Exact Solution of Large-Scale, Asymmetric Traveling Salesman Problems

G. CARPANETO
Università di Modena

M. DELL'AMICO
Politecnico di Milano
and
P. TOTH
Università di Bologna

A lowest-first, branch-and-bound algorithm for the *Asymmetric Traveling Salesman Problem* is presented. The method is based on the *Assignment Problem relaxation* and on a *subtour elimination branching scheme*. The effectiveness of the algorithm derives from reduction procedures and parametric solution of the relaxed problems associated with the nodes of the branch-decision tree. Large-size, uniformly, randomly generated instances of complete digraphs with up to 2000 vertices are solved on a DECstation 5000/240 computer in less than 3 minutes of CPU time. In addition, we solved on a PC 486/33 *no-wait flow shop* problems with up to 1000 jobs in less than 11 minutes and real-world *stacker crane* problems with up to 443 movements in less than 6 seconds.

Categories and Subject Descriptors: G.2.1 [**Discrete Mathematics**]: Combinatorics—*combinatorial algorithms*; G 2 2 [**Discrete Mathematics**]: Graph Theory—*graph algorithms*; *path and circuit problems*

General Terms: Algorithms

Additional Key Words and Phrases: Assignment problem, asymmetric traveling salesman problem, branch and bound, subtour elimination, reduction procedure

1. INTRODUCTION

Consider a complete *digraph* $G = (V, A)$ with vertex set $V = \{1, \dots, n\}$, arc set $A = \{(i,j) : i \in V, j \in V\}$, and a cost $a_{i,j}$ associated with each arc $(i,j) \in$

The work reported in this article was supported by M.U.R.S.T. (Italy) and by the Human Capital and Mobility Project CHRX-CT93-0087 of the European Community.
Authors' addresses: G. Carpaneto, Dipartimento di Economia Politica, Università di Modena, Italy; M. Dell'Amico, Dipartimento di Elettronica e Informazione, Politecnico di Milano, Italy; P. Toth, D.E.I.S., Università di Bologna, Italy.

A ($a_{i,i} = \infty \ \forall i \in V$). We define a new graph $G' = (V', A')$ with vertex set $V' = \{v_1, \ldots, v_p\} \subseteq V$ and arc set $A' = \{(v_1, v_2), (v_2, v_3), \ldots, (v_p, v_1)\} \subseteq A$ as a *tour* (or *Hamiltonian circuit*) if $p = n$ and a *subtour* if $p < n$. The cost of a tour is given by the sum of the costs of its arcs. The *Asymmetric Traveling Salesman Problem* (ATSP) is to find a tour with minimum cost z^*. The problem is known to be *NP-hard* and has many important applications (scheduling, distribution, wiring, FMS, ...).

ATSP can be mathematically formulated as:

$$z^* = min \sum_{i=1}^{n} \sum_{j=1}^{n} a_{i,j} x_{i,j} \tag{1}$$

subject to

$$\sum_{i=1}^{n} x_{i,j} = 1, \qquad j = 1, \ldots, n \tag{2}$$

$$\sum_{j=1}^{n} x_{i,j} = 1, \qquad i = 1, \ldots, n \tag{3}$$

$$\sum_{i \in S} \sum_{j \in S} x_{i,j} \le |S| - 1, \qquad \forall S \subset V, S \ne \emptyset \tag{4}$$

$$x_{i,j} \in \{0,1\}, \qquad i,j = 1, \ldots, n \tag{5}$$

where $x_{i,j} = 1$, if arc (i, j) belongs to the optimal tour; $x_{i,j} = 0$ otherwise. Without loss of generality, we will assume that costs are nonnegative integers. Equations (1), (2), (3), and (5) define the well-known *Assignment Problem* (AP). Constraints (4) exclude subtours (loop included).

Many algorithms have been developed for the exact solution of ATSP. The most-effective ones are the branch-and-bound methods proposed by Smith et al. [1977], Carpaneto and Toth [1980], Balas and Christofides [1981], and Pekny et al. [1991]; a survey of enumerative algorithms for the TSP is given in Balas and Toth [1985]. A parallel algorithm has recently been proposed by Miller and Pekny [1989] and Pekny et al. [1991]. As for sequential algorithms the maximum size of *uniformly, randomly generated* for which many instances have been solved is 5000, although single random instances with as many as 500,000 vertices (but small cost ranges) have been solved by Miller and Pekny [1991]. For the undirected graph case (Symmetric Traveling Salesman Problem), a *Euclidean* instance with 2392 vertices has been solved through a sequential branch-and-cut procedure, using facet-inducing linear inequalities, in more than 27 hours on a CYBER 205 (see Padberg and Rinaldi [1991]). A similar approach was recently presented by Applegate et al. at the SIAM Conference, 1993. They solved instances with 3038 and 4461 vertices.

We present a sequential, lowest-first, branch-and-bound algorithm based on the *AP relaxation* and a *subtour elimination branching scheme*. The Fortran implementation of the algorithm is given in Carpaneto et al. [1995]. The effectiveness of the algorithm derives from reduction procedures and parametric solution of the relaxed problems associated with the nodes of the branch-decision tree. Large-size, uniformly, randomly generated instances of complete digraphs with up to 2000 vertices are solved on a DECstation 5000/240 computer (with 16MB of main memory) in less than 3 minutes of CPU time. In addition, we solved on a PC 486/33 (with 8MB of main memory) *no-wait flow shop* problems (see Papadimitriou and Kanellakis [1980]) with up to 1000 jobs in less than 11 minutes and real-world *stacker crane* problems with up to 443 movements in less than 6 seconds. According to our experience the DECstation 5000/240 is about two times faster than the PC 486/33.

Finally, we note that the proposed approach is not useful for instances where the asymmetric nature of the problem disappears (Symmetric and Quasi Symmetric TSP). In particular many small ($n \leq 100$) instances in TSPLIB are of this kind, so our code cannot solve them.

A preliminary version of this article has been presented at the 13th AMASES Congress [Carpaneto et al. 1989].

2. ALGORITHM

The algorithm is derived from the lowest-first branch-and-bound procedure TSP1 presented in Carpaneto and Toth [1980]. At each node h of the decision tree TSP1 solves a *Modified Assignment Problem* (MAP_h) defined by Eqs. (1), (2), (3), (5), and the additional constraints associated with arc subsets E_h and I_h, where:

$$E_h = \{(i,j) \in A : x_{i,j} \text{ is fixed to } 0\} \quad (\textit{excluded arcs});$$
$$I_h = \{(i,j) \in A : x_{i,j} \text{ is fixed to } 1\} \quad (\textit{included arcs}).$$

If the optimal solution to MAP_h does not define a Hamiltonian circuit, and its value LB_n (giving the lower bound associated with node h) is less than the current optimal solution value, say UB, then m descending nodes are generated from node h according to the following branching scheme (which is a modification of the subtour elimination rule proposed by Bellmore and Malone [1971]).

Let $G_1, \ldots, G_d$ be the subtours defined by the optimal solution to MAP_h, where, for $q = 1, \ldots, d$, $G_q = (V_q, A_q)$ with $V_q = \{r_{q,1}, \ldots, r_{q,e_q}\}$, $A_q = \{(r_{q,1}, r_{q,2}), (r_{q,2}, r_{q,3}), \ldots, (r_{q,e_q}, r_{q,1})\}$, and e_q = number of vertices (and arcs) of the qth subtour.

The subtour, say G_p, having the minimum number of not-included arcs, i.e., the subtour such that

$$m = e_p - |A_p \cap I_h| = min_{q=1,\ldots,d}\{e_q - |A_q \cap I_h|\},$$

is chosen for branching.

Let $\bar{A} = \{(s_1, t_1), \ldots, (s_m, t_m)\} = A_p \setminus I_h$ be the subset of not-included arcs of A_p (the order of the arcs in $\bar{A}$ is the same as that of the corresponding arcs in A_p). The subset of the excluded and included arcs associated with the jth descending node, say $g(j)$, of node h is $(j = 1,\dots,m)$:

$$E_{g(j)} = E_h \cup \{(s_j, t_j)\};$$
$$I_{g(j)} = I_h \cup \{(s_i, t_i) : i = 1,\dots, j-1\}.$$

Each subset $E_{g(j)}$, with $j > 1$, is enlarged by adding arc (t_{j-1}, s_1) so as to avoid subtours corresponding to paths containing included arcs.

The new approach differs from that presented by Carpaneto and Toth [1980] mainly in the following respects:

(a) application at the root node of the branch-decision tree of a *reduction procedure* so as to remove from G the arcs which cannot belong to an optimal tour. In this way the original digraph G can be transformed into a sparse one, say $\tilde{G} = (V, \tilde{A})$, allowing the use of sparse cost-matrix procedures for the solution of the *MAP*s associated with the nodes of the branch-decision tree;

(b) the utilization of an efficient parametric technique for the solution of the *MAP*s, allowing each MAP_h to be solved in $O(|\tilde{A}| \log n)$ time;

(c) the introduction of an effective data structure to store the information associated with the nodes of the decision tree;

(d) the application at each node h of a *connecting procedure* to decrease the number of subtours defined by the optimal solution to MAP_h.

2.1 Reduction Procedure

At the root node, say node 0, of the branch-decision tree, the AP corresponding to the original complete cost-matrix, $(a_{i,j})$, is solved through the $O(n^3)$ primal-dual procedure CTCS presented in Carpaneto and Toth [1987]. Let (u_i) and (v_j) be the optimal solution of the dual problem associated with AP, i.e., the *dual variables* of AP, and LB_0 the corresponding solution value. It is well known that for each arc $(i, j) \in A$ the *reduced cost* $\bar{a}_{i,j} = a_{i,j} - u_i - v_j \geq 0$ represents a lower bound on the increase of the optimal solution value of AP corresponding to the inclusion of arc (i, j) in the solution of AP, hence in that of ATSP. If a feasible ATSP solution of value UB is known, then each arc $(i, j) \in A$ such that

$$\bar{a}_{i,j} \geq UB - LB_0$$

can be removed from arc set A, since its inclusion in any solution of ATSP cannot lead to a solution value less than UB. The original complete digraph G can thus be transformed into the equivalent sparse one, $\tilde{G} = (V, \tilde{A})$, where

$$\tilde{A} = \{(i, j) \in A : \bar{a}_{i,j} < UB - LB_0\}.$$

The feasible solution of value UB can be obtained through any heuristic procedure for ATSP. In our implementation we used the patching algorithm proposed by Karp [1979].

An alternative way to compute UB is to introduce an "artificial" upper bound $[\alpha LB_0]$ (with $\alpha > 1$) and to set

$$UB = [\alpha LB_0] + 1. \tag{6}$$

If, at the end of the branch-and-bound algorithm, no feasible solution of value less than UB is found, this means that $[\alpha LB_0]$ is not a valid upper bound; so α must be increased, and a new run, starting with the reduction procedure, must be performed.

2.2 Parametric Solution of *MAP*s

Since at each node of the decision tree a MAP is solved, the effectiveness of the ATSP algorithm depends greatly on the efficiency of the algorithm used to solve the *MAP*s. At each node h of the decision tree, instead of solving MAP_h from scratch, a parametric technique is adopted which finds only one *shortest augmenting path*. In fact, to generate a descending node h from its parent node k, we exclude only one arc, say (s, t), from the solution of MAP_k (with $(s, t) = E_h \setminus E_k$). So, to obtain the optimal solution of MAP_h from that of MAP_k, it is only necessary to satisfy constraint (2) for $j = t$ and constraint (3) for $i = s$, i.e., to find a new, shortest augmenting path from vertex s to vertex t in the bipartite graph corresponding to MAP_h by considering the current reduced cost-matrix $(\bar{a}_{i,j})$. Addition of the new included arcs (contained in subset $I_h \setminus I_k$) does not affect the assignment, they being in the optimal solution of MAP_k (the details of the technique used to impose the new constraints (arcs exclusion or inclusion) are discussed in the next subsection). As graph $\tilde{G}$ is sparse, the shortest augmenting path is found through a procedure derived from the labeling algorithm proposed by Johnson [1977] for the computation of shortest paths in sparse graphs, which utilizes a heap queue. Hence, the resulting time complexity for solving each MAP is $O(|\tilde{A}| \log n)$.

The computation of the shortest augmenting path at node h is stopped as soon as its current reduced cost (i.e., the value of the label of the next vertex to be included in the shortest path) is greater than or equal to the gap between the value UB of the best solution so far and the value of the MAP associated with the parent node of h.

2.3 The Decision Tree

There are two kinds of nodes in the decision tree: *active* nodes (i.e., nodes not yet branched) and *passive* nodes (i.e., nodes branched or fathomed). The active nodes are ordered according to nondecreasing values of the corresponding lower bounds; in case of a tie the ordering is based on the following rule: first the node with the maximum number of included arcs and, in case of a new tie, first the node with the maximum number of excluded arcs. To store the information associated with the nodes of the decision tree, a vector V and two matrices MF and MV are used; vector V contains the *scalar informa-*

tion, the matrices the *vectorial information*. For each node h the following scalar information is stored:

(a) the pointer to the active node preceding h in the ordered list;
(b) the pointer to the active node following h in the ordered list;
(c) the pointer to the parent node of h;
(d) the lower bound LB_h associated with h;
(e) the generation number of h between the nodes descending from the parent node k;
(f) the number m of not-included arcs of the subtour chosen for branching at node h;
(g) the pointer to the column of matrices MF and MV containing the vectorial information of node h;
(h) the m not-included arcs of the chosen subtour.

The vectorial information stored for each active node h is the vector (f_i), with $f_i = j$ if row i is assigned to column j, corresponding to the primal solution of MAP_h (in matrix MF) and the vector of the dual variables (v_j) associated with MAP_h (in matrix MV). The vectorial information of node h is used for the parametric solution of the MAPs corresponding to the nodes descending from h. (Note that the dual variables (u_i) associated with MAP_h are not stored, since they can easily be computed through the above information.)

Problem MAP_h corresponding to node h of the decision tree is defined through subsets E_h and I_h. The constraints associated with E_h and I_h are implicitly imposed by updating, with respect to the parent node k, cost-matrix $(a_{i,j})$, and dual variables (v_j) as follows:

(1) replace $a_{i,j}$ with $a_{i,j} + \lambda$ for each arc $(i,j) \in E_h \setminus E_k$,
(2) replace v_j with $v_j - \lambda$ for each vertex $j \in V_h \setminus V_k$,

where λ is a sufficiently large positive value and $V_p = \{j \in V :$ there exists an arc $(i,j) \in I_p\}$.

The first updating avoids the choice of any arc $(i,j) \in E_h$ in the optimal solution to MAP_h. The second updating prevents, in the shortest-augmenting-path computation performed at node h, the labeling of any column j associated with a vertex $j \in V_h$; in this way the assignment of column j in the optimal solution to MAP_h is not changed with respect to that corresponding to node k.

Note that at the end of the computation of the optimal solution to MAP_h, dual variables v_j, with $j \in V_h$, are not changed, while the remaining dual variables are generally updated.

In order to save main memory only one copy of the cost-matrix (that corresponding to the last node considered) is used, and, for each node h, subsets E_h and I_h are not explicitly stored. Hence the problem of implicitly updating the subsets of the excluded and included arcs corresponding to the nodes arises. Let r be the last node considered and k the next node to be explored. The current cost-matrix $(a_{i,j})$ (corresponding to node r) is given by the original elements with $a_{i,j}$ replaced by $a_{i,j} + \lambda$ for each arc $(i, j) \in E_r$. In order to obtain the cost-matrix associated with node k we find the lowest common ancestor, say q, of nodes r and k; then we remove, level by level, all constraints corresponding to arcs in $E_r \setminus E_q$ and impose, level by level, all constraints corresponding to arcs in $E_k \setminus E_q$. The current dual variables (v_j) associated with node k (which implicitly define the set of included arcs I_h) are directly obtained from the column of matrix MV corresponding to node k.

2.4 Connecting Procedure

Consider a node h of the decision tree for which several optimal solutions to MAP_h exist. In this case the optimal solution which generally leads to the smallest number of nodes in the subtree descending from h is that having the minimum number of subtours. A heuristic procedure which tries to decrease the number of subtours defined by the current optimal solution to MAP_h is obtained by iteratively applying the following rule.

Rule 2.4.1. Given two subtours $G_a = (V_a, A_a)$ and $G_b = (V_b, A_b)$, if there exists an arc pair $(i_a, j_a) \in A_a$ and $(i_b, j_b) \in A_b$ such that arcs (i_a, j_b) and (i_b, j_a) have zero-reduced costs (i.e., $\bar{a}_{i_a, j_b} = \bar{a}_{i_b, j_a} = 0$), then an equivalent optimal solution to MAP_h can be obtained by connecting subtours G_a and G_b to form a unique subtour $G_a = (V_a \cup V_b, A_a \cup A_b \setminus ((i_a, j_a) \cup (i_b, j_b)) \cup ((i_a, j_b) \cup (i_b, j_a)))$.

If at the end of the connecting procedure a Hamiltonian circuit is found, it corresponds to the optimal solution to the ATSP associated with node h, and no descending nodes are generated.

The connecting procedure is always applied at the root node of the decision tree. For the other nodes it is applied only if the total number of zero-reduced-cost arcs at the root node is greater than a given threshold β. Indeed, the procedure is effective only if the reduced graph contains a sufficiently large number of zero-cost arcs. Computational experiments have shown that an adaptive strategy, which counts the number of zero-cost arcs at each node and then decides on the opportunity to apply the procedure, gives worse results than the simple threshold method. In the computational analysis presented in Section 3, we set $\beta = 2.5n$.

2.5 Comparison with the Algorithms of Miller and Pekny

The most-effective procedures for the solution of the ATSP are those proposed by Miller and Pekny [Miller and Pekny 1989; 1991; Pekny and Miller 1992; Pekny et al. 1991]. In the same period we independently developed the code described in this article. All these procedures are based on the general approach presented in Carpaneto and Toth [1980]. Here we discuss the main differences and similitudes between these approaches. Miller and Pekny [1989] presented a preliminary algorithm which is a parallelization of the approach of Carpaneto and Toth, improved with the application of the patching heuristic [Karp 1979] at the root node. Randomly generated instances with up to 3000 vertices were solved on a Butterfly Plus computer with 14 processors in 1263.9 seconds. The entries of the cost-matrix were uniformly generated in the range $[0, 10^3]$. The algorithm presented by Pekny and Miller [1992] represents a substantial improvement of the original parallel procedure. The MAPs at the nodes are solved through an $O(n^2)$ procedure which computes a single augmenting path. This procedure was implemented using a *d-heap*. Moreover the patching algorithm was applied at the root node and to the other nodes "with decreasing frequency as search progresses." In addition the branch-and-bound phase was preceded by a sparsification of the cost-matrix obtained by removing all the entries with cost greater than a given threshold λ. A sufficient condition is given to check if the optimal solution obtained from the sparse matrix is optimal for the original matrix. Random instances with up to 10,000 vertices and with costs uniformly, randomly generated in $[0, n]$ were solved on a Butterfly Plus multiprocessor in less than 1300 seconds (on average). The algorithm presented by Pekny et al. [1991] is a modification of that presented by Pekny and Miller [1992], obtained with the application, at each node, of an exact procedure to find a Hamiltonian circuit on the subgraph defined by the arcs with zero-reduced cost. Instances with 3000 vertices and costs randomly, uniformly generated in $[0, 10^3]$ were solved in 102.38 seconds on a SUN 4/280 while for the instances with costs generated in $[0, 10^4]$ the average running time was of 1434.82 seconds. The initial cost-matrix sparsification was not applied for these computations. The most-sophisticated version of the Miller and Pekny code appears to be that presented by Miller and Pekny [1991], which includes all the improvements previously proposed by the authors. Many instances with 5000 vertices and costs uniformly, randomly generated in $[0, n]$ were solved on a SUN 4/330 in 38.1 seconds (this time does not include the construction of the sparse matrix). One instance with 500,000 vertices and costs randomly, uniformly generated in $[0, n]$ was solved on a CRAY 2 in 12,623 seconds.

The similarities among our approach and the algorithms of Miller and Pekny are the following: (a) the branching rule is that proposed in Carpaneto and Toth [1980], (b) the MAPs at the nodes are solved through an $O(n^2)$ procedure, (c) the patching algorithm is applied at the root node. The two approaches differ in the following aspects: (a) for the sparsification phase we propose a criterion based on the comparison between the reduced costs given by the initial linear assignment procedure and the gap between lower and upper bound (see Section 2.1). (If a true upper bound is used, we only eliminate arcs which cannot belong to the optimal solution; therefore a single run of the algorithm is required. On the contrary, using an artificial upper bound or the technique described by Miller and Pekny it can be necessary to run the algorithm more than one time.) (b) we propose an efficient technique to store and retrieve the subproblems so that the exploration of the branch-decision-tree is accelerated; (c) we apply a fast heuristic algorithm to find a Hamiltonian circuit on the subgraph defined by the arcs with zero-reduced cost.

Comparing the computational results obtained by Miller and Pekny with those presented in Section 3 of this article it appears that our code is slower than the algorithm presented by Miller and Pekny [1991], for small cost ranges (and random instances), but it seems to be faster for large cost ranges. Using our code, D. S. Johnson solved random instances with 4000 vertices and costs in $[0, 10^6]$, in only 14 minutes on an SGI Challenge (239 subproblems were solved).

3. COMPUTATIONAL RESULTS

The algorithm has been coded as a Fortran subroutine called CDT [Carpaneto et al. 1995]. Subroutine CDT has been tested on randomly generated test problems with up to 2000 vertices. We considered both instances with random costs and instances derived from real-like scheduling problems. In particular, we solved no-wait flow shop problems which can be stated as follows: n *jobs* and a set $\{1, 2, \dots, m\}$ of m *machines* are given. Each job must be scheduled on machines $1, 2, \dots, m$ in such a way that: (a) no machine processes two jobs at the same time; (b) the processing of a job on machine j starts exactly when the processing of the same job on machine $j - 1$ is completed. Let $p_{i,j}$ be the processing time of job i on machine j: the problem consists in finding a sequence of the n jobs which minimizes the completion time of the last job processed on machine m.

Papadimitriou and Kanellakis [1980] have shown that an instance of the no-wait flow shop problem can be transformed into an equivalent instance of ATSP with $n + 1$ vertices.

Eight classes of test problems were considered by generating the coefficients of the integer cost-matrix $(a_{i,j})$ as follows:

(a1) $a_{i,j}$ uniformly random in $[1, 10^3]$;
(a2) $a_{i,j}$ uniformly random in $[1, 10^4]$;
(a3) $a_{i,j}$ uniformly random in $[1, 10^6]$;
(t1) $a_{i,j}$ uniformly, randomly generated in $[1, 10^3]$ and then triangularized;
(t2) $a_{i,j}$ uniformly, randomly generated in $[1, 10^4]$ and then triangularized;
(t3) $a_{i,j}$ uniformly, randomly generated in $[1, 10^6]$ and then triangularized;
(f1) no-wait flow shop problems with 10 machines and $p_{i,j}$ uniformly random in $[1, 100]$;
(f2) no-wait flow shop problems with 20 machines and $p_{i,j}$ uniformly random in $[1, 100]$.

For each value of n and each class of problems, 50 different instances have been solved. Tables I to XI give the following information (the times are expressed in seconds):

—average, median, and maximum running times for CDT;
—average running time at the root node;
—average number of MAPs completely solved;
—average and (in brackets) maximum number of explored nodes (i.e., nodes which generated son nodes);

—average and (in brackets) maximum level of the decision tree at which the optimal solution was found;
—average number of son nodes generated by an explored node;
—average density of the sparse cost matrix (i.e., $|\tilde{A}|/n^2$);
—average (AP solution value at the root node)/z^* ratio.

Tables I to VI give the results obtained on a PC 486/33 for values of n from 100 to 10^3 for problems of classes $a1$, $a2$, and $a3$ and from 100 to 500 for problems of classes $t1$, $t2$, $t3$ (larger values of n for problems of classes $t1$, $t2$, and $t3$ have not been considered because the excessive computing time required for the triangularization of the cost-matrices). The value of upper bound UB has been obtained using the patching algorithm proposed by Karp [1979].

Tables I–III (uniform problems) show that the ratio between the lower bound at the root node (LB_0) and the optimal solution value (z^*) is always very close to 1 and increases with the value of n. The performances of the algorithm do not change very much when the cost ranges increase from $(1, 10^3)$ to $(1, 10^6)$; however, one can observe a tendency to an increment of the difficulty of the instances with the increment of the cost ranges. This is mainly due to the larger absolute gap between z^* and LB_0, which leads to a greater number of nodes in the decision tree.

Tables IV–VI (triangular problems) show that the running time required for solution of the MAPs is much greater than that corresponding to uniform problems. In fact, procedure CTCS [Carpaneto and Toth 1987], which is used for the solution of the AP at the root node, performs worse for these instances, and the computation of the shortest augmenting paths at the nodes of the decision tree is slower because of the higher density of the sparse cost-matrix. However, the average running time of CDT is less than that in Tables I–III because of the much smaller number of nodes generated by the branch-and-bound algorithm.

To consider large-size problems ($n > 10^3$) we ran subroutine CDT on a DECstation 5000/240 computer. Tables VII–IX give the results for problems of classes $a1$, $a2$, and $a3$, values of n from 500 to 2000. The algorithm has a behavior similar to that shown in Tables I–III.

Tables X and XI give the results for the problems of classes $f1$ and $f2$. The value of UB used by the reduction procedure was artificially obtained through (6) with $\alpha = 1.005$. For only three instances with 20 machines and less than 300 jobs it was necessary to increase the value of α to 1.01.

Finally, we considered some real-world stacker crane problems with up to 443 *movements*. The stacker crane problem arises in the reorganization of an inventory system which consists of a series of shelves where products are positioned and of an automatic crane which moves the products from the operator position (I/O area) to the shelves and vice versa. During the night, the crane reorganizes the system by moving products from a shelf to another. The shelves are positioned in a vertical rack and are identified by two coordinates. In order to perform the reorganization of the system, two problems have to be solved: (a) identify the movements of products from a shelf to

Table I. Class a1, PC 486/33 (in seconds)

	Running Time			Root Node Time	Completely Solved MAPs	Explored Nodes		Optimal Solution Level		Son Nodes	Sparse Matrix Density	LB_0 / z^*
n	Avg	Median	Max			Avg	(Max)	Avg	(Max)			
100	0.42	0.39	0.71	0.15	30.2	11.9	(31)	4.7	(7)	10.9	0.155	0.9866
200	1.63	1.16	3.96	0.52	32.0	11.9	(28)	4.9	(10)	13.8	0.167	0.9970
300	2.08	1.87	3.90	1.13	27.8	5.0	(13)	4.5	(7)	26.2	0.078	0.9997
400	3.48	3.30	7.25	2.42	22.1	3.8	(13)	4.4	(8)	17.3	0.102	0.9998
500	9.40	5.27	17.30	3.18	60.8	15.0	(41)	4.6	(7)	16.0	0.098	0.9983
600	19.13	11.27	45.16	4.66	82.9	20.0	(51)	6.3	(8)	35.5	0.128	0.9975
700	21.44	13.79	50.05	6.28	70.6	13.2	(44)	5.8	(9)	31.5	0.100	0.9987
800	23.04	19.39	42.47	7.99	66.9	14.4	(32)	5.4	(8)	55.0	0.064	0.9994
900	19.66	15.16	50.11	10.85	52.4	6.7	(21)	6.3	(12)	42.5	0.052	0.9994
1000	39.10	26.87	121.20	12.86	55.4	9.7	(36)	6.3	(10)	30.7	0.105	0.9995

Table II. Class a2, PC 486/33 (in seconds)

	Running Time			Root Node Time	Completely Solved MAPs	Explored Nodes		Optimal Solution Level		Son Nodes	Sparse Matrix Density	LB_0 / z^*
n	Avg	Median	Max			Avg	(Max)	Avg	(Max)			
100	0.29	0.27	0.55	0.13	23.1	6.4	(20)	3.7	(5)	8.9	0.108	0.9946
200	1.12	0.88	2.58	0.51	27.8	7.0	(23)	4.4	(6)	10.4	0.145	0.9977
300	2.19	1.32	4.72	1.11	23.3	5.8	(21)	4.5	(9)	16.2	0.098	1.0000
400	4.60	3.63	14.56	2.42	29.0	7.6	(46)	4.6	(8)	7.7	0.071	1.0000
500	7.43	7.14	14.17	3.12	42.1	9.1	(26)	4.8	(10	25.7	0.089	0.9987
600	15.43	13.79	34.34	4.85	82.4	16.5	(66)	6.4	(10)	31.8	0.120	0.9866
700	13.63	11.54	22.48	6.59	43.2	8.4	(30)	5.0	(6)	44.3	0.070	0.9993
800	27.01	21.92	73.46	8.28	91.1	25.2	(79)	5.8	(9)	48.6	0.077	0.9984
900	31.36	24.90	43.96	11.45	52.5	17.6	(29)	7.2	(11)	40.1	0.080	0.9971
1000	44.80	34.67	95.19	12.80	92.5	20.6	(87)	7.2	(11)	42.7	0.080	0.9991

Table III. Class a3, PC/486/33 (in seconds)

	Running Time			Root Node	Completely Solved	Explored Nodes		Optimal Solution Level		Son	Sparse Matrix	
n	Avg	Median	Max	Time	MAPs	Avg	(Max)	Avg	(Max)	Nodes	Density	LB_0 / z^*
100	0.38	0.33	0.71	0.13	27.9	8.1	(15)	4.7	(10)	10.4	0.211	0.9891
200	1.25	1.04	2.20	0.52	39.1	6.3	(18)	4.5	(7)	23.4	0.090	0.9989
300	2.95	1.87	7.69	1.20	35.6	7.0	(27)	4.6	(8)	18.5	0.115	0.9982
400	4.55	3.30	8.35	2.42	32.5	6.1	(19)	5.2	(9)	18.2	0.119	1.0000
500	10.68	8.52	20.61	3.49	55.9	11.6	(34)	5.3	(8)	28.0	0.105	0.9996
600	14.68	13.96	31.76	6.88	52.8	19.2	(40)	4.8	(8)	38.8	0.053	0.9978
700	15.09	14.01	23.74	5.23	68.0	27.0	(73)	5.0	(8)	36.1	0.076	0.9967
800	32.91	26.48	63.79	11.06	137.3	33.1	(99)	5.9	(8)	33.0	0.060	0.9990
900	42.19	41.59	75.27	12.15	87.0	28.4	(92)	4.7	(7)	46.8	0.037	0.9946
1000	47.56	43.08	88.24	14.52	89.3	11.4	(48)	7.3	(10)	39.3	0.061	0.9991

Table IV. Class t1, PC 486/33 (in seconds)

	Running Time			Root Node	Completely Solved	Explored Nodes		Optimal Solution Level		Son	Sparse Matrix	
n	Avg	Median	Max	Time	MAPs	Avg	(Max)	Avg	(Max)	Nodes	Density	LB_0 / z^*
100	0.38	0.27	1.05	0.17	6.9	4.3	(11)	3.10	(6)	3.6	0.460	0.9934
200	0.82	0.60	1.59	0.52	2.9	1.5	(4)	2.10	(4)	2.3	0.131	0.9989
300	1.90	1.87	2.30	1.30	2.5	1.2	(2)	2.40	(3)	1.8	0.123	0.9993
400	3.54	3.13	5.27	2.34	2.2	1.3	(3)	2.00	(3)	2.2	0.118	0.9966
500	8.00	6.43	11.98	5.22	5.4	2.4	(7)	3.30	(5)	3.4	0.395	0.9993

Table V. Class t2, PC 486/33 (in seconds)

	Running Time			Root Node	Completely Solved	Explored Nodes		Optimal Solution Level		Son	Sparse Matrix	
n	Avg	Median	Max	Time	MAPs	Avg	(Max)	Avg	(Max)	Nodes	Density	LB_0 / z^*
100	0.34	0.22	0.88	0.17	6.8	3.2	(8)	3.10	(5)	4.0	0.436	0.9930
200	0.90	0.82	1.48	0.66	2.8	1.5	(4)	2.20	(3)	2.6	0.134	0.9987
300	2.62	1.92	7.20	1.34	3.3	1.8	(5)	2.00	(5)	1.9	0.130	0.9992
400	4.61	4.07	7.25	2.58	2.6	1.6	(3)	2.00	(3)	2.1	0.242	0.9986
500	8.19	6.70	18.63	5.74	3.5	1.8	(7)	2.40	(5)	2.9	0.224	0.9994

Table VI. Class t3, PC 486/33 (in seconds)

	Running Time			Root Node	Completely Solved	Explored Nodes		Optimal Solution Level		Son	Sparse Matrix	
n	Avg	Median	Max	Time	MAPs	Avg	(Max)	Avg	(Max)	Nodes	Density	LB_0 / z^*
100	0.38	0.27	1.10	0.18	7.6	3.8	(14)	3.10	(5)	3.8	0.395	0.9930
200	0.90	0.88	1.32	0.83	2.8	1.5	(4)	2.20	(3)	2.7	0.114	0.9987
300	2.79	2.14	6.87	2.21	4.3	1.9	(6)	2.00	(4)	1.7	0.128	0.9992
400	5.03	4.51	7.91	4.35	2.7	1.6	(3)	2.00	(3)	2.2	0.237	0.9986
500	10.77	8.46	19.18	8.02	5.9	2.8	(5)	2.70	(4)	4.3	0.318	0.9990

Table VII Class a1, DECstation 5000/240 (in seconds)

	Running Time			Root Node	Completely Solved	Explored Nodes		Optimal Solution Level		Son	Sparse Matrix	
n	Avg	Median	Max	Time	MAPs	Avg	(Max)	Avg	(Max)	Nodes	Density	LB_0 / z^*
500	2.69	1.97	8.28	1.6	36.2	13.8	(64)	4.7	(9)	23.0	0.0635	0.9989
1000	14.14	8.72	29.69	6.1	69.3	13.3	(57)	6.3	(9)	59.0	0.0501	0.9997
1500	39.83	32.86	70.68	17.3	58.8	8.4	(18)	7.0	(11)	78.6	0.0473	0.9997
2000	48.41	42.09	82.44	26.4	89.7	18.2	(65)	6.8	(12)	55.6	0.0442	0.9999

Table VIII. Class a2, DECstation 5000/240 (in seconds)

	Running Time			Root Node	Completely Solved	Explored Nodes		Optimal Solution Level		Son	Sparse Matrix	
n	Avg	Median	Max	Time	MAPs	Avg	(Max)	Avg	(Max)	Nodes	Density	LB_0 / z^*
500	3.30	1.87	6.25	1.5	49.5	17.6	(34)	5.4	(8)	22.2	0.0774	0.9987
1000	16.69	13.68	27.78	7.2	129.0	23.9	(60)	5.8	(9)	51.0	0.0501	0.9993
1500	37.50	22.44	87.74	17.1	67.5	14.5	(43)	6.6	(12)	50.6	0.0621	0.9995
2000	55.75	54.13	79.05	28.3	115.2	20.4	(45)	5.5	(9)	67.2	0.0421	0.9996

Table IX. Class a3 (Artificial *UB*), DECstation 5000/240 (in seconds)

	Running Time			Root Node	Completely Solved	Explored Nodes		Optimal Solution Level		Son	Sparse Matrix	
n	Avg	Median	Max	Time	MAPs	Avg	(Max)	Avg	(Max)	Nodes	Density	LB_0 / z^*
500	3.24	2.39	5.75	1.2	54.4	19.3	(40)	5.5	(8)	24.1	0.0784	0.9987
1000	16.70	10.36	51.45	7.4	129.8	28.4	(139)	5.4	(9)	40.7	0.0538	0.9993
1500	41.43	27.06	124.65	19.2	303.0	38.8	(151)	6.2	(12)	79.0	0.0429	0.9992
2000	93.21	60.06	178.30	54.3	363.9	42.2	(114)	6.4	(12)	75.5	0.0315	0.9996

Table X. Class f1, PC 486/33 (in seconds)

	Running Time			Root Node	Completely Solved	Explored Nodes		Optimal Solution Level		Son	Sparse Matrix	
n	Avg	Median	Max	Time	MAPs	Avg	(Max)	Avg	(Max)	Nodes	Density	LB_0 / z^*
100	0.87	0.82	1.54	0.60	26.6	10.4	(24)	5.50	(7)	7.6	0.267	0.9988
200	4.05	3.90	4.89	3.24	27.3	9.9	(20)	5.10	(9)	13.3	0.069	0.9997
300	12.15	12.03	13.68	10.90	34.1	10.0	(28)	5.90	(9)	15.9	0.106	0.9998
400	28.04	26.59	37.80	24.58	80.4	13.5	(31)	6.70	(11)	21.0	0.131	0.9998
500	49.46	47.91	60.93	46.91	35.8	7.6	(16)	5.10	(9)	23.4	0.083	0.9998
600	89.89	85.66	112.69	78.74	73.8	19.2	(92)	7.40	(10)	24.6	0.211	0.9999
700	127.96	127.36	136.59	122.31	63.0	10.0	(26)	6.40	(10)	36.1	0.068	0.9999
800	190.30	185.33	217.09	176.92	61.8	15.1	(45)	7.40	(12)	35.8	0.085	0.9998
900	249.26	244.34	280.05	239.65	41.8	8.9	(45)	5.60	(10)	53.9	0.040	0.9999
1000	329.13	325.82	342.59	322.73	39.1	4.3	(10)	4.70	(9)	43.9	0.027	0.9998

Table XI. Class f2, PC 486/33 (in seconds)

n	Running Time			Root Node Time	Completely Solved MAPs	Explored Nodes		Optimal Solution Level		Son Nodes	Sparse Matrix Density	LB_0 / z^*
	Avg	Median	Max			Avg	(Max)	Avg	(Max)			
100	4.09	3.13	11.43	0.61	184.9	97.4	(236)	9.90	(13)	5.5	0.686	0.9961
200	26.41	12.91	82.31	3.90	440.5	239.8	(870)	11.50	(17)	6.0	0.717	0.9980
300	60.23	21.43	362.69	12.11	491.7	230.7	(1749)	10.90	(18)	7.2	0.594	0.9989
400	72.00	58.42	204.45	26.96	479.3	208.5	(1595)	7.10	(15)	7.8	0.189	0.9994
500	90.82	73.08	207.58	52.33	350.3	135.7	(514)	9.50	(17)	6.2	0.253	0.9994
600	113.72	108.02	158.46	89.84	127.4	48.7	(135)	9.60	(13)	7.4	0.324	0.9995
700	294.11	195.38	674.67	138.18	453.2	218.9	(787)	7.70	(14)	10.7	0.419	0.9995
800	350.64	291.54	769.89	200.24	298.0	120.4	(418)	9.20	(15)	13.4	0.473	0.9996
900	615.83	627.53	816.32	281.84	514.9	198.2	(421)	10.40	(20)	6.0	0.579	0.9994
1000	572.45	498.14	654.48	365.87	458.3	96.7	(236)	9.80	(16)	16.7	0.628	0.9997

Table XII. Stacker Crane Problems, PC 486/33 (in seconds)

Problem	n	Running Time	Root Node Time	Completely Solved MAPs	Explored Nodes	Optimal Solution Level	Son Nodes	Sparse Matrix Density	LB_0 / z^*
S48A	48	0.05	0.04	1	1	1	0.0	0.000	1.000
S49A	49	0.05	0.04	1	1	1	0.0	0.000	1.000
S50A	50	0.05	0.04	1	1	1	0.0	0.000	1.000
S50B	50	0.12	0.05	6	6	2	3.1	0.220	0.995
S50C	50	0.06	0.04	1	1	1	0.0	0.000	1.000
S50D	50	0.06	0.04	3	1	1	2.0	0.218	0.998
S323A	323	4.07	3.84	1	1	1	0.0	0.000	1.000
S341A	341	4.53	4.03	1	1	1	0.0	0.000	1.000
S358A	358	4.02	3.76	1	1	1	0.0	0.000	1.000
S378A	378	5.35	4.86	1	1	1	1.0	0.000	1.000
S399A	399	4.96	4.36	1	1	1	1.0	0.000	1.000
S403A	403	4.02	3.76	1	1	1	1.0	0.000	1.000
S416A	416	4.50	4.12	1	1	1	1.0	0.000	1.000
S423A	423	5.72	4.98	1	1	1	1.0	0.000	1.000
S443A	443	4.05	3.67	1	1	1	1.0	0.000	1.000

another and (b) decide the sequence of movements of the crane in order to minimize the total distance covered by the crane. Problem (b) determines an asymmetric traveling salesman problem. We solved real-world problems with up to 443 movements, derived from a Siemens factor in Augsburg. The corresponding results are given in Table XII. All the problems were easily solved with a maximum computing time of 5.7 seconds, on a PC 486/33. In many cases the problem was solved at the root node.

ACKNOWLEDGMENTS

We are grateful to Martin Grötschel and Norbert Ascheur (ZIB, Berlin) for the data of the stacker crane problem instances tested in Section 3. We are indebted to two anonymous referees and mainly to D. S. Johnson for several suggestions which have considerably improved the article.

REFERENCES

BALAS, E. AND CHRISTOFIDES, N. 1981. A restricted Lagrangean approach to the travelling salesman problem. *Math. Program. 21*, 19–46.

BALAS, E. AND TOTH, P. 1985. Branch and bound methods for the travelling salesman problem. In *The Traveling Salesman Problem*, G. Lawler, J. K. Lenstra, A. Rinnooy Kan, and D. Shmoys, Eds. John Wiley, New York, 361–401.

BELLMORE, M. AND MALONE, J. C. 1971. Pathology of traveling salesman subtour elimination algorithms. *Oper. Res. 19*, 278–307.

CARPANETO, G. AND TOTH, P. 1980. Some new branching and bounding criteria for the asymmetric travelling salesman problem. *Manage. Sci. 26*, 736–743.

CARPANETO, G. AND TOTH P. 1987. Primal-dual algorithms for the assignment problem. *Discr. Appl. Math. 18*, 137–153.

CARPANETO, G., DELL'AMICO, M., AND TOTH, P. 1989. Ricerca di percorsi Hamiltoniani in grafi orientati di grandi dimensioni. In the *13th AMASES Conference* (Verona, Italy).

CARPANETO, G., DELL'AMICO, M., AND TOTH, P. 1995. Algorithm 750: CDT: A subroutine for the exact solution of large-scale, asymmetric traveling salesman problems. *ACM Trans. Math. Softw. 21*, 4 (Dec.). This issue.

JOHNSON, D. B. 1977. Efficient algorithms for shortest paths in sparse networks. *J. ACM 24*, 1–13.

KARP, R. M. 1979. A patching algorithm for the nonsymmetric traveling salesman problem *SIAM J. Comput. 8*, 561–573.

MILLER, D. L. AND PEKNY, J. F. 1989. Results from a parallel branch and bound algorithm for the asymmetric traveling salesman problem. *Oper. Res Lett. 8*, 129–135.

MILLER, D. L. AND PEKNY, J. F. 1991. Exact solution of large asymmetric traveling salesman problems. *Science 251*, 754–761.

PADBERG, M. AND RINALDI, G. 1991. A branch-and-cut algorithm for the resolution of large scale symmetric traveling salesman problems. *SIAM Rev. 33*, 60–100.

PAPADIMITRIOU, C. J. AND KANELLAKIS, P. C. 1980. Flowshop scheduling with limited temporary storage. *J. ACM 27*, 533–549.

PEKNY, J. F., MILLER, D. L., AND STODOLSKY, D. 1991. A note on exploiting the Hamiltonian cycle problem substructure of the asymmetric traveling salesman problem. *Oper. Res. Lett. 10*, 173–176.

PEKNY, J. F. AND MILLER, D. L. 1992. A parallel branch and bound algorithm for solving large asymmetric traveling salesman problems. *Math. Program. 55*, 17–33.

SMITH, T. H. C., SRINIVASAN, V., AND THOMPSON, G. L. 1977. Computational performance of three subtour elimination algorithms for solving asymmetric traveling salesman problems. *Ann. Discr. Math. 1*, 495–506.

Received January 1990; revised May 1993; accepted December 1994