PERGAMON

Computers & Operations Research 26 (1999) 427–441

computers & operations research

An optimal solution procedure for the multiple tour maximum collection problem using column generation

Steven E. Butt[a,*,1], David M. Ryan[b,2]

[a] *Department of Industrial and Manufacturing Engineering, Western Michigan University, Kalamazoo, MI 49008, U.S.A.*
[b] *Department of Engineering Science, The University of Auckland, Private Bag 92019, Auckland, New Zealand*

Received January 1997; received in revised form August 1997

Scope and purpose

A common assumption of many vehicle routing and scheduling problems is that each customer is to be visited exactly once, on exactly one route (tour). In this paper, we consider a problem where this is not possible. We assume that the routes are constrained in such a way that all of the customers cannot be visited. For example, suppose that we have five 8 hour days to visit customers, but too many customers to visit in this amount of time.

In problems where it is possible to visit all customers, the typical objective is to determine the minimum amount of time (or distance) needed to make all of the visits. Conversely, if all of the customers cannot be visited, the main objective is to determine the optimal subset of customers to visit in the time allotted. In this paper we assume that there is a reward which is collected when visiting a customer, and that the problem objective is to maximize the total reward collected.

Abstract

The Multiple Tour Maximum Collection Problem (MTMCP) is closely related to the classical Traveling Salesman and Vehicle Routing Problems. One major difference with the MTMCP is that it is not possible to

* Corresponding author. Tel.: 001 616 387 3746; fax: 001 616 387 4075; e-mail: steven.butt@wmich.edu.

[1] Steven E. Butt is an Assistant Professor in the Department of Industrial and Manufacturing Engineering at Western Michigan University. He holds a B.A. in Mathematics and Chemistry from Earlham College and a M.S. and Ph.D. in Industrial Engineering and Operations Research from the Pennsylvania State University. Dr. Butt's research interests are in the areas of facility location, vehicle routing and employee scheduling. His work has appeared in *Computers and Operations Research*, *European Journal of Operational Research*, and *Socio-Economics Planning Sciences*.

[2] David M. Ryan is Professor of Operations Research and Head of the Department of Engineering Science at the University of Auckland, New Zealand. Professor Ryan has interests in the application of optimization methods to solve practical problems in business and industry, and he has published results of his work in journals such as the *European Journal of the Operational Research Society*, *Operations Research*, and *Mathematical Programming*. He has a particular interest in the use of set partitioning optimization methods to solve crew scheduling problems.

0305-0548/99/$ – see front matter
PII: S0305-0548(98)00071-9

visit all of the nodes in the graph in the total time allocated on a given set of tours. However, with the MTMCP, a reward is collected from each node that is visited. The objective of the MTMCP is to determine which nodes to visit on m time-constrained tours for which the total reward collected is maximized. It is assumed that each tour begins and ends at a central depot and that each node can be visited no more than one time, on at most one tour. In this paper, an optimal solution procedure is presented for the MTMCP. This procedure is based on a set-partitioning formulation and makes efficient use of both column generation and constraint branching. Numerical results are presented which show that this optimal procedure can be used to solve problems of realistic size in a reasonable amount of time.

Keywords: Vehicle routing; Traveling salesman; Constraint branching; Column generation; Optimization

1. Introduction

Vehicle routing analysis is still dominated by the assumption that all customers (nodes) can and must be visited. Two classical examples of this type of model are the Traveling Salesman Problem (TSP) and the Vehicle Routing Problem (VRP). There are many practical applications to which such models are not directly applicable. In this paper, we consider a problem in which the time to visit the customers is constrained such that all of the customers cannot be visited. Specifically, we consider the Multiple Tour Maximum Collection Problem (MTMCP).

The MTMCP is defined on a complete graph $G = [N, A]$, where $N = \{0, 1, 2, \ldots, n\}$ and $A = \{a_{ij} \mid i, j \in N\}$ are the node and arc sets, respectively. $D = [d_{ij}]$ is the travel time matrix, where d_{ij} is the time required to traverse arc a_{ij}. The reward for visiting $i \in N$ is r_i and the time necessary to collect r_i is b_i units (i.e., the visit or collection time). Node 0 denotes the depot, with $r_0 = 0$ and $b_0 = 0$, and m is the maximum number of tours which can be taken through G. The m tours are partitioned into q sets, such that each tour in set k must be completed in at most T_k time units, $k = 1, \ldots, q$ (this includes the time needed to collect rewards). The number of tours in set k is equal to t_k, $k = 1, \ldots, q$, which implies that $m = \sum_{k=1}^{q} t_k$. Now, with respect to set k, a feasible tour is defined as a cycle through a subset of nodes in G which begins and ends at the depot, visits each node in the subset *exactly* once, and is completed in *at most* T_k time units. Finally, the objective of the MTMCP is to find at most m nonoverlapping feasible tours through G which maximize the total reward collected.

There are several practical applications of the MTMCP found in the literature. Examples include recruiting athletes [1], underway replenishment of a dispersed carrier battlegroup [2, 3], the daily selection of customers for a fleet of trucks used to deliver fuel [4], and scheduling branch messengers at a commercial bank [5]. The authors' first exposure to this problem was through the first example in which a college recruiter is given a set number of days to recruit football players from area high schools. Due to the number of high schools within driving distance, it is known that all high schools cannot be visited in the total time allotted. Other constraints include the fact that the recruiter is not allowed any overnight stays (i.e., the recruiter must begin and end on the college campus each day) and the fact that there is a limited amount of time each day that the recruiter is allowed to visit the high schools (e.g., 8 a.m.–4 p.m.). A reward is assigned to each high school based on its recruiting potential (i.e., likelihood of encouraging a number of athletes to attend the college

if the high school is visited) and the objective of this problem is to maximize the recruiting potential by visiting the optimal set of high schools in the time allotted.

Solution procedures for these problems are divided into those which consider a single tour through G and those which consider multiple tours through G. In the case when only one tour can be taken through G, the MTMCP has been studied under such names as: the Orienteering Problem (OP), the Maximum Collection Problem (MCP), the Time-Constrained Traveling Salesman Problem (TCTSP), and the Traveling Salesman's Subtour Problem (TSSP). Because this special case is known to be NP-hard (see [6]), most of the solution procedures developed have been heuristic. Such procedures are found in papers by Gensch [7], Golden et al. [6, 8, 9], Tsiligirides [10], and Ramesh and Brown [11]. Exact solution procedures are also presented by Kataoka and Morito [12] and Ramesh et al. [13]. Both of these procedures involve relaxation techniques imbedded within a branch and bound framework. It should be noted that the test problems studied with respect to the heuristics range in size from 10 to 60 nodes, while the size of test problems considered with the exact solution procedures ranged from 10 to 150 nodes (for the larger test problems see [13]).

There are other models in the literature which are closely related to the single tour case of the MTMCP. These include the Multi-objective Vending Problem (MVP) [14], the Prize Collecting Traveling Salesman Problem (PCTSP) [15], and the Maximum Collection Problem with Time Dependent Rewards (MCPTDR) [16, 17]. In each of these models, the only significant difference from the MTMCP is in the objective function. The MVP involves minimizing the total distance traveled, as well as maximizing the total reward collected. In the PCTSP, while a reward (prize) is collected at each node visited, a penalty is imposed for each node that is not visited. Finally, with the MCPTDTR, the rewards at the nodes do not have constant values, but rather, they decrease linearly with time.

The multiple tour case of the MTMCP has received very little attention in the literature. A heuristic solution procedure called MAXIMP is presented in [1]. This procedure is a greedy heuristic which builds the tours sequentially. A weighting scheme based on the rewards and travel times is used to make the assignment of nodes to the tours. It is shown that this heuristic provides very good solutions when compared to the optimal solution of problems with 10, 15, and 20 nodes. While problems with up to 100 nodes were studied, comparison to the optimal solution of these larger problems is not reported. The optimal solutions reported were determined using MPSX.

In comparison to the TSP and VRP, the MTMCP has an added element of complexity. The MTMCP requires selecting which subset of nodes in the graph to visit, as well as determining a calling order on each tour. If all of the nodes in G can be visited within the time constraints, then this added element of complexity disappears. This implies that the TSP and the VRP are special cases of the MTMCP. Therefore, since the TSP and VRP are known to be NP-hard, it follows that the MTMCP is NP-hard.

The remainder of this paper is divided as follows. Section 2 presents a set-partitioning formulation for the MTMCP. In Section 3 we propose a solution procedure based on our set-partitioning formulation. This procedure incorporates an efficient use of column generation and constraint branching. Also in this section we describe a method to avoid solving the same TSP twice. In Section 4 we report our numerical results. Finally, in Section 5 we furnish a brief summary.

2. Formulation

A mathematical formulation for the MTMCP is given by Butt and Cavalier in [1]. Their definition of the MTMCP differs slightly in that each tour must be completed within the same number of time units. The decision variables in their formulation correspond to the arcs between the nodes in the graph, hence, the optimal set of tours is determined by choosing an optimal set of arcs for each tour. As Butt and Cavalier [1] have shown through testing, solving the MTMCP to optimality using their formulation is extremely time consuming, especially when employing a commercial software package that does not take into account the inherent structure of the problem.

In the remainder of this section we discuss an alternative formulation for the MTMCP, which is based on the MTMCP definition presented in Section 1. To begin, we let I be the index set of all nodes in G, excluding the depot. That is, $I = \{i \mid i = 1, \ldots, n\}$. Since there is a finite number of distinct subsets of I, we define ρ_j to be the jth subset. This allows us to define the following:

$\mathrm{TSP}(\rho_j) =$ the total travel time corresponding to the optimal Traveling Salesman tour through the nodes in ρ_j and the depot.

$$J^k = \left\{ j \mid \mathrm{TSP}(\rho_j) + \sum_{i \in \rho_j} b_i \leqslant T_k \right\}; \quad k = 1, \ldots, q,$$

$$a_{ij}^k = \begin{cases} 1 & \text{if } i \in \rho_j \text{ and } j \in J^k, \\ 0 & \text{otherwise,} \end{cases}$$

$$x_j^k = \begin{cases} 1 & \text{if the nodes in } \rho_j \text{ are to be visited on one of the } t_k \text{ tours from set } k, \\ 0 & \text{otherwise.} \end{cases}$$

Finally, set the total reward collected from the nodes in ρ_j to c_j^k, where

$$c_j^k = \sum_{i \in \rho_j} r_i = \sum_{i=1}^{n} r_i a_{ij}^k; \; j \in J^k, \; k = 1, \ldots, q.$$

Given these definitions, the MTMCP can be formulated as follows:

$$\text{GSP: maximize } z = \sum_{k=1}^{q} \sum_{j \in J^k} c_j^k x_j^k, \tag{1}$$

$$\text{subject to} \quad \sum_{k=1}^{q} \sum_{j \in J^k} a_{ij}^k x_j^k \leqslant 1, \quad i = 1, \ldots, n, \tag{2}$$

$$\sum_{j \in J^k} x_j^k \leqslant t_k, \quad k = 1, \ldots, q, \tag{3}$$

$$x_j^k = 0, 1 \text{ integer.} \quad \text{for } j \in J^k; \; k = 1, \ldots, q. \tag{4}$$

In this formulation, the objective function in Eq. (1) is to maximize the total reward collected on the tours taken through G. Constraint (2) insures that each node is visited *at most* once, and on *no more*

$$\left[\begin{array}{c|c|c|c|c|c} \mathbf{A}^1 & \mathbf{A}^2 & \ldots & \mathbf{A}^q & \mathbf{I}_n. & \mathbf{0} \\ \hline \mathbf{L}^1 & \mathbf{L}^2 & \ldots & \mathbf{L}^q & \mathbf{0} & \mathbf{I}_q \end{array}\right] \left[\begin{array}{c} \mathbf{x}^1 \\ \mathbf{x}^2 \\ \vdots \\ \mathbf{x}^q \\ \mathbf{s} \end{array}\right] = \left[\begin{array}{c} \mathbf{e} \\ \mathbf{t} \end{array}\right]$$

Fig. 1. Matrix representation of the generalized set-partitioning formulation for the MTMCP.

than one tour. Constraint (3) states that *at most* t_k tours of length T_k can be taken through G. And finally, Constraint (4) restricts the decision variables to the integer values of 0 or 1. Note that unlike the formulation adopted by Butt and Cavalier, the tour and node feasibility constraints in GSP are implicitly defined and satisfied.

To provide a better understanding of this formulation, consider the matrix representation found in Fig. 1.

In this figure, it is assumed that the slack variables have been added, and are denoted by the vector **s**. The remaining notation is defined as

$$\mathbf{A}^k = [a_{ij}^k], \quad \mathbf{L}^k = \mathbf{e}_k \mathbf{e}^T, \quad \mathbf{x}^k = [\mathbf{x}_j^k] \quad \text{for } j \in J^k, \quad \mathbf{s} = \begin{bmatrix} s_1 \\ s_2 \\ \vdots \\ s_{n+q} \end{bmatrix} \quad \text{and} \quad \mathbf{t} = \begin{bmatrix} t_1 \\ t_2 \\ \vdots \\ t_q \end{bmatrix}.$$

Note that because of the non-unit right-hand side values of the tour constraints, this model is referred to as a *generalized set-partitioning model.*

3. Solution procedure

While there are several efficient algorithms available to solve a set-partitioning problem, most of these algorithms require that all columns be available at the initialization of the solution procedure. With respect to the MTMCP, when all columns are enumerated a priori, even the linear relaxation of a moderate-sized problem is very difficult to solve. Fortunately, it is possible to generate feasible columns, as needed, through a column generation procedure that we describe in Section 3.1.

The proposed solution procedure, which we call MAXREW, involves first solving the linear relaxation of GSP, and then employing a standard branch-and-bound procedure to achieve an integer solution. The innovative parts of MAXREW are located in the column generation procedure, tour storage procedure, and constraint branching procedure. These parts of MAXREW will be described in detail.

The ZIP package [18], which is known to be very efficient for solving linear programming (LP) relaxations of set-partitioning problems, is used to implement MAXREW. This package includes state-of-the-art routines for the solution of highly degenerate linear programs and the evaluation of branch-and-bound trees (which the user can customize through a set of user-defined subroutines).

Please note that in the discussion which follows, it is assumed that GSP has been transformed into an equivalent minimization problem. That is, Eq. (1) is converted to

$$-\text{ minimize } z = \sum_{k=1}^{q} \sum_{j \in J^k} -c_j^k x_j^k. \tag{5}$$

3.1. Column generation

During the initialization of MAXREW, the columns corresponding to the *single node tours* of the MTMCP are added to the model. A single node tour is defined as a tour which departs from the depot, visits one node in G, and then returns to the depot. (It is assumed that all single node tours are feasible for each maximum tour length T_k, $k = 1, \ldots, q$.) The remaining columns are added to the model via the column generator. The column generator is embedded within the *entering variable* subroutine of the LP solver and it is called each time an entering variable cannot be found among the current set of variables. Optimality is achieved when the generator fails to produce any new columns.

The column generator is invoked at a particular basic feasible solution of the LP relaxation at which $\pi^{\mathrm{T}} = \mathbf{c_B}^{\mathrm{T}} \mathbf{B}^{-1}$. At this basic feasible solution, the reduced cost of x_j^k, denoted $\mathrm{rc}(x_j^k)$, is

$$\mathrm{rc}(x_j^k) = -c_j^k - \sum_{i=1}^{n} \pi_i a_{ij}^k - \pi_{n+k} = \sum_{i=1}^{n} -r_i a_{ij}^k - \sum_{i=1}^{n} \pi_i a_{ij}^k - \pi_{n+k} = \sum_{i=1}^{n} \bar{r}_i a_{ij}^k - \pi_{n+k}, \tag{6}$$

where

$$\bar{r}_i = -r_i - \pi_i, \quad i \in I. \tag{7}$$

and π_{n+k} is the dual variable corresponding to the tour constraint of this column. (Recall that we are considering the minimization problem.)

We have named $\bar{r}_i$ the *adjusted reward* of node i. This adjusted reward gives us a measure of the attractiveness of a node. The most attractive node is the one with the most negative adjusted reward, since it decreases the value of the reduced cost by the greatest amount. To take advantage of this attractiveness measure, on each call to the column generator the nodes are ranked in increasing order of their adjusted rewards. That is, the nodes are ranked according to

$$\bar{r}_{(1)} \leqslant \bar{r}_{(2)} \leqslant \cdots \leqslant \bar{r}_{(p)} \leqslant 0 \leqslant \bar{r}_{(p+1)} \leqslant \cdots \leqslant \bar{r}_{(n)}, \tag{8}$$

where the number in parenthesis is an index specifying the rank of the corresponding node, with respect to its adjusted reward. Since adding a node with a positive adjusted reward to a column increases the reduced cost of that column, the column generator only constructs columns based on the p nodes which have negative adjusted rewards.

To ensure that only feasible columns are returned to the optimization, the corresponding tours cannot violate the associated maximum time constraint. Checking feasibility involves solving the corresponding TSP and determining if the minimum travel time plus the sum of the visit times exceeds the time constraint. While there are several methods for solving the TSP, we present results using both the branch-and-bound algorithm for the asymmetric Traveling Salesman Problem developed by Carpaneto et al. [19], and a Farthest Insertion heuristic.

The column generator builds columns based on a procedure called *Next Availables* (see e.g. [20]). The procedure begins by forming a set of "available" nodes which consists of the p nodes in G that have negative adjusted rewards. Once this set is defined, the node with the most negative adjusted reward is removed from the set and placed in the first position of the candidate tour. Since all single node tours are initially included in the model, the next step is to generate a set of available nodes for the second position of the candidate tour. This second set includes all nodes remaining in the first set.

From the set of second position availables, the node with the most negative adjusted reward is removed and placed in the second position of the candidate tour. (Note that the position in the candidate tour does not necessarily correspond to the calling order on the TSP tour.) At this point, if (a) the corresponding reduced cost is negative, and/or (b) the number of nodes in the candidate tour exceeds a specified limit, L_1, then the feasibility of the candidate tour will be checked with respect to all maximal tour lengths (T_k, $k = 1, \ldots, q$). The use of L_1 is important because it is possible to add several nodes to the candidate tour before the reduced cost of the corresponding column becomes negative, and solving a TSP on a candidate tour after it reaches a specified size helps to eliminate infeasible candidate tours before the TSPs become large.

If the candidate tour (consisting of the two nodes) passes the feasibility test for any tour length and the corresponding reduced cost is negative, then a column representing this tour is added to the model. Furthermore, if the candidate tour passes the feasibility test, then a set of available nodes is generated for the third position of the candidate tour. This third set consists of all nodes remaining in the previous (second) set. If the candidate tour violates all of the time constraints, then the node currently placed in the second position is discarded. From the set of availables for this position, the node with the most negative adjusted reward is removed from the set and placed in the second position of the candidate tour. We then return to the feasibility test.

If at any time during the procedure the set of availables for a position in the candidate tour is empty, we move back one position in the candidate tour and replace the node in that position from the list of availables. This process of checking feasibility, moving forward to produce a set of availables for the next position when the candidate tour satisfies at least one maximum tour length, and replacing a node in the candidate tour when all time constraints are violated, continues until we step back to the first position and the available list is empty. At this point, all possible tours have been enumerated without duplication.

In MAXREW, there are three conditions under which column generation is terminated:

(i) Complete enumeration has been performed.
(ii) All candidate tours have been enumerated for a node placed in the first position of the candidate tour and at least one column with a negative reduced cost is added to the model.
(iii) The number of columns generated is greater than or equal to a specified limit, L_2.

Conditions (ii) and (iii) are included in the column generation to speed up MAXREW. These conditions force the MAXREW to return to the optimization before a complete enumeration is performed. Condition (iii) is of primary importance in the early stages of the solution procedure when several feasible columns are being generated. Conversely, condition (ii) is more important in the latter stages, when fewer columns are generated on each call to the column generator.

3.2. Storing tours

Recall that during column generation a TSP must be solved on a node set to check the feasibility of the corresponding candidate tour. Because solving a TSP is computationally expensive, we want to avoid solving a TSP on the same set of nodes more than one time. This lead us to the development of a storage system which keeps track of the node sets for which a TSP has been solved.

After solving a TSP, the corresponding node set is stored in one of two tree structures. One structure stores the node sets which *cannot* be visited on a tour of length T_k (the infeasible tree), and the other tree stores the node sets that *can* be visited on a tour of length T_k (the feasible tree). Before solving a TSP, the column generator scans the infeasible and feasible tree structures to check if the corresponding TSP has been solved on a previous call. If the node set does not appear in one of the trees, the TSP is solved and the results are stored.

To illustrate the storage system, let us consider the infeasible tree for T_k. Assume that the sets $\{0, 1, 5, 2\}$, $\{0, 2, 6, 1\}$, $\{0, 3, 1, 5\}$, $\{0, 2, 5\}$, and $\{0, 6, 3, 5\}$ are subsets of the node indices of G and the time needed to travel the Traveling Salesman tour corresponding to each of these sets, plus the associated visit times, violates T_k. For simplicity, let us assume that $n = 6$ (i.e., there are six nodes in G excluding the depot). To store each of these sets in the infeasible tree for T_k, we first arrange the node indices in increasing order. The sets are then stored in a *binary tree with a header node*, as shown in Fig. 2. We will assume that the sets were stored in the order in which they appear above.

In this figure, the number inside each tree node corresponds to a node index of G and the letters in parentheses are labels that will be used to identify the individual tree nodes. A node in a binary tree has at most two link fields, one that points to its left subtree and one that points to its right subtree. We follow the convention described in [21, pp. 221–245], in which the left arc of a binary node is used to point to its "oldest" child (no left arc implies that there are no children) and the right arc is used to point to its "next" sibling (no right arc indicates that there are no other siblings "to the right"). Since binary nodes have at most two link fields, the outdegree of each node is at most 2.

In our tree structure, we also use a *header node*. The header node corresponds to the depot (node 0), which is assigned to each tour. Unlike the binary nodes, the header node will have an outdegree of at most $n - 1$, and all arcs emanating from the header node point to its children.

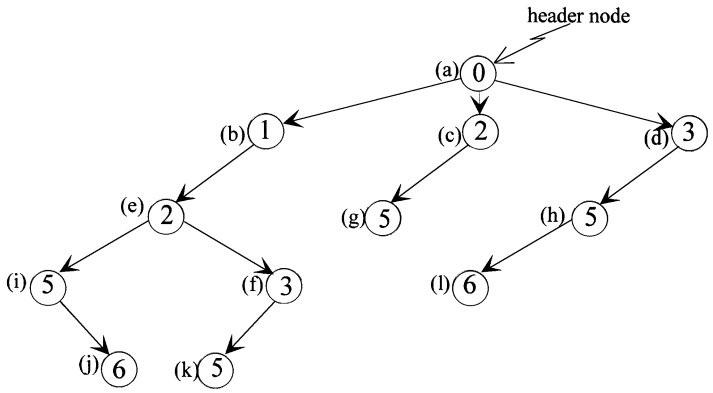

Fig. 2. Infeasible sets stored in a binary tree with a header node.

The benefit of this binary tree representation is that it can be stored as a linked list in a two-dimensional array. That is, we can construct an $r \times 3$ array in which (i) each row of the array corresponds to a binary node in the tree, (ii) column 1 holds the data stored at the tree node (i.e. a node index of G), (iii) column 2 stores the row number corresponding to the oldest child node (left subtree), and (iv) column 3 stores the row number corresponding to a sibling node (right subtree). A zero in column 2 or 3 is used to represent a null arc from the current tree node (row).

Since all single node tours of the MTMCP are feasible, the feasibility of these tours will never need to be checked. Therefore, the smallest infeasible tour contains at least two nodes, excluding the depot. This result, coupled with the arrangement of indices into ascending order before storage, is the reason that the header node has an outdegree of at most $n - 1$. We take advantage of this fact by not explicitly storing the header node, or any of its children. Their locations in the tree are implicitly defined by the first $n - 1$ rows of the storage array. For example, while the child of the header node whose index is w is not represented by a row in the storage array, its oldest child (if one exists) is stored in row w. That is, the first $n - 1$ rows correspond to the nodes located at the head of the arcs emanating from the children of the header node.

Given the discussion above, the tree in Fig. 2 can be stored as the array depicted in Fig. 3. Please note that only the smallest subset of nodes in G which force a tour to exceed T_k needs to be stored in the infeasible tree, since adding any other node to this subset results in a tour that is also infeasible. So, when the column generator checks the feasibility of a candidate tour and a subset of the nodes in the tour appears in the storage array, the candidate tour must be infeasible.

The *feasible* node sets for T_k are stored in an $r \times 3$ array which is based on the same binary tree structure used for the infeasible sets. The only difference is that the feasible sets must be stored explicitly. However, please note that any subset of a set appearing in the tree must also be feasible.

Finally, while we must define an infeasible and a feasible array for each maximum tour length (T_k, $k = 1, \ldots, q$), we do not build these arrays independent of one another. In fact, when we solve

Row (Tree Node)	Data (Node Index)	Left Subtree ("*Child*")	Right Subtree ("*Sibling*")
1(e)	2	6	8
2(g)	5	0	0
3(h)	5	10	0
4	0	0	0
5	0	0	0
6(i)	5	0	7
7(j)	6	0	0
8(f)	3	9	0
9(k)	5	0	0
10(l)	6	0	0
11	0	0	0
12	0	0	0
.	.	.	.
.	.	.	.
.	.	.	.
r			

Fig. 3. Array structure for the storage of infeasible sets of nodes.

a TSP with respect to a set of nodes in G, we check the feasibility with respect to all maximum tour lengths and store the node set in each appropriate array.

3.3. Constraint branching

In the branch-and-bound steps of MAXREW, we use constraint branching. So, instead of branching on a variable, we branch on a node pair (u, v), where $u, v \in G$. A 1-branch implies that if a feasible column includes u, it must also include v (and vice versa), and a 0-branch implies that a feasible column cannot include both u and v. Constraint branching has the advantage of banning several variables at each branch-and-bound node compared to a single variable with variable branching. As a result, the number of nodes in the branch-and-bound tree are significantly fewer and the size of the LP relaxations are much smaller.

It is known that a fractional solution of any set-partitioning model must contain at least one odd order cycle in the optimal basis, otherwise, the basis matrix is balanced and the solution is integer (see [20]). An odd order cycle implies that there exists at least one pair of constraint rows (nodes) that are covered fractionally together. Therefore, we can compute the fractional cover for each node pair (u, v) appearing in the basis, denoted $\mathrm{cov}(u, v)$, as follows:

$$\mathrm{cov}(u, v) = \sum_{k=1}^{q} \sum_{u,v \in \rho_j} x_j^k. \tag{9}$$

We then look for a node (row) pair for which

$$0 < \mathrm{cov}(u, v) < 1,$$

and impose a branch which either forces:

$$\mathrm{cov}(u, v) = 0 \quad \text{or} \quad \mathrm{cov}(u, v) = 1.$$

Either branch can be imposed by removing the variables which do not comply with the branch.

The node pair on which we branch is the one with the greatest fractional cover. Ties are broken by choosing the node pair which has the largest value for the following ratio:

$$\frac{r_u + r_v}{\min(d_{0u} + d_{uv} + d_{v0},\ d_{0v} + d_{vu} + d_{u0}) + (b_u + b_v)}. \tag{10}$$

While several complex rules were considered for breaking ties, this simple ratio seems to drastically reduce the number of branch-and-bound nodes explored. Intuitively it makes sense, since we want to branch on a node pair which gives us the greatest return for the time spent retrieving the rewards.

4. Numerical results

To test MAXREW, problems which closely simulate the football recruiting problem described in [1] were generated. We considered problems with 50, 75, and 100 nodes. The location of the nodes

were randomly placed in a Cartesian coordinate system within a 180 unit radius of the origin. (The depot was assumed to be situated at the origin.) The travel time between nodes was set equal to the Euclidean distance between the nodes. Visit (collection) times were randomly set to 30, 60, or 90 units, and rewards were randomly assigned from the range 1–100.

Each problem generated was solved by three methods on a SUN Sparc 5 (40 Mb, 85 mHz). The solution methods included:

(1) MAXREW1 MAXREW with an exact TSP algorithm [19]) ($L_1 = 2$, $L_2 = 10 *$ number of nodes in G).
(2) MAXREW2 MAXREW with a Farthest-Insertion TSP heuristic (each of the r nodes in the TSP are used as the starting node and the best of the r solutions is chosen). ($L_1 = 2$, $L_2 = 10 *$ number of nodes in G).
(3) MAXIMP (the 5 α-values described in the paper by Butt and Cavalier [1] are used and the best of the five solutions is chosen).

In the tables below, we include the average results of our experimentation with respect to each of the three solution methods. In these tables, IP Solution refers to the Integer solution, LP Solution refers to the LP relaxed solution of the corresponding MTMCP, and the No. of B and B nodes denotes the number of branch and bound nodes needed to find the Integer solution. We also include summary information about the number of nodes in the tours. Specifically, the average maximum and average minimum number of nodes found in any tour using each solution method, and the average number of nodes in all of the tours taken through G.

Table 1 presents the results of five 50 node instances with $q = 1$ and $t_1 = 5$. Each of five instances was solved three times with T_1 set to 480, 600, and 720 time units, respectively.

Table 2 considers the change in the number of tours taken through G. Here, a set of five 100 node instances were generated and then solved with 5, 10, and 15 tours. It was assumed that $q = 1$ and that $T_1 = 480$ time units.

Table 1
Changes in the tour length

Solution procedure	MAXREW1			MAXREW2			MAXIMP		
Time constraint (T_1)	480 units	600 units	720 units	480 units	600 units	720 units	480 units	600 units	720 units
Integer solution	1276.60	1626.40	1923.60	1276.60	1626.40	1923.60	1177.60	1504.60	1782.80
Solution time (min)	1.21	14.95	175.99	0.54	7.13	87.45	0.07	0.13	0.20
LP solution	1278.20	1634.74	1932.68	1278.20	1634.74	1932.68			
LP solution time (min)	0.99	9.22	91.37	0.33	2.45	22.08			
No. of B & B nodes	1.40	14.40	31.80	1.40	14.40	31.80			
Tours:									
Max. No. of nodes	4.40	5.60	7.20	4.40	5.60	7.00	5.00	6.40	7.60
Min. No. of nodes	2.60	4.00	5.00	2.60	4.00	5.00	2.60	3.20	4.00
Ave. No. of nodes	3.52	4.68	5.84	3.52	4.68	5.84	3.40	4.40	5.32

Table 2
Changes in the number of tours

Solution procedure	MAXREW1			MAXREW2			MAXIMP		
No. of tours (t_1)	5	10	15	5	10	15	5	10	15
Integer solution	1625.60	2644.20	3388.40	1625.60	2644.20	3388.40	1508.80	2484.20	3200.00
Solution time (min)	19.32	43.33	278.48	8.96	33.49	273.05	0.91	0.95	0.99
LP solution	1632.84	2653.82	3402.43	1632.84	2653.82	3402.43			
LP solution time (min)	13.92	13.14	12.50	4.46	4.25	4.61			
No. of B & B nodes	5.00	57.20	789.20	4.80	57.80	804.40			
Tours:									
Max. No. of nodes	5.00	5.40	5.00	5.00	5.40	5.00	5.20	5.20	5.20
Min. No. of nodes	3.60	2.80	2.00	3.60	2.80	2.00	3.20	2.40	2.00
Ave. No. of nodes	4.48	3.82	3.35	4.48	3.82	3.33	4.04	3.48	3.07

Table 3
Changes in the number of nodes

Solution procedure	MAXREW1			MAXREW2			MAXIMP		
No. of nodes (n)	50	75	100	50	75	100	50	75	100
Integer solution	1923.20	2469.60	2644.20	1923.20	2469.60	2644.20	1804.60	2304.80	2484.20
Solution time (min)	5.94	31.25	43.33	1.24	26.54	33.49	0.07	0.34	0.95
LP solution	1930.85	2483.55	2653.82	1930.85	2483.55	2653.82			
LP solution time (min)	0.89	6.04	13.14	0.29	1.89	4.25			
No. of B & B nodes	31.20	136.00	57.20	31.20	139.40	57.80			
Tours:									
Max. No. of nodes	4.20	4.80	5.40	4.20	4.80	5.40	4.80	5.40	5.20
Min. No. of nodes	2.00	2.00	2.80	2.20	2.00	2.80	1.80	2.20	2.40
Ave. No. of nodes	3.04	3.48	3.82	3.04	3.48	3.82	2.72	3.30	3.48

Table 3 presents the effect of increasing the number of nodes in G. We consider five instances of each of 50, 75, and 100 nodes. In all cases, it is assumed that $q = 1$, $t_1 = 10$, and $T_1 = 480$ time units.

Finally, in Table 4, we again consider the same five 100 node instances, however, this time we set $q = 2$, $t_1 = 5$, $t_2 = 5$, $T_1 = 360$ time units, and $T_2 = 480$ time units. (Note that MAXIMP cannot be used to solve these problems.) The results from the 100 node instances, where $q = 1$, $t_1 = 10$, and $T_1 = 480$, are also provided in this table for comparison.

As expected, from Table 1 we find that as the number of nodes in the tours increases, the corresponding solution time increases. This increase in solution time is a result of the larger TSPs which must be solved. From Table 2 it also appears that the solution time increases as the number

Table 4
Changes in the number of maximum tour lengths

Solution procedure	MAXREW1		MAXREW2	
No. of tour lengths (q)	1	2	1	2
Integer solution	2644.20	2325.40	2644.20	2325.40
Solution time (min)	43.33	79.04	33.49	72.15
LP solution	2653.82	2336.53	2653.82	2336.53
LP solution time (min)	13.14	13.15	4.25	7.70
No. of B & B nodes	57.20	40.00	57.80	40.00
Tours:				
Max. No. of nodes	5.40	4.20	5.40	4.20
Min. No. of nodes	2.80	2.20	2.80	2.20
Ave. No. of nodes	3.82	3.30	3.82	3.30

of tours taken through G increases. This is probably due to the increased number of combinations of nodes that can be included in a tour, which is a result of there being more time allocated for collection. Evidence of this can be seen by looking at the number of branch and bound nodes needed to find the optimal solution. Another expected result is that as the number of nodes in the problem increases (i.e., as the size of the problem increases) the solution time increases. This is obviously due to the increased number of node combinations considered, which translates into solving more TSPs. Finally, in Table 4 we find that increasing the number of tour lengths from 1 to 2 (i.e., increasing q) increases the solution time. For this particular set of problems the average solution time is doubled, even though the average number of nodes in a tour is reduced (recall that five of the tours have a shorter time limit). The most likely factor causing this increase is the extra work needed to keep track of two tour lengths. While the number of TSPs solved should be approximately the same, the amount of work in storing tours and searching the TSP storage arrays has doubled.

It can be seen that using the heuristic to solve the TSPs within MAXREW works very well. This can be attributed to the size of the TSPs being solved. In almost all cases, using this heuristic reduces the time to solve the problem without compromising the final solution.

The heuristic, MAXIMP, also performed fairly well on these test problems. On average, the sum of rewards collected were approximately 5.5–8.0% less than the optimal collections using MAXREW. If time is the most important criterion, MAXIMP may still have its place in some applications. However, a heuristic alternative is to crash MAXREW once a feasible integer solution has been found. We have found that with MAXREW, the optimal solution (or at least a nearly optimal solution) is often found very early in the branch and bound tree. A second alternative heuristic procedure is to change the TSP heuristic used in MAXREW2 to one which is less expensive. For example, the Farthest-Insertion heuristic could be run with fewer starting nodes.

In conclusion, MAXREW works extremely well as long as the number of nodes in the tours remains relatively small. Please note that with respect to most of the applications discussed in the introduction, the number of nodes one could possibly visit in any one tour is indeed small.

5. Summary

In this paper we considered the Multiple Tour Maximum Collection Problem (MTMCP). The MTMCP seeks to collect the maximum reward from a set of nodes in a graph by traveling to an optimal subset of these nodes on at most m nonoverlapping time constrained tours, where each of the m tours begins and ends at a central depot.

An optimal solution procedure for the MTMCP was described. This procedure is based on a generalized set-partitioning formulation and uses constraint branching and tour storage techniques to improve solution time. It was shown that the solution procedure works well on realistic size problems, particularly when the number of nodes visited in any tour is relatively small. It was also shown that by incorporating a Farthest-Insertion heuristic into this procedure to solve the TSPs, the solution time of the algorithm can be reduced with little degradation to the solution. Finally, it was mentioned that a good heuristic alternative procedure is to crash the optimal procedure prematurely and use the best integer solution, since a near optimal integer solution is often found very early in the process.

Future work under consideration is related to the development of better branching schemes specific to the given problem structure. We are also considering ways of extending the current methodology to cope with problems where it is possible to visit a node on more than one tour.

References

[1] Butt SE, Cavalier TM. A heuristic for the multiple tour maximum collection problem. Computers and Operations Research 1994;21(1):101–11.
[2] Wu T. Optimization models for underway replenishment of a dispersed carrier battle group. Masters Thesis, Naval Postgraduate School, Monterey, California, USA, March 1992.
[3] Dunn JS. Scheduling underway replenishment as a generalized orienteering problem. Masters Thesis, Naval Postgraduate School, Monterey, California, USA, June 1992.
[4] Golden B, Assad A, Dahl R. Analysis of a large scale vehicle routing problem with an inventory component. Large Scale Systems 1984;7:181–90.
[5] Davis S, Ceto N. Jr, Rabb JM. A comprehensive check processing simulation model. Journal of Bank Research 1982;13:185–94.
[6] Golden B, Levy L, Vohra R. The orienteering problem. Naval Research Logistics 1987;34:307–18.
[7] Gensch DH. An industrial application of the traveling salesman's subtour problem. AIIE Transactions 1978;10(4): 362–70.
[8] Golden B, Levy L, Dahl R. Two generalizations of the traveling salesman problem. OMEGA 1981;9(4):439–41.
[9] Golden B, Wang Q, Liu L. A multifaceted heuristic for the orienteering problem. Naval Research Logistics 1988;35: 359–66.
[10] Tsiligirides T. Heuristic methods applied to orienteering. Journal of the Operational Research Society 1984;35(9): 797–809.
[11] Ramesh R, Brown K. An efficient four-phase heuristic for the generalized orienteering problem. Computers and Operations Research 1991;18(2):151–65.
[12] Kataoka S, Morito S. An algorithm for the single constraint maximum collection problem. Journal of the Operations Research Society of Japan 1988;31(4):515–31.
[13] Ramesh R, Yoon Y, Karwan M. An optimal algorithm for the orienteering tour problem. ORSA Journal on Computing 1992;4(2):155–65.
[14] Keller C, Goodchild M. The multiobjective vending problem: a generalization of the travelling salesman problem. Environment and Planning B: Planning and Design 1988;15:447–60.

[15] Balas E. The prize collecting traveling salesman problem. Networks 1989;19:621–36.
[16] Brideau RJ, III, Cavalier TM. The maximum collection problem with time dependent rewards. Working Paper, Department of Industrial and Manufacturing Engineering, The Pennsylvania State University, University Park, Pennsylvania, USA, 1994.
[17] Erkut E, Zhang J. The maximum collection problem with time dependent rewards. Working Paper, Department of Finance and Management Science, University of Alberta, Canada, February 1995.
[18] Ryan DM. ZIP–a zero-one integer programming package for scheduling. Research Report #CSS85, AERE, Harwell, 1980.
[19] Carpaneto G, Dell'amico M, Toth P. Exact solution of large-scale, asymmetric traveling salesman problems. ACM Transactions on Mathematical Software 1995;21(4);394–409.
[20] Ryan DM, Falkner JC. On the integer properties of scheduling set partitioning models. European Journal of Operational Research 1988;35:442–56.
[21] Schneider GM, Bruell SC. Concepts in data structures and software development: a text for the second course in computer science. St. Paul, MN: West Publishing Company, 1991.