AN OPTIMAL SOLUTION METHOD FOR LARGE-SCALE MULTIPLE TRAVELING SALESMEN PROBLEMS

BEZALEL GAVISH
University of Rochester, Rochester, New York

KIZHANATHAN SRIKANTH
University of Illinois, Chicago, Illinois

(Received June 1983; revision received August 1984; accepted February 1986)

We develop an efficient branch-and-bound based method for solving the Multiple Traveling Salesman Problem, and develop lower bounds through a Lagrangean relaxation that requires computing a degree-constrained minimal spanning tree. A subgradient optimization procedure updates the Lagrange multipliers. We use fast sensitivity analysis techniques to increase the underlying graph sparsity and reduce the problem size. The algorithm was tested on 416 problems with up to 500 cities and 10 salesmen. We also present computational results on different data sets and parameters in order to identify the major factors that influence the performance of the developed code.

The Multiple Traveling Salesmen Problem (MTSP) is an extension of the well-known Traveling Salesmen Problem (TSP). MTSP can itself be generalized to a wide variety of routing and 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, Angel et al. 1972, Christofides and Eilon 1969, Gavish 1982, 1983, and Gavish and Srikanth 1979). Finding a good optimal solution method for the Multiple Traveling Salesman Problem is therefore of importance, particularly if the method can be applied to some of its extensions.

The MTSP 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, so that the total distance covered is minimized, subject to the constraint that each city (apart from city 1) is visited once by one and only one salesman.

The number of salesmen M may 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 more difficult problems mentioned previously.

We first define some terms we will frequently use.

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

S: $= V - \{1\}$.

c_{ij}: the distance from node i to node j. $C = \{c_{ij}\}$ is the matrix of c_{ij} values. If $c_{ji} = c_{ij}$ for all $i, j \in V$, C is symmetric. In this paper, C is called euclidean if it satisfies the euclidean norm. If this relationship is violated, then C is said to be non-euclidean.

E: the arc set $\{(i, j)\colon 1 \leqslant i < j \leqslant n\}$.

subtour: a set of k arcs $\{(i_1, i_2), (i_2, i_3), \ldots, (i_k, i_1)\}$ with $i_p \neq i_q$, $p \neq q$ and for all $1 \leqslant p, q \leqslant k$, is said to form a subtour (or cycle) of size k.

immediate subtour: the two arcs $\{(1, i), (i, 1)\}$ 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 travels from city 1 to city i and returns directly from city i to city 1.

$\lfloor x \rfloor$ = largest integer that is less or equal to x.

$\lceil x \rceil$ = smallest integer that is greater or equal to x.

Existing solution methods for the MTSP are, for the most part, extensions of algorithms developed for solving TSPs. Most existing solution methods for the TSP work by relaxing one or more of three major classes of constraints:

(i) *degree constraints*: require that every city (except city 1) must have exactly one arc entering it and one arc leaving it. When C is symmetric and $M = 1$, relaxing these constraints yields a *min-*

Subject classification: 491 Lagrangean relaxation and branch-and-bound, 627 traveling salesman problem.

0030-364X/86/3405-0698 $01.25

imal 1-*tree* problem (Held and Karp 1970, 1971; Helbig-Hansen and Krarup 1974).

(ii) *subtour elimination constraints*: eliminate any solution containing a subtour that does not include all cities. This relaxation yields an *assignment problem* (Christofides 1979, Balas and Christofides 1981, and Carpaneto and Toth 1980).

(iii) *integrality constraints*: require that all variables in the problem be integer. This relaxation yields a *linear programming* problem (Miliotis 1976, 1978; Crowder and Padberg 1980, Laporte and Nobert 1980).

Bellmore and Hong (1974) showed that for a given value of n, an n-city, M-salesman problem can be transformed to an $(n + M - 1)$-city, single-salesman TSP by adding M-1 duplicates of row and column 1 to the distance matrix. Svetska and Huckfeldt (1973) employed a similar transformation and solved asymmetric problems involving up to 60 cities and 10 salesmen, using relaxation to an assignment problem and subsequent enumeration techniques. Ali and Kennington (1980) developed a Lagrangean relaxation of the degree constraints and used a minimal M-forest algorithm to solve a small subset of problems, with sizes up to $n = 100$. Laporte and Nobert developed an algorithm for the MTSP (with M optionally variable or fixed) 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 has been conducted by Gavish (1976), Miller, Tucker and Zemlin (1960), Orloff (1974), Rao (1980), and Russell (1977).

In this paper, we develop an algorithm for solving MTSPs to optimality with C symmetric and M constant. In the next section, we develop an integer linear programming formulation of the problem which is the basis for the relaxations and solution procedures described in Section 2. Section 3 describes a sensitivity analysis technique for spanning tree structures that was found to be highly effective in reducing the size of the original problem. Section 4 describes the branch-and-bound procedure, and Section 5 discusses the computational results. Possible extensions of the algorithm are discussed in Section 6.

1. Formulation of the Problem

The integer linear programming formulation of the symmetric MTSP with a fixed number (M) of salesmen is

Problem P

Find binary variables x_{ij} that satisfy

$$Z^* = \min_x \left\{ \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} c_{ij}x_{ij} + \sum_{i \in S} c_{i1}x_{i1} \right\} \tag{1}$$

subject to

$$\sum_{j \in S} x_{1j} = M, \tag{2}$$

$$\sum_{i \in S} x_{i1} = M, \tag{3}$$

$$x_{j1} + \sum_{i<j} x_{ij} + \sum_{i>j} x_{ji} = 2 \quad \text{for all } j \in S, \tag{4}$$

$$\sum_{\substack{i,j \in S_k \\ i<j}} x_{ij} \leqslant |S_k| - 1 \quad \text{for all } S_k \subseteq S, \tag{5}$$

$$\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} x_{ij} = n - 1, \tag{6}$$

$$x_{ij} = 0, 1 \quad \text{for } 1 \leqslant i < j \leqslant n, \tag{7}$$

$$x_{i1} = 0, 1 \quad \text{for all } i \in S. \tag{8}$$

$x_{ij} (2 \leqslant i < j \leqslant n)$ is a decision variable that is equal to one if a salesman travels directly from city i to city j or from city j to city i. $x_{1i}(x_{i1})$ $i \in S$ is equal to one if a salesman travels directly from city $1(i)$ to city $i(1)$. x_{ij} is equal to zero otherwise.

The constraints in (2) and (3) ensure that exactly M salesmen depart from and return to city 1. The constraints in (4) ensure that any other city is visited by one and only one salesman. Constraint (5) ensures that no subtours can be formed among the nodes in S. Constraint (6) follows from the fact that any feasible solution to the problem contains $n + M - 1$ arcs, of which M arcs represent salesmen returning to city 1; this constraint is redundant for this formulation, but is useful in later relaxations of Problem P.

2. Lower Bounding Procedures

A lower bound on the optimal solution to Problem P is generated by multiplying the constraints in (4) by a vector of Lagrange multipliers $\pi = \{\pi_2, \pi_3, \ldots, \pi_n\}$ and adding them to the objective function. This op-

eration gives the following Lagrangean problem:

Problem Q

$$L(\pi) = \min\left\{\sum_{i=1}^{n-1}\sum_{j=i+1}^{n} c_{ij}x_{ij} + \sum_{i=2}^{n} c_{i1}x_{i1} + \sum_{j=2}^{n} \pi_j\left(2 - \sum_{i=1}^{j-1} x_{ij} - \sum_{i=j+1}^{n} x_{ji} - x_{j1}\right)\right\}$$

subject to (2), (3), (5–8).

After rearrangement of terms, the Lagrangean problem can be rewritten as

$$L(\pi) = \min\left\{\sum_{i=1}^{n-1}\sum_{j=i+1}^{n} \tilde{c}_{ij}x_{ij} + \sum_{i=2}^{n} \tilde{c}_{i1}x_{i1} + \sum_{j=2}^{n} 2 \cdot \pi_j\right\}$$

subject to (2), (3), (5–8),

where $\tilde{c}_{ij} = c_{ij} - \pi_i - \pi_j$ with π_1 set equal to zero.

The modified cost matrix $\tilde{C}$ remains symmetric, and therefore, for a fixed set of multipliers, the Lagrangean problem can be solved by the following algorithm.

Algorithm 1 (Solving the Lagrangean Problem Q)

Step 1. Solve a degree-constrained minimal spanning tree problem, spanning over the nodes in V, using $\tilde{C}$. Node 1 is restricted to have exactly M edges adjacent to it. Let the cost of this tree be $W(\pi)$.

Step 2. Using the cost matrix C, select the M minimal cost edges adjacent to node 1 as the return links for the M salesman. Let $\cup(\pi)$ denote the sum of these link costs.

Step 3. $L(\pi) = W(\pi) + \cup(\pi) + 2 \cdot \sum_{j=2}^{n} \pi_j$.

The time complexity of Step 1 is $O(n^2 + Mn)$ (Srikanth 1980). Step 2 can be done in $O(n)$ operations, leading to an overall time complexity of $O(n^2 + Mn)$.

This bound can be improved slightly by noticing that, for $M < n - 1$, an optimal solution to the MTSP can have at most $M - 1$ *immediate cycles*. In preliminary computational experiments, we observed that in the majority of cases tested, the edges selected in Step 2 of the algorithm formed M immediate cycles (i.e., a cycle that consists of node 1 and one more node) when combined with the edges in the tree. Therefore, Step 2 was modified to:

Step 2A. a) Select the M minimal cost edges adjacent to node 1.

b) If the M edges selected form, together with the spanning tree edges, at most $M - 1$ immediate cycles, then go to Step 3. Otherwise, replace the highest-cost edge among the M edges selected in a) with the $(M + 1)$th lowest cost edge adjacent to node 1. Denote the sum of costs of those M links by $\cup(\pi)$.

This modification has the same time complexity as the original algorithm, but generates tighter lower bounds. We let $\tilde{L}(\pi)$ denote the objective function of the Lagrangean obtained by the modified algorithm. $\tilde{L}(\pi)$ is a lower bound on Z^*. We are interested in the set of multipliers π^* that generates the tightest possible bound, i.e., $\tilde{L}(\pi^*) = \max_\pi\{\tilde{L}(\pi)\}$.

In order to obtain a good set of Lagrange multipliers π, we used a subgradient optimization procedure. This procedure is one of the more successful methods for obtaining good Lagrange multiplier values, as detailed in the paper by Held, Wolf and Crowder (1974). It is an iterative procedure that computes the subgradient directions at the k^{th} iteration as follows:

$$\gamma_j^k = 2 - \left(x_{j1} + \sum_{i=1}^{j-1} x_{ij} + \sum_{i=j+1}^{n} x_{ji}\right) \quad \text{for all} \quad j \in S.$$

Using a step size t_k, the method computes the new set of multipliers π^{k+1} as

$$\pi_j^{k+1} = \pi_j^k + t_k\gamma_j^k,$$

where

$$t_k = \frac{\lambda_k(\bar{Z} - \tilde{L}(\pi^k))}{\|\tilde{\gamma}^k\|^2}.$$

$\bar{Z}$ is an approximation of Z^*, the value of the optimal solution to Problem P. If $\bar{Z}$ is equal to $\tilde{L}(\pi^*)$, the sequence of $\{\pi_j^k\}$ is guaranteed to converge to optimality as $k \to \infty$; for computational convenience, however, $\bar{Z}$ is usually set equal to an upper bound on Z^*. While this choice does not guarantee convergence, we experienced no convergence problems in any of the large number of problems tested. λ_k is a user-determined step size; typically, its starting value is 2, and this value is halved every few (10–20) successive iterations in which there is no improvement in $\tilde{L}(\pi^*)$.

Outline of the Lower Bounding Procedure

The flowchart in Figure 1 depicts the first half of the MTSP algorithm. Its most important steps can be described in greater detail as follows.

1. Using a heuristic procedure, generate a feasible solution to the MTSP; use this value as the upper bound $\bar{Z}$ on the optimal solution value that is required for the subgradient procedure. (We used

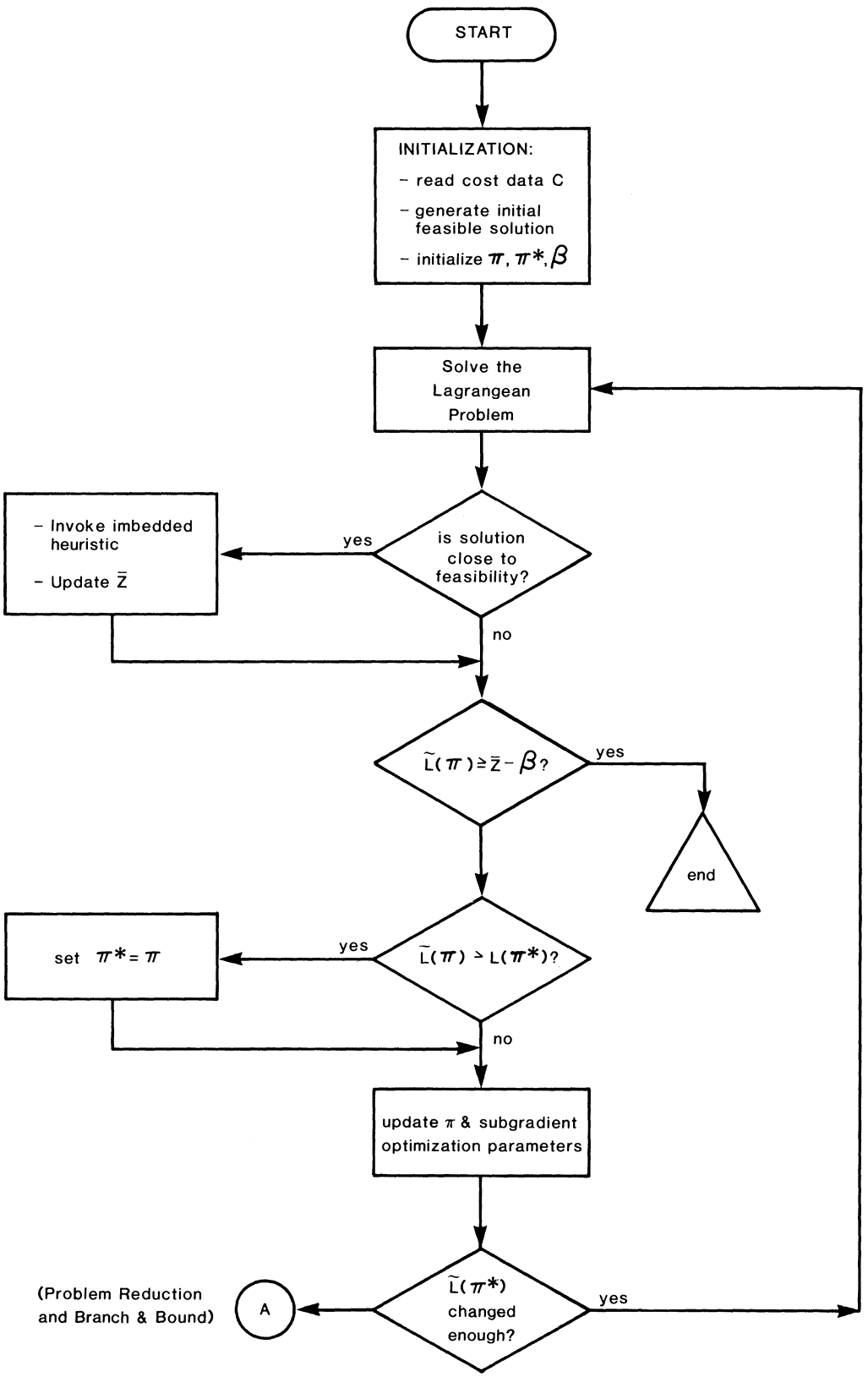

Figure 1. Flowchart of the MTSP algorithm up to the branch-and-bound stage.

a successive node insertion procedure similar to the one used by Lin and Kernighan 1973.) Initialize the Lagrangean vector π. (We obtained good results by initializing π_i for all $i \in S$ to be equal to the second minimum in $\{c_{ij}: j = 1, 2, \ldots, n, j \neq i\}$.) Set the parameter β to one if the cost data is integer, and to zero otherwise.
2. Compute the optimal solution to the Lagrangean problem.
3. If, in the optimal solution to the Lagrangean problem, the number of nodes not at their required degree is less than a given parameter, use a heuristic procedure to generate a feasible solution to Problem P, starting from the solution to the Lagrangean problem. If the associated cost of this procedure improves on the best-known feasible solution, update the value of $\bar{Z}$.
4. Terminate if $\tilde{L}(\pi) \geq \bar{Z} - \beta$; otherwise, continue with Step 5.
5. Update the vector π, using the subgradient optimization procedure.
6. Repeat Steps 2 through 5 until no further significant improvement in the lower bound value occurs.
7. Perform sensitivity analysis on the best Lagrangean problem generated. Reduce the problem size by fixing arcs to zero or one whenever possible. (Section 3 gives details of the problem reduction procedure.)
8. If the lower bound value equals the best feasible solution value known, stop. Otherwise, use a branch-and-bound procedure to generate and verify the optimal solution.

The efficiency of the reduction and branch-and-bound procedures depends to a large extent on the quality of the lower bounds and the ability to perform sensitivity analysis quickly on the best Lagrangean solution. The following sections describe the details of this procedure and the implementation of the branch-and-bound procedure.

3. Reduction in Problem Size

Reduction procedures have been found to be extremely useful on some classes of combinatorial optimization problems (Crowder and Padberg; Crowder, Johnson and Padberg 1983 and Gavish and Pirkul (1985a, b)). In this section, we develop and evaluate methods for performing sensitivity analysis on the solutions to Problem Q generated by the subgradient optimization procedure. Given an arbitrary π vector, consider the effect on $\tilde{L}(\pi)$ of setting to zero a variable x_{ij} that has a value of 1 in the optimal solution to Problem Q. If the resulting change in $\tilde{L}(\pi)$ is such that the new value of $\tilde{L}(\pi)$ exceeds $\bar{Z}$, the best-known upper bound on Z^*, then it is evident that the optimal solution to Problem P must have x_{ij} set equal to one. We can therefore solve Problem P, or any relaxation of it, with variables of this type set equal (or forced) to one.

At the start of the branch-and-bound procedure, sensitivity analysis of this type is performed on the relaxed Problem Q, using π^*. Any variables forced to one as a result of this sensitivity analysis can be set to one throughout the branch-and-bound procedure. Sensitivity analysis is also useful at any level of the branch-and-bound tree, where additional sets of variables can be forced to one in the appropriate subproblems; here, it is possible to identify variables that must be part of an optimal solution to the modified Problem P, i.e., the original Problem P, with the additional branch-and-bound variables fixed at zero or one.

It is worthwhile noting the following problem features.

1. Any subset of arcs forced to be in the solution that forms a path can be replaced by a single arc of appropriate length, and all interior nodes in the path removed from further consideration, thus reducing the problem size n. For example, if the variables corresponding to arcs (10, 2), (2, 15), (15, 3) are forced to 1, and $c_{10,2} + c_{2,15} + c_{15,3} = 218$, nodes 2 and 15 can be removed from the problem, and $c_{10,3}$ updated to 218.
2. If any subset of arcs forced to be in the solution forms a subtour (necessarily including node 1), a smaller problem can be generated as follows: Add the sum of the costs of the arcs in this subtour to the objective function as a constant, remove from further consideration all nodes in the subtour except node 1, and reduce M by 1.
3. Even if the sensitivity analysis does not force any variable to 1, knowledge of the relative magnitudes of changes in the solution value is valuable in choosing separation variables during the branch-and-bound procedure.

A solution to Problem Q can be represented by an arc set B that consists of a minimal spanning tree, together with M arcs returning to node 1. One way of obtaining the necessary sensitivity analysis information is by implicit enumeration (i.e., setting, in turn, $c_{ab} = \infty$ for each arc (a, b) in B, and solving the new problem Q), which would require $O(n^3)$ operations. However, Gavish and Srikanth (1980b) have shown that sensitivity analysis of the type discussed above can be performed simultaneously for all $n - 1$ arcs in a degree-constrained minimal spanning tree in $O(n^2)$

Table I
Effects of Sensitivity Analysis on Problem Characteristics of Symmetric Non-Euclidean Datasets
(for Problems with a Positive Integer Gap)

Original Problem Size		No. of Problems		New Problem Size		n'/n	M'/M	Average Time for Sensitivity Analysis (sec)
n	M	Solved	With Integer Gap	n'	M'			
150	2	5	2	99	2	66%	100%	0.13
	4	5	2	84	2	56	50	
	6	5	2	86	2	57	25	
	8	5	4	89	3	59	31	
	10	5	2	108	3	72	25	
200	2	5	1	31	1	16	50	0.25
	4	5	0	—	—	—	—	
	6	5	2	172	3	86	42	
	8	5	2	145	4	73	50	
	10	5	3	110	2	55	20	
250	2	4	0	—	—	—	—	0.4
	4	4	0	—	—	—	—	
	6	4	0	—	—	—	—	
	8	4	2	169	2	68	25	
	10	4	2	109	2	44	20	

operations. This procedure requires computing just one extra spanning tree, versus the $(n - 1)$ required by implicit enumeration. Performing sensitivity analysis on the remaining M arcs requires $O(n + M)$ operations, leading to an overall complexity of $O(n^2)$.

Sensitivity analysis was incorporated as part of the general solution procedure. Tables I and II show the reduction in problem size when sensitivity analysis is used at the start of the branch-and-bound procedure. Table I shows the results for non-euclidean data sets, while Table II gives the outcomes for euclidean data sets. The non-euclidean data sets consist of random integers uniformly distributed in the range [0, 400]; the coordinates for the euclidean data sets are random numbers in the range [0, 400], from which the distances between points were computed and rounded off to the nearest integer. Based on those computational experiments, it is clear that this approach leads to significant reductions in problem size, especially for non-Euclidean problems.

Other types of sensitivity analysis of the relaxed Problem Q are also possible, e.g., considering the effect of setting to one x_{ij} variables that are equal to zero in the current optimal solution, and forcing such vari-

Table II
Effects of Sensitivity Analysis on Problem Characteristics of Symmetric Euclidean Datasets
(for Problems with a Positive Integer Gap)

Original Problem Size		No. of Problems		New Problem Size		n'/n	M'/M	Average Time for Sensitivity Analysis (sec)
n	M	Solved	With Integer Gap	n'	M'			
60	2	5	5	56	2	93%	100%	0.02
	4	5	5	58	3	97	75	
	6	5	5	54	4	90	67	
	8	5	5	53	5	88	63	
	10	5	4	54	7	90	70	
80	2	5	5	77	2	96	100	0.03
	4	5	5	76	3	95	75	
	6	5	5	74	4	93	67	
	8	5	5	76	6	95	75	
	10	4	4	75	8	94	80	

ables to zero if the change in solution value is large enough. This type of sensitivity analysis must be performed on $((n^2 - n)/2 - (n + M - 1))$ variables, each requiring the solution of a degree constrained minimal spanning tree. Gavish and Srikanth (1980b) show that sensitivity analysis for *all* those variables can also be performed in $O(n^2)$ operations, again by computing just one extra spanning tree. We also (1983) performed experiments on the TSP utilizing both types of sensitivity analysis and eliminated up to 98% of all arcs from the original graph, while allowing for the use of special spanning tree algorithms suited for sparse graphs; this approach cut computing times by a factor of 70–95%.

4. The Branch-and-Bound Procedure

After obtaining the possible reductions in problem size, we utilized a depth-first, branch-and-bound procedure (see the flowchart, Figure 2). The procedure has the following features. At any level of the branch-and-bound tree, a separation variable is selected and two subproblems are generated—one in which that variable is set equal to zero, and one in which it is set equal to one. For each subproblem, a limited number of subgradient iterations are performed, seeking to improve the value of the lower bound $\tilde{L}(\pi)$ for the restricted problem. The sequence of iterations is halted if one of the following situations occurs: (i) either fathoming is achieved; (ii) there is little relative improvement in $\tilde{L}(\pi)$; or (iii) the number of iterations exceeds a preset limit. The next node selected for branching is the one with the lowest lower-bound value among the two subproblems. Whenever a node is fathomed, backtracking is performed to the lowest-level node along the path to the fathomed node. The best feasible solution found at the end of the branch-and-bound procedure is the optimal solution to Problem P.

4.1 Use of an Artificial Upper Bound

The performance of any depth-first, branch-and-bound procedure is determined, to a large extent, by the quality of the upper bound $\bar{Z}$ used at the beginning of the branching procedure. When the value of $\bar{Z}$ is significantly higher than Z^*, a large number of nodes might be generated in the branching tree, leading to prohibitive computing times and excessive storage requirements. Unfortunately, it is sometimes extremely difficult for a heuristic procedure to identify a feasible solution that is close enough to the optimal solution.

In order to overcome this difficulty, it is possible to use an artificial upper bound as a substitute for the best known feasible solution (see Bazaraa and Elshafei 1977, Ali and Kennington). An artificial upper bound is a value that is less than the best-known feasible solution value, but is higher than the highest lower-bound value. Our implemenation uses an artificial upper-bound value Z_f for fathoming and problem reduction in the branch-and-bound process. If a feasible solution with a value not greater than Z_f is identified in the course of the branch-and-bound process, it replaces Z_f and the solution process can be continued to termination. In that case, the optimal solution to Problem P has been identified and verified. If the branch-and-bound process does not identify a feasible solution with a value better or equal to Z_f, then Z_f is clearly not an upper bound on Z^* and the branch-and-bound process must be repeated with an artificial upper bound that is greater than Z_f.

Setting low values for Z_f significantly reduces solution times for problems whose optimal solution value is less than or equal to Z_f. On the other hand, low Z_f values increase the number of problems requiring repetition of the branch-and-bound procedure, increasing the expected solution times for those problems. Assuming that the objective of an algorithm intended to solve an NP-complete problem is to minimize the expected computing times required to generate and verify the optimal solution, the Z_f value selected must reduce solution times for a high proportion of the problems computed.

This argument is further strengthened by the observation that in many real-world applications, the original cost data may be only an approximation of the true cost values. Users of such an algorithm might be satisfied with a provably good feasible solution that is not necessarily optimal, but must lie within an acceptable range from the lower bound. In such cases, the best feasible solution $\bar{Z}$ found during the failed branch-and-bound procedure may be within the acceptable range (i.e., $(\bar{Z} - Z_f)/Z_f \leq \epsilon$), and there is no need to repeat the branch-and-bound process.

After experimenting with several rules for selecting the artificial upper bound, we adopted the following formula:

$$Z_f = \begin{cases} \bar{Z} & \text{if } \bar{Z} \leq (1 + g)\tilde{L}(\pi^*) \\ \tilde{L}(\pi^*) + \min\{g \cdot \tilde{L}(\pi^*); h \cdot (\bar{Z} - \tilde{L}(\pi^*))\} & \\ & \text{otherwise.} \end{cases}$$

The values of h and g determine the magnitude of Z_f. After some experimentation, we found that good com-

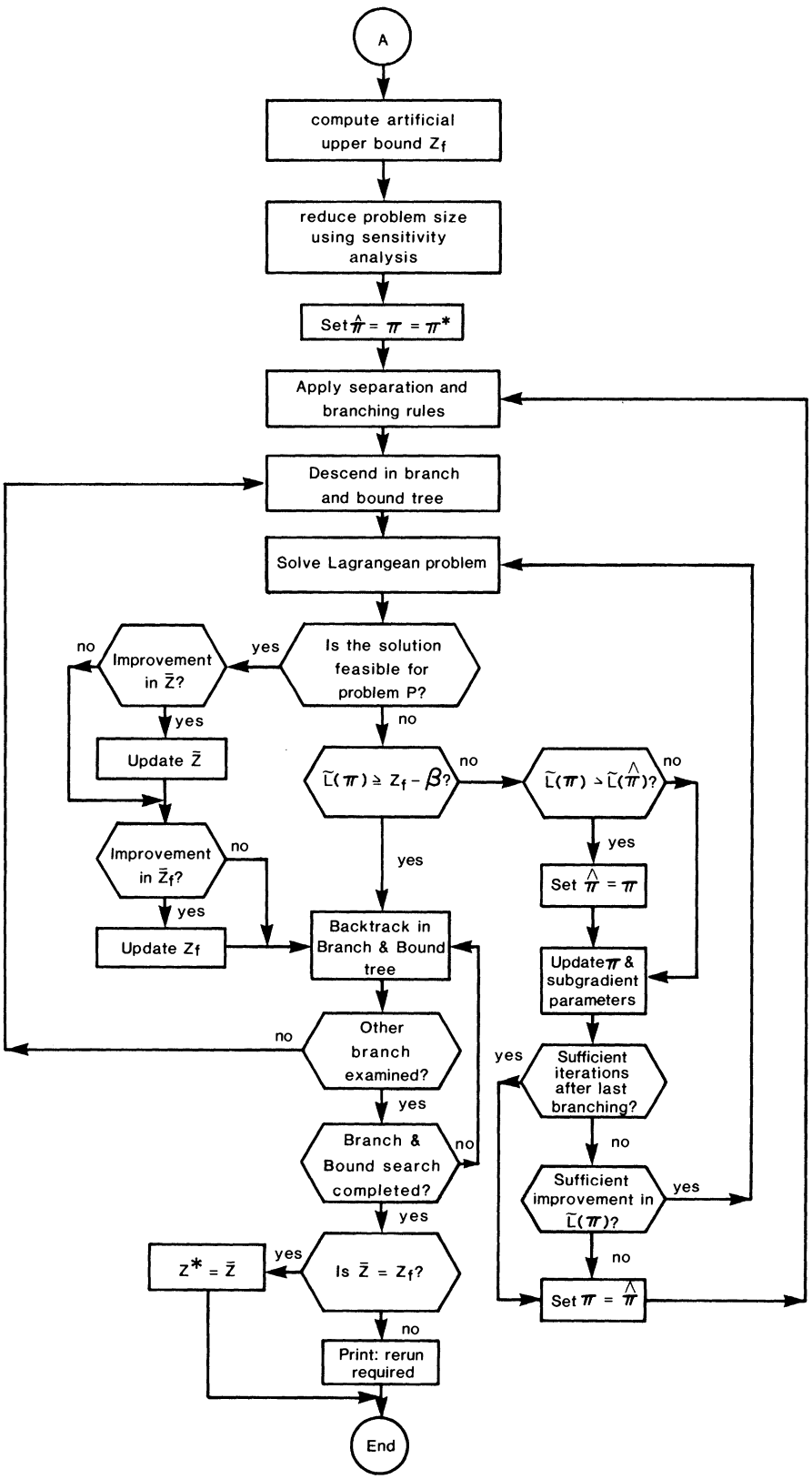

Figure 2. Flowchart of the branch-and-bound procedure.

Table III
Effect of Using an Artificial Upper Bound in the Branch-and-Bound Procedure for Symmetric, Non-Euclidean Problems

No. of Cities (n)	h[a]	g[a]	No. of Problems		Percent of Problems Solved with at Most One Pass through Branch and Bound
			Attempted	Requiring Repetition of Branch and Bound	
40	0.25	0.005	50	4	92
60	0.25	0.005	50	3	94
80	0.25	0.005	50	4	92
100	0.25	0.005	50	1	98
150	0.25	0.0025	25	0	96[b]
200	0.25	0.0025	25	3	88
250	0.25	0.0025	20	0	100
400	0.25	0.0025	5	0	100
Total			275	15	94.5

[a] h and g are defined in Section 4.1.
[b] One problem remained unsolved due to excessive computing time.

putational times are obtained by setting

$h = \begin{cases} 0.25 & \text{for non-euclidean distance matrices,} \\ 1.0 & \text{for euclidean distance matrices;} \end{cases}$

$g = 0.005$ to 0.0025, generally decreasing with the increase in problem size.

Table III shows the impact of using an artificial upper bound. Out of 275 problems attempted, this method solved over 94% without repetitions of the branch-and-bound procedure; all remaining problems required only a single repetition. This result clearly

Table IV
Computational Results for Small Symmetric Non-Euclidean Multiple Traveling Salesman Problems[a]

n	M	No. of Problems[b]				No. of Lagrangean Iterations						Percent Integer Gap[c]		No. of Branch-and-Bound (B&B) Nodes		
						Before B&B			Total							
		a	b	c	d	Min.	Avg.	Max.	Min.	Avg.	Max.	Avg.	Max.	Min.	Avg.	Max.
40	2	10	10	4	2	22	116	228	22	191	496	.14	.48	0	4	16
	4	10	10	8	0	30	82	208	30	96	289	.03	.19	0	1	6
	6	10	10	5	1	31	123	226	31	233	665	.16	.74	0	7	32
	8	10	10	5	1	31	139	253	31	302	939	.14	.62	0	10	46
	10	10	10	2	0	28	85	229	28	115	390	.01	.07	0	2	12
60	2	10	10	6	1	97	195	289	97	464	1286	.16	.77	0	14	54
	4	10	10	4	1	85	172	273	85	323	1269	.10	.76	0	8	52
	6	10	10	6	2	101	193	260	101	904	3607	.13	.47	0	26	148
	8	10	10	9	0	109	232	284	109	879	2675	.16	.35	0	30	108
	10	10	10	6	1	101	196	292	101	529	2220	.14	.80	0	17	99
80	2	10	10	7	0	83	220	292	83	780	1616	.10	.31	0	25	60
	4	10	10	8	0	138	232	281	138	1374	6256	.12	.47	0	54	282
	6	10	10	8	0	77	220	265	77	512	1437	.08	.22	0	14	52
	8	10	10	7	2	93	213	259	93	580	1841	.17	.65	0	19	72
	10	10	10	8	0	122	222	268	122	767	1546	.08	.19	0	27	58
100	2	10	10	7	0	123	232	311	123	863	3330	.05	.26	0	27	125
	4	10	10	8	0	87	234	290	87	423	1311	.06	.39	0	10	47
	6	10	10	7	0	191	249	292	191	832	3625	.11	.30	0	28	151
	8	10	10	8	1	92	228	274	92	587	1215	.07	.22	0	17	44
	10	10	10	5	0	112	216	284	112	454	1731	.03	.18	0	11	64

[a] Parameters of computing artifical upper bound (AUB): $h = 0.25$, $g = 0.0050$.
[b] Key: a, total number of problems attempted; b, number solved to optimality; c, number of problems solved to optimality without use of branch and bound; and d, of problems that required repeat of branch and bound (i.e., artificial upper bound

demonstrates the usefulness of the approach for the MTSP.

4.2. Separation and Branching Rules

Two other factors affecting the performance of branch-and-bound procedures are the separation rules and the branching rules that are used as part of the procedure. At any stage of the branch-and-bound tree, a separation variable must be selected. Knowledge of the effect on the solution value or structure of fixing a particular variable in the solution to zero or one is useful in determining the separation variables.

Define

B: $\{b_1, b_2, \ldots, b_{n+M-1}\} = \{(i_1, j_1), (i_2, j_2), \ldots, (i_{n+M-1}, j_{n+M-1})\}$ as the set of edges in a particular solution to Problem Q.

W_k: is the variable corresponding to edge b_k, $b_k \in B$, in Problem Q ($W_k = x_{i_k j_k}$).

L_k: $= \{i \mid i = i_k \text{ or } i = j_k, i \neq 1\}$.

p_k: change in solution value if W_k is set to zero.

q_k: change in solution value if W_k is set to one.

d_i: degree of node i in B.

Table IV
(Continued)

CPU (IBM 3032) Time (sec)					
Average			Total		
Setup	Heuristic	Remainder	Min.	Avg.	Max.
0.5	1.9	2	1.8	4.0	6.9
0.5	2.0	1	2.4	3.6	5.9
0.5	1.9	2	2.9	5.4	9.0
0.5	3.6	3	3.5	6.7	13.0
0.5	3.6	1	3.5	5.5	8.6
0.7	1.0	7	3	8	20
0.7	1.9	5	3	8	24
0.8	4.4	13	3	18	60
0.8	7.9	12	5	21	49
0.8	7.6	8	6	16	39
1.2	1.3	19	4	21	42
1.2	3.1	35	7	39	183
1.2	5.1	13	5	20	37
1.2	7.8	16	6	25	61
1.3	10.1	19	7	30	50
1.8	2.3	30	9	34	122
1.8	3.6	18	8	23	52
1.9	9.4	30	19	42	126
1.8	9.8	22	8	34	54
1.8	12.1	18	9	32	81

chosen at start of branch and bound was too low).

[c] Percent integer gap = 100 $(Z^* - w^*)/Z^*$. Minimum was 0.0 for all problem sizes.

N_k: number of variables in B forced to zero if W_k is fixed to one.

The p_k values can be computed for all edges $b_k \in B$ in $O(n^2 + Mn)$ operations. In computing a q_k value, we must take into consideration the fact that fixing W_k to one can change the solution structure (and possibly therefore the solution value) only if the following conditions are satisfied:

(i) W_k has not been set to one earlier in the branch-and-bound tree, and
(ii) there exists at least one node $j \in I_k$ with $d_j > 2$, and at least one edge incident at j (other than b_k) whose corresponding variable has been set earlier to one in the branch-and-bound tree.

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 setting W_k to one will not change the solution structure or value, and $q_k = 0$. If $N_k = 1$ and W_j is the variable forced to zero, then $q_k = p_j$, which is already computed. If $N_k > 1$, then to find the value of q_k, we must compute a new solution to Problem Q with W_k fixed to 1, which requires $O(n^2)$ operations; however, this situation occurs infrequently.

To summarize, all p_k and q_k values for a given solution B can be computed in $O(n^2)$ operations.

The p_k and q_k values are used to identify variables that have a high probability of being included/excluded in the optimal solution. The separation rule that led to the best computational results was to choose the variable W_k that met the following criteria, in decreasing order of importance:

(i) highest value of $\lceil q_k + \tilde{L}(\pi) \rceil$,
(ii) $N_k > 0$?
(iii)
$$\begin{cases} \max[0, d_{j_k} - 2] + \max[0, d_{i_k} - 2], & \text{if } i_k \neq 1, \quad j_k \neq 1, \\ \max[0, d_{j_k} - 2], & \text{if } i_k = 1, \\ \max[0, d_{i_k} - 2], & \text{if } j_k = 1, \end{cases}$$
(iv) highest value of $\lceil p_k + \tilde{L}(\pi) \rceil$,
(v) highest value of $p_k + \tilde{L}(\pi)$.

The "ceiling" function $\lceil\ \rceil$ is used only if the original distance matrix is all-integer.

If q_k is relatively large, the probability that W_k will be in the optimal solution is low; similarly, if p_k is relatively large, the probability that W_k will not be in the optimal solution is low. Therefore, once we choose W_k as the separation variable, the branching rule is to set the variable W_k to

$$W_k = \begin{cases} 1 & \text{if } \lceil \tilde{L}(\pi) + q_k \rceil \leq \lceil p_k + \tilde{L}(\pi) \rceil, \\ 0 & \text{otherwise.} \end{cases}$$

Table V
Computational Results for Large Symmetric Non-Euclidean Multiple Traveling Salesman Problems[a]
(CPU Times on an IBM 3032)

		No. of Problems[b]				No. of Lagrangean Iterations						Percent Integer Gap[c]		No. of Branch-and-Bound (B&B) Nodes		
						Before B&B			Total							
n	M	a	b	c	d	Min.	Avg.	Max.	Min.	Avg.	Max.	Avg.	Max.	Min.	Avg.	Max.
150	2	5	5	3	0	141	225	286	141	652	1467	.06	.15	0	20	54
	4	5	4	4	0	224	234	246	659	872	1341	.06	.13	17	29	49
	6	5	5	5	0	217	232	252	574	928	1777	.05	.13	15	31	65
	8	5	5	5	0	243	265	283	459	1174	1781	.09	.12	10	41	65
	10	5	5	4	0	196	263	314	196	1048	2671	.04	.10	0	35	105
200	2	5	5	4	0	198	273	319	198	835	1719	.03	.13	0	25	61
	4	5	5	4	0	224	274	302	224	790	1443	.00	.00	0	23	54
	6	5	5	5	1	222	250	302	865	2546	8446	.09	.31	26	100	352
	8	5	5	4	1	206	245	275	206	1426	3563	.08	.26	0	53	146
	10	5	5	5	1	232	278	303	1179	2488	4665	.09	.12	37	98	196
250	2	4	4	4	0	196	212	228	334	796	1639	.00	.00	5	24	61
	4	4	4	4	0	228	236	243	416	955	1672	.00	.00	6	30	62
	6	4	4	3	0	190	224	251	190	576	1130	.00	.00	0	16	42
	8	4	4	3	0	238	256	286	228	1706	3667	.06	.13	0	64	148
	10	4	4	4	0	206	234	281	462	978	1834	.06	.14	11	35	69
400	2	1	1	1	0		284			840		.00			20	
	4	1	1	1	0		225			777		.00			20	
	6	1	1	1	0		304			1283		.00			36	
	8	1	1	1	0		289			1117		.00			33	
	10	1	1	1	0		338			780		.00			16	
500	10	1	1	1	0		243			869		.00			25	

[a] Parameters for computing artificial upper bound (AUB): $h = 0.25$, $g = 0.0025$ for $150 \leq n \leq 250$, 0.0015 for $n \geq 400$.

[b] Key: a, total number of problems attempted; b, number solved to optimality; c, number of problems solved to optimality without use of branch and bound; and d, number of problems that required repeat of branch and bound (i.e., artificial upper bound

Table VI
Computational Results for Large Symmetric Non-Euclidean Multiple Traveling Salesman Problems
(CPU Times on an IBM 3081D)

n	M	No. of Problems[a]				No. of Lagrangean Iterations						Percent Integer Gap[b]		No. of Nodes in Branch-and-Bound (B&B)		
						Before B&B			Total							
		c	d	e	f	Min.	Avg.	Max.	Min.	Avg.	Max.	Avg.	Max.	Min.	Avg.	Max.
400	2	4	4	3	1	293	308	340	304	1206	3028	0.00	0.00	0	40	114
400	4	4	4	4	2	296	328	384	1016	1779	2877	0.06	0.16	32	66	120
400	6	4	4	4	0	196	385	543	743	982	1290	0.02	0.10	20	30	45
400	8	4	4	4	0	318	373	445	480	864	1435	0.00	0.00	13	25	47
400	10	4	4	4	0	410	432	458	1002	1704	2713	0.00	0.00	28	55	98
500	2	4	1[c]	1	0	363	363	363	1031	1031	1031	0.00	0.00	42	42	42
500	4	4	2	2	0	327	330	332	834	1155	1476	0.00	0.00	29	41	53
500	6	4	2	2	0	451	547	643	1574	1891	2208	0.00	0.00	55	67	79
500	8	4	3	3	0	389	458	557	1011	1762	2377	0.00	0.00	34	61	86
500	10	4	1	1	0	514	514	514	1229	1229	1229	0.00	0.00	34	34	34

[a] Key: c, number of problems attempted; d, number of problems solved to optimality; e, number of problems that required usage; and f, number of problems that required repeat of branch-and-bound because initial artificial upper bound was too low.

Table V
(Continued)

CPU (IBM 3032) Time (sec)					
Average			Total		
Setup	Heuristic	Remainder	Min.	Avg.	Max.
3.6	4.1	48	18	56	108
3.8	7.8	63	59	74	98
3.6	8.6	61	54	73	109
4.3	15.4	80	67	100	130
4.1	20.4	76	45	101	224
7.0	6.4	105	49	119	243
7.4	12.3	95	54	115	204
6.9	16.8	297	136	320	1008
6.2	17.0	167	54	190	468
6.7	26.3	298	155	331	647
9.2	6.4	166	99	182	342
9.8	10.2	201	130	221	360
10.1	8.6	124	78	143	246
11.9	15.5	338	86	366	800
9.9	9.4	213	120	233	423
17.1	58.0	511		586	
17.0	55.5	486		558	
20.3	64.3	813		887	
19.0	74.1	486		589	
17.1	80.6	445		543	
26.4	80.2	689		796	

chosen at start of branch and bound was too low).

[c] Percent integer gap = 100 $(Z^* - W^*)/Z^*$. Minimum was 0.0 for all problem sizes.

Table VI
(Continued)

CPU (IBM 3081D) Time (sec)					
Average			Total		
Setup	Heuristic	Remainder	Min.	Avg.	Max.
14.5	24.3	329	152	368	839
14.4	111.6	522	324	573	899
11.1	32.6	283	251	326	406
14.1	31.9	252	188	298	472
14.2	46.8	443	344	504	726
19.7	50.5	526	597	597	597
27.0	43.6	523	492	594	695
27.9	56.8	844	849	929	1008
22.7	63.8	748	505	834	1092
28.5	69.6	521	619	619	619

[b] Minimum integer gap was 0.0% for all problems.

[c] Unresolved problems exceeded CPU limit of 1300 seconds.

5. Computational Results

In this section, we present computational results obtained in a series of tests that used an implementation of the algorithm. The program was coded in FORTRAN and run on an IBM 3032 and an IBM 3081D computer. Computations were performed on both euclidean and non-euclidean distance matrices.

5.1. Non-Euclidean Distance Matrices

The algorithm was first tested on 276 non-euclidean problems with up to 500 cities and 10 salesmen. We constructed ten different distance matrices for problem sizes in the range $40 \leq n \leq 100$; budget constraints precluded the use of the same sample size for larger problem sizes. The underlying graph was complete with integer edge lengths c_{ij} that were generated randomly from a uniform distribution in the range 0–400.

Tables IV and V summarize the computational results. The average number of subgradient iterations required is relatively modest; the number of nodes examined in the branching tree never exceeds 400 for any problem size. Of particular interest is the behavior of the integer gap $(Z^* - \tilde{L}(\pi^*))/Z^*$. The average integer gap decreases as the problem size increases and is zero for *all* problems with $n \geq 400$. This result might imply that as problem sizes increase even further, a majority of problems could have an integer gap equal to zero. Most of the past experience with MTSP algorithms had shown that for a given n, the problems were easier to solve as M increased; however, no such pattern was exhibited in the computational experiments. Since all 400 and 500 city problems had a zero integer gap, a second set of 40 problems was run, using a larger sample size. These runs were done on an IBM 3081D, which is approximately 2.5 times faster than the IBM 3032. Table VI summarizes the results obtained in this experiment. All 400 city and nine of the 500 city problems were solved within the preset time limit of 1300 seconds, and 3 of the 400 city problems had a positive integer gap, while all 500 city problems solved had a zero gap.

In the previous set of experiments, many of the large-scale problems exhibited a zero integer gap. In order to generate problems with a higher level of difficulty, we modified the data generation method to represent cases with significant integer gaps. The cost matrix entries were generated randomly, using the formula $c_{ij} = K + U(0; 400)$. The shortest path between each pair of cities was computed and used as cost entries to the algorithm. By modifying the value of K, we generated different cost matrices. This data

Table VII
Computational Results for Large Symmetric Multiple Traveling Salesman Problems that Satisfy the Triangle Inequality

K	n	M	No. of Problems[a]				No. of Lagrangean Iterations						Percent Integer Gap[b]		No. of Nodes in Branch-and-Bound (B&B)		
							Before B&B			Total							
			c	d	e	f	Min.	Avg.	Max.	Min.	Avg.	Max.	Avg.	Max.	Min.	Avg.	Max.
25	250	2	4	4	4	0	273	302	326	552	1631	2461	0.00	0.00	15	61	95
	250	4	4	4	4	0	270	292	305	778	1060	1451	0.00	0.00	24	38	54
	250	6	4	4	4	1	260	323	351	400	749	1076	0.01	0.04	2	18	31
	250	8	4	4	3	0	285	309	326	285	1011	1840	0.00	0.00	0	28	45
	250	10	4	4	4	1	307	336	380	1102	3135	6238	0.03	0.11	29	87	143
15	250	2	4	4	4	0	289	299	313	374	860	1557	0.00	0.00	6	30	66
	250	4	4	4	4	0	291	300	316	635	1044	1522	0.00	0.00	22	35	53
	250	6	4	4	3	1	291	316	333	291	521	845	0.02	0.06	0	11	27
	250	8	4	4	4	1	317	345	387	517	724	929	0.07	0.28	13	20	28
	250	10	4	4	4	1	344	368	387	488	856	1330	0.06	0.22	10	23	37
10	250	2	4	4	4	0	271	288	297	370	594	794	0.00	0.00	8	17	27
	250	4	4	4	4	1	286	308	323	572	983	1271	0.06	0.25	14	31	42
	250	6	4	4	4	1	283	338	361	417	659	893	0.02	0.09	2	18	38
	250	8	4	4	3	1	295	357	403	295	876	1394	0.10	0.38	0	20	39
	250	10	4	4	4	1	307	359	380	717	1633	2564	0.14	0.56	22	51	78
5	250	2	1	0[c]	0	0		235			16150		0.98			1093	
	250	4	1	0	0	0		222			16450		1.38			879	
	250	6	1	0	0	0		376			16396		0.44			776	
	250	8	1	1	1	0		418			1876		0.00			62	
	250	10	1	0	0	0		341			21572		1.18			1394	

[a] Key: c, number of problems attempted; d, number of problems solved to optimality; e, number of problems that required branch-and-bound usage; and f, number of problems that required repeat of branch-and-bound because initial artificial upper bound was too low.

Table VIII
Computational Results for Symmetric Euclidean Multiple Traveling Salesman Problems[a]

n	M	No. of Problems				No. of Lagrangean Iterations						Percent Integer Gap			No. of Nodes in Branch-and-Bound (B&B)		
						Up to Start of B&B			Total								
		Total	Solved	Required B&B	Required B&B Repeat	Min.	Avg.	Max.	Min.	Avg.	Max.	Min.	Avg.	Max.	Min.	Avg.	Max.
30	2	5	5	1	1	25	80	195	25	326	1425	.00	.22	1.12	0	12	58
30	4	5	5	1	0	20	78	201	20	265	1136	.00	.07	.33	0	11	54
30	6	5	5	2	0	25	187	411	25	324	863	.00	.10	.40	0	7	34
30	8	5	5	1	1	24	102	181	24	325	1298	.00	.14	.68	0	12	60
30	10	5	5	1	0	2	70	191	2	205	863	.00	.06	.32	0	7	37
60	2	5	5	5	0	168	226	347	516	2721	8366	.28	.82	1.34	13	90	265
60	4	5	5	5	0	165	210	309	505	5054	21139	.27	.82	1.45	20	172	720
60	6	5	5	5	0	167	206	293	313	18472	85167	.26	.61	1.19	8	634	2925
60	8	5	5	5	0	189	243	318	1236	15759	71372	.17	.54	1.25	45	558	2531
60	10	5	5	4	0	110	213	275	110	13404	46919	.00	.47	1.14	0	476	1671
80	2	5	5	5	0	204	251	297	455	24332	88073	.21	.65	1.39	11	843	3064
80	4	5	5	5	0	219	252	291	444	18552	76175	.13	.60	1.34	11	677	2806
80	6	5	5	5	0	204	235	267	490	15981	68619	.12	.47	.95	10	616	2685
80	8	5	5	5	0	226	250	280	1314	24029	60599	.20	.41	.75	43	866	2142
80	10	5	4[b]	5	0	197	235	306	989	15565	33982	.16	.41	.69	29	598	1296
100	2	2	2	2	0	245	291	336	23851	45208	66564	.70	.83	.95	835	1518	2201
100	4	2	2	1	0	99	161	223	99	458	817	.00	.06	.12	0	11	21
100	6	2	2	1	0	124	190	256	124	3679	7234	.00	.09	.18	0	136	272
100	8	2	2	2	0	264	273	282	444	898	1352	.00	.05	.11	5	25	45
100	10	2	2	2	0	224	242	260	479	1635	2791	.00	.11	.21	7	46	84

[a] Parameters: $h = 1$, $g = \begin{cases} 0.005 & \text{when } n = 30, \\ 0.015 & \text{when } 60 \leq n \leq 100. \end{cases}$

[b] One problem was unsolved due to excessive computing time (>2040 CPU seconds).

Table VII
(Continued)

CPU (IBM 3081D) Time (sec)					
Average			Total		
Setup	Heuristic	Remainder	Min.	Avg.	Max.
15.7	11.4	115	89	143	199
16.8	12.2	99	112	128	144
16.8	12.7	83	94	113	142
16.6	15.8	92	68	124	166
17.0	18.9	256	149	292	449
17.1	11.3	72	74	100	142
16.7	13.9	101	95	132	183
16.5	14.9	62	69	93	108
16.9	16.2	91	107	124	161
17.7	18.1	103	97	139	225
5.0	10.3	59	56	74	84
5.6	12.8	99	106	117	133
5.7	13.3	79	83	98	131
5.8	15.4	122	55	144	252
5.8	18.4	134	108	158	198
15.3	7.1	578		>601	
15.7	6.5	580		>602	
15.8	11.2	573		>600	
15.5	12.5	166		194	
16.1	7.5	578		>602	

[b] Minimum integer gap was 0.0% for all problems with $K = 25$, 15 or 10. For unsolved problems, integer gap is computed using best-known upper value instead of Z^*.
[c] Unsolved problems exceeded CPU limit of 600 seconds.

Table VIII
(Continued)

CPU (IBM 3032) Time (sec)					
Average			Total		
Setup	Heuristic	Remainder	Min.	Avg.	Max.
0.1	0.1	1.6	0.4	1.9	7.1
0.2	0.2	1.4	0.5	1.8	6.3
0.2	0.6	2.2	0.5	3.0	6.1
0.2	1.0	1.7	0.6	2.9	7.7
0.2	1.0	1.0	0.5	2.2	6.2
0.4	0.5	39	10	40	119
0.4	1.0	81	10	83	342
0.4	1.4	305	7	307	1406
0.5	1.9	267	21	270	1214
0.5	2.7	225	4	228	796
0.7	0.5	550	14	551	1937
0.8	0.8	476	14	478	1993
0.8	1.4	431	18	433	1902
0.7	1.7	613	36	616	1554
0.7	2.4	408	33	411	957
1.0	0.6	1507	778	1509	2240
0.9	0.4	15	7	17	27
1.1	0.7	139	8	141	274
0.9	1.2	32	24	34	44
1.0	3.1	51	26	55	85

generation method satisfies the triangle inequality, but does not necessarily satisfy the euclidean norm. n was kept constant at 250 while M and K were modified. Sixty-five problems in total were generated. Table VII summarizes the results obtained by those experiments. (The runs were done on an IBM 3081D, which is approximately 2.5 times faster than the IBM 3032.) As K decreases, the problems increase in difficulty, leading, for $K = 5$, to problems with very significant integer gaps.

5.2. 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.

Eighty-four problems involving up to 100 cities and 10 salesmen were attempted with euclidean distance matrices. The performance of the algorithm was 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 the number in solutions to non-euclidean problems of equal size. Table VIII presents the computational results.

5.3. Comparison with Existing Algorithms

Table IX compares the performance of the algorithm to the only other algorithms that were developed for solving MTSPs, those of Laporte and Nobert, and Ali and Kennington. Laporte and Nobert solved problems with a fixed number of salesmen M, as well as problems with M variable; the algorithm developed in this paper can be easily modified to solve problems for which M is variable (see Section 6).

The Laporte-Nobert algorithm was implemented on a CYBER 173 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

Table IX
Comparison of Computational Results for Symmetric Multiple Traveling Salesman Problems

No. of Cities (n)	Algorithm[a]	No. of Salesmen (M)	Problem Type[b]	No. of Problems		Average No. of Iterations	Average No. of Nodes in Branch and Bound	Average Total CPU Time (sec)	Computer[c] Used
				Attempted	Solved				
30	GS	2–10	EU	25	25	289	10	2.4	IBM 3032
30	LN	Variable	EU	5	5		NA	7.2	CYBER 173
30	AK	1–6	EU	12	12	NA	8944	109.4	CDC 6600
60	GS	2–10	EU	25	25	11082	386	185.6	IBM 3032
60	LN	Variable	EU	5	5		NA	66.0	CYBER 173
60	AK	1–7	EU	2	2	NA	16055	904.7	CDC 6600
80	GS	2–10	EU	25	24[d]	19692	720	497.8[e]	IBM 3032
80	LN	Variable	EU	5	2[f]		NA	283.0[e]	CYBER 173
100	GS	2–10	EU	10	10	10376	347	351.2	IBM 3032
60	GS	2–10	NU	50	50	620	19	14.2	IBM 3032
60	LN	Variable	NU	5	5		NA	45.2	CYBER 173
80	GS	2–10	NU	50	50	803	28	27.0	IBM 3032
80	LN	Variable	NU	5	5		NA	137.0	CYBER 173
100	GS	2–10	NU	50	50	632	19	33.0	IBM 3032
100	LN	Variable	NU	5	3[d]		NA	208.3[e]	CYBER 173
100	AK	1–9	NU	50	15[d,f]	NA	8	80.5[e]	CDC 6600
250	GS	2–10	NU	20	20	1002	34	229.0	IBM 3032
400	GS	2–10	NU	5	5	959	25	594.7	IBM 3032

[a] GS, Gavish-Srikanth algorithm; LN, Laporte-Nobert algorithm; and AK, Ali-Kennington algorithm.
[b] NU, Non-euclidean cost matrix; EU, euclidean cost matrix.
[c] The IBM 3032 is estimated to be 4–6 times slower than either the CYBER 173 or the CDC 6600.
[d] Excessive computing time (CPU > 2040 seconds).
[e] Does not include times for problems that were terminated before optimality was proved.
[f] Some problems were not solved due to insufficient memory.

Laporte-Nobert algorithm solved 30% of all attempted problems. The algorithm presented in this paper did not suffer from this problem; it uses a depth-first branch-and-bound procedure, 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 the Ali-Kennington algorithm. The non-euclidean problem sizes which the algorithm can handle are, by a factor of 5, larger than the largest problems solved by either of the other algorithms.

As mentioned earlier, an n-city, M-salesman problem can be transformed to an equivalent $(n + M - 1)$ city, single TSP. We tested the effect of performing this transformation on non-euclidean problems with 40 cities and up to 10 salesmen. (Budget considerations precluded performance of such tests on larger problem sizes and/or euclidean data). The TSP algorithm used to solve the resulting TSP (see Gavish and Srikanth 1983) was the best one known for symmetric, non-euclidean problems. It fully exploits graph sparsity and is by at least a factor of two faster than the MTSP code for single salesman problems. Table X summarizes the computing results obtained for the transformed problems; Table XI shows the effect of repeating the runs using the known optimal solution values as the initial upper bound (to avoid the biasing effects of different initial feasible solution values). For easy comparison, these tables also show the performance of our MTSP algorithm on the same problems. Even allowing for the slightly larger problem sizes that result from the transformation, it is clear that for larger values of $M(M \geq 4)$, the transformed problems are much harder to solve than the original multiple-salesman problems

5.4. The Effect of Scaling Data

In order to test the effect on solution times of the number of significant digits in problem data, we divided the c_{ij} values by a scaling constant and

Table X
Effect of Transforming Symmetric Non-Euclidean Multiple Traveling Salesman Problems to $(n + M - 1)$ City Traveling Salesman Problems

Problem Size		Algorithm[a]	No. of Problems Solved to Optimality[b]	No. Lagrangean Iterations			CPU (IBM 3032) Time (sec)		
n	M			Min.	Avg.	Max.	Min.	Avg.	Max.
40	1	GS-T	10	57	303	984	1.2	3.5	9.0
40	2	GS	10	22	191	496	1.8	4.0	6.9
		GS-T	10	42	202	680	1.0	2.4	5.9
40	4	GS	10	30	96	289	2.4	3.6	5.9
		GS-T	10	215	282	516	2.6	3.2	4.7
40	6	GS	10	31	233	665	2.9	5.4	9.0
		GS-T	9	356	>870	>2400	4.0	>8.4	>18.0
40	8	GS	10	31	302	939	3.5	6.7	13.0
		GS-T	4	673	>1476	>2000	6.0	>15.6	>19.0
40	10	GS	10	28	115	390	3.5	5.5	8.6
		GS-T	4	691	>1267	>1700	8.0	>14.1	>21.9

[a] GS, Gavish-Srikanth algorithm for the MTSP, described in this paper; GS-T, Gavish-Srikanth (1983) algorithm for the TSP.
[b] Some problems (out of ten) remained unsolved due to excessive computing time (CPU > 19–22 seconds).

then rounded off to the nearest integer. Experiments were conducted on non-euclidean data. Four integer datasets were created with c_{ij} values in the range 0 to 10000; from each of these datasets, modified distance matrices C' were created by choosing $c'_{ij} = \lfloor (c_{ij}/b) + 0.5 \rfloor$ and values of the scaling constant b equal to 2.5, 10, 25, 100 and 1000. Budget constraints precluded running problems with all possible combinations of n and M values on these modified datasets; therefore, the set of 250 city problems was rerun with $M = 6$.

The results of those experiments are summarized in Table XII. For problems with nonzero integer gaps, the relative integer gap decreases as b decreases. Computing times generally increase as the value of the scaling constant b decreases; however, most of this increase is due to the fact that the parameters h and g used for computing the artificial upper bound value Z_f were unchanged over all runs. Problems with small b values could have been solved much more quickly by using lower values for g.

Since problems with scaled data generally seem to require less computing effort, it might be possible to use scaling as a heuristic for problems with a large

Table XI
Computational Results for Symmetric Non-Euclidean Multiple Traveling Salesman Problems Transformed to $(n + M - 1)$ City Traveling Salesman Problems with Optimal Solution Value as Initial Upper Bound

Problem Size		Algorithm[a]	No. of Problems Solved to Optimality[b]	No. of Lagrangean Iterations			CPU (IBM 3032) Time (sec)		
n	M			Min.	Avg.	Max.	Min.	Avg.	Max.
40	1	GS-T	10	16	132	484	0.4	1.4	4.0
40	2	GS	10	10	149	461	0.7	2.6	5.0
		GS-T	10	10	99	295	0.4	1.2	2.6
40	4	GS	10	17	86	295	1.1	2.3	4.7
		GS-T	10	39	97	211	0.7	1.4	2.5
40	6	GS	10	16	178	445	1.4	3.6	6.5
		GS-T	7	121	>954	>2600	1.8	>8.4	>18.2
40	8	GS	10	13	205	573	1.4	4.5	7.7
		GS-T	1	212	>1311	>1800	3.5	>17.0	>19.0
40	10	GS	10	16	62	299	1.8	3.1	6
		GS-T	2	341	>1037	>1400	5.6	>16.6	>19.3

[a] GS, Gavish-Srikanth algorithm for the MTSP, described in this paper; GS-T, Gavish-Srikanth (1983) algorithm for the TSP.
[b] Some problems (out of ten) stayed unsolved due to execssive computing time (CPU > 19–20 seconds).

Table XII
Computational Results for Symmetric, Non-Euclidean Multiple Traveling Salesman Problems with Scaled Data

n	M	No. of Problems: Total	Solved	Required B&B	$C(i,j)$ Range	No. of Lagrangean Iterations: Before B&B Min.	Before B&B Avg.	Before B&B Max.	Total Min.	Total Avg.	Total Max.	Percent Integer Gap: Min.	Avg.	Max.	No. of Nodes in Branch-and-Bound (B&B): Min.	Avg.	Max.
250	6	4	4	0	0–10	1	2	3	1	2	3	.00	.00	.00	0	0	0
250	6	4	4	4	0–100	234	253	275	676	1134	2038	.00	.00	.00	16	32	70
250	6	4	4	3	0–400	190	224	251	190	576	1130	.00	.00	.00	0	16	42
250	6	4	4	4	0–1000	258	289	326	541	2587	5021	.00	.01	.05	7	103	228
250	6	4	4	4	0–4000	284	308	334	405	3524	7408	.00	.01	.05	4	150	338
250	6	3	3	2	0–10000	301	342	392	301	5341	7984	.00	.01	.02	0	223	341

Table XIII
Computational Results for Symmetric Non-Euclidean Multiple Traveling Salesman Problems with Optimal Solution Value as Initial Upper Bound

n	M	No. of Problems: Total	Solved	B&B	Required Repetition	No. of Lagrangean Iterations: Before B&B Min.	Before B&B Avg.	Before B&B Max.	Total Min.	Total Avg.	Total Max.	Percent Integer Gap: Min.	Avg.	Max.	No. of Nodes in Branch-and-Bound (B&B): Min.	Avg.	Max.
250	2	4	4	0	0	73	79	89	73	79	89	.00	.00	.00	0	0	0
250	4	4	4	0	0	61	80	105	61	80	105	.00	.00	.00	0	0	0
250	8	4	4	0	0	40	67	96	40	67	96	.00	.00	.00	0	0	0
250	8	4	4	1	0	51	143	222	51	296	835	.00	.03	.12	0	7	27
250	10	4	4	0	0	66	127	177	66	127	177	.00	.00	.00	0	0	0
400	2	1	1	0	0	119	119	119	119	119	119	.00	.00	.00	0	0	0
400	4	1	1	0	0	106	106	106	106	106	106	.00	.00	.00	0	0	0
400	6	1	1	0	0	143	143	143	143	143	143	.00	.00	.00	0	0	0
400	8	1	1	0	0	112	112	112	112	112	112	.00	.00	.00	0	0	0
400	10	1	1	1	0	225	225	225	451	451	451	.15	.15	.15	11	11	11
500	10	1	1	0	0	108	108	108	108	108	108	.00	.00	.00	0	0	0

Table XIV
Computational Results for Symmetric Euclidean Multiple Traveling Salesman Problems with Optimal Solution Value as Initial Upper Bound

n	M	No. of Problems: Total	Solved	B&B	Required Repetition	No. of Lagrangean Iterations: Before B&B Min.	Before B&B Avg.	Before B&B Max.	Total Min.	Total Avg.	Total Max.	Percent Integer Gap: Min.	Avg.	Max.	No. of Nodes in Branch-and-Bound (B&B): Min.	Avg.	Max.
60	2	5	5	5	0	165	202	288	264	2849	12016	.28	.82	1.34	10	128	556
60	4	5	5	5	0	162	213	337	309	1505	5018	.27	.80	1.45	10	71	234
60	6	5	5	5	0	162	208	305	285	9937	48139	.26	.58	1.05	5	463	2261
60	8	5	5	5	0	190	226	313	440	11945	56701	.17	.51	1.12	13	569	2726
60	10	5	5	4	0	46	197	320	46	6043	27698	.00	.46	1.05	0	278	1303
80	2	5	5	5	0	203	244	323	301	11788	48251	.21	.66	1.42	5	515	2104
80	4	4	4	4	0	213	232	262	561	1968	3110	.16	.43	.85	19	88	138
80	6	5	5	4	0	198	252	316	206	2333	5882	.12	.51	.85	0	94	226
80	8	5	5	5	0	215	237	283	1305	5894	19583	.20	.43	.75	54	303	989
80	10	4	4	4	0	188	238	283	1035	1754	2122	.16	.39	.68	35	80	104

Table XII
(Continued)

CPU (IBM 3032) Time (sec)					
Average			Total		
Setup	Heuristic	Remainder	Min.	Avg.	Max.
4.9	8.6	0.9	10.1	14.5	19.7
6.3	121.8	243.5	298.2	371.5	511.1
6.7	87.1	123.8	166.0	222.6	320.5
7.7	142.0	491.4	334.0	641.1	1045.8
7.1	140.3	660.2	294.7	810.2	1523.3
7.0	151.9	1083.1	241.2	1242.0	1932.4

Table XIII
(Continued)

CPU (IBM 3032) Time (sec)					
Average			Total		
Setup	Heuristic	Remainder	Min.	Avg.	Max.
6.5	.0	22.0	25.8	28.5	31.4
6.9	.0	25.9	24.8	32.8	41.9
7.7	.0	20.2	21.4	27.9	37.5
7.5	.0	59.4	21.0	66.9	143.1
6.9	.0	40.1	26.5	46.9	62.7
16.3	.0	114.4	130.7	130.7	130.7
16.5	.0	86.2	102.6	102.6	102.6
20.1	.0	88.8	118.9	118.9	118.9
18.5	.0	73.6	92.1	92.1	92.1
16.6	.0	184.2	210.8	210.8	210.8
26.0	.0	122.6	148.6	148.6	148.6

Table XIV
(Continued)

CPU (IBM 3032) Time (sec)					
Average			Total		
Setup	Heuristic	Remainder	Min.	Avg.	Max.
.5	.0	42.3	4.5	42.8	179.3
.5	.0	24.3	5.3	24.9	84.1
.5	.0	167.1	5.2	167.6	811.3
.5	.0	211.3	8.2	211.8	1010.6
.5	.0	109.9	1.8	110.4	504.7
.8	.0	279.5	9.2	280.2	1134.2
.8	.0	46.6	13.3	47.4	69.6
.8	.0	70.8	7.6	71.6	210.5
.8	.0	139.6	28.4	140.4	467.2
.8	.0	47.1	28.4	47.9	88.0

range of c_{ij} values. From the matrix C, a scaled-down distance matrix C' can be defined, by choosing $c'_{ij} = \lfloor (c_{ij}/b) + 0.5 \rfloor$ and b as a value significantly greater than 1. The problem is solved to optimality using the matrix C', all feasible tours generated in the course of finding the optimal solution are evaluated using the original c_{ij} data, and the best solution value found is used as an initial upper bound for the original problem.

Gavish and Srikanth (1980) tested this procedure for each of the four datasets with c_{ij} values in the range 0–10000, with the scaling constant b set to 10. In at least one case, considerable savings in computing time have resulted. The limited results obtained so far indicate that this is an area that might be worth further investigation.

5.5. Computational Results with Known Optimal Solution Value

The performance of any algorithm using branch-and-bound techniques is highly dependent on the quality of the best-known feasible solution at the beginning of the branch-and-bound process. It is possible that an otherwise excellent algorithm might perform poorly simply because the heuristic methods used to generate initial feasible solutions were inferior.

To eliminate the biasing effects of (possibly) poor initial feasible solution values, the runs on the more difficult problems were repeated, using the known optimal solution value as the initial upper bound. Crowder and Padberg, for example, have used *known* optimal solutions to the TSP in order to fathom nodes in their solution algorithm. Essentially, these runs ensure that no better solution exists, i.e., they verify the optimality of the optimal solution. Tables XIII and XIV show the results. It must be emphasized that the computing times shown are meant for use as benchmarks only; they represent the performance of the algorithm under ideal conditions, not under "real" conditions for which the value of the optimal solution is *unknown* at the start of the solution procedure.

6. Possible Extensions of the Algorithm

This section presents possible extensions of the problem; these extensions can be handled by minor modifications of the algorithm.

6.1. Allowing for a Variable Number of Salesman M

Consider the MTSP stated in Section 1. If the problem is transformed to one for which M is also a variable,

with an upper bound m, it can be stated as:

Problem P1

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

$$Z^* = \min_{M,X} \left\{ \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} c_{ij}x_{ij} + \sum_{i \in S} c_{i1}x_{i1} \right\} \tag{9}$$

subject to constraints (2)–(8), where

$$1 \leqslant M \leqslant m, \quad \text{and} \tag{10}$$

$$M \text{ is an integer.} \tag{11}$$

Consider the effect of adding $m - 1$ new cities to the problem, and defining their cost values as

$$c_{ij} = \begin{cases} \infty & \text{if } i \in S, j > n, \quad \text{or } i > n, j \in S, \\ 0 & \text{otherwise.} \end{cases}$$

As Srikanth (1983) proved, solving the new $n + m - 1$ city, m salesman problem as an MTSP is equivalent to solving the desired n-city variable-M problem. Preliminary computational experience indicates that for $m = 10$, computing time increases by 50–80% for variable-M problems (compared to fixed-M problems of the same size).

6.2. Bounds on the Number of Immediate Subtours

Let U be a desired upper bound on the number of immediate subtours in the final solution. The sole effect of this constraint is on the selection of return arcs in the solution procedure for the Lagrangean subproblem. 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.

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

7. Conclusion

The algorithm demonstrates good computational performance, especially in the case of non-euclidean data. The fact that the mean 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 resources were consumed in computing the constrained minimal spanning trees. The algorithm currently in use is a modification of the Glover-Klingman algorithm (1975), which requires at least $O(n^2)$ operations. By performing extensive sensitivity analysis on all the arcs and exploiting the resulting graph sparsity by using an algorithm developed by Yao (1975), we can compute the spanning trees in $O(|A| \log \log n)$ operations. Preliminary computational experience indicates that for $|A| \cong O(n)$, the procedure using Yao's algorithm leads to significantly improved computing times, without affecting the number of nodes or iterations involved in the branch-and-bound procedure. An adaptation of those methods to the symmetric single traveling salesman problem in Gavish and Srikanth (1983) led to an algorithm that is comparable to the best-known algorithms for euclidean data and is faster by one to two orders of magnitude than other algorithms for non-euclidean data.

Acknowledgment

Professor Gavish's research was partially supported by National Science Foundation grant ECS-8020644. The authors gratefully acknowledge IBM's support in providing computing resources for the extensive computational tests.

References

ALI, A. I., AND J. L. KENNINGTON. 1980. The M-Traveling Salesman Problem: A Duality Based Branch and Bound Algorithm. Technical Report OR 80018 (August), Southern Methodist University, Dallas.

ANGEL, R. D., W. L. CAUDLE, R. NOONAN AND A. WHINSTON. 1972. Computer Assisted School Bus Scheduling. *Mgmt. Sci.* **18,** 279–288.

BALAS, E., AND N. CHRISTOFIDES. 1981. A Restricted Lagrangean Approch to the Traveling Salesman Problem. *Math. Program.* **21,** 19–46.

BAZARAA, M. S., AND A. N. ELSHAFEI. 1977. On the Use of Fictitious Bound in Tree Search Algorithms. *Mgmt. Sci.* **23,** 904–908.

BELLMORE, M., AND S. HONG. 1974. Transformation of Multi-Salesman Problem to the Standard Traveling Salesman Problem. *J. Assoc. Comput. Mach.* **21,** 500–504.

CARPANETO, G., AND P. TOTH. 1980. Some New Branching and Bounding Criteria for the Asymmetric Traveling Salesman Problem. *Mgmt. Sci.* **26,** 736–743.

CHRISTOFIDES, N. 1979. The Traveling Salesman Problem. In *Combinatorial Optimization*, N. Christofides (ed.). John Wiley & Sons, New York.

CHRISTOFIDES, N., AND S. EILON. 1969. An Algorithm for the Vehicle Dispatching Problem. *Opns. Res. Quart.* **20,** 309–318.

CROWDER, H., AND M. W. PADBERG. 1980. Solving Large-Scale Symmetric Traveling Salesman Problems to Optimality. *Mgmt. Sci.* **26,** 495–509.

CROWDER, H., E. L. JOHNSON AND M. PADBERG. 1983. Solving Large-Scale Zero-One Linear Programming Problems. *Opns. Res.* **31,** 803–834.

EVERETT, H. 1963. Generalized Lagrange Multiplier Method for Solving Problems of Optimum Allocation of Resources. *Opns. Res.* **11,** 399–417.

GABOW, H. N. 1978. A Good Algorithm for Smallest Spanning Trees with a Degree Constraint. *Networks* **8,** 201–208.

GAVISH, B. 1976. A Note on the Formulation of the M-Salesman Traveling Salesman Problem. *Mgmt. Sci.* **22,** 704–705.

GAVISH, B. 1982. Topological Design of Centralized Computer Networks—Formulations and Algorithms. *Networks* **12,** 355–377.

GAVISH, B. 1983. Formulations and Algorithms for the Capacitated Minimal Directed Tree Problem. *J. Assoc. Comput. Mach.* **30,** 118–132.

GAVISH, B., AND S. GRAVES. 1978. The Travelling Salesman Problem and Related Problems. Working Paper, Graduate School of Management, University of Rochester (May).

GAVISH, B., AND H. PIRKUL. 1985a. Efficient Algorithms for Solving Multiconstraint Zero-One Knapsack Problems to Optimality. *Math Program.* **31,** 78–105.

GAVISH, B., AND H. PIRKUL. 1985b. Zero-One Integer Programs with Few Constraints—Efficient Branch and Bound Procedures. *Eur. J. Opns. Res.* **22,** 35–43.

GAVISH, B., AND K. N. SRIKANTH. 1979. Mathematical Formulations for the Dial-a-Ride Problem. Working Paper, Graduate School of Management, University of Rochester (March).

GAVISH, B., AND K. N. SRIKANTH. 1980a. An Optimal Solution Method for Multiple Traveling Salesman Problems. Working Paper, Graduate School of Management, University of Rochester.

GAVISH, B., AND K. N. SRIKANTH. 1980b. $O(N^2)$ Algorithms for Sensitivity Analysis of Minimal Spanning Trees and Related Subgraphs. Working Paper, Graduate School of Management, University of Rochester.

GAVISH, B., AND K. N. SRIKANTH. 1983. Algorithms for Solving Large-Scale Symmetric Traveling Salesman Problems to Optimality. Working Paper Series No. QM8329, Graduate School of Management, University of Rochester.

GLOVER, F., AND D. KLINGMAN. 1975. Finding Minimum Spanning Trees with Fixed Number of Links at a Node. In *Combinatorial Programming: Methods and Applications*, pp. 191–201, B. Roy (ed.). D. Reidel Publishing, Dordrecht-Holland.

HELBIG-HANSEN, K. H., AND J. KRARUP. 1974. Improvements of the Held-Karp Algorithm for the Symmetric Traveling Salesman Problem. *Math. Program.* **7,** 87–96.

HELD, M., AND R. M. KARP. 1970. The Traveling Salesman Problem and Minimal Spanning Trees: Part 1. *Opns. Res.* **18,** 1138–1162.

HELD, M., AND R. M. KARP. 1971. The Traveling Salesman Problem and Minimal Spanning Trees: Part 2. *Math. Program.* **1,** 6–25.

HELD, M., P. WOLFE AND H. P. CROWDER. 1974. Validation of Subgradient Optimization. *Math. Program.* **6,** 62–88.

LAPORTE, G., AND Y. NOBERT. 1980. A Cutting Planes Algorithm for the M-Salesman Problem. *Opnl. Res. Quart.* **31,** 1017–1023.

LIN, S., AND B. W. KERNIGHAN. 1973. An Effective Heuristic Algorithm for the Traveling Salesman Problem. *Opns. Res.* **21,** 493–516.

MILIOTIS, P. 1976. Integer Programming Approaches to the Traveling Salesman Problem. *Math. Program.* **10,** 367–378.

MILIOTIS, P. 1978. Using Cutting Planes to Solve the Symmetric Traveling Salesman Problem. *Math. Program.* **15,** 177–178.

MILLER, C. E., A. W. TUCKER AND R. A. ZEMLIN. 1960. Integer Programing Formulation of Traveling Salesman Problems. *J. Assoc. Comput. Mach.* **7,** 326–332.

ORLOFF, D. S. 1974. Routing a Fleet of M Vehicles to/from a Central Facility. *Networks* **4,** 147–162.

RAO, M. R. 1980. A Note on the Multiple Traveling Salesman Problem. *Opns. Res.* **28,** 628–632.

RUSSELL, R. A. 1977. An Effective Heuristic for the M-Tour Traveling Salesman Problem with Some Side Constraints. *Opns. Res.* **25,** 517–524.

SRIKANTH, K. N. 1980. An Improved Algorithm for Computing Degree-Constrained Minimal Spanning Trees. Working Paper, Graduate School of Management, University of Rochester.

SRIKANTH, K. N. 1983. Travelling Salesman Problems and Related Vehicle Routing Problems. Doctoral Dissertation, Graduate School of Management, University of Rochester.

SVETSKA, J. A. 1976. Response to "A Note on the Formulation of the M-Salesman Traveling Salesman Problem. *Mgmt. Sci.* **22,** 706.

SVETSKA, J. A., AND V. E. HUCKFELDT. 1973. Computational Experience with an M-Salesman Traveling Salesman Algorithm. *Mgmt. Sci.* **19,** No. 7 (1973), pp. 790–799.

YAO, A. C. 1975. An $O(E|\log\log|V|)$ Algorithm for Finding Minimum Spanning Tress. *Inform. Proc. Lett.* **4,** 21–23.