A Lagrangian Based Approach to the Asymmetric Generalized Traveling Salesman Problem

Charles E. Noon
Department of Management Science
The University of Tennessee
Knoxville, TN 37996

James C. Bean
Department of Industrial and Operations Engineering
The University of Michigan
Ann Arbor, MI 48109-2117

Technical Report 89-13

April 1989

A Lagrangian Based Approach to the Asymmetric Generalized Traveling Salesman Problem

Charles E. Noon
Department of Management Science
The University of Tennessee, Knoxville 37996.

James C. Bean
Department of Industrial and Operations Engineering
The University of Michigan, Ann Arbor 48109.

September 23, 1989

ABSTRACT

This paper presents an optimal approach for the Asymmetric Generalized Traveling Salesman Problem (GTSP). The GTSP is defined on a directed graph in which the nodes are grouped into m predefined, mutually exclusive and exhaustive sets with the arcset containing no intraset arcs. The problem is to find a minimum cost m-arc directed cycle which includes exactly one node from each set. Our approach employs a Lagrangian relaxation to compute a lower bound on the total cost of an optimal solution. The lower bound and a heuristically determined upper bound are used to identify and remove arcs and nodes which are guaranteed not to be in an optimal solution. Finally, we use an efficient branch-and-bound procedure which exploits the multiple choice structure of the node sets. We present computational results for the optimal approach tested on a series of randomly generated problems. The results show success on a range of problems with up to 104 nodes.

1.0 Introduction and Mathematical Formulation

The Traveling Salesman Problem (TSP) is one of the oldest and most widely studied optimization problems in the field of operations research. The basic problem is to find a minimum cost Hamiltonian cycle, or *tour*, over the nodes of a graph (see Lawler, et al. [1985], Parker and Rardin [1983]). The TSP model assumes that the decision maker has determined a priori which nodes will be sequenced, i.e., which cities will be visited. This paper considers a generalization of the TSP which combines the decisions of node selection and node sequencing. Instead of preselecting the nodes to be visited, the generalized model assumes the nodes have been grouped into mutually exclusive and exhaustive node *sets*. The *Generalized Traveling Salesman Problem* (GTSP) is then to find a minimum cost cycle which includes exactly one node from each node set. It is *generalized* since a TSP is a special case with node sets of cardinality one.

The Generalized Traveling Salesman Problem allows node alternatives to be considered in the decision process. The first application of a GTSP was presented by Henry-Labordere [1969] for sequencing computer files. About the same time, Saksena [1970] modeled the routing of welfare clients through governmental agencies as a symmetric GTSP. Potential applications can be found in Noon [1988] with respect to warehouse order picking with multiple stock locations, airport selection and routing for courier planes, and certain types of flexible manufacturing scheduling.

An interesting GTSP application in the area of postal routing has been discussed in Laporte, Mercure and Nobert [1987] and Rousseau [1988]. In this application, a postal van must be routed to make pickups from a number of urban mail boxes. The problem is complicated by the fact that the postal van driver is prohibited from crossing a street on foot in order to make a pickup. This means that for a mail box situated at a street corner, the pickup can only be made from the van stopped in one of the two lanes immediately adjacent to the corner. Although these two potential stop locations are physically close, they result in the van being pointed in two very different directions, thereby yielding different distances to the next mailbox pickup. The problem is modeled as a GTSP by creating a node set for each mailbox to be picked up. Within each node set, a node is created for each of the two potential pickup locations.

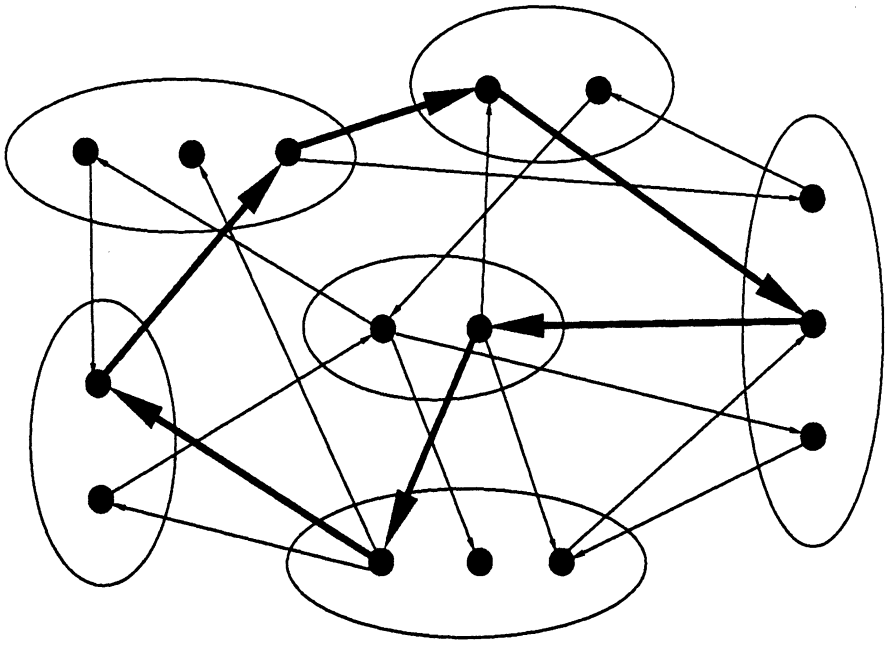

Figure 1: Example GTSP on a digraph (with feasible tour in bold).

The asymmetric GTSP is defined on a directed graph $\mathcal{D}$ with nodes $\mathcal{N}$ and connecting arcs $\mathcal{A}$. Let $\mathcal{N}$ be the union of m mutually exclusive and exhaustive node sets, $\mathcal{N} = S_1 \cup S_2 \cup \cdots \cup S_m$ and $S_I \cap S_J = \emptyset$, for all I, J, $I \neq J$. Assume that arcs are defined only between nodes belonging to different sets, that is, there are no intraset arcs. Each arc $(i, j) \in \mathcal{A}$ has a corresponding nonnegative cost c_{ij}.

The GTSP can be stated as *the problem of finding a minimum cost m-arc cycle which includes exactly one node from each node set.* Figure 1 displays an example GTSP defined on a digraph. The bold lines illustrate a feasible GTSP tour for the problem.

The GTSP simultaneously selects the nodes to include in the cycle and sequences the nodes along the cycle. In the problem form presented, given a preselection of nodes or a presequence of node sets, the remaining problem is well known. Given a preselection of nodes, the remaining problem is a TSP. Given a presequence of node sets, we could determine the nodes to pass through by solving a series of acyclic shortest path problems. These two characteristics will later play an important role in our optimal approach.

Although our problem form assumes considerable structure, we are not narrowly focusing

on a special case of the GTSP. It can be shown that a class of problems with overlapping node sets, intraset arcs, and sets which may be visited more than once, can be efficiently transformed to a problem in the form presented (Noon and Bean [1989]).

We can model the asymmetric GTSP as an integer program (P) with variable x_{ij} equal to 1 if arc (i,j) is used in a solution tour and 0 if it is not. Problem (P) is displayed as follows.

$$\text{Minimize} \sum_{(i,j)\in\mathcal{A}} c_{ij}x_{ij}$$

Subject to,

$$\left.\begin{aligned} \sum_{i\in S_I}\sum_{\substack{j\notin S_I \\ (i,j)\in\mathcal{A}}} x_{ij} &= 1 \\ \sum_{i\notin S_I}\sum_{\substack{j\in S_I \\ (i,j)\in\mathcal{A}}} x_{ij} &= 1 \end{aligned}\right\} \quad \text{for all sets, } S_I. \qquad (i)$$

$$\sum_{\substack{i\in\mathcal{N} \\ (i,j)\in\mathcal{A}}} x_{ij} - \sum_{\substack{k\in\mathcal{N} \\ (j,k)\in\mathcal{A}}} x_{jk} = 0 \quad \text{for all nodes } j \in \mathcal{N}. \qquad (ii)$$

$$\sum_{I\in T}\sum_{i\in S_I}\sum_{J\notin T}\sum_{\substack{j\in S_J \\ (i,j)\in\mathcal{A}}} x_{ij} \geq 1 \quad \text{for all sets } T,\ 2 \leq |T| \leq m-2, \text{ which are proper subsets of the collection of node sets.} \qquad (iii)$$

$$x_{ij} = 0 \text{ or } 1 \quad \text{for all } (i,j) \in \mathcal{A}. \qquad (iv)$$

Constraint group (i) contains two constraints for each node set. One constraint requires a solution to include exactly one of the arcs entering a set and the other requires a solution to include exactly one of the arcs leaving a set. Group (ii) contains one constraint for each node. They are equivalent to network flow conservation constraints and ensure that a solution tour is uninterrupted and continuous. Constraint group (iii) is needed to prevent subtours from occurring and may contain as many as $2^{m-1} - m - 1$ constraints. Constraint groups (i) and (iii) are generalizations of the assignment and subtour prevention constraints commonly used in formulations for the asymmetric TSP.

Early approaches for the GTSP were presented by Henry-Labordere [1969], Srivastava, et

al. [1969] and Saksena [1970]. These approaches are similar to the dynamic programming approach for the TSP (Gonzaléz [1962]) in which a state is defined by the sets already visited. One drawback to these types of approaches is that the number of states grows rapidly as the number of node sets increases. More recently, Laporte and Nobert [1983] proposed an integer programming approach for the symmetric GTSP which was successful on problems with up 50 nodes.

For the asymmetric GTSP, relaxation-based approaches are presented in two papers by Laporte, Mercure and Nobert [1985,1987]. In the first paper, the subtour prevention and node flow conservation constraints are relaxed with the latter being brought into the objective function. The relaxed problem is solved as a network flow problem and serves as a lower bound. In the second paper, the authors relax the subtour prevention constraints and a set of constraints which ensure that each set is visited. The relaxed problem is solved as an $|\mathcal{N}| \times |\mathcal{N}|$ assignment problem. In both approaches, the relaxations are used in a branch-and-bound algorithm which branches on the arcs of subtours or on nodes of unvisited sets. The later approach proved to be the most successful and was able to solve 100 node problems using several hundred cpu seconds on a mainframe computer.

2.0 An Optimal Approach

The optimal approach we present is a sequence of three separate procedures; problem bounding, arc/node elimination, and branch-and-bound enumeration. The procedures exploit the problem structure realized by mutually exclusive node sets, no intraset arcs, and the requirement of exactly one visit to each set. The first procedure uses a relaxation to efficiently compute lower bounds on the optimal objective value of the problem. The procedure also includes a heuristic for determining upper bounds. These bounds are then used in the second procedure to identify and remove many nonoptimal arcs and nodes. With the problem well-bounded and reduced in size, the third procedure uses implicit enumeration to guarantee an optimal solution.

2.1 Bounds for the GTSP

The first procedure employs a Lagrangian relaxation for bounding. The constraints in group (ii) are relaxed and brought into the problem's objective function via a vector of multipliers, λ. The relaxed problem reduces to that of finding a minimum cost tour over the node sets, without requiring the tour to enter and leave a set through the same node. We can show that the relaxed problem can be solved as an m-city TSP or closely bounded with an $m \times m$ assignment problem. An advantage of this approach is that the size of the relaxed problem depends only on the number of node sets.

Let λ be a vector of length $|\mathcal{N}|$ where λ_i is the multiplier applied to the node i constraint of group (ii). A nonzero λ_i value has the effect of subtracting that value from the costs of all in-arcs of node i and adding the value to costs of all out-arcs of node i. This does not change the order or cost of GTSP solutions, it merely redistributes the costs of the problem. We can define the following Lagrangian relaxation (PR_λ) as,

$$\begin{aligned} &\textbf{Minimize} \sum_{(i,j)\in\mathcal{A}} (c_{ij} - \lambda_i + \lambda_j)x_{ij} && (\mathbf{PR}_\lambda) \\ &\text{Subject to,} \qquad (i), (iii), (iv). \end{aligned}$$

For any problem $(\cdot)$, let $\mathcal{V}(\cdot)$ equal the problem's optimal objective value. For any given λ, $\mathcal{V}(PR_\lambda)$ is a lower bound on $\mathcal{V}(P)$. Our relaxation has the advantage that in solving (PR_λ), we need consider only the minimum cost arcs between node sets. For a given λ, let c^λ_{ij} be the adjusted arc costs, i.e., $c^\lambda_{ij} = c_{ij} - \lambda_i + \lambda_j$.

Lemma 1 : *There exists no optimal solution to* (PR_λ) *in which* $x_{ij} = 1, i \in S_I, j \in S_J$, *and* $c^\lambda_{ij} > c^\lambda_{kl}, k \in S_I, l \in S_J$,.

Proof : Let x^* be an optimal solution to (PR_λ) with element $x^*_{ij} = 1$ and $c^\lambda_{ij} > c^\lambda_{kl}$, with $i, k \in S_I$ and $j, l \in S_J$. We could reduce the cost of the tour x^* by replacing arc (i, j) with arc (k, l). The constraints of (PR_λ) require each set to be visited and prevent node set subtours. The solution obtained by replacing arc (i, j) with arc (k, l) would remain feasible since both arcs connect the same pair of sets. The new solution would be feasible with a lower cost. Hence, x^* could not be optimal. ■

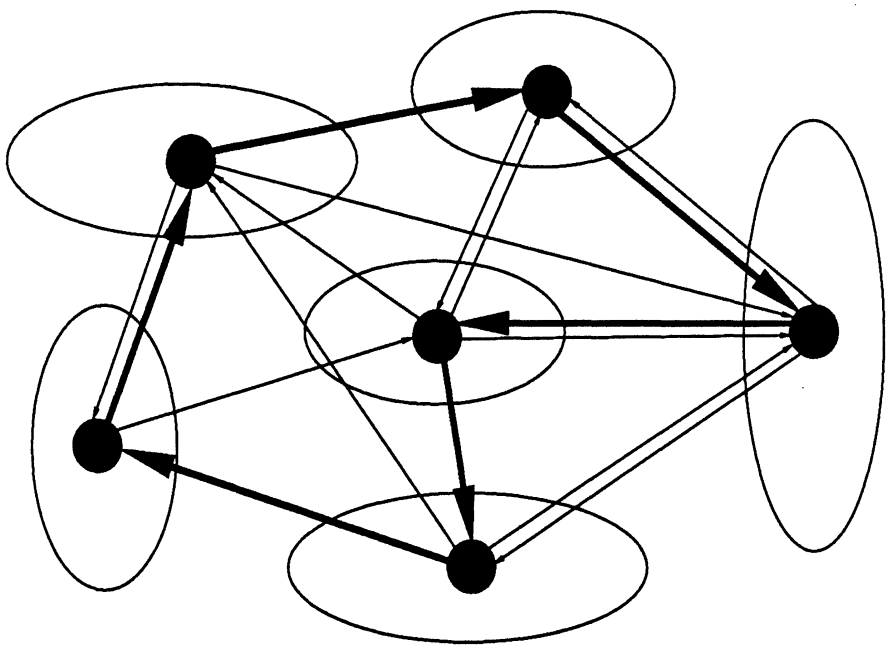

Figure 2: Aggregation of Figure 1 digraph (with translated feasible tour in bold).

In solving (PR_λ), Lemma 1 allows us to eliminate all arcs which are not arguments of the minimum costs between node sets. We can now define an aggregation of the original digraph based on the interset arc cost minimums.

Definition : Digraph $\mathcal{D}' = (\mathcal{N}', \mathcal{A}')$ is an aggregation of the original digraph $\mathcal{D}$. Let $\mathcal{N}'$ consist of m nodes where each node corresponds to a node set of $\mathcal{N}$. Given two nodes $I, J \in \mathcal{N}'$, an arc $(I, J) \in \mathcal{A}'$ will exist if and only if there is at least one arc $(i, j) \in \mathcal{A}$ with $i \in S_I$ and $j \in S_J$. Let the cost of arc $(I, J) \in \mathcal{A}'$ be defined as c'_{IJ} where $c'_{IJ} = \underset{i \in S_I, j \in S_J}{\text{minimum}} \{c^\lambda_{ij}\}$.

Lemma 2 : *If x is a feasible solution to (PR_λ) over $\mathcal{D}$, then there exists a feasible TSP tour y over $\mathcal{D}'$ with $c'y \leq c^\lambda x$.*

Proof : This follows from the construction of $\mathcal{D}'$. ■

From Lemma 2, we can translate a feasible set-tour over $\mathcal{D}$ to a feasible TSP over $\mathcal{D}'$. Figure 2 displays the aggregation of the example digraph given in Figure 1. The translated feasible

tour is given in bold. We may now define a mapping that allows us to translate a feasible TSP tour over $\mathcal{D}'$ to a feasible solution for (PR_λ) over $\mathcal{D}$.

Definition : Let y be a TSP tour over $\mathcal{D}'$. Let the *mapping* $\mathbf{f}$ be defined as $[\,\mathbf{f} : Y \rightarrow X]$, where Y is a subset of all $0-1$ vectors of order $|\mathcal{N}'|^2$ and X is all $0-1$ vectors of order $|\mathcal{N}|^2$. Given a solution y, $\mathbf{f}$ maps y to a solution x as follows:

- For $y_{IJ} = 1$, choose (arbitrarily) exactly one x_{ij}, $i \in S_I$, $j \in S_J$, such that $c'_{IJ} = c^\lambda_{ij}$ and set that $x_{ij} = 1$ and all other x_{kl}, $k \in S_I$, $l \in S_J$, equal to 0.
- For $y_{IJ} = 0$, set $x_{ij} = 0$ over all $i \in S_I, j \in S_J$.

From the construction of $\mathcal{D}'$ and the mapping $\mathbf{f}$, it follows that if y is feasible for the m-city TSP, then $x = \mathbf{f}(y)$ is feasible for (PR_λ) and the solutions have equal objective values. In fact, (PR_λ) is feasible if and only if the m-city TSP is feasible.

Theorem 3 : *For a given λ, an optimal solution to (PR_λ) can be obtained by solving an m-city TSP over the digraph $\mathcal{D}'$.*

Proof : Let y^* be an optimal solution to the TSP defined over $\mathcal{D}'$. We can map y^* to a solution, x^*, which is feasible for (PR_λ) and has objective $c'y^* = c^\lambda x^*$. If x^* is not optimal for (PR_λ), then there exists a feasible x^1 such that $c^\lambda x^1 < c^\lambda x^*$. From Lemma 2, there must exist a feasible TSP tour y^1 over $\mathcal{D}'$ with $c'y^1 \leq c^\lambda x^1$. This implies $c'y^1 \leq c^\lambda x^1 < c^\lambda x^* = c'y^*$ and contradicts the assumption of optimality on y^*. Hence, no such x^1 exists and x^* must be optimal for (PR_λ). ■

Theorem 3 allows us to solve (PR_λ) as an m-city TSP to determine lower bounds and, under certain conditions, optimal solutions for (P). Although the TSP is considerably smaller than (P), it is nevertheless an NP-hard problem and may be difficult to solve. Let (APR_λ) be the problem obtained by dropping the subtour elimination constraints of (PR_λ).

$$\textbf{Minimize} \sum_{(i,j)\in\mathcal{A}} (c_{ij} - \lambda_i + \lambda_j)x_{ij} \qquad (\mathbf{APR}_\lambda)$$

Subject to, $(i), (iv)$.

Since (APR_λ) is a further relaxation of (PR_λ), $\mathcal{V}(APR_\lambda)$ is a lower bound on $\mathcal{V}(PR_\lambda)$. From the usual relationship between the TSP and the assignment problem, Corollary 4 easily follows.

Corollary 4 : *For a given λ, an optimal solution to (APR_λ) can be obtained by solving an $m \times m$ assignment problem over the nodes of $\mathcal{D}'$.*

The problems are ordered as below.

$\mathcal{V}(APR_\lambda)$	$\leq$	$\mathcal{V}(PR_\lambda)$	$\leq$	$\mathcal{V}(P)$
$\{m \times m$ Assignment Problem$\}$		$\{$ m-city TSP$\}$		{Full GTSP}

The relaxation (APR_λ) allows us to obtain a fast lower bound on our original problem (P). The strength of the bound depends greatly on the vector λ. Ideally, we would like to find a vector of Lagrange multipliers such that $\mathcal{V}(APR_\lambda)$ is maximized. Since (APR_λ) has the *integrality property* (see Geoffrion [1974]), we know that $\mathcal{V}(APR_\lambda)$ is maximized by setting λ equal to the optimal dual values of a linear programming relaxation of (P) excluding constraint group (iii). Rather than optimally solve such a large LP using a direct method, we approximate the λ vector using an iterative subgradient algorithm (see Fisher [1981]). The subgradient approach is easier to code and, relative to direct methods, uses little storage.

Our bounding procedure begins with a nearest neighbor heuristic generated from each node. The heuristic is identical to the well known nearest neighbor heuristic for the TSP except that a node becomes ineligible once its node set has been visited. The heuristic provides the feasible incumbent (upper bound) as needed for the subgradient algorithm. At each step of the subgradient algorithm, (APR_λ) is solved as an assignment problem and its solution is used to compute a subgradient direction. Throughout the algorithm, if an assignment problem solution is also a feasible TSP tour, we use it to check for a better incumbent. A TSP tour over the aggregated digraph represents a sequence of node sets over the original digraph. Given a sequence of node sets, we can find a local optimum by solving acyclic shortest path problems over the sequence of node sets. This is often successful in finding better incumbent solutions and, in some cases, optimal solutions.

The subgradient algorithm terminates when either the lower bound equals the upper bound or the rate of improvement nears zero. The final output of the bounding procedure consists of a final vector of Lagrange multipliers $\hat{\lambda}$, a lower bound $\mathcal{V}(APR_{\hat{\lambda}})$, and a feasible incumbent with upper bound value Z.

2.2 Arc/Node Elimination

After the bounding procedure, we use arc and node elimination tests to reduce the size of the problem. The tests are performed using the *reduced* (or *relative*) arc costs for $(APR_{\hat{\lambda}})$. The reduced cost of an arc represents the minimum amount the objective value will increase over $\mathcal{V}(APR_{\hat{\lambda}})$ if the arc is used in a solution. Given the output $\hat{\lambda}$ vector from the bounding procedure, we can solve for the optimal dual solution of $(APR_{\hat{\lambda}})$. For node set S_I, let u_I, v_I be the optimal dual values of the constraints of (i) corresponding to S_I. In computing the reduced costs, u_I is subtracted from the costs of all arcs entering S_I, and v_I is subtracted from the costs of all arcs leaving S_I. For an arc (i,j), $i \in S_I, j \in S_J$, its reduced cost is given as $\bar{c}_{ij} = c_{ij} - \hat{\lambda}_i + \hat{\lambda}_j - u_I - v_J$ and, by dual feasibility, $\bar{c}_{ij} \geq 0$. These computations do not change the underlying problem, they merely remove costs that will necessarily be incurred in a problem. This allows us to work with the problem defined over the reduced costs. The cost of the current incumbent solution defined over the reduced costs is $[Z - \mathcal{V}(APR_{\hat{\lambda}})]$, where Z is the upper bound from the subgradient optimization.

One classic variable elimination test for integer programming was first discussed by Dakin [1965]. It was noted that a variable could be eliminated if its reduced cost exceeded the reduced cost of the incumbent solution. This simple yet powerful test is valid since any solution with the variable equal to one would have an objective value greater than the known feasible incumbent.

For the GTSP, the preceding variable elimination can be extended. Let the *shortest reduced cost path* be defined as the shortest directed path from one node to another over the reduced arc costs. Let $SP(i,j)$ be the total cost of the shortest reduced cost path from node i to node j.

Theorem 5 : *If* $[\bar{c}_{ij} + SP(j,i)] \geq [Z - \mathcal{V}(APR_{\hat{\lambda}})]$ *then the variable* x_{ij} *and its corresponding arc cannot be included in a feasible solution which is better than the current incumbent and can be eliminated from the problem.*

Proof : For a given arc (i,j), its reduced cost is separable, and hence, only represents the additional cost incurred by traveling from the arc's tail (node i) to its head (node j). Yet if the arc is to be used in a feasible solution it must be connected to a tour. Any tour which includes arc (i,j) also includes a "return path" from j to i. The length of a tour's "return path" must be *at least* as long as the shortest "return path" from j to i. Hence, $\bar{c}_{ij} + SP(j,i)$ is a lower bound on the reduced cost of any tour which includes arc (i,j). If this lower bound exceeds the reduced cost of the current incumbent, then arc (i,j) is guaranteed not to be in a feasible solution which is better than the incumbent and can be eliminated. ■

The arc elimination test can remove a considerable number of arcs from a problem. In some cases all arcs incident to a node might be eliminated, thus allowing the node to be eliminated. Typically, that is not the case and the eliminations merely result in a less connected digraph. The following node elimination test checks for nodes which are not fully connected to all other sets.

Theorem 6 : *For a given node i and any set S_K such that $i \notin S_K$, if* $\underset{j \in S_K}{Minimum} [SP(i,j) + SP(j,i)] \geq [Z - V(APR_{\hat{\lambda}})]$, *then node i cannot be included in a feasible solution which is better than the current incumbent and can be eliminated from the problem.*

Proof : For node i to remain under consideration, there must exist a round trip path between i and *at least* one node in S_K with a total path cost less than the current incumbent reduced cost. A lower bound on the round trip cost between some node $i \in S_I$ and a node $j \in S_K$ is equal to $SP(i,j) + SP(j,i)$. Otherwise, node i can be eliminated since there can exist no GTSP tour which visits node i and a member of S_K and has a cost less than the incumbent. ■

We begin the elimination procedure by calculating the reduced arc costs. The optimal dual values needed for the calculations can be obtained by solving the assignment problem associated with $(APR_{\hat{\lambda}})$. The dual solutions of the two problems are equivalent. Once the

reduced costs are computed, we apply the nearest neighbor heuristic. In our experiments, the heuristic applied over the reduced costs often produced an improved incumbent.

Next, an induction algorithm is employed for computing the shortest reduced cost path between all pairs of nodes. The algorithm requires $O(n^3)$ steps, where n is the total number of nodes. Note that since the shortest path computations are all performed on the nonnegative reduced costs, there is no chance for a negative cycle to appear.

The procedure then iterates between the elimination tests and the nearest neighbor heuristic. The arc elimination test checks each arc one at a time for elimination. The node elimination test is applied on each node and involves checking every other node set for round trip connectivity. After the tests are performed, the nearest neighbor heuristic is applied to the reduced costs of the remaining arcs. If a better incumbent is found, the elimination tests are repeated as well as the heuristic. In many cases, the eliminations allowed for better incumbents to be found. The procedure terminates when the heuristic fails to produce a better incumbent. Theoretically, the procedure could iterate as many times as there are arcs, however, in practice, the elimination procedure iterated at most three times.

2.3 Enumeration Procedure

If the upper bound is equal to the lower bound then the upper bound solution is optimal. If it is not, we must use an enumeration procedure to divide the feasible region and optimize over the components. One strategy for GTSP enumeration would be as follows. First, list all possible selections of exactly one node from each node set. For each of these combinations, optimally solve an m-city TSP over the selected nodes. The optimal TSP tour with the least cost among all combinations is the optimal tour for the GTSP. This methodology provides the framework for our approach. But rather than explicitly enumerate and list all node selection combinations, we use branch-and-bound to implicitly enumerate them.

The node sets possess a "multiple choice" structure since a feasible solution must include exactly one node from each set. We take advantage of this structure by using a branching scheme similar to the approach found in Bean [1984] for multiple choice variable sets. The basis of our branching strategy is that if we select a node to be *included* in the solution, then we are automatically fixing the other nodes of its set to be *excluded.* This allows us to divide

any problem into separable subproblems by choosing a node set and branching on each node of the set.

Any subproblem can be divided further by choosing an unbranched set and branching on each node of the set. If we branch down m levels, we reach subproblems at the bottom of the enumeration tree. A bottom-of-tree subproblem will have one node selected from each node set. Let us refer to these subproblems as *full combination* subproblems. Any subproblem at a higher level of the tree will have fewer than m nodes selected. Let us refer to these subproblems as *partial combination* subproblems.

In the spirit of branch-and-bound, we try to avoid explicit enumeration by eliminating (or *fathoming*) partial combination subproblems at the higher levels of the tree. For a given partial combination subproblem, we first check to see if the node most recently selected for inclusion in the subproblem can be eliminated. This is done by calculating the round trip costs between this node and every other *selected* node of the subproblem. If any one of these round trip costs exceeds the current incumbent, then we know that the subproblem cannot yield a better feasible incumbent and can be fathomed.

If a partial combination subproblem cannot be fathomed using the round trip test, we solve a relaxed problem to get a lower bound. The mathematical formulation of any partial combination subproblem is identical to the original GTSP problem except for the nodes that are automatically excluded by the selected nodes. Therefore, for the given subproblem, if we delete the arcs incident to the excluded nodes, we can use the Lagrangian relaxation $(APR_{\hat{\lambda}})$ to compute a lower bound on the subproblem. This is done by simply adjusting the aggregated arc costs c' corresponding to the node sets of the selected nodes and solving the resulting $m \times m$ assignment problem. The subproblem can be fathomed if its lower bound exceeds the cost of the incumbent since any solution that includes the selected nodes will have a total cost greater than or equal to the incumbent.

The enumeration algorithm begins by choosing a node set and branching. Each resulting subproblem is checked for fathoming. Subproblems which cannot be fathomed are put on a list with other unfathomed partial combination subproblems. The subproblem with the least lower bound is selected and we branch on its next unbranched node set. If branching has

the effect of creating a full combination subproblem (a bottom-of-tree subproblem), then we model the full combination using $(PR_{\hat{\lambda}})$ and solve it as an m-city TSP. The TSP solution tour either produces a new feasible incumbent or it demonstrates that the selection of nodes is not the optimal combination. Hence, bottom-of-tree subproblems are never added to the list. The algorithm continues until the subproblem list is empty or the least lower bound of the list is greater than the incumbent. At this point, the optimal solution is the feasible incumbent.

Throughout the algorithm, each assignment problem solution is checked for being a TSP tour. If a TSP tour is found, we search for a better incumbent by solving acyclic shortest paths over the sequence of sets. If a shortest path cost is lower than the current incumbent cost, the shortest path becomes the new incumbent.

Let $P^k_{i^1,i^2,\ldots,i^k}$ represent a subproblem with the first k node sets fixed so that a solution tour must pass through, in no specific order, node i^1 of set S_1, node i^2 of set S_2, ..., and node i^k of set S_k. Let Z represent the cost of the best feasible incumbent. The enumeration algorithm is given below.

Step 0 : Formulate the problem with no sets fixed, P^0, and solve $(APR_{\hat{\lambda}})$ for a lower bound. If $\mathcal{V}(APR_{\hat{\lambda}}) \geq Z$, stop, the incumbent is optimal. Otherwise, start a list with P^0 and go to Step 1.

Step 1 : If the list is empty, stop, the incumbent is optimal. Otherwise, choose and remove from the list the problem that has the least lower bound. Let that problem be $P^k_{i^1,\ldots,i^k}$. If its lower bound is greater than or equal to Z, stop, the incumbent is optimal. Otherwise, create a subproblem for each node in node set S_{k+1}, that is, $P^{k+1}_{i^1,i^2,\ldots,i^k,j}$ for all $j \in S_{k+1}$. If $k+1 = m$, go to Step 3. Otherwise, go to Step 2.

Step 2 : For each newly created subproblem, $P^{k+1}_{i^1,i^2,\ldots,i^k,j}$, $j \in S_{k+1}$, do the following: If $SP(i,j)+SP(j,i) \geq [Z-\mathcal{V}(APR_{\hat{\lambda}})]$ for any $i = i^1, i^2, \ldots, i^k$, then fathom $P^{k+1}_{i^1,i^2,\ldots,i^k,j}$. Otherwise, formulate $(APR_{\hat{\lambda}})$ for the subproblem and solve the associated $m \times m$ assignment problem. The optimal objective value of the assignment problem represents a lower bound on the subproblem. If the lower bound is greater than or equal to Z, fathom $P^{k+1}_{i^1,i^2,\ldots,i^k,j}$. Otherwise, add $P^{k+1}_{i^1,i^2,\ldots,i^k,j}$ to the list and check if its assignment problem

solution is a feasible TSP tour. If so, solve the acyclic shortest path problems defined by the tour sequence. If a shortest path yields a feasible GTSP tour with cost less than Z, then the path is a new incumbent.

When all newly created subproblems have been addressed, go to Step 1.

Step 3 : For each newly created subproblem, $P^m_{i^1,i^2,\ldots,i^{m-1},j}$, $j \in S_m$, do the following: Formulate $(APR_{\hat{\lambda}})$ for the subproblem and solve the associated $m \times m$ assignment problem. If the optimal assignment problem objective value is greater than or equal to Z, fathom $P^m_{i^1,i^2,\ldots,i^{m-1},j}$. Otherwise, formulate $(PR_{\hat{\lambda}})$ for the subproblem and solve the associated m-city TSP. If the optimal TSP objective value is greater than or equal to Z, fathom $P^m_{i^1,i^2,\ldots,i^{m-1},j}$. Otherwise, the TSP solution is a new feasible incumbent solution and set Z equal to the TSP objective value.

When all newly created subproblems have been addressed, go to Step 1.

3.0 Computational Results

The objectives of the computer implementation were to gain a better understanding of GTSP difficulty and to test the performance of our approach. The procedures were coded in FORTRAN, and tested on a series of randomly generated problems. In addition to specially written subprograms, the code employed routines for solving the following problems; Assignment Problem (Carpaneto, Martello and Toth [1988]), All Shortest Paths (described in Murty [1976]), Asymmetric TSP (Carpaneto and Toth [1980]). All computational tests were performed on the University of Tennessee's Vax 8800 computer.

Test problems were created by specifying the number of sets (m) and number of nodes per set (n). Since each of the m sets contained n nodes, a problem had a total of mn nodes. Arcs were implied to exist from each node to every other node *not* in the same set. Hence, the total number of arcs, $|\mathcal{A}|$, is given as $m(m-1)n^2$.

As in Laporte, Mercure and Nobert [1985,1987], arc costs were generated according to two methods, namely, *Euclidean* and *Non-Euclidean*. For *Euclidean* problems, $2mn$ points are randomly drawn on the $[0, 100]^2$ plane. Let the points be $P_1, P_2, \ldots, P_{mn}$ and $Q_1, Q_2, \ldots, Q_{mn}$.

The Euclidean problem arc costs were calculated as $c_{ij} = ||P_i - P_j||$ if $i < j$ and $c_{ij} = ||Q_i - Q_j||$ if $i > j$ and then rounded to the nearest integer. The *Non-Euclidean* problems were assigned arc costs drawn from a uniform [0,100] distribution and then rounded to the nearest integer.

Table 1 shows the combinations of m, n and cost generation used in our study. For each combination, five random problems were created from randomly drawn seeds. All combinations of five problems were successfully solved and the reported results represent average performance for the five problems.

Results for the bounding procedure show its ability to produce good bounds with little computational expense. Table 1, column 1(a), shows average final lower bounds attained by $(APR_{\hat{\lambda}})$ as a percent of the optimal objective, Z^*. The percentages tend to decrease as the number of nodes per set increases. The more nodes per set present, the more opportunity there is for fractional solutions to be present. As the number of sets increase, there appears to be no definite trend. This is probably due to the fact that we are using an assignment problem relaxation. An assignment problem solution can be decomposed so that each subtour is an optimal assignment over some subset of nodes. Therefore, as the number of sets increases, so will the number of solution subtours. Each subtour group, therefore, has approximately the same ratio of lower bound to optimal. The collection of these ratios stays approximately the same as the number of sets increases. The number of subgradient algorithm iterations averaged 45 for the Euclidean problems and 39 for the Non-Euclidean.

The percentages of arcs and nodes eliminated during the elimination procedure are shown in Table 1, columns 1(b) and 1(c). With some exceptions, the arc eliminations tend to increase as the number of nodes per set increases. This is due to the fact that for a fixed number of sets, an incumbent value obtained will generally be lower as the number of nodes per set increases. The reduced costs, however, will not tend to change with such an increase. Therefore, an increase in number of nodes per set will yield a lower incumbent solution and serve to eliminate a greater percentage of arcs.

The results also show a general tendency for arc eliminations to decrease as the number of sets increases. The problems with more node sets are more difficult to solve and, therefore, the incumbents obtained at this stage of the overall procedure will be relatively further from

Problem Specifications			Bounding	Elim.		Enumeration			
			1(a)	1(b)	1(c)	1(d)	1(e)	1(f)	1(g)
m	n	$\|\mathcal{A}\|$	$\frac{v(APR_1)}{z^*}\%$	% Eliminated Arcs	% Eliminated Nodes	No. AP's	Max Queue	No. TSP's	cpu secs
Euclidean									
8	3	504	87.3	83.6	15.0	89	20	2.6	0.6
	4	896	85.2	88.7	34.4	104	14	1.2	0.6
	6	2016	77.7	84.8	5.8	959	145	4.4	3.4
	10	5600	74.6	94.8	38.3	2863	306	1.2	14.1
	13	9464	68.9	95.4	20.4	2306	281	1.2	23.3
10	3	810	85.6	84.9	9.3	231	27	5.2	1.1
	4	1440	84.3	77.0	6.5	1571	256	13.2	15.0
	6	3240	76.9	81.0	1.0	5381	763	1.6	18.0
	10	9000	73.8	90.6	3.0	19,346	4267	2.6	122.4
12	3	1188	86.5	69.3	0.5	834	124	10.0	3.4
	4	2112	84.9	82.9	2.1	2012	255	7.8	9.0
	6	4752	82.3	90.2	13.6	4563	578	0.2	22.1
15	3	1890	84.6	61.6	3.1	6795	792	†75.8	34.0
	4	3360	87.3	69.3	0.3	6672	1462	6.0	35.7
18	3	2754	87.0	50.7	0.0	12,162	1795	47.0	83.1
Non-Euclidean									
8	3	504	83.0	84.4	21.7	59	12	0.8	0.3
	4	896	67.7	86.5	2.5	277	40	1.6	0.9
	6	2016	69.0	95.2	45.4	236	41	0.4	1.7
	10	5600	41.1	96.7	44.5	262	66	0.2	5.9
	13	9464	30.2	97.8	53.1	468	147	0.2	13.3
10	3	810	77.5	86.6	24.0	320	34	2.0	1.1
	4	1440	59.9	84.6	0.5	1570	178	8.4	4.6
	6	3240	73.3	95.9	50.0	761	229	0.4	4.3
	10	9000	36.2	95.0	11.0	7905	1519	1.6	46.8
12	3	1188	73.7	76.6	2.8	923	109	2.4	3.1
	4	2112	72.7	84.5	0.4	616	146	1.0	3.0
	6	4752	56.7	87.5	3.1	3572	842	0.8	17.4
15	3	1890	73.1	74.9	0.0	1927	558	6.6	8.9
	4	3360	59.3	82.5	0.0	10,558	1748	5.6	51.5
18	3	2754	69.2	71.9	0.0	7772	1027	19.4	47.4

† One problem had 237.

Table 1: Results for test problems (average of five problems for each combination).

optimal than would be for problems with fewer sets. The node eliminations occur mostly in problems with fewer node sets.

The heuristic applications within the elimination procedure often resulted in better incumbents or, in some cases, optimal solutions. The total number of optimal solutions found before the enumeration procedure were 23 out of 75 Euclidean problems and 21 out of 75 Non-Euclidean problems.

Column 1(d) shows the average number of partial combination subproblems for which an $m \times m$ assignment problem was solved. As expected, the numbers generally increase with problem size and difficulty. This contrasts sharply with column 1(e) which shows the maximum number of subproblems enqueued in the partial subproblem list. To store a listed subproblem, we need only an integer vector of length m (to identify its included nodes) and a single real value (to store its lower bound). The results indicate that although a considerable number of nodes are examined, the storage demands remain modest. This claim was supported by implementing our code on an Apple Macintosh SE. Although the cpu times were considerably longer, we were able to solve practically all of the test problems.

For unfathomed full combination subproblems at the bottom of the enumeration tree, we are required to solve an m-city TSP. Column 1(f) shows the average number of TSP's solved for the problem combinations. Overall, these numbers are very low. It means that virtually all of the fathoming occurs at the higher levels of the tree and that the methods for finding optimal solutions through acyclic shortest paths work well.

The overall optimal approach consisted of the bounding, elimination and enumeration algorithms. Table 1, column 1(g), displays the average total cpu times for the problem size combinations (excluding problem read-in). The Euclidean problems generally require more cpu than the Non-Euclidean problems. This is probably due to the fact that fewer arcs, on average, were eliminated from the Euclidean problems.

The test problems generated in Laporte, Mercure and Nobert [1987] differ slightly from our own by including intraset arcs and allowing each set to be visited more than once. However, the authors point out that for the Euclidean problems, it is not necessary to consider intraset arcs or multiple set visits when triangle inequality holds. This allows us to make direct comparisons

Problem Specs.			Laporte, et al. [1987]				Noon and Bean				
$\|\mathcal{N}\|$	m	$\|\mathcal{A}\|$	No. Succ	% $\|\mathcal{A}\|$ Elim.	No. AP's	† cpu secs	No. Succ.	% $\|\mathcal{A}\|$ Elim.	No. AP's	†† cpu secs	cpu Ratio
20	8	348	5	16.4	53	1.56	5	83.3	57	.37	4.2
	11	362	5	8.3	124	4.04	5	83.7	23	.44	9.2
30	9	798	5	14.4	465	32.50	5	67.0	278	1.47	22.1
	11	816	5	12.3	517	36.54	5	74.4	428	1.61	22.7
40	9	1420	5	17.4	265	37.05	5	84.6	723	2.77	13.4
	11	1452	5	12.9	1080	123.26	5	83.0	1585	5.81	21.2
50	8	2186	5	22.1	385	59.17	5	93.1	814	4.22	14.0
	9	2220	5	20.1	507	100.55	5	85.9	1285	5.21	19.3
	11	2270	5	14.7	396	127.79	5	67.8	3914	13.72	9.3
60	8	3148	5	23.7	317	82.00	5	89.1	1308	6.65	12.3
	9	3198	5	22.1	555	147.29	5	90.7	2391	10.03	14.7
	11	3270	1	16.7	1257	262.27	5	77.9	7295	31.04	8.4
70	8	4286	5	24.0	573	195.87	5	90.4	5382	18.54	10.6
	11	4452	1	16.5	519	419.48	5	89.1	4799	44.53	9.4
	13	4520	1	13.9	785	389.33	5	80.4	6679	36.02	10.8
80	8	5600	4	25.7	437	233.61	5	94.8	2863	14.19	16.5
	11	5816	2	18.7	220	223.23	5	88.7	8576	46.44	4.8
90	8	7086	3	27.8	372	269.06	5	91.9	8877	45.08	6.0
100	8	8748	2	26.7	358	275.08	5	96.7	870	17.22	16.0

† cpu seconds on a CYBER 173.

†† cpu seconds on a VAX 8800.

Table 2: Performance on random Euclidean problems.

between results for the Euclidean problems.

To test the relative performance of our approach, we created random Euclidean problems which were similar to those in Laporte, Mercure and Nobert [1987] in terms of size, configuration and arc cost generation. Table 2 displays a comparison of the results. For a given number of nodes $|\mathcal{N}|$ and sets m, the nodes were distributed as uniformly as possible over the sets. An arc was generated from each node to every other node *not* in the same set. The results are compared on the basis of successful attempts out of five problems, the percent of arcs eliminated, the number of bounded enumeration subproblems, and the overall cpu time.

Both approaches were successful on most problems, however, on problems with $|\mathcal{N}| \geq 60$, the Laporte, Mercure and Nobert [1987] approach solved 29 of 50 problems. Our approach successfully solved all 50 such problems. With respect to arc eliminations, our approach was able to significantly reduce the problem size, especially for the very large problems.

A comparison between the number of enumeration subproblems indicates that our approach required *significantly* more assignment problem solutions for lower bounds. It is important to note, however, that our method solves $m \times m$ assignment problems as compared to worst case $|\mathcal{N}| \times |\mathcal{N}|$ assignment problems. If compared on the basis of the total number of basic operations using an $O(n^3)$ assignment problem solver, our approach requird fewer.

The last column displays a ratio of cpu times for the two approaches. Part of the speedup is due to the VAX 8800 being approximately two and one-half times faster than the CYBER 173. However, even after considering the relative computer speeds, it indicates our approach to be faster, especially for medium sized problems.

4.0 Conclusion

The Generalized Traveling Salesman Problem is a difficult optimization problem with potential applications in the areas of distribution, warehousing and scheduling. Its enhancement over the traditional TSP is its ability to simultaneously combine selection and sequencing decisions.

The combined approach of bounding, elimination and enumeration proved to be a practical method for solving the asymmetric GTSP. The approach blends the tasks of searching for the

optimal solution and establishing its optimality in an efficient, organized fashion. As the computational tests have demonstrated, fairly large problems can be solved.

Acknowledgement

The authors would like to thank Paolo Toth for making available his code for the asymmetric TSP.

REFERENCES

Bean, J.C. [1984], "A Lagrangian Algorithm for the Multiple Choice Integer Program," **Operations Research**, Vol. 32, pp. 1185-1193.

Carpaneto, G., S. Martello and P. Toth [1988], "Algorithms and Codes for the Assignment Problem," in "Fortran Codes for Network Optimization," B. Simeone, P. Toth, G. Gallo, F. Maffioli, and S. Pallotino, ed., **Annals of Operations Research**, Vol. 13, J.C. Baltzer, Basel.

Carpaneto, G. and P. Toth [1980], "Some New Branching and Bounding Criteria for the Asymmetric Travelling Salesman Problem," **Management Science**, Vol. 26, pp. 736-743.

Dakin, R. [1965], "A Tree Search Algorithm for Mixed-Integer Programming Problems," **Computer Journal**, Vol 8., pp. 250-255.

Fisher, M.L. [1981], "The Lagrangian Relaxation Method for Solving Integer Programming Problems," **Management Science**, Vol. 27, No. 1, pp. 1-18.

Geoffrion, A.M. [1974], "Lagrangean Relaxation for Integer Programming," **Mathematical Programming Study 2**, pp. 82-114.

Gonzaléz, R.H. [1962], "Solutions of the Traveling Salesman Problem by Dynamic Programming on the Hypercube," Interim Technical Report No. 18, **Massachusetts Institute of Technology** .

Henry-Labordere, A.L. [1969], "The Record Balancing Problem: A Dynamic Programming Solution of a Generalized Travelling Salesman Problem," **RIRO**, B-2, pp. 43-49.

Laporte, G. and Y. Nobert [1983], "Generalized Travelling Salesman Problem Through n Sets of Nodes: An Integer Programming Approach," **INFOR**, Vol. 21, No. 1, pp. 60-75.

Laporte, G., H. Mercure and Y. Nobert [1985], "Finding the Shortest Hamiltonian Circuit Through n Clusters: A Lagrangean Approach," **Congressus Numerantium**, Vol. 48, pp. 277-290.

Laporte, G., H. Mercure and Y. Nobert [1987], "Generalized Travelling Salesman Problem Through n Sets of Nodes: The Asymmetrical Case," **Discrete Applied Mathematics**, Vol. 18, pp. 185-197.

Lawler, E.L., J.K. Lenstra, A.H. Rinnooy Kan and D.B. Shmoys [1985], **The Traveling Salesman Problem**, John Wiley & Sons Ltd., New York.

Murty, K.G. [1976], **Linear and Combinatorial Programming**, John Wiley & Sons Ltd., New York.

Noon, C.E. [1988], "The Generalized Traveling Salesman Problem," unpublished dissertation, Department of Industrial and Operations Engineering, The University of Michigan, Ann Arbor.

Noon, C.E. and J.C. Bean [1989], "An Efficient Transformation of the Generalized Traveling Salesman Problem," working paper, Department of Management, University of Tennessee, Knoxville.

Parker, R.G. and R.L. Rardin [1983], "The Traveling Salesman Problem: An Update of Research," **Naval Research Logistics Quarterly**, Vol. 30, pp. 69-96.

Rousseau, J. [1988], "Customization Versus a General Purpose Code for Routing and Scheduling Problems: A Point of View," in **Vehicle Routing: Methods and Studies**, B. Golden and A. Assad, ed., Elsevier Science Pub., Amsterdam, pp. 469-479..

Saksena, J.P. [1970], "Mathematical Model of Scheduling Clients Through Welfare Agencies," **CORS Journal**, Vol. 8, pp. 185-200.

Srivastava, S.S., S. Kumar, R.C. Garg and P. Sen [1969], "Generalized Traveling Salesman Problem Through n Sets of Nodes," **CORS Journal**, pp. 97-101.