AN OPTIMAL SOLUTION METHOD FOR
THE MULTIPLE TRAVELLING SALESMAN PROBLEM

by

Bezalel Gavish
and
Kizhanatham Srikanth

1980
Working Paper No. 8027

The Graduate School of Management
The University of Rochester
ABSTRACT

A new optimal solution method for the Multiple Travelling Salesman Problem is developed. A Lagrangean relaxation which requires the computation of a degree-constrained minimal spanning tree is utilized, and the Lagrangean multipliers are updated by the subgradient optimization method. Fast sensitivity analysis techniques are used to increase graph sparsity and reduce the problem size. The algorithm has been tested on problems with up to 400 cities and 10 salesmen. Work is in progress on an improved version of the algorithm that holds promise of being able to solve even larger problems.
INTRODUCTION

The Multiple Travelling Salesman Problem (MTSP) is an extension of the well-known Travelling Salesman Problem (TSP). The MTSP can itself be generalized to a wide variety of routing/scheduling problems, such as the Delivery Problem, the School Bus Routing Problem, the Dial-a-Ride Problem and Topological Design of Computer Networks (see, for example, references [7], [2], [13 - 15]. Finding a good optimal solution method for the MTSP is therefore of importance, particularly if it can be applied to some of its extensions.

The Multiple Travelling Salesman Problem can be stated as follows:

"Given a set of n cities, find a set of routes for M salesmen starting from and ending at the base city 1, such that the total distance covered is minimized, subject to the constraint that each city (apart from city 1) is visited by one and only one salesman."

The number of salesmen M can be a constant or a bounded variable. Adding restrictions on the maximum number of cities a salesman may visit, or the maximum distance he may travel transforms the problem into one of the harder problems mentioned earlier."
We first define some terms in frequent use throughout this paper.

\( V \) : the set of nodes (cities \( \{1,2,\ldots,n\} \)).

\( S \) : the set of nodes \( V-\{1\} = \{2,3,\ldots,n\} \).

\( C_{ij} \) : distance from node \( i \) to node \( j \). If \( C_{ji} = C_{ij} \) \( \forall i, j \in V \)

\( \{C_{ij}\} \) is symmetric. \( \{C_{ij}\} \) is euclidean if \( C_{ik} + C_{kj} > C_{ij} \)

\( \forall i, j, k \in V \). If this relationship does not necessarily hold for all valid \( i, j, k \) then the distance matrix is said to be non-euclidean.

\( A \) : the set of arcs \( \{(i,j) : 1 \leq i < j \leq n\} \).

Note that if \( \{C_{ij}\} \) is symmetric, the arc set \( A = \{(i,1) : i \in S\} \) is enough to define all existing arcs, vs. the set \( \{(i,j) : i, j \in V \} \) that would be required if \( \{C_{ij}\} \) were asymmetric.

\textbf{subtour} : a set of \( k \) arcs \( \{(i_1,i_2), (i_2,i_3), \ldots, (i_k,i_1)\} \)

where \( i \neq j \) \( \forall 1 \leq p, q \leq k \), is said to form a subtour (or cycle) of size \( k \).

\textbf{immediate subtour} : the two arcs \( \{(1,1),(i,1)\} \) for any \( i \in S \) constitute an immediate subtour. In any feasible solution to the MTSP, an immediate subtour represents the route of a salesman who goes from city 1 to city \( i \) and returns to city 1 immediately.
1. **Previous Research on the MTSP**

Existing solution methods for the MTSP are, for the most part, extensions of algorithms developed for optimal solutions to the TSP. Most existing solution methods for the TSP work by relaxing one of three major classes of constraints:

i) **Degree Constraints**: require that every city (except city 1) must have one arc entering and one arc leaving. When \( C_{ij} \) is symmetric, relaxation of these constraints yields a minimal 1-tree problem (Held and Karp [18,19], Hansen and Krarup [17]) when \( M = 1 \).

ii) **Subtour Elimination Constraints**: eliminate any solution containing a subtour that does not include all cities. Relaxation yields an assignment problem (Christofides [6], Balas and Christofides [3], Carpaneto and Toth [5]).

iii) **Integrality Constraints**: require that all variables in the problem be integer. Relaxation yields a linear programming problem (Miliotis [24,25], Padberg and Hong [28], Laporte and Nobert [22]).

Beilmore and Hong [4] showed that an \( n \) city, \( M \) salesman problem with \( M \) constant could be transformed to an \( (n+M-1) \) city TSP, by
adding M-1 duplicates of row/column 1 to the distance matrix. Svetska
and Huckfeldt [33] used a similar transformation and by relaxing the
subtour elimination constraints solved problems involving up to 60
cities and 10 salesmen using enumeration techniques. Ali and Kennington
[1] developed a Lagrangean relaxation of degree constraints and
used a minimal m-tree algorithm to solve a small subset of 100
city problems that were attempted. Laporte and Nobert [22]
developed an algorithm for the MTSP with M variable that
relaxed, in turn, the subtour elimination and integrality constraints
and used linear programming techniques to solve problems up to sizes
of n=100 and M in the range of 1 to 10. Related research on the
MTSP was done by Hong and Padberg [2], Miller, et. al. [26], Orloff
[27], Rao [29], Gavish [10], Svetska [32] and Russell [30].

In this paper an optimal solution method is developed for the
MTSP with \( C_{ij} \) symmetric and M constant. In Section 2 we develop
an integer linear programming formulation of the problem which is the
basis for the relaxations that are described in Section 3. Section 4
describes a sensitivity analysis technique that can be used to reduce
the problem size and increase its sparsity. Section 5 describes the
branch and bound procedure, and Section 6 discusses the computational
results. Possible extensions of the algorithm are discussed in Section 7.
2. **PROBLEM FORMULATION**

In this section we present an integer linear programming formulation of the MTSP is presented. In the formulation we assume that \( C_{ij} \) is symmetric and \( M \) is constant. In Section 7 it is shown how to relax the latter assumption.

2.1 **FORMULATIONS**

The concept of an immediate subtour is of use in constructing a set of constraints for the MTSP. Except for the trivial case where \( M=n-1 \), no feasible solution to the MTSP can have more than \( M-1 \) immediate subtours. The restriction on the number of immediate subtours is used in the following formulation:

Let:

\[
X_{ij} = \begin{cases} 
1 & \text{if a salesman travels from } \{ \text{city } i \text{ to city } j \} \text{, i, j} \in S \\
1 & \text{if a salesman travels from city } i \text{ to city } j \text{ and } j=1, i \in S \\
1 & \text{if a salesman travels from city } i \text{ to city } j \text{ and } i=1, j \in S \\
0 & \text{otherwise} 
\end{cases}
\]

The MTSP is formulated as:
Problem P1

Find binary variables $X_{ij}$, $\delta_j$ that satisfy

\[
Z = \min \{ \sum_{j=1}^{i-1} \sum_{i+1}^{n} C_{ij} X_{ij} + \sum_{i \in S} C_{il} X_{il} \}
\]

(1)

s.t.

\[
\sum_{j \in S} X_{ij} = M
\]

(2)

\[
\sum_{i \in S} X_{il} = M
\]

(3)

\[
X_{j1} + \sum_{i < j} X_{ij} + \sum_{i > j} X_{ji} = 2 \quad \forall j \in S
\]

(4)

\[
\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} X_{ij} = n - 1
\]

(5)

\[
\sum_{1 < j \in S_k} X_{i,j} \leq |S_k| - 1 \quad \forall S_k \subseteq S
\]

(6)

\[
X_{ij} = 0, 1 \quad \text{for} \ 1 \leq i < j \leq n
\]

(7)

\[
X_{il} = 0, 1 \quad \forall i \in S
\]

(8)

\[
X_{ij} + X_{jl} - \delta_j \leq 1 \quad \forall j \in S
\]

(9)

\[
\sum_{j \in S} \delta_j \leq M - 1
\]

(10)

\[
\delta_j = 0, 1 \quad \forall j \in S
\]

(11)
Lemma: An optimal solution to Problem P1 is an optimal solution to the MTSP.

Proof: A feasible solution to the MTSP must satisfy the following conditions:

(i) There are exactly M salesmen leaving city 1 and exactly M salesmen entering it.
(ii) Any other city i (i.e., i \in S) is visited by one and only one salesman.
(iii) No salesman may have a route that does not contain city 1.
(iv) The number of immediate subtours cannot exceed M-1, if M < n-1.

Constraints (2) and (3) ensure that M salesmen depart from and return to city 1 and therefore condition (i) is satisfied. Constraint (4) ensures that any other city is visited by one and only one salesman and so condition (ii) is satisfied. Constraint (5) follows from the fact that a feasible solution to the problem contains n+M-1 arcs, of which M arcs represent salesmen returning to city 1. Constraint (6) ensures that no subtours can be formed that do not include node 1 and so condition (iii) is satisfied. Constraints (7) and (8) are the integrality constraints. Constraints (9) - (11) are redundant in this formulation, but are of use in
the subsequent Lagrangean relaxation. Constraints (9) and (11) ensure that

\[ \delta_j = \begin{cases} 1 \text{ if } X_{ij} = X_{ij} = 1 & \forall j \in S \\ 0 \text{ or } 1 & \text{otherwise} \end{cases} \]

i.e., \( \delta_j = 0 \) only if node \( j \) is not part of an immediate subtour. Therefore, \( \sum_{j \in S} \delta_j \) is an upper bound on the number of immediate subtours in the solution, and constraint (10) ensures that the number of immediate subtours cannot exceed \( M-1 \). Obviously constraints (9) - (11) cannot be used in the trivial case where \( M=n-1 \).

In any feasible solution to Problem PI, the binary variables \( \{X_{ij}\} \) can be represented by an equivalent arc set \( B = \{(a_1,b_1), (a_2,b_2), \ldots, (a_{n+M-1}, b_{n+M-1})\} \) which is defined by \( (i,j) \in B \) if and only if, \( X_{ij} = 1 \). This representation is used in later stages of the paper.
3. SOLUTION PROCEDURE

In this section a Lagrangean relaxation of the problem is developed, together with a solution procedure for the Lagrangean problem.

3.1 A Lagrangean Relaxation of the Problem

To solve Problem P1, we perform a Lagrangean relaxation on constraint sets (4) and (9). Let \( \pi = \{ \pi_2, \pi_3, \ldots, \pi_n \} \) and \( \alpha = \{ \alpha_2, \alpha_3, \ldots, \alpha_n \} \) be the corresponding sets of Lagrange multipliers. Since in constraint set (9) the left hand side is required to be less than or equal to the right hand side, all \( \alpha_i \) values must be nonpositive.

Define:

\[
    \hat{c}_{ij} = \begin{cases} 
        c_{ij} - \pi_i - \pi_j & \text{if } i \in S \text{ and } j \in S \\
        c_{11} - \pi_i - \alpha_i & \text{if } i \in S \text{ and } j = 1 \\
        c_{1j} - \pi_j - \alpha_j & \text{if } i = 1 \text{ and } j \in S 
    \end{cases} 
\]

(12)

The Lagrangean relaxation leads to the following Lagrangean objective function:

\[
    L(\pi, \alpha) = \min \left\{ \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} c_{ij} x_{ij} + \sum_{i \in S} c_{1i} x_{1i} + \sum_{j \in S} \pi_j \left( 2 - \sum_{i<j} x_{ij} - \sum_{i>j} x_{ji} - x_{j1} \right) + \sum_{j \in S} \alpha_j \left( 1 - x_{1j} - x_{j1} - \delta_j \right) \right\} 
\
    = \hat{z} + \sum_{j \in S} \alpha_j + 2 \sum_{j \in S} \pi_j 
\]

(13)
where

\[
\hat{Z} = \min_{X, \delta} \left\{ \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \hat{c}_{ij} x_{ij} + \sum_{i \in S} \hat{c}_{i1} x_{i1} + \sum_{i \in S} \alpha_i \delta_i \right\}
\]  

(14)

The Lagrangean problem now reads:

**Problem LP1**

For a given set of values \(\pi\) and \(\alpha\), find binary variables \(x_{ij}\), \(\delta_i\) that satisfy

\[
L(\pi, \alpha) = \{ \hat{Z} + 2 \sum_{j \in S} \pi_j + \sum_{j \in S} \alpha_j \}
\]

subject to constraints (2) - (3), (5) - (8), (10), (11), (14).

In problem LP1, the subproblem of computing \(\hat{Z}\) is separable, i.e., the variables involved can be separated into at least two disjoint subsets. \(L(\pi, \alpha)\) can be rewritten as:

\[
L(\pi, \alpha) = \{ \tilde{Z}_1 + \tilde{Z}_2 + \tilde{Z}_3 + 2 \sum_{j \in S} \pi_j + \sum_{j \in S} \alpha_j \}
\]  

(15)

where \(\tilde{Z}_1\), \(\tilde{Z}_2\), and \(\tilde{Z}_3\) are defined by

**Problem LPla**

Find binary variables \(x_{ij}\), \(1 \leq i \leq j \leq n\) (or the equivalent set of arcs \(K = \{(a_1, b_1), (a_2, b_2), \ldots, (a_{n-1}, b_{n-1})\}\) that satisfy

\[
\tilde{Z}_1 = \min_{X} \left\{ \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \hat{c}_{ij} x_{ij} \right\}
\]  

(16)

subject to constraints (2), (5) - (7).
Problem LP1b

Find binary variables $X_{i1} \forall i \in S$ (or the equivalent set of arcs $L = \{ (l_1,1), (l_2,1), \ldots, (l_M,1) \}$) that satisfy

$$\tilde{Z}_2 = \min \{ \sum_{i \in S} C_{i1} X_{i1} \}$$

subject to constraints (3), (8).

Without loss of generality, we can require the arcs in $L$ to be arranged such that $\hat{C}_{l_i,1} \leq \hat{C}_{l_j,1} \forall 1 \leq i < j \leq M$.

Problem LP1c

Find binary variables $\delta_i, i \in S$ that satisfy

$$\tilde{Z}_3 = \min \{ \sum_{i \in S} \alpha_i \delta_i \}$$

subject to constraints (10), (11).
3.2 A Solution Procedure for the Lagrangean Relaxation

For a given \( \{x_i\} \) and \( \{\alpha_i\} \), \( L(\pi, \alpha) \) constitutes a lower bound on \( Z \), the optimal solution value for Problem P1. The following Lemma outlines the solution procedure for Problem LP1.

**Lemma 1:** A minimal spanning tree on the graph \( G(V, A) \) with distance function \( \{C_{ij}\} \) and with node 1 constrained to degree \( M \) represents the optimal solution to Problem LP1a. The \( M \) shortest arcs returning to node 1, measured on the distance function \( \{C_{ij}\} \) represent the optimal solution to Problem LP1b.

**Proof:** A spanning tree on \( G(V, A) \) has \( n-1 \) arcs (which satisfies (5)) and contains no subtours (which satisfies (6)). Constraint (7) is implicitly satisfied. A minimal spanning tree on \( G(V, A) \) with node 1 constrained to degree \( M \) further satisfies constraint (2) and minimizes the objective function defined in (16).

Problem LP1b seeks to minimize the sum of the distances of \( M \) arcs which must belong to the set \( \{(i,1) : i \in S\} \). The shortest \( M \) arcs in this set constitute the optimal solution.

Q.E.D.

Problem LP1c is a very simple case of a knapsack problem whose solution procedure is outlined in Lemma 2.
Lemma 2: For a given vector \( \alpha \), let \( R = \{ r_1, r_2, \ldots, r_{n-1} \} \) be a permutation of \( S \) such that

\[
\alpha_{r_{k-1}} \leq \alpha_{r_k} \leq \alpha_{r_{k+1}} \quad \forall \ k=2, \ldots, n-2
\]

and let \( R_1 = \{ r_k : \alpha_{r_k} < 0 \} \). An optimal solution to Problem LP1c can then be obtained as follows:

Case a) If \( |R_1| < M-1 \),

set

\[
\delta^*_{r_k} = \begin{cases} 
1 & \text{if } 1 \leq k \leq |R_1| \\
1 & \text{if } \alpha_{r_k} = 0 \text{ and } |R_1| + 1 \leq k \leq M-1 \\
0 & \text{otherwise}
\end{cases}
\]  \hspace{1cm} (19)

Case b) If \( |R_1| \geq M-1 \),

set

\[
\delta^*_{r_k} = \begin{cases} 
1 & \text{if } 1 \leq k \leq M-1 \\
0 & \text{otherwise}
\end{cases}
\]  \hspace{1cm} (20)

The proof is straightforward.
Note that in either case

\[
\delta_j = \begin{cases} 
\geq X_{ij} + X_{j1} - 1, & \text{if } a_j = 0 \\
= X_{ij} + X_{j1} - 1, & \text{if } a_j \neq 0 
\end{cases} \quad \forall i \in S
\]  \hspace{1cm} (21)

i.e., all constraints in (9) are satisfied and

\[
\sum_{j \in S} \alpha_j (1 - (X_{ij} + X_{j1} - \delta_j)) = 0 \quad \hspace{1cm} (22)
\]

Several algorithms exist for computing degree-constrained minimal spanning trees, (Glover and Klingman [16], Gabow [9], Srikanth [31]); the Glover-Klingman algorithm is best suited for complete graphs and low values of M and is used here. It has a complexity of \(O(n^2 + nM)\) operations. Finding the \(M\) shortest arcs returning to node 1 requires \(O(n \log n)\) operations, as does obtaining the solution to Problem LP1c.

3.3 Computing the Lagrange Multipliers

Given Lagrangean values \(\{\pi_i\}\) and \(\{\alpha_i\}\), it is possible now to solve Problem LP1. The overall objective is to find the best possible set of values \(\{\pi_i\}\) and \(\{\alpha_i\}\), i.e.,

Find \[
\max_{\pi, \alpha} L(\pi, \alpha) \quad \hspace{1cm} (23)
\]
s.t. constraints (2) - (3), (5) - (8), (10), (11), (14).
or,

\[ \text{Find } w^* = w(\pi^*) = \max_{\pi} w(\pi) \]  \hspace{1cm} (24)

where

\[ w(\pi) = \max_{\alpha} \{L(\pi, \alpha)\} \]  \hspace{1cm} (25)

s.t. constraints (2) - (3), (5) - (8), (10), (11), (14).

We first state and prove a lemma that defines sufficient conditions for the best possible \( \{a_{i}\} \) values, given a set of values \( \{\pi_{i}\} \).

**Lemma 3:** For a given set of values \( \{\pi_{i}\} \) suppose a set of values \( \{a_{i}^*\} \) exists such that in the corresponding solution to Problem LP1, for each \( j \in S \) exactly one of the following two conditions is satisfied:

1) \[ a_{1j} + a_{j1} - \delta_j = 1 \]  \hspace{1cm} (26)

2) \[ a_{1j} + a_{j1} - \delta_j < 1 \], and \( a_j \) is at the upper limit of 0.

Then no better set of values \( \{a_{i}\} \) can be found.

**Proof:** If the subgradient optimization procedure (see Section 3.4) is used to improve on the values \( \{a_{i}^*\} \), we find no change in these values is possible. As proven in [20], \( \{a_{i}^*\} \) is therefore the best possible set of values \( \{a_{i}\} \).

Q.E.D.
We now show how to compute the \( \{a^*_i\} \) values and the corresponding solutions \( K^* \) and \( L^* \) to Problems LP1a and LP1b, given a set of \( \{\pi_i\} \) multipliers.

Define \( a^0 = \{0,0,\ldots,0\} \) and set \( a = a^0 \). Let \( K^0 \) and \( L^0 \) be arc sets representing the corresponding optimal solutions for Problems LP1a and LP2b.

Define \( J = \{j : j \in S, (1,j) \in K^0, (j,1) \in L^0\} \)

Case (i) \( |J| \leq M-1 \) (i.e., there are no more than \( M-1 \) immediate subtours).

We can obtain the optimal solution to LP1c by setting

\[
\delta_j^* = \begin{cases} 
1 & \text{if } j \in J \\
0 & \text{otherwise}
\end{cases}
\]

Condition (26) is satisfied, and \( a^* = a^0, K^* = K^0, L^* = L^0 \).

Case (ii) \( |J| > M-1 \). Since constraints (2) and (3) are satisfied \( |J| = M \) i.e., there are exactly \( M \) immediate subtours.

It follows therefore that since \( \sum_{i \in S} \delta_i \) must be \( \leq M-1 \) (from constraint (10)), there is at least one value \( j \in J \) such that
(1,j) \in K^0, (j,1) \in L^0 \text{ and } \delta_j = 0

i.e., condition (26) is not satisfied, and so \( \alpha^0 \) is not the required \( \alpha^* \) vector.

Before proceeding to compute the required \( \alpha^* \) vector, we note that changing the \( \alpha \) vector affects only the values \( \hat{C}_{1j} \) and \( \hat{C}_{j1} \), \( \forall j \in S \). Consider the effects of decreasing uniformly the \( \alpha_j \) values for all \( j \in J \); the values \( \hat{C}_{1j} \) and \( \hat{C}_{j1} \) increase for all \( j \in J \), i.e., in all immediate subtours. Sooner or later, we will reach a point where there exists (in at least one of the subproblems LP1a and LP1b) an alternate optimal solution that can be used to compute the \( \alpha^* \) vector.

Compute

\[
e_1 = \hat{C}_{k_1,1} - \hat{C}_{j_1,1} = \min\{ (\hat{C}_{k_1} - \hat{C}_{j_1}) \colon j \in J, \text{ and } (k,1) \notin L^0 \} \tag{27}
\]

\[
= \min\{ \hat{C}_{k_1} \colon (k,1) \notin L^0 \} - \max\{ \hat{C}_{j_1} \colon j \in J \}
\]

and

\[
e_2 = \hat{C}_{1,k_2} - \hat{C}_{1,j_2} = \min\{ (\hat{C}_{1k} - \hat{C}_{1j}) \colon j \in J \text{ and } (1,k) \notin K^0 \} \tag{28}
\]

\[
= \min\{ \hat{C}_{1k} \colon (1,k) \notin K^0 \} - \max\{ \hat{C}_{1j} \colon j \in J \}
\]
Unless the original symmetric distance matrix is rendered temporarily asymmetric by the use of the branch and bound procedure (described in section 5) we will have $e_1 = e_2$, $j_1 = j_2$ and $k_1 = k_2$.

One of the following three cases must hold:

**Case (ii)a** \( e_1 < e_2 \)

If we set \(
\alpha_j = \begin{cases} 
-e_1, & \text{ if } j \in J \\
0 & \text{ otherwise}
\end{cases}
\)

we have \( C_{j_1,1} = C_{k_1,1} \). For this particular vector \( \alpha \) there are at least two optimal solutions to Problem LP1b: the current solution \( L^0 \) and an alternate solution \( L^1 = L^0 - (j_1,1) + (k_1,1) \).

If we choose the alternate solution for Problem LP1b, an optimal solution to Problem LP1c can be obtained by setting

\[
\delta_j^* = \begin{cases} 
1 & \text{ if } j \in J, j \neq j_1 \\
0 & \text{ otherwise}
\end{cases}
\]

Condition (26) is now satisfied and the vector \( \alpha^* \) is defined by
\[ a_j^* = \begin{cases} 
-e_1 & \text{if } j \in J \\
0 & \text{otherwise} 
\end{cases} \]

Further, \( K^* = K^0 \) and \( L^* = L^1 = L^0 - (j_1,1) + (k_1,1) \).

**Case (ii)b**  \[ e_1 = e_2 \]

Setting the \( a_j \) values as in case (ii)a yields \( \tilde{C}_{j_1,1} = \tilde{C}_{k_1,1} \); further, \( C_{j_1,j_2} = C_{k_1,k_2} \). As before, Problem LP1b has at least two optimal solutions; Problem LP1a may have more than one optimal solution as well. Whether Problem LP1a has an alternate optimal solution or not, proceeding exactly as in case (ii)a we get the same values for \( a^*, \delta^*, K^* \) and \( L^* \).

**Case (ii)c**  \[ e_1 > e_2 \]

As noted earlier, this case can arise only in the branch and bound procedure, e.g., if we want to solve the problem with \( X_{k_1}^{set} = 0 \), we add a very large value to \( C_{k_1,1} \). Under such circumstances, if \( X_{1,k_2} \) is not fixed to either 0 or 1, it is possible to have \( C_{1,k_2} < C_{k_2,1} \), and hence \( k_1 \neq k_2 \), \( e_1 > e_2 \). Evaluating the required \( \alpha^* \) vector at this point involves a lot of extra computation; this can be avoided by making sure that \( X_{k_2,1} \) is never set equal to 0 unless \( X_{1,k_2} \) is also fixed, either to 0 or to 1. Following this simple rule ensures that the case of \( e_1 > e_2 \) never arises, whether in the branch and bound procedure or otherwise.
We note the following points:

1) \( x^* = x^0 \), whether or not \( |J| \leq M-1 \).

2) If \( |J| \leq M-1 \), \( L^* = L^0 \), i.e. the \( M \) shortest arcs in the set 
   \( I = \{(i,j) : j \in S\} \). If \( |J| > M-1 \), \( L^* = L^0 - (j,1) + (k,1) \)
   i.e., \( L^0 \) - the longest arc \( e \in L^0 \) + the shortest arc \( e \in \{(I-L^0) \} \).

We can now state a lemma showing the general solution procedure
for Problem LPI, given a vector \( \pi \). The proof is straightforward.

Lemma 4: Set \( a \in \{0,0,\ldots,0\} \). Compute a minimal spanning tree with
node 1 constrained to degree \( M \); this is the optimal solution for
Problem LPI.a.

Select the \( M-1 \) shortest arcs in the arc set \( I = \{(i,j) : j \in S\} \).
From the remaining arcs in \( I \), choose the shortest arc whose inclusion
in the solution will not lead to the formation of \( M \) immediate
subtours. This is the optimal solution for Problem LPI.b. The
optimal values \( \alpha^* \) and \( \delta^* \) can now be computed as described earlier
in this section.

3.4 The Subgradient Optimization Procedure

The subgradient optimization procedure (Held, Wolfe and Crowder,
[20]) is one of the more successful methods for obtaining the
optimal Lagrangean multiplier values. It is an iterative procedure
where at the \( k^{th} \) iteration the subgradient directions are computed:
\[ Y_j^k = 2 - (X_{1j} + \sum_{i=1}^{j-1} X_{ij} + \sum_{i=j}^{n} X_{ji}) \quad \forall j \in S \]  

Using a step size \( t_k \) the new set of multipliers \( \pi^{k+1} \) is computed by

\[ \pi_j^{k+1} = \pi_j^k + t_k Y_j^k \]  

where

\[ t_k = \frac{\lambda_k (\overline{Z} - W(\pi_j^k))}{||Y_j^k||^2} \]  

and \( \overline{Z} \) is an upper bound on the value of the optimal solution.

3.5 **Summary of Solution Procedure**

1. Using a heuristic procedure, obtain a feasible solution to the MTSP; this is the upper bound for the optimal solution value that is required for the subgradient procedure. Initialize the Lagrangean vector \( \pi \). (Good results were obtained by initializing \( \pi_i \) to be equal to the second minimum in \( \{C_{ij} : j=1,2,\ldots,n\} \), \( \forall i \in V \). The actual heuristic used was a successive node insertion procedure similar to one form of the Lin and Kernighan heuristic [23]).

2. Compute the optimal solution to Problem LP1 as shown in Lemma 4.
3. If (i) the corresponding $w(n)$ value $\geq$ upper bound, or
   (ii) the optimal solution to LP1 is also a feasible solution to Problem P1,

   stop; the optimal solution to Problem P1 has been found.

4. If in the optimal solution to LP1 the number of nodes not at their required degree is less than a given parameter, use a heuristic procedure to search for a feasible solution to Problem P1. If a feasible solution is found that is an improvement on the best known feasible solution, update the value of the upper bound.

5. Update the vector $\pi$ using the subgradient procedure.

6. Repeat steps 2 through 5 until no significant improvements in the vector $\pi$ are possible.

7. Perform sensitivity analysis (described in section 4) to try and reduce the problem size.

8. Perform the Branch and Bound procedure (described in section 5).
4. SENSITIVITY ANALYSIS

Consider Problem LP1, whose solution consists of a set of arcs. There are two types of sensitivity analysis on such a problem that give us the change in the solution value \( L(\pi, \alpha) \) and therefore in \( w(\pi) \) if:

Type (1): an arc \((i,j)\) currently in the solution is eliminated (i.e., \( x_{ij} \) is currently 1 and is fixed equal to 0), or
Type (2): an arc \((i,j)\) not currently in the solution is brought into it (i.e., \( x_{ij} \) is currently 0 and is fixed equal to 1).

Sensitivity analysis can be put to very good use in obtaining the optimal solution to the original problem. For any vector \( \pi \), note that \( w(\pi) \) is a lower bound on \( Z \), the value of the optimal solution to Problem P1. If the change in the value of \( w(\pi) \) that results from fixing an \( x_{ij} \) variable to 0 (or 1) is such that the new value of \( w(\pi) \) is \( \geq \) an upper bound on \( Z \), that \( x_{ij} \) must take on a value of 1 (or 0) in the optimal solution, i.e., that \( x_{ij} \) variable can be forced to 1 (or 0) and its equivalent arc forced to exist/not exist in any solution to the problem or to a relaxation of the problem.

For the rest of this section, when an arc is eliminated we will speak of it as being forced to 0; similarly, when an arc is required to exist, we will speak of it as being forced to 1.
It is worthwhile noting the following points:

1) If a sufficient number of arcs are forced to 0, a sparse graph results; special algorithms that exploit this sparsity can be used to solve the various subproblems, with a corresponding reduction in computing time. On a sufficiently sparse graph, algorithms that enumerate Hamiltonian paths can be used to search for the optimal solution.

2) Any subset of arcs forced to 1 that form a chain can be replaced by a single arc of appropriate length, and all interior nodes in the chain removed from further consideration, thus reducing the problem size $n$.

3) If any subset of arcs forced to 1 forms a subtour (necessarily including node 1), all nodes in the subtour except node 1 can be removed; this reduces the values of $n$ and $M$.

4) Even if no arcs are forced to 0 or 1, the relative magnitudes of the changes in solution value assist in choosing a good separation variable in the branching stage of the branch and bound procedure.

A solution to Problem LPI can be represented by a degree-constrained minimal spanning tree, together with $M$ arcs returning to node 1. The authors have shown in [12] that Type (1) sensitivity analysis can be performed for all the $n-1$ arcs in such a spanning
tree in \( O(n^2) \) operations. Performing sensitivity analysis on the remaining \( M \) arcs requires \( O(n+M) \) operations.

To perform Type (2) sensitivity analysis on the \( \frac{n(n-1)}{2} - M \) arcs not in the current solution to LP1, note that if an arc \((i,j)\) not belonging to a minimal spanning tree is set equal to 1, the new minimal spanning tree differs from the old by one arc only; this arc is the largest arc in the cycle formed when \((i,j)\) is added to the old minimal spanning tree. In the case of a minimal spanning tree where a node (here node 1) is constrained to degree \( M \), removal of this largest arc after adding the new arc \((i,j)\) yields a spanning tree in which node 1 is of degree \( k \), where \( M-1 \leq k \leq M+1 \). As proved by the authors in [12] the solution value of this new spanning tree is a lower bound on the solution value of a degree-constrained minimal spanning tree on the changed graph and therefore can be used in forcing arcs to 0. Exploiting this fact, we can perform sensitivity analysis on all the arcs not in the solution to Problem LP1 in \( O(\log n+n^2) = O(n^2) \) operations.

It is possible, therefore, to perform sensitivity analysis of Types (1) and (2) on all arcs (whether in the solution or not) in \( O(n^2) \) operations.

Table 1 shows the result of applying both types of sensitivity analysis at the start of the branch and bound procedure. The
<table>
<thead>
<tr>
<th>Problem Characteristics</th>
<th>Effect of sensitivity analysis</th>
<th>CPU Sec. (IBM 3032) for sensitivity analysis</th>
</tr>
</thead>
<tbody>
<tr>
<td>Before sensitivity analysis</td>
<td>After sensitivity analysis</td>
<td></td>
</tr>
<tr>
<td>n</td>
<td>M</td>
<td>$</td>
</tr>
<tr>
<td>40</td>
<td>2</td>
<td>819</td>
</tr>
<tr>
<td>60</td>
<td>8</td>
<td>1829</td>
</tr>
<tr>
<td>60</td>
<td>10</td>
<td>1829</td>
</tr>
<tr>
<td>80</td>
<td>2</td>
<td>3239</td>
</tr>
<tr>
<td>100</td>
<td>2</td>
<td>5049</td>
</tr>
<tr>
<td>150</td>
<td>2</td>
<td>11324</td>
</tr>
<tr>
<td>150</td>
<td>4</td>
<td>11324</td>
</tr>
<tr>
<td>150</td>
<td>6</td>
<td>11324</td>
</tr>
<tr>
<td>150</td>
<td>8</td>
<td>11324</td>
</tr>
<tr>
<td>150</td>
<td>10</td>
<td>11324</td>
</tr>
<tr>
<td>200</td>
<td>2</td>
<td>20099</td>
</tr>
<tr>
<td>200</td>
<td>4</td>
<td>20099</td>
</tr>
<tr>
<td>200</td>
<td>6</td>
<td>20099</td>
</tr>
<tr>
<td>200</td>
<td>8</td>
<td>20099</td>
</tr>
<tr>
<td>200</td>
<td>10</td>
<td>20099</td>
</tr>
<tr>
<td>250</td>
<td>2</td>
<td>31374</td>
</tr>
<tr>
<td>250</td>
<td>4</td>
<td>31374</td>
</tr>
<tr>
<td>250</td>
<td>6</td>
<td>31374</td>
</tr>
<tr>
<td>250</td>
<td>8</td>
<td>31374</td>
</tr>
<tr>
<td>250</td>
<td>10</td>
<td>31374</td>
</tr>
</tbody>
</table>

**TABLE 1** - Effect of Sensitivity Analysis in Reducing Problem Size and Developing Graph Sparsity in Non-Euclidean Data Sets.
relative number of arcs eliminated increases with the number of cities n. Up to 98% of all arcs can be eliminated, resulting in a very sparse graph. As mentioned earlier, special algorithms that exploit this sparsity can be used to solve the subproblems in substantially reduced computing time. These special algorithms have not been incorporated in the general algorithm described in Sections 3 and 5, nor has Type (2) sensitivity analysis been used to eliminate arcs; a forthcoming paper by the authors combines these special algorithms with Type (2) sensitivity analysis and multiple Hamiltonian path search techniques to produce significantly improved computational results.
5. THE BRANCH AND BOUND PROCEDURE

In this section, we present features that are specific to our problem.

5.1 USE OF AN ARTIFICIAL UPPER BOUND

The performance of any branch and bound procedure is determined, to a large extent, by the quality of the upper bound in use at the beginning of the procedure. If the value of this upper bound is much larger than the value of the optimal solution, a lot of computation will be performed (and a lot of branch and bound nodes fathomed) unnecessarily.

It is worthwhile exploring the possibility of using an artificial upper bound AUB viz. a value greater than the best known lower bound but less than the best known feasible solution. If in the course of the branch and bound procedure we find at least one feasible solution with a value less than or equal to AUB then AUB is truly an upper bound; if no such feasible solution is found, then AUB is not an upper bound and the branch and bound procedure must be repeated with a higher value for AUB.

We have a tradeoff situation: with low AUB values, problems requiring only a single use of branch and bound will be solved faster but the number of problems requiring repetitions of the branch and bound procedure will increase, together with their computing times, and vice versa. Assuming that our objective is to
minimize the expected computing time required to get the optimal solution to a problem, we must choose an AUB value that satisfies this objective.

If in the branch and bound procedure a feasible solution is found that has a lower value than the current AUB value, this lower value (a true upper bound) replaces the AUB. Further, in many real world applications the original cost data \( \{ C_{ij} \} \) is subject to inaccuracy and the objective is to find a good solution (not necessarily the optimal) given those \( \{ C_{ij} \} \) values; in such cases, any feasible solutions found during the branch and bound procedure are useful.

Define

\[ ZHI = \text{value of best known feasible solution,} \]
\[ w(\pi^*) = \text{value of lower bound, at the start of the branch and bound procedure.} \]

From computational experience, the rule

\[
AUB = \begin{cases} 
ZHI & \text{if } ZHI \leq (1+g)w(\pi^*) \\
\left( w(\pi^*) + \text{Min} \{ h(ZHI - w(\pi^*)) \} \right) & \text{otherwise} 
\end{cases} 
\]

(32)
was found to provide good results, with
\[
    h = \begin{cases} 
        0.25 & \text{for non-euclidean distance matrices} \\
        1.0 & \text{for euclidean distance matrices} 
    \end{cases} \\
    g = 0.005 \text{ to } 0.0025, \text{ generally decreasing as the number of cities increased.}
\]

Table 2 shows the impact of using an artificial upper bound. Out of 225 problems attempted, over 95% were solved without repetitions of the branch and bound procedure; all remaining problems required only a single repetition. This clearly demonstrates the usefulness of that approach for the MTSP.

<table>
<thead>
<tr>
<th>Number of Cities</th>
<th>Number of Problems</th>
<th>Percent of Problems Solved with at Most 1 Pass Through Branch and Bound</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>h, g</td>
<td>Attempted</td>
</tr>
<tr>
<td>60</td>
<td>0.25, 0.005</td>
<td>50</td>
</tr>
<tr>
<td>80</td>
<td>0.25, 0.005</td>
<td>50</td>
</tr>
<tr>
<td>100</td>
<td>0.25, 0.005</td>
<td>50</td>
</tr>
<tr>
<td>150</td>
<td>0.25, 0.0025</td>
<td>25</td>
</tr>
<tr>
<td>200</td>
<td>0.25, 0.0025</td>
<td>25</td>
</tr>
<tr>
<td>250</td>
<td>0.25, 0.0025</td>
<td>20</td>
</tr>
<tr>
<td>400</td>
<td>0.25, 0.004</td>
<td>5</td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td>225</td>
</tr>
</tbody>
</table>

<sup>a</sup> One Problem remained unsolved due to excessive computing time.

<sup>b</sup> Parameters h and g are defined in Section 5.1.

TABLE 2 - Effect of Using an Artificial Upper Bound (AUB) in the Branch and Bound Procedure for Symmetric, Non-Euclidean Problems.
5.2 Separation and Branching Rules

Two other factors affecting the performance of any branch and
bound technique are the separation and branching rules. At any stage
of the branch and bound tree, how do we choose which arc to fix and
what value do we fix it to?

Knowledge of the effect on the solution value of fixing
a particular arc in the solution to 0 or 1 is useful here.

Define

\[ B : \{b_1, b_2, \ldots, b_{n+M-1}\} = \{(i_1,j_1), (i_2,j_2), \ldots, (i_{n+M-1}, j_{n+M-1})\} \]

= the set of arcs in a particular solution to problem LP1.

\[ p_k : \text{change in solution value if arc } b_k \text{ fixed to 0.} \]
\[ q_k : \text{change in solution value if arc } b_k \text{ fixed to 1.} \]
\[ d_i : \text{degree of node } i \text{ in the solution } B. \]
\[ N_k : \text{number of arcs forced to 0 in a solution } B \text{ if arc } b_k \]
\[ \text{is fixed to 1.} \]

The \( p_k \) values can be computed for all arcs \( b_k \in B \) in \( O(n^{2+M}) \)
operations, as mentioned in section 4. Define \( I_k = \{i : i = i_k \text{ or } j_k', i \neq 1\} \).
In computing a \( q_k \) value, we must note that fixing arc \( b_k \) to 1 causes
a change in solution structure (and possible therefore in solution value)
only if the following conditions are satisfied:
i) Arc \( b_k = (i_k, j_k) \) has not been fixed to 1 earlier in the branch and bound tree.

ii) There exists at least one node \( j \in I_k \) such that \( d_j > 2 \), and at least one arc incident at \( j \) (other than \( b_k \)) has been fixed to 1 earlier in the branch and bound tree.

Fig. 1(a) shows an example of a case where these conditions are satisfied. When \( (i_k, j_k) \) is set to 1 the node \( i_k \) now has two incident arcs fixed to 1 and the new solution cannot therefore include the free arc \( (i_k, f) \). Here \( N_k = 1 \).

For a given solution \( B \), it is easy to compute \( N_k \) for \( 1 \leq k \leq n^*M-1 \). If \( N_k = 0 \) for some \( k \), then fixing arc \( b_k \) to 1 will cause no change in the solution structure or value, and \( q_k = 0 \).

If \( N_k = 1 \) and \( b_j \) is the arc forced to 0, then \( q_k = p_j \), which is already computed. If \( N_k > 1 \) (see example in Fig. 2) then a new solution must be computed with \( (i_k, j_k) \) fixed to 1, which requires \( \Theta(n^2) \) operations.

To summarize, we can compute all \( p_k \) and \( q_k \) values for a given solution in \( O(n^2 + M \cdot |\tilde{N}| \cdot n^2) \) operations where \( \tilde{N} = \{N_k : N_k > 1\} \). In most cases, \( |\tilde{N}| = 0 \).
Figure 1. Example of change in solution structure when an arc 
\((i_k, j_k)\) is fixed to 1.

Figure 2. Example of more than 1 arc being forced to 0 when 
\((i_k, j_k)\) is fixed to 1.
The \( p_k \) and \( q_k \) values are used to identify arcs which have a high probability of being in the optimal solution, and to fix a particular arc in this subset. The separation rule which led to the best computational results was to choose the arc \( b_k \) that met the following criteria, in decreasing order of importance:

i) highest value of \( q_k + w(\pi) \)

ii) \( N_k > 0? \)

iii) \[
\begin{cases} 
\max[0, d_{jk} - 2], & \text{if } i_k \neq 1, j_k \neq 1 \\
\max[0, d_{ik}], & \text{if } i_k = 1 \\
\max[0, d_{ik}], & \text{if } j_k = 1 
\end{cases}
\]

iv) highest value of \( p_k + w(\pi) \)

v) highest value of \( \lceil p_k + (\pi) \rceil \) where the 'ceiling' function \( \lceil \cdot \rceil \) is defined by \( \lceil x \rceil = \text{smallest integer } \geq x \). This function is of use only if the original distance matrix is all-integer.

If \( q_k \) is relatively large, the probability that arc \( b_k \) will be in the optimal solution is low; similarly, if \( p_k \) is relatively large, the probability that \( b_k \) will be in the optimal solution is low. Therefore, once an arc \( b_k \) is chosen to be fixed, we fix it to

\[
b_k = \begin{cases} 
1 & \text{if } \lceil w(\pi) + q_k \rceil \leq \lceil p_k + w(\pi) \rceil \\
0 & \text{if } \lceil w(\pi) + q_k \rceil > \lceil p_k + w(\pi) \rceil 
\end{cases}
\]
6. COMPUTATIONAL RESULTS

In this section we present computational results that were obtained in a series of tests which used an implementation of the algorithm. The program was coded in FORTRAN and run on an IBM 3032 computer. Computations were performed on both euclidean and non-euclidean distance matrices.

6.1 Non-Euclidean Distance Matrices

The algorithm was tested on non-euclidean problems with up to 400 cities and 10 salesmen. Ten different distance matrices were constructed for problem sizes in the range \(60 \leq n \leq 100\); budget constraints precluded the same number of samples for larger problem sizes. The distance matrix elements \(c_{ij}\) were integer and generated randomly from a uniform distribution in the range 0-400.

The computational results are summarized in Table 3. The average number of subgradient iterations required is relatively modest; the number of nodes examined in the branching tree is less than 1000 even when \(n=400\). Of particular interest is the behavior of the integer gap \(\frac{Z-w^*}{Z}\). The integer gap decreases as the problem size increases; This might imply that as problem sizes increase even further, a majority of problems could have an integer gap close to (if not equal to) zero. Most of the past experience with M-TSP algorithms had shown that for a given \(n\), the problems were easier to solve as \(M\) increased; however, we were unable to detect any such pattern in our computational results.
<table>
<thead>
<tr>
<th>n</th>
<th>M</th>
<th>no. problems solved</th>
<th>% integer gap</th>
<th>no. of nodes in B&amp;B</th>
<th>CPU sec. (IBM 3032)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>read repeat</td>
<td>B&amp;B</td>
<td>total</td>
<td>read</td>
</tr>
<tr>
<td>60</td>
<td>2</td>
<td>1</td>
<td>97 195 289</td>
<td>97 464 1286</td>
<td>.16</td>
</tr>
<tr>
<td>60</td>
<td>4</td>
<td>10</td>
<td>83 172 273</td>
<td>83 323 1269</td>
<td>.10</td>
</tr>
<tr>
<td>60</td>
<td>8</td>
<td>10</td>
<td>101 232 262</td>
<td>101 904 3607</td>
<td>.13</td>
</tr>
<tr>
<td>60</td>
<td>10</td>
<td>1</td>
<td>101 186 272</td>
<td>101 529 2220</td>
<td>.14</td>
</tr>
<tr>
<td>80</td>
<td>2</td>
<td>10</td>
<td>83 220 292</td>
<td>83 780 1616</td>
<td>.10</td>
</tr>
<tr>
<td>80</td>
<td>4</td>
<td>10</td>
<td>138 232 281</td>
<td>138 137 4256</td>
<td>.12</td>
</tr>
<tr>
<td>80</td>
<td>6</td>
<td>10</td>
<td>77 220 245</td>
<td>77 512 1437</td>
<td>.08</td>
</tr>
<tr>
<td>80</td>
<td>8</td>
<td>10</td>
<td>92 220 274</td>
<td>92 587 1215</td>
<td>.07</td>
</tr>
<tr>
<td>80</td>
<td>10</td>
<td>10</td>
<td>122 222 268</td>
<td>122 747 1546</td>
<td>.08</td>
</tr>
<tr>
<td>100</td>
<td>2</td>
<td>10</td>
<td>123 232 311</td>
<td>123 863 3330</td>
<td>.05</td>
</tr>
<tr>
<td>100</td>
<td>4</td>
<td>10</td>
<td>87 234 290</td>
<td>87 423 1311</td>
<td>.06</td>
</tr>
<tr>
<td>100</td>
<td>6</td>
<td>10</td>
<td>191 249 292</td>
<td>191 832 3625</td>
<td>.11</td>
</tr>
<tr>
<td>100</td>
<td>8</td>
<td>10</td>
<td>92 228 274</td>
<td>92 587 1215</td>
<td>.07</td>
</tr>
<tr>
<td>100</td>
<td>10</td>
<td>10</td>
<td>112 214 284</td>
<td>112 454 1731</td>
<td>.03</td>
</tr>
<tr>
<td>150</td>
<td>2</td>
<td>5</td>
<td>141 225 286</td>
<td>141 652 1467</td>
<td>.06</td>
</tr>
<tr>
<td>150</td>
<td>4</td>
<td>5</td>
<td>224 234 246</td>
<td>224 872 1341</td>
<td>.06</td>
</tr>
<tr>
<td>150</td>
<td>6</td>
<td>5</td>
<td>217 232 252</td>
<td>217 822 1777</td>
<td>.05</td>
</tr>
<tr>
<td>150</td>
<td>8</td>
<td>5</td>
<td>243 264 283</td>
<td>243 1174 1761</td>
<td>.09</td>
</tr>
<tr>
<td>150</td>
<td>10</td>
<td>5</td>
<td>196 263 314</td>
<td>196 1048 2671</td>
<td>.04</td>
</tr>
<tr>
<td>200</td>
<td>2</td>
<td>5</td>
<td>198 273 319</td>
<td>198 835 1719</td>
<td>.03</td>
</tr>
<tr>
<td>200</td>
<td>4</td>
<td>5</td>
<td>224 274 302</td>
<td>224 790 1443</td>
<td>.00</td>
</tr>
<tr>
<td>200</td>
<td>6</td>
<td>5</td>
<td>222 250 302</td>
<td>225 2546 8446</td>
<td>.09</td>
</tr>
<tr>
<td>200</td>
<td>8</td>
<td>5</td>
<td>206 245 273</td>
<td>206 1426 3563</td>
<td>.08</td>
</tr>
<tr>
<td>200</td>
<td>10</td>
<td>5</td>
<td>232 270 303</td>
<td>1179 2488 4665</td>
<td>.09</td>
</tr>
<tr>
<td>250</td>
<td>2</td>
<td>4</td>
<td>196 212 228</td>
<td>334 796 1639</td>
<td>.00</td>
</tr>
<tr>
<td>250</td>
<td>4</td>
<td>4</td>
<td>228 236 243</td>
<td>416 955 1672</td>
<td>.06</td>
</tr>
<tr>
<td>250</td>
<td>6</td>
<td>4</td>
<td>190 224 251</td>
<td>190 576 1130</td>
<td>.00</td>
</tr>
<tr>
<td>250</td>
<td>8</td>
<td>4</td>
<td>238 256 286</td>
<td>238 1766 3647</td>
<td>.06</td>
</tr>
<tr>
<td>250</td>
<td>10</td>
<td>4</td>
<td>266 234 201</td>
<td>462 979 1834</td>
<td>.06</td>
</tr>
<tr>
<td>400</td>
<td>2</td>
<td>1</td>
<td>235 235 235</td>
<td>2258 2258 2258</td>
<td>.00</td>
</tr>
<tr>
<td>400</td>
<td>4</td>
<td>1</td>
<td>252 252 252</td>
<td>4507 4507 4507</td>
<td>.00</td>
</tr>
<tr>
<td>400</td>
<td>6</td>
<td>1</td>
<td>242 242 242</td>
<td>3059 3059 3059</td>
<td>.16</td>
</tr>
<tr>
<td>400</td>
<td>8</td>
<td>1</td>
<td>241 241 241</td>
<td>1642 1642 1642</td>
<td>.00</td>
</tr>
<tr>
<td>400</td>
<td>10</td>
<td>1</td>
<td>228 228 228</td>
<td>2410 2410 2410</td>
<td>.15</td>
</tr>
</tbody>
</table>

**TABLE 3** - Computational Results for Symmetric Non-Euclidean M-Travelling Salesman Problems

Parameters: $h=0.25$, $g = \begin{cases} 0.0005 & \text{when } 60 \leq n \leq 100 \\ 0.00025 & \text{when } 150 \leq n \leq 250 \\ 0.0004 & \text{when } n = 400 \end{cases}$

(a): $\%$ integer gap = 100. $(Z-w^*)/Z$. Minimum was 0.0 for $60 \leq n \leq 250$.

(b): 1 problem was not solved due to excessive computing time (>900 CPU sec.).
6.2 Euclidean Distance Matrices

Coordinates were generated randomly in a square whose sides were of length 400. The Euclidean distances between the coordinates (rounded off to the nearest integer) formed the elements of the distance matrix.

Problems involving up to 100 cities and 10 salesmen were attempted with euclidean distance matrices. Computational results were not as good as for non-euclidean data; the variance in performance for a specific problem size was far larger than with non-euclidean data. One possible reason for the poorer performance might be the fact that the number of immediate subtours in optimal solutions to euclidean problems usually proved to be less than in solutions to non-euclidean problems of equal size. It is possible that imposing upper limits on the number of immediate subtours allowed (discussed in Section 7) may improve the performance of the algorithm as far as euclidean data is concerned. The computational results are presented in Table 4.

6.3 Comparison with Existing Algorithms

In Table 5 the performance of the algorithm is compared to that of Laporte and Nobert [22] and Ali and Kennington [11]. In the problems solved by Laporte and Nobert, the number of salesmen M was assumed to be a free variable, rather than fixed; as explained in Section 7, the algorithm can be modified to solve problems where M is variable without a substantial increase in computing time.
<table>
<thead>
<tr>
<th>n</th>
<th>M</th>
<th>Problems solved</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>30</td>
<td>4</td>
<td>5, 5, 1</td>
</tr>
<tr>
<td>30</td>
<td>6</td>
<td>5, 5, 2</td>
</tr>
<tr>
<td>30</td>
<td>8</td>
<td>5, 5, 1</td>
</tr>
<tr>
<td>30</td>
<td>10</td>
<td>5, 5, 1, 1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>n</th>
<th>M</th>
<th>Problems solved</th>
</tr>
</thead>
<tbody>
<tr>
<td>60</td>
<td>2</td>
<td>5, 5, 5</td>
</tr>
<tr>
<td>60</td>
<td>4</td>
<td>5, 5, 5, 5</td>
</tr>
<tr>
<td>60</td>
<td>6</td>
<td>5, 5, 5, 3</td>
</tr>
<tr>
<td>60</td>
<td>8</td>
<td>5, 5, 5, 5</td>
</tr>
<tr>
<td>60</td>
<td>10</td>
<td>5, 5, 4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>n</th>
<th>M</th>
<th>Problems solved</th>
</tr>
</thead>
<tbody>
<tr>
<td>80</td>
<td>2</td>
<td>5, 5, 5</td>
</tr>
<tr>
<td>80</td>
<td>4</td>
<td>5, 5, 5, 5</td>
</tr>
<tr>
<td>80</td>
<td>6</td>
<td>5, 5, 5, 5</td>
</tr>
<tr>
<td>80</td>
<td>8</td>
<td>5, 5, 5, 5</td>
</tr>
<tr>
<td>80</td>
<td>10</td>
<td>4, 4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>n</th>
<th>M</th>
<th>Problems solved</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>100</td>
<td>4</td>
<td>2, 1</td>
</tr>
<tr>
<td>100</td>
<td>6</td>
<td>2, 1</td>
</tr>
<tr>
<td>100</td>
<td>8</td>
<td>2, 2</td>
</tr>
<tr>
<td>100</td>
<td>10</td>
<td>2, 2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CPU sec. (IBM 3032)</th>
</tr>
</thead>
<tbody>
<tr>
<td>avg setup hratic</td>
</tr>
<tr>
<td>avg rem</td>
</tr>
<tr>
<td>avg total</td>
</tr>
</tbody>
</table>

**Table 4 - Computational Results for Symmetric Euclidean M-Travelling Salesman Problems**

Parameters: \( h = 1.0, \quad g = \begin{cases} 0.0005 & \text{when } n = 30 \\ 0.0015 & \text{when } 60 \leq n < 100 \end{cases} \)
### TABLE 5
Comparison of Computational Results for Symmetric M-Travelling Salesman Problems

<table>
<thead>
<tr>
<th>No. Cities</th>
<th>Algorithm</th>
<th>Number of Salesmen M</th>
<th>Problem Type</th>
<th>Number of Problems Attempted</th>
<th>Number of Problems Solved</th>
<th>Reason for Failure to Solve</th>
<th>Average No. of Iterations</th>
<th>Average No. of Nodes in B&amp;B</th>
<th>Average Total CPU sec.</th>
<th>Computer Used</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>GS</td>
<td>2 - 10</td>
<td>EU</td>
<td>25</td>
<td>25</td>
<td></td>
<td>289</td>
<td>10</td>
<td>2.4</td>
<td>IBM 3032</td>
</tr>
<tr>
<td>30</td>
<td>LN</td>
<td>variable</td>
<td>&quot;</td>
<td>5</td>
<td>5</td>
<td></td>
<td>NA</td>
<td>7.2</td>
<td>10.6</td>
<td>CYBER</td>
</tr>
<tr>
<td>30</td>
<td>AK</td>
<td>1 - 6</td>
<td>&quot;</td>
<td>12</td>
<td>12-</td>
<td>NA</td>
<td>8944</td>
<td>109.4</td>
<td>CDC 6600</td>
<td></td>
</tr>
<tr>
<td>60</td>
<td>GS</td>
<td>2 - 10</td>
<td>&quot;</td>
<td>25</td>
<td>25</td>
<td></td>
<td>11082</td>
<td>386</td>
<td>185.6</td>
<td>IBM 3032</td>
</tr>
<tr>
<td>60</td>
<td>LN</td>
<td>Variable</td>
<td>&quot;</td>
<td>5</td>
<td>5</td>
<td></td>
<td>NA</td>
<td>66.0</td>
<td>CYBER</td>
<td></td>
</tr>
<tr>
<td>60</td>
<td>AK</td>
<td>1 - 7</td>
<td>&quot;</td>
<td>2</td>
<td>2</td>
<td>NA</td>
<td>16055</td>
<td>904.7</td>
<td>CDC 6600</td>
<td></td>
</tr>
<tr>
<td>80</td>
<td>GS</td>
<td>2 - 10</td>
<td>&quot;</td>
<td>25</td>
<td>25</td>
<td>i.m.</td>
<td>19692</td>
<td>720</td>
<td>497.8</td>
<td>IBM 3032</td>
</tr>
<tr>
<td>80</td>
<td>LN</td>
<td>Variable</td>
<td>&quot;</td>
<td>5</td>
<td>2</td>
<td>i.m.</td>
<td>NA</td>
<td>283.0</td>
<td>CYBER</td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>GS</td>
<td>2 - 10</td>
<td>&quot;</td>
<td>10</td>
<td>10</td>
<td></td>
<td>10376</td>
<td>347</td>
<td>351.2</td>
<td>IBM 3032</td>
</tr>
<tr>
<td>60</td>
<td>GS</td>
<td>2 - 10</td>
<td>NU</td>
<td>50</td>
<td>50</td>
<td></td>
<td>620</td>
<td>19</td>
<td>14.2</td>
<td>IBM 3032</td>
</tr>
<tr>
<td>60</td>
<td>LN</td>
<td>Variable</td>
<td>&quot;</td>
<td>5</td>
<td>5</td>
<td></td>
<td>NA</td>
<td>45.2</td>
<td>CYBER</td>
<td></td>
</tr>
<tr>
<td>80</td>
<td>GS</td>
<td>2 - 10</td>
<td>&quot;</td>
<td>50</td>
<td>50</td>
<td>i.m.</td>
<td>573</td>
<td>28</td>
<td>27.0</td>
<td>IBM 3032</td>
</tr>
<tr>
<td>80</td>
<td>LN</td>
<td>Variable</td>
<td>&quot;</td>
<td>5</td>
<td>5</td>
<td></td>
<td>NA</td>
<td>137.0</td>
<td>CYBER</td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>GS</td>
<td>2 - 10</td>
<td>&quot;</td>
<td>50</td>
<td>50</td>
<td></td>
<td>632</td>
<td>19</td>
<td>33.0</td>
<td>IBM 3032</td>
</tr>
<tr>
<td>100</td>
<td>LN</td>
<td>Variable</td>
<td>&quot;</td>
<td>5</td>
<td>3</td>
<td>i.m.</td>
<td>NA</td>
<td>208.3</td>
<td>CYBER</td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>AK</td>
<td>1 - 9</td>
<td>&quot;</td>
<td>50</td>
<td>15</td>
<td>i.m.</td>
<td>NA</td>
<td>8</td>
<td>80.5</td>
<td>CDC 6600</td>
</tr>
<tr>
<td>250</td>
<td>GS</td>
<td>2 - 10</td>
<td>&quot;</td>
<td>20</td>
<td>20</td>
<td></td>
<td>1002</td>
<td>34</td>
<td>229.0</td>
<td>IBM 3032</td>
</tr>
</tbody>
</table>

a - GS: Gavish-Srikanth algorithm, described in this paper.

LN: Laporte-Nobert algorithm, described in [22].

AK: Ali-Kennington algorithm, described in [1].

b - The IBM 3032 is estimated to be 4-6 times slower than the CYBER or the CDC 6600.

c - insufficient memory.

d - NU - Non-euclidean cost matrix.

EU - Euclidean cost matrix.
The Laporte-Nobert algorithm was implemented on a CYBER Computer (approximately 4 to 6 times faster than an IBM 3032); euclidean problems with up to 80 cities and non-euclidean problems with up to 100 cities were attempted. The Ali-Kennington algorithm was implemented on a CDC 6600 computer (approximately 4 to 6 times faster than an IBM 3032); euclidean problems with up to 59 cities and non-euclidean problems with up to 100 cities were attempted.

As problem sizes increased, the number of problems that these earlier algorithms were able to solve decreased markedly due to insufficient storage space, e.g., for non-euclidean data sets with n=100, the Laporte-Nobert algorithm solved 30% of all attempted problems. The algorithm presented here did not suffer from this problem; it uses a depth-first Branch and Bound procedure and so the extra storage space required for implementing the branching tree consists of a single path with a limited length.

The performance of the algorithm appears to be significantly superior to that of the other two algorithms in the case of non-euclidean distance matrices. For euclidean distance matrices, it seems to be competitive with the Laporte-Nobert algorithm and significantly faster than Ali-Kennington's algorithm. The non-euclidean problem sizes which the algorithm can handle are, by a factor of 4, larger than the largest problems solved by either of the other algorithms.
7. EXTENSIONS TO THE ALGORITHM

In this section possible extensions of the problem are presented; these extensions can be handled by minor modifications of the algorithm.

7.1 M Free vs. Fixed

Consider the MTSP stated in Section 2. Transforming the problem to one where $M$ is also a variable, with an upper bound $m$, can be stated as:

Problem P2

Find a variable $M$ and binary variables $x_{ij}, \delta_i$ that satisfy

$$Z = \min \min_M \left\{ \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} c_{ij} x_{ij} + \sum_{i \in S} c_{il} x_{il} \right\}$$

subject to constraints (2) - (11) and

$$1 \leq M \leq m$$

$$M \text{ integer}$$

The solution procedures stay substantially the same; the major difference is that for a given Lagrangean vector $\lambda$, several different solutions to Problem LP1 need to be computed, with $M$ set equal to different values in the range $1 \leq M \leq m$. 
The solution values \( \tilde{Z}_1 \) and \( \tilde{Z}_2 \) of problems LP1a and LP1b are convex functions of \( M \). For a given \( v \) vector, then, we need to compute solutions for problem LP2 for values of \( M \) from 1 to \( k \), where \( k \) is defined by

\[
\begin{aligned}
k &= \text{Min} \left\{ \min_{\ell} \left( \tilde{Z}_1 + \tilde{Z}_2 \mid \min_{M=\ell-1} \tilde{Z}_1 + \tilde{Z}_2 \geq \min_{M=\ell} \tilde{Z}_1 + \tilde{Z}_2 \right) \right\}
\end{aligned}
\]  

(36)

Computing \( k \) different solutions for problem LP1a (i.e., \( k \) minimal spanning trees where the degree of node 1 is constrained in turn to \( 1, 2, \ldots, k \)) requires \( O(n^2 + k \cdot n) \) computations [16]; computing \( k \) different solutions for problem LP1 requires \( O(n \log n) \) computations. Sensitivity analysis procedures are more complicated than before, but follow the same general principles. The algorithm can therefore be modified to treat \( M \) as a variable without a substantial degradation of performance.

7.2 Bounds on the Number of Immediate Subtours

Let \( U \) be the desired upper bound on the number of immediate subtours in the final solution. Constraint (10) can then be modified to

\[
\sum_{j \in S} e_j \leq U
\]

(37)

The sole effect of this changed constraint is on the solution
procedure for subproblems LP1b and LP1c: Select the $U$ shortest arcs in the arc set $I = \{(i,1): i\in S\}$. From the remaining arcs in $I$, choose in turn the $M-U$ shortest arcs whose inclusion does not lead to the formation of $U+1$ immediate subtours. The optimal values of $\alpha^*$ and $\delta^*$ can be computed as before.

A lower bound on the number of immediate subtours can be similarly accommodated.
8. CONCLUSION

The algorithm demonstrates good computational performance, especially in the case of non-euclidean data. The fact that the integer gap appears to decrease as \( n \) increases holds promise for good computational performance on even larger problems.

In the larger problems, most of the computing time was consumed in computing the constrained minimal spanning trees. The algorithm currently in use is a modification of the Glover-Klingman algorithm, which requires at least \( O(n^2) \) operations. By performing Type 2 sensitivity analysis on all the arcs and exploiting the resulting graph sparsity by using an algorithm by Yao [34], we can compute the spanning trees in \( O(|A| \log \log n) \) operations. The preliminary computational experience indicates that \(|A| = O(n)\) leading to significantly improved computing times, without affecting the number of nodes or iterations involved in the Branch and Bound procedure. Work is currently in progress on the inclusion of these modifications in the solution procedure and will be reported in a forthcoming paper.
9. REFERENCES


32. Svetska, J.A., "Response to 'A Note on the Formulation of the M-Salesman Travelling Salesman Problem'"; Management Science v. 22, no. 6, (1976), pp. 706-

