathematical expressions for $\frac{C_{opt}}{\sqrt{n}}$, where n is the size of the input; secondly, using the expressions just mentioned, the cost of the solution obtained by simulated annealing can be compared with the globally minimal cost [11, 12].

In summary from the various methods of performance evaluation, a few preliminary statements about simulated annealing seem justified.

1. The quality of the solutions from simulated annealing is at least as good and often better than that obtained by previous algorithms for most problems.

2. Simulated annealing can be applied to many problems for which no other algorithm exists.

3. Parallel execution of simulated annealing might greatly reduce computation times. The use of H processors theoretically reduces total computation time by a factor of H with the same quality of solution as the sequential algorithm.

To increase the speed of the statistical cooling algorithm, a number of researchers have implemented parallel versions of the algorithm [1, 2, 14, 17]. This is not an easy task since transitions are carried out in a sequential manner in simulated annealing. Nevertheless, some general parallel algorithms have been constructed [1, 17], as well as some parallel algorithms for specific problems [14, 87, 90]. Most of the tailored algorithms are for VLSI layout problems using the TimberWolf package [87, 90]. There are three approaches that have so far been tried in constructing general parallel simulated annealing algorithms.

The first approach, called the systolic algorithm, features the assignment of a Markov chain to each available processor and uses equal length Markov chains [1]. The chains are executed in parallel and information is transferred from one chain to the succeeding one. Each Markov chain is divided into sub-chains equal to the number of processors. Execution of Markov chain H is started whenever execution of the first sub-chain of Markov chain H − 1 is finished. Exchange of information between processors preserves quasi-equilibrium. For example, if processor i is working on Markov chain K, the initial configuration and value of the control parameter arrive from processor i − 1, working on Markov chain H − 1. Then a sub-chain is generated, and after this sub-chain is finished, there are two sets of data that can be used to continue the execution of chain K: the present configuration and value of c, and the configuration and value of processor i − 1, after the completion of its second sub-chain (Figure 5) [1].

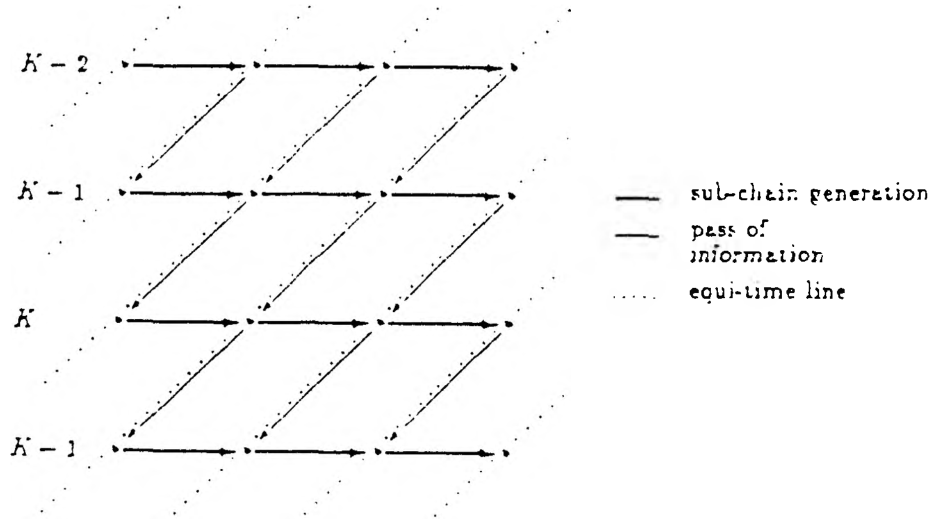

Figure 5. Structure of the Systolic Algorithm

A probabilistic choice is made between these two data sets. It is assumed that quasi-equilibrium is preserved even though execution of the next Markov chain begins before execution of the previous one is completed. In this implementation the processors each have about the same amount of work to do and there is little communication among processors.

The second approach, called the clustered algorithm, enlists all processors in a cooperative effort to generate the same Markov chain(Figure 6) [1]. For high values of c this is inefficient because nearly all transitions are accepted and accepted transitions must be dealt with consecutively, because each accepted transition necessitates an update of the current configuration and two updates cannot be done at once. While an update is occurring all processors must halt until the system is adapted to the new configuration. As c decreases, however, the use of the processors becomes more efficient. To make the algorithm more effective for large values of c, all processors generate separately a sub-chain of the same Markov chain. In this way the activities of each processor do not interfere with each other and a linear speedup may occur. Let the update ratio for the parallel simulated annealing algorithm be defined by $\chi(c) = \frac{t_{acc}}{t_{att}}$, where $t_{acc}$ is the number of accepted transitions and $t_{att}$ is the number of attempted transitions. Given H processors, the algorithm begins by having each processor evaluate a sub-chain. At some point in the cooling process, determined by estimating the ratio of the computation times to obtain a given update ratio for the clustered parallel algorithm and the sequential algorithm respectively, the processors are clustered and the sub-chains enlarged. Each enlarged sub-chain is now evaluated by a cluster of processors. This continues until all processors are in one cluster evaluating a whole Markov chain. Usually, the clusters are formed by combining two of the previously defined clusters and it is assumed that $|H| = 2^r$ for some positive integer r. After completion of this computation, a probabilistic decision is made to

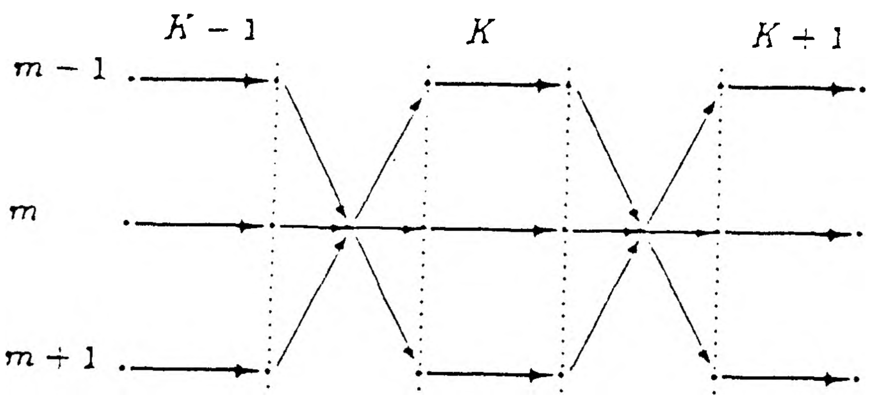

Figure 6. Structure of the Clustered Algorithm

determine which of the H available configurations should be used to start the succeeding sub-chain [1].

The third approach, called the dynamic decision tree algorithm, maps the simulated annealing procedure onto a dynamically structured binary tree of processors(Figure 7) [17]. It maintains the same move sequence as the sequential simulated annealing algorithm, which permits this version of parallel simulated annealing to avoid move conflicts and poor acceptance/rejection decisions. Because the decision to accept or reject a transition is binary, the simulated annealing algorithm can be viewed as choosing a particular path through a decision tree. In Figure 7 , each of the nodes is associated with a processor that performs a move/evaluate/decide task. The two children correspond to the two possible decision results. Assume there are communication paths between processors so that paths between decision nodes in the tree correspond to communication paths between processors. It is assumed that the

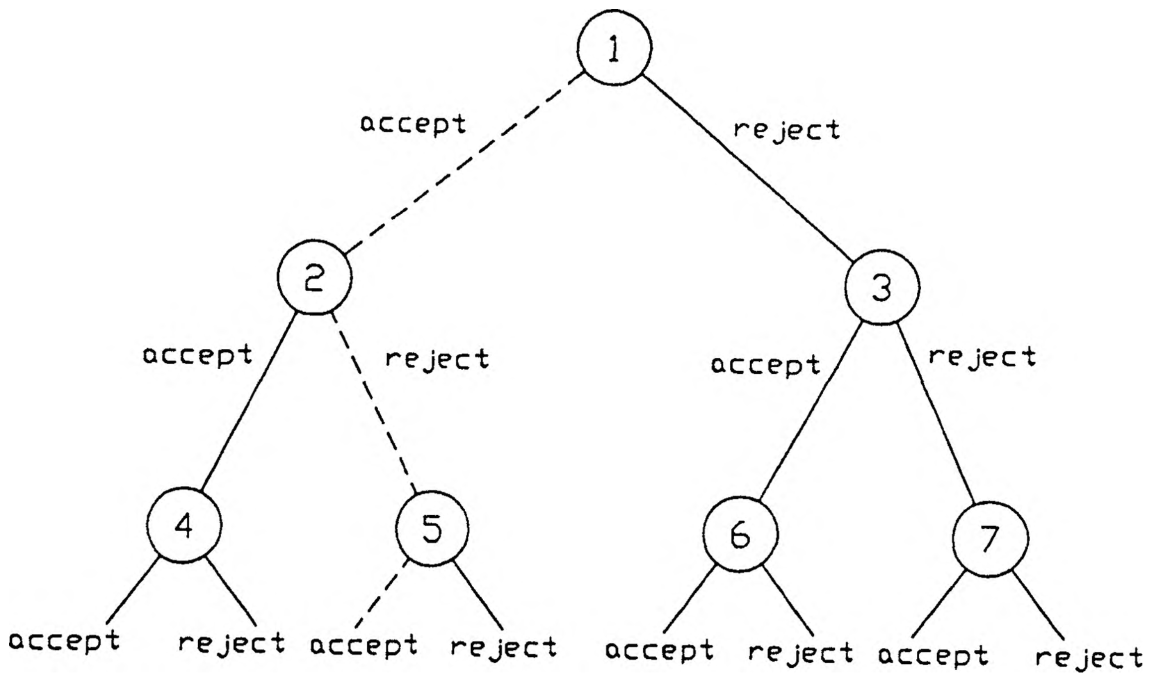

Figure 7. Structure of the Decision Tree Algorithm

time required for processor to processor communication is small compared to the time needed for move/evaluate/decide tasks. It is also assumed that each processor has enough memory to hold the system state and that the processors operate in a MIMD mode. Speedup of simulated annealing is engineered by doing as many move/evaluate/decide tasks as possible in parallel. Each of these tasks is given to a different processor. Two features of annealing are exploited for implementation of parallelism.

1. a processor along a reject path can start executing a move/evaluate/decide task as soon as its parent node has

indicated start of the process. No state information needs to be communicated.

2. a processor along an accept path can start executing a move/evaluate/decide task whenever it has been informed of the move of its parent.

The third approach to a parallel annealing algorithm starts by having the sequence of nodes initiate tasks down the decision tree. The procedure for determining the final path begins when node 1 communicates its decision to its selected child. After this child has finished its decision phase, it communicates this to its selected child. After a finite number of steps a leaf in the tree is reached. Since for a given value of c, hundreds of iterations may be needed, making the number of nodes larger than the conceivable number of processors, the processors must be dynamically reassigned. Chamberlain et al. [17] present three versions for doing this.

Numerical results indicate that the first approach becomes less desirable as the number of processors increases. The reason is that the length of the sub-chains decreases to the point where equilibrium can no longer be maintained [1]. The second approach theoretically reduces the total computation time by a factor H with no loss in the quality of the solution from that of the sequential algorithm whenever H processors are used. However, there is a growing likelihood of delays due to communication problems as the number of processors is increases. Experiments with the third approach to parallelizing simulated annealing indicate speedups of between $\log_2 H$ and $\frac{(H + \log_2 H)}{2}$ over the running times of the sequential algorithm [17].

Special dedicated hardware implementations of simulated annealing have also been presented [4, 54]. These implementations utilize massive parallelism similar to that of neural networks [56]. The Boltzmann machine [54] is one of the most often mentioned new architectures that have been used for this purpose. For two distinct 0-1 integer programming formulations of the Traveling Salesman Problem, Aarts and Korst [4] showed that near optimal solutions could be found by mapping the corresponding binary variables onto discrete computing elements and making the cost functions into the consensus function of the Boltzmann machine. Hinton and Sejnowski [54] have developed an algorithm for the Boltzmann machine which attempts to solve computer vision and pattern recognition problems by means of simulated annealing.

## F. APPLICATIONS

Simulated annealing has been used effectively in solving many optimization problems in the area of design automation for VLSI circuits. These include placement, floorplan design, routing, PLA folding, gate matrix layout, logic array optimization, logic minimization, testing, transistor sizing, and digital filter design [111]. Most of these applications use the TimberWolf package, an integrated set of placement and routing optimization programs which employ the simulated annealing technique [90, 111]. The most common graph theory problem to which annealing has been applied is the Traveling Salesman Problem [3, 11, 88, 101]. The results of these studies indicate that simulated annealing is competitive with the best iterative improvement algorithms for this problem. The graph partitioning problem, an NP-complete problem, is another topic from graph theory on which progress has been made through annealing [36, 60]. Near optimal solutions for problems with up to 200 vertices have been reported using the statistical mechanics approach. Research with annealing has also been done in the areas of biology [71], magnetics [73], code design [30] and game theory [106].

An example of a cooling schedule applied to a problem is the procedure used by Johnson et al. [60] to solve the graph partitioning problem. In this problem a graph $G(V, E)$ is given and the goal is to partition the set of vertices $V$ into two subsets $V_1$ and $V_2$ of the same cardinality such that the difference in the number of edges with one endpoint in $V_1$ and the other in $V_2$ is as small as possible. Let E represent the set of edges with endpoints in different subsets. Then the cost of a partition $(V_1, V_2)$ is given by

$$C(V_1, V_2) = |\mathrm{E}| + \lambda(|V_1| - |V_2|)^2, \tag{4.34}$$

where $\lambda$ is a weighting factor. This is the cost of a solution to the problem. The solution is not feasible unless $V_1$ and $V_2$ have the same number of elements, but partitions with $|V_1| \neq |V_2|$ may appear as intermediate solutions. The purpose of the weighting factor is to penalize partitions for which $|V_1|$ is not equal to $|V_2|$. The technique of using penalty functions is often useful because it allows additional ways to escape local optima. Two partitions are defined to be neighbors if one can be obtained from the other by moving exactly one vertex from one of its sets to the other. Initial solutions are constructed by flipping an unbiased coin for each vertex in order to determine whether that vertex should be placed in $V_1$ or $V_2$.

In order to obtain the initial control parameter $c_0$ the following equation is used,

$$c_0 = \frac{\overline{\Delta C}^{(+)}}{\ln(\chi_0^{-1})}, \tag{4.35}$$

where $\chi_0$ is a given value of the ratio of accepted transitions to generated transitions. The notation $\overline{\Delta C}^{(+)}$ represents the average increase in cost for a number of random transitions.

The control parameter is decremented according to equation (4.7) with $\alpha = .95$. The length of the Markov chains is a multiple of the size of the neighborhoods. This length in turn is found by requiring a minimum number of accepted transitions at each temperature. The terminating rule is that no improvements in the cost are discovered during a fixed number of consecutive Markov chains.

With this cooling schedule Johnson et al. [60] found that simulated annealing gave solutions for most graphs that were more accurate that those derived with the best of the more traditional heuristic algorithms.

# V. TAILORED COOLING ALGORITHM

Many researchers have noticed that simulated annealing takes a large amount of running time in order to perform well, and this may make it infeasible for some applications. In addition, even if the necessary computational time is available, annealing may or may not provide a substantial improvement over the results of other algorithms for the same problem. Thus, in order to compare annealing with other methods for a particular problem, it is necessary to tailor the parameters of the cooling algorithm to obtain as efficient an implementation as possible.

In this dissertation two methods are compared for adapting the annealing algorithm to the Directed Steiner Problem. In this chapter, the four generic parameters discussed in the previous chapter are adjusted for the first method. A series of experiments is performed on a test bed of graphs for this purpose. The algorithm that is obtained is referred to as the tailored cooling algorithm (TCA) throughout this dissertation. In the next chapter, experiments are performed on the second method, the adaptive algorithm of Aarts and van Laarhoven [4], in order to adjust the parameters, $\delta$ and $\varepsilon$, of their general scheme. This method is hereafter referred to as the dynamic cooling algorithm (DCA).

## A. GENERIC IMPLEMENTATION DETAILS

The following generic simulated annealing procedure, similar to the one employed by Johnson et al. [60], was used in the tailored cooling algorithm.

```
1. Call INITIAL-COST() to generate a random initial configuration F
        and return $cost(F)$. Let this cost be an initial upper
        bound denoted by $cost^*$.

2. Choose an initial control parameter, $T > 0$ so that the
        ratio $\zeta = upchanges/uphills$, where $upchanges$ is the number
        of accepted increases in cost, and $uphills$ is the number of
        generated increases in cost, begins approximately equal
        to INITIAL_PROB.

3. Set $endctr = 0$.

4. While ($endctr < coldlimit$) do:

        A. Set $trials = upchanges = uphills = 0$.

        B. While ($trials < CHAINFACTOR * (N - SPECIAL)$) do:

            (i). $trials = trials + 1$.
            (ii). Call GENERATE() to generate a neighbor $F'$ of F
                  and return $cost(F')$.
            (iii). Let $\Delta = cost(F') - cost(F)$.
            (iv). If $\Delta \leq 0$ then do:
                (a). Set $cost(F) = cost(F')$.
                (b). If $cost(F') < cost^*$, then set $cost^* = cost(F')$.
            (v). If $\Delta > 0$ then do:
                (a). $uphills = uphills + 1$.
                (b). Choose a random number r such that $0 \leq r < 1$.
                (c). If $r \leq \exp\left(-\frac{\Delta}{T}\right)$, then:

                        set $upchanges = upchanges + 1$,
                        set $cost(F) = cost(F')$,
                        and if $cost(F') < cost^*$, then set $cost^* = cost(F')$.

        C. Set $T = TEMPFACTOR * T$ in order to decrease the temperature.
            If the lowest cost so far found, $cost^*$, was changed during B.,
            then $endctr = 0$.
            If $\zeta < MINRATIO$, then $coldlimit = coldlimit + 1$.

5. Output the optimal configuration $F^*$.
```

The parameters in the above algorithm have the following meanings.

| | |
|---|---|
| INITIAL_PROB | The initial probability that an increase in cost is accepted. It is used to determine an initial control parameter(temperature). |
| N - SPECIAL | The neighborhood size. Special is the number of vertices that must be connected. N is the number of vertices. |
| CHAINFACTOR | The number of iterations at a given temperature is CHAINFACTOR * (N - SPECIAL). Thus, the number of iterations in a Markov chain is proportional to the number of neighbors regardless of the graph size. |
| TEMPFACTOR | This is the factor by which the cooling factor is reduced on succeeding Markov chains. |
| MINRATIO | This value is used to determine whether the ground state of the annealing run has been reached. If this has happened, then the algorithm is terminated. A counter called endctr is incremented by one each time a Markov chain is completed where $\zeta$ is less than or equal to MINRATIO, and it is set to zero each time a cost is found that improves on the previous optimal cost, $cost^*$. If endctr ever reaches coldlimit, then the annealing is finished. |

## B. SPECIFIC IMPLEMENTATION DETAILS FOR DSP

Let $S$ be the set of special nodes that must be joined in the Steiner Minimal Tree. For the annealing scheme, a solution is any partition $N - S = V_1 \bigcup V_2$ of the nonspecial nodes into the set $V_1$ of those nodes used in the Steiner Minimal Tree and the set $V_2$ of nodes that do not appear in the Steiner Minimal Tree. Two partitions are neighbors if one can be obtained from the other by moving a single vertex from one of its sets to the other. The cost of a partition $(V_1, V_2)$ is defined to be the cost of the minimum spanning tree of the graph $G'(S \bigcup V_1, A')$, where $A'$ is the set of arcs in the given graph $G(N, A)$ between nodes of $S \bigcup V_1$, after the leaves containing nodes of $V_1$ have been pruned.

Initial solutions are obtained by generating a random partition. This is done by flipping an unbiased coin for each vertex in $N/S$ to determine whether it should be in $V_1$ or $V_2$. The specification of the tailored cooling algorithm is now complete except for the assignment of values to the parameters. In order to make these assignments as efficient as possible, experiments were performed on a number of random graphs.

1. <u>The Experimental Sample of Graphs for Optimizing Parameters</u>. Randomly generated graphs were used in the experiments. They are defined in terms of three parameters, n, $p_e$, and special. The parameters represent the number of vertices, the probability that any two arbitrary vertices determine an edge, and the number of special vertices, respectively. Each test problem was generated in the following way: first a random directed spanning tree for the entire set of nodes, N, was generated in order to guarantee a connected network and hence a feasible solution to the DSP. Then additional arcs were randomly generated according to the value of $p_e$. The costs were randomly selected from the (0,1) interval.

In attempting to optimize parameters for an annealing implementation, it is essential to keep in mind that the conclusions are not necessarily applicable to every annealing implementation, nor are they even applicable to this one if the graphs are of a different size or of a particular structure different from the random graphs studied here. The experiments for both TCA and DCA were performed on graphs with the following set of parameters: $n = 20, 40, 60, 80$ ; $p_e = 0.05, 0.25, 0.50,$ and $0.90$ ; special = $\frac{n}{4}, \frac{n}{2}$ and $\frac{3n}{4}$. Ten graphs using each combination of values were generated, 480 graphs in all. In some experiments, multiple runs of annealing for some of these graphs were performed, each starting from a different initial configuration.

Since there were too many combinations of variables for the tailored cooling algorithm to consider all combinations of values, attention was focused on one or two factors at a time in order to isolate their effects. Although the experiments were not extensive, they at least address some of the questions that must be asked in order to optimize an annealing implementation, and they are at least as thorough as those done in other current research into simulated annealing [60].

2. Initialization and Termination Parameters. The first parameter that is considered is the initial temperature. A value is chosen so that the ratio $\zeta$ of accepted uphill moves to generated uphill moves is approximately equal to INITIAL_PROB. Thus, a choice for INITIAL_PROB must be made. The reason for using the ratio of accepted uphill moves over generated uphill moves rather than the ratio of accepted moves over generated moves is that often there are many partitions of the set $N/S$ which have the same cost, thus inflating the number of accepted moves. This occurs because many of the minimum spanning trees for partitions have the same nodes of $V_1$ after leaves containing Steiner nodes are subsequently pruned. The same problem occurs with the termination procedure. Thus, the ratio $\zeta$ is also used there.

Figure 8 shows the evolution of an annealing run on one of the standard random graphs with 60 vertices, $p_e = 0.05$, and $special = 30$. This graph is hereafter referred to as $G_{60}$. The parameters used were INITIAL_PROB = 0.90, CHAINFACTOR = 16, TEMPFACTOR = .9500, MINRATIO = 0.01, and coldlimit = 10. During the run, the mean value of the cost was recorded for each temperature. These values are plotted against the number of chainlengths. In this case a chainlength is 432 iterations. Hence the temperature decreases from left to right. From the figure it is evident that little progress is made at the end of the

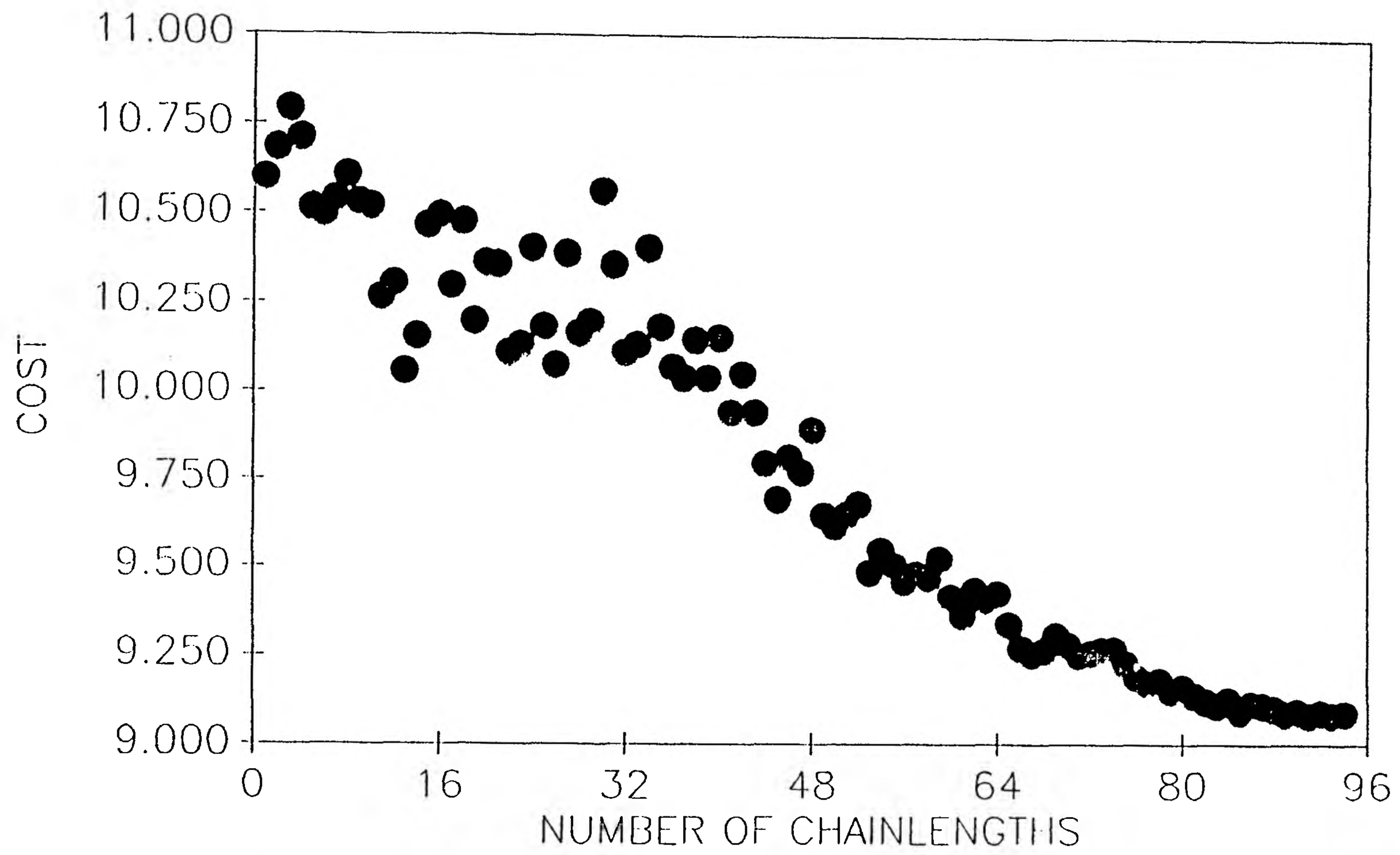

Figure 8. The evolution of the solution during annealing on a graph with 60 nodes, 30 Steiner nodes, and $p_e = 0.05$. Time increases(thus temperature decreases) along the X-axis. The Y-axis measures the current cost.

annealing schedule. Also, the time at the start of the schedule is of questionable merit.

It is necessary to determine a value of INITIAL_PROB that leads to a temperature small enough to eliminate the unnecessary time at the beginning of the annealing schedule but large enough to guarantee a solution as accurate as that given by the extended schedule. To find such a value for the DSP a battery of experiments was performed. One graph with each combination of values for $n, p_r$, and *special*, 48 graphs in all, was selected. Ten runs were performed for each graph for each of nine different values of INITIAL_PROB from 0.1 to 0.9. The other parameters were fixed as follows: $MINRATIO = 0.02$, $CHAINFACTOR = 16$, and $TEMPFACTOR = .9500$. Figures 9 and 10 contain the results for $G_{60}$. In the figures, the circles represent the average cost or average running time for a set of 10 runs with the indicated value of INITIAL_PROB. The bars show the distance of one standard deviation above and below the mean. Similar relations among INITIAL_PROB, final cost, and running time were obtained for the other 47 graphs in the test ensemble. The values of INITIAL_PROB from .3 to .9 give approximately the same degree of accuracy for all of the graphs. Running time, on the other hand, grows quickly as INITIAL_PROB increases. To keep running time as low as possible without diminishing the quality of the solution, INITIAL_PROB was assigned the value .3 for the rest of the work with TCA.

The long tail at the end of Figure 8 results from the low value of 0.01 for MINRATIO, and from the large value of 10 for coldlimit. The length of the tail was reduced by giving coldlimit the value of 5, and conducting some experiments to find the appropriate value for MINRATIO. It is important to keep MINRATIO high enough that time is not wasted at the end of the cooling schedule, while at the

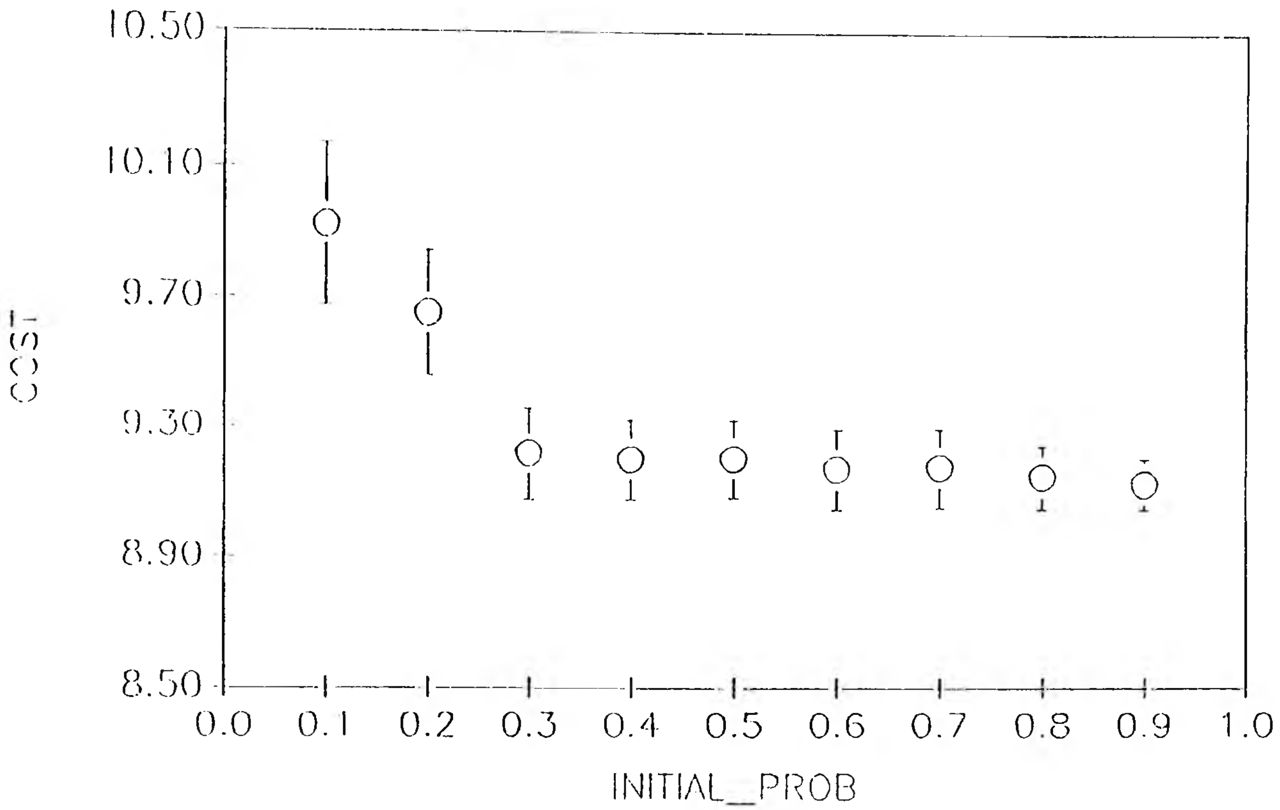

Figure 9. The Effect of INITIAL_PROB on Cost for $G_{60}$.

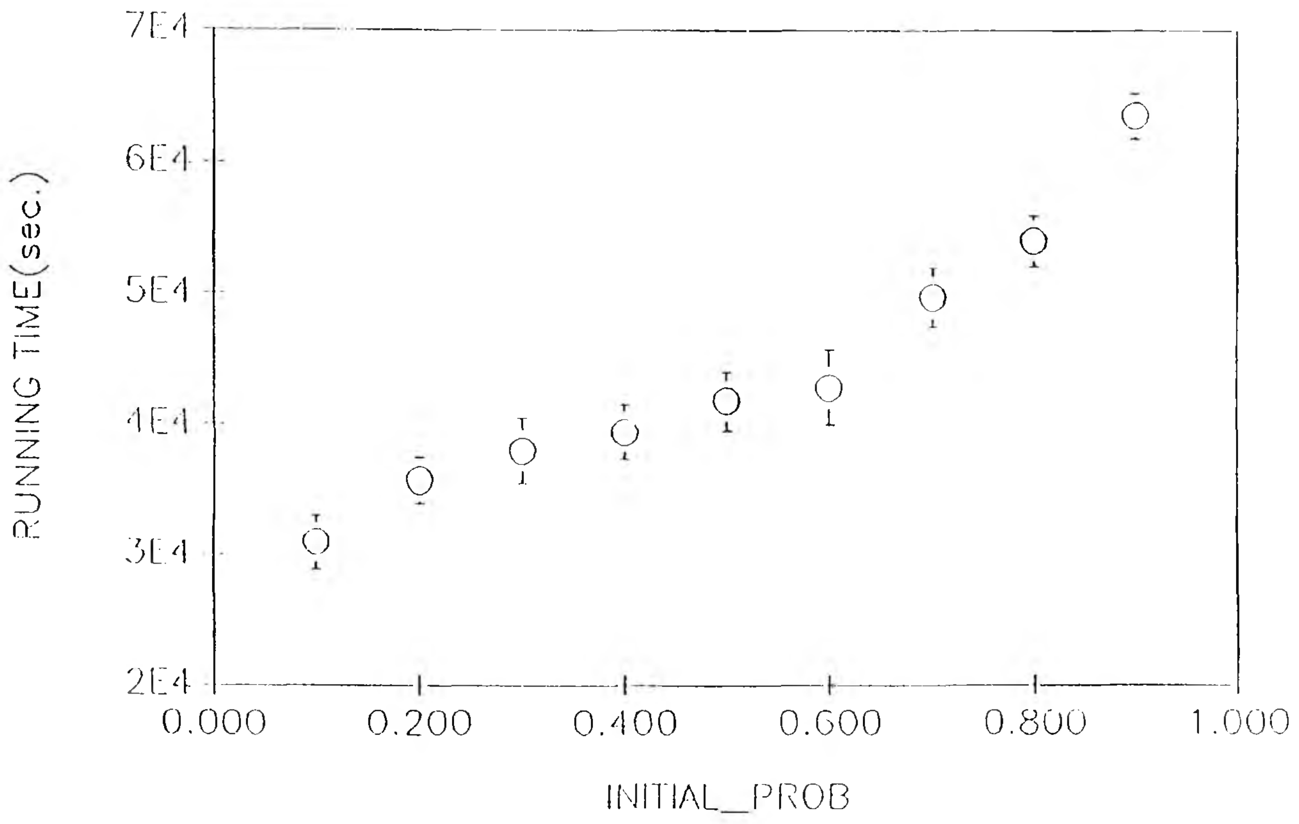

Figure 10. The Effect of INITIAL_PROB on Running Time for $G_{60}$.

same time, preserving convergence to a near optimal solution. Since the purpose of this study is to output the best solution rather than the last, it is not crucial that the schedule converge to the optimum cost. On the other hand, if the process is stopped too early, the near-optimum cost may not ever be attained.

Table II. COMPARISON OF DIFFERENT VALUES OF MINRATIO ON $G_{60}$ FOR THE TAILORED COOLING ALGORITHM

| MINRATIO | Average Running Time | Average Answer/Opt.Cost |
|---|---|---|
| 0.01 | 44586 sec. | 1.002 |
| 0.02 | 39723 sec. | 1.004. |
| 0.03 | 38552 sec. | 1.018 |
| 0.04 | 37664 sec. | 1.024 |
| 0.05 | 36905 sec. | 1.026 |
| 0.06 | 35697 sec. | 1.037 |
| 0.07 | 33718 sec. | 1.036 |
| 0.08 | 32126 sec. | 1.049 |
| 0.09 | 31272 sec. | 1.053 |

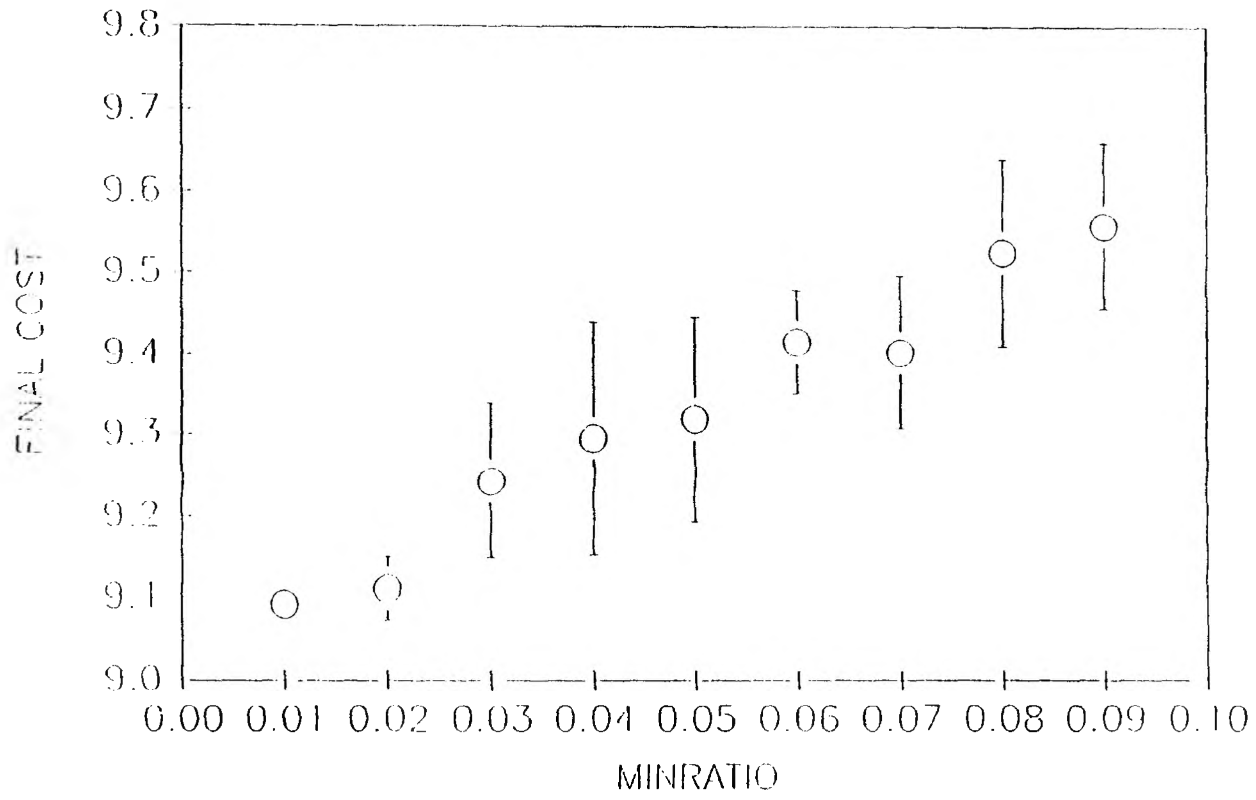

Figure 11. The Effect of MINRATIO on Cost for $G_{60}$.

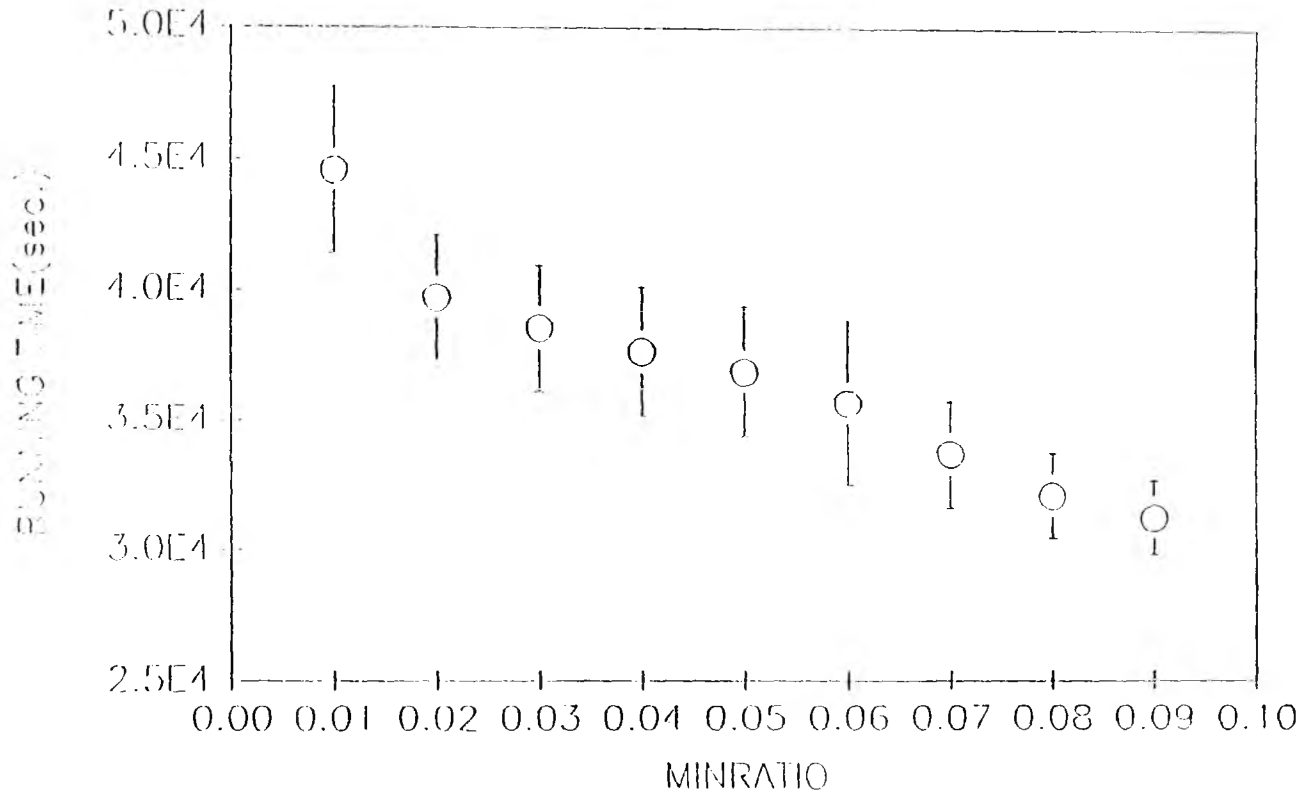

Figure 12. The Effect of MINRATIO on Running Time for $G_{60}$.

Experiments similar to the ones for INITIAL_PROB were applied to find a value for MINRATIO that satisfied the needs of this implementation. Ten runs were performed on each of the 48 graphs mentioned in the discussion of INITIAL_PROB for each of nine different values of MINRATIO from .01 to .09. The fixed parameters were assigned the values $INITIAL\_PROB = 0.9$, $CHAINFACTOR = 16$, $TEMPFACTOR = .9500$, and $coldlimit = 5$. Table II presents the information generated by the test runs for $G_{60}$ . Similar relations were obtained for the other graphs in the test bed. Figures 11 and 12 show the graphs relating MINRATIO to the final solution derived by annealing and total running time, respectively. In most cases, the final solution began to deteriorate quickly after MINRATIO reached .03. Also, running time was approximately the same for values of MINRATIO from .02 to .06. As with INITIAL_PROB, the aim was to keep the running time as short as possible without sacrificing accuracy. With this in mind, MINRATIO was assigned the value .02.

3. Tempfactor and Chainfactor. The remaining parameters in the general form of the annealing algorithm are tempfactor and chainfactor. These two parameters together determine how much time is taken in cooling from the starting temperature to the final one. The other parameters were fixed at INITIAL_PROB = 0.30, MINRATIO = 0.02, and coldlimit = 5. Tempfactor and chainfactor assumed all combinations of values from { 0.4401, 0.6634, 0.8145, 0.9025, 0.9500, 0.9747, 0.9873 } and { 0.5, 1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024 }, respectively. Each value of tempfactor is the square root of the value to its left. Increasing either parameter to the next larger one should approximately double the running time. The averages presented in Table III and Table IV emerged from running the TCA 10 times for each combination of values on $G_{60}$. Similar results were also obtained for two graphs with 40 and 80 nodes, respectively.

Table III illustrates the effects of the parameters on running time. The prediction that doubling chainfactor doubles the running time seems to be correct from the table, but as tempfactor is replaced by its square root, cooling time only increases by a factor of approximately 1.75. The cause of this may be that when the algorithm spends more time at a given temperature than is necessary for the probability distribution to reach equilibrium there is little further progress made at that temperature. Another reason for this may be that when tempfactor is small, satisfying the criterion of $\zeta < MINRATIO$ is more difficult. Thus, if the temperature prematurely becomes small before freezing sets in, the tail of the annealing curve may be extended.

Table IV gives the average cost found for each combination of parameter values. As expected increasing running time brings about more accurate solutions, although this does not seem to be cost effective beyond a certain point. For example, the exact answer for $G_{60}$ is 9.0774, which was found on all ten runs with tempfactor equal 0.9873 and chainfactor = 16. However, this answer took more than $10\frac{1}{3}$ times as much time as the answer with chainfactor equal 8 and tempfactor equal 0.9025, while the latter set of parameters gave an average solution which was approximately 1.7% more than the optimal. Moreover, on five of the ten runs the result with tempfactor of 0.9025 and chainfactor of 8 was optimal. Thus, it was decided to use these parameters for further tests.

Another observation from the table is that doubling chainfactor or taking the square root of tempfactor have about the same effect on accuracy. Thus, combining the information from Tables III and IV suggests that increasing tempfactor rather than doubling chainfactor is the better way to improve the solution when more time becomes available for the cooling schedule.

Table III. RUNNING TIME AS A FUNCTION OF CHAINFACTOR AND TEMPFACTOR FOR $G_{60}$

(Number of Trials) (N - Special)

| CHAIN | TEMPFACTOR | | | | | | |
|---|---|---|---|---|---|---|---|
| FACTOR | .4401 | .6634 | .8145 | .9025 | .9500 | .9747 | .9873 |
| .5 | -- | -- | -- | -- | -- | -- | 65 |
| 1 | -- | -- | -- | -- | -- | 87 | 194 |
| 2 | -- | -- | -- | -- | 104 | 178 | 317 |
| 4 | -- | -- | -- | 120 | 208 | 351 | 629 |
| 8 | -- | -- | 147 | 243 | 413 | 715 | 1270 |
| 16 | -- | 167 | 280 | 487 | 813 | 1434 | 2523 |
| 32 | 188 | 308 | 524 | 902 | 1579 | 2720 | -- |
| 64 | 377 | 623 | 1060 | 1862 | 3222 | -- | -- |
| 128 | 659 | 1118 | 1958 | 3447 | -- | -- | -- |
| 256 | 1337 | 2140 | 3711 | -- | -- | -- | -- |
| 512 | 2626 | 4250 | -- | -- | -- | -- | -- |
| 1024 | 5120 | -- | -- | -- | -- | -- | -- |

Table IV. AVERAGE COST AS A FUNCTION OF CHAINFACTOR AND TEMPFACTOR FOR $G_{60}$

| CHAIN | TEMPFACTOR | | | | | | |
|---|---|---|---|---|---|---|---|
| FACTOR | .4401 | .6634 | .8145 | .9025 | .9500 | .9747 | .9873 |
| .5 | -- | -- | -- | -- | -- | -- | 9.6740 |
| 1 | -- | -- | -- | -- | -- | 9.4968 | 9.2597 |
| 2 | -- | -- | -- | -- | 9.4537 | 9.2398 | 9.2146 |
| 4 | -- | -- | -- | 9.4082 | 9.2412 | 9.2253 | 9.1625 |
| 8 | -- | -- | 9.3980 | 9.2357 | 9.2079 | 9.1749 | 9.1254 |
| 16 | -- | 9.4139 | 9.2455 | 9.2262 | 9.1733 | 9.1467 | 9.0774 |
| 32 | 9.4221 | 9.2622 | 9.2202 | 9.1799 | 9.1472 | 9.1145 | -- |
| 64 | 9.2840 | 9.2377 | 9.1841 | 9.1617 | 9.1228 | -- | -- |
| 128 | 9.2413 | 9.1830 | 9.1760 | 9.1453 | -- | -- | -- |
| 256 | 9.2025 | 9.1622 | 9.1526 | -- | -- | -- | -- |
| 512 | 9.1665 | 9.1501 | -- | -- | -- | -- | -- |
| 1024 | 9.1346 | -- | -- | -- | -- | -- | -- |

This hypothesis was tested by doing 50 runs on $G_{60}$ with *chainfactor* = 1 and *tempfactor* = .9873(i.e. about $(.9025)^{\frac{1}{8}}$). The average running time was 13% better than that in Table III, while the average solution was 9.2460, only approximately 0.11% above the solution in Table IV for the values *chainfactor* = 8 and *tempfactor* = .9025. Since the purpose of this study was to determine whether the optimal or near optimal solution for the DSP can be found faster on the average with simulated annealing than with the algorithm of Wong, it was much more important to get a very accurate best answer quickly than to guarantee that the value at termination was optimal. The gain in running time was substantial, while the statistical significance of the slight difference in accuracy was dubious at best. Thus, it was decided to use *chainfactor* = 1, and *tempfactor* = 0.9873 as standard parameters for the TCA.

## C. MODIFYING THE TAILORED COOLING ALGORITHM

The objective in performing the experiments discussed in the preceding section was to find values for the parameters that gave good running times without sacrificing much accuracy. In this section, the goal is to look at some alternative ways of structuring the algorithm in order to achieve the same goals.

1. Choosing Neighbors by Random Permutations. In the general form of the tailored cooling algorithm configuration $F'$ is derived from configuration $F$ by choosing at random one vertex from the set of Steiner vertices and changing its membership status in the set of nodes that must be connected. That is, if it was previously not a member, then it becomes one and vice versa. This method is certainly simple, but it is not very efficient if only a few of the Steiner nodes affect the costs of the Steiner trees. This, in fact, is often the situation because many

Steiner vertices are always pruned from the minimum spanning tree that is formed from the set of nodes that it is necessary to connect. Thus it seemed reasonable to choose a random permutation of the Steiner nodes before each block of n trials and use that permutation to generate the next n moves. Johnson, Aragon, McGeoch, and Schevon [60] used this technique to improve their algorithm. In order to study the effect of changing the method for making succeeding moves, the following procedure was utilized.

Simulated annealing was applied 50 times to $G_{60}$ using the standard parameters of $INITIAL\_PROB = 0.3$, $CHAINFACTOR = 1$, $TEMPFACTOR = 0.9873$, and $MINRATIO = 0.02$, with moves chosen by randomly selecting a vertex to remove from, or add to, the set of special vertices. The results were compared with those found by running the algorithm 10 times, with the same parameters, but selecting neighbors through the use of random permutations. The averages in Table V show that the running times were approximately the same for both methods, but the mean cost found was lower when moves were found with the use of random permutations. Since the best average cost found with the five groups of ten runs that comprised the 50 runs was 9.2292, the differences in mean cost seemed to be significant. To further substantiate this hypothesis, the tailored cooling algorithm was applied to one of the standard graphs, hereafter denoted $G_{80}$, with 80 nodes, $p_e = 0.50$, and $special = 40$. Five runs each were made on this graph with and without the technique of choosing neighbors with random permutations. Table VI has the values that were found on these runs. These values also show an improvement in accuracy with about the same running time when random permutations are employed. When the same set of parameter values and initial configurations were incorporated into TCA with the exception that chainfactor was .5 rather than 1, the average cost went up to 1.0798 which was about what it was when the original neighbor selection method was used. However, the running times, as expected, were approximately half of what they were before. Hence, selection of moves according to random permutations was adopted both for the TCA and the DCA, while the value for chainfactor was maintained at 1.

2. An Alternative Neighborhood Structure. To pick a random neighbor under the method described above, a random permutation, $\Sigma$, of the set of Steiner points is initially chosen. Then at each call for a new neighbor the next element in $\Sigma$ is chosen, until $\Sigma$ is exhausted. Let $\varphi = |\{ \text{ Steiner points } \}|$. After each $\varphi$ moves, a new permutation is generated and the process is repeated. This helps make the annealing schedule terminate with a local minimum, because if a smaller solution exists, it must appear within the succeeding 2n trials. A similar technique can be used to design more extensive neighborhoods. Let $N(k)$ be the chosen neighborhood of a configuration such that each configuration in the neighborhood differs by k or fewer Steiner vertices from the original configuration. Up to now $N(1)$ has been used. Thus, when CHAINFACTOR = 1, $|N(1)| = \varphi$ was the number of permutations, which was equal the number of iterations at each temperature. The number of possibilities for $N(2)$ for an arbitrary configuration is $\varphi + \frac{\varphi(\varphi - 1)}{2} = \frac{(\varphi^2 + \varphi)}{2}$. Thus, as k increases, the number of possibilities grows rapidly. To allow all of these possibilities for a neighborhood means that the number of moves the algorithm must make at each temperature has a lower bound which is a function of $\varphi^k$. Hence, consideration of $N(k)$ for $k > 2$ was dismissed as infeasible.

To compare the results of using a 1-change neighborhood with the results obtained with a 2-change neighborhood, a series of experiments was performed on the 120 graphs with 40 nodes, and on some of the graphs with 80 nodes. In the first experiment, annealing was applied to all the 40-node graphs using a 1-change neighborhood. The best solutions and the total running time were recorded. Then Then local optimization was applied to the same graphs. This was done by generating random configurations using 1-change neighborhoods at each iteration and accepting only downhill moves(i.e. improving solutions). For each graph the runs of local optimization had a total running time equal to the running time for simulated annealing. Table VII shows a comparison of the best solutions for annealing and local optimization for this experiment. K represents the number of special nodes in Tables VII to X. A "Near Solution" is one that is within 3% of the optimum. In Table VII, simulated annealing outperforms local optimization by a clear margin.

In the second experiment, the same procedure was applied to the same set of graphs except that 2-change neighborhoods were used for both simulated annealing and local optimization. Table VIII indicates that the annealing results are only slightly better than those in Table VII, and the much larger neighborhoods have increased running time so much that local optimization performs better than annealing in some cases.

The third experiment involved graphs with 80 nodes and each combination of $p_e$ and special(12 graphs in all). Each of these graphs was run with 1-change neighborhoods, and the outcomes were compared with those of local optimization found for an equivalent amount of running time. Table IX shows that simulated annealing outperformed local optimization again, this time even more convincingly than in Table VII.

Finally, in the last experiment, annealing and local optimization were tried with 2-change neighborhoods on the 12 graphs with 80 vertices used in the third experiment. A 2-day running time bound was placed on both annealing and local optimization for each graph. Within this bound, neither annealing nor local optimization did as well as they did with 1-change neighborhoods. Table X has the results of this series of runs. The poor performance of both algorithms was probably due to the fact that the neighborhoods were so long in some cases that annealing was only able to work through a few Markov chains in the allotted time, and local optimization spent long periods making little, if any, progress.

From the four experiments, the following conclusions can be made. Within the range of sizes of the test ensemble of graphs, the 2-change neighborhoods improve the accuracy of the cooling algorithm only slightly and at the expense of a great increase in running time that is not at all cost effective. The experiments also imply that the running time becomes so long with 2-change neighborhoods that local optimization is just as effective. Finally, it appears that 2-change neighborhoods become less and less useful as n increases. Thus, for the remainder of this study 1-change neighborhoods are always used for both TCA and DCA.

Table V. COMPARISON BETWEEN METHODS FOR CHOOSING NEIGHBORS ON $G_{60}$ FOR THE TAILORED COOLING ALGORITHM

| | Random Vertex | Random Permutation |
|---|---|---|
| Mean Solution | 9.2415 | 8.2169 |
| Mean Running Time | 9068 sec. | 9123 sec. |

Table VI. COMPARISON BETWEEN METHODS FOR CHOOSING NEIGHBORS ON $G_{80}$ FOR THE TAILORED COOLING ALGORITHM

| | Random Vertex | Random Permutation |
|---|---|---|
| Mean Solution | 1.0641 | 1.0538 |
| Mean Running Time | 25598 sec. | 26053 sec. |

Table VII. COMPARISON OF THE TAILORED COOLING ALGORITHM AND LOCAL OPTIMIZATION FOR GRAPHS OF 40 VERTICES AND 1-CHANGE NEIGHBORHOODS

| | | Tailored Cooling Algorithm | | Local Optimization | |
|---|---|---|---|---|---|
| $p_e$ | K | # Opt. Solutions | # Near Solutions | #Opt. Solutions | # Near Solutions |
| .05 | 10 | 12 | 0 | 9 | 3 |
| | 20 | 10 | 2 | 10 | 2 |
| | 30 | 11 | 1 | 9 | 2 |
| .25 | 10 | 11 | 1 | 9 | 2 |
| | 20 | 12 | 0 | 8 | 2 |
| | 30 | 10 | 2 | 9 | 3 |
| .50 | 10 | 12 | 0 | 10 | 2 |
| | 20 | 11 | 1 | 8 | 3 |
| | 30 | 12 | 0 | 11 | 1 |
| .90 | 10 | 10 | 2 | 10 | 2 |
| | 20 | 10 | 2 | 9 | 3 |
| | 30 | 12 | 0 | 11 | 1 |

Table VIII. COMPARISON OF THE TAILORED COOLING ALGORITHM AND LOCAL OPTIMIZATION FOR GRAPHS OF 40 VERTICES AND 2-CHANGE NEIGHBORHOODS

| | | Tailored Cooling Algorithm | | Local Optimization | |
|---|---|---|---|---|---|
| $p_e$ | K | # Opt. Solutions | # Near Solutions | #Opt. Solutions | # Near Solutions |
| .05 | 10 | 12 | 0 | 12 | 0 |
| | 20 | 12 | 0 | 11 | 1 |
| | 30 | 11 | 1 | 12 | 0 |
| .25 | 10 | 11 | 1 | 11 | 1 |
| | 20 | 12 | 0 | 12 | 0 |
| | 30 | 11 | 1 | 12 | 0 |
| .50 | 10 | 12 | 0 | 12 | 0 |
| | 20 | 12 | 0 | 12 | 0 |
| | 30 | 12 | 0 | 12 | 0 |
| .90 | 10 | 12 | 0 | 12 | 0 |
| | 20 | 11 | 1 | 11 | 1 |
| | 30 | 12 | 0 | 12 | 0 |

Table IX. COMPARISON OF THE TAILORED COOLING ALGORITHM AND LOCAL OPTIMIZATION FOR GRAPHS OF 80 VERTICES AND 1-CHANGE NEIGHBORHOODS

| | | Tailored Cooling Algorithm | Local Optimization | |
|---|---|---|---|---|
| $p_e$ | K | $\frac{\text{bestanswer}}{\text{opt}}$ | $\frac{\text{bestanswer}}{\text{opt}}$ | $\frac{\text{meananswer}}{\text{opt}}$ |
| .05 | 20 | 1.000 | 1.012 | 1.058 |
| | 40 | 1.003 | 1.074 | 1.097 |
| | 60 | 1.007 | 1.010 | 1.048 |
| .25 | 20 | 1.000 | 1.000 | 1.009 |
| | 40 | 1.015 | 1.015 | 1.032 |
| | 60 | 1.000 | 1.000 | 1.013 |
| .50 | 20 | 1.012 | 1.028 | 1.037 |
| | 40 | 1.000 | 1.016 | 1.024 |
| | 60 | 1.000 | 1.000 | 1.015 |
| .90 | 20 | 1.018 | 1.049 | 1.066 |
| | 40 | 1.016 | 1.044 | 1.063 |
| | 60 | 1.000 | 1.007 | 1.018 |

Table X. COMPARISON OF THE TAILORED COOLING ALGORITHM AND LOCAL OPTIMIZATION FOR GRAPHS OF 80 VERTICES AND 2-CHANGE NEIGHBORHOODS

| | | Tailored Cooling Algorithm | Local Optimization | |
|---|---|---|---|---|
| $p_e$ | K | $\frac{\text{bestanswer}}{\text{opt}}$ | $\frac{\text{bestanswer}}{\text{opt}}$ | $\frac{\text{meananswer}}{\text{opt}}$ |
| .05 | 20 | 1.056 | 1.042 | 1.061 |
| | 40 | 1.045 | 1.080 | 1.109 |
| | 60 | 1.016 | 1.000 | 1.061 |
| .25 | 20 | 1.024 | 1.039 | 1.055 |
| | 40 | 1.038 | 1.000 | 1.049 |
| | 60 | 1.043 | 1.043 | 1.066 |
| .50 | 20 | 1.025 | 1.052 | 1.063 |
| | 40 | 1.012 | 1.026 | 1.074 |
| | 60 | 1.000 | 1.031 | 1.068 |
| .90 | 20 | 1.025 | 1.000 | 1.058 |
| | 40 | 1.046 | 1.017 | 1.031 |
| | 60 | 1.013 | 1.013 | 1.042 |

# VI. DYNAMIC COOLING ALGORITHM

In 1985, Aarts and van Laarhoven [4], developed a new general cooling schedule emphasizing feasibility and robustness. The parameters are determined in such a way as to foster fast convergence towards near-optimal answers. Moreover, the quality of the optimization is independent of the problem size. With this adaptive schedule, Aarts and van Laarhoven [4] proved that the execution time of the simulated annealing algorithm is proportional to

$$\max_{i \in I_\Lambda} |\Lambda_i| \ln |\Lambda|, \tag{6.1}$$

where i is any configuration, $I_\Lambda = \{1, 2, \ldots, |\Lambda|\}$ is the set of state labels of the system configurations contained in $\Lambda$. The notation $\Lambda_i$ represents the configurations in a neighborhood of i. The term ln $|\Lambda|$ is an upper bound for the number of steps in the decrementing of temperature. For the Directed Steiner Problem, $|\Lambda| = 2^{(n-special)}$ and $|\Lambda_i| = (n - special)$. Consequently, the algorithm designed by Aarts and van Laarhoven has a time complexity proportional to $O((n - special)^2)$. Since the criterion for decrementing the control parameter and the rule for termination in the schedule depend on averages found during execution, the algorithm is referred to as the dynamic cooling algorithm (DCA) in this dissertation. The underlying reasoning behind the DCA is that more time should be spent at those temperatures where the current average cost is declining rapidly. This thinking is analogous to that used in physics where gases are cooled more slowly at those points where the specific heat increases quickly.

This chapter shows how the parameters for the DCA were chosen in order to maximize the effectiveness of the algorithm for the Directed Steiner Problem. In addition, data is presented showing the total CPU time as a function of n, $p_e$, and *special*, the parameters used to construct random graphs.

## A. IMPLEMENTATION OF THE DYNAMIC COOLING ALGORITHM

The dynamic cooling algorithm can be stated as follows.

```
1.        Initialize(CHAINLENGTH, $c_0$, $\overline{C}(c_0)$ );
2.        Set $c := c_0$ ;
3.        Repeat
   (a).        For i:= 1 to CHAINLENGTH do
               begin
      (i).         Call TRANSITION(state i → state j, $\Delta C_{ij}$) ;
      (ii).        If $\Delta C_{ij} \leq 0$ then accept
                   else if $\exp(\frac{-\Delta C_{ij}}{c}) > \text{random}[0, 1]$
                   then accept;
      (iii).       If accept then call UPDATE;
               end;
   (b).        Compute( $\sigma(c), \frac{d\overline{C}_s(c)}{dc}$ ) ;
   (c).        Set $c = \frac{c}{\left(1 + \frac{\ln(1+\delta)c}{3\sigma(c)}\right)}$ ;
          until $\frac{d\overline{C}_s(c)}{dc} \frac{c}{\overline{C}(c_0)} < \varepsilon$
```

In this algorithm the termination condition for the repeat loop is equation (4.20), and the formula for decrementing the control parameter in 3(c). is equation (4.26). The notation $\overline{C}_s(c)$ is the smoothed value of $\overline{C}(c)$ as c gets small. The smoothed value is obtained by calculating a running average over a number of Markov chains. This allows a good estimate of the average difference in cost function $\overline{\Delta C}(c) = \overline{C}_s(c) - C(\lambda_{\cdot 0})$, where $C(\lambda_{\cdot 0})$ is the minimum value of the cost function [4]. If $c << 1$, then

$$\overline{\Delta C}(c) \simeq c \frac{d\overline{C}_s(c)}{dc} . \tag{6.2}$$

Since $\overline{C}(c)$ fluctuates rapidly, it is necessary to use the smoothed values in order to accurately determine the derivative $\frac{d\overline{C}(c)}{dc}$. If the difference $\overline{\Delta C}(c)$ is relatively small compared to $\overline{C}(c_0)$, then the optimization is almost completed. This is the reasoning behind the terminating condition in the DCA. The value of CHAINLENGTH is $\max_{i \in I_A} |A_i|$, which in this case is (n - special). The parameter CHAINFACTOR used in the implementation of TCA is always equal to one for the DCA.

Figure 13 shows the evolution of an annealing run of DCA on $G_{60}$. This figure can be compared with Figure 8. The difference shows the importance of the implementation of simulated annealing. The parameters used were $INITIAL\_PROB = .9$, $\delta = 0.1$, and $\varepsilon = 0.0000001$. The value of $\overline{C_s}(c)$ is obtained throughout this dissertation by averaging $\overline{C}(c)$ over 3 Markov chains. In Figure 13 the schedule converges extremely rapidly, showing the power of the dynamic cooling algorithm for proper values of $\delta$ and $\varepsilon$. Each chainlength is 27 iterations. Thus, the entire process took only 432 iterations as compared with 40,608 iterations in Figure 8. On the other hand, when the same procedure was run with $\delta$ equal to 0.00001, after more than 15,000 iterations the algorithm stopped with a solution that was more than 30% larger than the optimum. Hence, it is necessary to adapt this algorithm to the problem just as it was with the tailored cooling schedule.

The reason that a smaller value of $\delta$ led to a decrease in final accuracy is that when INITIAL_PROB is .9, the initial temperature is high. The graph may contain many configurations with the same solution. Every time a sequence of equivalent configurations is encountered the average change in cost is zero for that sequence. This causes the expression $\Psi$ used in the stopping rule to be unrealistically small. The phenomenon can occur even when the temperature is quite high. Thus, the slow decrease in temperature due to the smallness of $\delta$ leads to premature termination of execution before the schedule begins to converge to a near optimal solution. Often this does not harm the quality of the best answer because the running time is substantial anyway for small values of epsilon. This condition does, however, adversely affect the final solution. Thus, it is necessary to look at both the best answer and the final answer for each combination of parameter values before deciding which are the optimum values of $\delta$ and $\varepsilon$ to use for a particular problem.

If the starting temperature is close to the point at which convergence begins, then small values of delta cause smoother convergence and better final answers. Hence, the rate at which the cooling parameter is decremented is a function of the initial temperature as well as the length of the Markov chains.

Random initial solutions and initial temperatures are found in the same way as they were for the tailored cooling algorithm. In addition, the same set of 480 graphs is used for experiments. For the DCA there are not as many parameters as there were for the TCA. There are only two that must be adjusted: $\delta$ and $\varepsilon$. In Figures (14 - 15), INITIAL_PROB = 0.3. Figure 14 shows the deviation from the optimum as a function of the expression $\Psi = \frac{d\overline{C}_t(c)}{dc}\frac{c}{\overline{C}(c_0)}$ for a graph with 80 nodes, special = 40, and $p_e$ = .25. In the figure, $\delta = 0.001$. The figure shows that the stop criterion $\Psi$ permits the DCA to reach near optimal solutions. Figure 15 demonstrates the deviation from the optimum as a function of $\delta$ for the same graph as Figure 14 In Figure 15, $\varepsilon$ has the value 0.0000001. The graph shows that for a fixed value of $\varepsilon$, the parameter $\delta$ can be determined so that near optimal solutions are obtained.

In order to find the combination of $\delta$ and $\varepsilon$ that makes the DCA most effective, experiments were done as follows. The parameters $\delta$ and $\varepsilon$ assumed all combinations of values from { 0.00001, 0.00005, 0.0001, 0.0005, 0.001, 0.005, 0.01, 0.05, 0.1 } and { 0.0000001, 0.000001, 0.00001, 0.0001, 0.001 }, respectively. The dynamic cooling algorithm was run on $G_{60}$, $G_{80}$, and a graph with 40 nodes, $p_e = .25$, and special = 10 five times for each combination of possible values assumed by $\delta$ and $\varepsilon$. For these experiments a different initial configuration was found for each run. INITIAL_PROB was equal to 0.3. The averages shown in Tables (XI - XIII) are for $G_{60}$. Similar relations were found for the other two graphs.

Table XI shows the effects of the parameters on total running time for the annealing schedule. As $\delta$ and $\varepsilon$ get smaller, the running time increases. Table XII compares the average solution found during the final Markov chain with the optimum solution for the problem. The values at the conclusion of the schedule are all near optimal if $\varepsilon \geq 0.001$. Table XIII indicates that the average values of the best answers improve steadily to the optimum as the parameters become smaller. The results shown in Tables ((XI - XIII) are not unexpected since Aarts and van Laarhoven [4] proved that their algorithm ran with a worst case time complexity that was polynomial in the size of the input, but they did not claim that it converged to an optimal solution. They only stated that if the parameters were chosen to fit the problem, then a near optimal solution would be produced in polynomial time.

The final solutions for all three test graphs were always within 0.6% of the optimum when $\varepsilon \leq 0.000001$ and $\delta$ was between .0005 and .005. The best answers were always optimum whenever $\varepsilon \leq 0.000001$ and $\delta \leq 0.005$. Running time decreased sharply when $\delta$ decreased below 0.00005. Hence, in the interest of both running time and accuracy, the parameters were assigned the values 0.000001 for $\varepsilon$ and 0.001 for $\delta$.

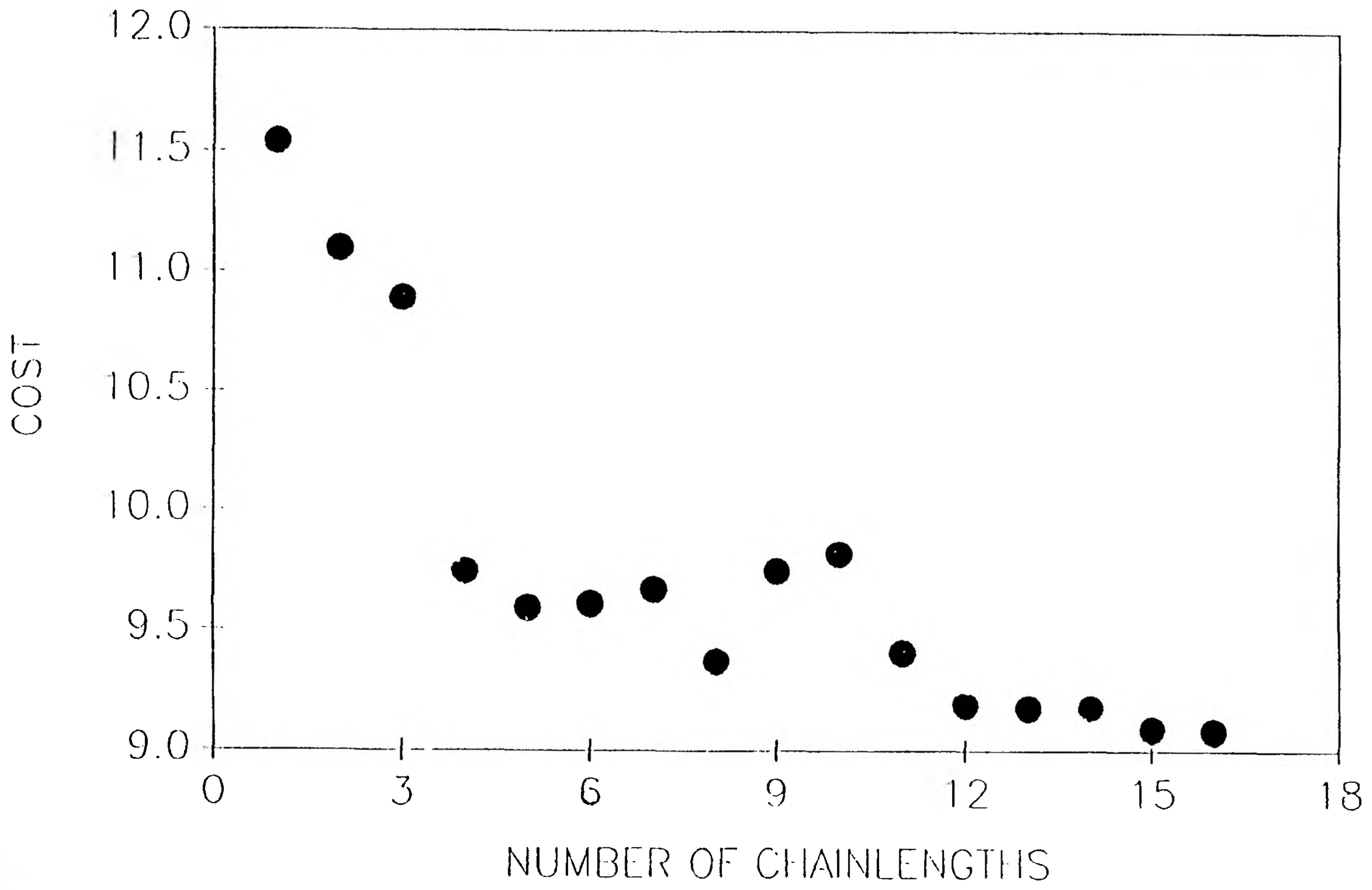

Figure 13. The evolution of the solution value during annealing on $G_{60}$ with the DCA. Time increases(thus temperature decreases) along the X-axis. The Y-axis measures the current cost.

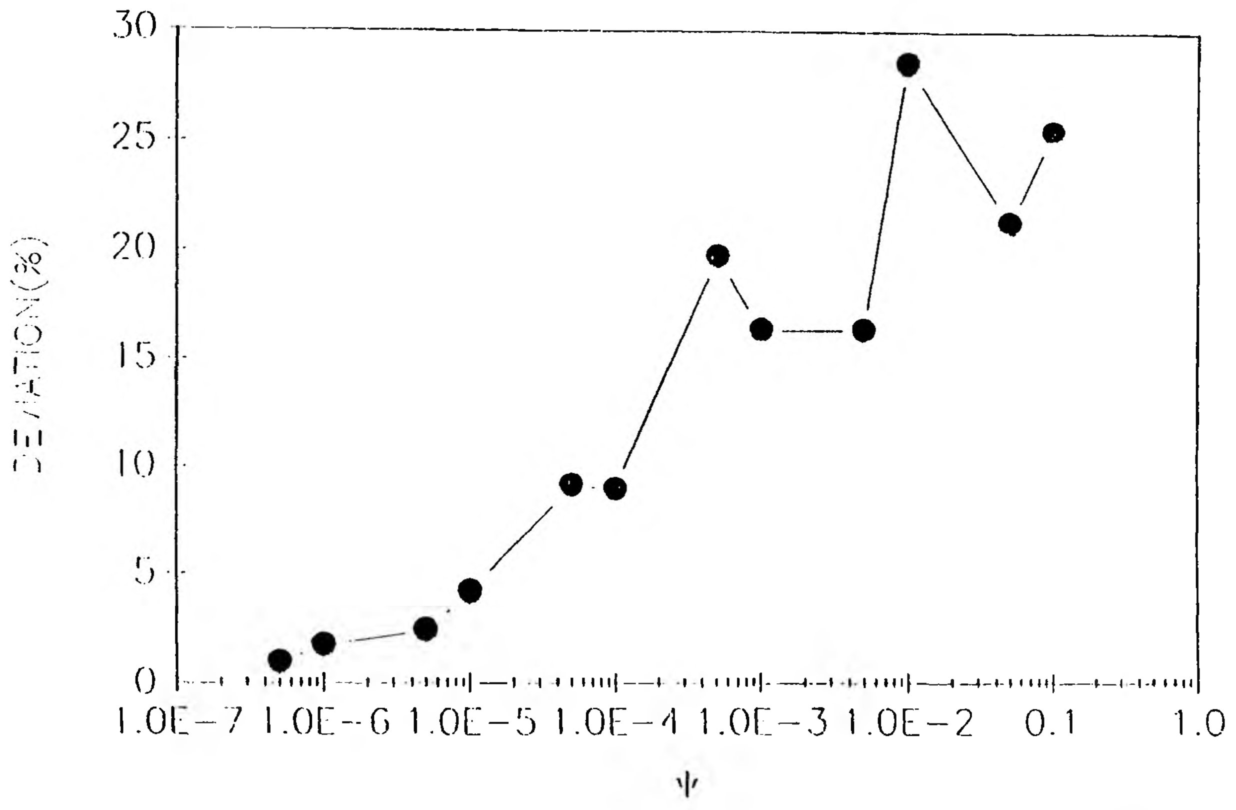

Figure 14. The deviation from the optimum for a graph of 80 nodes, special = 40, and $p_e = .25$ as a function of the stopping rule Ψ

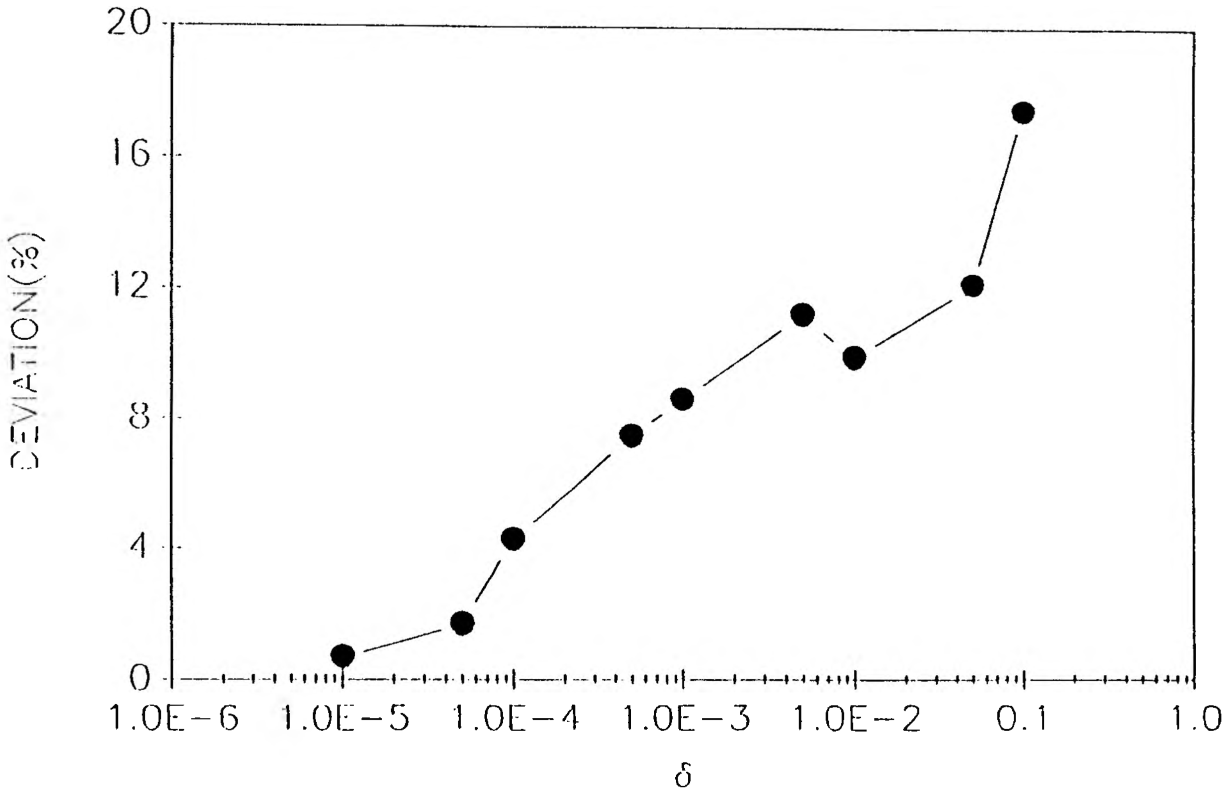

Figure 15. The deviation from the optimum for a graph of 80 nodes, special = 40, and $p_r = .25$ as a function of the parameter $\delta$.

Table XI. RUNNING TIME AS A FUNCTION OF $\delta$ and $\varepsilon$ FOR $G_{60}$

(Number of Trials)/(N - Special)

| $\delta$ | $\varepsilon$ | | | | |
|---|---|---|---|---|---|
| | .0000001 | .000001 | .00001 | .0001 | .001 |
| .00001 | 39 | 39 | 39 | 8 | 4 |
| .00005 | 25 | 39 | 22 | 8 | 4 |
| .0001 | 16 | 18 | 12 | 8 | 4 |
| .0005 | 18 | 18 | 10 | 8 | 4 |
| .001 | 13 | 12 | 10 | 8 | 4 |
| .005 | 13 | 10 | 8 | 8 | 4 |
| .01 | 12 | 10 | 10 | 7 | 4 |
| .05 | 13 | 10 | 10 | 7 | 4 |
| .1 | 13 | 6 | 6 | 6 | 4 |

Table XII. (FINAL COST) / OPTIMUM AS A FUNCTION OF $\delta$ and $\varepsilon$ FOR $G_{60}$

| $\delta$ | $\varepsilon$ | | | | |
|---|---|---|---|---|---|
| | .0000001 | .000001 | .00001 | .0001 | .001 |
| .00001 | 1.001 | 1.001 | 1.000 | 1.010 | 1.070 |
| .00005 | 1.019 | 1.015 | 1.038 | 1.010 | 1.070 |
| .0001 | 1.019 | 1.018 | 1.019 | 1.010 | 1.070 |
| .0005 | 1.000 | 1.000 | 1.001 | 1.010 | 1.070 |
| .001 | 1.000 | 1.002 | 1.000 | 1.010 | 1.070 |
| .005 | 1.000 | 1.000 | 1.010 | 1.010 | 1.070 |
| .01 | 1.000 | 1.000 | 1.013 | 1.013 | 1.070 |
| .05 | 1.000 | 1.006 | 1.013 | 1.013 | 1.070 |
| .1 | 1.003 | 1.013 | 1.013 | 1.013 | 1.070 |

Table XIII. (BEST COST) / OPTIMUM AS A FUNCTION OF $\delta$ and $\varepsilon$ FOR $G_{60}$

| $\delta$ | $\varepsilon$ | | | | |
|---|---|---|---|---|---|
| | .0000001 | .000001 | .00001 | .0001 | .001 |
| .00001 | 1.000 | 1.000 | 1.000 | 1.006 | 1.060 |
| .00005 | 1.000 | 1.000 | 1.000 | 1.006 | 1.060 |
| .0001 | 1.000 | 1.000 | 1.000 | 1.006 | 1.060 |
| .0005 | 1.000 | 1.000 | 1.000 | 1.006 | 1.060 |
| .001 | 1.000 | 1.000 | 1.000 | 1.006 | 1.060 |
| .005 | 1.000 | 1.000 | 1.006 | 1.010 | 1.060 |
| .01 | 1.000 | 1.006 | 1.010 | 1.010 | 1.060 |
| .05 | 1.000 | 1.006 | 1.003 | 1.010 | 1.060 |
| .1 | 1.000 | 1.010 | 1.010 | 1.010 | 1.060 |

## B. COMPARISON OF TCA AND DCA

In this section the two versions of simulated annealing for the Directed Steiner Problem are compared in terms of accuracy and running time.

1. Accuracy of the TCA and DCA. The 480 graphs in the test bed were each run with both the tailored cooling schedule and the dynamic cooling schedule. The relationship between the parameters n, $p_e$, and special and the deviation from the optimum of the best answer and the final answer for each of the two schedules was investigated. For example, in Figure 16 there is a comparison of the deviation from the optimum of various optimization processes as a function of the number of vertices. The results represent averages for the 10 graphs at each value of n when $p_e$ equals .50 and $special = \frac{n}{2}$. Curve (a) indicates the averages for the solutions produced by doing local optimization once for each of the 10 graphs, curve (b) shows the averages for the final answers of TCA, curve (c) shows the averages for the final answers of DCA, curves (d) and (e) demonstrate the averages for the best answers for TCA and DCA respectively. Figure 16 shows that iterative

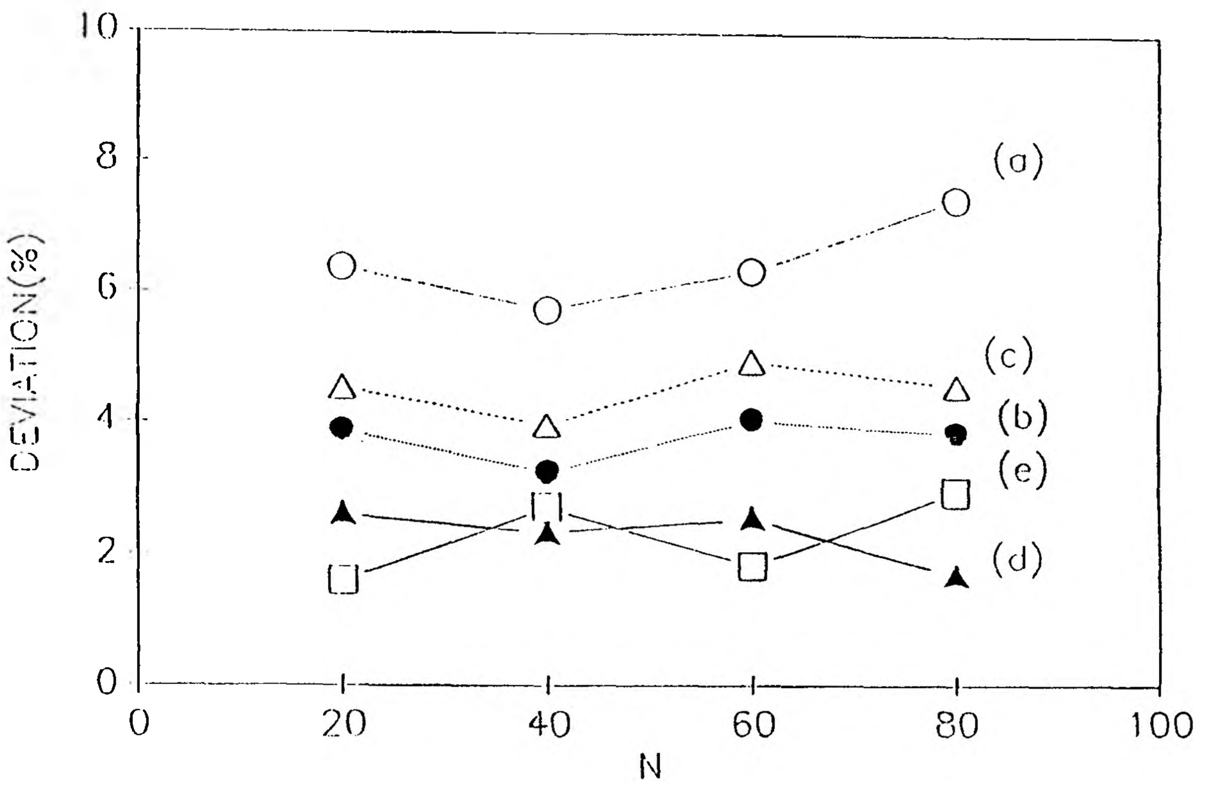

Figure 16. Deviation from the optimum of the final result as a function of n. $p_c = 0.50$ and special $= \frac{n}{2}$.
(a) local optimization (b) and (c) final answers for the TCA and DCA, respectively (d) and (e) best answers for the TCA and the DCA, respectively.

improvement leads to answers that deviate from the optimum by approximately 6.5%, and that both of the simulated annealing schedules are much better. The lines between points are drawn to guide the eye.

The final solutions for the average deviation of the final costs for both TCA and DCA show that the deviation is independent of the number of vertices if the density is .50 and $special = \frac{n}{2}$. This characteristic of the algorithms appears only if the parameters of the cooling schedules are chosen properly. Different parameters may destroy the independence of the final result from the size of the problem. To test the independence of the algorithms from the size of the problem, each of the 480 graphs was run once. The deviations of the final and best solutions from the optimum were recorded. The average deviations for each combination of parameters n, $p_e$, and special were calculated. In 462 of the 480 graphs the best solution for the TCA was within 3% of the optimum. In those graphs that were exceptions, the worst deviation from the optimum was 4.8%. For the DCA only 25 of the 480 graphs did not have a best solution within 3% of optimum, and the worst result among the 25 exceptions was within 6.1% of optimum. For the final solutions, the tailored cooling algorithm yielded costs that were within 3% of optimum for all but 57 of the 480 graphs. The worst deviation among the 18 was 6.4%. With the DCA the final solutions failed to be within 3% in 66 cases with the largest deviation equal to 6.7%. The average results for each combination of parameters for both DCA and TCA are shown in Tables (XIV-XV). From these tables it can be seen that the quality of the solutions yielded by both cooling schedules is independent of the size of the problem. The data indicates that the TCA usually provides more accurate answers than does the DCA, but the average difference is only approximately 1%. For the 48 combinations of parameter values in 32 instances the TCA gave the better average best answer, and in 37 instances the

TCA yielded the better final answer. In all cases but one, both algorithms provided average best answers that were within 3% of optimum. In the only exception, the ten graphs with 60 nodes, $p_e = 0.05$, and special = 30, the DCA returned an average best answer of 3.02% above optimal.

2. Running Times of TCA and DCA. In this section a comparison is made of the total running times of the tailored cooling algorithm and the dynamic cooling algorithm. In addition, some results of experiments conducted on the distribution of running times until a near optimal solution(i.e. within 3% of the optimal) appears are presented. This is the first time that an investigation of this topic has been done.

a. Total Running Times. As mentioned at the start of this chapter, the order of the computation effort for the Directed Steiner Problem with the dynamic cooling algorithm has a worst-case upperbound on the order of $O((n - special)^2)$. Each iteration of an annealing schedule requires four steps: firstly, the system has to be perturbed by generating at random a new configuration, secondly, the difference in cost function must be calculated, thirdly, a decision must be made whether to adopt a new configuration, and fourthly, the system is updated if the new configuration is accepted. The perturbation process is rather simple, the next Steiner vertex in a permutation changes status. If it was included in the Steiner tree, then it is removed; if it was not present previously, then it is included. The decision concerning accepting the new configuration is based on no more than a single exponential function evaluation and the generation of a random number. Updating the system involves a couple of assignment statements to keep track of the current solution. The most time consuming part of the iteration by far is the calculation of the cost function. Unfortunately, for a directed graph there is no way to incrementally determine the minimum spanning tree from the minimum spanning tree of a graph that is identical to the present one except for the addition or deletion of one vertex and the edges incident to it. Thus, each iteration requires the use of the minimum spanning tree algorithm. This algorithm has a worst-case running time of $O(t(\ln t)^2 + m)$, where t is the number of vertices and m is the number of edges in the graph [99]. Since $t \leq n$, the DCA has a worst-case running time of $O((n - special)^2(n \ln n^2 + m))$.

A comparison of the running times of TCA and DCA was done by comparing the total running times for the 480 graphs in the test bed. The average total running time for each combination of parameters was calculated. Tables (XVI-XVII) contain the information gathered. For all values of n, $p_e$, and special the TCA required considerably more time than did the DCA. It appears that for both algorithms that the larger the number of special nodes the faster is the average total running time. Tables (XVI-XVII) contain the collected data for running times. Thus, the more sophisticated methods of decrementing the control parameter and of determining when to stop execution of the algorithm which characterize the dynamic approach yield faster schedules than the more simple-minded, but easier to implement, approach of the tailored algorithm. Furthermore, as the problems become larger, the difference in total running times increases.

b. Running Times Until Near-Optimal Solutions. In this portion of the dissertation the distribution of the first occurrences of near-optimal answers is discussed. Near optimal answers are those which are within 3% of the optimum. Figure 17 shows the distribution of the first occurrence of at least a near-optimal solution for 50 graphs with 40 vertices, $p_e = 0.05$, and special = 10 for both the TCA and DCA. This figure shows that for most of the runs a near-optimal solution appeared within the first two chainlengths. For all of the 50 graphs a near optimal solution was found. The distributions for both algorithms are similar. In addition, for each of the following combinations of parameter values 50 graphs were run and histograms were plotted:

1. n = 40, $p_e$ = 0.05, special = 20,
2. n = 40, $p_e$ = 0.05, special = 30,
3. n = 60, $p_e$ = 0.05, special = 15,
4. n = 60, $p_e$ = 0.05, special = 30,
5. n = 60, $p_e$ = 0.05, special = 45,
6. n = 60, $p_e$ = 0.90, special = 15,
7. n = 60, $p_e$ = 0.90, special = 30,
8. n = 60, $p_e$ = 0.90, special = 45.

The histograms for these cases are very similar to those of Figure 17. They are shown in Figures (18-25). The graphs with 60 nodes took more iterations to find the near-optimal costs than did those with 40 nodes, but the distribution of chainlengths was approximately the same. The number of iterations required by the TCA did not grow as fast as the number needed by the DCA to find a near optimal solution. In addition, for those graphs with 60 nodes, the distributions with $p_e$ = 0.90 and $p_e$ = 0.05 had histograms that were nearly identical when special was fixed. Thus, the density of the graph does not appear to affect the number of iterations that occur before a near-optimal solution appears. If graphs with the same values for n and $p_e$ are compared, the results indicate that simulated annealing finds near-optimal solutions faster for those graphs with the larger proportion of special nodes. These histograms indicate that the dynamic cooling algorithm takes more iterations on the average to find a near-optimal solution than does the tailored cooling schedule. The time spent per iteration is less for the TCA. Thus, although

DCA has a shorter total running time, the TCA seems to generally obtain a near optimum solution in less time.

In order to further compare the two forms of simulated annealing with respect to the occurrence of high quality answers in the cooling schedules, 120 random graphs for each algorithm were generated and each run from five different starting configurations. The number of iterations until either a near-optimum solution appeared or the best answer occurred was recorded. The test bed of graphs for this experiment consisted of 5 graphs with each combination of parameters N(number of nodes), $p_e$(density), and Special(number of nodes that are required to be connected), where $N \in \{20, 40, 60, 80\}$, $p_e \in \{0.05, 0.90\}$, and Special $\in \left\{\frac{N}{4}, \frac{N}{2}, \frac{3N}{4}\right\}$. Thus, the number of iterations until a high quality answer was reached was collected for 600 runs for each algorithm. The data were then subjected to a regression analysis in order to see if the two algorithms affect the number of iterations differently.

The list of possible independent variables that were considered initiallly consisted of the following: DA, a classification variable representing the different algorithms(0 for the DCA and 1 for the TCA), DG, a classification variable representing each of the five graphs considered for each combination of parameters, N, $p_e$, Special, and all possible product terms of the preceding five variables. Thus, a total of 31 independent variables were considered initially. Since the five examples of graphs considered for each combination of N, $p_e$, and Special were not universally recognizable types, but merely constructed to introduce variety into the set of problems on which the two algorithms were to be tested, it was decided not to treat DG as an independent variable. Thus, any variation in the number of iterations due to DG was treated as part of the error or "noise" component of the models considered. Together with DG, all product terms involving DG were also dropped.

Let $p-1$ represent the number of independent variables in any model. It is assumed that all regression models have an intercept term. Thus the regression function for $p-1$ independent variables has $p$ parameters. Since it is possible that some of the variables DA, N, $p_e$, Special, or their product terms have no significant effect on the number of iterations, various statistical model building and variable selection techniques were used to eliminate unimportant independent variables. Since the total number of independent variables, that is, the four variables DA, N, $p_e$, Special, and all of the possible product terms of the latter three variables, was very large, some model selection techniques such as fitting all possible regression models were not initially feasible. Hence, other methods such as stepwise regression, MAXR, and MINR procedures were used at the beginning in order to do the following: a. identify those independent variables that appeared to have considerable effect on the number of iterations; b. determine a reasonable upper bound on the number of independent variables that should be included in any candidate model. The three procedures mentioned above are automated and work by adding or deleting independent variables according to one or more statistical criteria. Once the highly influential of the independent variables and an upper bound for the number of variables in a model were identified, regression analysis was conducted on all possible models that included the highly influential variables and had no more variables than the upper bound obtained at the initial stage. The resulting models were compared using three criteria, namely multiple correlation coefficient, $R_p^2$, the adjusted multiple correlation coefficient $R_{a(p)}^2$, and the Mallows $C_p$[78]. All models that failed to satisfy the $C_p$ criterion were eliminated from consideration, and the remaining models were evaluated using the other two criteria. This produced a handfull of candidates for the final model. The validity of these candidates was verified by using the automated procedures such as Stepwise. Finally, simpler candidate models were tested for loss of explanatory power against more complex candidates by using an F-test. Before the process of selection is discussed further some necessary preliminary notation is introduced.

Let $I_i$ represent the value of the ith actual observation in the regression model, and let $\hat{I}_i$ represent the estimated mean of the ith observation. Let $\bar{I}$ be the average of all the observations of iterations made. Then if there are k observations, the sum of squares total is defined as follows.

$$SSTO = \sum_{i=1}^{k} (I_i - \bar{I})^2. \tag{6.3}$$

The error sum of squares or residual sum of squares of a model with $p-1$ independent variables is defined as:

$$SSE_p = \sum_{i=1}^{k} (I_i - \hat{I}_i)^2. \tag{6.4}$$

The difference between the sum of squares total error and the residual sum of squares is the regression sum of squares, denoted by $SSR_p$. That is,

$$SSR_p = \sum_{i=1}^{k} (\hat{I}_i - \bar{I})^2. \tag{6.5}$$

The mean square error, denoted by $MSE_p$, is an unbiased estimator of the variance of the regression model. It is given by:

$$MSE_p = \frac{SSE_p}{k-p}. \tag{6.6}$$

The mean square due to regression, $MSR_p$, is defined as:

$$MSR_p = \frac{SSR_p}{p-1}. \tag{6.7}$$

The three criteria for initially reducing the set of independent variables can now be defined. The coefficent of multiple correlation is

$$R_p^{\ 2} = \frac{SSR_p}{SSTO} = 1 - \frac{SSE_p}{SSTO}. \tag{6.8}$$

The equation defining the adjusted multiple correlation coefficient is

$$R_{a(p)}^{\ \ 2} = 1 - \frac{MSE_p}{\frac{SSTO}{k-1}}. \tag{6.9}$$

Finally, for a model having a subset of p independent variables out of a possible total of $p' - 1$ independent variables, the Mallows $C_p$ statistic is expressed by:

$$C_p = \frac{SSE_p}{MSE_{p'}} - (k - 2p), \tag{6.10}$$

where $MSE_{p'}$ is the mean square error of the model with $p' - 1$ independent variables.

There are three main automated procedures for selecting independent variables for the regression function: STEPWISE, MAXR, AND MINR. They can all be specified as options in the SAS PROC STEPWISE statement. Before discussing the selection process for the model describing the number of iterations used by the two annealing algorithms, a brief description of these three SAS procedures is presented.

STEPWISE is a combination of two other selection procedures, namely FORWARD selection and BACKWARD elimination. Forward selection begins by finding the variable that produces the optimum single variable model in terms of the multiple correlation coefficient. Succeeding iterations add one variable at a time, the variable which results in the largest increase in $R_r^2$. The process continues until no variable considered for addition to the model gives a statistically significant reduction in the $SSE_r$. The level of statistical significance is determined in advance by the analyst. Backward elimination begins by computing the regression function with all independent variables. The statistics for the partial regression coefficients are used to find the coefficient with the smallest F value. The variable corresponding to that coefficient is deleted from the model. The process continues until all coefficients left in the model are statistically significant. The Stepwise procedure used in the work for this study begins like forward selection, but after the addition of a variable, the coefficients of the resulting regression equation are checked to see if they have a statistically significant F value. If not, a backward elimination process is initiated. This procedure terminates only when no more additions or deletions are possible for the specified signifcance level. MAXR begins by choosing the best models with one and two variables according to the forward selection method. Then the procedure tries all possible interchanges of variables in the model with those outside of it. The interchange that causes the largest increase in $R_a^2$ is accepted. Then a third variable is added by forward selection. The interchanging of variables within the model with those without is repeated. This pattern continues until a model with all of the possible variables is obtained. MAXR checks more models than does STEPWISE. Hence, it requires much more computer time and gives more reliable information. MINR is like MAXR except that interchanges are implemented for those variables causing minimum rather than maximum improvement. It considers more models than does MAXR. The process of selecting a regression model for the annealing algorithms can now be presented.

In the full model, there were initially 15 independent variables: DA, N, $p_e$, Special, and their product terms. In order to reduce the number of variables to a more manageable size, it was decided to apply MAXR, MINR, and Stepwise to the full model in an attempt to find insignificant variables that could be immediately eliminated. This resulted in a model with these 12 variables: DA, N, Special, $p_e$, N*S, N*$p_e$, Special*DA, $p_e$*Special, $p_e$*DA, N*Special*DA, N*$p_e$*DA, and N*$p_e$*Special*DA. The SAS General Linear Model procedure(GLM) was applied to this model. This procedure uses an $F$ test to examine for each independent variable the hypothesis that the coefficient of that variable is zero. In the model with 12 variables described above, the probability that the coefficient of N was zero was 0.0001, and the probability that the coefficient of Special was zero was equal to 0.0005. The variable DA was of interest because it represented each of the algorithms. Hence, in the next step when the procedure RSQUARE was used to evaluate all of the possible models whose sets of variables were subsets of the 12 remaining variables, it was decided that the only models that RSQUARE needed to consider were those including DA, N, and Special.

The goal was to find a model with less than 15 independent variables that would account for nearly as much of the variation in results as the 15-variable model did. One of the measures of interest reported by RSQUARE was the Mallows, $C_p$, statistic for each of the various subset regression models. The value should be small and less than $p$, where $p$ is the number of parameters in the model. All of the models with less than six independent variables had $C_p$ values that were considerably greater than p. The best value of the measure for a six-variable model (i.e. p = 7) was 6.527164. The variables DA, N, S, N*DA, S*DA, and N*S were in this model. The second best value for a six-variable model was $C_p = 7.218883$. The best three values of $C_p$ for seven-variable models were 6.460516, 6.866647, and 7.090416. The difference in the $R_p^2$ between the best six- and seven-variable models was 0.0012. For eight-variable models, the best three values of $C_p$ were 7.2304[illegible]5, 7.827507, and 7.845395, and the difference in $R_p^2$ between the best seven- and eight-variable models was 0.0007. The difference between the values of $R_p^2$ for the best eight- and nine-variable models was 0.0001. Since the difference was so small, models with more than eight variables were not considered. For the eight-variable model with the lowest value of $C_p$, the multiple correlation coefficient of was 0.31845915, thus accounting for approximately 99% of the value of the multiple correlation coefficient for the 15 variable model.

In order to confirm the choice of the best six-, seven- and eight- variable models, the automated SAS procedures STEPWISE, MAXR, and MINR were run. The Stepwise regression procedure found only six variables that met the 0.15 significance level criterion for entry into the model. The six, in order of importance, were N, S, N*DA, S*DA, DA, and N*S. These were the variables that made up the best six-variable model according to the $C_p$ criterion. MAXR and MINR produced the same six-variable model with the exception that DA was replaced by N*S*DA. These procedures also produced models that differed from those given by using the $C_p$ criterion only in that DA was replaced by an interaction term containing DA.

It was decided on the basis of the information gathered from the automated procdures that the best three best six-, seven-, and eight-variable models selected according to the $C_p$ criterion should be subjected to the PROC REG SAS procedure to examine the adjusted coefficient of multiple correlation. The adjusted coefficient is important since it takes into cosideration the number of parameters in the model, whereas $R_p^2$ never decreases as p increases. The definition of the adjusted multiple correlation coefficient shows that $R_{a(p)}^2$ increases exactly when $MSE_p$ decreases. When PROC REG was applied to the nine models selected on the basis of $C_p$ and the automated procedures, $R_{a(p)}^2$ was 0.3131 for the six-variable model described above. This was the best value for this measure for any of the models with six variables. The best that any seven-variable model did with this measure was 0.3137, and the best for any of the models with eight variables was 0.3139. The best model with seven variables was the same as the best model with six variables with the exception that the variable P*S was added. The best eight-variable model contained the six variables that appeared in the best six-variable model along with N*P and P. The small differences in $R_{a(p)}^2$ and the fact that the six most important independent variables were identical among the best models strongly suggested the choice of the model with DA,N,S,N*DA,S*DA, and N*S as the final regression model. An F-test, based on the reduction in the error sum of squares in going from the best six variable to the best eight variable model, was conducted to determine if the two additional variables N*P and P were needed in the model. The test showed that the two terms involving P and N*P are statistically insignificant. Hence, the best six-variable model was chosen to explain much of the variance in the number of iterations required by the DCA and the TCA to find high quality answers. The regression function was found to be

$$\begin{aligned} Iterations = {} & 32.35666667{*}DA + 5.14224378{*}N - 5.48882488{*}Special \\ & - 2.05090000{*}N{*}DA + 1.34573333{*}Special{*}DA \\ & + 0.02092537{*}N{*}Special - 56.41462687 \end{aligned} \tag{6.11}$$

The significance of the algorithm in determining the number of iterations is seen from the fact that the variable DA appears in three of the terms in the regression function. The graphs in Figures (26-28) show the relationship between the two regression curves for the DCA and TCA for the different proportions of special vertices. From these graphs it is apparent that the tailored cooling schedule was superior to the dynamic cooling schedule for large N in the range of values of N tested in the experiment. Furthermore, it appears that the superiority of the tailored algorithm increases as the number of nodes increases.

Table XIV. AVERAGE DEVIATION OF LENGTH FROM THE OPTIMUM FOR EACH COMBINATION OF PARAMETER VALUES FOR 20 AND 40 VERTICES

| | | | Average Deviation Above Optimum (%) | | | |
|---|---|---|---|---|---|---|
| | | | TCA | | DCA | |
| n | $p_e$ | special | Best Answer | Final Answer | Best Answer | Final Answer |
| 20 | .05 | 5 | 1.85 | 2.65 | 2.01 | 4.22 |
| | | 10 | 1.28 | 2.94 | 1.39 | 2.47 |
| | | 15 | 1.03 | 1.54 | 2.20 | 3.30 |
| | .25 | 5 | 1.55 | 0.56 | 2.07 | 3.56 |
| | | 10 | 1.75 | 3.34 | 1.46 | 3.77 |
| | | 15 | 0.62 | 2.01 | 1.49 | 2.98 |
| | .50 | 5 | 2.47 | 3.15 | 2.76 | 3.05 |
| | | 10 | 2.43 | 3.87 | 1.60 | 4.55 |
| | | 15 | 1.04 | 2.18 | 0.75 | 3.91 |
| | .90 | 5 | 1.33 | 3.85 | 2.85 | 4.03 |
| | | 10 | 2.30 | 3.74 | 2.03 | 3.72 |
| | | 15 | 0.98 | 2.44 | 1.34 | 2.77 |
| 40 | .05 | 10 | 0.95 | 3.22 | 1.79 | 3.64 |
| | | 20 | 2.53 | 2.76 | 2.87 | 4.19 |
| | | 30 | 2.59 | 3.69 | 2.88 | 3.63 |
| | .25 | 10 | 1.84 | 3.73 | 1.53 | 4.70 |
| | | 20 | 0.77 | 3.03 | 1.29 | 4.05 |
| | | 30 | 1.05 | 2.77 | 1.49 | 3.63 |
| | .50 | 10 | 2.31 | 3.05 | 2.61 | 4.49 |
| | | 20 | 2.25 | 3.24 | 2.74 | 3.97 |
| | | 30 | 1.03 | 2.37 | 1.09 | 4.72 |
| | .90 | 10 | 2.42 | 2.87 | 1.34 | 3.43 |
| | | 20 | 1.40 | 2.45 | 1.14 | 4.58 |
| | | 30 | 0.92 | 3.59 | 1.90 | 3.07 |

Table XV. AVERAGE DEVIATION OF LENGTH FROM THE OPTIMUM FOR EACH COMBINATION OF PARAMETER VALUES FOR 60 AND 80 VERTICES

| | | | Average Deviation Above Optimum (%) | | | |
|---|---|---|---|---|---|---|
| | | | TCA | | DCA | |
| n | $p_e$ | special | Best Answer | Final Answer | Best Answer | Final Answer |
| 60 | .05 | 15 | 1.82 | 3.62 | 1.76 | 4.14 |
| | | 30 | 2.47 | 3.85 | 3.02 | 4.29 |
| | | 45 | 1.84 | 3.17 | 2.25 | 3.92 |
| | .25 | 15 | 2.14 | 3.29 | 1.98 | 3.82 |
| | | 30 | 2.03 | 2.89 | 2.47 | 4.01 |
| | | 45 | 1.37 | 1.70 | 2.81 | 3.38 |
| | .50 | 15 | 2.49 | 3.71 | 2.97 | 4.28 |
| | | 30 | 2.48 | 3.75 | 1.84 | 4.37 |
| | | 45 | 1.02 | 3.15 | 2.08 | 2.66 |
| | .90 | 15 | 2.03 | 2.76 | 2.22 | 3.07 |
| | | 30 | 2.61 | 3.84 | 2.46 | 3.71 |
| | | 45 | 0.97 | 1.38 | 1.05 | 1.83 |
| 80 | .05 | 20 | 2.19 | 3.75 | 1.97 | 3.22 |
| | | 40 | 1.96 | 2.83 | 1.53 | 4.16 |
| | | 60 | 1.66 | 2.07 | 1.69 | 3.95 |
| | .25 | 20 | 2.37 | 2.70 | 2.45 | 3.11 |
| | | 40 | 2.31 | 3.74 | 2.85 | 4.78 |
| | | 60 | 2.09 | 3.68 | 2.12 | 3.45 |
| | .50 | 20 | 2.24 | 3.03 | 2.17 | 2.94 |
| | | 40 | 1.71 | 3.86 | 2.98 | 4.61 |
| | | 60 | 2.11 | 2.60 | 1.53 | 4.07 |
| | .90 | 20 | 1.32 | 3.26 | 1.84 | 3.67 |
| | | 40 | 2.38 | 3.76 | 2.19 | 3.11 |
| | | 60 | 1.95 | 2.83 | 2.05 | 4.56 |

Table XVI. AVERAGE TOTAL RUNNING TIME FOR EACH COMBINATION OF PARAMETER VALUES FOR GRAPHS WITH 20 OR 40 NODES

| | | | TCA | DCA |
|---|---|---|---|---|
| n | $p_e$ | special | Run Time (sec.) | Run Time (sec.) |
| 20 | .05 | 5 | 37 | 33 |
| | | 10 | 33 | 39 |
| | | 15 | 28 | 23 |
| | .25 | 5 | 41 | 29 |
| | | 10 | 34 | 30 |
| | | 15 | 25 | 23 |
| | .50 | 5 | 38 | 30 |
| | | 10 | 37 | 32 |
| | | 15 | 32 | 25 |
| | .90 | 5 | 34 | 24 |
| | | 10 | 27 | 22 |
| | | 15 | 22 | 18 |
| 40 | .05 | 10 | 154 | 127 |
| | | 20 | 125 | 102 |
| | | 30 | 109 | 87 |
| | .25 | 10 | 168 | 149 |
| | | 20 | 112 | 104 |
| | | 30 | 106 | 99 |
| | .50 | 10 | 217 | 181 |
| | | 20 | 185 | 136 |
| | | 30 | 118 | 106 |
| | .90 | 10 | 205 | 177 |
| | | 20 | 189 | 136 |
| | | 30 | 102 | 71 |

Table XVII. AVERAGE TOTAL RUNNING TIME FOR EACH COMBINATION OF PARAMETER VALUES FOR GRAPHS WITH 60 OR 80 NODES

| | | | TCA | DCA |
|---|---|---|---|---|
| n | $p_e$ | special | Run Time (sec.) | Run Time (sec.) |
| 60 | .05 | 15 | 3476 | 1128 |
| | | 30 | 3096 | 1238 |
| | | 45 | 3174 | 1053 |
| | .25 | 15 | 3979 | 1248 |
| | | 30 | 2890 | 1146 |
| | | 45 | 3081 | 1063 |
| | .50 | 15 | 3540 | 1424 |
| | | 30 | 3058 | 1296 |
| | | 45 | 2540 | 1377 |
| | .90 | 15 | 2561 | 1703 |
| | | 30 | 1888 | 1425 |
| | | 45 | 1355 | 451 |
| 80 | .05 | 20 | 23267 | 17219 |
| | | 40 | 16542 | 14845 |
| | | 60 | 9873 | 11596 |
| | .25 | 20 | 28324 | 15083 |
| | | 40 | 14399 | 12580 |
| | | 60 | 9225 | 10406 |
| | .50 | 20 | 24778 | 12673 |
| | | 40 | 15671 | 11655 |
| | | 60 | 9733 | 9072 |
| | .90 | 20 | 35328 | 13749 |
| | | 40 | 24302 | 10418 |
| | | 60 | 9355 | 8947 |

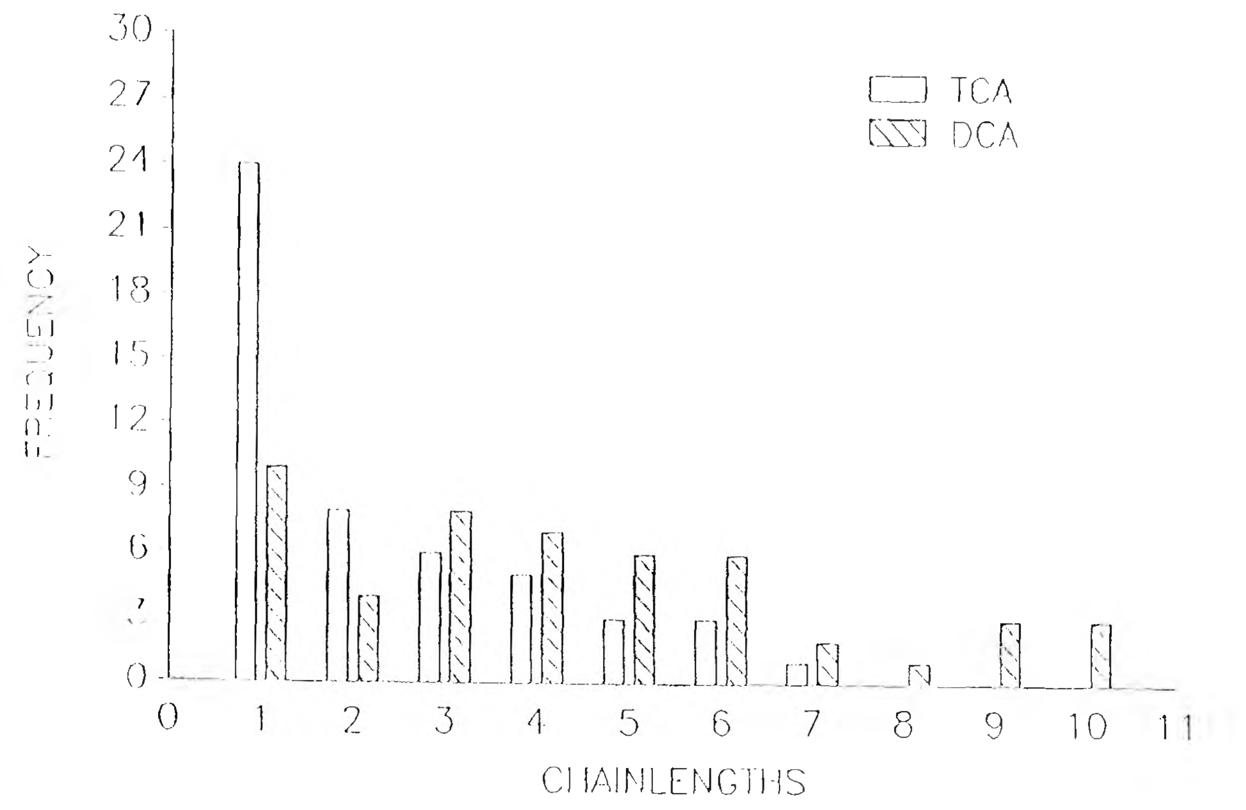

Figure 17. Distribution of First Occurrences of a Near-Optimal Solution for 50 Graphs with Parameter values of n = 40, $p_r = 0.05$, and special = 10.

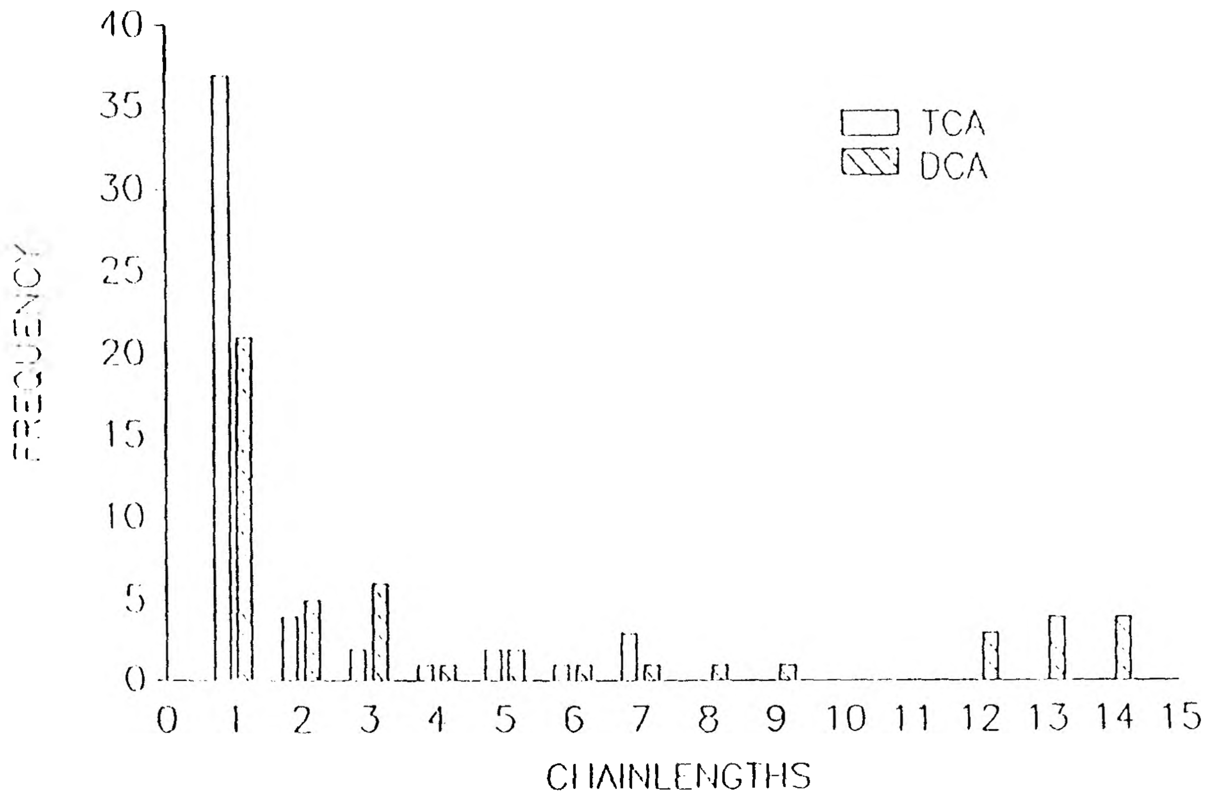

Figure 18. Distribution of First Occurrences of a Near-Optimal Solution for 50 Graphs with Parameter values of n = 40, $p_s$ = 0.05, and special = 20.

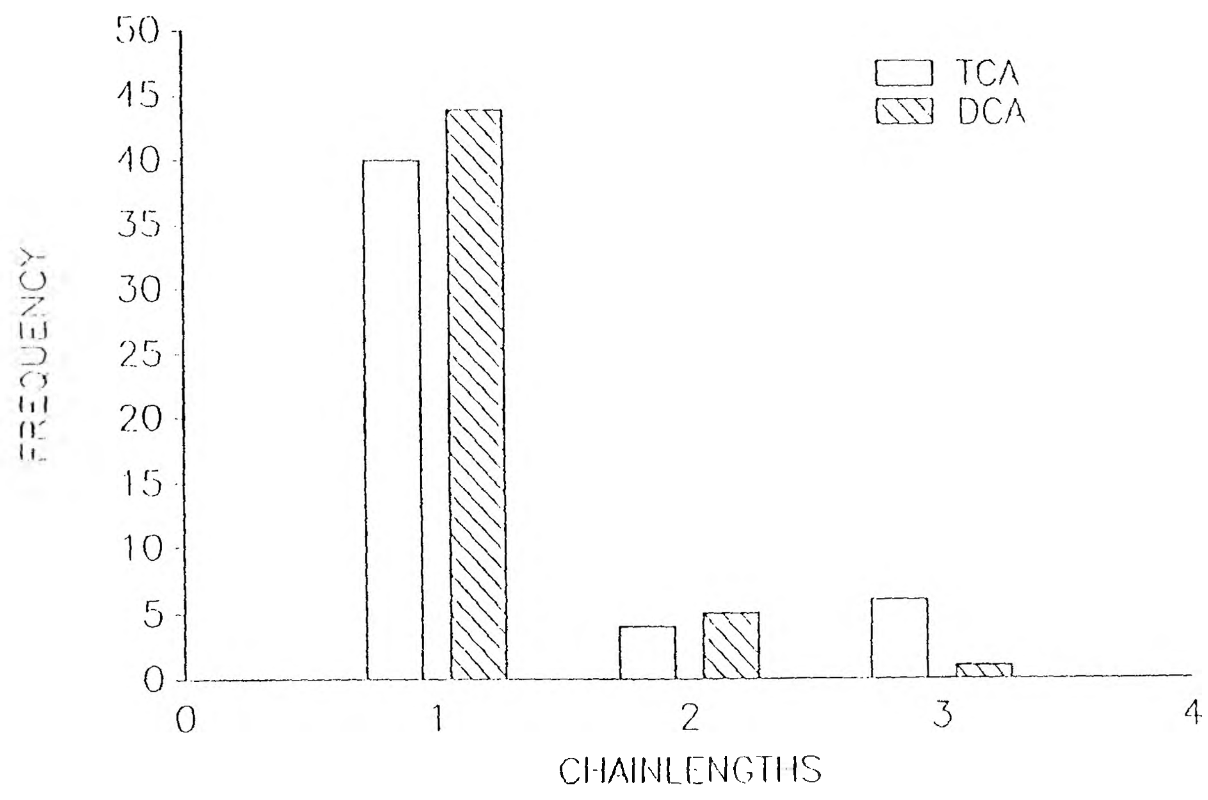

Figure 19. Distribution of First Occurrences of a Near-Optimal Solution for 50 Graphs with Parameter values of $n = 40$, $p = 0.05$, and special = 30.

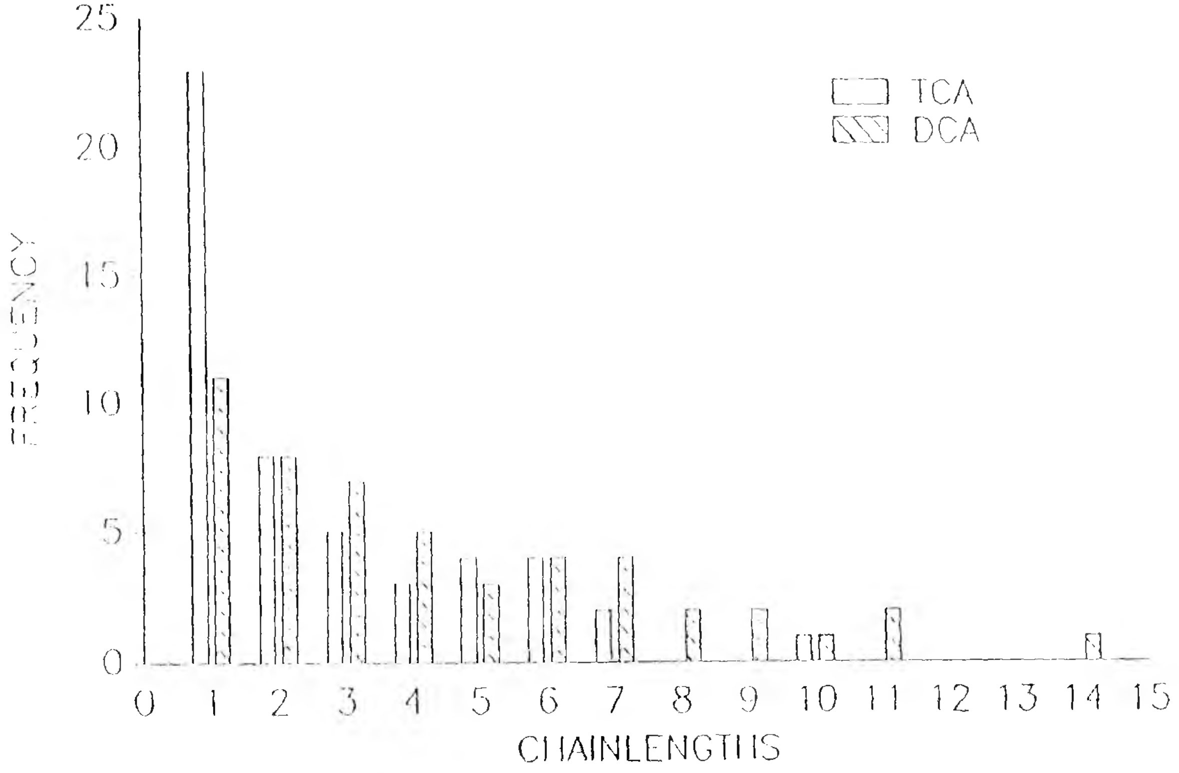

Figure 20. Distribution of First Occurrences of a Near-Optimal Solution for 50 Graphs with Parameter values of n = 60, $p_e = 0.05$, and special = 15.

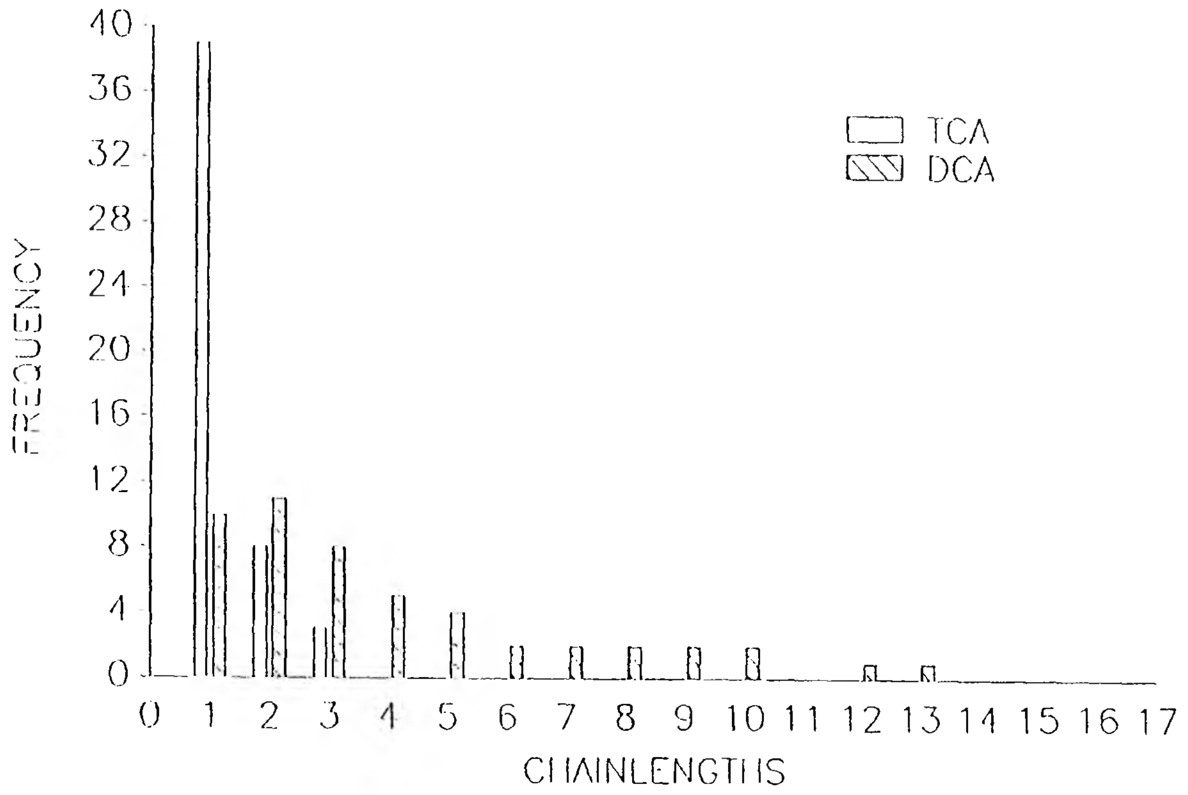

Figure 21. Distribution of First Occurrences of a Near-Optimal Solution for 50 Graphs with Parameter values of n = 60, $p_e = 0.05$, and special = 30.

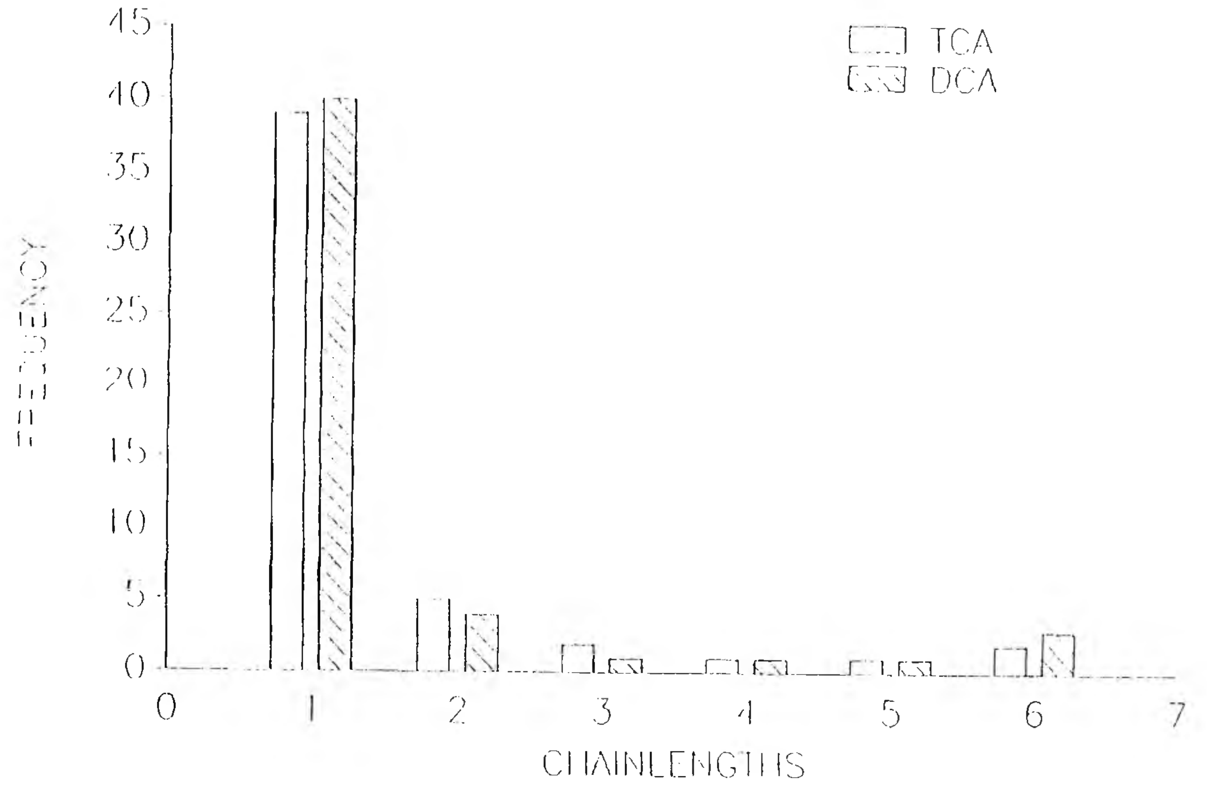

Figure 22. Distribution of First Occurrences of a Near-Optimal Solution for 50 Graphs with Parameter values of n = 60, $p_e$ = 0.05, and special = 45.

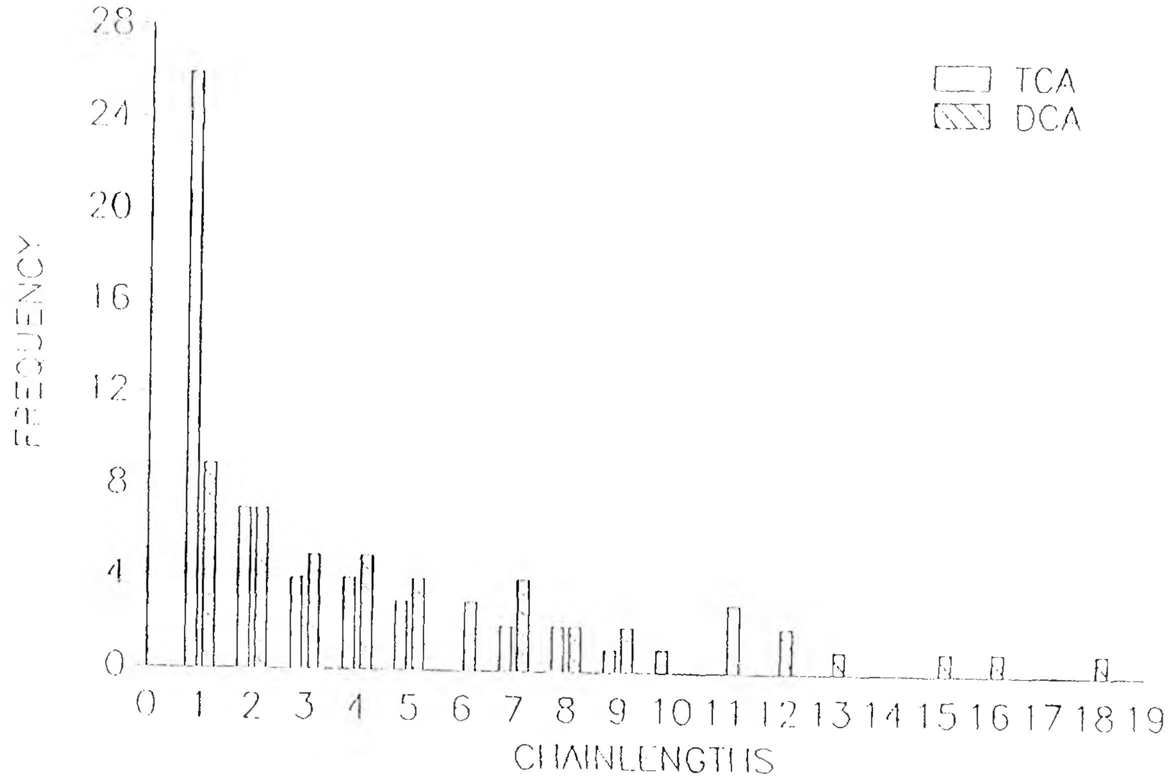

Figure 23. Distribution of First Occurrences of a Near-Optimal Solution for 50 Graphs with Parameter values of n = 60, $p_r$ = 0.90, and special = 15.

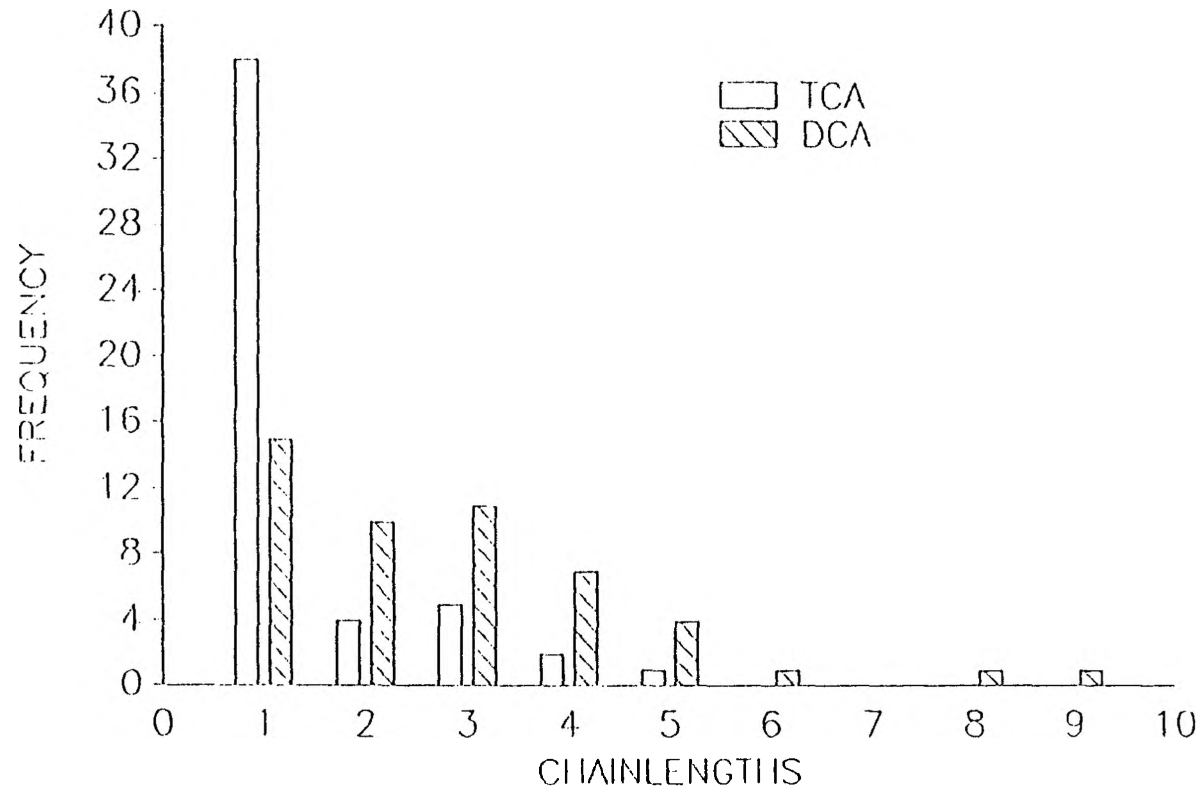

Figure 24. Distribution of First Occurrences of a Near-Optimal Solution for 50 Graphs with Parameter values of n = 60, $p_e = 0.90$, and special = 30.

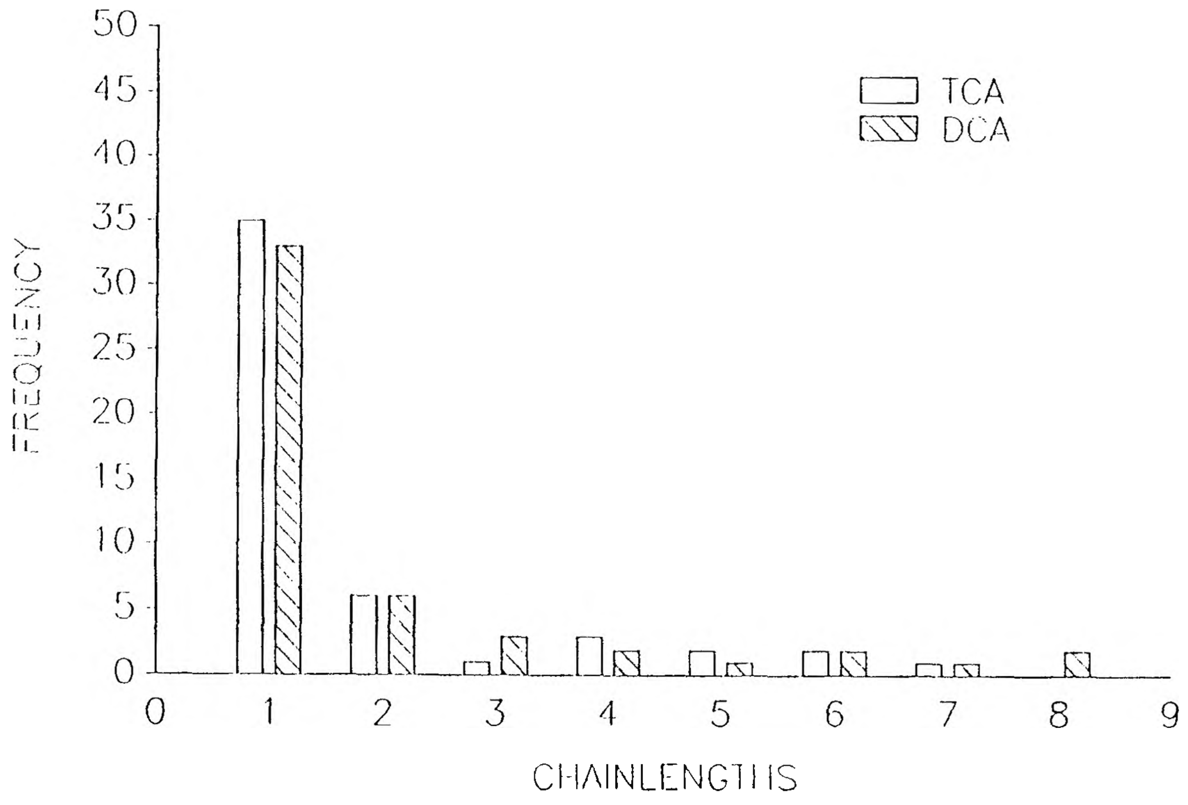

Figure 25. Distribution of First Occurrences of a Near-Optimal Solution for 50 Graphs with Parameter values of n = 60, $p_r = 0.90$, and special = 45.

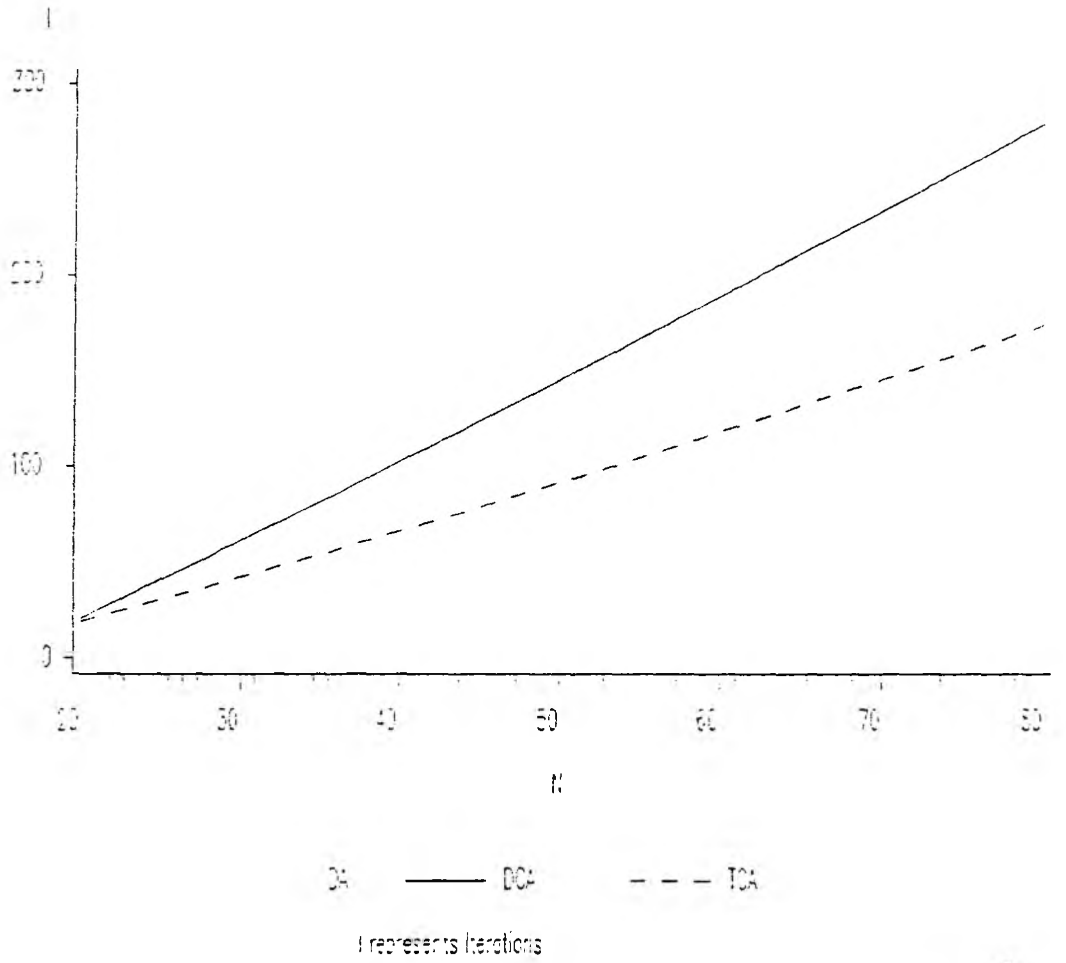

Figure 26. Regression Functions for the DCA and the TCA Special = $\frac{n}{4}$

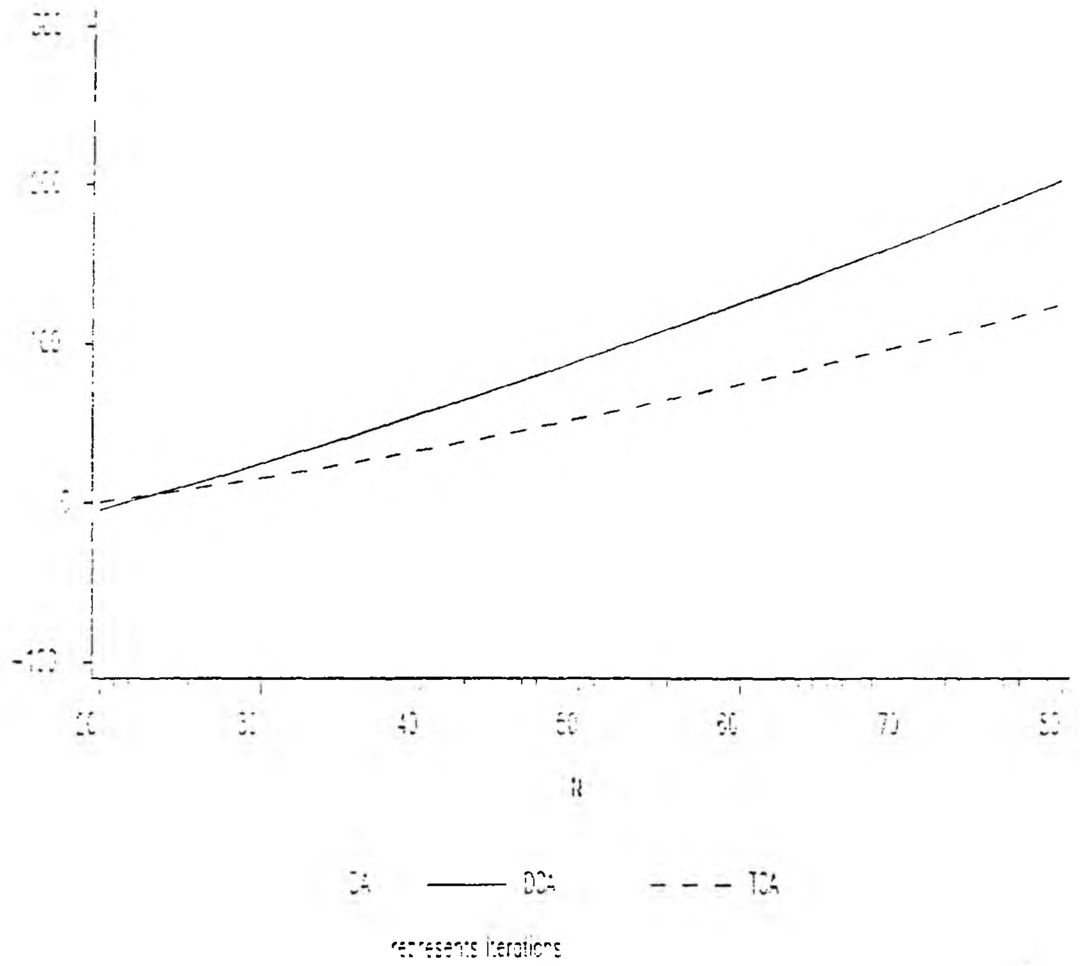

Figure 27. Regression Functions for the DCA and the TCA Special = $\frac{n}{2}$

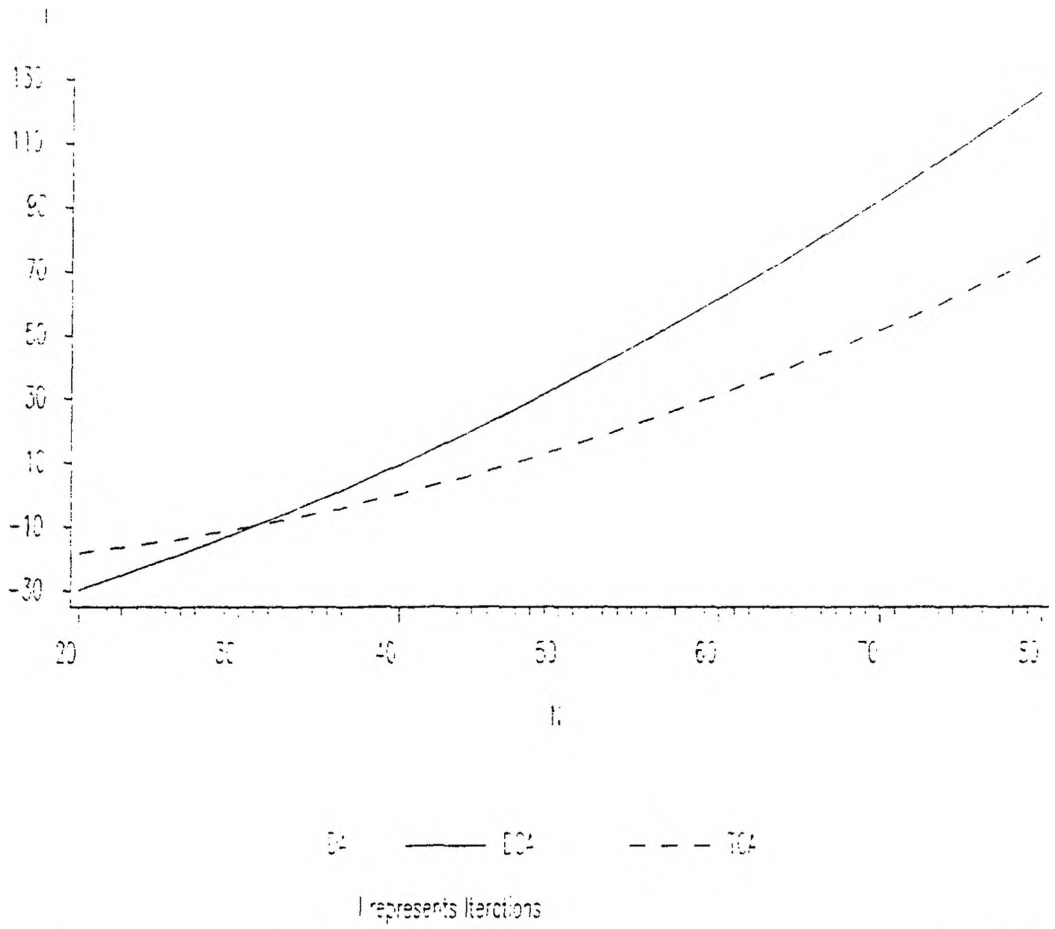

Figure 28. Regression Functions for the DCA and the TCA Special = $\frac{3n}{4}$

# VII. SIMULATED ANNEALING, BRANCH AND BOUND, AND THE DSP

In this chapter the tailored simulated annealing algorithm, the more reliable of the two annealing algorithms discussed in this dissertation, is compared with the branch and bound technique on a binary search tree with respect to running time and accuracy in the solution of the Directed Steiner Problem on graphs. The dual ascent algorithm of Richard T. Wong [112] is used to obtain lower bounds for the branch and bound process. An initial feasible upper bound is found by applying the minimum spanning tree algorithm for directed graphs [21, 29, 99] to a subset of the nodes found by Wong's procedure. In effect, the method employed by Wong is a means for making an intelligent guess about which Steiner points should be included in the minimal length Steiner tree. If the lower bound produced by the dual ascent algorithm agrees with the feasible upper bound, then the problem is solved optimally and a branch and bound procedure is not necessary. Otherwise, the branch and bound technique is applied to find the optimal solution. The heuristic algorithm given by Wong is as follows [112].

1. Use the dual ascent procedure to find a lower bound for the optimal Steiner tree length.
   Let $G'(Q, E')$ be the graph obtained when the ascent algorithm terminates. All of the special nodes including node 1 must belong to $Q$, because the algorithm terminates only when the set of special nodes is connected.
2. Find a minimal directed spanning tree T on the graph $G'$.
3. If a node s is a leaf of T and s is a Steiner vertex, then delete s and the arc incident to it. Repeat this process until no further deletions are possible. Denote the resulting tree by $T'$. This arcs in this tree constitute a feasible solution to the DSP since every special vertex in $T'$ is connected to node 1. The cost of $T'$ is an upper bound for the optimum cost.

For each combination of parameters the number of graphs in the set of test problems for which the initial feasible solution was optimal or near-optimal and the average running time for the algorithm of Wong are shown in Tables (XVIII-XIX). There were 10 graphs for each combination of parameters. The data in Tables(XVIII-XIX) indicate that the average total running time increased rapidly as the number of vertices grew. The running times in most cases declined for fixed values of n and $p_e$.

Table XVIII. RESULTS OF THE APPLICATION OF THE ALGORITHM OF WONG TO GRAPHS WITH 20 OR 40 NODES

| n | $p_e$ | special | Optimum Answer | Near Optimum Answer | Average Run Time (sec.) |
|---|---|---|---|---|---|
| 20 | .05 | 5 | 6 | 4 | 0.7 |
| | | 10 | 6 | 4 | 0.6 |
| | | 15 | 9 | 1 | 0.3 |
| | .25 | 5 | 4 | 5 | 1.5 |
| | | 10 | 7 | 2 | 1.7 |
| | | 15 | 8 | 2 | 0.5 |
| | .50 | 5 | 4 | 4 | 1.4 |
| | | 10 | 5 | 3 | 1.2 |
| | | 15 | 7 | 2 | 0.4 |
| | .90 | 5 | 5 | 1 | 0.8 |
| | | 10 | 5 | 2 | 0.7 |
| | | 15 | 7 | 2 | 0.5 |
| 40 | .05 | 15 | 5 | 1 | 52.2 |
| | | 30 | 5 | 2 | 42.0 |
| | | 45 | 7 | 2 | 27.8 |
| | .25 | 15 | 5 | 2 | 34.3 |
| | | 30 | 4 | 4 | 30.1 |
| | | 45 | 6 | 2 | 19.8 |
| | .50 | 15 | 4 | 4 | 39.5 |
| | | 30 | 5 | 3 | 35.6 |
| | | 45 | 7 | 2 | 20.1 |
| | .90 | 15 | 5 | 3 | 20.7 |
| | | 30 | 5 | 3 | 8.5 |
| | | 45 | 6 | 2 | 15.2 |

Table XIX. RESULTS OF THE APPLICATION OF THE ALGORITHM OF WONG TO GRAPHS WITH 60 OR 80 NODES

| n | $p_e$ | special | Optimum Answer | Near Optimum Answer | Average Run Time (sec.) |
|---|---|---|---|---|---|
| 60 | .05 | 10 | 3 | 4 | 73.7 |
| | | 20 | 3 | 3 | 65.5 |
| | | 30 | 5 | 2 | 41.3 |
| | .25 | 10 | 4 | 1 | 86.9 |
| | | 20 | 5 | 2 | 88.4 |
| | | 30 | 6 | 2 | 53.8 |
| | .50 | 10 | 3 | 3 | 80.3 |
| | | 20 | 5 | 2 | 77.9 |
| | | 30 | 4 | 3 | 60.4 |
| | .90 | 10 | 4 | 3 | 118.0 |
| | | 20 | 5 | 2 | 78.2 |
| | | 30 | 6 | 3 | 64.1 |
| 80 | .05 | 20 | 2 | 5 | 265.9 |
| | | 40 | 3 | 3 | 178.5 |
| | | 60 | 5 | 2 | 115.2 |
| | .25 | 20 | 3 | 3 | 293.3 |
| | | 40 | 4 | 2 | 261.6 |
| | | 60 | 4 | 3 | 152.7 |
| | .50 | 20 | 4 | 3 | 353.2 |
| | | 40 | 5 | 2 | 312.0 |
| | | 60 | 5 | 3 | 189.8 |
| | .90 | 20 | 4 | 1 | 319.6 |
| | | 40 | 5 | 2 | 289.5 |
| | | 60 | 5 | 3 | 186.1 |

Figure 29 shows the percentage of optimal answers from an application of Wong's method as a function of the number of vertices n. As n gets larger, the procedure becomes increasingly less accurate. If the rate of decline in accuracy indicated in Figure 29 continues as n increases beyond 80, then it is of interest to compare the commonly used deterministic method of branch and bound with a probabalistic one such as simulated annealing. A comparison of these methods in solving the Directed Steiner Problem on graphs is the topic of the remainder of this dissertation.

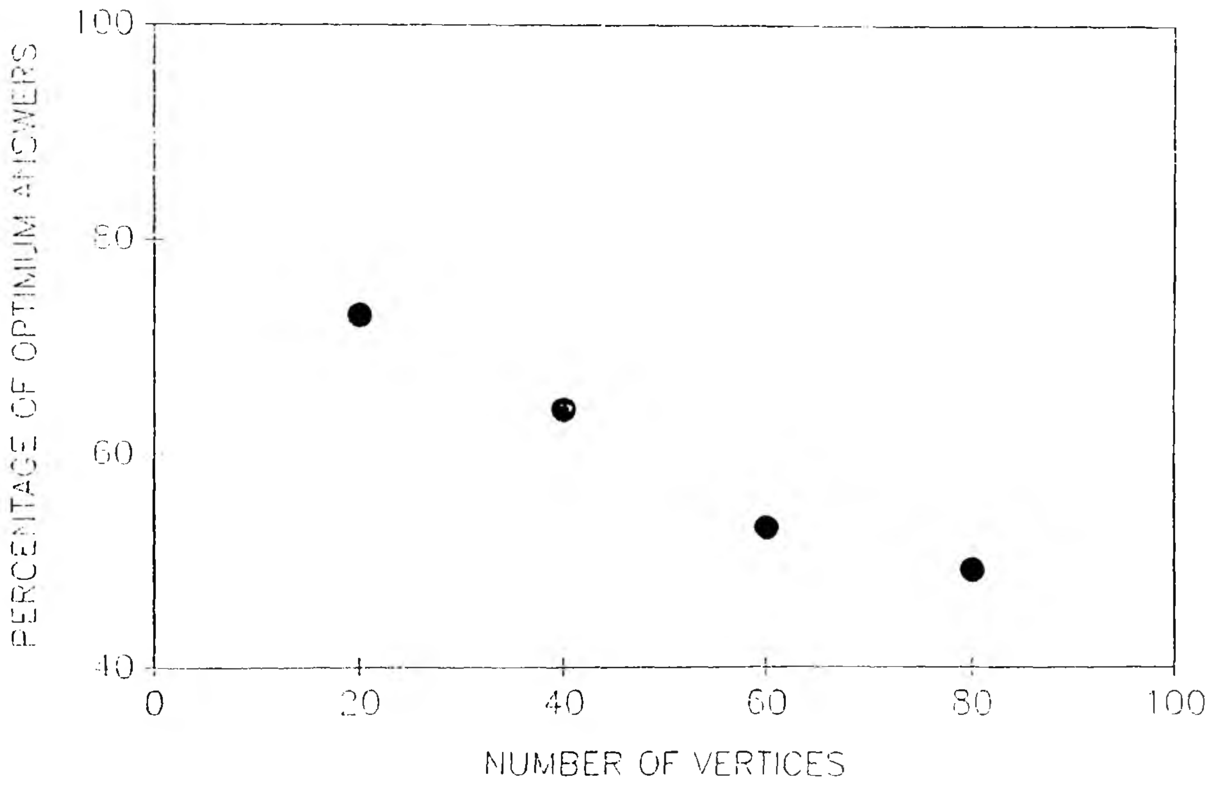

Figure 29. The percentage of optimum answers using the Algorithm of Wong as a function of the number of vertices.

The idea of comparing branch and bound to simulated annealing brings up the question of whether to begin the cooling schedule with the better-than-average answer found by Wong's algorithm, which is the starting point for branch and bound, or to begin with a random configuration as was done in the comparison of the two versions of simulated annealing. In order to answer this question, the tailored cooling algorithm was applied to the graphs with 60 or 80 nodes five times starting from random initial configurations and five times starting from the final configuration yielded by the heuristic technique of Wong. The cooling schedules were allowed to run only three chainlengths. The average mean cost of the third chainlength was calculated in each case. Table XX shows the average of the solutions for the five iterations for each set of parameters for both starting procedures. The averages were better in all but two cases when the simulated annealing algorithm began with the final configuration derived by the algorithm of Wong. Hence, it was decided that in the comparison with branch and bound that simulated annealing would begin with the better-than-average solution obtained from Wong's heuristic. However, in the two cases in which the random starting configuration led to a more accurate answer than did the last configuration from Wong's algorithm, the dual ascent algorithm produced a solution that included several Steiner points that were not in the optimum solution, but which did yield a local minimum. That is, the solutions which were close to the one given by the heuristic, in the sense that they involved only one or two changes in the choice of Steiner points to use, were larger than the one derived from Wong's algorithm. The optimum, however, required in each case that at least four changes be made to the result of the heuristic method. Thus, although for the majority of graphs a better-than-average beginning configuration leads to a faster near optimal solution, there are examples where the opposite is true. The conclusion from this experiment is that in order for a superior starting configuration to be valuable, the kind of solutions obtained from the heuristic method of Wong must take advantage of insights into the problem instance that are not possible for a general procedure like simulated annealing that attacks every instance in the same way. If the structure of the graph is such that Wong's technique does not adequately analyze it, then a random initial configuration is as good or better for annealing.

In order to determine whether the TCA was as efficient as branch and bound in locating near-optimal solutions, the following experiment was conducted. For all combinations of standard parameters with the exception of those with 20 nodes, 4 graphs were found for which the heuristic procedure did not reach a near-optimal answer. Thus, the TCA was applied to 144 graphs, and the initial configuration for each run was the configuration at which Wong's heuristic had terminated. The time required by annealing until a solution within 3% of the optimum or the best answer achieved was recorded for each run. Then the result was compared to the time required by branch and bound to obtain a solution at least as good as the one found by annealing. Graphs with 20 nodes were not included because a near optimal or better solution was almost always found by an application of Wong's algorithm. Tables (XXI-XXIV) show the information gathered by this experiment. For 114 of the graphs the running time for the simulated annealing algorithm was shorter than the running time for the branch and bound method.

A regression analysis was applied to the data in Tables (XXI-XXIV) in an effort to find some reliable statistical way to measure the performance of the tailored simulated annealing algorithm relative to the branch and bound method for the Directed Steiner Problem in graphs. The variables N, $p_e$, and Special used to generate the random graphs, and a classification variable, DA, representing the two methods(0 for the branch and bound technique and 1 for the annealing algorithm) were selected as likely important independent variables for a regression model. The

SAS General Linear Models Procedure(GLM) was first applied to the full model with N, $p_e$, Special, and DA and all of their product terms, a total of 15 variables. The multiple correlation coefficient for this model was 0.357217. This model was compared to the reduced model with all of the terms containing DA removed. For this reduced model, $R_p^2$ was equal to 0.270101. An $F$ test using the formula

$$F = \frac{\frac{SSE_{full} - SSE_{reduced}}{df_{full} - df_{reduced}}}{MSE_{full}}, \tag{7.1}$$

where $df$ stands for the number of degrees of freedom in the respective sum of squares, was applied in order to compare the full model with all 15 variables to the second model. Substituting the values from the models into Equation 7.1 gave

$$F = \frac{\frac{749235357.8 - 850779376.4}{272 - 280}}{2754541.8}. \tag{7.2}$$

This yielded *Calculated* $F = 4.61$. From the tables for the $F$ distribution, $F(8, 280, .999) = 3.27$ [78]. Thus, it was concluded with less than a 0.1% chance of error that at least one of the terms containing DA was necessary to the model. Next, the variable DA was retained, but the product terms containing DA were deleted. For this second reduced model, $R_p^2$ was equal to 0.326668. An $F$ test was done to compare this new reduced model to the full model with 15 variables. From Equation 7.1,

$$F = \frac{\frac{749235357.8 - 784669420.4}{272 - 279}}{2754541.8}. \tag{7.3}$$

This reduced to *Calculated* $F = 1.84$. Tables for the $F$ distribution give $F(7, 279, .90) = 1.72$ [78]. Hence, with less than a 10% chance of error it was determined that at least one of the product terms containing DA was significant enough to be in the model.

A SAS RSQUARE procedure was applied to the full model with 15 variables. The variable DA was required in all the models reported. There were two models with four variables, six models with five variables, and twelve models with six variables that had values of $C_p < p$. Some of the models with seven and eight variables also had low values of $C_p$, but the difference in the multiple correlation coefficient between the best six-variable model and the best seven-variable model was only approximately 0.0013. The increase in complexity did not seem to be worth such a small gain in $R_p^2$. Thus, the number of models studied further was limited to twenty.

The automated procedures STEPWISE, MAXR, AND MINR were examined. Only the four variables DA, N, N*P, and N*S were able to meet the 0.15 significance requirement for the STEPWISE procedure. MAXR AND MINR both produced the same four-variable model as the Stepwise procedure. This model differed from the two models suggested by RSQUARE. Likewise, MAXR AND MINR produced identical best five- and six-variable models, but these models were not suggested by the application of the "smallest $C_p$" criterion to the models given by RSQUARE.

In order to understand the models better, the predicted values of Time were plotted against N for all combinations of $\frac{S}{N}$ and $p_e$ for the twenty models suggested by RSQUARE, and the three models given by the automated procedures. All of the models showed graphs that for some combinations of the parameters predicted negative values for Time at N = 40 for the simulated annealing algorithm. Also, the values of Time predicted by the full model with 15 variables were plotted against

N. For this model, several of the graphs showed negative values at N = 40 and little difference in Time between N = 60 and N = 80 for the TCA. These results led to the conclusion that other variables needed to be included in any model designed to show the differences in Time produced by the branch and bound technique and the tailored cooling schedule.

The following set of independent variables was added to the original set of 15 variables to form a set of candidate variables to be included in a model: $N^2$, $N^3$, $\exp^{\sqrt{N}}$. $\ln N$, $\sqrt{N}$, $N^2P$, $S^2$, $\frac{S}{N}$, $N^2DA$, $S \ln N$, $\sqrt{N}\,S$, $\frac{S^2}{N}$. Three criteria for selection of a model were used. They were:

1. $R_{a(p)}{}^2$,

2. Realistic predictions of values of Time for all
   all combinations of parameters,

3. Evidence of low multicollinearity between independent
   variables.

The second criterion meant that predicted values of Time were not permitted to be negative, nor were they allowed to decrease or show otherwise clearly incorrect patterns of change as N increased. In some candidate models, predicted values of Time decreased as N increased from 40 to 60 or from 60 to 80. Some of this behavior was due to correlations among the independent variables themselves. Hence, the third criterion followed naturally from the second. In order to measure multicollinearity for each model studied, the variance inflation factors for each independent variable, which indicate the presence or absence of multicollinearity, and the condition numbers were obtained by a SAS REG procedure.

The following steps were repeated until a satisfactory model was obtained. First, a SAS RSQUARE procedure was applied to a set of 15 of the twenty-seven candidate variables. Each of the models reported by RSQUARE was required to include DA. Models with 5, 6, 7, or 8 variables that had values of $C_p \leq p + 1$, and whose variables figured prominently in the automated procedures STEPWISE, MAXR, and MINR were initially accepted for further consideration. Second, for the models selected in the first step, the predicted values of Time were plotted against N for all combinations of parameters. For all of the candidate models, some of the graphs that were produced violated the second constraint by exhibiting obviously incorrect predictions of running time for the TCA. The reason that it was difficult for models to give satisfactory results for all cases was that the rate of increase in running time as N increased varied not only between algorithms, but also among values of the ratio $\frac{S}{N}$. As this ratio grew larger, the rate of increase in time diminished for both algorithms. Very few models were flexible enough to accomodate these different rates of change, which were further complicated by the densities and individual topological characteristics of the graphs. Third, for those models that showed reasonable looking graphs with only a few exceptions, multicollinearity statistics were collected. Variables were then deleted from the model or exchanged for other variables in order to remove any substantial multicollinearity. Plots of the resulting models were made in order to study the effects of the changes. Those models which looked promising frequently had large $C_p$ values indicating that the models were underspecified. However, since the narrow range of data necessarily made it likely that the predictions of running time were not very reliable for at least some of the parameter combinations, it was decided that a marginally underspecified model with reasonably shaped graphs was preferable to one with better $C_p$ values but which resulted in a few very eccentric looking plots. Given data over a larger range of N, the highly specified models might have a much different composition anyway. Moreover, the purpose of the regression analysis was only to study the relative importance of the effects of the different algorithms on the near-optimum running time. It was not to make highly accurate predictions about actual values of near-optimum running times for each algorithm at given values of the parameters. Thus, values of $C_p$ slightly above p were tolerated. It should be noted that $C_p$ is only an estimate of a quantity that would be less than or equal to $p+1$ for models that are not underspecified. Sampling variations can produce a $C_p$ value slightly higher than $p+1$ even when the model is not underspecified.

Forty models reached Step 2 of the testing process. For these models, $R_{a(p)}{}^2$ ranged from 0.3601 to 0.3875. These values were considered to be sufficiently close together that any of them was acceptable. Only one model produced by the process satisfied all three criteria. It consisted of the variables DA, $N^2$, N*S, S, N*DA, N*P, and N*P*DA. It has an adjusted multiple correlation coefficient of 0.3631. There is little multicollinearity in the model. The largest variance inflation factor is approximately 48.4 for $N^2$, and the largest condition number is approximately 46.3. For this model, the plots of predicted values of near-optimum running time against N showed no negative values nor any unusual patterns of change as N increased. The regression function obtained was the following.

$$\begin{aligned} Time = {} & 1318.56993*DA + 1.17338997*N^2 - 1.53803718*N*P \\ & - 43.86114141*N*DA - 15.52781185*N*P + 59.36700110*S. \\ & + 13.01862537*N*P*DA - 1266.59934 \end{aligned} \tag{7.4}$$

Graphs of the regression curves for branch and bound and the TCA are shown in Figures (30-41).

The small values of the adjusted multiple correlation coefficient are perhaps due to the randomness of the graph structures, because the algorithm of Wong produces vastly different running times for graphs generated with the same set of parameters, but with different seeds to the random number generator. In addition, the method of simulated annealing itself injects randomness into the sequence of configurations examined. Another factor is the order in which Steiner nodes are examined in the depth first search of the binary tree in branch and bound. If the order is fortuitous, the tree is quickly pruned and an optimum solution found, but, in the average case, a large number of nodes must be checked.

In this chapter we have shown that the quality of the solutions obtained by the heuristic used by Wong to find feasible solutions appears to decline as the size of the problem increases. Evidence has been presented suggesting that simulated annealing generally finds a near-optimal answer faster by starting at the final configuration of the algorithm of Wong than by beginning at a random configuration. A comparison between the near-optimal running times for the TCA and branch and bound methods when applied to the DSP on 144 graphs has been presented. A regression analysis of the data indicates that the method is an important independent variable in any model attempting to understand the data. For the range of values used as parameters in this dissertation, the graphs shown in Figures (30-41) clearly illustrate the difference in the two techniques of simulated annealing and branch and bound when applied to the DSP.

Table XX. A COMPARISON OF AVERAGE SOLUTIONS AFTER 3 CHAINLENGTHS OF THE TCA BEGINNING AT A RANDOM CONFIGURATION AND AT THE CONFIGURATION YIELDED BY THE HEURISTIC OF WONG

| | | | Random Configuration | Final Heuristic Configuration |
|---|---|---|---|---|
| n | $p_e$ | special | Average Solution | Average Solution |
| 60 | .05 | 15 | 6.924 | 6.846 |
| | | 30 | 8.711 | 8.562 |
| | | 45 | 9.439 | 9.299 |
| | .25 | 15 | 1.814 | 1.787 |
| | | 30 | 2.531 | 2.506 |
| | | 45 | 3.174 | 3.241 |
| | .50 | 15 | 0.994 | 0.976 |
| | | 30 | 1.298 | 1.285 |
| | | 45 | 1.563 | 1.485 |
| | .90 | 15 | 0.396 | 0.405 |
| | | 30 | 0.763 | 0.710 |
| | | 45 | 0.892 | 0.845 |
| 80 | .05 | 20 | 7.420 | 7.601 |
| | | 40 | 9.888 | 10.068 |
| | | 60 | 10.836 | 10.024 |
| | .25 | 20 | 1.747 | 2.369 |
| | | 40 | 2.245 | 2.239 |
| | | 60 | 3.926 | 3.754 |
| | .50 | 20 | 1.070 | 1.056 |
| | | 40 | 1.165 | 1.143 |
| | | 60 | 1.583 | 1.542 |
| | .90 | 20 | 0.385 | 0.322 |
| | | 40 | 0.585 | 0.564 |
| | | 60 | 0.944 | 0.931 |

Table XXI. A COMPARISON OF THE TAILORED COOLING ALGORITHM AND THE BRANCH AND BOUND METHOD APPLIED TO THE DIRECTED STEINER TREE PROBLEM ON GRAPHS OF .05 DENSITY

| | | | TCA | | Branch and Bound | |
|---|---|---|---|---|---|---|
| $p_e$ | n | special | Run Time (sec.) | $\frac{\text{Solution}}{\text{Optimum}}$ | Run Time (sec.) | $\frac{\text{Solution}}{\text{Optimum}}$ |
| .05 | 40 | 10 | 25 | 1.005 | 186 | 1.000 |
| | | | 348 | 1.013 | 1397 | 1.010 |
| | | | 540 | 1.000 | 363 | 1.000 |
| | | | 73 | 1.000 | 2439 | 1.000 |
| | | 20 | 3 | 1.018 | 8 | 1.006 |
| | | | 288 | 1.023 | 694 | 1.015 |
| | | | 22 | 1.000 | 572 | 1.000 |
| | | | 334 | 1.012 | 138 | 1.010 |
| | | 30 | 2 | 1.014 | 12 | 1.012 |
| | | | 103 | 1.000 | 274 | 1.000 |
| | | | 388 | 1.000 | 735 | 1.000 |
| | | | 203 | 1.008 | 124 | 1.000 |
| | 60 | 15 | 298 | 1.000 | 31 | 1.000 |
| | | | 942 | 1.000 | 2234 | 1.000 |
| | | | 11 | 1.012 | 3855 | 1.008 |
| | | | 384 | 1.000 | 948 | 1.000 |
| | | 30 | 85 | 1.009 | 777 | 1.000 |
| | | | 2 | 1.000 | 6 | 1.000 |
| | | | 370 | 1.014 | 972 | 1.000 |
| | | | 573 | 1.018 | 561 | 1.000 |
| | | 45 | 36 | 1.000 | 92 | 1.000 |
| | | | 197 | 1.016 | 384 | 1.009 |
| | | | 428 | 1.000 | 2452 | 1.000 |
| | | | 396 | 1.011 | 259 | 1.014 |
| | 80 | 20 | 3320 | 1.024 | 14704 | 1.024 |
| | | | 6710 | 1.018 | 8439 | 1.010 |
| | | | 2475 | 1.000 | 3761 | 1.000 |
| | | | 1982 | 1.024 | 3812 | 1.016 |
| | | 40 | 1964 | 1.009 | 2901 | 1.000 |
| | | | 519 | 1.000 | 744 | 1.000 |
| | | | 934 | 1.021 | 6229 | 1.000 |
| | | | 452 | 1.006 | 2676 | 1.000 |
| | | 60 | 836 | 1.024 | 582 | 1.005 |
| | | | 367 | 1.015 | 624 | 1.000 |
| | | | 1193 | 1.004 | 4056 | 1.000 |
| | | | 758 | 1.029 | 3875 | 1.020 |

Table XXII. A COMPARISON OF THE TAILORED COOLING ALGORITHM AND THE BRANCH AND BOUND METHOD APPLIED TO THE DIRECTED STEINER TREE PROBLEM ON GRAPHS OF .25 DENSITY

| | | | TCA | | Branch and Bound | |
|---|---|---|---|---|---|---|
| $p_e$ | n | special | Run Time (sec.) | $\frac{\text{Solution}}{\text{Optimum}}$ | Run Time (sec.) | $\frac{\text{Solution}}{\text{Optimum}}$ |
| .25 | 40 | 10 | 374 | 1.027 | 405 | 1.013 |
| | | | 508 | 1.006 | 277 | 1.000 |
| | | | 945 | 1.000 | 4629 | 1.000 |
| | | | 253 | 1.000 | 829 | 1.000 |
| | | 20 | 31 | 1.000 | 85 | 1.000 |
| | | | 398 | 1.012 | 790 | 1.000 |
| | | | 1156 | 1.000 | 2162 | 1.000 |
| | | | 468 | 1.010 | 379 | 1.003 |
| | | 30 | 74 | 1.000 | 152 | 1.000 |
| | | | 6 | 1.000 | 194 | 1.000 |
| | | | 118 | 1.018 | 454 | 1.008 |
| | | | 165 | 1.014 | 359 | 1.011 |
| | 60 | 15 | 298 | 1.000 | 847 | 1.000 |
| | | | 854 | 1.000 | 1159 | 1.000 |
| | | | 393 | 1.025 | 728 | 1.017 |
| | | | 617 | 1.018 | 935 | 1.004 |
| | | 30 | 531 | 1.019 | 506 | 1.007 |
| | | | 1937 | 1.018 | 1495 | 1.016 |
| | | | 1371 | 1.025 | 8214 | 1.019 |
| | | | 680 | 1.000 | 743 | 1.000 |
| | | 45 | 248 | 1.000 | 189 | 1.000 |
| | | | 176 | 1.021 | 320 | 1.011 |
| | | | 53 | 1.000 | 656 | 1.000 |
| | | | 476 | 1.000 | 462 | 1.000 |
| | 80 | 20 | 4661 | 1.021 | 10988 | 1.015 |
| | | | 2742 | 1.000 | 2511 | 1.000 |
| | | | 1840 | 1.019 | 2376 | 1.008 |
| | | | 3596 | 1.014 | 5282 | 1.013 |
| | | 40 | 18 | 1.000 | 9031 | 1.000 |
| | | | 245 | 1.020 | 1893 | 1.011 |
| | | | 2850 | 1.028 | 4798 | 1.017 |
| | | | 4136 | 1.018 | 8205 | 1.000 |
| | | 60 | 126 | 1.000 | 744 | 1.000 |
| | | | 537 | 1.026 | 2141 | 1.013 |
| | | | 611 | 1.000 | 1483 | 1.000 |
| | | | 1337 | 1.032 | 2915 | 1.025 |

Table XXIII. A COMPARISON OF THE TAILORED COOLING ALGORITHM AND THE BRANCH AND BOUND METHOD APPLIED TO THE DIRECTED STEINER TREE PROBLEM ON GRAPHS OF .50 DENSITY

| | | | TCA | | Branch and Bound | |
|---|---|---|---|---|---|---|
| $p_e$ | n | special | Run Time (sec.) | $\frac{\text{Solution}}{\text{Optimum}}$ | Run Time (sec.) | $\frac{\text{Solution}}{\text{Optimum}}$ |
| .50 | 40 | 10 | 426 | 1.000 | 338 | 1.000 |
| | | | 235 | 1.000 | 459 | 1.000 |
| | | | 4 | 1.016 | 372 | 1.013 |
| | | | 201 | 1.024 | 596 | 1.019 |
| | | 20 | 13 | 1.008 | 179 | 1.000 |
| | | | 9 | 1.000 | 204 | 1.000 |
| | | | 28 | 1.000 | 267 | 1.000 |
| | | | 104 | 1.000 | 83 | 1.000 |
| | | 30 | 4 | 1.000 | 12 | 1.000 |
| | | | 15 | 1.000 | 6 | 1.000 |
| | | | 4 | 1.000 | 649 | 1.000 |
| | | | 27 | 1.000 | 10 | 1.000 |
| | 60 | 15 | 754 | 1.000 | 2366 | 1.000 |
| | | | 838 | 1.000 | 651 | 1.000 |
| | | | 455 | 1.023 | 212 | 1.018 |
| | | | 68 | 1.000 | 152 | 1.000 |
| | | 30 | 503 | 1.000 | 828 | 1.000 |
| | | | 342 | 1.024 | 575 | 1.021 |
| | | | 497 | 1.031 | 619 | 1.014 |
| | | | 1185 | 1.000 | 2451 | 1.000 |
| | | 45 | 332 | 1.018 | 9107 | 1.015 |
| | | | 170 | 1.012 | 2529 | 1.010 |
| | | | 44 | 1.014 | 1172 | 1.006 |
| | | | 221 | 1.000 | 157 | 1.000 |
| | 80 | 20 | 768 | 1.000 | 1690 | 1.000 |
| | | | 2484 | 1.000 | 3673 | 1.000 |
| | | | 1173 | 1.038 | 1026 | 1.024 |
| | | | 188 | 1.029 | 949 | 1.029 |
| | | 40 | 21 | 1.000 | 165 | 1.000 |
| | | | 492 | 1.013 | 1048 | 1.013 |
| | | | 311 | 1.000 | 862 | 1.000 |
| | | | 84 | 1.011 | 425 | 1.005 |
| | | 60 | 147 | 1.000 | 834 | 1.000 |
| | | | 476 | 1.000 | 362 | 1.000 |
| | | | 13 | 1.000 | 19 | 1.000 |
| | | | 132 | 1.000 | 592 | 1.000 |

Table XXIV. A COMPARISON OF THE TAILORED COOLING ALGORITHM AND THE BRANCH AND BOUND METHOD APPLIED TO THE DIRECTED STEINER TREE PROBLEM ON GRAPHS OF .90 DENSITY

| | | | TCA | | Branch and Bound | |
|---|---|---|---|---|---|---|
| $p_e$ | n | special | Run Time (sec.) | $\frac{\text{Solution}}{\text{Optimum}}$ | Run Time (sec.) | $\frac{\text{Solution}}{\text{Optimum}}$ |
| .90 | 40 | 10 | 373 | 1.000 | 294 | 1.000 |
| | | | 614 | 1.036 | 1188 | 1.000 |
| | | | 116 | 1.000 | 523 | 1.004 |
| | | | 368 | 1.000 | 190 | 1.000 |
| | | 20 | 8 | 1.000 | 204 | 1.000 |
| | | | 9 | 1.000 | 19 | 1.000 |
| | | | 325 | 1.033 | 518 | 1.024 |
| | | | 95 | 1.021 | 894 | 1.000 |
| | | 30 | 3 | 1.017 | 145 | 1.012 |
| | | | 38 | 1.000 | 68 | 1.000 |
| | | | 61 | 1.015 | 172 | 1.011 |
| | | | 36 | 1.002 | 156 | 1.000 |
| | 60 | 15 | 174 | 1.026 | 246 | 1.000 |
| | | | 402 | 1.006 | 371 | 1.000 |
| | | | 689 | 1.015 | 565 | 1.009 |
| | | | 230 | 1.000 | 1821 | 1.000 |
| | | 30 | 109 | 1.017 | 217 | 1.000 |
| | | | 4 | 1.000 | 305 | 1.000 |
| | | | 431 | 1.011 | 1449 | 1.001 |
| | | | 106 | 1.009 | 2410 | 1.004 |
| | | 45 | 92 | 1.000 | 862 | 1.000 |
| | | | 178 | 1.014 | 256 | 1.000 |
| | | | 493 | 1.000 | 395 | 1.000 |
| | | | 44 | 1.000 | 221 | 1.000 |
| | 80 | 20 | 6418 | 1.043 | 7574 | 1.000 |
| | | | 4975 | 1.052 | 10683 | 1.009 |
| | | | 847 | 1.010 | 3028 | 1.000 |
| | | | 5403 | 1.014 | 4892 | 1.002 |
| | | 40 | 1394 | 1.006 | 2787 | 1.000 |
| | | | 112 | 1.000 | 86 | 1.000 |
| | | | 9 | 1.015 | 579 | 1.000 |
| | | | 334 | 1.000 | 1618 | 1.000 |
| | | 60 | 148 | 1.000 | 314 | 1.000 |
| | | | 1264 | 1.027 | 5591 | 1.014 |
| | | | 844 | 1.000 | 1093 | 1.000 |
| | | | 2329 | 1.015 | 3588 | 1.000 |

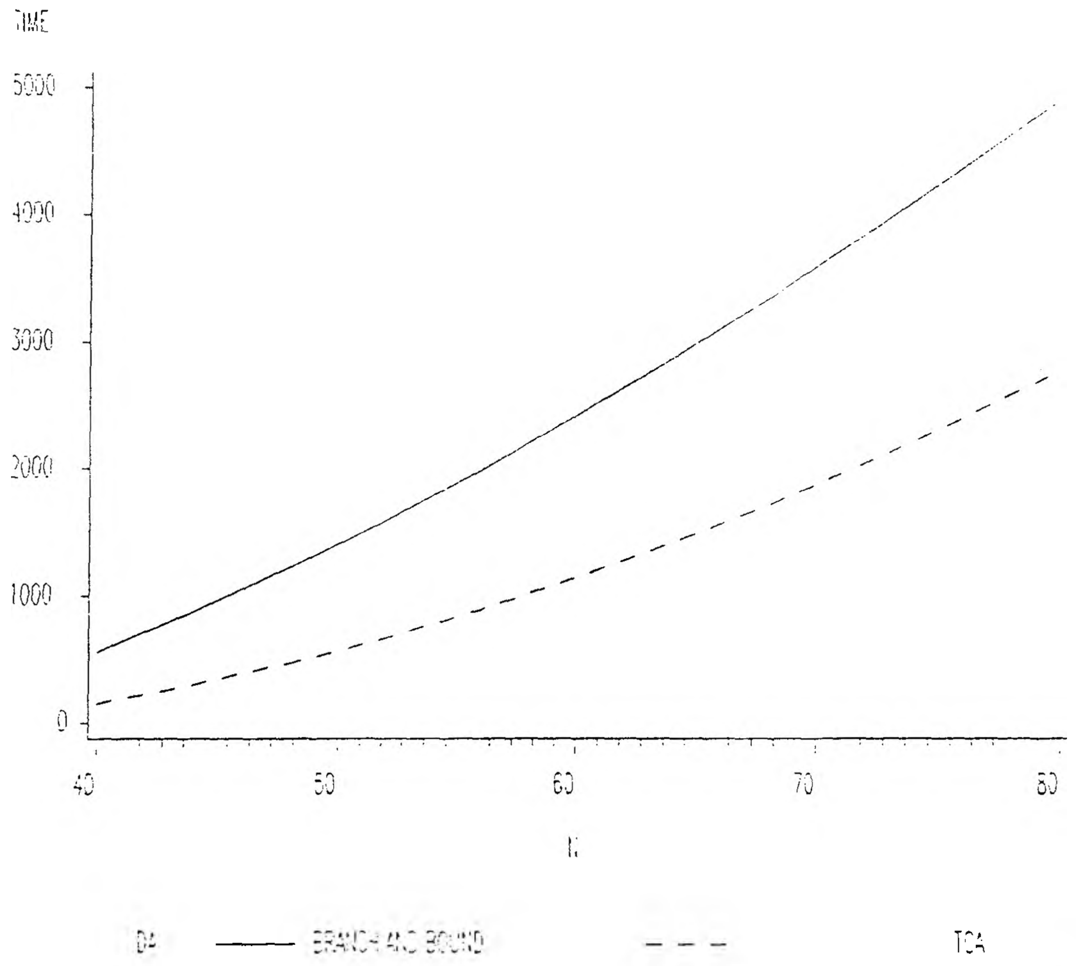

Figure 30. Regression Functions for Branch and Bound and the TCA Special = $\frac{n}{4}$ and $p_e$ = 0.05.

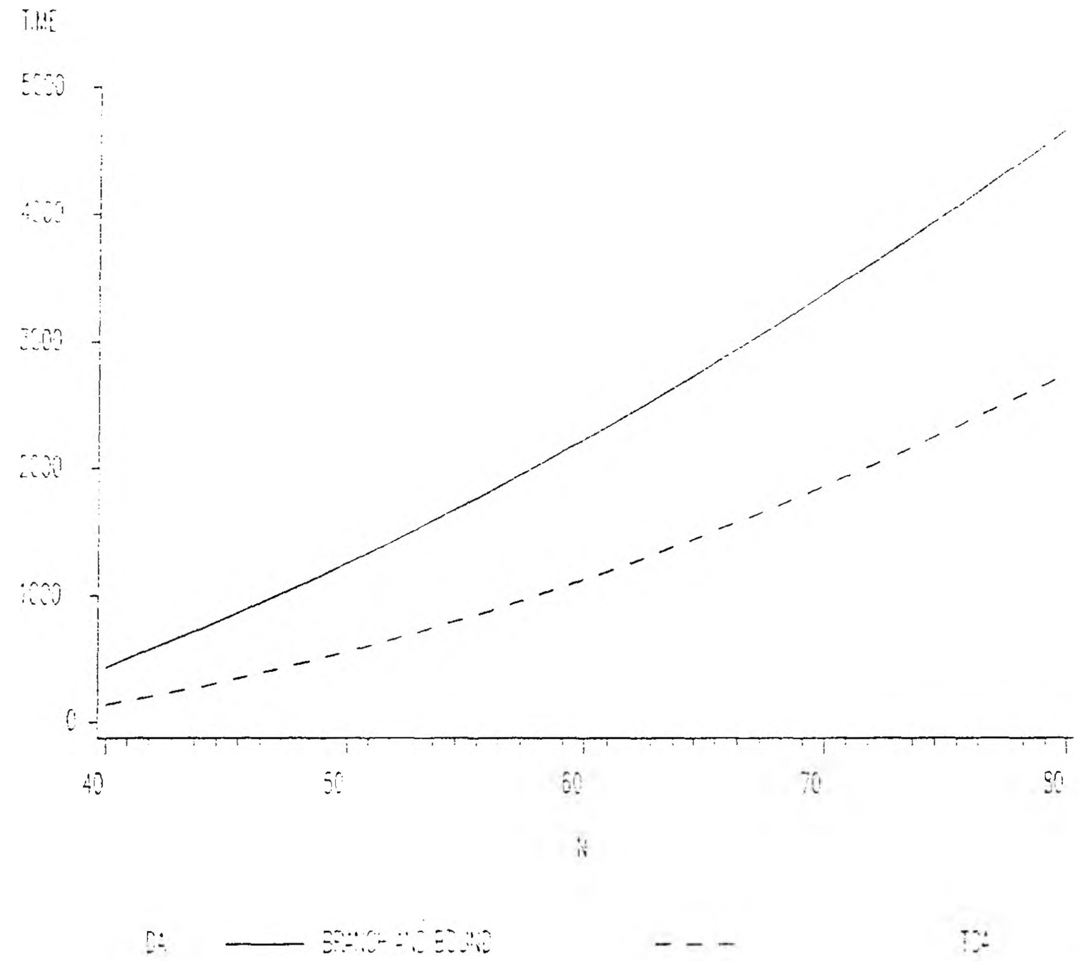

Figure 31. Regression Functions for Branch and Bound and the TCA Special = $\frac{n}{4}$ and $p_e$ = 0.25.

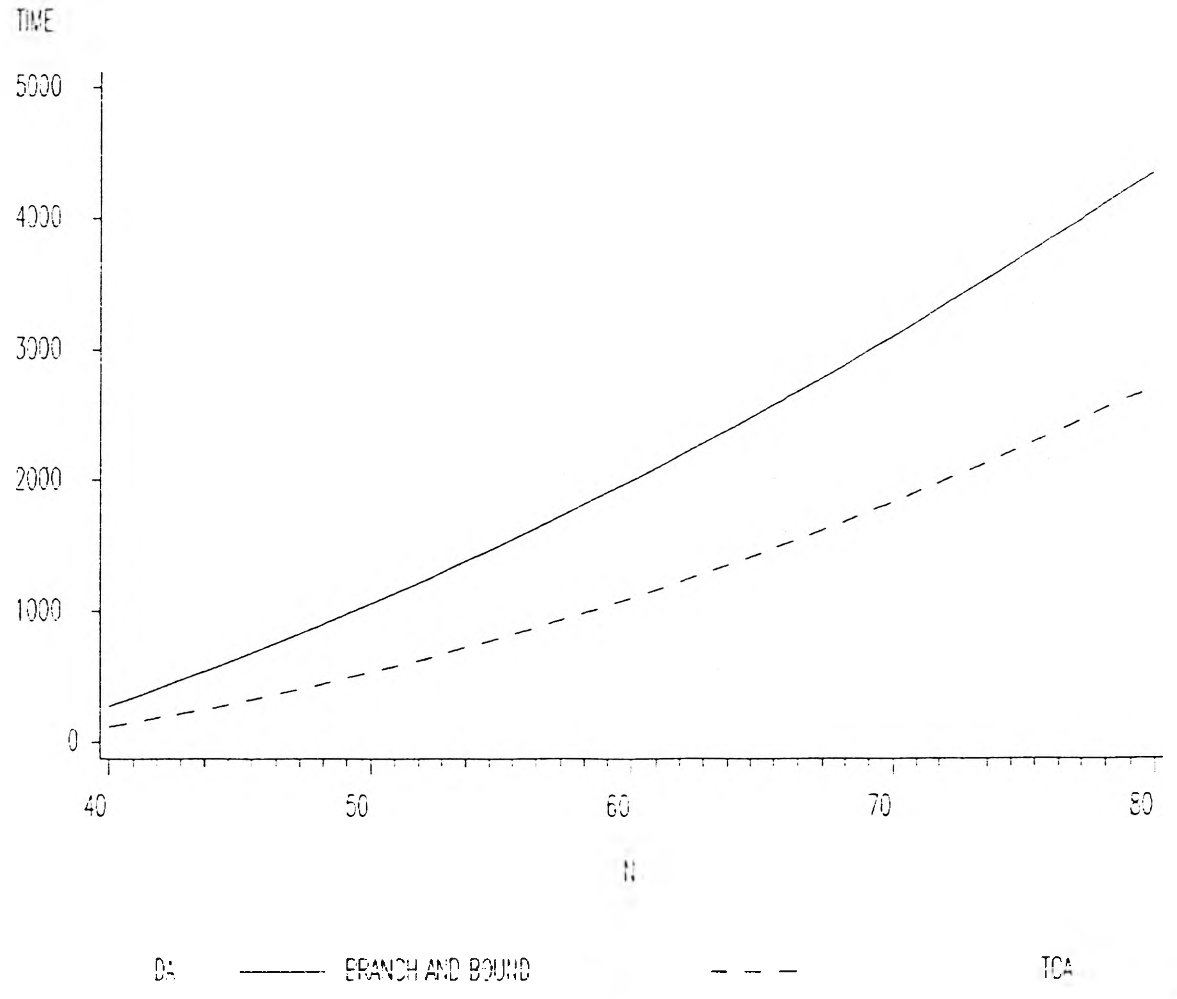

Figure 32. Regression Functions for Branch and Bound and the TCA Special $= \frac{n}{4}$ and $p_e = 0.50$.

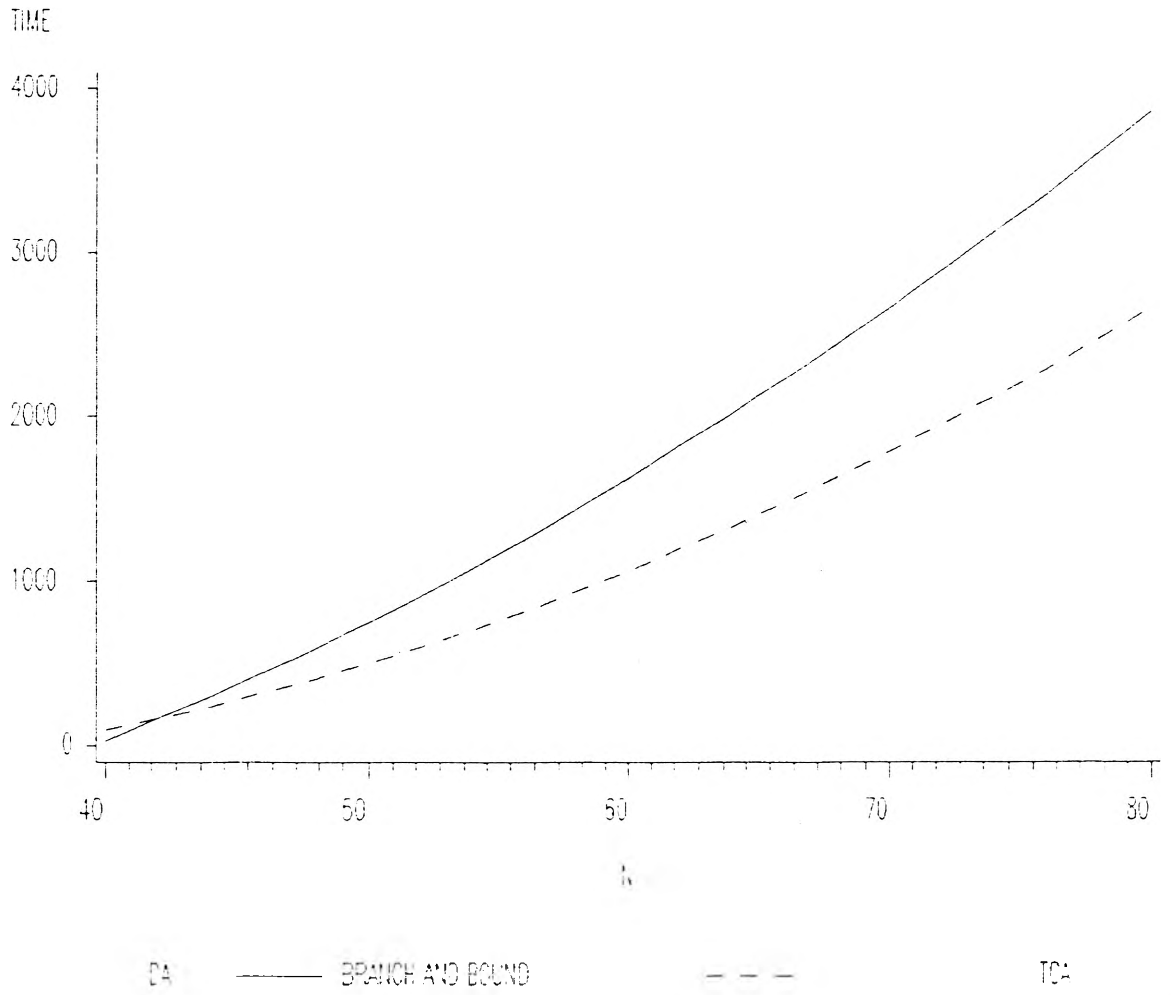

Figure 33. Regression Functions for Branch and Bound and the TCA Special = $\frac{n}{4}$ and $p_e$ = 0.90.

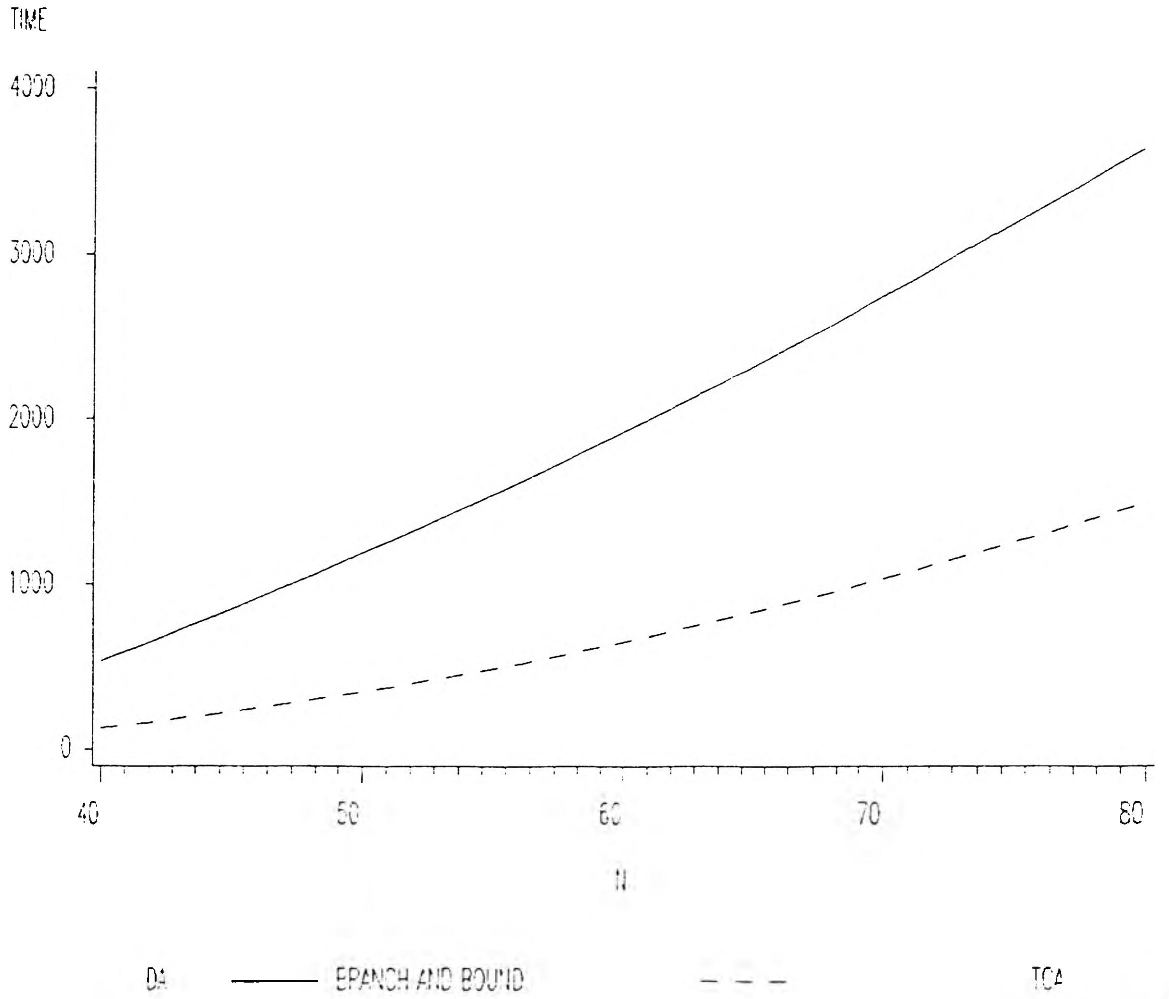

Figure 34. Regression Functions for Branch and Bound and the TCA Special $= \frac{n}{2}$ and $p_e = 0.05$.

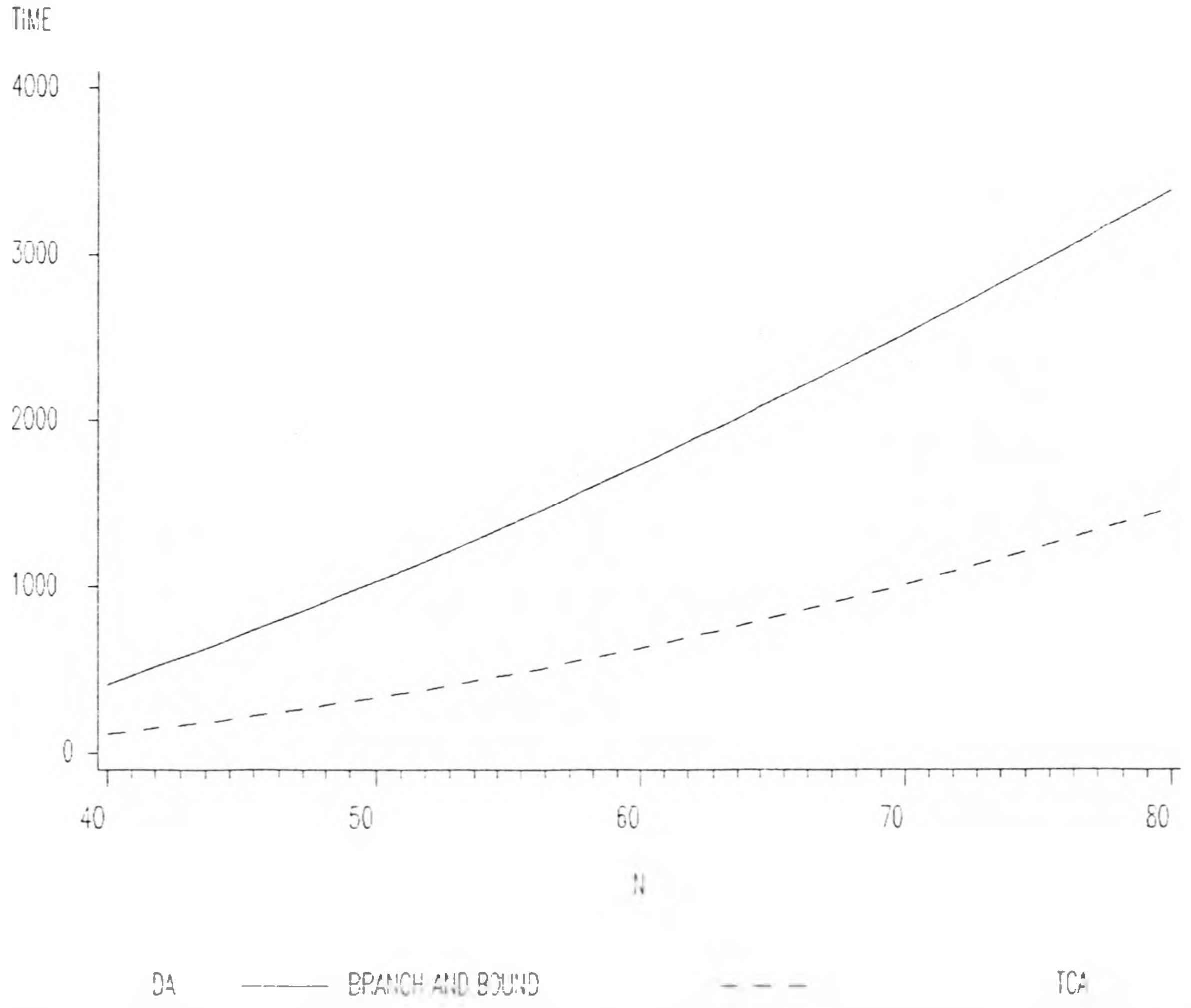

Figure 35. Regression Functions for Branch and Bound and the TCA Special = $\frac{n}{2}$ and $p_e$ = 0.25.

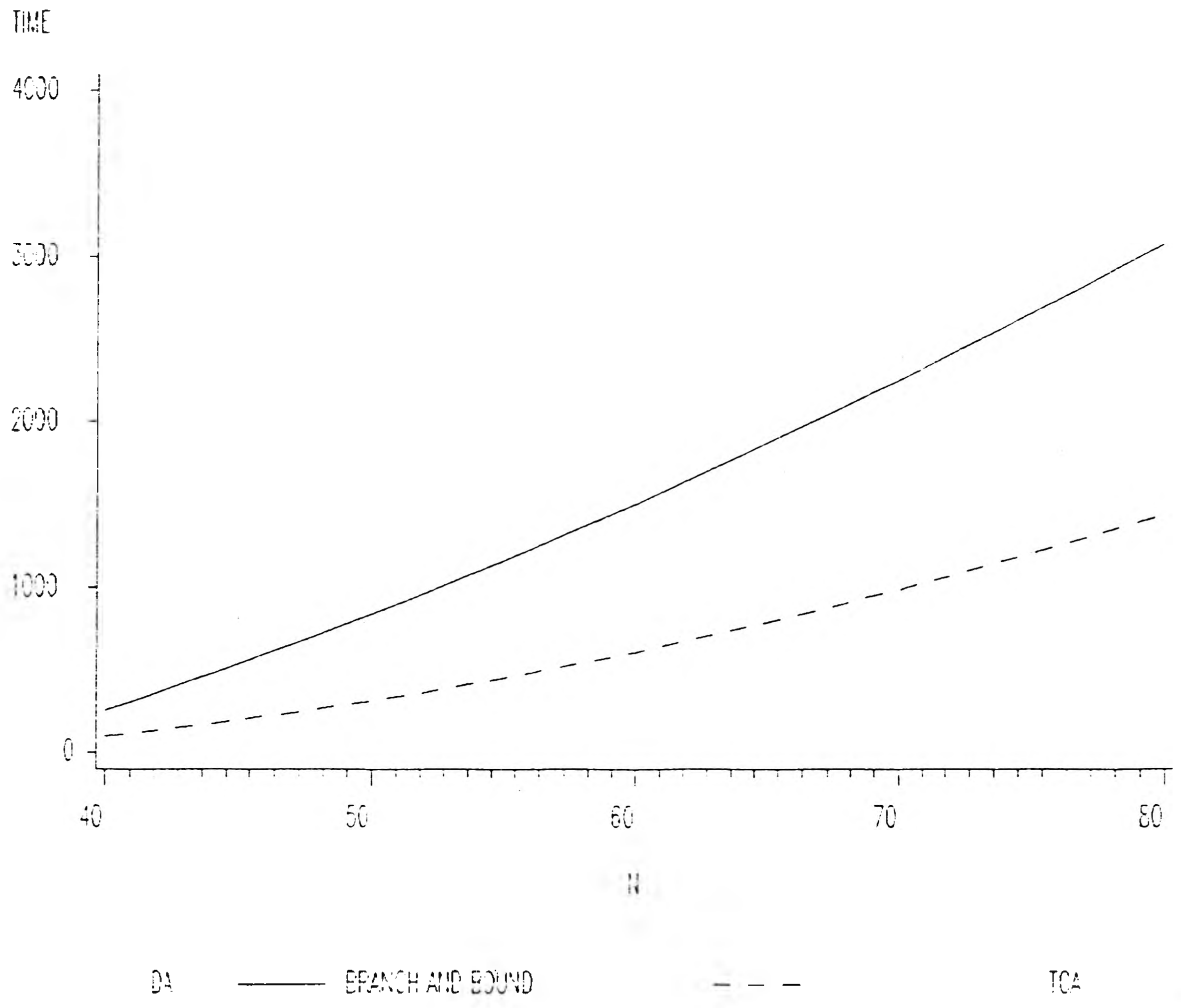

Figure 36. Regression Functions for Branch and Bound and the TCA Special = $\frac{n}{2}$ and $p_e$ = 0.50.

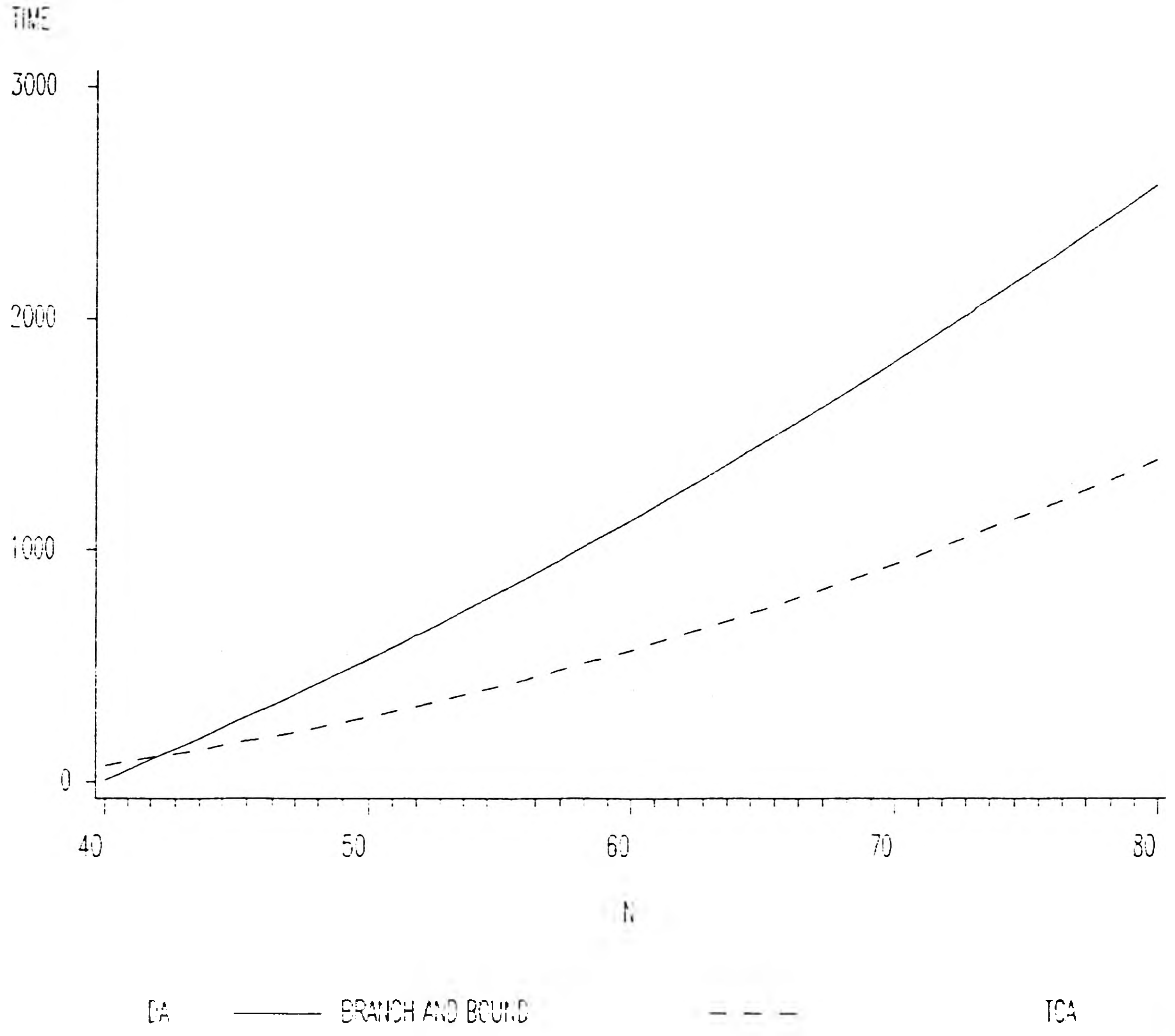

Figure 37. Regression Functions for Branch and Bound and the TCA Special = $\frac{n}{2}$ and $p_e$ = 0.90.

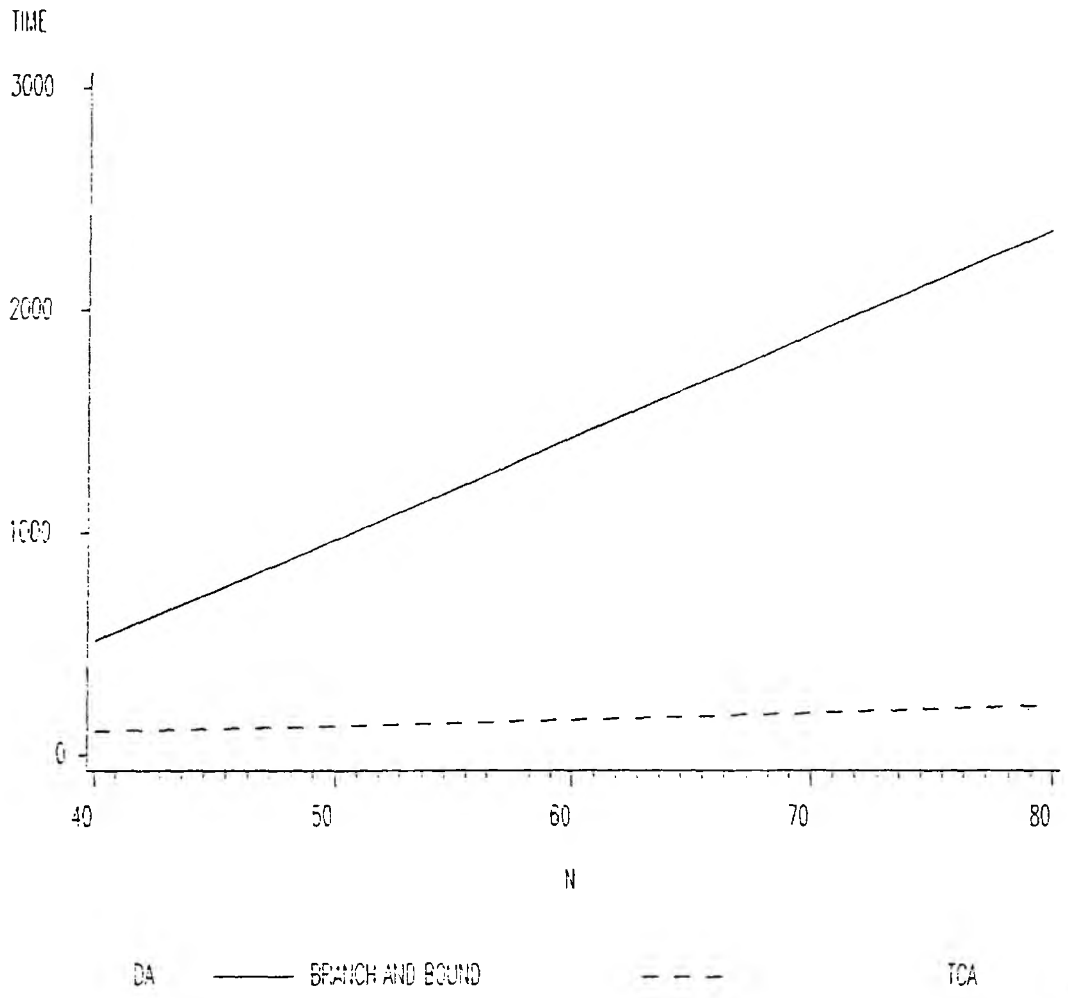

Figure 38. Regression Functions for Branch and Bound and the TCA Special = $\frac{3n}{4}$ and $p_e$ = 0.05.

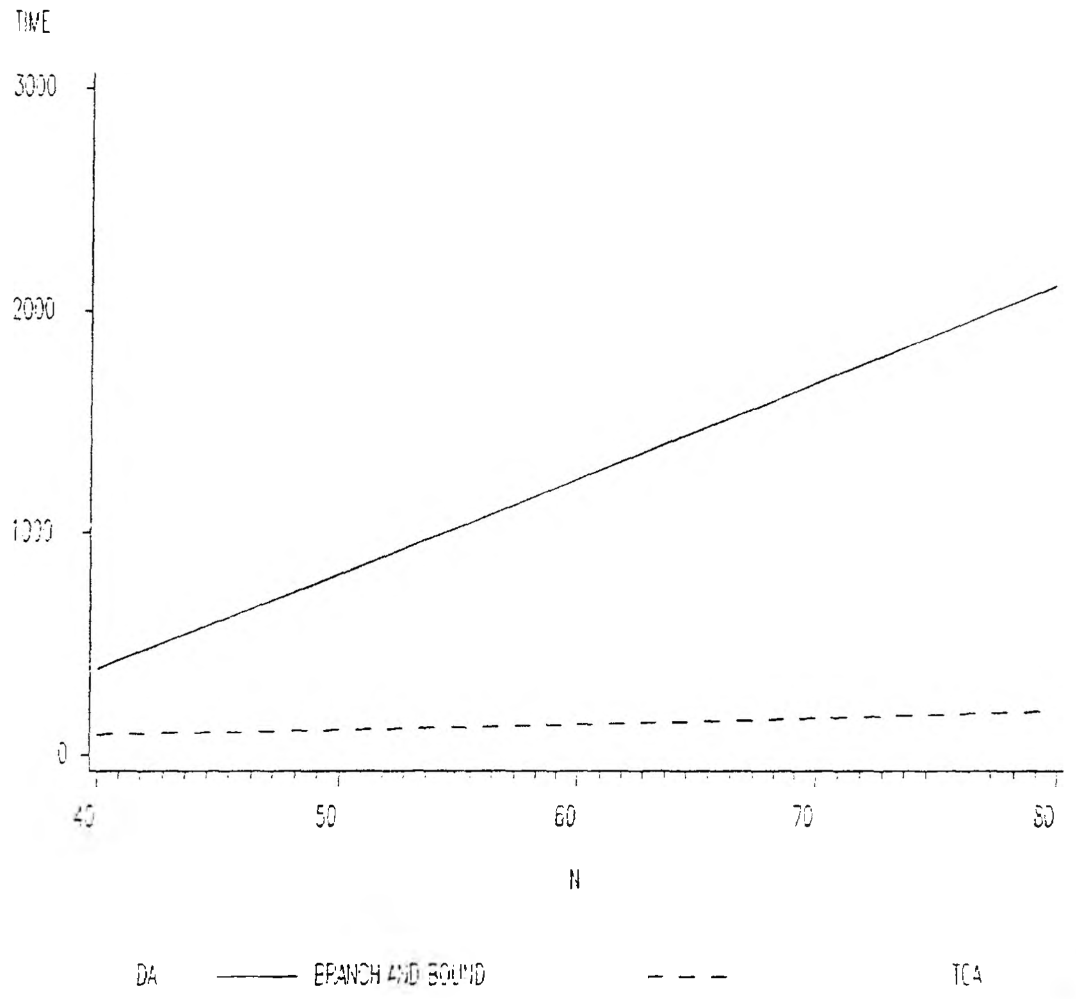

Figure 39. Regression Functions for Branch and Bound and the TCA Special = $\frac{3n}{4}$ and $p_e$ = 0.25

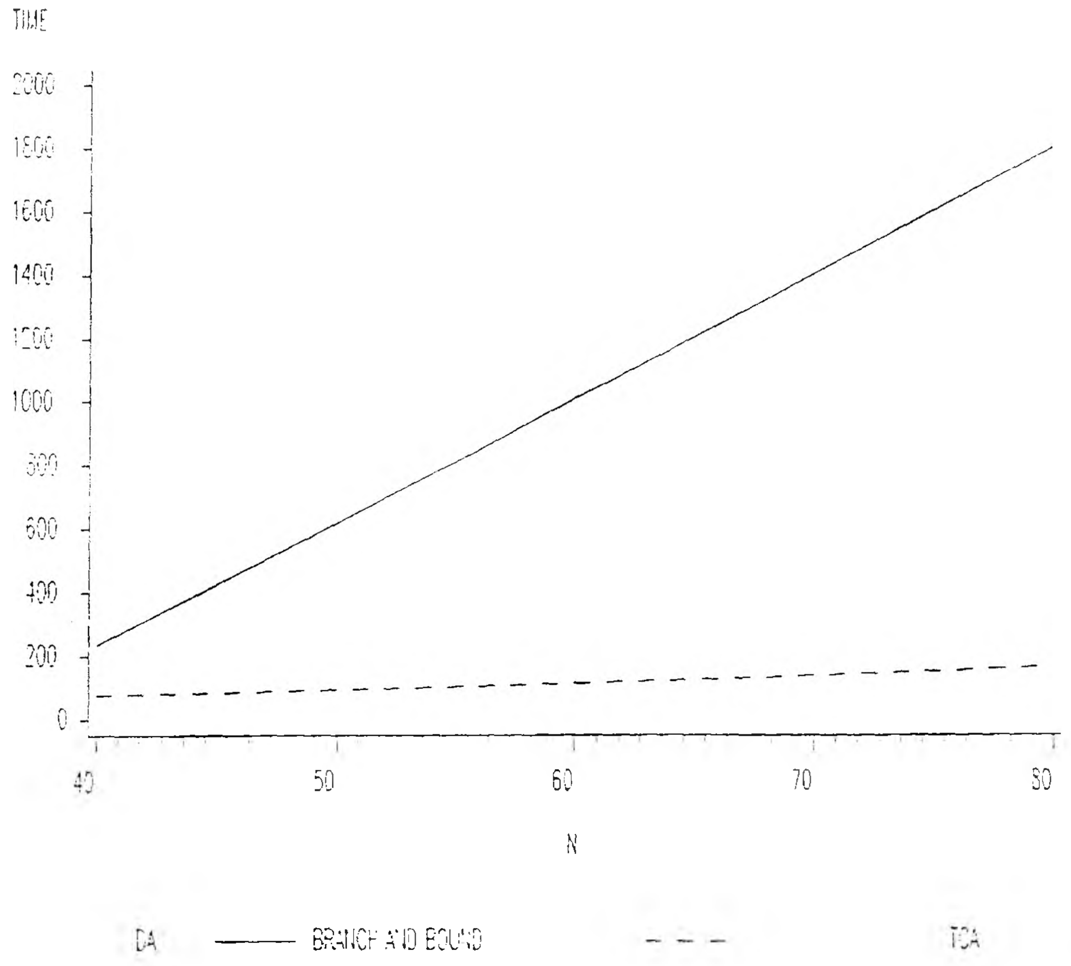

Figure 40. Regression Functions for Branch and Bound and the TCA Special = $\frac{3n}{4}$ and $p_e$ = 0.50.

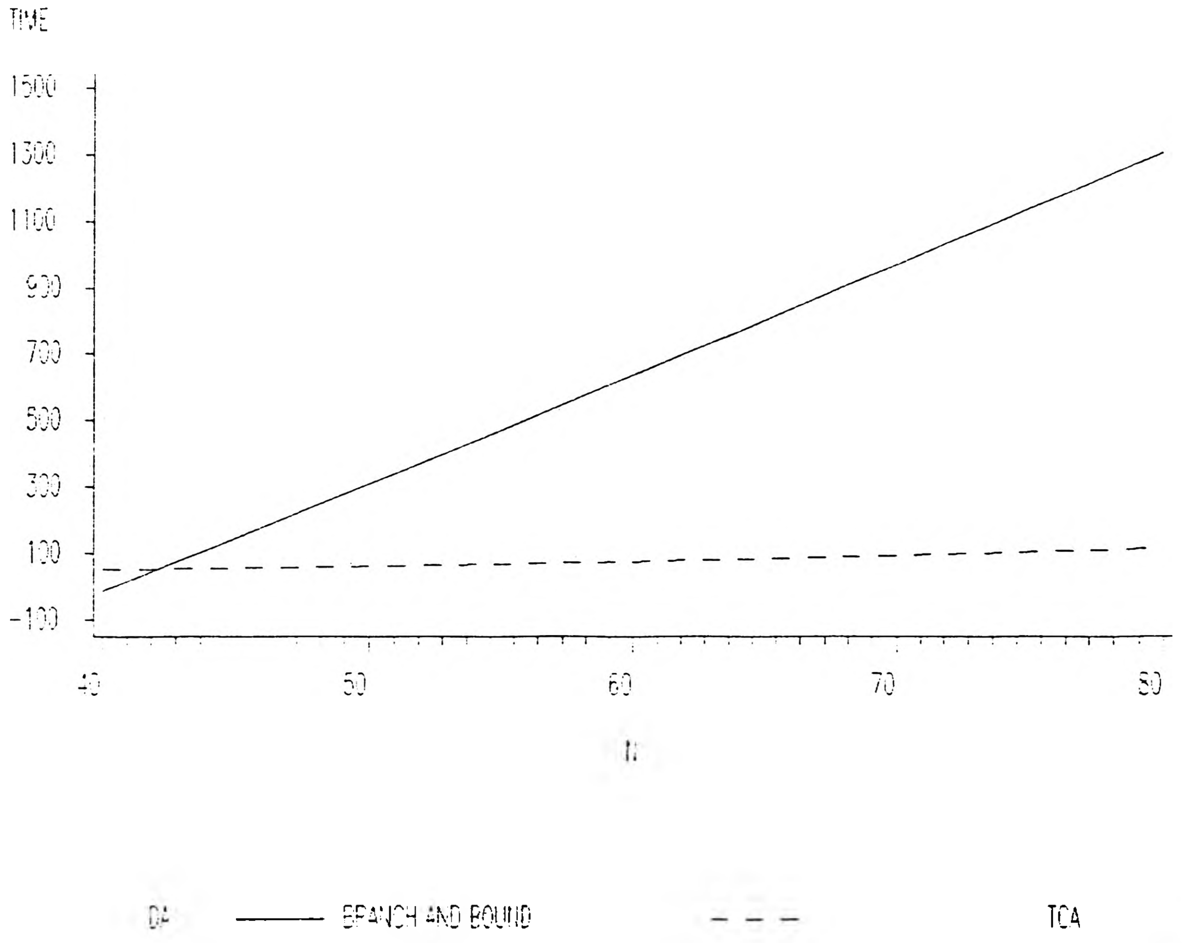

Figure 41. Regression Functions for Branch and Bound and the TCA Special = $\frac{3n}{4}$ and $p_e$ = 0.90

# VIII. CONCLUSIONS AND FUTURE RESEARCH

One immediate conclusion from the results of the experiments undertaken in this study is that simulated annealing is an effective technique for finding a near-optimum solution to the Directed Steiner Problem on graphs. The information displayed in Tables (XIV-XV) show that not only are the average best answers discovered by either the tailored or the dynamic versions of the cooling algorithms close to the optimum, but also that the level of accuracy is independent of the size and density of the graph. Moreover, neither of the simulated annealing cooling schedules produced an answer more than 6.8% above the optimum for any graph. The experiments conducted in Chapter V show that the tailored algorithm outperforms local optimization in the accuracy of the answers obtained, even when the longer running time for annealing is taken into account. In Chapter VII, the branch and bound technique provided exact answers eventually for all the problems, but the tailored version of annealing algorithm usually gave near-optimal answers more quickly.

One reason for the high quality of the results yielded by annealing for the DSP on graphs may be that the barrier between two configurations with near-optimum lengths is highly degenerate. That is, there is often a big change in the solution whenever one Steiner point is added to or removed from the set of points that must be joined. Hence, there are many paths which a sequence of solutions can take which lead from one basin to another in the configuration landscape. While each of these paths is unlikely to occur often, it is probable that the process will frequently encounter a near-optimum local minimum. Finding a measure for the degeneracy of a DSP and relating that measure to the effectiveness of simulated annealing is a challenging open problem. It may be possible to find a general measure of degeneracy that could be applied to other combinatorial optimization problems as well. The result of such research might be the discovery of ways in which annealing could prune the configuration space of shallow valleys that consume large quantities of running time. A starting point for this research might be the work in physics that has been done in configuration space landscapes with respect to spin glasses [63].

A drawback to the use of simulated annealing is the amount of running time that the method requires. In the case of the DSP, the heuristic algorithm of Richard T. Wong [112] provides an initial solution that, for most of the random graphs within the range of the parameters used in this study, is more accurate than that yielded by simulated annealing over the same amount of time. This is not surprising because simulated annealing is a general method applicable to numerous diverse problems, while the dual ascent algorithm of Wong is designed to take advantage of the special characteristics of the DSP. However, the accuracy of the heuristic used by Wong to obtain feasible solutions seems to decrease as the size of the problem increases. The traditional way to obtain a high quality answer for large combinatorial optimization problems including the DSP on graphs was to apply an algorithm like that of Wong in a branch and bound scheme. The experiments performed in Chapter VII on graphs with 40 to 80 vertices indicate that simulated annealing, using a minimum spanning tree algorithm to evaluate the length of a configuration, and starting from the configuration yielded by the heuristic algorithm of R. T. Wong, may find a high quality answer more quickly on the average than does the branch and bound scheme with the dual ascent algorithm for developing lower bounds. Tables (XXI-XXIV) and the regression curves in Figures (30-41) suggest that annealing is superior to branch and bound in finding a quick near-optimum solution for random graphs. Since the effectiveness of branch and bound always depends on the availability of tight lower bounds and the accuracy of the algorithm of Wong declines as N becomes larger, further research may show that the trends exhibited in this work hold over a larger range of random graphs.

Simulated annealing can only promise a global minimum as the number of iterations approaches infinity. The asymptotic convergence property of annealing is based on the properties of stationary probability distributions of Markov chains. An area for further research is the exploration of methods for determining bounds on the number of iterations required for attaining a certain confidence level in the global optimality of a solution. These new methods may lead to new ways of choosing the 4 parameters that distinguish different cooling schedules or to the use of distributions other than the Boltzmann. The directed Steiner Problem on graphs is a good problem on which to test new variations of simulated annealing because of its diverse applications, its responsiveness to annealing, and the existing algorithms and computational results with which to compare.

It is not clear whether a tailored algorithm with a simple way of determining annealing parameters or a more robust dynamic algorithm with a more complicated means of choosing parameters is better. Tables (XVI-XVII) indicate that the total running time for the DCA is on the average less than the total running time for the TCA. Thus, the experiments conducted in this study show that dynamic methods of determining the amount to decrement the control parameter between chains and of calculating the terminating condition are effective. The two annealing schemes do not differ a great deal in the quality of final solutions, although Tables (XIV-XV) suggest that the TCA is a little more reliable. The choice between the two versions becomes more problematic after an examination of the experiments done in Chapter V on the first occurrences of near-optimal solutions. Figures (17-25) suggest that the first occurrences of near-optimal solutions follow an exponential distribution. In these graphs the means of the distributions for the TCA are always smaller than the corresponding means of the distributions for the DCA. A regression analysis applied to these results showed a significant difference in the behavior of the two annealing algorithms. There seems to be a tradeoff in using one or the other method. The DCA seems to be faster, but the TCA gives slightly more accurate answers. The method of Chapter V used to tailor the simulated annealing procedure to the Directed Steiner Problem on graphs, moreover, is very time consuming. Hence, it appears that there is currently no answer to the question of which method is superior. It would be premature to suggest that a simple cooling schedule tailored to a given problem cannot be improved by a more elaborate means of choosing the parameters. Much more research needs to be accomplished in comparing cooling techniques before it is possible to formulate a schedule that is optimal for all types of problems.

With respect to the Directed Steiner Problem on graphs itself, the annealing method should be applied to much larger problems consisting of thousands of vertices. To date, problems of this magnitude have only been attempted once, and, in that case, it was for undirected graphs [9]. A Cray X-MP/48 programmed with a branch and bound algorithm employing Lagrangian relaxation to achieve lower bounds was applied to problems as large as 2500 nodes and 62,500 edges. In some of the instances, no solution was found even after 21,600 seconds of CPU time on the Cray. Most published results are for graphs with less than 100 vertices. One of the problems with algorithms such as that of Wong or Lagrangian relaxation is that massive amounts of memory are needed to keep track of lists required by the algorithms. This becomes an even bigger problem when these algorithms are incorporated into branch and bound schemes where backtracking often is necessary. Thus, most of the applications of traditional methods have concentrated on extremely sparse networks with densities less than 0.05. Annealing, however, does not demand much memory since there is no need for backtracking and the cost algorithm is usually a simple one. In the work done in this study, the minimum spanning tree algorithm for directed trees served as the means of calculating the cost for each configuration.

Another rather obvious approach to large-scale DSP problems is a parallel implementation. Massive parallelism could be used with annealing with each processor given the task of evaluating a local change in a global configuration. As mentioned in Chapter IV, work toward a parallel annealing algorithm has already begun. New parallel algorithms using either conventional methods or annealing might be suitable for the DSP. In addition, there may be important classes of graphs for which a parallel implementation is a natural one for approaching the DSP. For these classes very fast algorithms might be developed.

A problem for further empirical or theoretical study concerning the Directed Steiner Problem on graphs is the relationship between the quality of the final solution found by annealing and the size of the parameters. Such a study might discover a mathematical expression for the probability that the error in the final solution is less than any given value. Such work has been done for a special class of Travelling Salesman Problems [3].

One final remark concerning the results of the experiments should be made. The graphs generated in the experiments were entirely random. They did not belong to any special class of graphs. Thus, the conclusions that have been drawn from these experiments concerning the superior effectiveness of one method or another may not be relevant to a Directed Steiner Problem on graphs that originates from a practical application.

## REFERENCES

1. Aarts, E. H. L., Bont, F. M. J. de, Habers, E. H. A., and Van Laarhoven, P. J. M., (1986). Parallel implementations of the Statistical Cooling Algorithm, Integration, 4, pp. 209-238.

2. Aarts, E. H. L., Korst, J. H. M., (1986). Simulation of Learning in Parallel Networks Based on the Boltzmann Machine, Proceedings of the Second European Simulation Congress, Antwerp, Belgium, pp. 391-398.

3. Aarts, E. H. L., Korst, J. H. M., and Van Laarhoven, P. J. M., (1988). A Quantitative Analysis of the Simulated Annealing Algorithm: A Case Study for the Traveling Salesman Problem, Journal of Statistical Physics, 50, pp. 189-206.

4. Aarts, E. H. L. and Van Laarhoven, P. J. M., (1985). A General Approach to Combinatorial Optimization Problems, Philips Journal of Research, 40. pp. 193-226.

5. Aho, A. V., Garey, M. R., and Hwang, F. K., (1977). Rectilinear Steiner Trees: Efficient Special Case Algorithms, Networks, 7, pp. 37-58

6. Aneja, Y. P., (1980). An Integer Linear Programming Approach to the Steiner Problem in Graphs, Networks, 10, pp. 167-178.

7. Balakrishnan, A. and Patel, N. R., (1987). Problem Reduction Methods and a Tree Generation Algorithm for the Steiner Network Problem, Networks, 17, pp. 65-85.

8. Beasley, J. E., (1984). An Algorithm for the Steiner Problem in Graphs, Networks, 14, pp. 147-159.

9. Beasley, J. E., (1989). An SST-Based Algorithm for the Steiner Problem in Graphs, Networks, 19, pp. 1-16.

10. Bellmore, M. and Ratliff, H. D., (1971). Set-Covering and Involutory Bases, Management Science, 18, pp. 194-206.

11. Bonomi, E. and Lutton, J.-L., (1984). The N-city Travelling Salesman Problem: Statistical Mechanics and Metropolis Algorithm, SIAM Review, 26, pp. 209-217.

12. Bonomi, E. and Lutton, J.-L., (1986). The Asymptotic Behavior of Quadratic Sum Assignment Problems: A Statistical Mechanics Approach, European Journal of Operational Research, 26, pp. 295-300.

13. Boyce, W. M., (1977). An Improved Program for the Full Steiner Tree Problem, ACM Transactions of Mathematical Software, 3, pp. 359-385.

14. Brouwer, R. J. and Banerjee, P., (1988). A Parallel Simulated Annealing Algorithm for Channel Routing on a Hypercube Multiprocessor, Proceedings IEEE International Conference on Computer-Aided Design. pp. 4-7.

15. Burkard, R. E. and Rendl, F., (1984). A Thermodynamically Motivated Simulation Procedure for Combinatorial Optimization Problems, European Journal of Operations Research , 17, pp. 169-174.

16. Černy , V., (1985). Thermodynamical Approach to the Traveling Salesman Problem: An Efficient Simulation Algorithm, Journal of Optimization Theory Applications, 45, pp. 41-51.

17. Chamberlain, Roger D., Edelman, Mark N., Franklin, Mark A., and Witte, Ellen E., (1988). Simulated Annealing on a Multiprocessor, Proceedings IEEE International Conference on Computer-Aided Design, pp. 540-544.

18. Chang, S. K., (1972). The Generation of Minimal Trees with a Steiner Topology, Journal of the ACM, 19, pp. 669-711.

19. Christofides, N., (1975). Graph Theory: An Algorithmic Approach, Academic Press, New York.

20. Christofides, N. and Whitlock, C. A., (1981). "Network Synthesis with Connectivity Constraints-A Survey", in Operational Research '81, J. P. Brans (ed.), North-Holland, Inc., New York.

21. Chu, Y. J. and Liu, T. H., (1965). On the Shortest Arborescences of a Directed Graph, Scientia Sinica, 14, pp. 1396-1400.

22. Cockayne, E. J. and Schiller, D. G., (1972). "Computation of Steiner Minimal Trees", in Combinatorics, D. J. A. Welsh and D. R. Woddal(eds.), Inst. Math. Appl. Southend-on-Sea, Essex. pp. 53-71.

23. Courant, R. and Robbins, H., (1941). What is Mathematics?, Oxford University Press, New York.

24. Coxeter, H. S. M., (1961). Introduction to Geometry, John Wiley and Sons, Inc., New York.

25. Crowder, H. and Padberg, M. W., (1980). Solving Large-Scale Symmetric Travelling Salesman Problems to Optimality, Management Science, 26, pp. 495-509.

26. Dijkstra, E. W., (1959). A Note on Two Problems in Connection With Graphs, Numerische Mathematik, 1, pp. 269-271.

27. Dreyfus, S. E. and Wagner, R. A., (1971). The Steiner Problem in Graphs, Networks, 1, pp. 195-207.

28. Duin, C. W. and Volgenant, A., (1987). Some Generalizations of the Steiner Problem in Graphs, Networks, 17, pp. 353-364.

29. Edmonds, J., (1967). Optimum Branchings, Journal of Research of the National Bureau of Standards-B. Mathematics and Mathematical Physics, 71B, pp. 233-240.

30. El Gamal, A., Hemachandra, L. A., Shperling, I., and Wei, V. K., (1987). Using Simulated Annealing to Design Good Codes, IEEE Transactions on Information Theory, IT-33, pp.116-123.

31. Fisher, M. L., (1981). The Lagrangian Relaxation Method for Solving Integer Programming Problems, Management Science 27, pp. 1-18.

32. Floyd, R. W., (1962). Algorithm 97: Shortest Path, Communications of the ACM, 5, pp. 345.

33. Foulds, L. R. and Graham, R. L., (1982). The Steiner Problem in Phylogeny is NP-Complete Advances in Applied Mathematics, 3,pp. 43-49.

34. Foulds, L. R. and Rayward-Smith, V. J., (1983). Steiner Problem in Graphs: Algorithms and Applications, Engineering Optimization, 7, p.7-16.

35. Frank, H. and Frisch, I. T., (1976). "Network Analysis", in Large Scale Networks: Theory and Design, IEEE Press, New York.

36. Fu, Yaotian and Anderson, P. W., (1986). Application of statistical mechanics to NP-complete problems in combinatorial optimisation, Journal of Physics A: Math. Gen., 19, pp. 1605-1620.

37. Gabow, H. N., (1977). Two Algorithms for generating weighted spanning trees in order, SIAM Journal of Computing 6, pp. 139-150.

38. Garey, M. R., Graham, R. L., and Johnson, D. S., (1977). The Complexity of Computing Steiner Minimal Trees SIAM Journal of Applied Mathematics, 32, pp. 835-859.

39. Garey, M. R. and Johnson, D. S., (1977). The Rectilinear Steiner Problem is NP-Complete, SIAM Journal of Applied Mathematics, 32, pp. 826-834.

40. Garey, M. R. and Johnson, D. S., (1979). Computers and Intractability - A Guide to the Theory of NP-Completeness, Freeman, San Francisco.

41. Gelfand, S. B. and Mitter, S. K., (1985). Analysis of Simulated Annealing for Optimization, Proceedings IEEE International Conference on Decision and Control, 24, pp. 779-786.

42. Geoffrion, A. M., (1974). Lagrangian Relaxation for Integer Programming, Mathematical Programming Study 2, North-Holland Publishing Company, New York.

43. Gidas, B., (1985). Optimization via the Langevin Equation, Proceedings IEEE International Conference on Decision and Control, 24, pp. 774-778.

44. Gilbert, E. N. and Pollak, H. O., (1968). Steiner Minimal Trees, SIAM Journal of Applied Mathematics, 16, pp. 1-29.

45. Grover, L. K., (1986). A New Simulated Annealing Algorithm for Standard Cell Placement, Proceedings IEEE International Conference on Computer-Aided Design, Santa Clara, pp. 378-380.

46. Hajek, B., (1985). A Tutorial Survey of Theory and Applications of Simulated Annealing, Proceedings IEEE International Conference on Decision and Control, 24, pp. 755-760.

47. Hajek, B., (1988). Cooling Schedules for Optimal Annealing, Mathematics of Operations Research, 13, 2, pp. 311-329.

48. Hakimi, S. L., (1971). Steiner Problem in Graphs and its Implications, Networks, 1, pp. 113-133.

49. Hanan, M., (1966). On Steiner's Problem with Rectilinear Distance, SIAM Journal of Applied Mathematics, 14, pp. 255-265.

50. Hanan, M., 1975. Layout, Interconnection, and Placement, Networks, 5, p.85-88.

51. Hanan, M. and Kurtzburg, J. M., (1972). "Placement Techniques", in Design Automation of Digital Systems, M. Breuer (ed.), Prentice-Hall, Englewood Cliffs, N. J.

52. Harary, F., (1969). Graph Theory, Addison-Wesley, Reading, Massachusetts.

53. Held, M., Wolfe, P., and Crowder, H., (1974). Validation of Subgradient Optimization Mathematical Programming 6, North-Holland Publishing Company, New York.

54. Hinton, G. E. and Sejnowski, T. J., (1986). "Learning and Relearning in Boltzmann Machines", in Parallel Distributed Processing, Explorations in the Microstructure of Cognition, McClelland, J. L., and Rumelhart, D. E. (eds.), Volume 1, Chapter VII. MIT Press, Cambridge, Massachusetts.

55. Hoel, P. G., Port, S. C., and Stone, C. J., (1972). Introduction to Stochastic Processes, Houghton Mifflin Company, Boston, Massachusetts.

56. Hopfield, J. J., (1985). Neural Computation of Decisions in Optimization Problems, Biological Cybernetics, 52, pp. 141-152.

57. Huang, M. D., Romeo, F. and Sangiovanni-Vincentelli, A. L., (1986). An Efficient General Cooling Schedule for Simulated Annealing, Proceedings IEEE International Conference on Computer-Aided Design, Santa Clara, pp. 381-384.

58. Hwang, F. K., (1978). The Rectilinear Steiner Problem, Journal of Design Automation and Fault-Tolerant Computing, 2, pp. 303-310.

59. Hwang, F. K., (1979). An O(n log n) Algorithm for Suboptimal Rectilinear Steiner Trees, IEEE Transactions on Circuits and Systems, CAS-26, pp. 75-77.

60. Johnson, D.S., C.R. Aragon, L.A. McGeoch and C. Schevon, (1987). Optimization by Simulated Annealing: an Experimental Evaluation, Parts I and II, AT & T Bell Laboratories preprint.

61. Karp, R. M., (1972). "Reducibility Among Combinatorial Problems", in Complexity of Computer Computations, Miller and Thatcher (eds.), Plenum Press, New York.

62. Kirkpatrick, S., Gelatt, C.D., and Vecchi, M. P., (1983). Optimization by Simulated Annealing, Science, 220, pp. 671-680.

63. Kirkpatrick, S., and Toulouse, G., (1985). Configuration Space Analysis of Travelling Salesman Problems, J. Physique, 46, pp. 1277-1292.

64. Kou, L., Markowsky, G., and Berman, L., (1981). A Fast Algorithm for Steiner Trees, Acta Informatica, 15, pp. 141-145.

65. Kruskal, J. B., (1956). On the Shortest Spanning Subtree of a Graph and the Traveling Salesman Problem, Proceedings of the American Mathematical Society, 7, pp. 48-50.

66. Lawler, E. L., (1976). Combinatorial Optimization: Networks and Matroids, Holt, Rinehart, and Winston, New York.

67. Lee, J. H., Bose, N. K., and Hwang, F. K., (1976). Use of Steiner's Problem in Suboptimal Routing in Rectilinear Metric, IEEE Transactions on Circuits and Systems, CAS-23, pp. 470-476.

68. Leong, H. W. and Liu, C. L., (1985). Permutation Channel Routing, Proceedings IEEE International Conference on Computer Design, Port Chester, pp. 579-584.

69. Levin, A. J., (1971). Algorithm for the Shortest Connection of a Group of Graph Vertices, Doklady Akademii Nauk Sssr, 12, pp. 1477-1481.

70. Lin, S., (1965). Computer Solutions of the Traveling Salesman Problem, Bell System Technical Journal, 44, pp. 2245-2269.

71. Lundy, M., (1985). Applications of the annealing algorithm to combinatorial problems in statistics , Biometrika, 72, pp. 191-198.

72. Lundy, M. and Mees, A., (1986). Convergence of an Annealing Algorithm, Mathematical Programming, 34, pp. 111-124.

73. Lyberatos, A., Wohlfarth, P., and Chantrell, R. W., (1985). Simulated Annealing: An Application in Fine Particle Magnetism, IEEE Transactions of Magnetics, MAG-21. pp. 1277-1282.

74. Magnanti, T. L., and Wong, R. T., (1984). Network Design and Transportation Planning: Models and Algorithms, Transportation Science, 18, pp. 1-55.

75. Melzak, Z. A., (1961). On the Problem of Steiner, Canadian Mathematical Bulletin, 4, pp. 143-148.

76. Metropolis, N. A., Rosenbluth, A., Rosenbluth, M., Teller, A., and Teller, E., (1953). Equation of State Calculations by Fast Computing Machines, Journal of Chemical Physics, 21, pp. 1087-1092.

77. Mitra, D., Romeo, F. and Sangiovanni-Vincentelli, A. L., (1985). Convergence and Finite-Time Behavior of Simulated Annealing, Proceedings IEEE International Conference on Decision and Control, 24, pp. 761-767.

78. Neter, J., Wasserman, W., and Kutner, M. H., (1983). Applied Linear Regression Models, Richard D. Irwin, Inc., Homewood, Illinois.

79. Otten, R. H. J. M. and Van Ginneken, L. P. P. P., (1984). Floorplan Design using Simulated Annealing, Proceedings IEEE International Conference on Computer-Aided Design, pp. 96-98.

80. Otten, R. H. J. M. and Van Ginneken, L. P. P. P., (1988). Stop Criteria in Simulated Annealing, Proceedings IEEE International Conference on Computer-Aided Design, pp. 549-552.

81. Papadimitriou, C. H., (1982). Combinatorial Optimization: Algorithms and Complexity, Prentice-Hall, New York.

82. Plesnik, J., (1981). A Bound for the Steiner Tree Problem in Graphs, Mathematica Slovaca, 31, pp. 155-163.

83. Prim, R. C., (1957). Shortest Connection Networks and Some Generalizations, Bell System Technical Journal, 36, pp. 1389-1401.

84. Rayward-Smith, V. J., (1983). The Computation of Nearly Minimal Steiner Trees in Graphs, International Journal of Mathematics Education in Science Technology 14, pp. 15-23.

85. Rayward-Smith, V. J. and Clare, A., (1986). On Finding Steiner Vertices, Networks, 16, pp. 283-294.

86. Romeo, F. and Sangiovanni-Vincentelli, A. L., (1985). Probabilistic Hill Climbing Algorithms: Properties and Applications, Proceedings 1985 Chapel Hill Conference of VLSI, pp. 393-417.

87. Rose, J.S., Blythe, D. R., Snelgrove, W. M., and Vranesic, Z. G., (1986). Fast, High Quality VLSI Placement on an MIMD Multiprocessor, Proceedings IEEE International Conference on Computer-Aided Design, Santa Clara, pp. 42-45.

88. Rossier, Y., Troyon, M., and Liebling Th. M. (1986). Probabilistic Exchange Algorithms and Euclidean Traveling Salesman Problems, OR Spektrum, 8, pp. 151-164.

89. Sasaki, G. H. and Hajek, B., (1988). The Time Complexity of Maximum Matching by Simulated Annealing, Journal of the ACM, 35, pp. 387-403.

90. Sechen, C. and Sangiovanni-Vincentelli, A. L., (1984). The Timberwolf Placement and Routing Package, IEEE Journal of Solid State Circuits, SC-20, pp. 522-527.

91. Segev, A., (1987). The Node-Weighted Steiner Tree Problem, Networks, 17, pp. 1-17.

92. Shannon, C. E., (1948). A Mathematical Theory of Communication, Bell System Technical Journal, 27, pp. 379-623.

93. Shapiro, Jeremy F., (1979). A Survey of Lagrangean Techniques for Discrete Optimization, Annals of Discrete Mathematics, 5, pp. 113-138.

94. Shore, M. L., Foulds, L. R., and Gibbons, P. B., (1982). Algorithm for the Steiner Problem in Graphs, Networks, 12, pp. 323-333.

95. Smith, J. M., Lee, D. T., and Liebman, J. S., (1980). An O(n log n) Heuristic Algorithm for the Rectilinear Steiner Minimal Tree Problem, Engineering Optimization, 4, pp. 179-192.

96. Smith, J. M., MacGregor, J., Lee, D. T., and Liebman, J. S., (1981). An O(n log n) Heuristic for Steiner Minimal Tree Problems on the Euclidean Metric, Networks, 11, pp. 23-39.

97. Smith, J. M. and Liebman, J. S., (1979). Steiner Trees, Steiner Circuits, and the Interference Problem in Building Design, Engineering Optimization, 4, pp. 15-36.

98. Takahashi, H. and Matsuyama, A., (1980). An Approximate Solution for the Steiner Problem in Graphs, Mathematica Japonica, 24, pp. 573-577.

99. Tarjan, R. E., (1977). Finding Optimum Branchings, Networks, 7, pp. 25-35.

100. Van Laarhoven, P. J. M. and Aarts, E. H. L., (1988). Simulated Annealing: Theory and Applications, Kluwer Academic Publishers, Boston, Massachusetts.

101. Vannimenus, J. and Mezard, M., (1984). On the Statistical Mechanics of Optimization Problems of the Travelling Salesman Type, Le Journal De Physique-Lettres, 45. pp. 1145-1153.

102. Wald, J. A. and Colbourn, C. J., (1982). Steiner Trees in Outerplanar Graphs, Congressus Numerantium, 36, pp. 15-22.

103. Wald, J. A. and Colbourn, C. J., (1983). Steiner Trees, Partial 2-Trees, and Minimum IFI Networks, Networks, 13, pp. 159-167.

104. Waxman, Bernard M. and Imase, Makoto, (1988). Worst-Case Performance of Rayward-Smith's Steiner Tree Heuristic, Information Processing Letters, 29, pp. 283-287.

105. White, S. R., (1984). Concepts of Scale in Simulated Annealing, Proceedings of IEEE International Conference on Computer Design, Port Chester, pp. 646-651.

106. Wille, L. T., (1987). The Football Pool Problem for 6 Matches: A New Upper Bound Obtained by Simulated Annealing, Journal of Combinatorial Theory A, 45, pp. 171-177.

107. Winter, P., (1985). An Algorithm for the Steiner Problem in the Euclidean Plane, Networks, 15, pp. 323-345.

108. Winter, P., (1985). Generalized Steiner Problem in Outerplanar Networks, BIT, 25, pp. 485-496.

109. Winter, P., (1986). Generalized Steiner Problem in Series-Parallel Networks, Journal of Algorithms, 7, pp. 549-566.

110. Winter, P., (1987). Steiner Problem in Networks: A Survey, Networks, 17, pp. 129-167.

111. Wong, D. F., Leong, H. W., and Liu, C. L., (1988). Simulated Annealing for VLSI Design, Kluwer Academic Publishers, Boston, Massachusetts.

112. Wong, R. T., (1984). A Dual Ascent Approach for Steiner Tree Problems on a Directed Graph, Mathematical Programming, 28, pp. 271-287.

113. Wu, Y. F., Widemayer, P., and Wong, C. K., (1986). A Faster Approximation Algorithm for the Steiner Problem in Graphs, Acta Informatica, 23, pp. 223-229.

# VITA

Lawrence J. Osborne was born August 9, 1946 in Terre Haute, Indiana. Lawrence received his elementary education in Cuyahoga Falls, Ohio. In 1963, he moved to House Springs, Missouri, where he graduated from Northwest High School in 1964. He received a B.S. in Education with a mathematics major in 1968 from Southeast Missouri State University. After serving three years in the Peace Corps in Jamaica, he taught high school mathematics at several Missouri schools. He received an M.A. in Mathematics at the University of Missouri-Columbia in 1982. After graduation he began teaching in the Mathematics Department at Southwest Missouri State University in Springfield, Missouri. He joined the Computer Science Department there when it was formed in 1983. In 1984, Lawrence returned to graduate school at the University of Missouri-Rolla, earning an M.S. in Computer Science in 1985.

While studying for his doctorate at UMR, Lawrence continued to work at Southwest Missouri State University for one year. For the next two years he worked as a Teaching Assistant at UMR. He is currently employed as an Assistant Professor of Computer Science at the SMSU.