EXACT SOLUTION OF LARGE SCALE ASYMMETRIC TRAVELLING SALESMAN PROBLEMS.

G. Carpaneto (+), M. Dell'Amico (†), P. Toth (‡)

(+) Dipartimento di Economia Politica, Università di Modena, Italy.
(†) Dipartimento di Elettronica e Informazione, Politecnico di Milano, Italy.
(‡) D.E.I.S., Università di Bologna, Italy.

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: Branch and Bound, Asymmetric Travelling Salesman Problem, Assignment Problem, Subtour Elimination, Reduction Procedure.

Abstract

A lowest-first branch and bound algorithm for the *Asymmetric Travelling 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 2,000 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 1,000 jobs in less than 11 minutes, and real world *stacker crane* problems with up to 443 movements in less than 6 seconds.

1 INTRODUCTION

Consider a complete *digraph* $G = (V, A)$ with vertex set $V = \{1, ..., n\}$, arc set $A = \{(i, j) : \ i \in V, \ j \in V\}$, and a cost $a_{i,j}$ associated with each arc $(i, j) \in A$ ($a_{i,i} = \infty \ \forall i \in V$). We define a new graph $G' = (V', \ A')$ with vertex set $V' = \{v_1, ..., v_p\} \subseteq V$ and arc set $A' = \{(v_1, v_2), \ (v_2, v_3), ..., (v_p, v_1)\} \subseteq A$ 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 Travelling 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:

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

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

$$\sum_{i \in S} \sum_{j \in S} x_{i,j} \leq \mid S \mid -1 \quad , \qquad \forall S \subset V, \quad S \neq \emptyset \tag{4}$$

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

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 non-negative integers. (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, Srinivasan and Thompson [16], Carpaneto and Toth [4], Balas and Christofides [1] and Pekny, Miller and Stodolsky [14]; a survey of enumerative algorithms for the TSP is given in Balas and Toth [2]. A parallel algorithm has recently been proposed by Miller and Pekny [10, 14]. As for sequential algorithms the maximum size of *uniformly randomly-generated* for which many instances have been solved is 5,000, although single random instances with as many as 500,000 vertices (but small cost ranges) have been solved by Miller and Pekny [11]. For the undirected graph case (Symmetric Travelling Salesman Problem), a *Euclidean* instance with 2,392 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 [12]). A similar approach was recenty presented by Applegate, Bixby, Chvatál and Cook at the SIAM conference 1993. They solved instances with 3,038 and 4,461 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 [7]. 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 2,000 vertices are solved on a DECstation 5000/240 computer (with 16 Mbytes of main memory) in less than 3 minutes of CPU time. In addition, we solved on a PC 486/33 (with 8 Mbytes of main memory) *no-wait flow shop* problems (see Papadimitriou and Kanellakis [13]) with up to 1,000 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 500/240 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 the paper has been presented at the 13-th AMASES congress [6].

2 ALGORITHM

The algorithm is derived from the lowest-first branch and bound procedure TSP1 presented in Carpaneto and Toth [4]. At each node h of the decision tree TSP1 solves a *Modified Assignment Problem* (MAP_h) defined by (1), (2), (3), (5) and the additional constraints associated with arc subsets E_h and I_h, where:

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

If the optimal solution to MAP_h does not define a Hamiltonian circuit and its value LB_h (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 [3]).

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 $q-th$ subtour.

The subtour, say G_p, having the minimum number of not included arcs, that is the subtour such that

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

is chosen for branching.

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

$$E_{g(j)} = E_h \cup \{(s_j, t_j)\};$$

$$I_{g(j)} = I_h \cup \{(s_i, t_i) : i = 1, \ldots, j-1\}.$$

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 in [4] 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 $\widetilde{G} = (V, \widetilde{A})$, 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 $O(\mid \widetilde{A} \mid \; log\ n)$ 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 [5]. 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 hnown that for each arc $(i,j) \in A$ the *reduced cost* $\overline{a}_{i,j} = a_{i,j} - u_i - v_j \geq 0$ 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 an ATSP feasible solution of value UB is known, then each arc $(i,j) \in A$ such that

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

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 so be transformed into the equivalent sparse one, $\widetilde{G} = (V, \widetilde{A})$, where

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

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 [9].

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 greatly depends 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 $(s,t){=}E_h \backslash E_k$). 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 $(\overline{a}_{i,j})$. Addition of the new included arcs (contained in subset $I_h \backslash I_k$), 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 $\widetilde{G}$ is sparse, the shortest augmenting path is found through a procedure derived from the labelling algorithm proposed by Johnson [8] for the computation of shortest paths in sparse graphs, which utilizes a heap queue. Hence, the resulting time complexity for solving each MAP is $O(\mid \widetilde{A} \mid \; log\ n)$.

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 the non-decreasing 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 information*, 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 MAP's 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:

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

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 labelling 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 considered node) 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 considered node 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 $E_r \backslash E_q$ and impose, level by level, all constraints corresponding to arcs in $E_k \backslash E_q$. The current dual variables (v_j) associated with node k (which implicitly define the set of included arcs I_k) 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.
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. $\overline{a}_{i_a,j_b} = \overline{a}_{i_b,j_a} = 0$), then an equivalent optimal solution to MAP_h can be obtained by connecting subtours G_a and G_b to form a unique subtour $G_a = (V_a \cup V_b, A_a \cup A_b \backslash ((i_a, j_a) \cup (i_b, j_b)) \cup ((i_a, j_b) \cup (i_b, j_a)))$.
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 worst 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 in the first of the nineties [10,11,14,15]. In the same period we independently developed the code described in this paper, whose first version was presented at the 13-th AMASES conference, 1989, Verona [6]. All these procedures are based on the general approach presented in Carpaneto and Toth [4]. Here we discuss the main differences and similitudes between these approaches. In [10] Miller and Pekny presented a preliminary algorithm which is a parallelization of the approach of Carpaneto and Toth, improved with the application of the patching heuristic [9] at the root node. Randomly generated instances with up to 3,000 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 in [15] represent a substantial improvement of the original parallel procedure. The MAP's 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 sufficent 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 in [14] is a modification of that presented in [15], 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 3,000 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 in [11], which includes all the improvements previously proposed by the authors. Many instances with 5,000 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 12623 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 [4], (b) the MAP's 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 subsection 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 efficent 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 the last section of this paper it appears that our code is slowler than the algorithm presented in [11], 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 instance with 4,000 vertices and costs in [0,10^6], in only 14 minutes on a SGI Challenge (239 subproblems were solved).

3. COMPUTATIONAL RESULTS

The algorithm has been coded as a FORTRAN subroutine called CDT [7]. Subroutine CDT has been tested on randomly generated test problems with up to 2,000 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 $\{1, 2, ..., m\}$ of m *machines* are given. Each job must be scheduled on machines $1, 2, ..., m$ 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 minimises the completion time of the last job processed on machine m.
Papadimitriou and Kanellakis [13] have shown that an instance of the no-wait flow shop problems 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 1 to 11 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 $MAP's$ 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. $\mid \widetilde{A} \mid /n^2$);
- average (AP solution value at the root node)/z^* ratio.

Tables 1 to 6 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 $t1$, $t2$ and $t3$ (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 [9].

Tables 1,2 and 3 (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 tendence 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 4, 5 and 6 (triangular problems) show that the running time required for solution of the $MAP's$ is much greater than that corresponding to uniform problems. In fact, procedure CTCS [5], 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 1, 2 and 3, 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 7,8 and 9 give the results for problems of classes $a1$, $a2$ and $a3$, values of n from 500 to 2,000. The algorithm has a behaviour similar to that shown in Tables 1, 2 and 3.

Tables 10 and 11 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 then 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 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 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 Travelling Salesman Problem. We solved real world problems with up to 443 movements, derived from a Siemens's factory in Augsburg. The corresponding results are given in Table 12. All the problems were easily solved with a maximun computing time of 5.7 seconds, on a PC 466/33. In many cases the problem was solved at the root node.

ACKNOWLEDGEMENTS

We are grateful to professors Martin Grötschel and Norbert Ascheur (ZIB, Berlin) for the data of the stacker crane problem instances tested in Section 3. We are indebited to two anonymous referees and mainly to Prof. D.S. Johnson for several suggestions which have considerably improved the paper. Work supported by M.U.R.S.T., Italy.

REFERENCES

1. Balas, E., Christofides, N., "A Restricted Lagrangean Approach to the Traveling Salesman Problem", **Math. Progr.** 21, 1981, 19-46.

2. Balas, E. , Toth, P., "Branch and Bound Methods for the Travelling Salesman Problem", in **"The Traveling Salesman Problem"** (G. Lawler, J. K. Lenstra, A. Rinnooy Kan, D. Shmoys, eds.), J. Wiley, 1985, 361-401.

3. Bellmore, M., Malone, J. C., "Pathology of Traveling Salesman Subtour Elimination Algorithms", **Op. Res.** 19, 1971, 278-307.

4. Carpaneto, G., Toth, P., "Some new Branching and Bounding Criteria for the Asymmetric Travelling Salesman Problem", **Management Science** 26, 1980, 736-743.

5. Carpaneto, G., Toth, P., "Primal-Dual Algorithms for the Assignment Problem", **Discrete Applied Mathematics** 18, 1987, 137-153.

6. Carpaneto G., Dell'Amico M., Toth P., "Ricerca di Percorsi Hamiltoniani in Grafi Orientati di Grandi Dimensioni", **13-th AMASES conference**, Verona, 1989.

7. Carpaneto G., Dell'Amico M., Toth P., "Algorithm CDT: a Subroutine for the Exact Solution of Large-Scale Asymmetric Travelling Salesman Problems", **Tech. Report, Dipartimento di Economia Politica**, University of Modena, Italy, 1990.

8. Johnson D.B., "Efficient Algorithms for Shortest Paths in Sparse Networks", **J.A.C.M.**, 24, 1977.

9. Karp, R.M., "A Patching Algorithm for the Nonsymmetric Traveling Salesman Problem" **SIAM J. Comput** 8, 1979, 561-573.

10. Miller, D. L. and Pekny, J. F., "Results from a Parallel Branch and Bound Algorithm for the Asymmetric Traveling Salesman Problem", **Operations Research Letters**, 8, 1989, 129-135.

11. Miller, D. L. and Pekny, J. F., "Exact Solution of Large Asymmetric Traveling Salesman Problems", **Science**, 251, 1991, 754-761.

12. Padberg, M., Rinaldi, G., “A Branch-and-Cut Algorithm for the Resolution of Large Scale Symmetric Traveling Salesman Problems”, **SIAM Review**, 33, 1991, 60-100.

13. Papadimitriou, C.H., and Kanellakis P.C., “Flowshop Scheduling with Limited Temporary Storage”, **J. ACM**, 27, 1980, 533-549.

14. Pekny, J. F., Miller, D. L., Stodolsky, D. “A note on exploiting the Hamiltonian cycle problem substructure of tge Asymmetric Traveling Salesman Problem”, **Operations Research Letters**, 10, 1991, 173-176.

15. Pekny, J. F., Miller, D. L. “ A parallel branch and bound algorithm for solving large asymmetric traveling salesman problems”, **Mathematical Programming**, 55, 1992, 17-33.

16. Smith, T. H. C., Srinivasan, V., Thompson, G. L., “Computational Performance of Three Subtour Elimination Algorithms for Solving Asymmetric Traveling Salesman Problems”, **Ann. Discrete Math.** 1, 1977, 495-506.