A Memory Adaptive Reasoning Technique for Solving the Capacitated Minimum Spanning Tree Problem

R. Patterson[1], E. Rolland[2], and H. Pirkul[3]

[1,3] *School of Management*
The University of Texas at Dallas
Richardson, Texas 75083-0688
Phone: (972) 883-2736
rpatters@utdallas.edu; hpirkul@utdallas.edu

[2] *Fisher College of Business*
Department of Accounting and MIS
The Ohio State University
Columbus, OH 43210
Phone: (614) 292-7692
rolland.1@osu.edu

Date: September 4, 1998

Abstract.

In this paper we propose a hybrid memory adaptive heuristic for solving the Capacitated Minimum Spanning Tree (CMST) problem. We augment the problem formulation with additional non-redundant constraints via use of adaptive memory, to improve upon the performance of an elementary heuristic (the Esau-Williams heuristic). Our methodology is tested against many of the previously reported heuristics for the CMST. We conclude that our generalized procedure performs on par with the best of these approaches in terms of solution quality, while expending a very modest amount of computational effort.

Key words: capacitated minimum spanning tree, heuristic search, telecommunications network design

1. INTRODUCTION

The capacitated spanning tree problem (CMST) is one of connecting a set of demand nodes to a central node through a minimum-cost tree network. This problem plays an important role in the design of local access telecommunications networks, and is known to be NP-complete (Papadimitriou, 1978). The CMST has in the past been solved using heuristics (Esau & Williams, 1966; Kershenbaum & Chou, 1974; Karnaugh, 1976; Kershenbaum et al., 1980; Gavish & Altinkemer, 1986; Amberg et al., 1996; Sharaiha et al. 1997), cutting-plane methods (Gouveia & Martins, 1995; Hall, 1996), and integer programming (Gavish, 1983). The heuristic approaches are often faster than the cutting-plane and integer programming approaches and thus suitable for solving larger problem instances, but the quality of the solutions are typically not as good.

The goal of our research efforts is the development of an effective, problem independent, heuristic procedure which produces high-quality solutions. To accomplish this, we develop the adaptive reasoning technique (ART). Let us define a solution path as being the sequence of choices made by a greedy heuristic, which result in a feasible solution to the problem. The ART methodology is a generalized approach that externally modifies the solution path of a greedy heuristic. Thus, the greedy solution technique is self-adaptive to specific problem instances by learning from the problem instance at hand. The main algorithm learns from information contained in the solutions found by the greedy heuristic. The modifications made by the ART methodology are external to the greedy heuristic because the greedy heuristic remains unchanged. Rather than changing the greedy heuristic, additional constraints change the problem the greedy heuristic is solving. The ART methodology adds intelligence regarding the problem instance to the knowledge already embedded in the greedy solution technique, and it adds this knowledge by creating additional constraints which simply prohibit the use of particular decision variables. By altering a single choice in the solution path, the subsequent choices made by the greedy heuristic are altered. The knowledge added by the ART methodology, together with the problem knowledge embedded in the greedy heuristic, create a very powerful solution technique.

This paper is structured as follows: in the next section we outline a formulation for the CMST, and some of the previous heuristic solution methods are reviewed in section 3. We propose a memory based solution procedure in section 4. Section 5 demonstrates the computational results, while our conclusions are summarized in section 6.

2. PROBLEM FORMULATION

The Capacitated Minimum Spanning Tree problem is one of connecting a set of demand nodes to a central node through a minimum-cost tree network. Given a graph, *G(V, E)*, where *V* is the set of demand nodes (with the associated demand vector D_v), *E* is the set of possible edges in the graph (with the associated arc cost vector C_e, and an arc capacity *K*), and V^* is a designated goal node, the objective is to find a minimum-cost spanning tree that is rooted in node V^* where each sub-tree branch from node V^* does not contain more than *K* nodes. Various formulations have been proposed (Gavish, 1982 and 1983; Gouviea, 1993 and 1995). Gavish (1983) has formulated the goal-directed capacitated minimum spanning tree problem as follows:

$$\mathit{Minimize}\ Z = \{\sum_{i=2}^{n} \sum_{\substack{j=1 \\ j \neq i}}^{n} C_{ij} X_{ij}\} \qquad (1)$$

Subject to :

$$\sum_{\substack{j=1 \\ j \neq i}}^{n} X_{ij} = 1 \qquad i = 2, ..., n, \qquad (2)$$

$$\sum_{\substack{j=1\\ j\neq i}}^{n} Y_{ij} - \sum_{\substack{j=2\\ j\neq i}}^{n} Y_{ji} = 1 \qquad i = 2, \ldots, n, \qquad (3)$$

$$Y_{ij} \leq KX_{ij} \qquad i = 2, \ldots, n, \quad j = 1, 2, \ldots, n \text{ and } i \neq j \qquad (4)$$

$$Y_{ij} \geq 0 \text{ and } X_{ij} = 0 \text{ or } 1 \qquad \forall i, j \qquad (5) \text{ and } (6)$$

where X_{ij} = 1 if arc *(i,j)* is included in the spanning tree; 0 otherwise;
Y_{ij} = the flow on the arc *(i,j)*;
K = the capacity restriction limit on the flow on any arc.

We here assume fixed capacity on each arc of *K*, fixed load requirements from each node of 1 unit, and non-uniform arc costs of C_{ij} for establishing a link between nodes *i* and *j*. The objective function then minimizes the total cost of the communication arcs which are used. Unit flows originating at every node are sent to a single goal node. Constraint set (2) implies that the flow from every node is sent to only one other node. In this formulation, node 1 is designated as the goal node. Constraint set (3) implies that the cumulative flow coming out of every non-goal node is one more than the cumulative load going into that node. The average load for each node is set to one. Constraint set (4) implies that the flow coming out of any node on an activated (or used) arc will not exceed the capacity (*K*) of that arc. Constraint set (5) implies that flows on all arcs are non-negative, and constraint set (6) implies that an arc is either used or is not used. The cost of using an arc is fixed regardless of the volume of flow on the arc. The costs of opening particular arcs in this problem are not identical.

3. PREVIOUS RESEARCH

One of the first, and best-known, heuristics for the CMST was proposed by Esau and Williams in 1966 (we refer to this heuristic as the EW algorithm). This procedure starts with an "empty" spanning tree (no arcs are used), and then adds arcs in a greedy fashion based on a modified cost structure without violating the capacity constraint, until it finds a feasible spanning tree. The EW modifies the cost structure for an arc (i,j) to be the arc cost (c_{ij}) less the minimum distance directly to the goal node (min { c_{iV*}, c_{jV*}}). The time complexity of the EW heuristic is $O(n^2 \log_2 n)$ (Amberg et al, 1996).

Gavish and Altinkemer (1986) and Altinkemer and Gavish (1988) proposed the Parallel Savings Algorithm (PSA) method for solving the CMST. The PSA is a solution procedure which has been applied with various degrees of success. Its' computational complexity is of order $O(n^3 \log_2 K)$ (Amberg et al., 1996). Hall (1996) presents a cutting plane algorithm that often produces good solutions, but typically requires excessive CPU times. Gouveia and Martins (1995) use a cutting plane algorithm with sub-tour elimination constraints, but their method often requires extraordinarily long CPU times.

Given the limitations and shortcomings of the cutting-plane and heuristic solution procedures, a meta-heuristic algorithm is a viable solution approach. Sharaiha et al. (1997) developed a tabu search (TS) procedure for solving the CMST. This procedure produced some good solutions (compared to the bounds) for some well-known CMST instances, but the results were not formally compared to those of previously proposed, high-quality, solution methods. This algorithm is designed specifically for the CMST problem, using techniques related to problem structure to improve on the solutions.

Amberg et al. (1996) provided an overview of past solution approaches to the CMST. They found that most of the past solution methods are focused on arcs when generating or transforming a solution. As a result of their findings, they proposed an algorithm that focuses on transformations of node partitions (rather than arc partitions). Given the node assignments of a particular solution, they propose a local search procedure based on node exchanges. To enable the search to escape local optima, they investigate the use of simulated annealing and tabu search as meta-heuristics. In addition to studying the effects of parameters for the various methods they proposed, our interpretation of the results as related to well-known test problems leads us to believe that their solution approach is a superior one in terms of solution quality (see section 5 below for a discussion of solution quality). Further, the computational complexity of their approach is also very reasonable, being linear with problem size if changes in the cost matrix are allowed to be stored. The time complexity, as stated by the authors, is $O(nK^3)$ if exchange moves are allowed between node sets, and $O(nK^2)$ when allowing for simple moves (no exchanges) only.

Karnaugh (1976) proposed a second-order heuristic incorporating a modified EW solution method for the multipoint network optimization problem. The second-order heuristic is somewhat related to the modern memory-based methods. His technique proposes two subroutines, *Inhibit* and *Join*, which when used within the framework of a modified EW procedure produce high-quality solutions. Either subroutine alone, or applied successively, are used to rule arcs in (*Join*) or out (*Inhibit*) of a given solution.

The main idea behind Karnaugh's *Inhibit* procedure is to test what would happen to the solution if a particular arc was ruled out of consideration (this can be operationalized by selecting an arc, setting its cost high, and applying the EW heuristic. This must be repeated for every arc). After testing the status of making every chosen arc inhibited, the inhibition yielding the lowest cost solution is made permanently inhibited (or tabooed or prohibited). This approach is intriguing, but rather inefficient from a computational standpoint (approximately $O(n^4)$, Amberg et al., 1996).

The *Join* procedure is a node inspection procedure which determines if a node should be connected to its nearest neighbor, or to its nearest neighbor closer to the goal node than the node itself (unless this is already the case in the current solution). The corresponding arc is then locked into the solution, and EW is run to find the new cost of the solution. The best Join is then made permanent, and the procedure is repeated. The complexity of the procedure is again approximately on the order of $O(n^4)$, (Amberg et al., 1996).

Kershenbaum et al. (1980) proposed a second-order heuristic for the CMST which attempts to improve upon the EW solution. This algorithm forces inclusion of one or more arcs found by solving the minimum spanning tree (MST) problem which the EW solution of the CMST problem omitted. The procedure iteratively solves the MST problem and the EW algorithm for the CMST problem, compares the solutions, and then forces arcs into the next iteration of the EW solution by modifying the arc costs. The time complexity is bounded above by $O((n-1)^3)$. This is a worst-case analysis of the time complexity where none of the arcs in the MST solution are present in the EW solution and limiting the examined subsets of solutions to contain not more than 2 elements (Amberg et al., 1996).

In creating a new heuristic for the CMST, we focused on addressing some of the weaknesses of previously proposed heuristics, and at the same time construct a procedure that would be easily generalizable to other problem domains.

4. A SOLUTION APPROACH TO THE CMST

Many meta-heuristic solution approaches, and most adaptive memory programming techniques, belong to the class of "generate-and-test" search techniques. In pure generate-and-test, a candidate solution is generated and then sent to an evaluator for testing. If the candidate solution is not satisfactory, then the procedure is repeated. In a memory-based meta-heuristic, we retain some information regarding our past solutions, and often prohibit the immediate re-use of certain elements of those past solutions by creating a memory of used solution elements. In addition, one can also ensure that certain solution elements remain available, and are never prohibited. Tabu search is one such search technique, and is based on (human) memory functions (for a comprehensive summary, see Glover, 1997). In tabu search the generate and test procedure is typically (but not always) applied to candidate solutions that are generated from a "neighborhood" of previously tested solutions using certain tabu search memory functions. The transformation from one solution to a neighboring solution is known as a move.

In this paper, we propose the learning-based meta-heuristic ART which uses some of the memory functions of tabu search, but conceptually deviates significantly from the tabu methodology. Primarily, we do not utilize the notion of solution neighborhoods and move functions to traverse such neighborhoods. Instead, we propose a constructive procedure that is applied iteratively. The basis of our proposed algorithm is to iteratively solve the CMST by using the EW heuristic, and at each iteration modify a set of additional constraints. These constraints, which when applied to the simplistic EW procedure, provide an efficient means of searching the problem space. The conceptual difference between our procedure and regular tabu search, is that we do not define search neighborhoods and transition functions (moves). Rather, we impose dynamic constraints on a simple heuristic procedure. One imposed constraint alone is enough to yield a solution that may be entirely different from a solution found when no such constraint was present. We will use the CMST problem and the EW heuristic to demonstrate our generalized learning and heuristic design rules in the next sections.

The Esau-Williams heuristic solution procedure for the CMST is deterministic, extremely modest in computational requirements, and always obtains a feasible solution. However, computational experiments show that the weakness of the EW procedure is in its greedy functionality: that is, an arc that “looks good” (i.e., the arc is cheap) early on in the arc selection process, may lead to later arc selections that result in a high-cost solution. Adding constraints to prohibit the selection of particular arcs may compensate for biases which cause the EW heuristic to greedily make the “wrong” decision. We recognize that prohibited arcs may in fact be arcs that should be included in an optimal solution. The “expiration time” of the constraint prohibiting a particular arc will allow a previously prohibited arc to be reconsidered later in the search, and possibly in the context of a new set of additional arc constraints. We use the prohibition of arcs as a method of forcing the simplistic EW procedure to search other regions of the problem space.

Terminology and Description of the Algorithm

The *constraint memory* is a list containing a *time duration* (a prohibition time similar to a tabu time) during which selection of the decision variable is prohibited. The time duration is measured in *number of iterations* of executing the EW heuristic. The *depth of learning* is the length of commitment (in number of iterations) that a new constraint prohibiting the use of a particular decision variable will endure. We set the depth of learning initially to 40 (which is a data dependent parameter setting) and reduce the depth of learning by 10% at the end of each phase. A *phase* is a complete cycle of our algorithm with a single depth of learning (see Figure 1). We execute a total of 25 phases. The stored constraint memory associated with the best solution is used as the starting constraint memory for each phase. *Forgetting* is the process of removal of constraints whose time durations have lapsed. *Shortening the constraint memory* reduces the length (time duration) of current prohibitions on the use of decision variables. Forgetting can be accelerated by dynamically shortening select elements of the constraint memory. In our experiments, a shortening function of 20% was used. This means that for 20% of the arcs, the length of prohibition was reduced proportionally based on a random draw from a uniform distribution in the range [0,1] at the end of each phase. Also, after each iteration of the EW heuristic, if a decision variable has been prohibited, then there is a very small chance that the memory for this decision variable will be reset to zero (the prohibition is removed). The probability of prohibition removal is dynamic and is equal to $\frac{0.1}{\mathit{Phase\ number}}$.

The *learning rate* is the frequency with which you decide to make a decision variable constrained (prohibited). This frequency is measured in percent. The *learning rate increase* (LRI) is the stepwise increase in the learning rate over a specific time frame. Five (5) learning rate *cycles* are made within each phase, where LRI is increasing over the cycles. The learning rate is proportionally increased by the LRI as a function of the cycles as follows: Learning Rate=$2*(1+\mathrm{LRI})^{(\mathrm{Cycle}-1)}$. That is, the learning rate starts out limited at 2%, and then increases by LRI% at each consecutive *cycle*. In our experiments we chose LRI to be 80%, resulting in learning rates of 2%, 3.6%, 6.48%, 11.66%, and finally 20.99% at each cycle respectively. In essence, as the learning rate cycle increases, the more likely ART is to prohibit constraints. This concept is one of gaining confidence in the prohibition choices as time lapses. The learning rates were determined by trial and error, and found to be reasonably effective, although not terribly sensitive. As a guideline, we found that the learning rate itself was most effective when started at some relatively small fraction (1-5%). The key here was not to exclude arcs too early in the search, and thus the low initial learning rate. Further, the LRI was effective both at the 80% and 90% levels. The key issue we found here was that we should be more confident about prohibitions later in the search process. For each learning rate, the *learning process* continues until 15 consecutive iterations occur without an improvement to the best feasible solution. The *best feasible solution* is the lowest cost solution found to that point in the algorithm. The learning process consists of solving the CMST using EW subject to additional arc constraints, then evaluate the solution, and create additional constraints which are valid for a particular time duration. Once you decide to add a constraint (prohibit a decision variable), the time duration with which the constraint will endure is based on the depth of learning, the cost of the arc, and a random probability (the time duration will be discussed in detail below when Figure 4 is discussed). Further rules are added to increase the probability that certain arcs will be chosen to be constrained (see below).

After having completed a reasonable number[1] of iterations, a subset of best solutions can be analyzed to determine the commonalities of their memories. This requires that the constraint memories associated with the best solutions are saved. The most fruitful analysis of the subset of "best memories" is to determine which decision variables the best answers have all left unconstrained. Once the algorithm has performed 200 iterations of the EW, we prohibit additional constraints on arcs which are not constrained in the constraint memories associated with <u>all</u> of the ten (10) best solutions found. Intuitively, after 200 iterations of the EW procedure, the 10 best solutions contain most of the worthwhile constraints. By limiting the subsequent search to the constraints contained in the top 10 "best memories", we substantially improve the solutions by focusing our search in a limited (and fruitful) search space. As the constraint memory expires, decision variables become available for use. If any decision variable is unconstrained in all of the top 10 best solutions at any point subsequent to the 200th iteration, then that decision variable can no longer be considered for prohibition. In this intensification process, the memory typically converges to a local minimum. In the 100 node problems which we examined in detail (discussed in the next section), we typically had about 30 decision variables (arcs) which were still allowed to be constrained at the 200th iteration mark. The number of constrained decision variables converged to between 5 and 10 by the end of the algorithm, so that the best solutions actually had very few constraints. The learning rules were sufficient to hone in on a (local or global) minimum point. This lends credence to the idea that inappropriate constraints expired and the appropriate decision variables were allowed to be used. Further, this enables us to propose an algorithm that does not use explicit local search.

Figures 1 through 4 depicts the flowcharts associated with the meta-heuristic algorithm described above. Figure 1 is an overview of the ART procedure. Figure 2 depicts the process of dynamically creating and removing arc constraints, whereas Figure 3 depicts the process of adding arc constraints.

We label an arc "chosen" if it is included in the most recent EW solution. Figure 4 depicts the probabilistic constraint establishment (PCE) for a chosen arc, and explains the probabilistic constraint establishment process noted in each process box of Figure 3. Note that the Learning Rate varies from low to high within each phase, and only impacts whether or not a particular constraint will be added. If a constraint is to be added to prohibit the selection of a particular arc, then the time duration for this additional constraint must be selected. The time duration is dependent on three variables: the depth of learning, a multiplier selected randomly from a uniform distribution [0,1], and the cost of the arc. The allocation of additional time steps is further described in Figure 3: one

[1] The 200 iteration mark was selected empirically as a reasonable number of iterations.

time step is added (to the constraint memory) if the PCE is applied to all arcs chosen by this iteration of the EW heuristic; two (2) time steps are added if the PCE is applied to the 3% most costly arcs chosen by this iteration of the EW heuristic; five (5) time steps are added if the PCE is applied to the highest cost arc on the longest CMST branch[2] chosen by this iteration of the EW heuristic; and five (5) time steps are added if the PCE is applied to the highest cost chosen arc on every CMST branch. Thus, a very costly chosen arc may have 4 chances to be prohibited: once for being a chosen arc, twice for being among the 3% of the most costly chosen arcs, three times for being the highest cost chosen arc on the longest CMST branch, and four times for being the highest cost chosen arc on this arc's particular branch. More expensive arcs are more likely to be chosen to be prohibited, and because the time duration is dependent upon the cost of the arc and the reason for the prohibition, their time duration is likely to be longer than for a cheaper arc. The specific choices for the additional time steps were determined empirically.

Comparing the ART method to the second-order heuristics, ART uses the learning rules and memory to establish additional constraints whereas the second-order heuristics use exhaustive local search among all of the arcs chosen to find the best decision variable to prohibit and require. ART seems to be effective than the second-order heuristics at finding a good set of additional constraints (see next section). As evidenced by the computational results in the next section, we believe that the particular ART learning rules and memory which we have implemented for the CMST are more effective than the second-order heuristic procedures. However, the worst-case time complexity of ART is less than that of the second-order heuristics. The time complexity of ART for the CMST problem is equal to that of the EW heuristic ($O(n^2 \ log_2 n)$) whereas the time complexity of the Karnaugh second-order heuristic implemented for the CMST problem is $O(n^4)$ and

[2] The longest CMST branch is the branch with the highest total cost.

the worst-case time complexity of the Kershenbaum et al. second-order heuristic implemented for the CMST problem is $O((n\text{-}1)^3)$.

The main advantage of the ART methodology is that the dynamic learning rule and memory functions try different combinations of additional constraints on the decision variables. With ART, the prohibition of a particular decision variable can enter, be removed, reenter, be removed again, and so on. All the while, prohibitions on other decision variables are being added and deleted. The ART learning rules and memory functions allow it to overcome its own early mistakes. In comparison, after each iteration of the constructive heuristic (in this case, EW), the second-order heuristics choose the best single prohibition and/or inclusion given all of the previous prohibitions, and then lock-in that constraint permanently. This technique does not allow the second-order heuristics to overcome a prohibition and/or inclusion that was greedily made but should not have been.

5. COMPUTATIONAL RESULTS

We obtained 20 well know problem sets from the OR-Library maintained by J. Beasley at the WEB site http://mscmga.ms.ic.ac.uk/info.html. The first ten problem instances (*tc40_i.txt* and *tc80_i.txt*, *i*=1..5) have the goal node in the center of the graph, whereas the other ten problem instances (*te40_i.txt* and *te80_i.txt*, *i*=1..5) have the goal node at the end of the node scatter. The problems consist of 40 nodes (*te40_i* and *tc40_i*) and 80 nodes (*te80_i* and *tc80_i*) respectively. The 40 node problems were solved using arc capacities of 3, 5, and 10, whereas the 80 node problems were solved with arc capacities of 5, 10, and 20. Thus, a total of 60 known problem instances were solved. The 60 problem instances are all fully connected graphs. The ART heuristic for the OR test bank problems was coded in Fortran 77, and executed on a 266MHz Pentium II-based personal computer with 96 MB of memory, running Windows 98. All problems were solved with unit demand data.

For the known problem sets (obtained from the OR-Library) we solved each problem instance using the EW heuristic and our own heuristic memory adjusting procedure (ART). For ART, we report the best solution found from 100 runs, as well as the average CPU time for each run. The bounds were kindly provided to us by Luis Gouveia (1995; and personal communication based on his research in progress). We further report results from Sharaiha (TS) et al. (1997) as reported in their paper. The actual results from Amberg (AMB) et al. (1996) do not explicitly appear in their paper, and were provided to us by Luis Gouveia. We also report the solutions from the Karnaugh algorithm (KAR) (1976), the Kerschenbaum et al. algorithm (KBO) (1980), and an improved version of the algorithm proposed by Gavish & Altinkemer (IPSA) (1986) as implemented for the CMST problem[3].

The results for the *tc* instances are reported in Table 1, and the results for the *te* instances are reported in Table 2. We see that for the *tc* problems, the average gap between the bounds and the EW procedure was 4.44%, compared to 1.08% for TS, 0.32% for AMB, 1.09 % for KAR, 3.29% for KBO, 4.74% for IPSA, and 0.98% for ART. For the *te* problems, the gaps were 5.57% for EW, 3.8% for TS, 0.65% for AMB, 2.34% for KAR, 5.17% for KBO, 9.11% for IPSA, and 1.39% for ART. Across both *te* and *tc*, the gap results were 5.01% for EW, 2.44% for TS, 0.49% for AMB, 1.72% for KAR, 4.23% for KBO, 6.93% for IPSA, and 1.18% for ART.   In conclusion, we see that Amberg et al. (1996) had devised a problem specific tabu search procedure that delivers very high quality solutions. Surprisingly, Karnaugh's second order heuristic from 1976 produces solutions that are better than those of more recent algorithms, with the exception of AMB and ART. We see a clear favor in using AMB and ART (over the other approaches) for solving large problem instances where the goal (or root) node is an end vertex. We further refer to Hall (1996, p. 228), who states: *"… we believe that by placing the root node in the center we are effectively making the capacity constraints less restrictive, compared to placing the root node in the corner; that is, there is an interplay between the capacity and the location of the root node."* We agree with this assessment, and conclude that both the AMB and ART method produces better solutions on average than the other methods when the problem constraints are more restrictive.

While CPU comparisons between the different solutions approaches are not directly available, it seems that the CPU times for the TS and ART procedures are fairly similar. AMB is limited to 600 seconds of CPU time on a fairly dated 66MHz 486 PC. This may indicate that the AMB approach is the most computationally efficient of all of the above approaches. The remaining heuristics, KAR, KB and IPSA, have associated computational complexities that are by far inferior to AMB and ART.

We verified that twelve of the known lower bounds for the 40 node *tc* problems are indeed equal to the optimal solution. For the *te* problems we also verified that six of the known lower bounds are indeed equal to the optimal solution. For all the test problems with capacity 5 or higher, we were not able to improve on the solutions found by Amberg et al. (1996), and thus we cannot claim that we have found any new best solutions to the test problems with capacity higher than 5. We have verified that the bounds for test problems *tc40_1* though *tc40_5* with capacity 3 are indeed equal to the optimal solutions. Further, we verified that the bounds for test problems *te40_1* and *te40_2* with capacity 3 are also equal to the optimal solutions. Gouveia (1995) states that the lower bounds for the tc40 and te40 problems with capacity 3 are in fact optimal.

---

^{3} These results were kindly provided to us by Luis Gouveia.

To comparatively test our algorithm on graphs structurally different from the known test problems, we generated three additional sets of graphs: 20 graphs with Euclidean arc distances, 20 with random arc distances, and 20 with mixed Euclidean and random arc distances (on the average, 50% Euclidean and 50% random arc distances). The rationale for using the latter graph set is that most real-world problems exhibit some mix of Euclidean and random arc distances. The randomness, or non-Euclidean features, are typically caused by physical obstacles, such as mountains, lakes, rivers, etc. For each of these graph types (Euclidean, Random, Mixed) we generated five problems with 50 nodes, 10 with 100 nodes, and 5 with 150 nodes. The graphs were generated by allocating random nodes with *x* and *y* integer coordinates taken from a uniform distribution in the [0,100] range. For the Euclidean problems, the Euclidean distances are used as arc costs. For the Random problems we use random costs on all arcs, drawn from a uniform distribution in the [0,141] range (141 is approximately the length of the diagonal in a 100x100 square). For the Mixed problems we generated the arc costs by using the Euclidean cost as a base, and adding a random component in the range [0, Euclidean cost base of the arc]. All costs are symmetric for all graph types.

We use the origin as the goal node (or root node) for the total flows (the origin is the corner node which is the lower left corner of the node scatter). For each graph we generated numerous problem instances by considering only subset of the total arcs in the graphs. These subsets were created by setting an arc density of "*n*", by selecting only the *n* cheapest arcs attached to each node from the complete graph. We then created nine problem instances by varying *n* from 5 to 45 for the 50 node graph; 19 problem instances by varying *n* from 5 to 95 for the 100 node graph, and 29 problem instances by varying *n* from 5 to 145 for the 150 node graph (for each of the two graph types). Thus we generated a total 45 problem instances with 50 nodes, 190 problem instances with 100 nodes, and 145 problem instances with 150 nodes for each graph type (Euclidean, Random, Mixed). For each 50 node instance, we solved the problem set the arc capacity to 4, and for the 150 node instances the arc capacity was 10. For the 100 node problems, we generated separate problem instances with arc capacities of 5 and 10. Thus, we generated a total 1710 problem instances.

In order to provide a basis for comparison for our proposed memory-adjusting heuristic, in addition to coding our own heuristic, we obtained code for two popular heuristics for solving the CMST: Esau-Williams' Heuristic (Esau and Williams, 1966) and Gavish-Altinkemer's Parallel Savings Algorithm (PSA) Heuristic (Gavish & Altinkemer, 1986). Note that PSA is the original implementation by Gavish and Altinkemer (1986), rather than the improved version by Gouveia and Paixao (1991). The algorithms for these additional problems were coded in Fortran 77, and executed on a 133 MHz R4000 SGI Indigo. All problems were solved with unit demand data. Esau-Williams' heuristic procedure is extremely fast, but typically does not produce high-quality solutions compared to the more recently developed algorithms.

We solved each of the Euclidean, Random, and Mixed problem instances once (for each problem instance) using the EW, ART using only 10 life-cycle phases and a simpler memory loss function, and the PSA methods. In Table 3, we report the percentage difference between average solution values from the PSA to EW and ART for the Euclidean, Random, and Mixed graphs. From Table 3 it is clear that the ART heuristic on average provides better solutions than the PSA heuristic. The average improvement of ART over PSA is about 0.53% for Euclidean graphs, 11.58% for Random graphs, and 6.52% for mixed graphs. EW produces solutions that are on the average 1.47% worse as compared to those of PSA for Euclidean graphs, 13.98% worse for Random graphs, and 12.12% worse for Mixed graphs.

The improvements of ART over EW range from a minimum of about 1%, to a maximum of around 30%, with an average improvement of approximately 25% for the graphs with Random arc costs and 2.5% for the graphs with Euclidean arc costs. The improvements over EW due to the ART procedure for the random arc weights are approximately 25%! The ART procedure was able to overcome the poor performance of the embedded EW heuristic to significantly outperform the PSA. This demonstrates the ability of the ART procedure to significantly improve upon the performance of even the most simplistic of heuristics.

In Table 3 we also report the average computational times for ART and PSA. The EW procedure's CPU times are not reported, since this heuristic was found to have negligible CPU requirements (much less than a second for most problem instances). Note that the PSA heuristic is superior with respect to CPU needs to that of the ART, requiring approximately 1/10 of the ART procedure, depending upon the number of nodes in the problem. However, we must note that the ART procedure is far superior to the PSA in solution quality. While testing a version of the ART algorithm with 4 life-cycle phases (as opposed to 10) we found that we can reduce its CPU requirements by 60%. This reduction comes at a price of lower solution quality: the average improvement over PSA was now 0.26% and 8.8% for the Euclidean and Random graphs respectively using 4 life-cycle phases. The improvement of 8.8% (over PSA) for the random graphs is still a considerable enhancement.

In Table 4 we depict a brief summary of the minimum solution value found for EW and ART (compared to the minimum solution value found by PSA) for each of the data sets we generated. In light of this table, we see that the ART results are better than the average results reported in Table 3. For the Euclidean graphs the average minimum improvement of ART over PSA is 0.6%, while the improvement is 10.15% for the mixed problems, and 14.54% for the random problems.

The computational complexity of the ART procedure is fully dependent on the primary heuristic. Since the EW heuristic has a time complexity of $O(n^2 log_2 n)$, the complexity of ART is bounded by this measure as well. It is worth noting that the ART heuristic does not use any form of local search to find local optima. The reason for omitting local search is two-fold: first, the additional time complexity of most local search procedures (at least $O(n^2)$ for performing a complete neighborhood search for the CMST) causes the total complexity of the algorithm to increase significantly. Secondly, a local search procedure may not be easily generalizable across problem domains. Note from the results that the ART procedure seems remarkably good at finding the local optima on its own!

6. SUMMARY AND CONCLUSIONS

In this paper we have suggested a meta-heuristic based on memory adjustments and learning for solving the CMST problem. The contributions of our research are as follows: we have proposed a new memory-adjusting heuristic solution procedure that appears to be easily generalizable for solving many combinatorial optimization problems. We demonstrated the memory-adjusting principles in a solution procedure for the CMST problem, and concluded that our memory-adjusting procedure performs, on the average, better than most other current heuristics with respect to solution quality. The computational effort expended by the ART heuristic is modest.

In summary, there has been much research in the past decade into the solution of the CMST problem. We have provided a comprehensive comparison of our proposed method to other techniques in the same class of heuristic procedures. Solution times for several of these algorithms increase substantially with problem size (e.g. KAR, KBO, and

IPSA).

Additionally, we have demonstrated that the performance results using Euclidean–based costs for the CMST problem instances may differ from results using random-based costs. The ART method performs well compared to the EW and PSA methods regardless of the underlying cost structure.

We postulate that the ART technique derives its success from being able to make cuts in the problem space (the introduction of new constraints), and successfully learn about the effectiveness of these cuts and the combinations of various cuts. This type of cut-and-learn technique could be useful for solving a wide range of problems. In our current research, we are indeed exploring the use of ART to other problems that are structurally unrelated to the CMST, and we have seen evidence of success in applying our methodology to scheduling problems (Rolland et al, 1998).

We are indebted to Luis Gouveia for providing us with the bounds for the tc and te test problems and solution values for several solution methods to the tc and te test problems which were not available in the literature.

BIBLIOGRAPHY.

1. Altinkemer, K. and B. Gavish, 1988, Heuristics with Constant Error Guarantees for the Design of Tree Networks, *Management Science*, Vol. 34, No. 3, pp. 331-341.

2. Amberg, A., W. Domschke, and S. Voss, 1996, Capacitated minimum spanning trees: Algorithms using intelligent search, *Combinatorial Optimization: Theory and Practice*, Volume I, No. 1.

3. Esau, L.R. and K.C. Williams, 1966, On Teleprocessing Systems Design, Part II—A Method for approximating the optimal network, *IBM Systems Journal*, Vol. 5, No. 3, pp. 142-147.

4. Gavish, B., 1985, Augmented Lagrangian Based Algorithms for Centralized Network Design, *IEEE Trans. Communications*, Vol. 33, pp. 1247-1257.

5. Gavish, B., 1983, Formulations and Algorithms for the Capacitated Minimal Directed Tree Problem, *Journal of the Association of Computing Machinery*, Vol. 30, pp. 118-132.

6. Gavish, B., 1982, Topological design of centralized computer networks – Formulations and algorithms, *Networks*, Vol. 12, pp. 355-377.

7. Gavish, B. and K. Altinkemer, 1986, Parallel Saving Heuristics for the Topological Design of Local Access Tree Networks, *Proc. Of IEEE INFOCOM '86*, Fifth Annual Conference on Computers and Communications Integration Design, Analysis, Management, pp. 130-139.

8. Glover, F., 1997, Tabu Search and Adaptive Memory Programming – Advances, Applications and Challenges, in *Advances in Metaheuristics, Optimization, and Stochastic Modeling Technologies*, R.S. Barr, R.V. Helgason & J.L Kennington (eds.), Kluwer Academic Publisher, Boston, MA.

9. Gouveia, L., and J. Paixão, 1991, Dynamic Programming Based Heuristics for the Topological Design of Local Access Networks, *Annals of Operations Research*, Vol. 33, pp. 305-327.

10. Gouveia, L., 1993, A Comparison of Directed Formulations for the Capacitated Minimum Spanning Tree Problem, *Telecommunications Systems*, Vol. 1, pp. 51-76.

11. Gouveia, L., 1995, A 2*n* Constraint Formulation for the Capacitated Minimum Spanning Tree Problem, *Operations Research*, Vol. 43, pp. 130-141.

12. Gouveia, L., and P. Martins, 1995, An Extended Flow Based Formulation for the Capacitated Minimal Spanning Tree, presented at the Third ORSA Telecommunications Conference, Boca Raton, FL.

13. Hall, L., 1996, Experience with a Cutting Plane Algorithm for the Capacitated Spanning Tree Problem, *INFORMS Journal on Computing*, Vol. 8, No. 3, pp. 219-234.

14. Karnaugh, M., 1976, A new class of algorithms for multipoint network optimization, *IEEE Transactions on Communications*, Vol. 24, No. 5, pp. 500-505.

15. Kershenbaum, A. R. Boorstyn, and R. Oppenheim, 1980, Second-order greedy algorithms for centralized network design, *IEEE Transactions on Communications,* Vol. Com-22, No. 11, pp. 1835-1838.

16. Kershenbaum, A. and W. Chou, 1974, A Unified Algorithm for Designing Multidrop Teleprocessing Networks, *IEEE Transactions on Communications*, Vol. Com-22, No. 11 (November).

17. Papadimitriou, C.H., 1978, The Complexity of the Capacitated Tree Problem, *Networks,* Vol. 4, pp. 217-230.

18. E. Rolland, R. Patterson and B. Dodin, 1998, A Memory Adaptive Reasoning Technique for Solving the Audit Scheduling Problem, *Financial Accounting Abstracts Working Paper Series,* June 8. Working Paper No. WP1998-002, Center for Advanced Information and Telecommunication Technology Applications, School of Management, The University of Texas at Dallas, Richardson, TX.

19. Sharaiha, Y.M, M. Gendreau, G. Laporte, and I.H. Osman, 1997, A Tabu Search Algorithm for the Capacitated Shortest Spanning Tree Problem, *Networks*, Vol. 29, pp. 161-171.

Figure 1. The Memory Adaptive Reasoning Technique (ART)

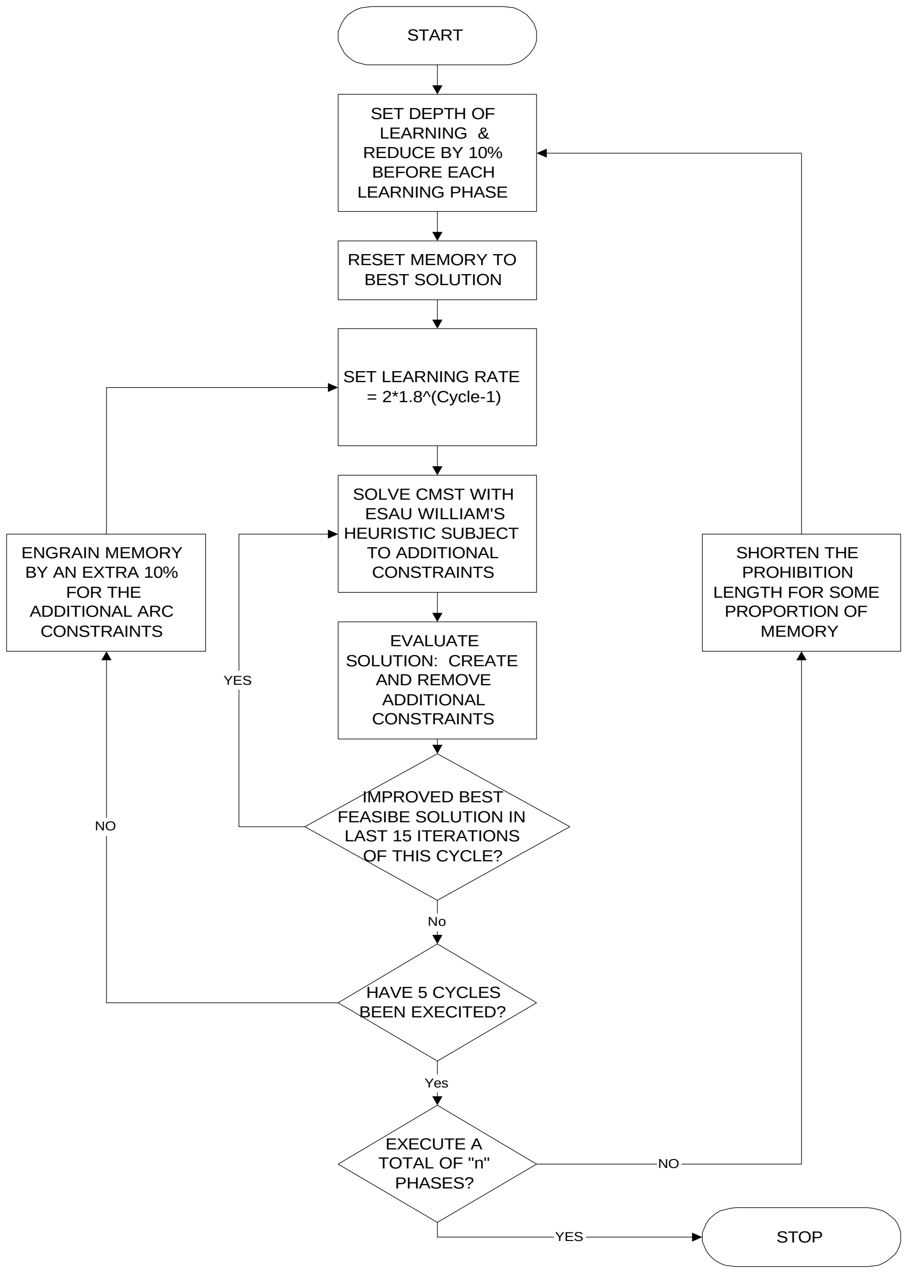

START
SET DEPTH OF LEARNING & REDUCE BY 10% BEFORE EACH LEARNING PHASE
RESET MEMORY TO BEST SOLUTION
SET LEARNING RATE = 2*1.8^(Cycle-1)
SOLVE CMST WITH ESAU WILLIAM'S HEURISTIC SUBJECT TO ADDITIONAL CONSTRAINTS
ENGRAIN MEMORY BY AN EXTRA 10% FOR THE ADDITIONAL ARC CONSTRAINTS
SHORTEN THE PROHIBITION LENGTH FOR SOME PROPORTION OF MEMORY
EVALUATE SOLUTION: CREATE AND REMOVE ADDITIONAL CONSTRAINTS
YES
IMPROVED BEST FEASIBE SOLUTION IN LAST 15 ITERATIONS OF THIS CYCLE?
NO
No
HAVE 5 CYCLES BEEN EXECITED?
Yes
EXECUTE A TOTAL OF "n" PHASES?
NO
YES
STOP

Figure 2. Evaluate Solution: Create and Remove Additional Arc Constraints

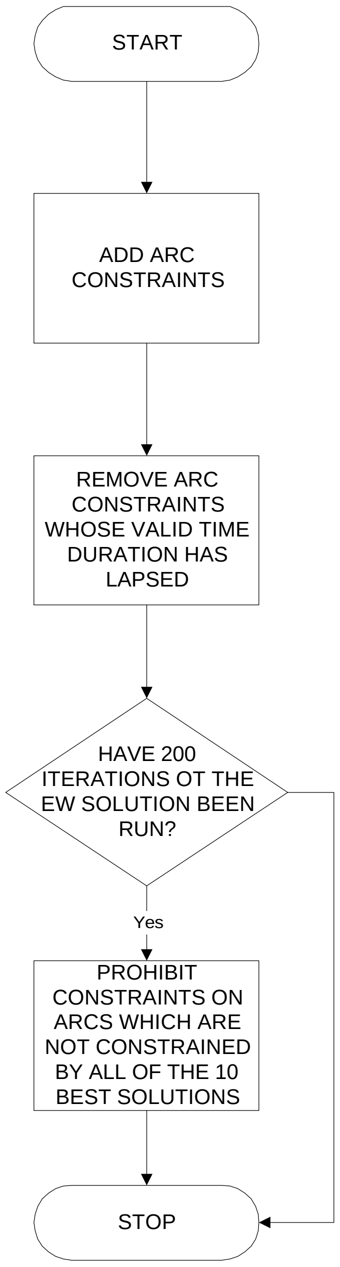

START
ADD ARC CONSTRAINTS
REMOVE ARC CONSTRAINTS WHOSE VALID TIME DURATION HAS LAPSED
HAVE 200 ITERATIONS OT THE EW SOLUTION BEEN RUN?
Yes
PROHIBIT CONSTRAINTS ON ARCS WHICH ARE NOT CONSTRAINED BY ALL OF THE 10 BEST SOLUTIONS
STOP

Figure 3. An Algorithm for Addition of a Constraint on an Arc

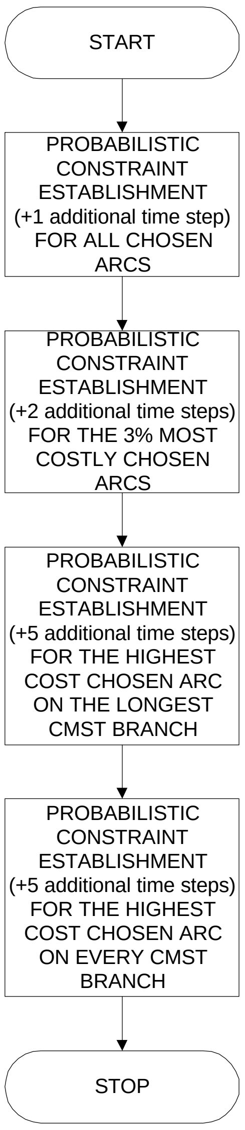
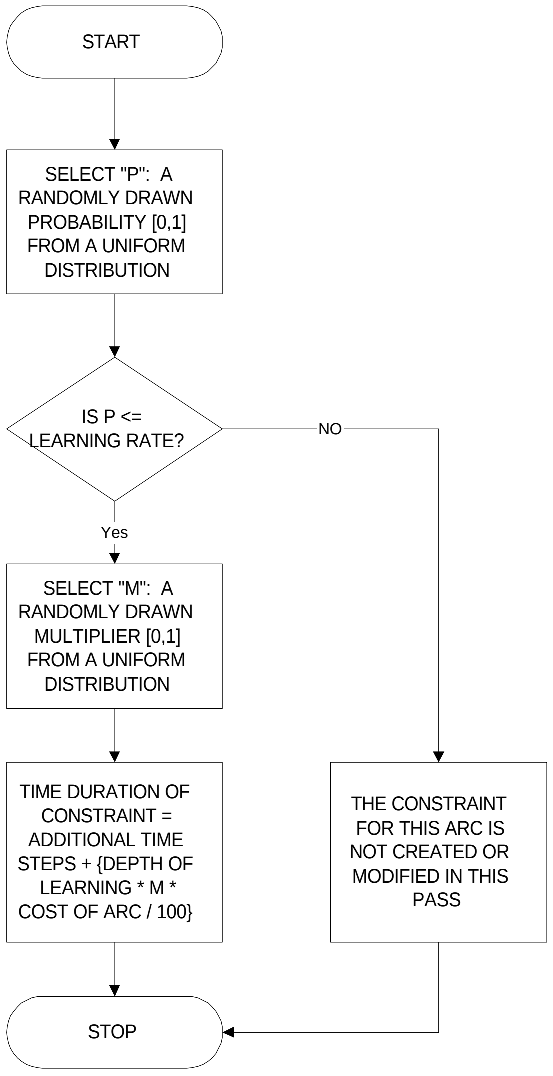

START
PROBABILISTIC CONSTRAINT ESTABLISHMENT (+1 additional time step) FOR ALL CHOSEN ARCS
PROBABILISTIC CONSTRAINT ESTABLISHMENT (+2 additional time steps) FOR THE 3% MOST COSTLY CHOSEN ARCS
PROBABILISTIC CONSTRAINT ESTABLISHMENT (+5 additional time steps) FOR THE HIGHEST COST CHOSEN ARC ON THE LONGEST CMST BRANCH
PROBABILISTIC CONSTRAINT ESTABLISHMENT (+5 additional time steps) FOR THE HIGHEST COST CHOSEN ARC ON EVERY CMST BRANCH
STOP

Figure 4. Probabilistic Constraint Establishment for a Chosen Arc


START
SELECT "P": A RANDOMLY DRAWN PROBABILITY [0,1] FROM A UNIFORM DISTRIBUTION
IS P <= LEARNING RATE?
NO
Yes
SELECT "M": A RANDOMLY DRAWN MULTIPLIER [0,1] FROM A UNIFORM DISTRIBUTION
TIME DURATION OF CONSTRAINT = ADDITIONAL TIME STEPS + {DEPTH OF LEARNING * M * COST OF ARC / 100}
THE CONSTRAINT FOR THIS ARC IS NOT CREATED OR MODIFIED IN THIS PASS
STOP

Problem		Previous solutions						ART		Lower	% Gap from lower bound						
ID	Capacity	EW	TS	AMB	KAR	KBO	IPSA	Solution	Time[1]	Bound	EW	TS	AMB	KAR	KBO	IPSA[2]	ART
tc40_1	3	774	744	n/a	n/a	n/a	n/a	742	11.25	742	4.31%	0.27%	n/a	n/a	n/a	n/a	0.00%
tc40_2	3	749	728	n/a	n/a	n/a	n/a	717	11.11	717	4.46%	1.53%	n/a	n/a	n/a	n/a	0.00%
tc40_3	3	728	722	n/a	n/a	n/a	n/a	716	11.06	716	1.68%	0.84%	n/a	n/a	n/a	n/a	0.00%
tc40_4	3	804	793	n/a	n/a	n/a	n/a	775	11.05	775	3.74%	2.32%	n/a	n/a	n/a	n/a	0.00%
tc40_5	3	760	741	n/a	n/a	n/a	n/a	741	11.1	741	2.56%	0.00%	n/a	n/a	n/a	n/a	0.00%
tc40_1	5	595	590	586	586	608	619	588	11.75	586	1.54%	0.68%	0.00%	0.00%	3.75%	5.63%	0.34%
tc40_2	5	588	585	578	583	603	608	583	11.69	578	1.73%	1.21%	0.00%	0.87%	4.33%	5.19%	0.87%
tc40_3	5	602	577	577	577	612	621	577	11.96	577	4.33%	0.00%	0.00%	0.00%	6.07%	7.63%	0.00%
tc40_4	5	645	618	617	617	566	650	617	11.87	617	4.54%	0.16%	0.00%	0.00%	-8.27%	5.35%	0.00%
tc40_5	5	615	602	600	605	613	630	605	11.95	600	2.50%	0.33%	0.00%	0.83%	2.17%	5.00%	0.83%
tc40_1	10	516	500	498	498	519	506	498	12.06	498	3.61%	0.40%	0.00%	0.00%	4.22%	1.61%	0.00%
tc40_2	10	505	490	490	490	511	508	490	12.12	490	3.06%	0.00%	0.00%	0.00%	4.29%	3.67%	0.00%
tc40_3	10	517	500	500	508	512	520	500	12.29	500	3.40%	0.00%	0.00%	1.60%	2.40%	4.00%	0.00%
tc40_4	10	524	513	512	512	513	528	512	11.98	512	2.34%	0.20%	0.00%	0.00%	0.20%	3.13%	0.00%
tc40_5	10	540	504	504	504	520	536	504	12.42	504	7.14%	0.00%	0.00%	0.00%	3.17%	6.35%	0.00%
Average for *tc40*:								*611.0*	*11.7*		*3.40%*	*0.53%*	*0.00%*	*0.33%*	*2.23%*	*4.75%*	*0.14%*
tc80_1	5	1182	1133	1099	1112	1159	1154	1120	85.24	1094	8.04%	3.56%	0.46%	1.65%	5.94%	5.48%	2.38%
tc80_2	5	1170	1124	1100	1115	1128	1147	1122	82.44	1090	7.34%	3.12%	0.92%	2.29%	3.49%	5.23%	2.94%
tc80_3	5	1131	1095	1073	1083	1110	1127	1080	80.12	1067	6.00%	2.62%	0.56%	1.50%	4.03%	5.62%	1.22%
tc80_4	5	1151	1108	1080	1098	1136	1137	1101	79.95	1070	7.57%	3.55%	0.93%	2.62%	6.17%	6.26%	2.90%
tc80_5	5	1338	1324	1287	1298	1325	1339	1293	97.12	1268	5.52%	4.42%	1.50%	2.37%	4.50%	5.60%	1.97%
tc80_1	10	920	901	888	907	920	910	896	86.45	878	4.78%	2.62%	1.14%	3.30%	4.78%	3.64%	2.05%
tc80_2	10	917	886	877	889	909	919	889	80.58	875	4.80%	1.26%	0.23%	1.60%	3.89%	5.03%	1.60%
tc80_3	10	916	880	880	892	904	914	888	80.62	869	5.41%	1.27%	1.27%	2.65%	4.03%	5.18%	2.19%
tc80_4	10	915	874	868	874	924	924	880	80.24	863	6.03%	1.27%	0.58%	1.27%	7.07%	7.07%	1.97%
tc80_5	10	1069	1005	1002	1020	1036	1084	1031	83.71	998	7.11%	0.70%	0.40%	2.20%	3.81%	8.62%	3.31%
tc80_1	20	856	834	834	838	858	852	842	84.2	834	2.64%	0.00%	0.00%	0.48%	2.88%	2.16%	0.96%
tc80_2	20	836	820	820	824	842	844	826	80.07	820	1.95%	0.00%	0.00%	0.49%	2.68%	2.93%	0.73%
tc80_3	20	856	828	828	832	832	842	832	80.78	828	3.38%	0.00%	0.00%	0.48%	0.48%	1.69%	0.48%
tc80_4	20	866	820	820	824	836	836	824	80.83	820	5.61%	0.00%	0.00%	0.49%	1.95%	1.95%	0.49%
tc80_5	20	971	916	916	922	955	958	937	91.66	916	6.00%	0.00%	0.00%	0.66%	4.26%	4.59%	2.29%
Average for *tc80*:									*83.6*		*5.48%*	*1.63%*	*0.53%*	*1.60%*	*4.00%*	*4.74%*	*1.83%*
Average for *tc*:									**47.7**		**4.44%**	**1.08%**	**0.32%**	**1.09%**	**3.29%**	**4.74%**	**0.98%**

1 Measured in seconds on a 266 MHz Pentium II computer running Windows 98
2 This is the improved PSA algorithm

Problem		**Previous solutions**						**ART**		**Lower**	**% Gap from lower bound**						
ID	**Capacity**	**EW**	**TS**	**AMB**	**KAR**	**KBO**	**IPSA**	**Solution**	**Time**[1]	**Bound**	**EW**	**TS**	**AMB**	**KAR**	**KBO**	**IPSA**[2]	**ART**
te40_1	3	1215	1192	n/a	n/a	n/a	n/a	1190	11.47	1190	2.10%	0.17%	n/a	n/a	n/a	n/a	0.00%
te40_2	3	1134	1117	n/a	n/a	n/a	n/a	1103	11.17	1103	2.81%	1.27%	n/a	n/a	n/a	n/a	0.00%
te40_3	3	1146	1115	n/a	n/a	n/a	n/a	1117	11.51	1115	2.78%	0.00%	n/a	n/a	n/a	n/a	0.18%
te40_4	3	1153	1144	n/a	n/a	n/a	n/a	1134	11.22	1132	1.86%	1.06%	n/a	n/a	n/a	n/a	0.18%
te40_5	3	1147	1115	n/a	n/a	n/a	n/a	1115	11.28	1104	3.89%	1.00%	n/a	n/a	n/a	n/a	1.00%
te40_1	5	857	875	830	847	859	905	835	12.66	830	3.25%	5.42%	0.00%	2.05%	3.49%	9.04%	0.60%
te40_2	5	839	812	792	796	822	880	794	12.02	792	5.93%	2.53%	0.00%	0.51%	3.79%	11.11%	0.25%
te40_3	5	820	822	797	797	838	881	797	12.54	797	2.89%	3.14%	0.00%	0.00%	5.14%	10.54%	0.00%
te40_4	5	854	835	814	820	853	893	815	12.21	814	4.91%	2.58%	0.00%	0.74%	4.79%	9.71%	0.12%
te40_5	5	816	796	784	789	848	842	797	12.08	784	4.08%	1.53%	0.00%	0.64%	8.16%	7.40%	1.66%
te40_1	10	658	614	596	602	636	653	608	14.35	596	10.40%	3.02%	0.00%	1.01%	6.71%	9.56%	2.01%
te40_2	10	632	591	573	577	611	645	573	12.41	573	10.30%	3.14%	0.00%	0.70%	6.63%	12.57%	0.00%
te40_3	10	596	591	568	572	592	652	572	12.33	568	4.93%	4.05%	0.00%	0.70%	4.23%	14.79%	0.70%
te40_4	10	638	608	596	598	609	676	596	12.5	596	7.05%	2.01%	0.00%	0.34%	2.18%	13.42%	0.00%
te40_5	10	597	572	572	574	612	647	572	12.44	572	4.37%	0.00%	0.00%	0.35%	6.99%	13.11%	0.00%
Average for *te40*:									12.1		4.77%	2.06%	0.00%	0.70%	5.21%	11.12%	0.45%
te80_1	5	2604	2570	2544	2559	2577	2671	2555	81.2	2531	2.88%	1.54%	0.51%	1.11%	1.82%	5.53%	0.95%
te80_2	5	2633	2574	2551	2571	2586	2711	2564	82.64	2522	4.40%	2.06%	1.15%	1.94%	2.54%	7.49%	1.67%
te80_3	5	2723	2741	2612	2675	2640	2694	2644	82.75	2593	5.01%	5.71%	0.73%	3.16%	1.81%	3.90%	1.97%
te80_4	5	2624	2672	2558	2612	2612	2637	2574	82.11	2539	3.35%	5.24%	0.75%	2.88%	2.88%	3.86%	1.38%
te80_5	5	2593	2557	2469	2553	2489	2609	2475	82.59	2458	5.49%	4.03%	0.45%	3.86%	1.26%	6.14%	0.69%
te80_1	10	1746	1688	1631	1658	1708	1711	1672	84.12	1631	7.05%	3.49%	0.00%	1.66%	4.72%	4.90%	2.51%
te80_2	10	1748	1678	1643	1685	1713	1746	1659	85.5	1602	9.11%	4.74%	2.56%	5.18%	6.93%	8.99%	3.56%
te80_3	10	1828	1775	1688	1743	1736	1772	1719	82.33	1660	10.12%	6.93%	1.69%	5.00%	4.58%	6.75%	3.55%
te80_4	10	1685	1906	1629	1708	1732	1760	1655	83.17	1614	4.40%	18.09%	0.93%	5.82%	7.31%	9.05%	2.54%
te80_5	10	1712	1685	1603	1663	1660	1684	1631	88.39	1586	7.94%	6.24%	1.07%	4.85%	4.67%	6.18%	2.84%
te80_1	20	1330	1311	1275	1285	1301	1363	1293	83.4	1256	5.89%	4.38%	1.51%	2.31%	3.58%	8.52%	2.95%
te80_2	20	1289	1266	1225	1238	1317	1339	1248	83.58	1201	7.33%	5.41%	2.00%	3.08%	9.66%	11.49%	3.91%
te80_3	20	1340	1329	1267	1297	1344	1413	1280	87.16	1257	6.60%	5.73%	0.80%	3.18%	6.92%	12.41%	1.83%
te80_4	20	1343	1337	1265	1320	1370	1403	1289	84.68	1247	7.70%	7.22%	1.44%	5.85%	9.86%	12.51%	3.37%
te80_5	20	1334	1259	1240	1251	1336	1340	1245	87.13	1231	8.37%	2.27%	0.73%	1.62%	8.53%	8.85%	1.14%
Average for *te80*:									*84.05*		*6.38%*	*5.54%*	*1.09%*	*3.43%*	*5.14%*	*7.77%*	*2.32%*
Average for *te*:									**48.1**		**5.57%**	**3.80%**	**0.65%**	**2.34%**	**5.17%**	**9.11%**	**1.39%**
Grand average for *te* and *tc* problems:									**55.2**		**5.01%**	**2.44%**	**0.49%**	**1.72%**	**4.23%**	**6.93%**	**1.18%**

1 Measured in seconds on a 266 MHz Pentium II computer running Windows 98

2 This is the improved PSA algorithm

		Solution Values						CPU Times					
		Euclidean		Mixed		Random		Euclidean		Mixed		Random	
NODES	CAPACITY	EW	ART	EW	ART	EW	ART	ART	PSA	ART	PSA	ART	PSA
50	4	0.91%	-0.40%	11.56%	-5.66%	11.19%	-9.04%	66.06	3.83	66.27	4.39	66.14	4.46
100	5	1.20%	-0.21%	15.66%	-6.46%	18.57%	-10.83%	462.57	34.01	471.13	41.47	471.68	42.21
100	10	1.73%	-1.23%	12.44%	-6.31%	14.65%	-11.79%	492.32	35.92	485.91	43.75	486.09	44.06
150	10	2.04%	-0.27%	8.81%	-7.65%	11.50%	-14.67%	1504.60	126.99	1483.45	154.12	1482.20	154.72
Average:		1.47%	-0.53%	12.12%	-6.52%	13.98%	-11.58%	631.39	50.19	626.69	60.93	626.53	61.36

These are the average percentages of the solution value for the average of each dataset above (below) PSA

		Euclidean		Mixed		Random	
NODES	**CAPACITY**	**EW**	**ART**	**EW**	**ART**	**EW**	**ART**
50	4	0.92%	-0.78%	11.01%	-8.70%	10.85%	-12.44%
100	5	1.01%	-0.69%	18.00%	-8.79%	21.83%	-12.22%
100	10	3.25%	-0.25%	12.67%	-10.05%	16.16%	-15.13%
150	10	2.41%	-0.68%	8.76%	-13.04%	14.35%	-18.35%
Average:		1.90%	-0.60%	12.61%	-10.15%	15.80%	-14.54%

Solutions reported are the average percentages of the minimum solution value over all graph densities for each dataset above (below) PSA