CANADIAN THESES

THÈSES CANADIENNES

NOTICE

The quality of this microfiche is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Previously copyrighted materials (journal articles, published tests, etc.) are not filmed.

Reproduction in full or in part of this film is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30.

AVIS

La qualité de cette microfiche dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

Les documents qui font déjà l'objet d'un droit d'auteur (articles de revue, examens publiés, etc.) ne sont pas microfilmés.

La reproduction, même partielle, de ce microfilm est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30.

THIS DISSERTATION
HAS BEEN MICROFILMED
EXACTLY AS RECEIVED

LA THÈSE A ÉTÉ
MICROFILMÉE TELLE QUE
NOUS L'AVONS REÇUE

On A Generalized Travelling Salesman Problem

by

Anil Kumar Gupta

ISBN 0-315-33322-7

UNIVERSITÉ D'OTTAWA
UNIVERSITY OF OTTAWA

On A Generalized Travelling Salesman Problem

by

Anil Kumar Gupta

A thesis
presented to the University of Ottawa
in partial fulfillment of the
requirements for the degree of
M.Sc. Systems Science
in
Department of Computer Science
University of Ottawa
Ottawa.

OTTAWA, Ontario, 1985

ABSTRACT

In this thesis we study the MINMAX Travelling Salesman Problem (MTSP), which is formally defined as follows. Let $K = (V, E)$ be a complete undirected graph and $C_{ij} = (C_{ij}^k)$, $k = 1,\ldots r$ denote a cost vector associated with arc $(i,j) \in E$. For any subset $E' \subset E$ define $h^k(E') = \sum_{(i,j)\in E'} C_{ij}^k$, $k=1,\ldots r$ and $H(E') = \text{Max } \{h^k(E')\}$. We seek a tour T^* minimizing $H(T)$ over all tours T. For $r = 1$ the MTSP reduces to the Travelling Salesman Problem. We develop good lower bounds on the optimal solution of the MTSP. The bounds are computed using subgradient optimization techniques, and yield, as a by-product, certain 1-trees. We present a procedure which generates feasible tours from the 1-trees obtained from the subgradient optimization. These feasible tours are then used as starting feasible tours in a neighbourhood search heuristic, from which we obtain tours which are demonstratably nearly optimal for MTSP.

ACKNOWLEDGEMENTS

I am grateful for the direction of Prof. A. Warburton for introducing me to this area of research. His help, encouragement and useful suggestions throughout the course of this study have contributed substantially in bringing it to its present form. Also, I wish to thank my thesis supervisor Prof. R.L. Probert who reviewed the thesis and made many useful suggestions. I am thankful to him for his time and patience. I am also thankful to the University of Ottawa for providing me with facilities for research and financial assistance during the course of this research. Finally my indebtedness goes to my wife Chitra and daughter Monic Radhika for allowing me very liberally to take time off especially at the final stages of the thesis.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER I

INTRODUCTION

The Travelling Salesman Problem (TSP) is the problem of finding the shortest tour in a finite weighted undirected graph (the formal definitions are given in Chapter II). This problem has been the subject of intensive investigation in combinatorial optimization [1, 3, 4, 7, 9, 10, 11, 12, 13, 14, 18, 19, 20, 21, 22, 23, 24, 25]. Many combinatorial problems in scheduling and production can be formulated as or shown to be equivalent to the TSP. On the other hand, the TSP is also of theoretical interest because it is an 'NP-Hard' combinatorial problem [6].

We define and study a generalization of the TSP, the MINMAX Travelling Salesman Problem (MTSP) in which we seek a tour which minimizes the maximum of r linear functions defined on the edge set of the underlying graph. The only previous work on this problem is [27], and we shall use some of the results described there. The following are some applications of MTSP:

(i) Generation of trade off curves for 2 criteria problems (e.g. cost/time),

(ii) Sequence dependent scheduling,

(iii) Stochastic scheduling.

These will be considered in detail in Chapter II when we formally define the problem.

In this thesis we concentrate on the approximate solution of MTSP and present a powerful heuristic which yields nearly optimal tours. More specifically, we develop a bounding procedure which provides tight lower bounds on the optimal solution of MTSP. The bounding procedure also allows us to construct certain tours, which are then improved using neighbourhood search techniques. Ultimately we obtain a very good tour for MTSP together with a very good lower bound on the optimal solution. Experiments have confirmed that our approach yields tours which are generally within 5 per cent of optimal on problems with up to 200 cities.

Our bounding procedure is a direct generalization of the work of Held & Karp [13, 14], who exploited subgradient optimization techniques to solve the TSP for r = 1. They introduced the concept of a 1-tree as a graph with nodes 1, 2, ..., n containing a spanning tree on nodes 2, 3, ..., n together with two edges incident with node 1. They observed that:

(i) a tour is precisely a 1-tree in which each node has degree 2,

(ii) a minimum 1-tree is easy to compute and;

(iii) cost tranformations of the form $C_{ij} \rightarrow C_{ij} + \pi_i + \pi_j$ $(\pi = (\pi_i) \in R^n)$ leave the optimal solution to the TSP invariant but change the minimum 1-tree.

Using these observations, they defined an infinite family of lower bounds $W(\pi)$ on W^* (the cost of the optimal tour) and showed that $\max_{\pi} W(\pi) = W^*$ precisely when a certain well-known linear program has an

optimal solution in integers. They developed a subgradient method for computing W^* and a branch and bound method to solve the TSP exactly. We extend and apply their work to the MTSP problem, yielding a subgradient procedure to compute lower bounds on the optimal solution.

Our neighbourhood search techniques are inspired by Lin and Kernighan [20], who gave a highly effective heuristic procedure for generating optimum and near-optimum solutions for the symmetric TSP. Their method is based on exchange of edges in a search for a locally optimal solution starting from an initial feasible tour (generated by some means). We develop a new method for generating initial feasible tours from the output of our bounding procedure, and then apply a heuristic similar to the local optimum search heuristic of Lin and Kernighan [20].

The thesis is organized as follows. In Chapter II we present the problem MTSP and develop lower bounds on its optimal solution. Chapter III describes the subgradient procedure used to compute the lower bounds developed in Chapter II. Chapter IV describes the neighbourhood search procedure and develops a method for generating feasible tours from minimum 1-trees produced by the bounding procedure. In Chapter V, we report the computational results which confirm the effectiveness of our approach. Finally, we suggest directions for further research in Chapter VI.

CHAPTER II

THE PROBLEM, THEORY AND SOME PRELIMINARIES

In this chapter we will formally define the MINMAX Travelling Salesman Problem and develop some powerful lower bounds on its optimal solution. Our results are a direct generalization of corresponding results obtained by Held and Karp for the TSP in [13], [14].

Let $K = (V,E)$ be a complete undirected graph on n vertices $V = (1, 2, \ldots, n)$ with edge set

$$E = \{ (i,j) : i,j \in V,\ i \neq j \}.$$

Here (i,j) is taken to mean the unordered pair consisting of i and j. For each edge $(i,j) \in E$, define r weights $\{C_{ij}^k , k=1,\ldots,r\}$. These weights represent the cost of traversing an edge.

We shall refer to the symmetric matrices $C^k = (C_{ij}^k)_{nxn}$ $k=1,\ldots,r$ as COST MATRICES. For any subset $E' \subseteq E$, define the following r weights:

$$h^k(E') = \sum_{E'} C_{ij}^k, \quad k=1,\ldots,r.$$

Further, let

$$H(E') = \max_k \{ h^k(E') \}$$

denote the MAXIMUM WEIGHT of E' w.r.t. the above weights.

The MINMAX TRAVELLING SALESMAN PROBLEM (MTSP) is to find a tour of minimum maximum weight; that is, a tour T^* such that

$$H(T^*) = \text{Min} \{ H(T) : T \text{ is a tour} \}.$$

Recall that a tour T in K can be specified by a permutation $(i_1,\ldots,i_n)$ of $\{1,\ldots,n\}$. The salesman begins at city i_1, then visits i_2, and so on, finally returning to city i_1 from city i_n.

We will write $H^* = H(T^*)$. Note that for r=1, the above problem reduces to the standard TRAVELLING SALESMAN PROBLEM, abbreviated as TSP. The following are some example instances of the MTSP:

(i) GENERATION OF TRADE OFF CURVES FOR 2-CRITERIA PROBLEMS (E.G. COST/TIME):

Consider a TSP in which there are several alternative arcs between city i and city j, with each arc representing a different travel mode. Each arc has an associated 'cost' and 'time' (the idea can easily be extended to more than 2 criteria associated with each arc).

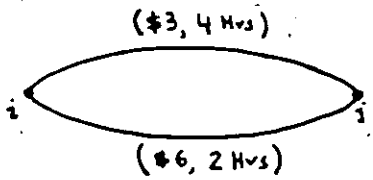

Figure 2.1: A TSP With Several Edges Between 2 Vertices

Each tour T then has a 'cost and a 'time'. We seek the 'trade off' curve, i.e. we seek values (Cost(Tour), Time(Tour)) such that there is no other tour which does better both in terms of cost and time. Bowman [2] shows that a representative set of 'Pareto optimal' alternatives for such a problem would involve solving the following parameteric family of problems:

$$L(\lambda_1, \lambda_2) = \underset{T \in \text{All Tours}}{\text{Min Max}} \{\lambda_1.\text{Cost}(T), \lambda_2.\text{Time}(T)\}$$

for each $\lambda = (\lambda_1, \lambda_2) \in R_+^2$.

To approximate such a curve we must provide an approximate method for the solution of the above MTSP $L(\lambda_1, \lambda_2)$.

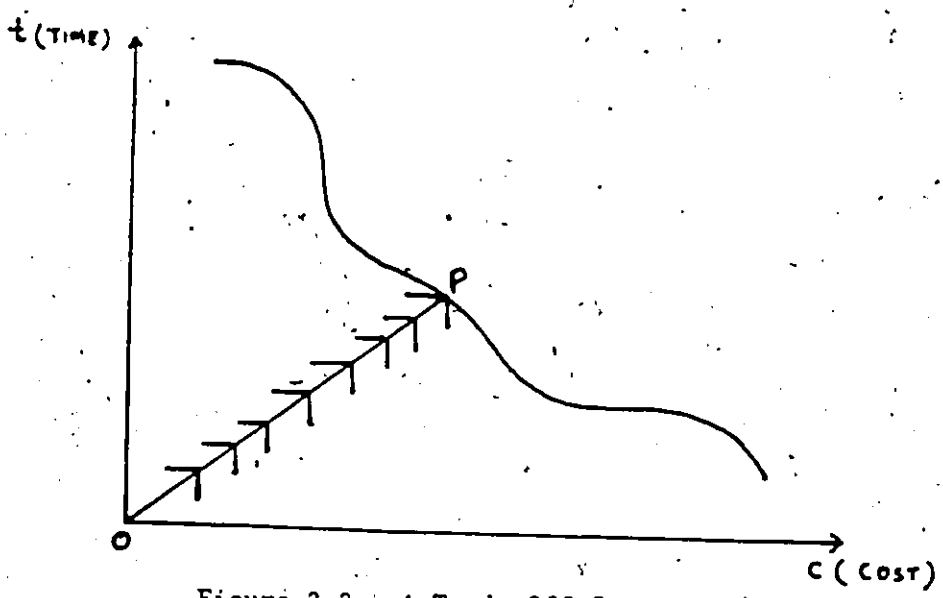

Figure 2.2: A Trade-Off Curve

Note that level curves for a given pair $\lambda \in R_+^2$ will lead us to a point P on the 'trade off' curve (see Figure 2.2). One would miss such a point P on the 'trade off' curve if one was restricted to taking linear combinations of the criteria and then solving the corresponding TSP. After fixing λ, $L(\lambda_1, \lambda_2)$ becomes a MTSP and thus can be solved using our method. In practice we are forced to choose a discrete set of λ's and get an approximation of the 'trade-off' curve.

(ii) SEQUENCE DEPENDENT SCHEDULING PROBLEM:

In this problem, there are r parallel processors and the same n jobs to be scheduled on each processor. The processing time is sequence

dependent and the processing sequence is the same on each processor. Further, when all the jobs are done, the processors are to be reset to the initial setting. It may happen that a processor is still busy even though some other processors have finished. Let C_{ij}^{k} denote the processing time for job j on processor k if the immediately preceding job on processor k was job i. The problem is to schedule the n jobs such that the maximum time on any processor is minimized.

(iii) STOCHASTIC SCHEDULING PROBLEM

Suppose we have r processors and n jobs to be done. We don't know which processor will be available at a given point in time and further we have a processing sequence of these n jobs to be processed. The problem is to find an upper bound on the cost of this tour. The MTSP provides for such an upper bound. Note that if the sequence in which the n jobs will be performed is not known beforehand, the above problem reduces to solving r TSPs.

SOME DEFINITIONS AND NOTATION:

In order to develop the theory required to solve the MTSP, we require some definitions and notation.

SUBGRAPH: $K' = (V',E')$ is called a SUBGRAPH of K if $V' \subseteq V$, $E' \subseteq E$ and $i, j \in V'$ for each $(i,j) \in E'$.

Let $K'=(V',E')$ be a subgraph of K.

PATH: A PATH in $K'=(V',E')$ is a sequence of 2 or more distinct vertices $v_1, \ldots, v_k$ in V' such that $(v_i, v_{i+1}) \in E'$, $1 \leq i \leq k-1$.

CIRCUIT: A CIRCUIT of length $k \geq 3$ in $K'=(V',E')$ is a sequence of nodes $(v_1, v_2, \ldots, v_k, v_1)$ in K', such that $(v_1,\ldots,v_k)$ is a path in K' and $(v_k, v_1) \in E'$.

TOUR: A TOUR in K is a circuit of length n.

CONNECTED SUBGRAPH: The subgraph K' is connected if there exists a path in K' between any two vertices of V'.

SPANNING TREE: A SPANNING TREE of K is a subgraph of K which contains all the vertices of K, is circuit free and connected.

1-TREE: A 1-TREE of K is a subgraph of K consisting of a spanning tree on vertices 2, 3, ..., n together with two edges incident to vertex 1.

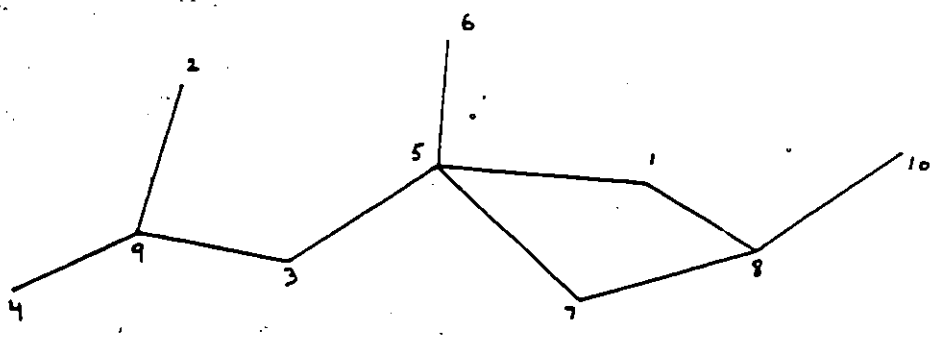

Figure 2.3: A 1-tree

The familiar minimum spanning tree problem seeks a spanning tree of minimum weight, and can be solved by an algorithm that inspects each edge of K exactly once [5, 19]. For our purposes a variant of this problem will be required, in which we seek a MINMAX 1-tree; i.e., a 1-tree of minimum maximum weight over all 1-trees of K.

TWO FUNDAMENTAL OBSERVATIONS:

We now make two fundamental observations which are analogues of Held and Karp's on the TSP. These observations are vital to ensure the validity of our approach.

(a) A MINMAX 1-tree gives a lower bound for the solution of the MTSP.

This follows since every tour is a 1-tree and a 1-tree is a tour if each of its nodes has degree 2. Clearly, if a MINMAX 1-tree is a tour, it is the solution to the MTSP.

(b) An optimal tour for MTSP remains optimal under transformations on costs of the form P

$$P(\pi) : C_{ij}^k \rightarrow C_{ij}^k + \pi_i + \pi_j \qquad k = 1,\ldots,r, \qquad (2.1)$$

where $\pi = (\pi_1,\ldots, \pi_n)$ is an n-vector in R^n.

Observation (b) follows from the more general Lemma 2.1 below.

Let $H(E',\pi)$ be the maximum weight of E'C E when the edge weights are transformed by $P(\pi)$. Observe that $H(E',0) = H(E')$, where 0 denotes the zero vector.

LEMMA 2.1: Let $V' = U\{i,j : (i,j) \in E'\}$, be the set of vertices spanned by E', and d_i be the degree of vertex i in the graph $K'=(V',E')$. Then

$$H(E',\pi) = H(E') + \sum_{i\in V'} \pi_i . d_i$$

PROOF: We have

$$\sum_{(i,j)\in E'} (C_{ij}^k + \pi_i + \pi_j) = \sum_{(i,j)\in E'} C_{ij}^k + \sum_{i\in V'} \pi_i . d_i$$

$$= h^k(E') + \sum_{i\in V'} \pi_i . d_i \qquad k=1,\ldots,r.$$

Hence,

$$H(E', \pi) = \max_k \{h^k(E') + \sum_{i\in V'} \pi_i . d_i\}$$

$$= \max_k \{h^k(E')\} + \sum_{i\in V'} \pi_i . d_i$$

$$= H(E') + \sum_{i\in V'} \pi_i . d_i$$

Q.E.D.

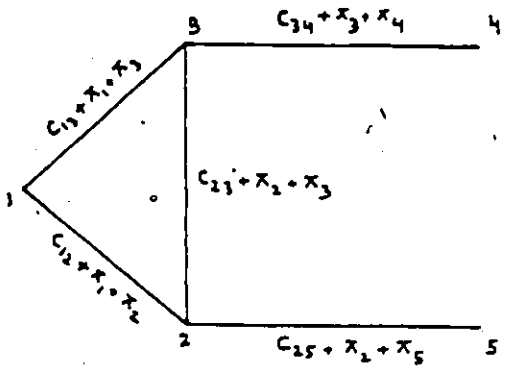

Figure 2.4: An illustration of the transformation $P(\pi)$ (r=1)

For the graph in Figure 2.4, Lemma 2.1 is illustrated as follows:

$$\sum (C_{ij}^k + \pi_i + \pi_j) = (C_{12}^k + \pi_1 + \pi_2) + (C_{13}^k + \pi_1 + \pi_3) + (C_{23}^k + \pi_2 + \pi_3)$$
$$+ (C_{25}^k + \pi_2 + \pi_5) + (C_{34}^k + \pi_3 + \pi_4)$$
$$= H(E') + 2\pi_1 + 3\pi_2 + 3\pi_3 + \pi_4 + \pi_5$$
$$= H(E') + \sum_{i=1}^{5} \pi_i . d_i$$

LEMMA 2.2.: The relative cost of tours remains unchanged under transformations of the form $P(\pi)$.

PROOF: It suffices to show that under $P(\pi)$ the cost of any tour T changes by a constant which is independent of T. Since the degree of every vertex in a tour is 2, we have by Lemma (2.1)

$$H(T,\pi) = H(T) + 2 \sum_{i \in V} \pi_i$$

Hence,

$$H(T,\pi) - H(T) = H(T) + 2 \sum_{i \in V} \pi_i - H(T)$$

$$= 2 \sum_{i \in V} \pi_i$$

which is independent of T.

Q.E.D.

Observation (b) follows directly from Lemma 2.2.

A FAMILY OF LOWER BOUNDS FOR THE SOLUTION OF THE MTSP:

Inspired by Held and Karp, we can now outline a strategy for solving the MTSP. Note that even though the relative costs of tours does not change under the transformation $P(\pi)$ the same is not true for 1-trees. Hence, such transformations do in general affect the structure of a MINMAX 1-tree. Thus, a possible strategy for solving the MTSP is to seek a vector π such that a MINMAX 1-tree with respect to $P(\pi)$ is also a tour. Such a tour would solve the MTSP.

Further, observations (a) and (b) enable us to obtain a lower bound on the cost of an optimal tour for MTSP. For each π

$$H^* + 2 \sum_{i \in V} \pi_i \geq \min_{\rho \in OT} \{H(\rho, \pi)\}$$

where H^* is the cost of an optimal tour, ρ is a 1-tree, and OT is the set of 1-trees. By Lemma 2.1, we have

$$H^* + 2 \sum_{i \in V} \pi_i \geq \min_{\rho \in OT} \{H(\rho) + \sum_{i \in V} \pi_i . d_{i\rho}\}$$

where $d_{i\rho}$ denotes the degree of node i in the 1-tree ρ.
Hence for each π

$$H^* \geq \min_{\rho \in OT} \{H(\rho) + \sum_{i \in V} \pi_i \cdot (d_{i\rho} - 2)\}$$

Let $V(\rho)$ be a vector whose i^{th} component is $(d_{i\rho} - 2)$. The above can be rewritten as

$$H^* \geq \min_{\rho \in OT} (H(\rho) + \pi . V(\rho))$$

$$= W(\pi)$$

where '.' denotes the inner product.

Thus a family of lower bounds $W(\pi)$ has been provided for H^*, one for each $\pi \in R^n$. Let the best of these bounds be denoted by W^*. Then

$$W^* = \max_{\pi} \{W(\pi)\}$$

$$= \max_{\pi} \min_{\rho} \{H(\rho) + \pi . V(\rho)\}$$

W^* can also be expressed as the solution of the linear program,

$$W^* = \text{Max } Z$$

$$\text{subject to} \quad Z \leq H(\rho) + \pi.V(\rho) \quad \text{for all } \rho \varepsilon OT.$$

Note that there is one constraint for each 1-tree. Thus the order of the number of constraints is gigantic - e.g. it is easy to see that there are at least (n-1)! spanning trees for a graph containing n vertices.

The following question immediately arises concerning W^*. Is there a practical technique for computing W^*?

For r=1, Held and Karp answered the question in the affirmative by solving the above linear program. They used a subgradient procedure that requires the calculation of a sequence of minimum weight 1-trees, each of which is easily calculated. (Using PRIM's [16] or any other good algorithm for minimum spanning trees, one can easily find minimum spanning tree on vertices 2, 3, ..., n. Then, adjoining two minimum weight edges at vertex 1 will yield a minimum weight 1-tree). In order to apply a subgradient procedure to calculate W^* for $r \geq 2$, an efficient means of calculating MINMAX 1-trees is required. This will be difficult since the MINMAX 1-tree decision problem is NP-complete. In order to circumvent this problem, we weaken the bound W^* slightly to a related bound t^*.

THE BOUND t^*

Let $S = \{ (\lambda^1, \ldots, \lambda^r) : \Sigma \lambda^k = 1, \lambda^k \geq 0, \quad k=1,\ldots,r\}$ (2.2)

be the unit simplex in R^r. Then, for any 1-tree ρ, we have

$$H(\rho)=\max_{k=1,\ldots,r} \{h^k(\rho)\}$$

$$\geq \sum_{k=1}^{r} \lambda^k . h^k(\rho) \qquad \text{for any } \lambda \in S.$$

Hence, for any $\pi = (\pi_1, \ldots, \pi_n)$ and for any $\lambda=(\lambda^1,\ldots,\lambda^r)\in S$

$$H^* + 2 \sum_{i=1}^{n} \pi_i \geq \min_{\rho} (H(\rho) + \sum_{i} \pi_i . d_{i\rho})$$

$$\geq \min_{\rho} (\sum_{k=1}^{r} \lambda^k . h^k(\rho) + \sum_{i=1}^{n} \pi_i . d_{i\rho})$$

where $\sum_{k=1}^{r} \lambda^k . h^k(\rho)$ becomes the weighted sum of the original weights.

Therefore,

$$H^* \geq \min_{\rho} (\sum_{k=1}^{r} \lambda^k . h^k(\rho) + \sum_{i=1}^{n} \pi_i . (d_{i\rho} - 2)).$$

Letting $h(\rho) = (h^1(\rho), \ldots, h^r(\rho))$, this can be rewritten as

$$H^* \geq \min_{\rho} (\lambda . h(\rho) + \pi . V(\rho)) = t(\lambda, \pi).$$

Thus, we obtain a family of lower bounds $t(\lambda,\pi)$ on H^*, the best of which is our weakened bound t^*. That is

$$t^* = \max_{\lambda,\pi} \{ t(\lambda,\pi)\}.$$

We then have $H^* \geq W^* \geq t^*$, and

$$t^* = \max_{\lambda} \max_{\pi} \min_{\rho} (\lambda . h(\rho) + \pi . V(\rho)).$$

We can write the above expression for t^* as a linear program in the following way:

$$\begin{aligned} t^* &= \max Z \\ \text{subject to} \quad & Z \le \lambda.h(\rho) + \pi.V(\rho) \text{ for all } \rho \in OT, \qquad (2.3) \\ & \lambda.e_r = 1 \\ & \lambda \ge 0 \end{aligned}$$

where e_r is an r-vector of 1's.

For fixed λ and π, the LP (2.3) can be solved as a simple minimum 1-tree problem with edge weights $\{(\sum_{k=1}^{r} \lambda^k C_{ij}^k + \pi_i + \pi_j),\ k=1, \ldots, r\}$. We will exploit this fact in the next chapter, where we develop a subgradient procedure which solves the linear program (2.3). The procedure computes a sequence of values $\{t(\lambda,\pi)\}$, obtained by solving minimum 1-tree problems, which converge to t^*.

PROPERTIES OF THE BOUND t^*

We now consider the quality of the lower bound t^*.

Following Held & Karp [13] we shall establish a relationship between our bound t^* and an LP relaxation of MTSP. The development closely follows the corresponding theory of Held and Karp [13] for TSP.

Let us define, for each edge (i,j), a variable x_{ij} as follows:

$$x_{ij} = \begin{cases} 1 & \text{if } (i, j) \in E, \\ 0 & \text{otherwise.} \end{cases}$$

Further, define $x = (x_{ij})$ to be an $\binom{n}{2}$ - vector in the space $R^{n(n-1)/2}$. We also need the following definition.

NODE-EDGE INCIDENCE MATRIX: Let $G = (N, E)$ be a graph. A node-edge incidence matrix A of G is a matrix with one row for each node i in N, and one column for each edge $e \in E$. This column is of the form

$$(0\ 0\ \underset{i}{1}\ 0\ 0\ \underset{j}{1}\ 0\ 0)^T,$$

where e connects node i and node j.

In the special case when G = K, the complete graph on n nodes, A is an $n \times n(n-1)/2$ matrix

$$A = \begin{bmatrix} 1\ 1\ .\ .\ 1\ 0\ 0 & & & 0 & & & 0\ 0\ 0 \\ 1 & & 1\ 1\ .\ .\ 1\ 0\ 0\ .\ .\ 0\ .\ .\ . & & & 0\ 0\ 0 \\ & 1 & 0\ 1 & & 0\ 1\ 0\ ..\ 0\ 0 & & 0\ 0\ 0 \\ & & & & & & \\ & & & & & & \\ & & & & & & 10 \\ & & & & & & 0\ 1 \\ & & 1 & & 1 & 1 & 0\ 0\ 0\ 1\ 1 \end{bmatrix},$$

where the n(n-1)/2 columns correspond to the edges (1, 2), (1, 3), .., (1, n), (2, 3), .., (2, n), .., (n-2, n-1), (n-2, n), (n-1, n) respectively. Let $e_n = (1, 1\ .\ .,\ 1)$ denote an n-vector of 1's.

Now define

$$S_{2M} = \{\ x : Ax = 2e_n,\ 0 \leq x \leq 1\ \}, \tag{2.4}$$

$$S_{1T} = \{\ x : A'x \leq b'\}, \tag{2.5}$$

where matrix A' and vector b' are such that the linear system $A'x \leq b'$ satisfies the following constraints:

$$\sum_{j=1}^{n} x_{1j} = 2 \qquad (2.6)$$

$$\sum_{1 \leq i < j \leq n} x_{ij} = n \qquad (2.7)$$

$$\sum_{\substack{i,j \in S \\ i<j}} x_{ij} \leq |S| - 1 \qquad (2.8)$$

for any subset $S \subset \{2, 3, \ldots n\}$

$$0 \leq x_{ij} \leq 1 \text{ for all } i, j. \qquad (2.9)$$

Suppose that x is required to be an integer vector. Then (2.4) above defines a set of '2 Matching' constraints. (2.6) says that node 1 has degree 2, (2.7) implies that there are exactly n edges, and (2.8) defines a set of 'subtour elimination constraints', that is, no cycle is formed among any subset of {2, 3, ..., n}. Note that (2.5), which is a combination of (2.6) - (2.9), represents 1-tree constraints. Further, any feasible solution to (2.4) and (2.5) must be a tour.

We need the following result of Held & Karp Theorem 1, [13].

THEOREM 2.4: Let $T^1, T^2, \ldots, T^q$ be the 1-trees on the vertex set {1, 2, ... n}. Associate with T^k the $\binom{n}{2}$ - vector (e_{ij}^k) defined by:

$$e_{ij}^k = \begin{cases} 1 & \text{if } (i, j) \text{ is an edge of } T^k, \\ 0 & \text{otherwise.} \end{cases}$$

Then the extreme points of the polyhedron $A'x \leq b'$ are the points (e_{ij}^k) where $1 \leq k \leq q$ and $1 \leq i < j \leq n$.

Thus, integrality of x is automatic in the 1-tree constraints (2.5). As a consequence of the above theorem, a linear program of the form

$$\text{Min } \{ dx : A' x \leq b', x \geq 0 \}$$

represents the problem of finding a minimum 1-tree w.r.t. the weights d_{ij} $(d=(d_{ij}))$.

Let $C^k=(c^k_{ij})$ denote the k^{th} cost vector. Then MTSP can be expressed as an IP in the following way:

$$\begin{aligned} & \text{Min } Z \\ \text{subject to} \quad & Ax = 2e_n \\ & A'x \leq b' \\ & -Z + C^k.x \leq 0 \quad k = 1, \ldots, r \\ & x_{ij} \text{ is 0 or 1.} \end{aligned} \tag{2.10}$$

We now show that our bound $t^* = \underset{\lambda,\pi}{\text{Max}}\, t(\lambda,\pi)$ is, in fact, equal to the value of the optimal solution of the LP-relaxation of the above IP for MTSP.

Dropping the integrality restriction, we can write the dual of (2.10) as:

$$\begin{aligned} & \text{Max } (2ye_n + y'b') \\ \text{subject to} \quad & yA + y'A' + \mu C \leq 0 \\ & -\mu e_r = 1 \\ & y' \leq 0, \ \mu \leq 0 \end{aligned}$$

i.e.

$$\begin{aligned} & \text{Max } (-2\pi e_n - \pi'b') \\ \text{subject to} \quad & -\pi A - \pi'A' - \lambda C \leq 0 \\ & \lambda . e_r = 1 \\ & \pi' \geq 0, \ \lambda \geq 0 \end{aligned}$$

$$\begin{array}{ll} \text{i.e.} & \text{Max } (-\pi b - \pi' b') \\ \text{subject to} & \pi A + \pi' A'' + \lambda C \geq 0 \\ & \lambda . e_r = 1 \\ & \pi' \geq 0,\ \lambda \geq 0 \end{array}$$

where $e_r = (1, \ldots 1)$ is an r-vector of 1's, $b = 2e_n$, and

$$C = (C^1, C^2, \ldots C^r)^T.$$

We can write the above as

$$\underset{\substack{\pi,\ \lambda \\ \lambda . e_r = 1 \\ \lambda \geq 0^r}}{\text{Max}} \left[\begin{array}{ll} & \underset{\pi}{\text{Max}} (-\pi b - \pi' b') \\ \text{subject to} & \pi' A' \geq -\pi A - \lambda C \\ & \pi' \geq 0 \end{array} \right]$$

where π and λ are fixed in the inner maximization.

Dualizing the inner maximization problem with π regarded as constant, we have

$$\underset{\substack{\pi,\ \lambda \\ \lambda . e_r = 1 \\ \lambda \geq 0^r}}{\text{Max}} \left[\begin{array}{ll} & \text{Min } (-\pi A x - \lambda C x - \pi b) \\ \text{subject to} & A' x \geq -b' \\ & x \leq 0 \end{array} \right]$$

i.e.

$$\underset{\substack{\pi,\ \lambda \\ \lambda . e_r = 1 \\ \lambda \geq 0^r}}{\text{Max}} \left[\begin{array}{ll} & \text{Min } (-\pi A(-x) - \lambda C(-x) - \pi b) \\ \text{subject to} & A'(-x) \geq -b' \\ & (-x) \leq 0 \end{array} \right]$$

i.e.

$$\underset{\substack{\pi,\ \lambda \\ \lambda . e_r = 1 \\ \lambda \geq 0^r}}{\text{Max}} \left[\begin{array}{ll} & \text{Min } (\pi A x + \lambda C x - \pi b) \\ \text{subject to} & A' x \leq b' \\ & x \geq 0 \end{array} \right]$$

i.e.

$$\begin{bmatrix} \text{Min } (\pi(Ax - b) + \Sigma\lambda^k C^k x) \\ \text{subject to} \quad A'x \leq b' \\ x \geq 0 \end{bmatrix}$$

$$\text{Max}_{\pi,\ \lambda,\ \lambda.e_r=1,\ \lambda \geq 0^r}$$

$$= \text{Max}_{\lambda,\ \pi} \ t(\lambda,\pi) \quad \text{(by definition of } t(\lambda,\pi))$$

$$\lambda.e_r = 1$$

$$\lambda \geq 0.$$

ANOTHER INTERPRETATION FOR t^*

The LP (2.3) for t^* is

$$t^* = \text{Max } Z$$

$$\text{subject to} \quad Z \leq \sum_{k=1}^{r} \lambda^k h_\rho^k + \sum_i \pi_i V_{i\rho} \quad \text{for all } \rho$$

$$\sum_{k=1}^{r} \lambda^k = 1$$

$$\lambda^k \geq 0 \qquad k=1,\ldots,r.$$

Its dual is

$$\text{Min } W$$

$$\text{subject to} \quad y_\rho \geq 0$$

$$\sum_\rho y_\rho = 1$$

$$W \geq \sum_\rho y_\rho h_\rho^k \qquad k = 1, \ldots, r,$$

$$\sum_\rho y_\rho.(-V_{i\rho}) = 0 \qquad i = 1, \ldots, n.$$

That is

$$\text{Min} \max_k \{\Sigma_\rho y_\rho h_\rho^k\}$$

subject to

$$y_\rho \geq 0 \qquad (2.11)$$

$$\Sigma_\rho y_\rho = 1 \qquad (2.12)$$

$$\Sigma_\rho y_\rho \cdot (V_{i\rho}) = 0, \quad i = 1, \ldots, n. \qquad (2.13)$$

Constraints (2.11) and (2.12) define weights and (2.13) states that on the average each node has degree 2 over all 1-trees. Thus (recall $V_{i\rho} = d_{i\rho} - 2$) we seek a convex combination of 1-trees such that each vertex has on the average degree 2, and which minimizes the maximum average weight over the r criteria.

In this chapter we have derived some theoretical properties of the bound t^* and have shown that it may be obtained by solving a certain LP relaxation of MTSP. In the next chapter we develop a practical technique for the computation of t^*.

CHAPTER III

COMPUTATION OF THE LOWER BOUND

In this chapter we outline a subgradient procedure for computing the bound t^* defined by the LP (2.3). We shall see in Chapter V that the procedure has solved (2.3) for problems with up to 200 cities. We draw freely on results in [13], [14], [15].

For completeness we present the lemma 3.1 below which provides the rationale for the procedure, and points the way towards the determination of t^* via a sequence of minimum 1-tree calculations. Let $\rho(\lambda,\pi)$ be a minimum 1-tree at (λ,π). That is, $\rho(\lambda,\pi)$ is a 1-tree minimum with respect to the edge weights

$$\sum_{k=1}^{r} \lambda^k c_{ij}^k + \pi_i + \pi_j, \qquad k=1,\ldots,r.$$

LEMMA 3.1 : Let $(\bar{\lambda},\bar{\pi})$ be such that $t(\bar{\lambda},\bar{\pi}) \geq t(\lambda,\pi)$. Then

$$\left((\bar{\lambda},\bar{\pi}) - (\lambda,\pi) \right).\left(h(\rho(\lambda,\pi)),\ V(\rho(\lambda,\pi))\right) \geq 0$$

PROOF:

$$t(\lambda,\pi) = h(\rho(\lambda,\pi)).\lambda + \pi.V(\rho(\lambda,\pi))$$

$$= \left(h(\rho(\lambda,\pi)),\ V(\rho(\lambda,\pi))\right) \ .\ (\lambda,\pi)$$

$$t(\bar{\lambda},\bar{\pi}) = \operatorname{Min}_{\rho\in OT} \left(h(\rho),V(\rho)\right).(\bar{\lambda},\bar{\pi})$$

$$\leq \left(h(\rho(\lambda,\pi)),V(\rho(\lambda,\pi))\right).(\bar{\lambda},\bar{\pi})$$

Thus

$$0 \leq t(\bar{\lambda},\bar{\pi}) - t(\lambda,\pi)$$

$$\leq \left(h(\rho(\lambda,\pi)),V(\rho(\lambda,\pi))\right).\left((\bar{\lambda},\bar{\pi}) - (\lambda,\pi)\right).$$

Q.E.D.

Lemma 3.1 states that all points $(\bar{\lambda},\bar{\pi})$ such that $t(\bar{\lambda},\bar{\pi}) - t(\lambda,\pi) \geq 0$, lie in the positive half space defined by the hyperplane through (λ, π) with normal $(h(\rho(\lambda,\pi)), V(\rho(\lambda,\pi)))$. In particular, any optimal solution (λ^*,π_*) of the LP(2.3) defining t^* lies in that half space. This suggests that, if (λ,π) is not optimal, we move into the half space. The following iterative procedure accomplishes this: we begin with λ^0, t_0^0, t_1^0, π_0 and $\rho(\lambda^0,\pi_0)$ and repeat the following step

$$\lambda^{k+1}=P_S[\lambda^k + t_1^k.h(\rho(\lambda^k,\pi_k))], \tag{3.1}$$

$$\pi_{k+1}=\pi_k+t_0^k.V(\rho(\lambda^k,\pi_k)), \tag{3.2}$$

where $\rho(\lambda^k,\pi_k)$ is the minimum 1-tree at (λ^k,π_k); t_0^k and t_1^k are positive scalars. P_S denotes projection onto the unit simplex S defined by (2.2), and is necessary for normalization of λ^{k+1}.

(3.1) and (3.2) define a sequence $\{(\lambda^k,\pi_k)\} \subset S \times R^n$ which, for an appropriate choice of scalars $\{t_0^k\}$ and $\{t_1^k\}$, converges to an optimal solution (λ^*,π_*) of the LP(2.3) defining t^* [14], [15]. The details of initializations, the choice of step sizes, the projection operator P_S and the stopping rule for the procedure are given in the next section.

The main idea of the convergence proof is this: Let (λ^*, π_*) be any point in the optimal set, that is, $t(\lambda^*,\pi_*) = t^*$. By Lemma 3.1, $(h(\rho(\lambda^k,\pi_k)), V(\rho(\lambda^k,\pi_k))$ is a subgradient at (λ^k,π_k), and

$$(h(\rho(\lambda^k, \pi_k)), V(\rho(\lambda^k, \pi_k))).((\lambda^*, \pi_*) - (\lambda^k, \pi_k)) \geq 0.$$

Thus if (λ^k,π_k) is not an optimal point, then the direction $(h(\rho(\lambda^k,\pi_k)), V(\rho(\lambda^k,\pi_k)))$ makes an acute angle with the ray from (λ^k,π_k) through (λ^*,π_*). Consequently, if t_o^k and t_1^k are sufficiently small the point $(\lambda^k+t_1^k.(h(\rho(\lambda^k,\pi_k)),\pi_{k+1})$ will be closer to (λ^*, π_*) than was (λ^k,π_k). It is easy to show that the same is true for the projected point $(\lambda^{k+1},\pi_{k+1})$. Thus the sequence (λ^k,π_k) approaches the optimal set although the objective function values $t(\lambda^k,\pi_k)$ are not necessarily monotonic. Sufficient conditioning for the convergence of $t(\lambda^k,\pi_k)$ are that $t_0^k \to 0$, $t_1^k \to 0$ and $\sum_{k=1}^{\infty} (t_0^k+t_1^k) = \infty$. For sufficiently large values of k, $t_0^k = t_1^k$.

We now discuss in more detail the choice of step sizes t_0^k, t_1^k in (3.1) and (3.2). Our choices are based partly upon the discussion in [15], where several different possibilities are suggested.

CHOICE OF STEP SIZES

A general method which often (but not always) works well is the following [15]:

Set $t_0^k = t_1^k = 2$ for 2n iterations (where n is a measure of the problem size), and then successively halve both the value of t_0^k, t_1^k and the number of iterations until the number of iterations reaches some threshold value Z; from that point on the step sizes are halved every Z iterations until the resulting steps are very small. Our computational experiments indicated that using the same step size $t_0^k = t_1^k$ to update π and λ is not efficient, and leads to many unnecessary iterations in which there is no substantial improvement in the value of $t(\lambda^k,\pi_k)$. For problems with upto 200 cities and r=2, we found that the following

choices for t_0^k, t_1^k and Z worked very well: Choose Z=5, initialize and update t_0^k as described above, and use the following rule to update the step size t_1^k.

$$t_1^k = \text{Min } (t_0^k,\ 1/(2 \sum_{k=1}^{r} h^k)) \qquad (3.3)$$

RATIONALE

To explain the reason for the above choice of t_1^k we refer to Figure 3.1. If t_1^k is not small enough, the point $\lambda + t_1^k.H$ can bounce far off and then project onto one of the boundary points (1, 0) or (0, 1) of the simplex. This will repeat itself for many iterations until t_1^k is such that $\lambda + t_1^k$ lies within the region P. On the other hand a very small t_1^k will not take us anywhere. Thus a prudent choice of t_1^k is necessary.

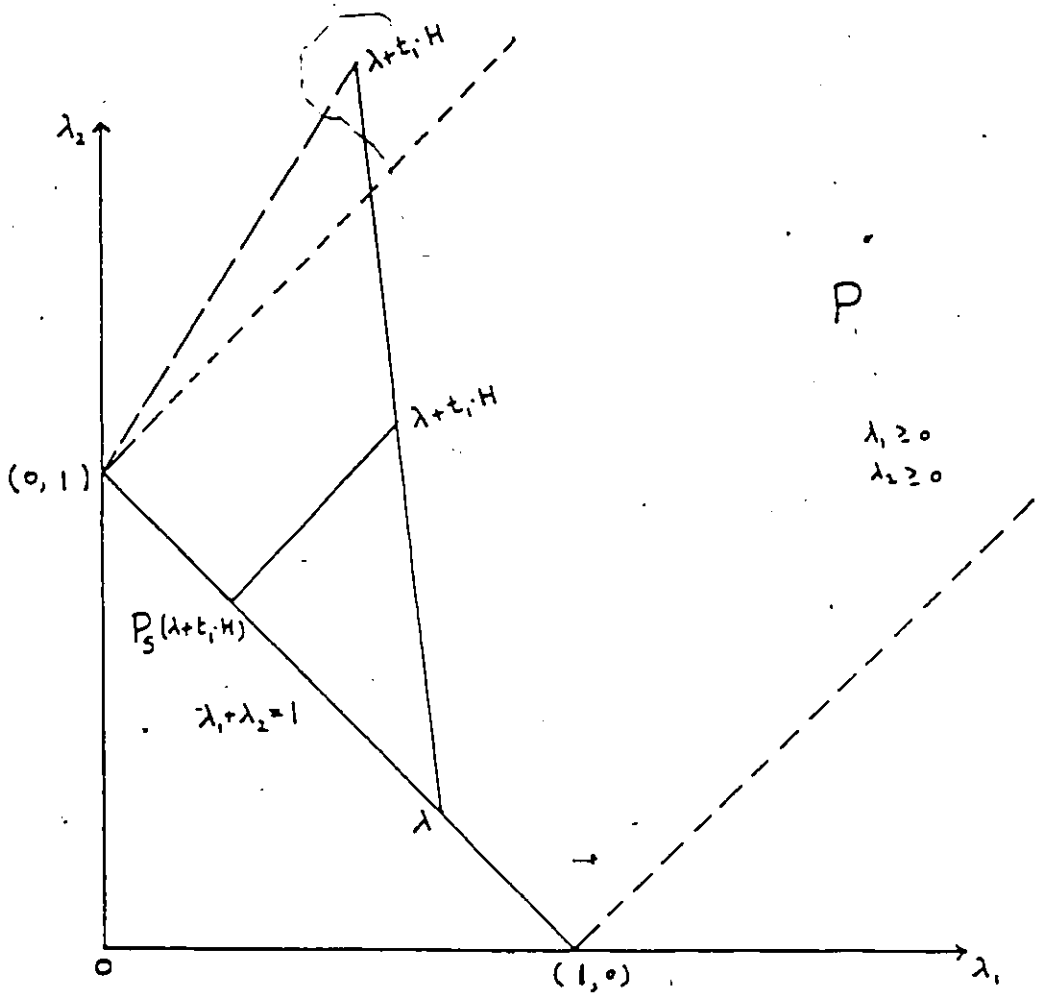

Figure 3.1: Illustration of Step Size t_1^k for λ.

MORE DETAILED DESCRIPTION OF SUBGRADIENT PROCEDURE

With the t_0^k, t_1^k values chosen as above, the iterations proceed as follows. Initially

$$\pi_1=(0,\ldots,0),$$
$$\lambda^1=(1/r,\ldots,1/r),$$
$$t_0^1=2.0,$$
$$t_1^1=t_0^1.$$

We now use Prim's algorithm [16] to compute a minimum 1-tree with edge weights defined, for $(i,j)\varepsilon E$, by

$$d_{ij}^1 = \frac{1}{r}\sum_{k=1}^{r} c_{ij}^k,$$

thus obtaining a 1-tree $T_1=\rho(\lambda^1,\pi_1)$. T_1 yields the vectors

$$h(T_1)=(h_1(T_1),\ldots h_r(T_1))$$

and

$$V(T_1)=(Deg_1-2,\ldots,Deg_n-2),$$

where Deg_i-2 is the degree of vertex i in T_1, i=1,...,n.

The vectors $h(T_1)$ and $V(T_1)$, together with t_0^1 and t_1^1, yield, via (3.1) and (3.2) respectively, vectors $\lambda^2=(\lambda_1^2,\ldots,\lambda_r^2)\varepsilon S$ and $\pi_2=(\pi_2^1,\ldots,\pi_2^n)\varepsilon R^n$. We now use λ^2 and π_2 to update the edge weights; viz

$$d_{ij}^2 = \sum_{k=1}^{r}\lambda_k^2 c_{ij}^k + \pi_2^i + \pi_2^j.$$

A minimum 1-tree $T_2=\rho(\lambda^2,\pi_2)$ is computed w.r.t. the edge weights d_{ij}^2, yielding $h(T_2)$ and $V(T_2)$, which are then used to compute λ^3,π_3, via (3.2), and (3.3); etc. The process continues until no further significant increase in the values $t(\lambda^k,\pi_k)$ is achieved or until an iteration limit is reached. More precisely, the stopping rule operates as described below.

STOPPING RULE

We define TOL to be the acceptable level of fluctuation in the value of $t(\lambda,\pi)$. We set its value to .01 in our experiments.

If iteration limit reached, then STOP.

If absolute change in the weight of a best 1-tree so far and weight of a best 1-tree 30 interations before is less than TOL and also absolute change in the weight of the minimum 1-tree at the present iteration from the weight of a minimum 1-tree 30 iterations before is less than TOL then STOP (this testing is necessary because our objective function $t(\lambda, \pi)$ is not monotonic and thus we must monitor its fluctuation along with the change in the value of the best lower bound so far).

THE PROJECTION OPERATOR P_S

Following [15], we describe details of the projection operator P_S in (3.1).

Let S be the unit simplex in R^r. Let P_S be the projection operator projecting R^r onto S; that is, for any $x \in R^r$, the point $P_S(x)$ is the unique point of S nearest x.

Suppose $\bar{\lambda} \in R^r$. Then $P_S(\bar{\lambda})$ can be calculated by solving the following convex quadratic programming problem:

$$\text{Min}\{(1/2)(\lambda - \bar{\lambda})^T.(\lambda - \bar{\lambda}) : \Sigma \lambda_j = 1, \lambda_j \geq 0, j = 1, 2, \ldots r\}$$

By the Kuhn-Tucker theorem [15], a necessary and sufficient condition that $\lambda=(\lambda_1,\ldots,\lambda_r) \in S$ solve the above problem is the existence of multipliers u and $v_1, v_2, \ldots, v_r$ such that

$$\lambda_j - \bar{\lambda}_j = v_j - u$$
$$v_j \geq 0$$
$$\lambda_j . v_j = 0 \quad \text{for all } j$$

Note that if we set

$$\lambda_j(u) = \max \{ \bar{\lambda}_j - u, 0\}$$

$$v_j(u) = \max \{ -(\bar{\lambda}_j - u), 0\}$$

then the Kuhn-Tucker conditions are satisfied for $\lambda = \lambda(u)$, $v = v(u)$ and we have $\lambda \geq 0$. It is then only necessary to find that value u^* of u such that $\sum_j \lambda_j(u) = 1$. u^* is found as follows.

We can assume that the coordinates of $\bar{\lambda}$ are ordered so that

$\bar{\lambda}_1 \leq \bar{\lambda}_2 \leq \bar{\lambda}_3 \leq \ldots, \leq \bar{\lambda}_r$. Let J_0 be the first index such that

$$\sum_{i>J_0} (\bar{\lambda}_i - \bar{\lambda}_{J_0}) \leq 1.$$

That is

$$\sum_{i=J_0+1}^{r} \bar{\lambda}_i - (r-J_0)\,\bar{\lambda}_{J_0} \leq 1.$$

Hence

$$(\sum_{i=J_0+1}^{r} \bar{\lambda}_i - 1)/(r-J_0) \leq \bar{\lambda}_{J_0}.$$

Set $u^* = ((\sum_{i=J_0+1}^{r} \bar{\lambda}_i - 1)/(r-J_0))$.

Then $u^* \leq \bar{\lambda}_{J_0} + 1$ and

$$\sum_{i=J_0+1}^{r} (\bar{\lambda}_i - u^*) = \sum_{i=J_0+1}^{r} \bar{\lambda}_i - (r - J_0)u^* = 1.$$

SUMMARY

We have seen that the iterative procedure (with steps 3.1 and 3.2) is well defined and under appropriate conditions converges to the desired bound t^*. At the end of the iterative procedure we have available a close approximation to t^* together with a corresponding 1-tree. This 1-tree will be used in the following chapters to construct an initial feasible tour which is then refined by a local optimum search strategy to yield a good upper bound to MTSP.

CHAPTER IV

TOUR GENERATION BY NEIGHBOURHOOD SEARCH

In this chapter we develop a heruristic for the generation of tours. The bounding procedure of Chapter III enables us to evaluate the quality of tours obtained according to the ratio (UB - LB)/LB, where UB is an upper bound obtained from the heuristic of this chapter and LB is a lower bound on the cost of an optimal tour as obtained in Chapter III. Thus we have both upper and lower bounds on the cost of an optimal tour. We will see in Chapter V that the tours we obtained were extremely good, and varied between bounds of 1.5 per cent and 3.2 per cent from the optimal tour.

The most widely applied general heuristic technique is that of 'Neighbourhood Search' described by Lin and Kernighan in [20]. A preselected set of local operations is used to repeatedly improve upon an initial feasible solution, continuing until no further local improvement is possible and a locally optimal solution has been obtained. We choose to follow their approach here.

We repeat the general strategy described in [20].

LOCAL OPTIMUM

1. Generate by some means an initial feasible tour T.
2. Attempt to find an improved feasible solution T' by some transformation of T.

3. If an improved solution is found then replace T by T' and repeat from Step 2.
4. If no improvement can be achieved, T is called a locally optimal solution. Repeat from Step 1 until either computational time runs out or answers are satisfactory.

The main step in the above algorithm is Step 2 where we transform an initial tour into a locally optimal tour, hopefully approaching a global optimum at the same time. In step 2, the transformation that has been applied to a variety of problems is the exchange of a fixed number k of edges from T with k edges from E-T such that the resulting set T' of edges is a feasible solution and better than T. This is repeated as long as such groups can be found. Eventually it will not be possible to improve T further by such exchanges, at which time we have, by definition, a locally optimal tour.

THE NEIGHBOURHOOD SEARCH HEURISTIC:

We closely follow the LOCAL OPTIMUM heuristic of [20] described above. However, we do adopt a slightly more restrictive definition of local optimality and restrict ourselves to the values k=2 and k=3. We obtained very good results from this heuristic as is evident from the computational results to be described in Chapter V. We defer the discussion of selecting initial tours (Step 1 of LOCAL OPTIMUM) until later. A tour T' is called a k-NEIGHBOUR of a tour T if it is obtained from T by exchanging k edges of T with k edges from E-T.

For our purposes a tour is locally optimal if it cannot be improved by edge exchange of the type described in our procedure below.

We start with a tour $T=(T_1, T_2, \ldots, T_n)$ in K as shown in Figure 4.1. Let $\ell(1\leq\ell\leq r)$ be the criterion for which $H(T)=h^{\ell}(T)$.

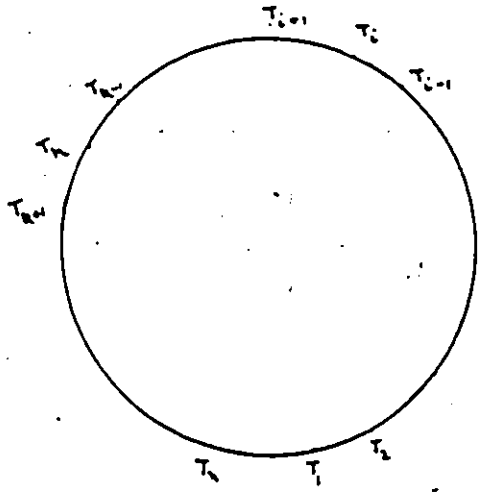

Figure 4.1: A Tour

Our procedure is explained by the flow-chart of Figure 4.2. We will describe the procedure TOUR following the description of the flow-chart. Starting with a tour T and j=1, we reindex T such that $T_1=1$. We then subject it to the procedure TOUR for improvement. If we obtain a better tour in the process then we replace T with the improved tour T' and repeat the whole process with the new T. Otherwise T is reindexed such that $T_1=j+1$ and we repeat the process. Ultimately the procedure will yield a locally optimal tour.

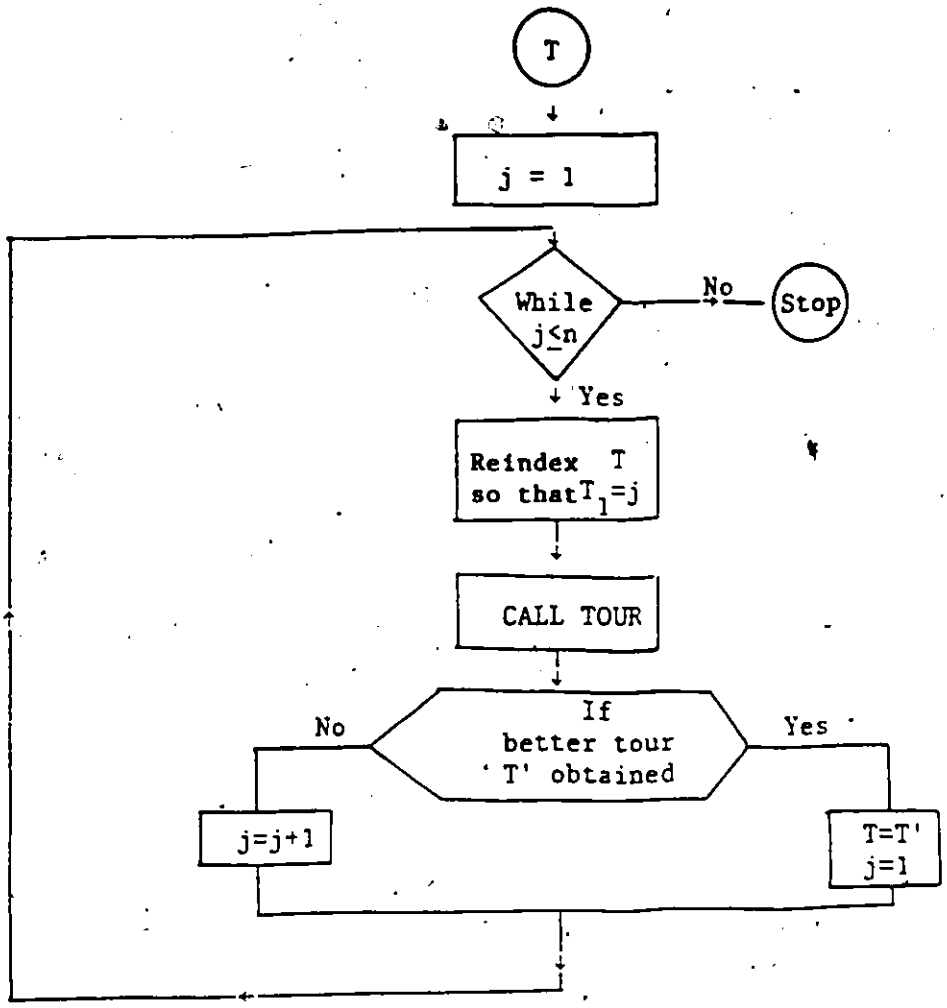

Figure 4.2: The Neighbourhood Search Heuristic

Now we describe the procedure TOUR.

We start with an initial tour T. In the search for a 2-neighbour we search for a replacement for the edge T_nT_1 which is better than T_nT_1 with respect to the criterion ℓ (i.e. ℓ^{th} criterion weight on the replacement edge is less than the ℓ^{th} criteria weight on T_nT_1). If no such improved edge is found, we return. Otherwise, let T_i be the first vertex (moving anti-clockwise from T_1) for which such an improved edge is found.

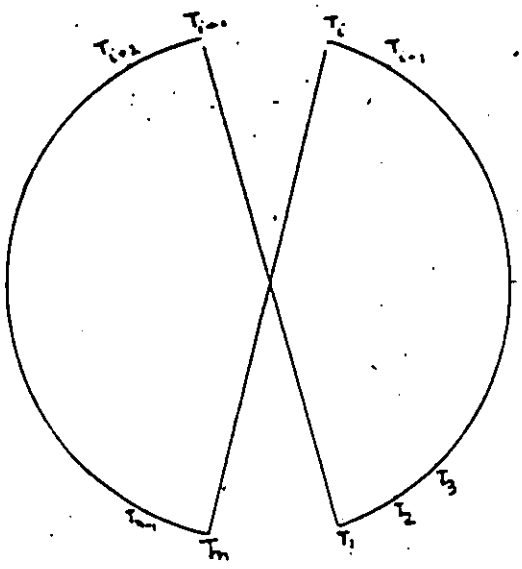

Figure 4.3: A 2-Neighbour T'' of T

The 2-neighbour T'' of T will then look as in Figure 4.3. Next we check if the cost of T'' with respect to criteria ℓ is better than $h^{\ell}(T)=H(T)$. If so, we attempt to find (in a way to be described below) a 3-neighbour T' of T such that $H(T')<H(T)$. If such a tour T' is found, we initialize T=T' and restart the procedure TOUR with the new T. Otherwise, we check if, in fact, $H(T'')<H(T)$. If so, we initialize T=T'' and restart the procedure TOUR with the new T. Otherwise we continue searching for another T'' by finding a new replacement edge for T_nT_1 which is better with respect to criteria ℓ, starting our search at T_{i+1}.

The search for T' is described as follows: we attempt to exchange the edge T_1T_{i+1} of T'' with another edge to obtain a tour T' which is a 3-neighbour of T and such that $H(T')\leq H(T)$. (The reason for

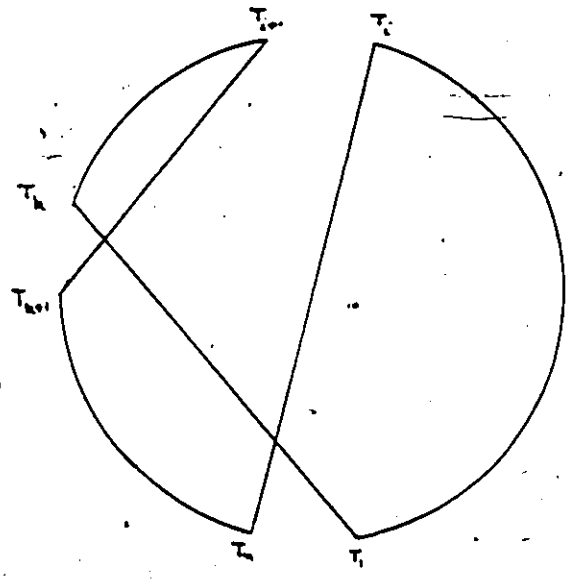

Figure 4.4: A 3-Neighbour T' of T

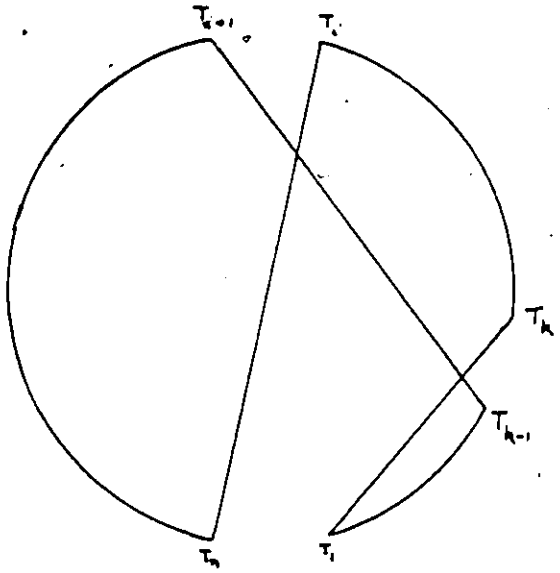

Figure 4.5: A 3-Neighbour T' of T

attempting to exchange the edge T_1T_{i+1} is that, whereas the edge T_nT_i was selected, the edge T_1T_{i+1} was forced on us so as to close the Tour T''). Starting with T'', the new tour T' is as shown in Figure 4.4 or Figure 4.5, depending upon whether we found a better edge T_1T_k to the left of vertex T_{i+1} or to the right of vertex T_i. We look for a replacement of the edge T_1T_{i+1} searching the edges both to the right of the vertex T_i and left of T_{i+1} and considering only those edges which are better than T_1T_{i+1} whith respect to criterion ℓ.

Clearly, there are numerous possibilities for further backtracking and more exhaustive local search, and the optimum can be found, in principle, by full backtracking at all levels. However, the execution time for such a procedure will be immense. As will be shown in the next chapter, by using tours obtained as a byproduct of the subgradient procedure as initial tours in the above heuristic, we obtained excellent results. Thus the limited backtracking we do seems an effective compromise between optimality and execution time of the algorithm.

CONSTRUCTION OF INITIAL TOURS

For step 1 of LOCAL OPTIMUM we tried

(1) randomly generated initial tours;

(2) tours obtained by transforming the 1-tree output from the subgradient procedure.

In the next chapter we will see that, starting with these initial tours and using neighbourhood search, we are able to achieve a

narrow gap between the value of the lower bound from the subgradient procedure and the upper bound from feasible tours. We found that (2) is much better than (1) w.r.t. time and bound quality.

We now consider (2) in more detail.

CONSTRUCTING TOURS FROM MINIMUM 1-TREES

As well as giving a lower bound on the solution of the MTSP, the subgradient procedure yields a minimum 1-tree as an approximation to the optimal tour. As the 1-trees obtained from the subgradient procedure had mostly nodes of degree 2 (more than 90%), and thus were very close to a tour, we developed a procedure to construct deterministically some starting tours from the 1-trees, which were then fed to the neighbourhood search heuristic as initial tours.

The main idea behind this procedure is to convert a given minimum 1-tree into a tour by replacing some edges of the 1-tree with new edges. To exploit the minimality of the 1-tree we refrain from disturbing nodes of degree 2 of the given 1-tree. The tours thus constructed serve as very good initial feasible tours for our neighbourhood search heuristic. Now we describe the procedure.

Let T be a minimum 1-tree. T is arranged as a set of n edges $\{(T(i,1), T(i,2)): i=1,2,\ldots,n\}$, where n is the size of the problem. For each node in T find the set of all the nodes in T which are adjacent to it. For a given node s this is done by scanning through edges $(T(i,1), T(i,2))$ sequentially $i=1,2,\ldots,n$. Any time s matches with a

node from an edge, the other node of the edge is recorded as an adjacent node of s. The adjacent nodes of s are recorded sequentially as they are found. We refer to these sets as initial neighbourhood sets of nodes. A neighbourhood set is a subset of an initial neighbourhood set. Starting at an odd degree node we build up our tour sequence, say TOUR, selecting one node at a time. The next node selected is chosen from the neighbourhood set of the most recently selected node whose neighbourhood set is nonempty. For the selection of the nodes from the neighbourhood set, nodes are tested in the same order as they were recorded in the initial neighbourhood set. Then the neighbourhood sets and the TOUR sequence are updated as follows: The new node selected is added to the TOUR sequence at its end and deleted from all the neighbourhood sets to which it belongs. Thus, if at some point the neighbourhood set of the last selected node is empty, we select the new node of TOUR from the neighbourhood set of the most recently selected node of TOUR having a nonempty neighbourhood set. The procedure terminates when TOUR sequence has covered all the nodes of T. At this point all the neighbourhood sets are empty.

Because of the special role played by node 1 in the definition of a 1-tree, we also generate a tour starting at node 1 (an even degree node). It may happen that some of the tours generated by the above procedure may repeat themselves. Note also that a sequence of degree 2 nodes in T will simply be appended to the TOUR by this procedure so that long paths in the tree remain intact in the corresponding tour. This will become clear when we go through the following example.

AN EXAMPLE ILLUSTRATING THE PROCEDURE FOR GENERATING TOURS FROM MINIMUM 1-TREES:

The following example illustrates the above procedure. Figure 4.6 gives the graph of a minimum 1-tree obtained from the Ascent procedure for n=30 and r=2.

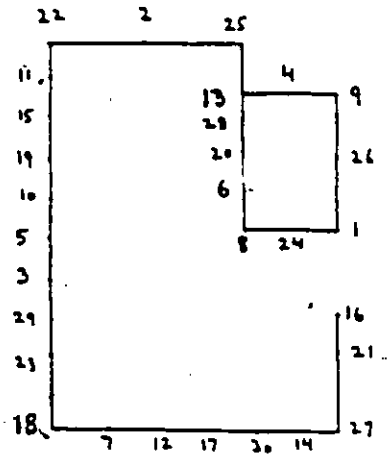

Figure 4.6: A Minimum 1-tree

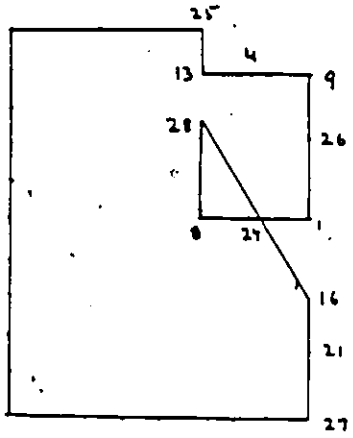

Figure 4.7: A Tour from a Minimum 1-tree

The odd degree nodes in the 1-tree of Figure 4.6 are 13 and 16. The neighbourhood sets of these nodes are respectively {25, 28, 4}

and {21}. The neighbourhood set of node 1 is {24, 26}. Every other node in the above 1-tree has degree 2 and has its neighbourhood set consisting of its two adjacent nodes. Figure 4.7 gives the TOUR constructed when we started with node 13. Starting at node 13, the next node selected from the neighbourhood set of 13 is node 25. The neighbourhood sets of node 13 and node 25 are updated to {28, 4} and {2} respectively, forcing the selection of node 2. The only choice from node 2 is node 22, and from there nodes 11, 15,...,27, 21 and 16 are similarly selected. Now the neighbourhood set of node 16 is empty. So we look at the preceding node 21, but its neighbourhood set is also empty. We thus move back through the nodes previously selected until we find one with a non-empty neighbourhood set. The first such node we encounter is node 13. The next node selected is node 28 and so on to node 4. The procedure terminates there because we have selected all 30 nodes and have completed the TOUR.

Next we find a TOUR starting at node 16. There is only one choice and that is node 21. The TOUR continues upto node 13. The neighbourhood set of node 13 is {28, 4}. Thus the next node selected is node 28 and so on upto node 4. At this point neighbourhood sets of all the nodes are empty and we have a TOUR as depicted is Figure 4.8. We can check easily that in this example the TOUR obtained by starting at node 1 is the same as the one obtained above by starting at node 16.

We observe that each of the above tours leave long chains of the 1-tree intact. The first tour is obtained by deleting the edge (13, 28) and including the edge (28, 16) in the 1-tree and the second tour by replacing the edge (13, 4) with the edge (4, 16). We note that node 13

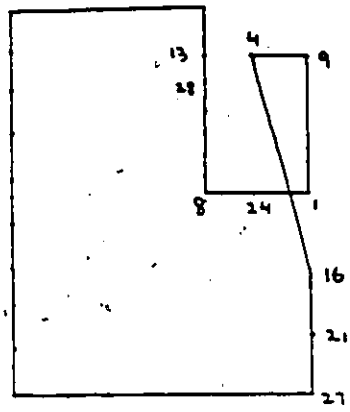

Figure 4.8: A Tour From a Minimum 1-tree

had degree 3 and node 16 had degree 1 in the minimum 1-tree. The above tours deleted an edge emanating at 13 and added an edge terminating at node 16 to convert the 1-tree into a tour.

An alternative source of initial feasible tours could be the use of a Greedy algorithm or some other simple heuristic. We do not consider such alternatives in this thesis.

This completes our presentation of the upper bound procedure for MTSP. In the next chaper we present and discuss experimental results on the performance of this procedure based on each of our approaches for selecting initial feasible tours.

CHAPTER V

COMPUTATIONAL RESULTS

OVERVIEW OF EXPERIMENTAL METHODOLOGY

In this chapter we will report our computational experience. We implemented the procedures described in this thesis on the University of Ottawa's AMDHAL/VM-7 computer using Pascal-8000 as the programming language.

We implemented 3 procedures: namely, the subgradient procedure, the neighbourhood search heuristic, and the method for generating tours from minimum 1-trees. Details of the procedures will appear in a technical report. We have tested our procedures on randomly generated problems (cf. appendix A). This involved generating one or more cost matrices, depending upon the number of criteria, and a number of randomly generated initial tours. In appendices B and C we give source listings of the random generators used for the above purposes. The random generator COSTMATRICES of appendix B is used to generate cost matrices which generates coordinates x_i and y_i, uniformly and independently, with values between 1 and 70 for i=1,2,...,n. It takes as input n, the number of cities, r, the number of criteria, and, for each criteria, a seed (a positive integer). The output from COSTMATRICES is a set of r symmetric cost matrices, each of dimension nxn. These cost matrices are used as input to both the subgradient and the heuristic procedure. We tested the heuristic using both randomly generated initial tours, and initial tours obtained from 1-trees (as described in Chapter IV). The

random tours used as input to the neighbourhood search heuristic are generated by the random generator RANDOMTOUR of appendix C which takes as input n, the number of cities and an integer value for the number of random tours required. We have tested our procedures on problems of up to 200 cities and 3 criteria (for the 200 city problem we considered only 2 criteria).

In the following tables, we report the average performance of our procedures on sets of 10 problems of different sizes (10-100 cities) using a different set of seeds for each problem. Because of the cost considerations we solved only one problem of size 200 and r = 2. We also describe the performance of our procedure on a given typical problem. In all tables and figures, 'Ascent' refers to the subgradient procedure.

It must be born in mind that the CPU time we report here was obtained by inserting the clock function, a built-in function in Pascal-8000, both at the beginning and at the end of the program to achieve an accurate measurement of the CPU time used. However, as this may vary depending on the load on the CPU, one has to be cautious in using the reported CPU time. To partially overcome this problem, we tested each problem of a given size on 10 different sets of seeds both for the subgradient procedure and the heuristic, ran it at different times of the day and then reported the average performance. The CPU time reported anywhere in this thesis is in units of 1/1000 of a second. In the discussion of the tables of results, we will specify the underlying assumptions made, so as to avoid any confusion.

EXAMPLES OF MINIMUM 1-TREES FROM THE SUBGRADIENT PROCEDURE:

As explained in Chapters II and III, the bound t^* of (2.3) calculated by the subgradient procedure provides good lower bounds to the solution of the MTSP. These lower bounds are in the form of minimum 1-trees. We have observed that these minimum 1-trees tend to be very close to being a tour. We give examples of three typical minimum 1-trees (Figures 5.1, 5.2, 5.3) which were obtained from the subgradient procedure. These 1-trees were obtained by implementing the subgradient procedure as described in Chapter III and running it on cost matrices generated by the random generator COSTMATRICES (appendix B) on the seeds as given in the figures.

In Figure 5.1 we have a problem of size 100 cities with 2 criteria and as we see the best minimum 1-tree from the subgradient procedure has 98 nodes of degree 2 and therefore is almost a tour. In Figure 5.2 the subgradient procedure actually produced a tour.

N = 100 r = 2 Seeds = (13, 84).

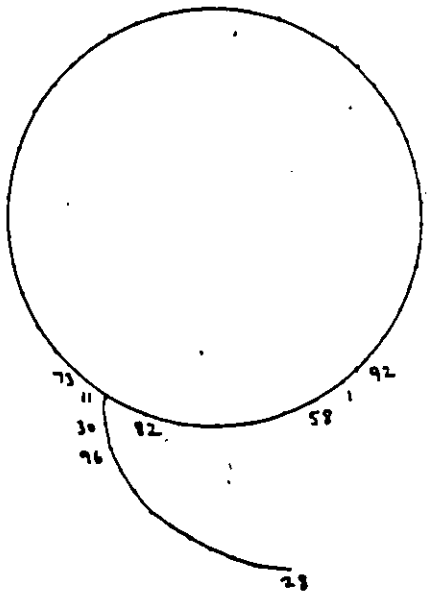

Figure 5.1: A Minimum 1-tree

Figure 5.3 gives the worst 1-tree we got from the subgradient procedure during our runs. Even in this example we still have 84% of nodes with degree 2.

N = 50 r = 2 Seeds = (47, 13)

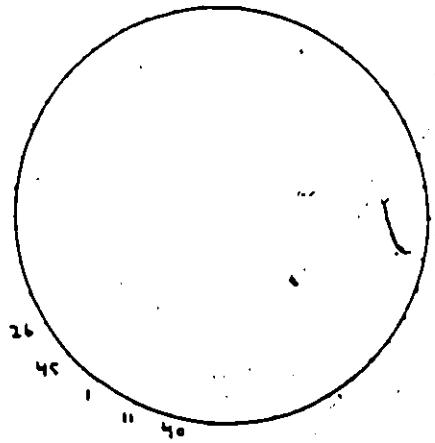

Figure 5.2: A Minimum 1-tree

N = 100 r = 2 Seeds = (47, 13)

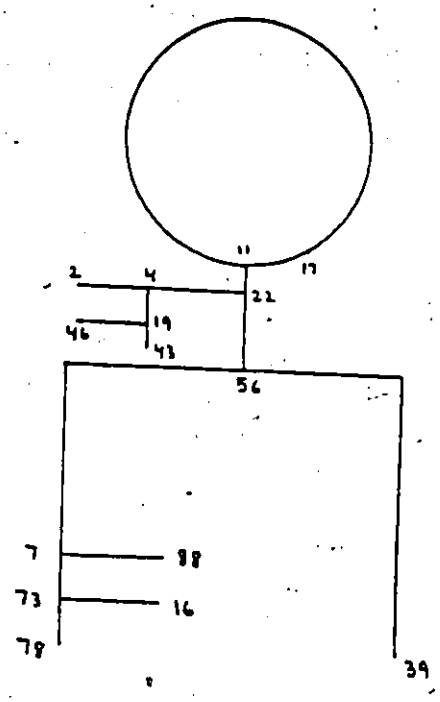

Figure 5.3: A Minimum 1-tree

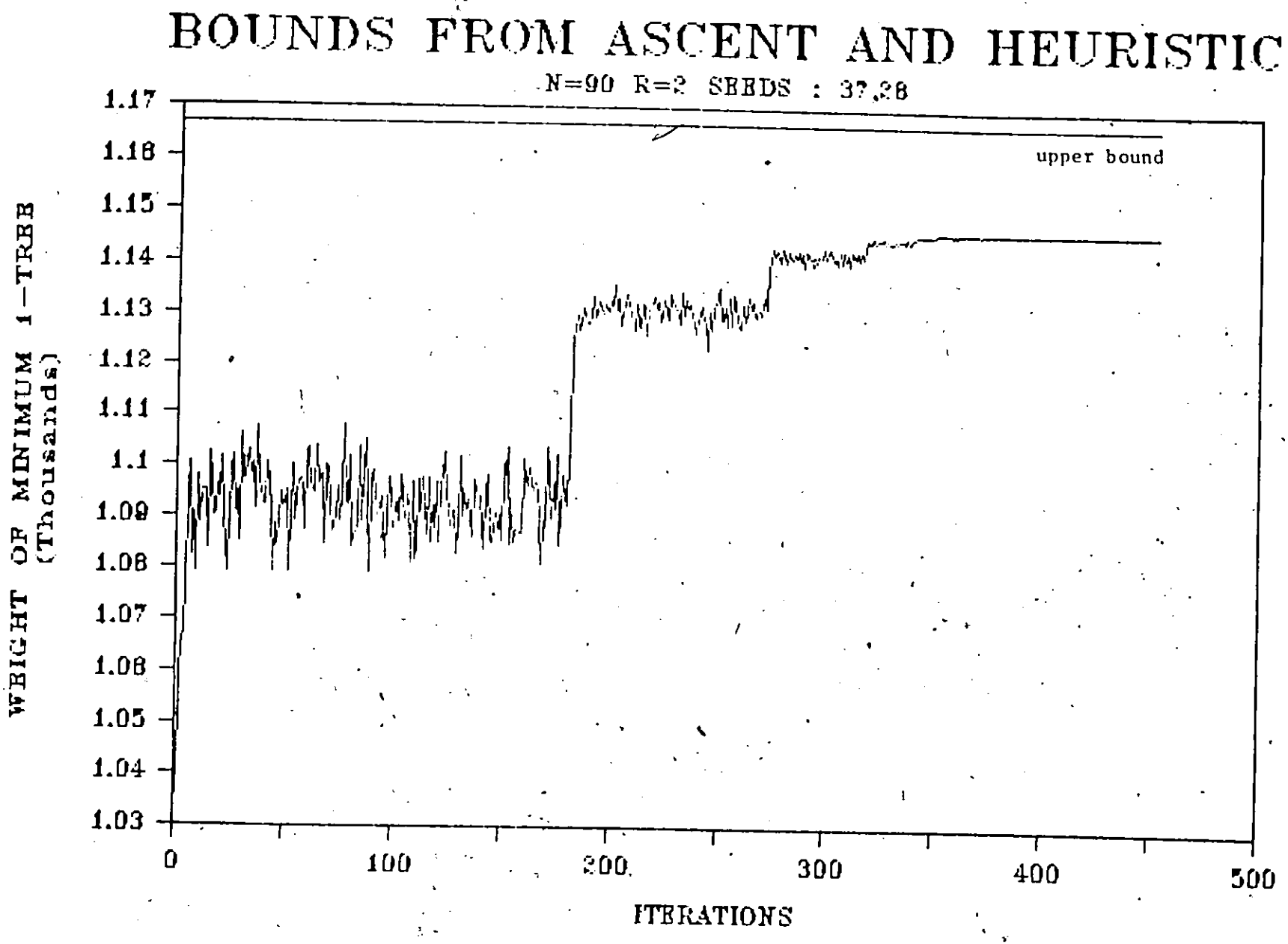
BOUNDS FROM ASCENT AND HEURISTIC
N=90 R=2 SEEDS : 37,38
upper bound
WEIGHT OF MINIMUM 1-TREE
(Thousands)
1.17
1.16
1.15
1.14
1.13
1.12
1.11
1.1
1.09
1.08
1.07
1.06
1.05
1.04
1.03
0
100
200
300
400
500
ITERATIONS

Figure 5.4

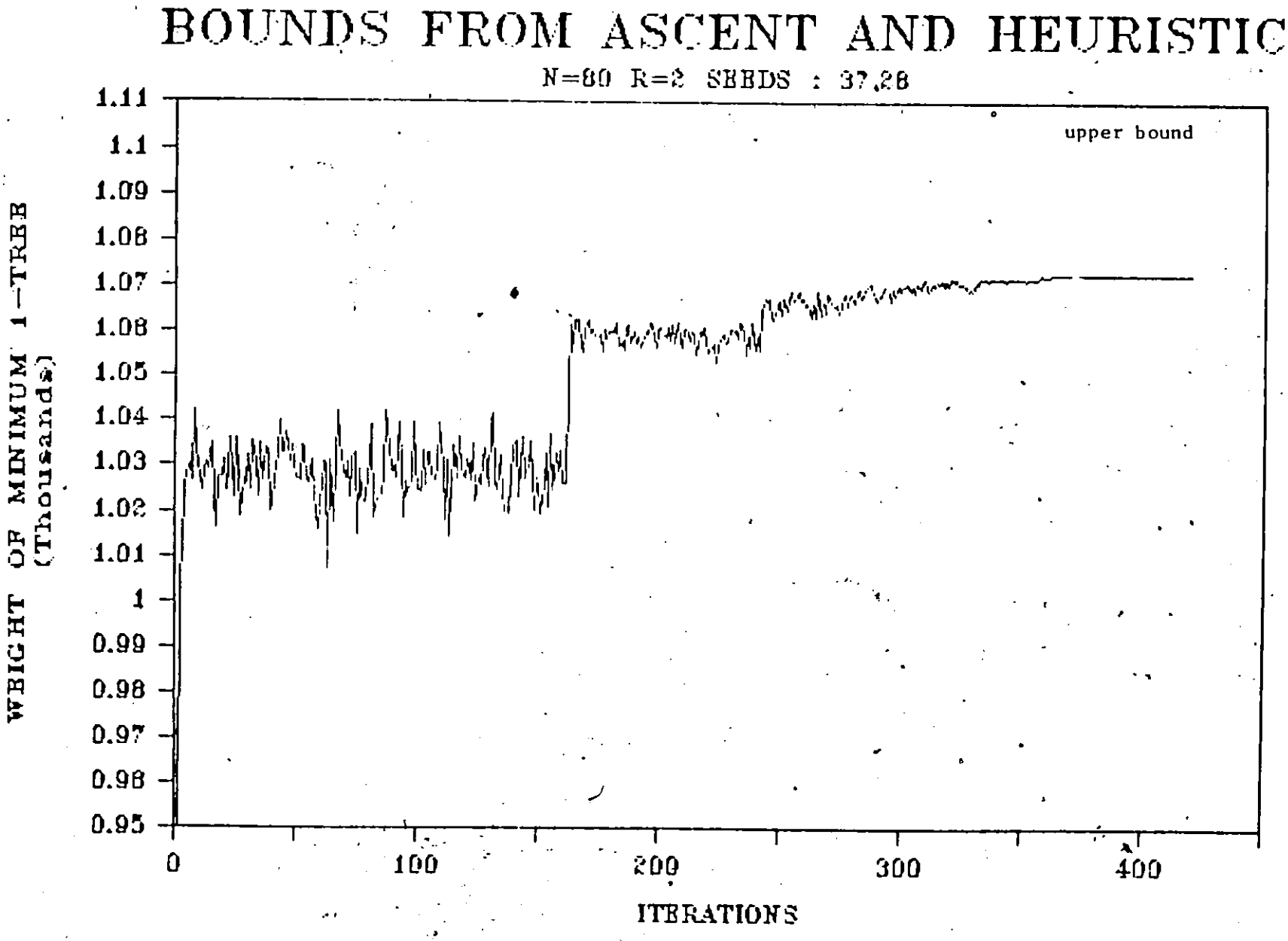
BOUNDS FROM ASCENT AND HEURISTIC
N=80 R=2 SEEDS : 37,28
upper bound
WEIGHT OF MINIMUM 1-TREE
(Thousands)
1.11
1.1
1.09
1.08
1.07
1.06
1.05
1.04
1.03
1.02
1.01
1
0.99
0.98
0.97
0.96
0.95
0
100
200
300
400
ITERATIONS

Figure 5.5

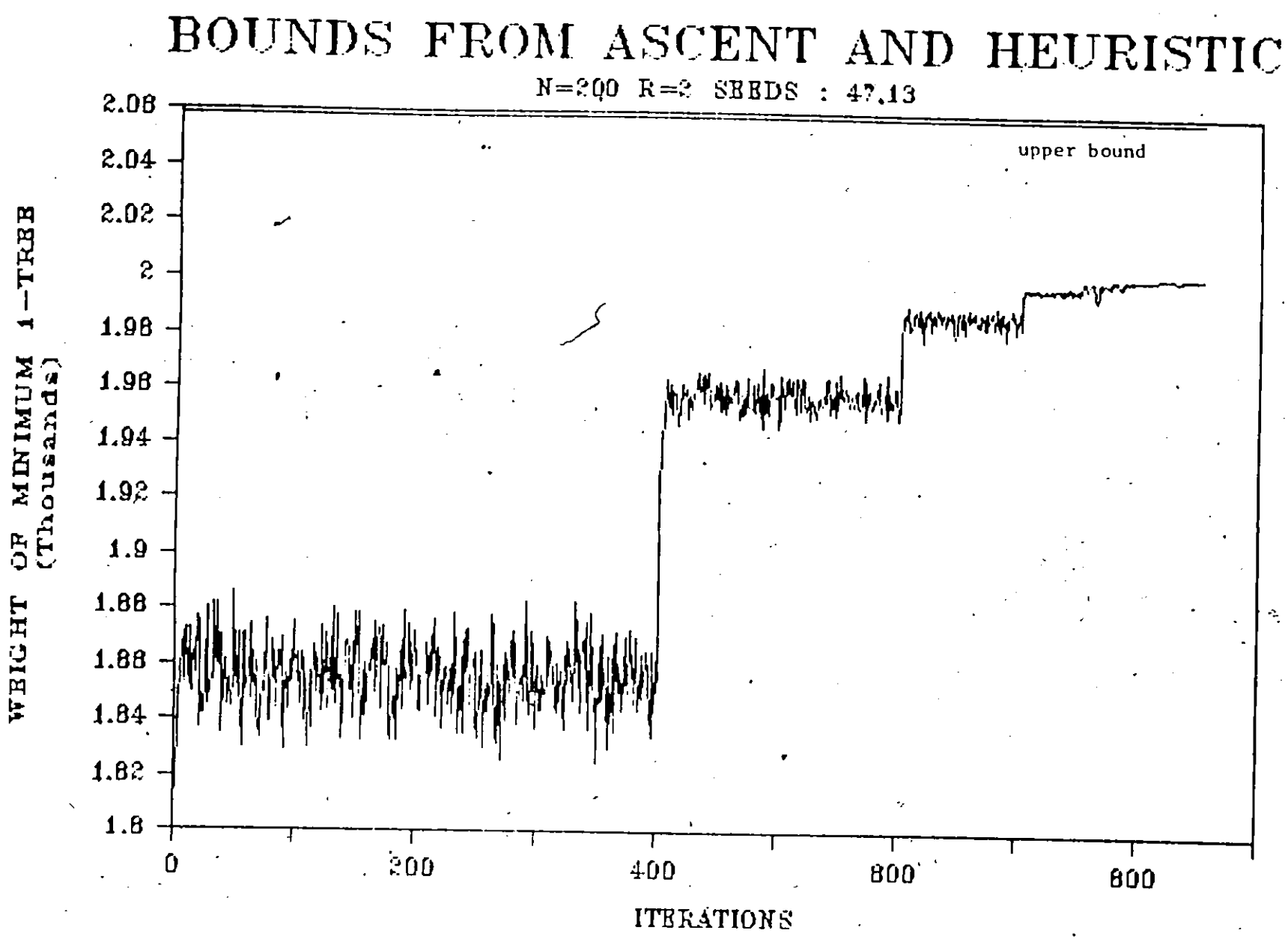
BOUNDS FROM ASCENT AND HEURISTIC
N=200 R=2 SEEDS : 47,13
upper bound
WEIGHT OF MINIMUM 1-TREE (Thousands)
2.08
2.04
2.02
2
1.98
1.96
1.94
1.92
1.9
1.88
1.86
1.84
1.82
1.8
0
200
400
600
800
ITERATIONS

Figure 5.6

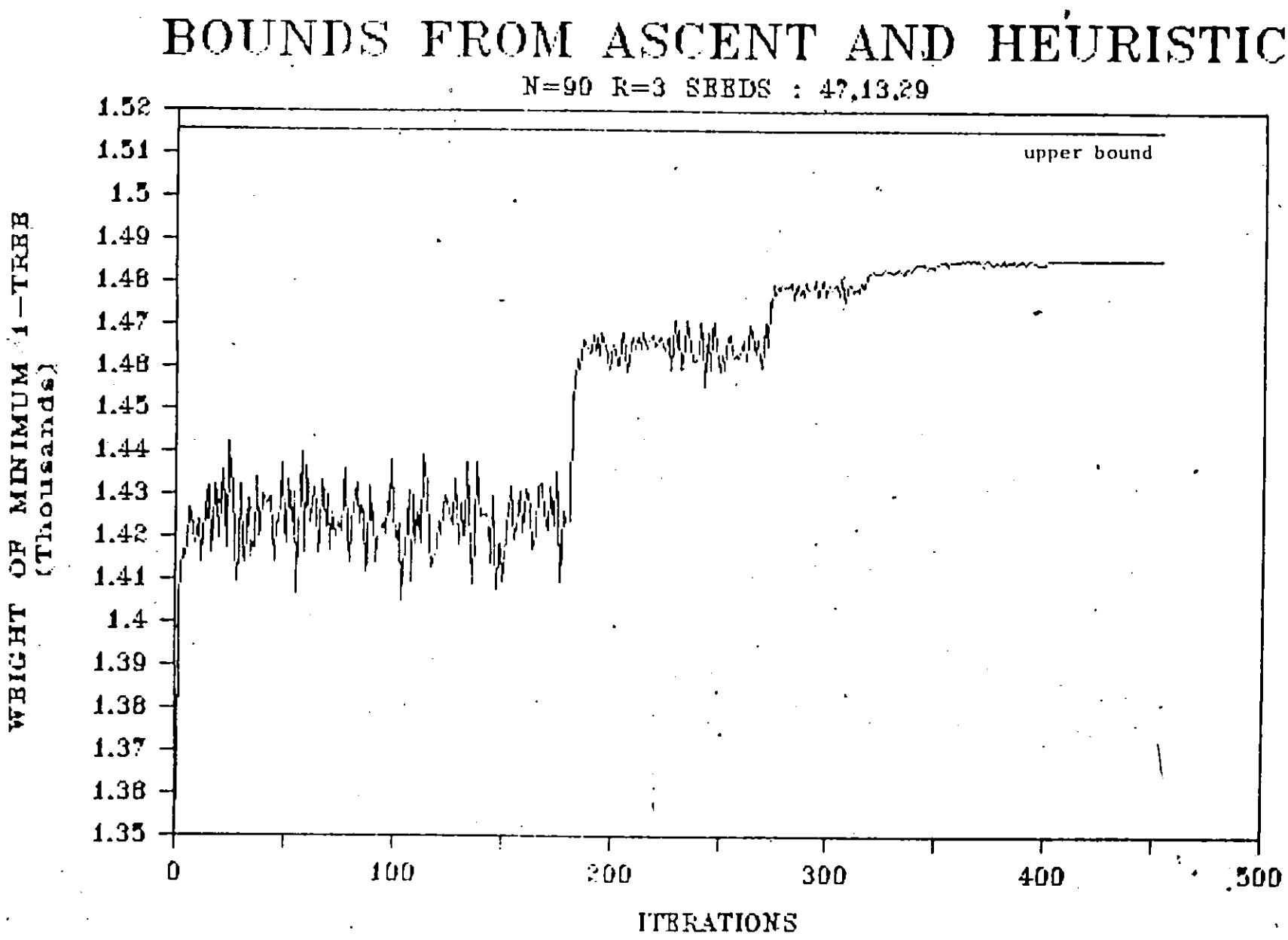
BOUNDS FROM ASCENT AND HEURISTIC
N=90 R=3 SEEDS : 47,13,29
upper bound
WEIGHT OF MINIMUM 1-TREE
(Thousands)
1.52
1.51
1.5
1.49
1.48
1.47
1.46
1.45
1.44
1.43
1.42
1.41
1.4
1.39
1.38
1.37
1.36
1.35
0
100
200
300
400
500
ITERATIONS

Figure 5.7

GRAPHICAL REPRESENTATION OF CONVERGENCE OF THE LOWER AND THE UPPER BOUNDS FOR THE MTSP:

Figures 5.4 through 5.7 give graphical representation of the progress of values $t(\lambda^k,\pi_k)$ through iterations of the subgradient procedure on 4 typical problems. These graphs show both the lower bounds from the subgradient procedure and the upper bounds from our neighbourhood search heuristic. As is evident from these graphs, the final gap between our lower bounds and the upper bounds is very small, thereby illustrating the strength of our procedures. These graphs indicate that perhaps convergence can be achieved even faster by a variation in the strategy for the choice of the step sizes.

EXPERIMENTAL RESULTS:

Appendix A gives Tables A-1 to A-10 obtained from running our procedures on 10 different sets of 10 problems each (10-100 cities). In these tables we also give the seeds that were used to generate a given problem, both for the cost matrices and for the random initial tours. The first column in these tables represents the size of the problem. The second column contains results of running the subgradient procedure and shows, consecutively, the cost of a minimum 1-tree, the CPU time used and the number of iterations needed. Each of column 3 and 4 reports consecutively, the cost of an upper bound, the CPU time used and the number of initial feasible tours used with the heuristic. In column 3 the number of initial tours is as generated from a minimum 1-tree obtained from the subgradient procedure, whereas in column 4 it is the number of random tours used. The CPU time reported in column 2 is the time taken

Table 5.1: Comparison of costs: Ascent vs Heuristic applied to random tours.

Seeds : Costs (47, 13) Random Tours 53

# of Cities	ASCENT Cost LB	Heuristic Random Tours Cost UB	Ratio (UB - LB)/LB
10	255	262	0.027
20	422	434	0.027
30	547	572	0.045
40	639	664	0.038
50	759	796	0.048
60	870	925	0.063
70	971	1018	0.048
80	1084	1125	0.038
90	1187	1259	0.061
100	1197	1287	0.075
200	2002	2148	0.073

Table 5.2: Comparison of costs: Heuristic as applied to tours from Ascent vs. Heuristic applied to random tours.

Seeds : Costs (47, 13) Random Tours 53

# of Cities	Heuristic Ascent Tours Cost	Heuristic Random Tours Cost
10	262	262
20	425	434
30	562	572
40	646	664
50	764	796
60	895	925
70	985	1018
80	1105	1125
90	1207	1259
100	1233	1287
200	2059	2148

Table 5.3: Comparison of optimality of tours from Ascent vs Random tours.
Ratio = (UB - LB)/LB

Seeds : Costs (47, 13) Random Tours 53

# of Cities	Tours from Ascent Ratio	Random tours Ratio
10	0.027	0.027
20	0.006	0.027
30	0.027	0.045
40	0.010	0.038
50	0.006	0.048
60	0.029	0.063
70	0.014	0.048
80	0.019	0.038
90	0.017	0.061
100	0.030	0.075
200	0.029	0.073

Table 5.4: Comparison of average costs as obtained from Heuristic on tours from Ascent vs Heuristic on random tours.

# of Cities	LB from Ascent	Ascent + Heuristic Cost	Heuristic on Random tours Cost	Average (UB - LB)/LB Ascent + Heuristic	Average (UB - LB)/LB Random Tours
10	255	264	263	0.032	0.030
20	398	406	410	0.018	0.029
30	536	550	561	0.025	0.046
40	657	669	689	0.018	0.048
50	764	780	804	0.020	0.052
60	872	889	913	0.020	0.047
70	954	971	1001	0.018	0.049
80	1064	1087	1125	0.022	0.058
90	1137	1161	1205	0.021	0.060
100	1231	1250	1292	0.015	0.061

by the subgradient procedure, and the time in columns 3 and 4 is the CPU time taken by the heuristic on the number of tours reported in respective columns. In all cases we used 10 random tours. For larger problems ($\geq$ 60 cities) it is not wise to use many random tours as initial feasible tours for the heuristic. This strategy is very expensive in terms of the CPU time and does not seem to provide much improvement in upper bounds.

DISCUSSION OF EXPERIMENTAL RESULTS:

Tables 5.1,5.2, 5.3, 5.5, 5.7 and 5.8 report the performance of our procedures on a selected set of problems and Tables 5.4, 5.6, 5.9, 5.10 and 5.11 give the average performance of our procedures. In particular Table 5.1 compares the lower bound obtained from the subgradient procedure with the cost of a nearly optimal tour obtained by applying the heuristic procedure to random tours. The last column shows that the relative difference between the upper and the lower bounds vary between 2.7% to 7.5% for problems sizes upto 200 cities. In Table 5.3 we compute the same ratio using upper bounds obtained from the heuristic when applied on tours from minimum 1-trees (reported in Table 5.2), and compare it with the corresponding ratios from Table 5.1. The superiority of the 1-tree approach is clear. Table 5.4 gives averages of corresponding best ratios on average. These are obtained by finding the individual ratios and then taking the average. The variation over here of 1.5% to 3.2% indicates that our solutions are very close to the optimal solutions. Table 5.4 also reports the same results for absolute numbers instead of ratios. These numbers were obtained by computing the average cost over 10 problems for each of the three columns.

Table 5.5: Performance of Ascent + Heuristic on tours from Ascent vs Heuristic on Random tours.
Note: CPU time is in 1/1000 of a second.

Seeds : Costs (47, 13) Random Tours 53

# of Cities	Ascent + Heuristic CPU time	Heuristic on 10 Random tours CPU time
10	924	262
20	4033	3194
30	10358	11808
40	24315	29700
50	34703	60038
60	94403	113765
70	122752	242256
80	145167	371199
90	274634	523940
100	596924	725248
200*	4234804	9023218

Table 5.6: Average performance of Heuristic on tours from Ascent vs Heuristic on random tours.
Note: The CPU time is in 1/1000 of a second.

# of Cities	Ascent + Heuristic CPU time	Heuristic on 10 random tours CPU time
10	847	302
20	3864	2878
30	11025	11759
40	26877	30281
50	51401	64074
60	84446	126362
70	122726	226881
80	199630	357072
90	267991	545816
100	374621	738036

* Note that only one problem of size 200 was run.

Table 5.7: Closeness of the minimum 1-tree generated by Ascent to a tour (Costs Seeds (37, 28) and r=2)

# of Cities	# of different degree nodes in 1-tree (1, 2, 3, 4, >4)
10	1, 8, 1
20	0, 20, 0
30	1, 28, 1
40	4, 32, 4
50	3, 44, 3
60	1, 58, 1
70	1, 69, 1
80	6, 68, 6
90	4, 82, 4
100	3, 94, 3

Table 5.8: Closeness of the minimum 1-tree generated by Ascent to a tour (Costs seeds (47, 13, 29) and r=3)

# of Cities	# of different degree nodes in 1-tree (1, 2, 3, 4, >4)
10	1, 8, 1
20	2, 16, 2
30	2, 26, 2
40	3, 34, 3
50	3, 44, 3
60	0, 60, 0
70	3, 64, 3
80	4, 72, 4
90	5, 80, 5
100	4, 92, 4

Table 5.9: Average number of tours generated by Ascent.

# of Cities	Average # of tours generated by Ascent
10	3
20	3
30	4
40	6
50	7
60	7
70	7
80	8
90	8
100	8
200*	15

* Note that only one problem of size 200 was run.

In Table 5.5 we report the actual CPU time used for both the subgradient procedure and the heuristic on tours from the subgradient procedure vs. the CPU time for the heuristic alone on 10 random tours. As the problem size increases ($\geq$ 30) the combined time of the subgradient procedure and the heuristic on tours from the subgradient procedure tends to be better than the heuristic on random tours. For very large problems the difference becomes very evident. When we look at the averages of these results (Table 5.6), the conclusions are similar. This suggests that a good strategy for solving the MTSP is to find a minimum 1-tree using the subgradient procedure and then generate tours from the 1-tree which are then input as initial feasible tours to the heuristic.

Tables 5.7 and 5.8 give us the degree structure of the minimum 1-trees from the subgradient procedure with 2 and 3 criteria respectively. Both these tables show that the minimum 1-trees obtained from the subgradient procedure are indeed very close to being a tour. This claim is supported by Table 5.9 where we give the average number of tours generated from the minimum 1-trees from the subgradient procedure. Recall that we generate one tour for each odd degree node in a minimum 1-tree and also an additional tour starting at node 1. Note further that in all these problems we obtained minimum 1-trees with nodes of degree 1, 2 and 3 only. In addition the number of nodes of degree 1 equalled number of degree 3 nodes. Therefore in the 10 city case, of the 3 tours generated one started from node 1, one started from a node of degree 1 and one from a node of degree 3. Generalizing these arguments, we observe, for example, that on average there are 83 nodes of degree 2 for a problem of size 90 (note that because of averaging we are getting an odd number of nodes of degree 2).

Table 5.10: Average performance of Ascent on a set of 10 problems of each size (10-100 cities).
Note: The CPU time is in 1/1000 of a second.

# of Cities	Average # of iter.	Average CPU time/iter
10	142	6
20	172	20
30	213	45
40	247	79
50	295	123
60	333	178
70	361	242
80	406	315
90	458	384
100	533	480
200*	874	1927

* Note that only one problem of size 200 was run.

Table 5.11: Average performance of Heuristic on set of 10 problems of each size (10-100 cities), on Random tours vs tours from Ascent.
Note: The CPU time is in 1/1000 of a second.

# of Cities	Random tours CPU time/tour	Ascent tours CPU time/tour
10	30	18
20	288	120
30	1176	371
40	3028	1027
50	6407	1944
60	12636	3416
70	22688	5131
80	35707	8451
90	54582	11535
100	73803	13593
200*	902323	170035

* Note that only one problem of size 200 was run.

In Table 5.10 we give the average number of iterations for the subgradient procedure along with the CPU time for one iteration for each problem size. Note that the average number of iterations for convergence of the subgradient procedure grows with the problem size and so does the CPU time taken per iteration of the Ascent. Whereas the CPU time for an iteration on a problem of size 10 is 6/1000 of a second, it takes 80 times more CPU time to complete an iteration on a problem of size 100 and 320 times more to go through an iteration on a 200 city problem. This shows the increase in complexity of the problem as we increase the number of cities.

In the last table (Table 5.11) we compare the performance of our neighbourhood search heuristic on random tours vs tours generated from the subgradient procedure. The CPU time reported here is the average time taken by the heuristic to find a locally optimal tour starting from an initial feasible tour. In column 2 initial tours were randomly generated whereas in the last column the initial tours were deterministically generated from the minimum 1-trees obtained from the subgradient procedure. Note that uniformly it takes less time for the heuristic to process a tour generated from the subgradient procedure than a random tour. This happens because the minimum 1-trees obtained from the subgradient procedure are very close to being a tour and the tours we generate from the minimum 1-trees keep most of the edges of the minimum 1-tree intact. This table shows that on the average, the heuristic takes 2 to 5 times more of the CPU time to find a locally optimal tour starting from a random tour than on a tour from the subgradient procedure (on problems ranging from 10 to 100 cities). This suggests that we start with tours from the subgradient procedure as initial feasible tours.

Thus, based on our experimentation, we conclude that our approach of starting with tours for the subgradient procedure yields substantial improvements over the random initial tour approach.

CHAPTER VI

CONCLUSIONS AND SUGGESTIONS FOR FURTHER RESEARCH

In this thesis we developed a bounding procedure, which gives good lower bounds to the solution to the MTSP, and modified a popular neighbourhood search heuristic to obtain good feasible solutions to the MTSP. Minimum 1-trees from the bounding procedure were used to generate tours which were in turn used to initialize our heuristic. This method seems to be an improvement over using the neighbourhood search method which begins with randomly generaled tours. In fact, our empirical results strongly suggest that the 1-tree approach leads to savings in CPU time and increased accuracy. Additionally, it provides a lower bound against which we can compare our heuristic tour.

There is scope for improvement in our heuristic procedure. In particular, we could consider a heuristic with a wider search base and more backtracking. There is clearly a trade-off in implementing a more exhaustive search heuristic; the increased running time of such a heuristic may outweigh the improvement in the upper bound.

We would also like to investigate the performance of the neighbourhood search heuristic when initial tours are constructed by a greedy or other simple algorithm. Our bounds could also be used in a branch and bound procedure to obtain the exact optimum solution to the MTSP.

Finally, our graphs indicate that further experimentation with step sizes may be profitable to increase convergence rate for the subgradient procedure itself.

APPENDIX A

Table A-1

Seeds: Costs (5, 51)
Random Tours 13
r = 2 .

Note: CPU time is in 1/1,000 of a second

# of Cities	Ascent Cost CPU Time Iterations	Heuristic Ascent Tours Cost CPU Time # of Tours	Random Tours Cost CPU Time # of Tours
10	243.2 856 151	252 34 3	252 309 10
20	404.6 3,691 181	409 617 3	417 2,620 10
30	526.4 9,623 213	534 2,241 5	554 13,619 10
40	678.0 19,179 241	695 4,791 5	709 30,825 10
50	767.3 33,885 274	787 17,132 7	805 65,666 10
60	850.7 59,422 333	875 40,155 9	886 115,148 10
70	963.1 87,441 361	976 37,455 9	1,005 210,672 10
80	994.7 123,895 391	1,015 56,328 7	1,061 344,031 10
90	1,110.4 166,501 424	1,144 60,343 7	1,181 551,934 10
100	1,196.2 232,026 484	1,220 91,611 7	1,286 770,056 10

Table A-2

Seeds: Costs (13, 84)
Random Tours 21
r = 2

Note: CPU time is in 1/1,000 of a second

# of Cities	Ascent Cost CPU Time Iterations	Heuristic Ascent Tours Cost CPU Time # of Tours	Random Tours Cost CPU Time # of Tours
10	272.8 851 151	275 82 3	274 389 10
20	404.5 3,703 181	412 430 3	420 3,601 10
30	515.1 9,602 213	534 811 3	534 12,436 10
40	674.4 21,530 271	687 28,773 15	697 34,191 10
50	784.5 33,903 274	806 23,075 9	829 76,500 10
60	850.0 59,458 333	881 37,395 9	909 108,726 10
70	954.1 87,922 361	976 37,455 9	988 275,446 10
80	1,092.5 123,417 391	1,121 60,671 7	1,167 381,656 10
90	1,146.6 177,499 454	1,179 162,612 13	1,217 628,622 10
100	1,233.0 232,042 484	1,243 48,381 5	1,317 722,979 10

Table A-3

Seeds: Costs (29, 51)
Random Tours 35
r = 1

Note: CPU time is in 1/1,000 of a second

# of Cities	Ascent Cost CPU Time Iterations	Heuristic Ascent Tours Cost CPU Time # of Tours	Heuristic Random Tours Cost CPU Time # of Tours
10	267.4 851 151	286 52 3	288 252 10
20	376.0 3,699 181	385 447 3	396 2,448 10
30	551.3 9,597 213	565 1,958 5	578 10,952 10
40	634.6 19,160 241	640 3,917 5	662 29,177 10
50	747.2 37,591 304	765 28,767 11	785 85,144 10
60	904.4 59,882 333	919 27,473 7	934 132,535 10
70	934.9 88,384 361	956 58,780 9	994 204,304 10
80	1,033.9 132,954 421	1057 26,437 5	1118 345,152 10
90	1,086.5 176,353 454	1,106 45,477 5	1,141 481,621 10
100	1,248.6 231,860 484	1,258 43,455 5	1,307 742,737 10

Table A-4

Seeds: Costs (37, 28)
Random Tours 45
r = 2

Note: CPU time is in 1/1,000 of a second

# of Cities	Ascent Cost CPU Time Iterations	Heuristic Ascent Tours Cost CPU Time # of Tours	Random Tours Cost CPU Time # of Tours
10	237.2 831 151	248 54 3	248 293 10
20	381.4 2,420 121	382 71 1	382 2,924 10
30	519.0 9,364 213	532 651 3	545 11,636 10
40	674.0 18,713 241	685 8,354 9	706 28,283 10
50	814.4 36,695 304	834 14,355 7	858 64,754 10
60	860.1 57,694 333	877 12,551 3	900 115,237 10
70	963.8 92,096 391	982 12,181 3	1,010 216,450 10
80	1,072.5 129,358 421	1,110 128,876 13	1,133 324,838 10
90	1,146.6 176,423 454	1,167 75,993 9	1,198 514,192 10
100	1,235.3 231,971 484	1,258 121,877 7	1,325 652,767 10

Table A-5

Seeds: Costs (47, 13)
Random Tours 53
r = 2

Note: CPU time is in 1/1,000 of a second

# of Cities	Ascent	Heuristic	
		Ascent Tours	Random Tours
	Cost	Cost	Cost
	CPU Time	CPU Time	CPU Time
	Iterations	# of Tours	# of Tours
10	255.1	262	262
	859	65	347
	151	3	10
20	422.4	425	434
	3,694	339	3,194
	181	3	10
30	547.2	562	572
	9,585	773	11,808
	213	3	10
40	639.4	646	664
	21,541	2,774	29,700
	271	3	10
50	759.2	764	796
	33,906	797	60,038
	274	1	10
60	870.0	895	925
	59,373	35,030	113,765
	333	9	10
70	971.0	985	1,018
	87,210	35,542	242,256
	361	7	10
80	1,084.1	1,105	1,125
	123,988	21,179	371,199
	391	5	10
90	1,187.0	1,207	1,259
	177,260	97,374	523,940
	454	7	10
100	1,197.0	1,233.0	1,287
	232,109	364,815	725,248
	484	15	10
200	2,001.7	2,059	2,148
	1,684,268	2,550,536	9,023,218
	874	15	10

Table A-6

Seeds: Costs (13, 29)
Random Tours 18
r = 2

Note: CPU time is in 1/1,000 of a second

# of Cities	Ascent Cost CPU Time Iterations	Heuristic Ascent Tours Cost CPU Time # of Tours	Heuristic Random Tours Cost CPU Time # of Tours
10	250.4 851 151	258 50 3	258 265 10
20	367.5 3,703 181	375 389 3	381 2,458 10
30	557.1 9,610 213	575 1,298 3	577 11,249 10
40	654.6 19,197 241	662 3,378 5	688 32,754 10
50	773.7 37,606 304	787 11,383 7	813 75,175 10
60	866.7 59,454 333	874 11,046 5	891 139,180 10
70	904.6 79,944 331	916 39,723 7	952 233,885 10
80	1,051.5 132,671 421	1,060 61,648 7	1,096 331,419 10
90	1,107.4 177,287 454	1,117.0 87,706 7	1,180 562,235 10
100	1,217.2 218,327 454	1,232 34,808 3	1,285 678,270 10

Table A-7

Seeds: Costs (7, 38)
Random Tours 15
r = 2

Note: CPU time is in 1/1,000 of a second

# of Cities	Ascent Cost CPU Time Iterations	Heuristic Ascent Tours Cost CPU Time # of Tours	Heuristic Random Tours Cost CPU Time # of Tours
10	225.0 512 91	228 17 1	228 357 10
20	390.0 3,078 151	391 73 1	396 2,498 10
30	536.6 9,597 213	551 1,515 3	555 11,160 10
40	673.1 19,197 241	688 8,352 7	707 33,252 10
50	750.6 37,569 304	753 3,058 3	788 60,099 10
60	892.5 59,401 333	900 18,947 5	930 148,902 10
70	939.5 87,211 361	958 41,433 7	986 213,517 10
80	1,086.5 123,173 391	1,109 71,394 7	1,150 334,864 10
90	1,090.8 177,324 454	1,120 59,762 5	1,154 569,697 10
100	1,237.0 233,449 484	1,251 60,234 5	1,316 794,266 10

Table A-8

Seeds: Costs (12, 84)
Random Tours 20
r = 2

Note: CPU time is in 1/1,000 of a second

# of Cities	Ascent Cost CPU Time Iterations	Heuristic Ascent Tours Cost CPU Time # of Tours	Random Tours Cost CPU Time # of Tours
10	267.2 855 151	276 41 3	276 263 10
20	413.3 3,696 181	431 654 5	418 2,798 10
30	530.0 9,589 213	546 1,123 3	559 13,779 10
40	651.4 19,234 241	664 3,069 3	695 27,955 10
50	695.0 37,611 304	716 9,035 7	736 63,046 10
60	888.3 59,466 333	909 33,940 11	910 118,397 10
70	985.7 88,325 361	997 30,697 5	1,025 260,986 10
80	1,076.3 132,581 421	1,105 91,930 11	1,133 408,452 10
90	1,136.5 177,767 454	1,164 102,553 9	1,218 591,177 10
100	1,243.8 233,993 484	1,259 208,172 13	1,309 774,604 10

Table A-9

Seeds: Costs (26, 51)
Random Tours 34
r = 2

Note: CPU time is in 1/1,000 of a second

# of Cities	Ascent Cost CPU Time Iterations	Heuristic Ascent Tours Cost CPU Time # of Tours	Heuristic Random Tours Cost CPU Time # of Tours
10	269.5 681 121	276 17 1	273 288 10
20	422.4 3,702 181	435 327 3	441 3,202 10
30	551.8 9,647 213	563 2,104 5	571 11,762 10
40	673.7 19,170 241	691 5,095 5	708 29,232 10
50	805.4 37,626 304	833 31,698 11	848 58,424 10
60	900.5 59,454 333	910 5,482 3	948 130,491 10
70	943.9 87,672 361	959 16,848 5	986 204,229 10
80	1,092.3 123,271 391	1,106 54,178 7	1,160 384,286 10
90	1,181.6 177,970 459	1,219 188,867 11	1,246 568,442 10
100	1,319.3 480,555 1,004	1,346 155,871 11	1,386 812,732 10

Table A-10

Seeds: Costs (10, 35)
Random Tours 19
r = 2

Note: CPU time is in 1/1,000 of a second

# of Cities	Ascent Cost CPU Time Iterations	Heuristic Ascent Tours Cost CPU Time # of Tours	Random Tours Cost CPU Time # of Tours
10	267.0 855 151	275 53 3	273 314 10
20	400.8 3,681 181	410 229 3	413 3,040 10
30	528.3 9,605 213	535 1,956 5	562 9,173 10
40	615.4 19,138 241	627 4,212 5	650 27,441 10
50	744.3 37,583 304	753 10,731 5	779 61,889 10
60	832.8 59,870 333	847 28,966 11	893 141,234 10
70	979.6 87,427 361	1,004 43,511 7	1,041 207,063 10
80	1,053.7 132,570 421	1,081.0 145,783 11	1,107.0 344,824 10
90	1,174.7 166,201 424	1,189 48,639 5	1,253 466,298 10
100	1,186.0 231,769 484	1,199 58,881 5	1,262 706,696 10

APPENDIX - B

```
PROGRAM COSTMATRICES ( INPUT, OUTPUT );
TYPE
  ALPHA = ARRAY (.1..200.) OF INTEGER;
  VAR
    B,X,Y : ALPHA;
    COST  : ARRAY(.1..200,1..200,1..3.) OF INTEGER;
    DIST  : ARRAY (.1..200,1..200.) OF INTEGER;
    SEED,NCITY,MODULUS,I,J,C,R : INTEGER;
  PROCEDURE RG ( VAR SEED, NCITY : INTEGER;
                 VAR A : ALPHA);
   CONST
    MULTIPLIER = 25173;
    INCREMENT  = 13849;
    VAR
      I,J : INTEGER;
    FUNCTION RANDOM (VAR SEED : INTEGER) : INTEGER;
      BEGIN (* RANDOM *)
        SEED   := ( MULTIPLIER * SEED + INCREMENT ) MOD 65536
        RANDOM := SEED;
      END; (* RANDOM *)
    BEGIN (* RG *)
      FOR I := 1 TO NCITY DO
        BEGIN
          J := RANDOM (SEED);
          A(.I.) := J
        END
  END; (* RG *)
  BEGIN (* PROGRAM *)
    READLN (NCITY,R);
    C := 1;
    WHILE ( C <= R) DO
      BEGIN (* WHILE *)
        READ(SEED);
        RG(SEED,NCITY,B);
        FOR I := 1 TO NCITY DO
         X(.I.) := B(.I.) MOD 71;
        RG(SEED,NCITY,B);
        FOR I := 1 TO NCITY DO
         Y(.I.) := B(.I.) MOD 71;
        FOR I := 1 TO NCITY DO
          BEGIN
            FOR J := I + 1 TO NCITY DO
             BEGIN
               DIST(.I.,J.) := TRUNC(SQRT(SQR((X(.I.)-X(.J.))+SQR(Y(.I.)
                                -Y(.J ))) + 0.5);
               COST(.I.J.C.) := DIST(.I,J.);
               COST (.J,I,C.) := COST(.I,J,C.)
             END
          END;
       C := C + 1
```

```
  END;(* WHILE *)
FOR J := 1 TO R DO
  BEGIN
    FOR I := 1 TO NCITY DO
      COST(.I,I,J.) := 0
  END ;
  C := 1;
  WHILE ( C <= R    ) DO
    BEGIN
      FOR I := 1 TO NCITY DO
        BEGIN
          FOR J := 1 TO NCITY DO
            WRITE ( COST(.I,J,C.) ) ;
            WRITELN
        END;
        WRITELN ;
      C := C + 1
    END
END. (* PROGRAM *)
```

APPENDIX C

```
PROGRAM RANDOMTOUR ( INPUT,OUTPUT );
TYPE
  ALPHA = ARRAY (.1..200.) OF INTEGER;
  VAR
    A                                   : ALPHA  ;
    SEED , NCITY  ,J, C,  I, TRIAL : INTEGER;
  PROCEDURE RT ( VAR SEED : INTEGER ; VAR A : ALPHA );
   CONST
    MULTIPLIER = 25173
    INCREMENT  = 13849
    VAR
      I,J,FLAG,X  :  INTEGER;
  FUNCTION RANDOM (VAR SEED : INTEGER ) ; INTEGER;
     BEGIN (* RANDOM *)
       RANDOM := SEED;
       SEED   := ( MULTIPLIER * SEED + INCREMENT ) MOD 65536
     END; (* RANDOM *)
   BEGIN (* RT *)
     I := 0;
     WHILE ( I <  NCITY ) DO
       BEGIN (* WHILE *)
         X := RANDOM(SEED) MOD NCITY  + 1;
         FLAG := 0;
         FOR J := 1 TO I DO
             IF X = A(.J.)  THEN FLAG := 1;
         IF FLAG = 0 THEN
           BEGIN
             I := I + 1;
             A(.I.) := X
           END
       END (* WHILE *)
  END;  (* RT *)
BEGIN (* RTOUR *)
   READ (NCITY, TRIAL, SEED);
   C := 1;
   WHILE ( C <= TRIAL ) DO
      BEGIN (* WHILE *)
        RT ( SEED , A);
        FOR J := 1 TO NCITY DO
          WRITE( A(.J.) );
        WRITELN;
        C := C + 1
      END (* WHILE *)
END. (* RANDOMTOUR *)
```

REFERENCES

[1] M. Bellmore and G.L. Nemhauser. The Travelling Salesman Problem: A Survey, Operations Research 16 (1968), pp 538-558.

[2] V.J. Bowman. On the Relationship of the Tchebycheff Norm and the Efficient Frontier of Multiple-Criteria Objectives, in Multiple-Criteria Decision Making, S. Zionts and H. Thiriez, eds., Lecture Notes in Economics and Mathematical Systems 130, Springer-Verlag, Berlin, Heidelberg, New York, 1975.

[3] N. Christofides. Worst Case Analysis of a New Heuristic for the Travelling Salesman Problem.

[4] G.A. Croes. A Method for Solving Travelling Salesman Problems, Operations Research 6 (1958).pp 791-812.

[5] E.W. Dijkstra. A Note on Two Problems in Connexion with Graphs, Numerische Mathematik 1(1959) pp. 269-271.

[6] Michael R. Garey and David S. Johnson. Computers and Interactibility - A Guide to the Theory of NP-Completeness, W.H. Freeman and Co., San Francisco, 1979.

[7] J. Gavett. Three Heuristic Rules for Sequencing Jobs to a Single Production Facility, Management Science 11 (1965) pp 166-176.

[8] A.M. Geoffrion. Lagrangian Relaxation for Integer Programming, Math. Programming Study 2 (1974) pp. 82-114.

[9] M. Grötschel. On the Symmetric Travelling Salesman Problem: Solution of a 120 City Problem, Mathematical Programming 12 (1980) pp 61-77.

[10] M. Grötschel and M.W. Padberg. On the Symmetric Travelling Salesman Problem I: Inequalities, Mathematical Programming 16 (1979) pp 265-280.

[11] M. Grötschel and M.W. Padberg. On the Symmetric Travelling Salesman Problem II: Lifting Theorems and Facets, Mathematical Programming 16 (1979) pp 281-302.

[12] K. Helbig Hansen and J. Karup. Improvements of the Held-Karp Algorithm for the Symmetric Travelling Salesman Problem, Mathematical Programming 7 (1974) pp 87-96.

[13] Michael Held and Richard M. Karp. The Travelling Salesman Problem and Minimum Spanning Trees, Operations Research 18 (1970) pp. 1138-1162.

[14] Michael Held and Richard M. Karp. The Travelling Salesman Problem and Minimum Spanning Trees: Part II, Math. Programming 1 (1971) pp. 6-26.

[15] Michael Held, Philip Wolfe and Harlan P. Crowder. Validation of Subgradient Optimization, Math. Programming 6 (1974) pp. 62-88.

[16] Ellis Horowitz and Sartaj Sahni. Fundamentals of Computer Algorithms, Computer Science Press Inc., Computer Software Engineering Series, 1978.

[17] R.M. Karp. Reducibility Among Combinatorial Problems, Complexity of Computer Computations, Plenum Press, New York, 1972.

[18] P. Krolak, W. Felts and G. Marble. A Man Machine Approach Towards Solving The Travelling Salesman Problem, Communication of the Association for Computing Machinery 14 (1971) pp 327-334.

[19] J.B. Kruskal. On the Shortest Spanning Subtree of a Graph and the Travelling Salesman Problem, Proc. Am. Math. Soc. 1(1956) pp. 48-50.

[20] S. Lin and B.W. Kernighan. An Effective Heuristic Algorithm for the Travelling Salesman Problem, Operations Research 21 (1973) pp. 498-616.

[21] P. Miliotis. Integer Progamming Approaches to The Travelling Salesman Problem, Mathematical Programming 10 (1976) pp 367-378.

[22] T.A.J. Nicholson. A Sequential Method for Discrete Optimization Problems and its Application to The Assignment, Travelling Salesman, and Three Machine Scheduling Problems, J. Inst. Math. Appl. 3 (1967) pp 362-375.

[23] M.W. Padberg and S. Hong. On The Travelling Salesman Problem: A Computational Study, Mathematical Programming Study 12 (1980) pp 78-107.

[24] S. Reiter and G. Sherman. Discrete Optimizing, J. Soc. Indust. Appl. Math. 13 (1965) pp 864-889.

[25] S.M. Roberts and B. Flores. An Engineering Approach to The Travelling Salesman Problem, Managements Science 13 (1966) pp 269-288.

[26] J.F. Shapiro. A Survey of Lagrangian Techniques for Discrete Optimization, Annals of Discrete Mathematics 5 (1979) pp. 113-138.

[27] A. Warburton. A Branch and Bound Approach to the MinMax Travelling Salesman Problem, ORSA-TIMS, 1982 Joint National Meeting, San Diego, Ca. Oct. 25-27, 1982.