Computers Ops Res. Vol. 21, No. 1, pp. 101–111, 1994
Printed in Great Britain.

0305-0548/94 $6.00+0.00

A HEURISTIC FOR THE MULTIPLE TOUR MAXIMUM COLLECTION PROBLEM

STEVEN E. BUTT† and TOM M. CAVALIER‡
Department of Industrial & Management Systems Engineering and Graduate Program in Operations Research, The Pennsylvania State University, 207 Hammond Building, University Park, PA 16802, U.S.A.

(Received September 1991; in revised form December 1992)

Scope and Purpose—There are many practical applications to which the classical Traveling Salesman Problem and the classical Vehicle Routing Problem are not directly applicable. The purpose of this paper is to investigate one of these applications. Specifically, the case when a time constraint is imposed on the completion of any tour permitting only a subset of the total number of nodes in the graph to be serviced. Thus, the solution to this problem requires not only determining a calling order on each tour, but also selecting which subset of nodes in the graph to service.

Abstract—The multiple tour maximum collection problem (MTMCP) consists of determining the m optimal time constrained tours which visit a subset of weighted nodes in a graph such that the total weight collected from the subset of nodes is maximized. In this paper, a heuristic for the MTMCP is developed. The results of computational testing reveal that the heuristic has three very attractive features. First, the heuristic will always produce a feasible solution if one exists. Second, the heuristic produces very good solutions. In most cases where the exact solution is known, it produces the optimal solution. And finally, the heuristic has the ability to handle large problems in a very short amount of computation time.

1. INTRODUCTION

A common assumption of routing and scheduling problems, like the traveling salesman problem (TSP) and the vehicle routing problem (VRP), is that all of the nodes in the graph are to be visited exactly one time. However, in some instances, a problem may be constrained such that visiting every node in the graph is not possible. An example of such a problem is the multiple tour maximum collection problem (MTMCP).

The MTMCP may be stated as follows. Define a complete graph $G=[N, A]$, such that $N=\{0, 1, \ldots, n\}$ denotes the node set and $A=[a_{ij}]$ denotes the arc set. Let $D=[d_{ij}]$ denote the travel time (or distance) matrix, such that d_{ij} represents the time it takes to travel arc a_{ij}. Let c_i denote the reward which is collected upon visiting node i and let b_i denote the time it takes to collect c_i. [Thus, $C=[c_i]$ denotes the array of rewards (weights) and $B=[b_i]$ denotes the array of collection (visit) times.] Let node 0 denote the depot, with $c_0=0$ and $b_0=0$. Finally, a feasible tour through a subset of G must begin and end at the depot, visit each node in the subset exactly one time, and be completed in at most T time units. Given these definitions, the objective of the MTMCP is to find the m distinct tours through G which maximize the total reward collected on all of the tours.

---

†S. E. Butt is a doctoral student in the Department of Industrial and Management Systems Engineering and the Graduate Program in Operations Research, The Pennsylvania State University, University Park, Pa. He received his B.A. in Mathematics and Chemistry from Earlham College, Richmond, Ind., and his M.S. in Industrial Engineering and Operations Research from The Pennsylvania State University, University Park, Pa. His research interests include mathematical programming, network optimization, vehicle routing and scheduling, and location theory. He is a member of ORSA, TIMS and IIE.

‡T. M. Cavalier is an Associate Professor of Industrial Engineering at The Pennsylvania State University, and Chair of the Intercollege Graduate Program in Operations Research. He received a B.S. degree in Electrical Engineering from West Virginia Institute of Technology, an M.A. in Mathematics from West Virginia University, and a Ph.D. in Industrial Engineering and Operations Research from Virginia Polytechnic Institute and State University. He has published in many journals including *Mathematical Programming*, *European Journal of Operational Research*, *Computers & Operations Research*, *Journal of the Operational Research Society*, and *Transportation Science*. His research interests include mathematical programming, location problems, and network optimization. Dr Cavalier is a member of ORSA, SIAM, TIMS, and Tau Beta Pi.

Note that if all of the nodes in the graph can be visited, then the MTMCP could be reduced to a TSP or a VRP. That is, the TSP and VRP can be thought of as special cases of the MTMCP. This is due to the fact that the solution to the MTMCP requires not only determining a calling order on each tour, as in the TSP and VRP, but also selecting which subset of nodes in the graph to visit.

The problem which initiated this study came from the athletic department of Earlham College in Richmond, Ind. As with most colleges, Earlham's football program is faced with the task of recruiting football players each year. One successful method of recruiting used at many small NCAA Division III schools, is to visit high school campuses and meet with the senior members of the football team. In the particular instance that was presented to us, Earlham wished to visit as many high schools as possible within a 100-mile radius of campus. But, they knew from prior experience that visiting all of the schools in this area was not possible. Therefore, they wanted to visit the best subset of schools in this area which would maximize their potential for recruiting future players. To distinguish between the recruiting potential of each high school, a positive weight (reward) was assigned to each school (node) based upon known and projected data. Other assumptions were also present in this problem. First of all, in order to reduce the cost of recruiting visits, no overnight stays were permitted. That is, each day the recruiter had to begin and end on the college campus (depot). Second, the high schools had to be visited during the hours the students were in class, which introduces an upper bound on the number of hours available each day (T) for making visits. And finally, there was a limited number of days (tours) that the recruiter was allowed to make visits. It was from this recruiting problem, that we were able to develop the general statement of the MTMCP.

There are other practical applications of the MTMCP which can be found in the literature, examples include the following. In [1], Golden *et al.* present a very complex inventory/routing problem in which a fleet of trucks is used to deliver fuel to a number of customers. The step in this problem which involves the selection of the customers to be serviced on a given day, can be solved as an MTMCP. The MTMCP can be used to model a variety of scheduling problems, such as the daily operations of a steel rolling mill [2] or the branch messengers at a commercial bank [3].

A very interesting application of the MTMCP is to the optimization problem faced by competitors in the sport of score orienteering. In score orienteering, there is a starting point, an ending point and a number of other locations which have associated scores (rewards). The objective is for the competitors to select a subset of locations to visit, such that their total score is maximized as they travel from the start to the finish in a fixed amount of time. Those competitors which arrive at the ending location after the allotted time are disqualified. Golden *et al.* [4] have shown that this problem, which has been named the 'orienteering problem' (OP), is *NP*-hard. In view of the fact that the OP is the special case of the MTMCP when only one tour is to be taken through the graph, it follows that the MTMCP is also *NP*-hard.

The MTMCP is also closely related to the multiobjective vending problem (MVP) [5]. The only difference between the MVP and the MTMCP is in their objective functions. In the MVP, the objective function involves minimizing the total distance traveled, as well as maximizing the reward collected. Keller and Goodchild [5] list several applications of the MVP. One application involves the scheduling of a mobile vending service which must balance the time lost in traveling from site to site with the magnitude of the potential gains at each one. (Several other applications of the MTMCP can be found in [6] and [7].)

Solution procedures for the MTMCP have only been developed for the special case when there is one tour taken through the graph (i.e. when $m=1$). Among these solution procedures are exact procedures [8, 9], which use relaxation techniques within a branch-and-bound framework, and a vast assortment of heuristic approaches [2, 4, 5–7, 10–12] which rely on everything from Lagrangean relaxation to Monte Carlo techniques to solve this single-tour case.

The remainder of this paper is divided into four sections. In Section 2, the MTMCP is formulated as an integer programming problem. Section 3 presents a heuristic solution procedure which is designed to handle the case when $m \geqslant 1$. In this section a simple numerical example is solved to illustrate the solution procedure. In Section 4, computational results for the heuristic solution procedure are presented. Exact solutions are used as a reference point to evaluate the effectiveness of the heuristic algorithm. Results which illustrate the different characteristics of the heuristic are also presented. Finally, Section 5 provides a brief statement of conclusions.

2. FORMULATION

The mathematical formulation of the MTMCP, found in Fig. 1, corresponds to the problem description as stated in Section 1. In this formulation the variables are defined as follows:

$x_{ijk} = 1$, if node j is visited immediately following node i on tour k
$x_{ijk} = 0$, otherwise
y_{ijk} = the flow from node i to node j on tour k

and the problem data is given by:

c_i = the reward associated with node i
b_i = the collection time at node i
d_{ij} = the time to travel from node i to node j
T = the maximum time allowed to complete each tour
m = the number of tours
n = the number of nodes in the graph (excluding the depot, which is denoted by node 0).

Equation (2.1) states that the total reward collected is to be maximized over all of the tours. Equations (2.2) and (2.3) ensure that each node is visited on at most one tour, except for the depot which is to be visited on each of the m tours. Equation (2.4) represents the tour continuity constraint, that is, this constraint guarantees that if a node is entered, it must also be exited on the same tour. Equation (2.5) is the time constraint, which forces one to complete each tour within T time units, including the time it takes to collect the rewards.

Equations (2.6)–(2.9) are the subtour elimination constraints which force all tours to include the depot. These constraints are very similar to those which are developed by Gavish and Graves [13] for the TSP. The only difference is that they have been generalized for the multiple tour case. To understand what these constraints are trying to accomplish, imagine each tour as a delivery problem. In each problem, we are to deliver one item to each of the p nodes. All of the items to be delivered are stored at the depot and the p nodes must all receive delivery on the same tour. So, as we travel around this tour, we have one less item when we leave a node than when we enter a node. Finally, when we travel from the last node to the depot, we return with no items.

$$\text{maximize} \sum_{k\in M}\sum_{i\in N}\sum_{j\in D} c_i x_{ijk} \qquad (2.1)$$

subject to:

$$\sum_{k\in M}\sum_{j\in D} x_{ijk} \le 1 \qquad i\in N \qquad (2.2)$$

$$\sum_{j\in N} x_{0jk} = 1 \qquad k\in M \qquad (2.3)$$

$$\sum_{j\in D} x_{ijk} - \sum_{j\in D} x_{jik} = 0 \qquad i\in D,\ k\in M \qquad (2.4)$$

$$\sum_{i\in D}\sum_{j\in D} d_{ij} x_{ijk} + \sum_{i\in N}\sum_{j\in D} b_i x_{ijk} \le T,\ k\in M \qquad (2.5)$$

$$y_{ijk} \le (n-m+1)\, x_{ijk} \qquad i\in D,\ j\in N,\ k\in M \qquad (2.6)$$

$$y_{i0k} = 0 \qquad i\in D,\ k\in M \qquad (2.7)$$

$$\sum_{j\in N} y_{0jk} = \sum_{i\in N}\sum_{j\in D} x_{ijk} \qquad k\in M \qquad (2.8)$$

$$\sum_{j\in D} y_{jik} - \sum_{j\in D} y_{ijk} \le 1 \qquad i\in N,\ k\in M \qquad (2.9)$$

$$x_{ijk} = 0, 1 \qquad i\in D,\ j\in D,\ k\in M \qquad (2.10)$$

$$y_{ijk} \ge 0, \text{ integer} \qquad i\in D,\ j\in D,\ k\in M \qquad (2.11)$$

$$N = \{1,2,\ldots,n\},\ M = \{1,2,\ldots,m\},\ D = \{0,1,\ldots,n\}$$

Fig. 1. Mathematical formulation of the MTMCP.

This mathematical programming formulation was not developed with the idea of solving the MTMCP, but to provide insights into the structure of the problem. In Section 1, we observed that the MTMCP falls within a class of *NP*-hard problems, so solving even a moderate sized problem with this formulation would be very impractical. In view of the complexity of this problem, we were led to consider a heuristic procedure to solve the MTMCP, which we present in the next section.

3. HEURISTIC PROCEDURE

The heuristic solution procedure described in this section is titled MAXIMP, for MAXimize IMPortance (reward).

MAXIMP is based upon a very simple transformation of the objective function of the MTMCP. This transformation is derived from the following. On each tour we are allowed to use as many as T time units for collection and travel. So it would appear, that to maximize the total reward collected, we would want to use as much time as is allotted, in order to collect from a greater number of nodes. If we would use T time units on every tour, then the reward collected per time unit would be the total reward collected on the m tours divided by mT. Note, that under the assumption that all mT time units are used, maximizing either the total reward collected on all of the tours or the reward collected per time unit would yield an equivalent solution (since they differ by only the constant, mT). Therefore, MAXIMP considers maximizing the latter. While in reality we do realize that the assumption that all mT time units are used may not be satisfied, we have found through computational testing that this heuristic approach is still very effective.

In the discussion which follows, we will show that MAXIMP produces a solution to the MTMCP by four basic steps:

(1) assignment of weights to node pairs
(2) preprocessing of node pairs
(3) assignment of nodes to tours
(4) a final tour check to see if visiting a single node on a tour increases the total reward collected on that tour.

Here we define a node pair, denoted (i, j), to be a set of the two nodes, i and j.

To construct a set of tours with a high reward per time unit, we need some sort of criteria for choosing the nodes which will be placed in the same tour. MAXIMP resolves this problem through a weighting scheme. Weights are assigned to each node pair such that the greater the weight, the more beneficial it is to not only visit those two nodes, but to visit them on the same tour. As a result, the best two-node combinations are determined.

The weighting scheme of MAXIMP is based upon a convex combination of two distinct weight types. Using the same notation as in Section 2 and assuming that all of the travel times are symmetrical, the two weight types are defined as follows:

$$W_{1ij} = [(c_i + c_j)/T] * t_{ij} \tag{3.1}$$

and

$$W_{2ij} = [(c_i + c_j)/t_{ij}] * T \tag{3.2}$$

where,

$$t_{ij} = d_{0i} + d_{ij} + d_{j0} + b_i + b_j.$$

Therefore, the weight for each node pair (i, j) is defined as:

$$W_{ij}(\alpha) = \alpha W_{1ij} + (1-\alpha) W_{2ij}, \quad \text{for } \alpha \in [0, 1]. \tag{3.3}$$

The two weight types take different approaches in defining the best two-node combinations. Let us look at W_{1ij}. First, note that the quantity $(c_i + c_j)$ is the total reward collected on tour k if only nodes i and j are visited on k. This two-node tour must begin at the depot, visit and collect from i and j, and then return to the depot. So assuming that all T time units are used on tour k, a lower bound on the reward per time unit would be $(c_i + c_j)/T$. But in actuality we know that the time to

complete this tour is t_{ij}, and if $t_{ij} < T$, then we could possibly place other nodes in this tour as well. So with a little rearranging, we can see that this weight type defines the best two node combinations as those node pairs in which the greatest reward is collected for the actual fraction of the total tour time (t_{ij}/T) needed to visit and collect the rewards.

Unlike W_{1ij}, W_{2ij} considers only the time needed to visit and collect from nodes i and j. This weight type is simply the ratio of the reward collected vs the actual time needed to collect from i and j (i.e. $[c_i + c_j]/t_{ij}$). [T appears in W_{2ij} only as a scaling factor; without this factor, W_{1ij} would dominate W_{2ij} in equation (3.3).] So this weight type suggests that the best two node combinations are those node pairs whose ratio of reward vs travel and collection time is the greatest.

The purpose of using both weight types in the weighting scheme is 2-fold. First of all, we have found that the distribution of the nodes in the graph plays an important role in the solution of the MTMCP. And secondly, we have observed that both weight types alone can be used by MAXIMP to construct the optimal set of tours to some MTMCPs. If we look back to equation (3.1), we see that W_{1ij} would yield the greatest values when t_{ij} is large. Therefore, the node pairs which are the greatest distance from the depot would be favored by this weight type. Conversely, equation (3.2) shows us that W_{2ij} yields the greatest values when t_{ij} is small. In this case, those node pairs which are the closest to the depot are favored. So by varying α, in equation (3.3), we could conceivably change the weights such that any node pair located in the graph is favored. Thus, alternative solutions to the MTMCP can be determined by solving the problem for several values of α. In this way, we can at least minimize the dependency of the heuristic on the distribution of the nodes, by choosing the best solution from the alternatives.

To aid in the assignment of nodes to tours, we need to define WGT to be a list of the node pairs in decreasing order of their corresponding weight, $W_{ij}(\alpha)$. We also need to define RWD to be a list of the node indices in decreasing order of their corresponding rewards.

Before we begin assigning nodes to tours, observe that it is possible that there are pairs of nodes in the graph which can never be assigned to the same tour (i.e. when $t_{ij} > T$). Thus, these node pairs could be removed to help reduce the size of the problem. MAXIMP does just that, by performing a preprocessing step in which any node pair (i, j) where $t_{ij} > T$ is removed from WGT.

Once preprocessing is complete, we are ready to assign nodes to tours. MAXIMP constructs each of the m tours using the same procedure. So without loss of generality, suppose that we are currently constructing tour k. The first two nodes to be assigned to k, correspond to the node pair currently at the top of WGT. This initial assignment is the only time in which multiple nodes are assigned to k, subsequent assignments involve inserting only one node into the tour at a time. Candidate nodes are determined by moving down the WGT list from our present position. We stop at the first node pair that contains one node which has been assigned to k, and one node, denoted Y, which has not been assigned to any tour (the candidate node). An attempt is then made to insert Y into k. That is, we must determine if there exists a tour that includes the depot, the nodes assigned to k, and Y which does not violate the time constraint, T. To do this, we first solve the associated TSP. Next, we determine if the time it takes to travel the tour, generated by the TSP, plus the time it takes to collect from each of the nodes is greater than T time units. If it is not, then Y is assigned to k and we begin looking for the next candidate node by once again returning to the top of the WGT list and following the same procedure. Otherwise, we look for the next candidate node by moving down the WGT list from our present position. This process is repeated until we reach the end of the WGT list, at which time all candidate nodes have been explored. We then proceed to the final tour check.

Since MAXIMP initially assigned two nodes to k, all single node tours went unconsidered. But, there are cases in which a single node cannot be visited on a tour with any others, or at least not with any unassigned nodes. Therefore, the purpose of this final tour check is to determine if the reward associated with the node at the top of RWD (which is currently the node having the greatest reward) is greater than the total reward that would be collected from those nodes already assigned to k. If it is, then those nodes which are presently assigned to k are removed, and only the node at the top of RWD will be assigned to k.

Once the final tour check has been completed, all of the node pairs and indices which correspond to a node assigned to k are removed from WGT and RWD, respectively. The same procedure is then repeated for the construction of the next tour.

Here, we summarize MAXIMP.

Initialization. Construct RWD, which is a list of the node indices in descending order of their corresponding reward. (RWD does not contain the depot.)

Weight assignments. For $\alpha \in [0, 1]$, calculate the weight, $W_{ij}(\alpha)$, for every node pair (i, j), in the graph, where $i \neq j$, $i \neq 0$ and $j \neq 0$. Construct WGT which is a list of the node pairs in descending order of their corresponding weights.

Preprocessing. Remove any node pair from WGT for which $t_{ij} > T$.

Assignment of nodes to a tour. Assign nodes to tour k as follows:

(1) Assign the two nodes corresponding to the node pair at the top of the WGT list to k. (If WGT is empty, begin the final tour check.)
(2) Determine a candidate node by moving down the WGT list from the present position until a node pair is found which contains one node assigned to k and one node, Y, which is unassigned (Y is the candidate node). (If a candidate node is not found, begin the final tour check.)
(3) Solve the TSP corresponding to the depot, the nodes assigned to k and Y.
(4) If the time needed to travel the TSP tour plus the time to collect from the nodes in the TSP tour is not greater than T time units, then assign Y to k, move to the top of WGT, and repeat Step 2.
Otherwise, repeat Step 2.

Final tour check.

(1) If the reward associated with the node at the top of RWD, denoted Z, is greater than the sum of the rewards collected from the nodes assigned to k, then remove the nodes currently assigned to k (these nodes are again considered unassigned) and assign only node Z to k.
(2) Remove all node pairs and indices which correspond to a node assigned to k from WGT and RWD, respectively.
(3) If all m tours have been constructed, STOP!
Otherwise, begin the assignment of nodes to the next tour.

Note that due to the way in which the tours are constructed with this solution procedure, MAXIMP will always produce a feasible solution, if one exists. MAXIMP can also be generalized to the asymmetric case by simply setting $t_{ij} = \min[d_{0i} + d_{ij} + d_{j0} + b_i + b_j, d_{0j} + d_{ji} + d_{i0} + b_i + b_j]$. (The procedure, as stated above, assumes that for every node i in the graph, $d_{0i} + d_{i0} + b_i \leqslant T$.)

Next, we solve a small example to demonstrate MAXIMP. The problem is defined as follows:

$m = 2$; $n = 6$; $T = 40$

$b_i = 1$, for $i = 1, 2, \ldots, 6$

$c_1 = 1$ $c_4 = 8$

$c_2 = 5$ $c_5 = 5$

$c_3 = 4$ $c_6 = 8$

d_{ij}:	0	1	2	3	4	5	6
0	—	1	8	15	9	9	14
1	1	—	9	15	8	9	15
2	8	9	—	9	15	15	7
3	15	15	9	—	18	23	11
4	9	8	15	18	—	15	22
5	9	9	15	23	15	—	18
6	14	15	7	11	22	18	—

A network representation of this problem can be found in Fig. 2.

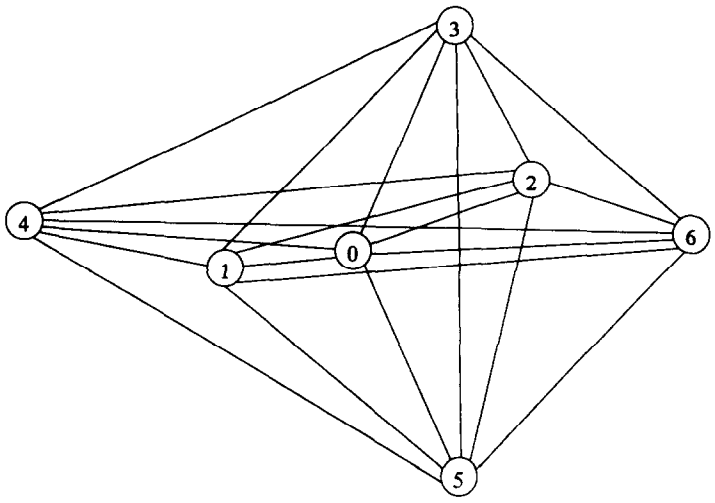

Fig. 2. Network for the example problem.

Table 1. Lists for the example problem

RWD		WGT	
Node index (*i*)	Reward (*Ci*)	Node pair (*i*, *j*)	*Wij*(0.5)
4	8	(2,6)	13.425
6	8	(2,4)	13.172
2	5	(4,5)	13.116
5	5	(1,4)	11.250
3	4	(2,5)	10.132
1	1	(1,6)	9.225
		(2,3)	9.119
		(1,2)	7.500
		(1,5)	7.289
		(1,3)	5.093

This problem was solved with $\alpha = 0.5$. The two lists, RWD and WGT, corresponding to this example are found in Table 1. In this table, WGT appears after preprocessing has taken place.

Solution. ($\alpha = 0.5$).

$$\text{tour } 1 = \{0-1-2-6-0\} \qquad \text{Reward} = 14$$

$$\text{tour } 2 = \{0-4-5-0\} \qquad \text{Reward} = 13$$

$$\text{Total Reward} = 27$$

Therefore, with $\alpha = 0.5$, the total reward collected using MAXIMP was 27. This happens to be the optimal solution to this problem as well. The tours corresponding to the solution are pictured in Fig. 3.

4. COMPUTATIONAL RESULTS

To facilitate testing, MAXIMP was coded in Fortran 77 and all of the test problems were solved using the IBM 3090/600 mainframe computer at Penn State University. The TSP, associated with tour construction, was solved heuristically using the Farthest Insertion Algorithm [14] (all of the nodes were used as the starting node, and the best tour was chosen from the alternatives).

Test problems were randomly generated by placing a specified number of nodes in a graph then setting the travel time between nodes i and j equal to the Euclidean distance between nodes i and j. Thus, all the test problems were symmetric. The reward and the collection time was also randomly generated for each node in the graph. A description of the sets of test problems is listed in Table 2.

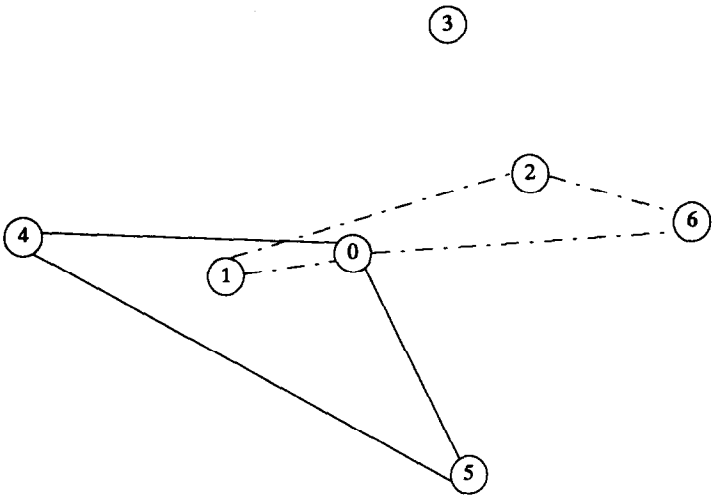

Fig. 3. Heuristic solution to the example problem.

Table 2. Test problem sets

Set No.	Nodes (n)	Tours (m)	Number of instances
1	10	2	10
2	15	3	5
3	20	3	5
4	50	8	5
5	100	15	5

Table 3. Results of the 10 node instances (set 1)

Problem	No. 1	No. 2	No. 3	No. 4	No. 5	No. 6	No. 7	No. 8	No. 9	No. 10
Optimal										
Solution	336	526	386	255	275	388	331	433	407	409
cpu (s)	704.26	809.55	107.68	75.21	56.57	563.53	11.31	73.10	158.78	779.71
No. of nodes	7	9	6	4	5	7	4	9	6	7
MAXIMP										
$\alpha=0.0$										
Solution	304	513	386	243	262	385	258	413	388	402
cpu (s)	0.020	0.031	0.014	0.011	0.015	0.026	0.011	0.035	0.025	0.023
No. of nodes	6	8	6	5	5	8	4	8	8	6
$\alpha=0.25$										
Solution	304	513	386	243	262	385	331	413	388	402
cpu (s)	0.016	0.025	0.008	0.007	0.010	0.021	0.005	0.030	0.019	0.019
No. of nodes	6	8	6	5	5	8	4	8	8	6
$\alpha=0.50$										
Solution	304	513	386	243	275	385	331	413	407	402
cpu (s)	0.014	0.024	0.008	0.005	0.009	0.020	0.005	0.029	0.017	0.018
No. of nodes	6	8	6	5	5	8	4	8	6	6
$\alpha=0.75$										
Solution	336	513	386	255	275	352	331	433	407	402
cpu (s)	0.012	0.019	0.008	0.005	0.009	0.019	0.005	0.020	0.016	0.017
No. of nodes	7	8	6	4	5	6	4	9	6	6
$\alpha=1.00$										
Solution	336	513	386	255	275	307	331	433	407	369
cpu (s)	0.011	0.019	0.008	0.005	0.008	0.009	0.004	0.020	0.016	0.011
No. of nodes	7	8	6	4	5	5	4	8	6	5
$\alpha=0.00$, 0.25, 0.50, 0.75 and 1.00										
Total cpu (s)	0.073	0.118	0.046	0.033	0.051	0.095	0.030	0.134	0.093	0.088
Average cpu (s)	0.0146	0.0236	0.0092	0.0066	0.0102	0.0190	0.0060	0.0268	0.0186	0.0176

In all problem instances, T was chosen such that:

(a) It is possible for each node in the graph to be assigned to at least one feasible tour, even if that tour consists of only that node. That is, $d_{0i}+d_{i0}+b_i \leqslant T$, for $i=1, \ldots, n$.

(b) It is not possible to visit all n nodes in the graph on m tours, where each of the m tours must be completed in T time units.

Exact solutions to small MTMCPs (10, 15 and 20 node problems) were found by first formulating the problems using the mathematical formulation of Section 2, and then employing MPSX to solve the problems to optimality. Using MAXIMP, each of the test problems was solved for five values of α (0.0, 0.25, 0.50, 0.75, 1.00). The results are found in Tables 3–7.

In Tables 3–7, the optimal solutions as well as the heuristic solutions to the test problems are summarized. From these tables, we find that the heuristic solution was optimal in 13 out of the 17 instances which were solved to optimality, using MPSX. And in the worst case, the heuristic solution was within 5.7% of the optimal solution. Furthermore, we see that the computation time needed to find the exact solution is several orders of magnitude greater than the total computation time needed to find the heuristic solution with all five values of α.

The number of nodes in the best solution is in effect a result of the tightness of the time constraint on the tours. We see from Tables 3–7 that as the number of nodes in the solution increases, the computation time of MAXIMP also increases. This result is expected, since a tight time constraint would imply that fewer nodes could be assigned to any one tour, thus reducing the size and the number of the TSPs which must be solved during the assignment of nodes to tours. Also during testing, it was noted that the computation time of MAXIMP was much more dependent on the number of nodes in the problem instance than on the number of tours.

Again looking at Tables 3–7, we can spot instances in which each of the five α values produces the best solution. Yet, the best solution was captured by the α values of 0.0, 0.5, and 1.0 in all but two cases. So while the use of all five α values results in a solution in a reasonable amount of time, the use of only the three α values mentioned above may be more easily justified. In effect, we could

Table 4. Results of the 15 node instances (set 2)

Problem	No. 1	No. 2	No. 3	No. 4	No. 5
Optimal					
Solution	613	634	870	615	867
cpu (s)	1417.55	9267.11	9880.84	6204.06	1308.88
No. of nodes	13	10	12	11	13
MAXIMP					
$\alpha=0.00$					
Solution	554	552	870	598	818
cpu (s)	0.141	0.058	0.067	0.072	0.119
No. of nodes	12	10	12	11	13
$\alpha=0.25$					
Solution	554	552	870	598	818
cpu (s)	0.136	0.050	0.061	0.066	0.115
No. of nodes	12	10	12	11	13
$\alpha=0.50$					
Solution	554	589	870	609	818
cpu (s)	0.138	0.040	0.061	0.065	0.114
No. of nodes	12	11	12	11	13
$\alpha=0.75$					
Solution	554	634	870	615	818
cpu (s)	0.134	0.027	0.055	0.040	0.094
No. of nodes	12	10	12	10	13
$\alpha=1.00$					
Solution	613	634	660	615	718
cpu (s)	0.077	0.030	0.026	0.039	0.043
No. of nodes	13	10	8	10	10
$\alpha=0.0, 0.25, 0.50, 0.75$ and 1.00					
Total cpu (s)	0.626	0.205	0.270	0.282	0.485
Average cpu (s)	0.1252	0.0410	0.0540	0.0564	0.0970

Table 5. Results of the 20 node instances (set 3)

Problem	No. 1	No. 2	No. 3	No. 4	No. 5
Optimal					
Solution	685	751			
cpu (s)	4535.58	8130.26			
No. of nodes	13	15			
MAXIMP					
$\alpha=0.00$					
Solution	685	751	926	662	806
cpu (s)	0.313	0.262	0.289	0.269	0.150
No. of nodes	13	15	16	15	12
$\alpha=0.25$					
Solution	685	736	859	671	806
cpu (s)	0.310	0.248	0.281	0.264	0.144
No. of nodes	13	14	14	14	12
$\alpha=0.50$					
Solution	685	736	872	644	809
cpu (s)	0.301	0.247	0.273	0.261	0.140
No. of nodes	13	14	14	13	12
$\alpha=0.75$					
Solution	644	734	749	594	737
cpu (s)	0.241	0.129	0.097	0.086	0.113
No. of nodes	13	14	11	10	11
$\alpha=1.00$					
Solution	651	685	752	586	740
cpu (s)	0.092	0.062	0.097	0.089	0.058
No. of nodes	13	9	12	10	11
$\alpha=0.00$, 0.25, 0.50, 0.75 and 1.00					
Total cpu (s)	1.257	0.948	1.037	0.969	0.605
Average cpu (s)	0.2514	0.1896	0.2074	0.1938	0.1210

Table 6. Results of the 50 node instances (set 4)

Problem	No. 1	No. 2	No. 3	No. 4	No. 5
MAXIMP					
$\alpha=0.00$					
Solutuon	2231	2166	2160	2152	1959
cpu (s)	11.339	7.716	4.036	7.884	3.997
No. of nodes	45	42	41	42	40
$\alpha=0.25$					
Solution	2231	2147	2160	2152	1914
cpu (s)	11.479	7.556	4.064	7.850	3.910
No. of nodes	45	41	41	42	37
$\alpha=0.50$					
Solution	2312	2249	2160	2203	1935
cpu (s)	11.612	7.581	4.106	7.663	3.932
No. of nodes	48	44	41	43	37
$\alpha=0.75$					
Solution	2165	2182	2059	2140	1710
cpu (s)	4.801	4.826	2.651	2.764	1.601
No. of nodes	43	41	36	40	31
$\alpha=1.00$					
Solution	2135	1881	1878	2073	1486
cpu (s)	2.476	1.931	1.598	1.993	1.293
No. of nodes	40	29	28	37	24
$\alpha=0.00$, 0.25, 0.50, 0.75 and 1.00					
Total cpu (s)	41.707	29.610	16.455	28.154	14.733
Average cpu (s)	8.3414	5.9220	3.2910	5.6308	2.9466

reduce the total computation time by *c.* 40%. It also appears that as the number of nodes increases, α values >0.5 no longer produce the best solutions. Therefore, based on the problems tested, it may be more practical to concentrate on α values which do not exceed 0.5 when $n \geqslant 20$, or at least when $n \geqslant 50$.

Finally, an interesting observation was made concerning $\alpha=1.0$. We found that the tours generated by this value of α are very balanced. That is, on all of the tours, the number of nodes visited is almost equal. With other values of α this was rarely the case, especially as α approached zero.

Table 7. Results of the 100 node instances (set 5)

Problem	No. 1	No. 2	No. 3	No. 4	No. 5
MAXIMP					
$\alpha = 0.00$					
Solution	4688	4804	4303	4192	5140
cpu (s)	81.972	91.803	80.526	66.160	76.845
No. of nodes	87	91	93	84	91
$\alpha = 0.25$					
Solution	4688	4774	4289	4205	5150
cpu (s)	83.234	93.642	80.867	68.737	78.838
No. of nodes	87	88	92	79	92
$\alpha = 0.50$					
Solution	4704	4633	4244	4155	4961
cpu (s)	85.635	94.495	79.796	62.792	77.49
No. of nodes	90	80	84	79	81
$\alpha = 0.75$					
Solution	4630	4387	4293	3915	4910
cpu (s)	81.398	95.330	83.281	63.094	94.761
No. of nodes	81	74	88	63	79
$\alpha = 1.00$					
Solution	3989	3529	4103	3417	4123
cpu (s)	23.505	21.249	27.724	21.328	25.206
No. of nodes	61	46	80	48	57
$\alpha = 0.00$, 0.25, 0.50, 0.75 and 1.00					
Total cpu (s)	355.744	396.519	352.194	282.111	353.140
Average cpu (s)	71.1488	79.3038	70.4388	56.4222	70.6280

5. CONCLUSION

In this paper a heuristic solution procedure was presented for the multiple tour maximum collection problem (MTMCP).

Testing verified that the proposed heuristic procedure is an effective tool for solving the MTMCP. It was shown that the quality of the solution and the time needed to solve the problem were both very acceptable. In addition, it was demonstrated that the heuristic could solve problems with as many as 100 nodes in *c.* 70 s.

It is hoped that future study of the MTMCP may develop an exact solution procedure which exploits the network structures contained in the proposed integer formulation.

REFERENCES

1. B. Golden, A. Assad and R. Dahl, Analysis of a large scale vehicle routing problem with an inventory component. *Large Scale Syst.* **7**, 181–190 (1984).
2. E. Balas, The prize collecting traveling salesman problem. *Networks* **19**, 621–636 (1989).
3. S. Davis, N. Ceto Jr and J. M. Rabb, A comprehensive check processing simulation model. *J. Bank Res.* **13**, 185–194 (1982).
4. B. Golden, L. Levy and R. Vohra, The orienteering problem. *Naval. Res. Logist.* **34**, 307–318 (1987).
5. C. Keller and M. Goodchild, The multiobjective vending problem: a generalization of the travelling salesman problem. *Environ. Plan. B: Plan. Design* **15**, 447–460 (1988).
6. D. H. Gensch, An industrial application of the traveling salesman's subtour problem. *AIIE Trans.* **10**, 362–370 (1978).
7. R. Ramesh and K. Brown, An efficient four-phase heuristic for the generalized orienteering problem. *Computers Ops Res.* **18**, 151–165 (1991).
8. S. Kataoka and S. Morito, An algorithm for the single constraint maximum collection problem. *J. Ops Res. soc. Japan* **31**, 515–531 (1988).
9. R. Ramesh, Y. Yoon and M. Karwan, An optimal algorithm for the orienteering tour problem. *ORSA J. Comput.* **4**, 155–165 (1992).
10. T. Tsiligirides, Heuristic methods applied to orienteering. *J. Ops Res. Soc.* **35**, 797–809 (1984).
11. B. Golden, Q. Wang and L. Liu, A multifaceted heuristic for the orienteering problem. *Naval Res. Logist.* **35**, 359–366 (1988).
12. B. Golden, L. Levy and R. Dahl, Two generalizations of the traveling salesman problem. *OMEGA* **9**, 439–441 (1981).
13. B. Gavish and S. Graves, The traveling salesman problem and related problems. Working paper, Graduate School of Management, University of Rochester (1978).
14. M. Syslo, N. Deo and J. Kowalik, *Discrete Optimization Algorithms with Pascal Programs*, pp. 361–367. Prentice-Hall, Englewood Cliffs, N.J. (1983).