The Generalized Covering Salesman Problem

Bruce Golden
Robert H. Smith School of Business, University of Maryland, College Park, MD 20742, USA.
bgolden@rhsmith.umd.edu

Zahra Naji-Azimi
DEIS, University of Bologna, Viale Risorgimento 2, 40136 Bologna, Italy.
zahra.najiazimi2@unibo.it

S. Raghavan
Robert H. Smith School of Business, University of Maryland, College Park, MD 20742, USA.
raghavan@umd.edu

Majid Salari, Paolo Toth
DEIS, University of Bologna, Viale Risorgimento 2, 40136 Bologna, Italy.
majid.salari2@unibo.it, paolo.toth@unibo.it

Abstract:

Given a graph $G = (N, E)$, the Covering Salesman Problem (CSP) is to identify the minimum length tour "covering" all the nodes. More specifically, it seeks the minimum length tour visiting a subset of the nodes in N such that each node i not on the tour is within a predetermined distance d_i of a node on the tour. In this paper, we define and develop a generalized version of the CSP, and refer to it as the Generalized Covering Salesman Problem (GCSP). Here, each node i needs to be covered at least k_i times and there is a cost associated with visiting each node. We seek a minimum cost tour such that each node i is covered at least k_i times by the tour. We define three variants of the GCSP. In the first case, each node can be visited by the tour at most once. In the second version, visiting a node i more than once is possible, but an overnight stay is not allowed (i.e., to revisit a node i, the tour has to visit another node before it can return to i). Finally, in the third variant, the tour can visit each node more than once consecutively. In this paper, we develop two local search heuristics to find high-quality solutions to the three GCSP variants. In order to test the proposed algorithms, we generated datasets based on TSP Library instances. Since the CSP and the Generalized Traveling Salesman Problem are special cases of the GCSP, we tested our heuristics on both of these two problems as well. Overall, the results show that our proposed heuristics find high-quality solutions very rapidly.

Key Words: Covering Salesman Problem, Generalized Covering Salesman Problem, Generalized Traveling Salesman Problem, Heuristic Algorithms, Local Search.

1. Introduction

The Traveling Salesman Problem (TSP) is one of the most celebrated combinatorial optimization problems. Given a graph $G=(N,E)$, the goal is to find the minimum length tour of the nodes in *N*, such that the salesman, starting from a node, visits each node exactly once and returns to the starting node (see Dantzig et al., 1954). In recent years, many new variants such as the TSP with profits (Feillet et al., 2005), the Clustered TSP (Chisman, 1975), the Generalized TSP (Fischetti et al., 1997), the Prize Collecting TSP (Fischetti and Toth, 1988), and the Selective TSP (Laporte and Martello, 1990) have been introduced and studied. The recent monograph by Gutin and Punnen (2002) has a nice discussion of different variations of the TSP and solution procedures.

Current (1981) defined and introduced a variant of the TSP called the Covering Salesman Problem (CSP). In the CSP, the goal is to find a minimum length tour of a subset of *n* given nodes, such that every node *i* not on the tour is within a predefined covering distance d_i of a node on the tour. If $d_i = 0$ or $d_i < \min_j c_{ij}$, where c_{ij} denotes the shortest distance between nodes *i* and *j*, the CSP reduces to a TSP (thus, it is NP-hard). Current and Schilling (1989) referred to several real-world examples, such as routing of rural healthcare delivery teams where the assumption of visiting each city is not required since it is sufficient that each city is near to a stop on the tour (the inhabitants of those cities which are not on the tour are expected to go to their nearest stop). Current and Schilling (1989) also suggested a heuristic for the CSP where in the first step a Set Covering Problem (SCP) over the given nodes is solved. Specifically, to solve the Set Covering Problem, a zero-one $n \times n$ matrix *A* in which the rows and columns correspond to the nodes is considered. If node *i* can be covered by node *j* (i.e., $d_i \geq c_{ij}$), then a_{ij} is equal to 1, otherwise it is 0. Since the value of covering distance d_i varies for each node *i*, it should be clear that *A* is not a symmetric matrix, but for each node *i* we have $a_{ii} = 1$. We should also mention that, in the CSP, there is no cost associated with the nodes, so the column costs of matrix *A* are all equal to one. Therefore, a unit cost Set Covering Problem is solved in the first step of this algorithm to obtain the cities visited on the tour. Then the algorithm finds the optimal TSP tour over these cities. Since there might be multiple optimal solutions to the SCP, Current and Schilling suggest that all optimal solutions to the SCP be tried out and the best solution be selected. The algorithm is demonstrated on a sample problem, but no additional computational results are reported.

Arkin and Hassin (1994) introduced a geometric version of the Covering Salesman Problem. In this problem, each node specifies a compact set in the plane, its neighborhood, within which the salesman should meet the stop. The goal is computing the shortest length tour that intersects all of the neighborhoods and returns to the initial node. In fact, this problem generalizes the Euclidean

Traveling Salesman Problem in which the neighborhoods are single points. Unlike the CSP, in which each node *i* should be within a covering distance d_i of the nodes which are visited by the tour, in the geometric version it is sufficient for the tour to intersect the specific neighborhoods without visiting any specific node of the problem. Arkin and Hassin (1994) presented simple heuristics for constructing tours for a variety of neighborhood types. They show that the heuristics provide solutions where the length of the tour is guaranteed to be within a constant factor of the length of the optimal tour. Mennell (2009) studied a special case of the above problem known as the close-enough traveling salesman problem. More specifically, if a salesman gets to within a specified distance (say, *r* units) of a node, then the node is considered to have been visited. Mennell implemented and compared a number of geometry-based heuristics. Several of these are very fast and produce high quality solutions.

Other than Current (1981), Current and Schilling (1989), and Arkin and Hassin (1994), the CSP does not seem to have received much attention in the literature. However, some generalizations of the CSP have appeared in the literature. One generalization and closely related problem discussed in Gendreau et al. (1997) is the Covering Tour Problem (CTP). Here, some subset of the nodes must be on the tour while the remaining nodes need not be on the tour. Like the CSP, a node *i* not on the tour must be within a predefined covering distance d_i of a node on the tour. When the subset of nodes that must be on the tour is empty the CTP reduces to the CSP, and when the subset of nodes that must be on the tour consists of the entire node set, the CTP reduces to the TSP. Gendreau et al. (1997) proposed a heuristic that combines GENIUS, a high quality heuristic for the TSP (Gendreau et al., 1992), with PRIMAL1, a high quality heuristic for the SCP (Balas and Ho, 1980).

Vogt et al. (2007) considered the Single Vehicle Routing Allocation Problem (SVRAP) that further generalizes the CTP. Here, in addition to tour (routing) costs, nodes covered by the tour (that are not on it) incur an allocation cost, and nodes not covered by the tour incur a penalty cost. If the penalty costs are set high and the allocation costs are set to 0, the SVRAP reduces to the CTP. Vogt et al. (2007) discussed a tabu search algorithm for the SVRAP that includes aspiration, path relinking, and frequency based-diversification.

All of the earlier generalizations of the CSP assume that when a node is covered, its entire demand can be covered. However, in many real-world applications this is not necessarily the case. As an example, suppose we have a concert tour which must visit or cover several cities. Since each show has a limited number of tickets, and large metropolitan areas are likely to have ticket demand which exceeds ticket supply for a single concert, there must be concerts on several nights in each large city in order to fulfill the ticket demand. Also, in the rural healthcare delivery problem, discussed in Current and Schilling (1989), when we create a route for the rural medical team, on each day a limited number of people can benefit from the services, so the team should visit some places more than once. Consequently, rather than assuming that a node's demand is completely covered when either it or a node that can cover it is visited, we generalize the CSP by specifying the coverage demand k_i which denotes the number of times a node *i* should be covered. In other words, node *i* must be covered k_i times by a combination of visits to node *i* and visits to nodes that can cover node *i*. If k_i=1 for all nodes, we obtain the CSP. This generalization significantly complicates the problem, and is quite different from the earlier generalizations that effectively deal with unit coverage (i.e., k_i=1). In addition, since in many applications there is a cost for visiting a node (e.g., cost of hotel for staying in a city for one night) we include node visiting costs (for nodes on the tour) in the GCSP. In the next section, we introduce and explain in more detail three different variants that can arise in the GCSP (that deal with whether a node can be revisited or not). All of these variants are strongly NP-Hard, since they contain the classical TSP as a special case.

The rest of this paper is organized as follows. In Section 2, we formally define the Generalized Covering Salesman Problem, and describe three variants. We also present a mathematical model for the problem. Section 3 describes two local search heuristics for the GCSP. Section 4 discusses our computational experience on the three different variants of the GCSP, as well as the CSP and the Generalized TSP (GTSP), which are special cases of the GCSP. Section 5 provides concluding remarks and discusses some possible extensions of the GCSP.

2. Problem Definition

In the *Generalized Covering Salesman Problem* (GCSP), we are given a graph $G = (N, E)$ with $N = \{1,2,...,n\}$ and $E = (\{i, j\} : i, j \in N, i < j)$ as the node and edge sets, respectively. Without loss of generality, we assume the graph is complete with edge lengths satisfying the triangle inequality, and let c_{ij} denote the cost of edge $\{i, j\}$ (c_{ij} may be simply set to the cost of the shortest path from node *i* to node *j*). Each node *i* can cover a subset of nodes D_i (note that $i \in D_i$, and when coverage is based on a covering distance the set D_i can be computed easily from the edge costs c_{ij}) and has a predetermined coverage demand k_i. F_i is the fixed cost associated with visiting node *i*, and a solution is feasible if each node *i* is covered at least k_i times by the nodes in the tour. The objective is to minimize the total cost, which is the sum of the tour length and the fixed costs associated with the visited nodes.

We discuss three variants of the GCSP: *Binary GCSP*, *Integer GCSP without overnights*, and *Integer GCSP with overnights*. Next, we explain each of these variants.

Binary Generalized Covering Salesman Problem: In this version, the tour is not allowed to visit a node more than once and after visiting a node we must satisfy the remaining coverage demand of that node by visiting other nodes that can cover it. We use the qualifier binary as this version only permits a node to be visited once.

Integer Generalized Covering Salesman Problem without Overnights: Here, a node can be visited more than once, but an overnight stay is not allowed. Therefore, after visiting a node, the tour can return to this node after having visited at least one other node. In other words, the tour is not allowed to visit a node more than one time consecutively. We use the qualifier integer as this version allows a node to be visited multiple (or an integer number of) times.

Integer Generalized Covering Salesman Problem with Overnights: This version is similar to the previous one, but one or more overnight stays at a node are allowed.

In the CSP, k_i=1 for all nodes $i \in N$. Clearly, the CSP is a special case of the binary GCSP. When there are unit demands there is no benefit to revisiting a node, consequently the CSP can also be viewed as a special case of the integer variants of the GCSP. Thus, the CSP is a special case of all three variants of the GCSP. As the TSP is a special case of the CSP, all three GCSP variants are strongly NP-Hard.

We now discuss the issue of feasibility of a given instance of the problem. For the binary GCSP, the problem is feasible if demand is covered when all nodes in the graph are visited by the tour. In other words if h_j denotes the number of nodes that can cover node j (i.e., h_j counts each node $i \in N$ for which $j \in D_i$), then the problem is feasible if $k_j \le h_j$. For the integer GCSP with and without overnights, the problem is always feasible, since a tour on all nodes in the graph may be repeated until all demand is covered.

We now formulate the three different variants of the GCSP. We first provide an integer programming formulation for the binary GCSP, and then an integer programming formulation for the integer GCSP. Our models are on directed graphs (for convenience, as they can easily be extended to asymmetric versions of the problem). Hence, we replace the edge set E by an arc set A, where each edge $\{i, j\}$ is replaced by two arcs (i, j) and (j, i) with identical costs. Also, from the problem data we have available

$$a_{ij} = \begin{cases} 1 & \text{if node j can cover node i,} \\ 0 & \text{otherwise.} \end{cases}$$

We introduce the decision variables:

$$w_i = \begin{cases} 1 & \text{if node } i \text{ is on the tour} \\ 0 & \text{otherwise} \end{cases}$$

and

$$x_{ij} = \begin{cases} 1 & \text{if arc } (i, j) \text{ is chosen to be in the solution,} \\ 0 & \text{otherwise.} \end{cases}$$

The integer programming model can now be stated as:

(**BinaryGCSP**) $$\text{Min} \sum_{(i,j)\in A} c_{ij} x_{ij} + \sum_{i\in N} F_i w_i \quad (1)$$

subject to:

$$\sum_{j:(j,i)\in A} x_{ji} = \sum_{j:(i,j)\in A} x_{ij} = w_i \qquad \forall i \in N \quad (2)$$

$$\sum_{j\in N} a_{ij} w_j \geq k_i \qquad \forall\ i \in N \quad (3)$$

$$\sum_{l\in S} \sum_{k\in N\setminus S} x_{lk} + \sum_{k\in N\setminus S} \sum_{l\in S} x_{kl} \geq 2\ (w_i + w_j - 1) \qquad S \subset N,\ 2 \leq |S| \leq n-2\ , \quad i \in S,\ j \in N\setminus S \quad (4)$$

$$x_{ij} \in \{0,1\} \qquad \forall\ (i, j) \in A \quad (5)$$

$$w_i \in \{0,1\} \qquad \forall\ i \in N. \quad (6)$$

The objective is to minimize the sum of the tour costs and the node visiting costs. Constraint set (2) ensures that for each on-tour customer, we have one incoming and one outgoing arc. Constraint set (3) specifies that the demand of each node must be covered. Constraint set (4) is a connectivity constraint that ensures that there are no subtours. Note that there are an exponential number of connectivity constraints. Constraints (5) and (6) define the variables as binary.

For the integer GCSP without overnights we introduce two additional variables to represent the number of times a node is visited, and the number of times an arc is traversed in the tour. Let

y_i = number of times that node i is visited by the tour, and

z_{ij} = number of times arc (i,j) is traversed by the tour.

The integer programming model can now be stated as:

(**IntegerGCSP**) $$\text{Min} \sum_{(i,j)\in A} c_{ij} z_{ij} + \sum_{i\in N} F_i y_i \quad (7)$$

subject to:

$$\sum_{j:(j,i)\in A} z_{ji} = \sum_{j:(i,j)\in A} z_{ij} = y_i \qquad \forall i \in N \quad (8)$$

$$\sum_{j\in N} a_{ij} y_j \geq k_i \qquad \forall\ i \in N \quad (9)$$

$$y_i \leq L w_i \qquad \forall i \in N \quad (10)$$

$$z_{ij} \leq L x_{ij} \qquad \forall (i, j) \in A \quad (11)$$

$$\sum_{l \in S}\sum_{k \in N \setminus S} x_{lk} + \sum_{k \in N \setminus S}\sum_{l \in S} x_{kl} \geq 2\ (w_i + w_j - 1) \qquad S \subset N,\ 2 \leq |S| \leq n-2\ , \quad i \in S,\ j \in N \setminus S \quad (12)$$

$$x_{ij} \in \{0,1\}, z_{ij} \in Z^+ \qquad \forall\ (i, j) \in A \quad (13)$$

$$w_i \in \{0,1\}, y_i \in Z^+ \qquad \forall\ i \in N, \quad (14)$$

where L is a sufficiently large positive value. The objective is to minimize the sum of the tour costs and the node visiting costs. Constraint set (8) ensures that if node i is visited y_i times, then we have y_i incoming and y_i outgoing arcs. Constraint set (9) specifies that the demand of each node must be covered. Constraint sets (10) and (11) are linking constraints, ensuring that w_i and x_{ij} are 1 if y_i or z_{ij} are greater than 0 (i.e., if a node is visited or an arc is traversed). Note that it suffices to set $L = \max_{i \in N}\{k_i\}$. Constraint set (12) is a connectivity constraint that ensures that there are no subtours. Note again, that there are an exponential number of connectivity constraints. Finally, constraint sets (13) and (14) define the variables as binary and integer as appropriate. For the integer GCSP with overnights, the above integer programming model (IntegerGCSP) is valid if we augment the arc set A with self loops. Specifically, we add to A the arc set $\{(i,i) : i \in N\}$ (or $A = A \cup \{(i,i) : i \in N\}$) with c_{ii} the cost of self loop arcs (i,i) set to 0.

Note that both the binary GCSP and the integer GCSP formulations rely heavily on the integrality of the node variables. Consequently, the LP-relaxations of these models can be quite poor. Further, these models have an exponential number of constraints, implying that this type of model can only be solved in a cutting plane or a branch-and-cut framework. Thus, considerable strengthening of the above formulations is necessary, before they are viable for obtaining exact solutions to the GCSP. We do not pursue this direction in this paper. Rather, we focus on local search algorithms to develop high-quality solutions for the GCSP.

3. Local Search Algorithms

In this section we propose two local search solution procedures, and refer to them as LS1 and LS2, respectively. They are designed to be applicable to all variants of GCSP. In both algorithms, we start from a random initial solution. As we discussed in Section 2, assuming that a problem is feasible (which can be checked easily for the binary GCSP), any random order of the n nodes produces a feasible solution for the binary GCSP, and repeating this ordering a number of times until all demand is covered produces a feasible solution for the integer GCSP. We provide an initial solution to our local search heuristics by considering a random initial ordering of the nodes in the graph and repeat this ordering for the integer variants (if necessary) to cover all of the demand. A natural question at this point may be whether an initial solution different from a random initial ordering provides better final solutions. We actually tested two alternate possibilities for an initial solution: (i) the TSP tour obtained using the Lin-Kernighan procedure over the *n* nodes in the instance, and (ii) the TSP tour obtained using the Lin-Kernighan procedure (Lin and Kernighan, 1973) over the nodes obtained by solving the associated SCP problem to optimality (this solution may not be feasible for the binary GCSP). We found that these two alternatives did not dominate the random initial ordering which is significantly faster in generating an initial solution. As a result, we generate an initial solution using a random initial ordering.

A solution is represented by the sequence of nodes in the tour. Thus, for the binary GCSP, no node may be repeated on the tour, while in the integer GCSP, nodes may be repeated on the tour. For the integer GCSP with no overnights, a repeated node may not be next to itself in the sequence, while in the integer GCSP with overnights a repeated node is allowed to be next to itself in the sequence. Thus, <1,2,3,4,5,8,9>, <1,2,3,4,3,2,8>, and <1,1,2,3,3,8> represent tour sequences that do not repeat nodes, repeat nodes but not consecutively, and repeat nodes consecutively. Observe that if the costs are non-negative, then in the integer GCSP with overnights, there is no benefit to leaving a node and returning to revisit it later.

3.1. LS1

LS1 tries to find improvements in a solution *S* by replacing some nodes of the current tour. It achieves this in a two-step manner. First, LS1 deletes a fixed number of nodes. (The number of nodes removed from the tour is equal to a predefined parameter, *Search-magnitude*, multiplied by the number of nodes in the current tour. If this number is greater than 1 it is rounded down; otherwise, it is rounded up.) It removes a node *k* from the current solution *S* with a probability that is related to the current tour and computed as

$$P_k = C_k / \sum_{s \in S} C_s \,, \qquad (15)$$

where C_k is the amount of decrease in the tour cost by deleting node *k* from *S*; while keeping the rest of the tour sequence as before. (This weighted randomization allows the search procedure to explore a larger set of the feasible solution space and in our preliminary testing worked much better than choosing the node with the largest value of C_k for deletion.) Since the deletion of some nodes from the tour *S* may result in a tour S' that is no longer feasible, LS1 attempts to make the solution feasible by inserting new nodes into S'. We refer to this as the *Feasibility Procedure*. Suppose that *P* is the set of nodes that can be added to the current tour. For the binary GCSP, *P* consists of the nodes not in the tour S', while in the integer GCSP, *P* consists of all nodes that do not appear more than *L* times in S'. We select the node $k \in P$ for which

$$I_k / N_k^2 = \min_{j \in P}(I_j / N_j^2) \,. \qquad (16)$$

Here, I_k is the amount of increase in the tour cost obtained by inserting node *k* into its best position in the tour, and N_k is the number of uncovered nodes (or uncovered demand) which can be covered by node *k*. (A node that can cover a larger number of uncovered nodes is desirable because it reduces the length of the tour as well as reduces the fixed costs associated with visiting the nodes. We found that squaring N_k strengthens this impact in the selection.) We update the calculation of N_k for all nodes in *P* and repeat the selection and insertion of nodes until we obtain a feasible solution. After this step, LS1 checks for the possible removal of "redundant" nodes from the current tour in the *Delete_Redundant_Nodes Procedure*. A node is redundant if, by removing it, the solution remains feasible.

Next, in case LS1 finds an improvement, i.e., the cost of S' is less than the cost of *S*, we would like to try and improve the tour length (and, thus, the overall cost) by applying the *Lin-Kernighan Procedure* to the solution S'. Since this procedure is computationally expensive, we only apply it after *max_k* (a parameter) improvements over the solution *S*. We use the Lin-Kernighan code LKH version 1.3 of Helsgaun (2000) that is available for download on the web.

In order to escape from locally optimum solutions, and to search through a larger set in the feasible solution space, we apply a *Mutation Procedure* whenever the algorithm is not able to increase the quality of the solution after a given number of consecutive iterations. In the mutation procedure, a node is selected randomly and, if the node does not belong to the solution, it is added to the solution in its best place (i.e., the place which causes the minimum increase in the tour length); otherwise, it is removed from the solution. In the latter case, the algorithm calls the feasibility procedure to ensure the solution is feasible, and updates the best solution if necessary.

To add to the diversity within the search procedure, we allow uphill moves with respect to the best solution that LS1 has found. In other words, if the cost of the solution *S'* that LS1 obtains is less than (1+α) times the best solution found, we keep it as the current solution (over which we try to find an improvement). Otherwise, we use the best solution obtained so far as the current solution. The stopping criterion for LS1 is a given number of iterations that we denote by *max_iter*. The pseudo-code of LS1 is given in Algorithm 1. The parameters to be tuned for LS1 and their best values obtained in our computational testing are described in Table 1 (see Section 4).

3.2. LS2

This local search procedure tries to improve the cost of a solution by either deleting a node on the tour if the resulting solution is feasible or by extracting a node and substituting it with a

Algorithm 1: Local Search Algorithm 1 (LS1) for GCSP

Begin
  S := An initial random tour of n nodes, S^* := S and *BestCost* := Cost(S^*);
  C_s = Decrease in the tour cost by short cutting node s;
  I_s = Increase in the tour cost by adding node s to its best position in the tour;
  N_s = max{1, Number of uncovered nodes covered by node s};
  No_Null_Iter = Number of iterations without improvement;
  Set k := 0; *No_Null_Iter* := 0;
  For i =1, …, *max_iter* **do**
    For j =1, …, *Search-magnitude* × |S| **do**
      Delete node k from S according to the probability $C_k / \sum_{s \in S} C_s$;
    End For;
    S' := Restricted solution obtained by shortcutting the nodes deleted in the previous step;
    Apply *Feasibility Procedure* (S');
    Apply *Delete_Redundant_Nodes Procedure* (S');
    If Cost(S') < Cost(S) **then**
      If k = *max_k* **then** Obtain *TSP_tour*(S') by calling *Lin-Kernighan Procedure* and set k := 0;
      Else k := k+1;
    End If;
    If *Cost(* S' *)* > *BestCost* × (1+α) **then**
      S:=S^*;
      No_Null_Iter:= *No_Null_Iter* +1;
    Else
      $S := S'$;
      If Cost(S) < *BestCost* **then**
        Update S^* := S, *BestCost* := *Cost(S)*,and *No_Null_Iter* := 0;
      End If;
    End If;
    If *No_Null_Iter* > *Mutation_Parameter* **then** apply *Mutation Procedure* (S);
  End For;
  Obtain TSP_tour(S*) by calling *Lin-Kernighan Procedure*. Output the solution S*.
End.

***<u>Feasibility Procedure</u>* (S'):**
  P = The set of nodes that can be entered into the solution;
  While there exist uncovered nodes **do**
    Select node $k \in P$ such that $I_k / N_k^2 = \min_{j \in P}(I_j / N_j^2)$;
    Insert node k in its best position in S' ;
    For each node j update the remaining coverage demand, I_j and N_j ;
  End While.

***<u>Delete_Redundant_Nodes Procedure</u>* (S'):**
  For $i \in S'$ **do**
    If by removing node i from S' the solution remains feasible, **then** remove node i;
  End For.

***<u>Mutation Procedure</u>* (S):**
  Select a random node k from the set of nodes P;
  If node $k \notin S$ **then** add node k to S in its best position;
  Else remove node k from S and call *Feasibility Procedure* (S);
  If Cost(S) < *BestCost* **then** update S^* := S, *BestCost* := *Cost(S)*.

promising sequence of nodes. In contrast to LS1, this local search algorithm maintains feasibility (i.e., it only considers feasible neighbors in the local search neighborhood).

LS2 mainly consists of two iterative procedures: the *Improvement Procedure* and the *Perturbation Procedure*. In the *Improvement Procedure*, the algorithm considers extraction of nodes from the current tour in a round-robin fashion. (In other words, given some ordering of nodes on the tour, it first tries to delete the first node on the tour, and then it tries to delete the second node on the tour, and so on, until it tries to delete the last node on the tour.) If by removing a node on the tour the solution remains feasible, the tour cost has improved and the node is deleted from the tour. On the other hand, extracting a node from the tour may cause some other nodes to no longer be fully covered and the solution becomes infeasible. Consequently, in such cases, we try to obtain a feasible solution by substituting the deleted node with a new subsequence of nodes. To this aim, the algorithm considers the T nodes nearest to the extracted node and generates all the promising subsequences with cardinality one or two. Then, it selects the subsequence s that has the minimum insertion cost (i.e., the cost of the tour generated by substituting the deleted node by subsequence s minus the cost of tour with the deleted node). In the case of improvement in the tour cost, we make this substitution; otherwise, we disregard it (i.e., reinsert the deleted node back into its initial position) and continue. The improvement procedure is repeated until it cannot find any improvements (i.e., no change is found while extracting nodes from the current tour in a round-robin fashion). At the end of the improvement procedure, we apply the 2-Opt procedure to the current tour. (We prefer 2-Opt to the Lin-Kernighan procedure here because it is significantly less expensive computationally.)

In the *Perturbation Procedure*, LS2 tries to escape from a locally optimum solution by perturbing the solution. In the perturbation procedure, we iteratively add up to K nodes to the tour. It randomly selects one node from among the nodes eligible for addition to the tour (in the binary GCSP, the nodes must be selected from those not in the current tour, while, for the two other GCSP variants, the nodes can be selected from those visited in the current tour also) and inserts it in the tour in its best possible position. Since the tour is feasible prior to the addition of these nodes, the tour remains feasible upon addition of these K nodes.

In one iteration of the procedure, the improvement phase and perturbation phase are iteratively applied J times. After one iteration, when the best solution has improved (i.e., an iteration found a solution with lower cost), we use the *Lin-Kernighan Procedure* to improve the current tour length (and, thus, the cost of the solution). The stopping criterion for LS2 is a given number of iterations that we denote by *max_iter*. The pseudo-code for LS2 is given in Algorithm 2, and the parameters to be tuned for LS2 and their best values obtained in our computational testing

Algorithm 2: Local Search Algorithm 2 (LS2) for the GCSP

Begin
  S := An initial random tour of n nodes, S^* := S and *BestCost* := Cost(S^*);
  $N(S)$ = Number of nodes in S;
  For i = 1, …, *max_iter* **do**
    bestimprove := false;
    For j = 1, …, J **do**
      improve := true;
      Repeat
        Improvement Procedure (S,*improve*);
      Until (*improve* = false)
      2-OPT(S);
      If Cost(S) < *BestCost* **then**
        S^* := S;
        BestCost := Cost(S);
        bestimprove := true;
      Else
        S := S^*;
      End If;
      Perturbation Procedure (S);
    End For;
    If (*bestimprove* = true) **then**
      Obtain TSP_tour(S*) by calling *Lin-Kernighan Procedure*. Output the solution S*;
      S := S^* and *BestCost* := Cost(S^*);
    End If;
  End For;
End.

***Improvement Procedure*(*S*,*improve*):**
Begin
  improve := false;
  r := 1;
  While $\left(r \le N\left(S\right)\right)$ **do**
    Extract the r^{th} node of the tour from the current solution S;
    If the new solution (obtained from extracting the r^{th} node of S) is feasible **then**
      S := new solution;
      improve := true;
    Else
      Generate all subsequences with cardinality 1 or 2, by considering the T closest nodes to the extracted node**;**
      Extra_Cost := Extra cost for the subsequence with the minimum insertion cost;
      If *Extra_Cost* < 0 **then**
        Update S by substituting the new subsequence for the extracted node;
        improve := true;
      End If;
    End If;
    r := r+1;
  End While;
End.

***Perturbation Procedure*(*S*):**
Begin
  For i = 1,…, K **do**
    Randomly select an eligible node;
    Insert the node in its best feasible position in the tour;
  End For;
End.

are described in Table 2 (see Section 4).

4. Computational Experiments

In this section, we report on our computational experience with the two local search heuristics LS1 and LS2 on the different GCSP variants. We first consider the CSP, and compare the performance of the two proposed heuristics LS1 and LS2, with that of the method proposed by Current and Schilling (1989) for the CSP. Next, we compare LS1 and LS2 on a large number of GCSP instances for the three variants. We also consider a Steiner version of the GCSP, and report our experience with the two local search heuristics. Finally, in order to assess the quality of the solutions found by the two heuristics, we compare them with existing heuristics for the GTSP where there exist well-studied instances in the literature. All of the experiments suggest that the heuristics are of a high quality and run very rapidly.

4.1. Test Problems

Since there are no test problems in the literature for the CSP (nor for the variants of the GCSP we introduce), we created datasets based on the TSP library instances (Reinelt, 1991). Our analysis involves two datasets: one consisting of small to medium instances with up to 200 nodes, and the other consisting of large instances with up to 1000 nodes.

The first dataset is constructed based on 16 Euclidean TSPLIB instances whose size ranges from 51 to 200 nodes. In the instances created, each node can cover its 7, 9, or 11 nearest neighbor nodes in addition to itself (resulting in 3 instances for each TSPLIB instance), and each node *i* must be covered k_i times, where k_i is a randomly chosen integer number between 1 and 3. We generated the instances to ensure that a tour over all of the nodes covers the demand (i.e., we ensured that the binary GCSP instances were feasible). Although the cost of visiting a node can be different from node to node, for the first dataset, we consider the node-visiting costs to be the same for all nodes in an instance. In fact, if we assign a large node-visiting cost, the problem becomes a Set Covering Problem (as the node-visiting costs dominate the routing costs) under the assumption that a tour over all the nodes covers the demand. On the other hand, if the node-visiting cost is insignificant (i.e., the routing costs dominate), there is no difference between the integer GCSP with overnights and the CSP. This is because if there is no node-visiting cost, a salesman will stay overnight at a node (at no additional cost) until he/she covers all the demand that can be covered from that node. After testing different values for the node-visiting cost, to ensure that its effect was not at either extreme (Set Covering Problem or CSP), we fixed the node-visiting cost value to 50 for

Table 1: Parameters for LS1

Parameters	Different values tested	Best value
Search-magnitude	{0.1, 0.2, 0.3, 0.4, 0.5, 0.6}	0.3
Mutation_parameter	{5, 10, 15, 20}	15
max_k	{5, 10, 15, 20}	20
α	{0, 0.1, 0.01, 0.001}	0.1
max_iter	{1500, 3500, 5500, 7500, 8500}	3500 (CSP & Binary GCSP) 7500 (Integer GCSP)

Table 2: Parameters for LS2

Parameters	Different values tested	Best value
J	{50, 100, 150, 200, 250, 300}	300
K	{5, 10, 15, 20}	10
T	{5, 10, 15}	10
max_iter	{15, 20, 25, 30, 35, 40, 45, 50, 55, 60}	25 (CSP & Binary GCSP) 50 (IntegerGCSP)

the instances in this first dataset. In this fashion, we constructed 48 instances in this first dataset for our computational work.

The second dataset is constructed based on 15 large Euclidean TSPLIB instances whose size ranged from 225 to 1000 nodes. They are constructed in the same fashion as the first dataset, except that (i) each node can cover its 10 nearest neighbor nodes, and (ii) the node-visiting costs are different for the nodes. The node-visiting costs are set randomly in three different ranges. If TL denotes the total length of the solution (using LS2) to the instance as a CSP problem, the node-visiting costs are set in the following three ranges (resulting in 3 instances for each TSPLIB instance), where (int) indicates we round down to the nearest integer.

A: (int) (Random value in [0.01, 0.05]) * TL.
B: (int) (Random value in [0.001, 0.005]) * TL.
C: (int) (Random value in [0.0001, 0.0005]) * TL.

Tables 1 and 2 show the different values that were tested for various parameters and the best value obtained for the parameters in LS1 and LS2. To obtain these best parameter values we tested all possible combinations for the parameter values on a set of small test instances consisting of five selected TSPLIB instances (Pr107, Pr124, Pr136, Pr144, and Pr152) for each of the four problems (CSP, binary GCSP, integer GCSP without overnights, and integer GCSP with overnights) where each node can cover its 10 nearest neighbor nodes. These five instances (with a total of 20 instances over the four problems) comprise our training set while our testing set will consist of the instances in the two datasets. This approach is consistent with the one widely used in the data mining and machine learning literature.

Both LS1 and LS2 were implemented in C and tested on a Windows Vista PC with an Intel Core Duo processor running at 1.66 GHz with 1 GB RAM. As is customary in testing the performance of randomized heuristic algorithms, we performed several independent executions of the algorithms. In particular, for each benchmark instance, 5 independent runs of the algorithms LS1 and LS2 were performed, with 5 different seeds for initializing the random number generator and the best and the average performances of the two heuristics are provided.

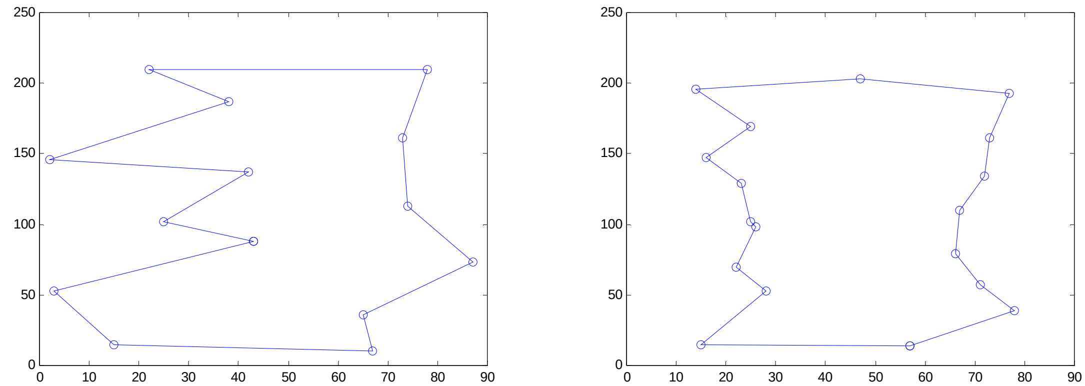
In all tables reporting the computational performance of the heuristics, the first column is related to the instance name which includes the number of nodes. The second column (NC) gives the number of nearest neighbor nodes that can be covered by each node. Moreover, for each method, the best and the average cost, the number of nodes in the best solution (NB), the average time to best solution (Avg.TB), i.e., the average time until the best solution is found (note the local search heuristic typically continues after this point until it reaches its termination criterion), and the average total time (Avg.TT) are reported (TT is the total time for one run of the local search heuristic). In all tables, in each row the best solution is written in bold and the last two rows give the average of each column (Avg) and the number of best solutions found by each method (No.Best), respectively. All the computing times are expressed in seconds.

4.2. Comparison of LS1, LS2, and Current and Schilling's Heuristic for the CSP

Since Current and Schilling (1989) introduced the CSP and proposed a heuristic for it, we compare the performance of LS1 and LS2 against their heuristic. Recall, their algorithm was described in Section 1. Since there are no test instances or computational experiments reported in Current and Schilling's paper, we coded their algorithm to compare the performance of the heuristics. For Current and Schilling's method, we used CPLEX 11 (2007) to generate all optimal solutions of the SCP, and since solving the TSP to optimality is computationally quite expensive on these instances, we use the *Lin-Kernighan Procedure* to find a TSP tour for each solution. Sometimes, finding all the optimal solutions of an SCP instance is quite time consuming, so we only consider those optimal solutions for the SCP that CPLEX can find in less than 10 minutes of running time.

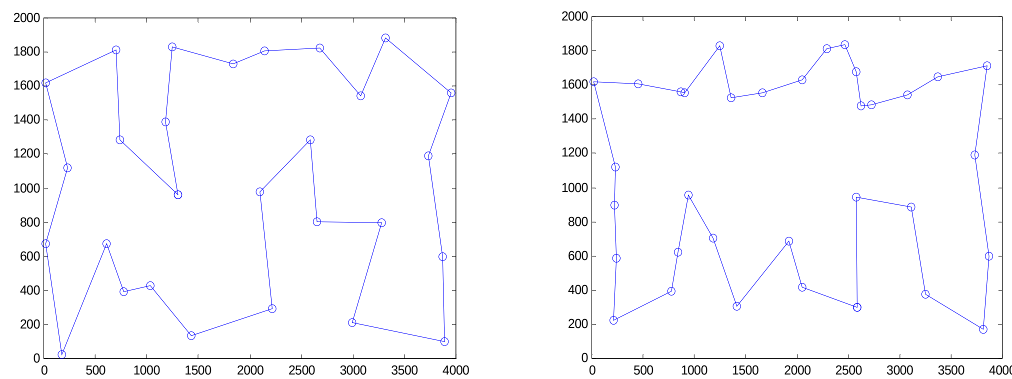
Table 3 reports the results obtained by LS1, LS2, and our implementation of Current and Schilling's method. In this table, the number of optimal solutions (NO) of the set covering problem is given. In Table 3, instances for which all the optimal solutions to the set covering problem cannot be obtained within the given time threshold are shown with an asterisk. As can be seen in Table 3 for the CSP, both LS1 and LS2 can obtain, in a few seconds, better solutions than Current and Schilling's method. In every case but one (where there is a tie), LS1 and LS2 outperform Current and Schilling's method, while they are several orders of magnitude faster than Current and Schilling's method. Between LS1 and LS2, LS2 outperforms LS1 as it obtains the best solution in

Figure 1. An example of decreasing the tour length by increasing the number of nodes in Rat99 (NC=7).



a) Number of nodes in the tour: 14, Tour length: 572
b) Number of nodes in the tour: 18, Tour length: 486

Figure 2. An example of decreasing the tour length by increasing the number of nodes in KroA200 (NC=7).



a) Number of nodes in the tour: 28, Tour length: 14667
b) Number of nodes in the tour: 34, Tour length: 13285

all 48 instances, while LS1 only obtains the best solution in 39 out of the 48 instances. We applied the Wilcoxon signed rank test (Wilcoxon, 1945), a well-known nonparametric statistical test to perform a pairwise comparison of the heuristics. In each comparison, the null hypothesis is that the two heuristics being compared are equivalent in performance (with the alternate hypothesis that one of them is superior). Based on this analysis we found that LS1 outperforms Current and Schilling's heuristic at a significance level of $\alpha<0.0001$ (in other words, we can reject the null hypothesis that the performance of LS1 and Current and Schilling's heuristic are equivalent at a significance level of $\alpha<0.0001$ and accept the alternate hypothesis that LS1 outperforms Current and Schilling's heuristic). We also found that LS2 outperforms Current and Schilling's heuristic at a significance level of $\alpha<0.0001$. When we compared LS2 and LS1, we found that LS2 outperforms LS1 at a significance level of $\alpha=0.002$.

From Table 3 we can make the following observation. Sometimes by selecting a larger number of nodes to visit, we are able to find a shorter tour length, so the optimal solution of the Set

Covering Problem is not necessary a good solution for the Covering Salesman Problem. This is contrary to the intuition at the heart of Current and Schilling's method. Figures 1 and 2 illustrate two examples of CSP (Rat99 and KroA200) in which, by increasing the number of nodes in the tour, the tour length is decreased.

4.3. Comparison of LS1 and LS2 on GCSP Variants

In Table 4 the results of the two local search heuristics on the binary GCSP are given. As can be seen in this table, for the binary GCSP, the two local search heuristics are very competitive with each other. On average, LS1 is a bit faster than LS2. In terms of the average cost, the number of best solutions found, and average time, LS1 is slightly better than LS2. However, LS2 is faster in terms of the average time to best solution. Over the 48 instances, the two heuristics were tied in 21 instances. In 14 instances, LS1 is strictly better than LS2, and, in 13 instances, LS2 is strictly better than LS1. When we compared LS1 and LS2 using the Wilcoxon signed-rank test, we found that the difference between the two was not statistically significant. (Here α=0.1469, and we used a 5% significance level for the hypothesis tests. In other words we reject the null hypothesis that the performance of the heuristics is equal only if α<0.05).

Table 5 provides a comparison of LS1 and LS2 on the integer GCSP without overnights. Here, the table contains one additional column reporting the number of times a solution revisits cities (NR). Here, over 48 test instances, LS1 is strictly better than LS2 in 15 instances, LS2 is strictly better than LS1 in 8 instances, while they are tied in 25 instances. Again, the running times of both LS1 and LS2 are generally small, taking no more than 32 seconds, even for the largest instances. When we compared LS1 and LS2 using the Wilcoxon signed rank test, we found that the difference between the two was not statistically significant (here α=0.1264).

Table 6 compares LS1 and LS2 on integer GCSP with overnights. Here, the table contains one additional column reporting the number of times a solution stays overnight at a node (ON). Over 48 test instances, LS1 is strictly better than LS2 in 9 instances, LS2 is strictly better than LS1 in 26 instances, and they are tied in 13 instances. However, the running time of LS1 increases significantly with instance size compared to LS2. This increase in running time appears to be due to a significant increase in the number of times LS1 calls the *Lin-Kernighan Procedure*. Overall, LS2 appears to be a better choice than LS1 for the integer GCSP with overnights. When we compared LS1 and LS2 using the Wilcoxon signed rank test, we found that LS2 outperforms LS1 at a significance level of α=0.0027.

Notice that a solution to the binary GCSP is a feasible solution to the integer GCSP without overnights, and a feasible solution to the integer GCSP without overnights is a feasible solution for the integer GCSP with overnights. Hence, we should expect that the average cost of the solutions found should go down as we move from Table 4 to Table 6. This is confirmed in our experiments.

4.4. GCSP with Steiner Nodes

In our earlier test instances, every node had a demand. We now construct some Steiner instances, i.e., ones where some nodes have k_i set to zero (the rest of the demands remain unchanged). In these cases, a tour could contain some "Steiner nodes" (i.e., nodes without any demand) that can help satisfy the coverage demand of the surrounding (or nearby) nodes. On the other hand, if fewer nodes have demands then it is likely that fewer nodes need to be visited (in particular, the earlier solutions obtained are feasible for the Steiner versions), and, thus, we would expect the cost of the solutions to the GCSP with Steiner nodes to be lower compared to the instances of the GCSP without Steiner nodes. Table 7 confirms this observation. Here, we compare LS1 and LS2 on the CSP with Steiner nodes. For each CSP instance (in Table 3) we select 10 percent of the nodes randomly and set their corresponding demands to zero. The behavior of LS1 and LS2 is similar to that of the earlier CSP instances. Specifically, over the 48 test instances, LS2 obtains the best solution in all 48 instances while LS1 only obtains the best solution in 40 out of 48 instances. Although LS2 outperforms LS1 in terms of finding better solutions, overall, LS1 runs faster than LS2. When we compared LS1 and LS2 using the Wilcoxon signed rank test, we found that LS2 outperforms LS1 at a significance level of α=0.0039. For brevity, we have limited the Steiner nodes comparison to the CSP.

4.5. Large-Scale Instances

Tables 8, 9, and 10 report on our experience with the second dataset consisting of the large scale instances with varying node-visiting costs. As the results show, LS2 significantly outperforms LS1. In particular for the binary GCSP, LS2 is strictly better than LS1 in 43 out of 45 instances, while LS1 is strictly better than LS2 in 2 out of 45 instances. The Wilcoxon signed rank test indicates that LS2 outperforms LS1 at a significance level of α<0.0001. In terms of average running time, LS2 is an order of magnitude (about 10 times) faster than LS1. For the GCSP without overnights, LS2 is strictly better than LS1 in 44 out of 45 instances, while LS1 is strictly better than LS2 in 1 out of 45 instances. The Wilcoxon signed rank test indicates that LS2 outperforms LS1 at a significance level of α<0.0001. Again, in terms of running time, LS2 is an order of magnitude faster than LS1. Even in the largest instances, LS2 takes no more than 4 minutes, while LS1 can take over 20 minutes to run. For the GCSP with overnights, LS2 is strictly better than LS1 in 44 out of 45 instances, while LS1 is strictly better than LS2 in 1 out of 45 instances. The Wilcoxon signed rank test indicates that LS2 outperforms LS1 at a significance level of $\alpha<0.0001$. Again, in terms of running time, LS2 is much faster than LS1, taking only about 3 minutes even for the largest 1000 node instance.

Taken together with the results on the first dataset, overall, LS2 seems to be a better choice than LS1. On the first dataset it outperforms LS1 (in a statistically significant sense) on the CSP, the Steiner CSP, and the integer GCSP with overnights. While the difference between LS1 and LS2 for the binary GCSP and the integer GCSP without overnights on the first dataset is not statistically significant. However, the results on the larger instances in the second dataset clearly show the superiority of LS2 over LS1 in terms of performance on all three problems, as well as its robustness in terms of running time.

4.6. Analyzing the Quality of LS2 on the Generalized TSP

Since we do not have lower bounds or optimal solutions for the CSP and GCSP instances, it is hard to assess the quality of the solutions. Noting that the generalized TSP (GTSP) is a special case of the CSP (we explain how momentarily), we use some well-studied GTSP instances in the literature (see Fischetti et al., 1997) and compare LS2 with eight different heuristics designed specifically for the GTSP; as well as to the optimal solutions on these instances obtained by Fischetti et al. (1997) using a branch-and-cut method. In the GTSP, the set of nodes in the graph are clustered into disjoint sets and the goal is to find the minimum length tour over a subset of nodes so that at least one node from each cluster is visited by the tour. This can be formulated as a CSP, where each node has unit demand (i.e., $k_i=1$ for each node i) and each node in a cluster covers every other node in a cluster (and no other nodes).

We executed LS2 on the benchmark GTSP dataset (see Fischetti et al., 1997) by first tuning its parameters. The tuned parameters of LS2 are configured as follows: J=300, K=10, T=10, and *max_iter* = 50 and 10 independent runs of LS2 were performed. We compared LS2 to eight other heuristics in the literature that are listed below.

1. MSA: A Multi-Start Heuristic by Cacchiani et al. 2011,
2. mrOX: a Genetic Algorithm by Silberholz and Golden 2007,
3. RACS: a Reinforcing Ant Colony System by Pintea et al. 2007,
4. GA: a Genetic Algorithm by Snyder and Daskin 2006,
5. GI^3 : a composite algorithm by Renaud and Boctor 1998,
6. NN: a Nearest Neighbor approach by Noon 1988,
7. *FST-Lagr* and *FST-root*: Two heuristics by Fischetti et al. 1997.

In order to perform a fair comparison on the running times of the different heuristics, we scaled the running times for the different computers as indicated in Dongarra (2004). The computer factors are shown in Table 11. The columns indicate the computer used, solution method used, *Mflops* of the computer, and *r* the scaling factor. Thus the reported running times in the different papers are appropriately multiplied by the scaling factor *r*. We note that an identical approach was taken in Cacchiani et al. (2011) to compare across these heuristics for the GTSP. Since no computer information is available for the RACS heuristic, we use a scaling factor of 1.

Table 12 reports on the comparison. For each instance, we report the percentage gap with respect to the optimal solution value and the computing time (expressed in seconds and scaled according to the computer factors given in Table 11) for all the methods except for B&C (for which we report only the computing time). Some of the methods (RACS, GI^3, and NN) only reported solutions for 36 of the 41 instances. Consequently, in the last four rows of Table 12, we report for each algorithm, the average percentage gap and the average running time on the 36 instances tested by all the methods, as well as over all 41 instances (for all methods except RACS, GI^3, and NN). We also summarize the number of times the optimum solution was found by a method. As Table 12 indicates, although LS2 was not explicitly developed for the GTSP (but rather for a generalization of it), it performs quite well. On average, it takes 2.2 seconds, and finds solutions that are 0.08% from optimality, and it found optimal solutions in 30 out of 41 benchmark GTSP instances.

5. Summary and Conclusions

In this paper, we considered the CSP, and introduced a generalization quite different from earlier generalizations of the CSP in the literature. Specifically, in our generalization, nodes must be covered multiple times (i.e., we introduce a notion of coverage demand of a node). This may require a tour to visit a node multiple times (which is not the case in earlier generalizations), and there are also node-visiting costs. We discussed three variants of the GCSP. The binary GCSP where revisiting a node is not permitted, the integer GCSP without overnights where revisiting a node is permitted only after another node is visited, and the integer GCSP with overnights where revisiting a node is permitted without any restrictions. We designed two local search heuristics, LS1 and LS2, for these variants. In a certain sense, LS1 is a very large-scale neighborhood search (VLNS) algorithm (because it relies heavily on the Lin-Kernighan procedure and a neighborhood whose size is variable and large), and LS2 is like an iterated local search (ILS) algorithm (because it applies an improvement and perturbation procedure iteratively and the sophisticated Lin-Kernighan procedure is applied infrequently). Nevertheless, the Lin-Kernighan procedure plays a significant role in the performance of both algorithms. Indeed, if the Lin-Kernighan procedure were replaced by the 2-OPT procedure in the algorithms, the performance of the heuristics is significantly worse. Overall, LS2 appears to be superior in terms of its running time as well as its performance with respect to the number of times it found the best solutions for the different variants. When LS2 is compared to 8 benchmark heuristics for the GTSP (that were specifically designed for the GTSP), LS2 performs quite well, finding high-quality solutions rapidly.

We introduced two integer programming models for the binary and integer GCSP, respectively. However, both these models require considerable strengthening and embedding in a branch-and-cut framework in order to obtain exact solutions to the GCSP. This is a natural direction for future research on the GCSP (as it will provide an even better assessment of the quality of heuristics for the GCSP), and we hope researchers will take up this challenge.

Some natural generalizations of the GCSP (along the lines of the earlier generalizations of the CSP) may also be considered in future research. The earlier generalizations of the CSP (see Vogt et al., 2007) included constraints in terms of (i) requiring some nodes to be on the tour, (ii) requiring some nodes not to be on the tour, (iii) allowing a node not to be covered at a cost (for our GCSP that would mean the covering demand of a node could be partially covered at a cost), and (iv) including a cost for allocating nodes not on the tour to the tour. These would be natural generalizations of this multi-unit coverage demand variant of the CSP that we have introduced.

References

Arkin E.M. and Hassin R. 1994. Approximation Algorithms for the Geometric Covering Salesman Problem, *Discrete Applied Mathematics* **55**(3), 197-218.

Balas E. and Ho A. 1980. Set Covering Algorithms Using Cutting Planes, Heuristics, and Subgradient Optimization: A Computational Study. *Math. Programming*, **12**, 37-60.

Cacchiani V., Fernandes Muritiba A.E., Negreiros M., and Toth P. 2011. A Multi-Start Heuristic For the Equality Generalized Traveling Salesman Problem, *Networks*, to appear.

Chisman, J. A. 1975. The Clustered Traveling Salesman Problem, *Computers and Operations Research*, **2**(2), 115-119.

Current J.R. 1981. Multi-objective design of Transportation Networks, Ph.D thesis, Department of Geography and Environmental Engineering. The Johns Hopkins University, Baltimore.

Current J.R., and Schilling D.A. 1989. The Covering Salesman Problem. *Transportation Science*, **23**(3), 208-213.

Dantzig G.B., Fulkerson R., and Johnson S.M. 1954. Solution of a Large Scale Traveling Salesman Problem. *Operations Research*, **2**(4), 393-410.

Dongarra J.J. 2004, Performance of various computers using standard linear equations software, Technical Report CS-89-85, Computer Science department, University of Tennessee.

Feillet D., Dejax P., and Gendreau M. 2005. Traveling Salesman Problems with Profits. *Transportation Science*, **39**(2), 188-205.

Fischetti M., and Toth P. 1988. An Additive Approach for the Optimal Solution of the Prize-Collecting Traveling Salesman Problem. In Vehicle Routing: Methods and Studies. Golden B. L. and Assad A. A. (eds.). North-Holland, Amsterdam, 319-343.

Fischetti M., Salazar-Gonzalez J.J., and Toth P. 1997. A Branch and Cut Algorithm for the Symmetric Generalized Traveling Salesman Problem. *Operations Research*, **45**(3), 378-394.

Gendreau M., Hertz A., and Laporte G. 1992. New Insertion and Postoptimization Procedures for the Traveling Salesman Problem. *Operations Research,* **40**(6), 1086-1094.

Gendreau M., Laporte G., and Semet F. 1997. The Covering Tour Problem. *Operations Research*, **45**(4), 568-576.

Gutin G. Punnen A.P. (Eds.). 2002. The Traveling Salesman Problem and Its Variations. Kluwer Academic publishers, Netherlands.

Helsgaun K. 2000. An Effective Implementation of the Lin-Kernighan Traveling Salesman Heuristic. *European Journal of Operational Research,* **126** (1), 106-130.

ILOG Cplex 11.0, 2007. User's Manual and Reference Manual, ILOG, S.A., http://www.ilog.com.

Laporte G., and Martello S. 1990. The Selective Traveling Salesman Problem. *Discrete Applied. Mathematics*, **26**, 193-207.

Lin S., and Kernighan B.W. 1973. An Effective Heuristic Algorithm for the Traveling Salesman Problem. *Operations Research,* **21**(2), 498-516.

Mennell W. 2009. Heuristics for solving three routing problems: Close enough traveling salesman problem, close-enough vehicle routing problem, sequence-dependent team orienteering problem. Ph.D thesis, University of Maryland, College Park.

Noon C. E. 1988. The generalized traveling salesman problem, Ph.D. thesis, University of Michigan.

Pintea C.M., Pop P.C., and Chira C. 2007. The Generalized Traveling Salesman Problem Solved with Ant Algorithms, *Journal of Universal Computer Science,* **13**(7), 1065-1075.

Reinelt G. 1991. A Traveling Salesman Problem Library. *ORSA Journal on Computing*, **3**(4), 376-384.

Renaud J., and Boctor F.F. 1998. An efficient composite heuristic for the symmetric generalized traveling salesman problem, *European Journal of Operational Research,* **108**(3), 571-584.

Silberholz J. and Golden B. 2007. The Generalized Traveling Salesman Problem: a new Genetic Algorithm approach, in Baker E.K., Joseph A., Mehrotra A., and Trick M.A.(eds.) Extending the Horizons: Advances in Computing, Optimization Decision Technologies, Springer, 165-181.

Snyder L.V., and Daskin M.S. 2006. A random-key genetic algorithm for the generalized traveling salesman problem, *European Journal of Operational Research,* **174**(1), 38-53.

Vogt L., Poojari CA. and Beasley JE. 2007. A Tabu Search algorithm for the Single Vehicle Routing Allocation Problem, *Journal of Operational Research Society,* **58**(4), 467-480.

Wilcoxon, F. 1945. Individual comparisons by ranking methods, *Biometrics*, **1**(6), 80–83.

Table 3. Comparison of Current and Schilling's method with LS1 and LS2 for CSP

Instance	NC	Current and Schilling					LS1					LS2				
		NO	Cost	NB	TB	TT	Best Cost	Avg. Cost	NB	Avg. TB	Avg. TT	Best Cost	Avg. Cost	NB	Avg. TB	Avg. TT
Eil 51	7	13	194	7	0.07	0.21	**164**	164.0	10	0.05	0.73	**164**	164.0	10	0.04	1.05
	9	309	169	6	1.92	1.97	**159**	159.0	8	0.16	1.51	**159**	159.0	9	0.04	0.78
	11	282	167	5	0.59	1.70	**147**	147.0	7	0.04	0.69	**147**	147.0	7	0.04	0.87
Berlin 52	7	2769	4019	8	19.39	21.04	**3887**	3887.0	12	0.42	0.73	**3887**	3887.0	11	0.42	1.05
	9	11478	**3430**	7	26.08	94.14	**3430**	3430.0	7	0.03	1.32	**3430**	3430.0	7	0.04	0.95
	11	11	3742	5	0.22	0.26	**3262**	3262.0	6	0.02	0.74	**3262**	3262.0	6	0.04	0.55
St 70	7	32832	297	10	232.24	454.07	**288**	288.0	11	0.08	0.87	**288**	288.0	11	0.06	1.48
	9	18587	271	9	173.87	176.00	**259**	259.0	10	0.03	1.45	**259**	259.0	10	0.06	1.65
	11	1736	269	7	13.21	13.74	**247**	247.0	10	0.12	0.83	**247**	247.0	10	0.05	1.37
Eil 76	7	241	241	11	1.15	2.46	**207**	207.0	17	0.12	0.70	**207**	207.0	15	0.27	1.35
	9	1439	193	9	7.43	13.95	186	186.0	11	0.37	1.59	**185**	185.0	12	0.06	1.43
	11	7050	180	8	30.48	78.88	**170**	170.0	12	0.24	0.78	**170**	170.0	11	0.06	1.50
Pr 76	7	26710	53255	11	54.20	170.41	**50275**	50275.0	15	0.15	0.85	**50275**	50275.0	14	0.10	1.89
	9	326703*	45792	10	6743.66	9837.36	**45348**	45389.6	13	0.57	2.45	**45348**	45462.2	12	0.06	1.50
	11	20	45955	7	0.11	0.20	**43028**	43028.0	11	0.29	0.69	**43028**	43028.0	10	0.13	1.39
Rat 99	7	3968	572	14	22.74	32.75	**486**	486.0	19	0.20	0.98	**486**	486.0	18	0.11	2.45
	9	170366	462	12	1749.66	2729.67	**455**	455.0	15	0.65	2.47	**455**	455.0	15	0.15	2.45
	11	16301	456	10	88.87	140.18	**444**	444.0	14	0.49	0.95	**444**	444.0	13	0.09	2.26
KroA 100	7	208101*	10306	15	6303.03	6475.95	**9674**	9674.0	19	0.52	1.06	**9674**	9674.0	19	0.36	2.78
	9	95770	9573	12	524.49	1365.42	**9159**	9159.2	16	0.83	1.98	**9159**	9159.0	15	0.17	2.30
	11	33444	9460	10	409.47	433.97	**8901**	8901.0	13	0.37	0.95	**8901**	8901.0	13	0.09	2.05
KroB 100	7	4068	11123	14	45.62	48.35	**9537**	9537.6	22	0.65	0.85	**9537**	9537.0	20	0.71	2.53
	9	133396	9505	12	2112.57	2623.79	**9240**	9240.0	16	0.98	1.55	**9240**	9240.0	15	0.11	2.62
	11	90000*	9049	10	1056.27	2895.35	**8842**	8842.0	15	0.26	0.88	**8842**	8842.0	13	0.10	2.48
KroC 100	7	129545*	10367	15	3391.82	4212.98	**9723**	9723.0	17	0.38	0.99	**9723**	9723.0	17	0.26	2.55
	9	5028	9952	12	35.91	52.25	**9171**	9171.0	13	0.21	1.56	**9171**	9171.0	13	0.10	2.42
	11	75987*	9150	10	1389.84	2482.00	**8632**	8632.0	13	0.26	0.92	**8632**	8632.0	13	0.10	2.44
KroD 100	7	1392	11085	14	10.29	15.58	**9626**	9629.0	20	0.56	0.96	**9626**	9626.0	20	0.19	2.36
	9	700	10564	11	6.18	7.74	**8885**	8885.0	13	0.65	1.69	**8885**	8885.0	13	0.11	2.73
	11	85147*	9175	10	968.39	2761.51	**8725**	8725.0	13	0.24	0.88	**8725**	8725.0	13	0.11	2.54
KroE 100	7	92414*	11323	15	1971.32	3075.58	**10150**	10154.8	19	0.17	1.00	**10150**	10150.0	19	0.13	2.16
	9	85305*	9095	12	1918.72	2764.70	8992	8995.2	14	0.26	1.57	**8991**	8991.0	14	0.11	2.39
	11	70807*	8936	10	609.81	2335.43	**8450**	8450.0	14	0.37	0.88	**8450**	8450.0	13	0.12	2.64
Rd 100	7	2520	4105	14	24.43	4196.23	**3461**	3463.0	18	0.16	1.00	**3461**	3485.6	18	0.66	2.30
	9	95242*	3414	12	1798.14	3118.93	**3194**	3195.4	21	0.93	1.66	**3194**	3194.0	16	0.13	2.31
	11	1291	3453	10	8.60	22.11	**2922**	2922.0	13	0.03	0.88	**2922**	2922.0	13	0.09	1.97
KroA150	7	97785*	12367	22	2252.50	3499.43	11426	11517.4	27	0.98	1.62	**11423**	11800.2	28	1.24	4.10
	9	69377*	11955	17	2454.99	2477.69	10072	10072.0	23	0.56	1.92	**10056**	10062.4	26	1.35	3.53
	11	169846*	10564	15	5483.07	5518.26	**9439**	9440.6	21	0.32	1.20	**9439**	9439.0	21	0.61	3.46
KroB 150	7	14400	12876	21	196.85	270.94	11493	11577.6	30	1.31	1.60	**11457**	11491.2	30	1.55	4.30
	9	137763*	11774	18	2760.03	4572.81	**10121**	10130.0	24	0.95	2.12	**10121**	10121.0	24	0.40	3.50
	11	1431	10968	14	26.64	46.96	**9611**	9648.6	22	0.81	1.23	**9611**	9611.0	21	0.15	3.69
KroA 200	7	53686*	14667	28	537.60	1170.37	13293	13327.5	34	1.42	2.67	**13285**	13666.4	35	0.43	5.64
	9	64763*	12683	23	1504.07	1628.36	11726	11726.2	28	2.49	3.88	**11708**	11716.8	28	2.11	5.12
	11	29668*	12736	19	398.25	671.55	10837	10837.0	28	1.15	1.78	**10748**	10848.6	29	1.43	4.61
KroB 200	7	107208*	14952	29	365.08	3351.89	13101	13197.6	36	1.52	2.67	**13100**	13511.6	36	0.10	5.42
	9	38218*	13679	23	637.66	805.04	**11900**	11864.2	29	0.04	0.15	**11900**	11964.8	31	0.80	5.11
	11	67896*	12265	20	493.64	1410.60	**10676**	10678.4	29	0.96	1.83	**10676**	10809.6	30	1.02	5.01
Avg		55896	9808.02	13	1017.94	1626.68	9029.6	9037.5	17	0.51	1.34	9026.0	9060.5	17	0.35	2.56
No.Best			1				39					**48**				

Table 4. Comparison of LS1 and LS2 on Binary GCSP

Instance	NC	LS1					LS2				
		Best Cost	Avg. Cost	NB	Avg. TB	Avg. TT	Best Cost	Avg. Cost	NB	Avg. TB	Avg. TT
Eil 51	7	**1190**	1210.4	19	0.32	1.28	**1190**	1190.0	19	0.29	1.59
	9	**991**	991.8	15	0.97	1.58	**991**	999.6	15	0.35	1.57
	11	**844**	845.2	13	0.23	1.04	**844**	853.6	13	0.20	1.55
Berlin 52	7	**5429**	5429.0	17	0.37	1.24	**5429**	5619.8	17	0.26	1.57
	9	**4807**	4807.0	14	0.19	1.04	**4807**	4807.0	14	0.06	1.53
	11	**4590**	4590.0	13	0.24	1.05	**4590**	4590.0	13	0.09	1.31
St 70	7	**1831**	1831.0	29	0.78	2.13	1837	1837.6	29	0.58	1.97
	9	1461	1461.0	22	0.17	1.59	**1460**	1466.4	22	0.19	2.05
	11	**1268**	1268.0	19	0.71	1.35	**1268**	1273.2	19	0.14	2.00
Eil 76	7	**1610**	1612.6	26	0.95	1.91	1651	1652.6	27	0.63	2.25
	9	**1278**	1294.6	20	0.90	1.58	1297	1302.8	21	1.05	2.46
	11	**1117**	1117.0	18	0.12	2.21	**1117**	1122.6	18	0.56	2.13
Pr 76	7	**66455**	66616.6	29	1.19	1.79	**66455**	66455.0	29	0.29	2.39
	9	**62494**	62524.0	25	0.87	1.59	62907	63155.8	25	0.27	2.36
	11	**52175**	52175.0	19	0.59	1.33	**52175**	52303.6	19	0.19	1.97
Rat 99	7	**2341**	2345.8	34	1.93	2.51	**2341**	2345.2	34	0.99	4.36
	9	1938	1942.6	27	1.36	2.21	**1934**	1950.4	27	1.08	3.50
	11	1712	1713.2	24	0.82	2.64	**1683**	1709.2	23	0.69	3.52
KroA 100	7	**14660**	14660.0	41	1.68	2.89	**14660**	14790.4	41	0.87	5.25
	9	**12974**	12974.0	33	0.49	2.00	**12974**	12974.0	33	0.32	0.16
	11	**11942**	11950.8	29	1.91	2.32	**11942**	12118.6	29	0.39	3.46
KroB 100	7	14390	14469.8	42	1.65	2.83	**14323**	14511.0	44	1.84	5.32
	9	12223	12223.0	34	0.62	2.11	**12222**	12225.0	33	0.73	3.43
	11	**11276**	11277.2	28	0.92	2.10	**11276**	11276.0	28	1.04	3.50
KroC 100	7	**13830**	13831.4	41	0.78	2.85	**13830**	13830.0	41	0.57	5.30
	9	**12149**	12154.0	33	0.76	1.85	**12149**	12197.2	33	0.45	3.26
	11	**11032**	11032.0	26	0.10	2.02	**11032**	11032.0	26	0.39	3.09
KroD 100	7	**13567**	13584.8	38	1.92	2.75	**13567**	13917.0	38	1.52	5.31
	9	12413	12414.0	32	0.84	2.24	**12409**	12437.4	31	0.65	3.62
	11	11486	11491.0	28	1.50	1.90	**11443**	11483.8	29	0.33	3.63
KroE 100	7	**15321**	15333.0	41	1.96	3.04	15357	15426.6	40	0.50	3.20
	9	**12482**	12482.0	32	0.84	4.64	**12482**	12482.0	32	0.36	3.33
	11	**11429**	11461.4	29	0.63	1.81	11487	11487.0	29	0.85	3.69
Rd 100	7	**6174**	6188.6	37	2.08	2.77	6209	6243.0	38	0.78	3.47
	9	5469	5498.2	29	1.55	5.03	**5454**	5596.8	29	0.58	3.42
	11	**4910**	4940.6	28	0.77	1.84	**4910**	4914.8	28	0.58	3.29
KroA150	7	**17214**	17221.0	56	2.68	6.15	17258	17338.6	55	1.98	5.62
	9	15030	15067.4	46	3.32	9.95	**15007**	15280.6	46	1.49	6.09
	11	**13666**	13666.0	40	2.85	3.67	13743	13987.8	40	1.68	5.48
KroB 150	7	17712	17778.8	59	6.05	6.44	**17700**	18017.8	60	2.59	5.47
	9	**15472**	15532.0	50	3.63	9.55	15505	15940.2	50	1.29	5.85
	11	**13719**	13727.2	44	2.41	4.81	13738	13825.0	41	1.72	6.02
KroA 200	7	**21410**	21476.0	78	8.95	13.88	21433	21493.6	79	2.87	7.60
	9	**17912**	18004.2	57	7.44	0.26	18131	18292.4	60	1.92	7.69
	11	**16478**	16565.6	58	6.68	8.73	16556	16729.6	55	2.77	7.18
KroB 200	7	20864	20973.6	79	7.01	15.89	**20769**	20973.0	79	1.64	7.80
	9	18284	18361.0	66	7.10	0.27	**18249**	18580.6	67	1.65	7.75
	11	16078	16108.0	55	6.83	7.29	**16024**	16152.0	55	4.42	6.84
Avg		13022.9	13046.3	35	2.06	3.42	13037.8	13128.9	35	0.97	3.86
No.Best		**35**					34				

Table 5. Comparison of LS1 and LS2 on Integer GCSP without overnights

Instance	NC	LS1						LS2					
		Best Cost	Avg. Cost	NB	NR	Avg. TB	Avg. TT	Best Cost	Avg. Cost	NB	NR	Avg. TB	Avg. TT
Eil 51	7	**1185**	1185.0	19	1	0.87	3.07	**1185**	1187.0	19	1	1.46	5.43
	9	**991**	991.2	15	0	0.86	2.56	**991**	994.8	15	0	1.54	4.64
	11	**843**	843.8	13	1	0.14	2.27	**843**	843.2	13	1	0.96	4.49
Berlin 52	7	**5429**	5429.0	17	0	0.17	2.65	**5429**	5603.6	17	0	0.76	4.64
	9	**4785**	4785.0	15	1	0.45	2.70	**4785**	4807.6	15	1	0.09	3.99
	11	**4590**	4599.6	13	0	0.90	2.26	**4590**	4599.6	13	0	0.10	3.51
St 70	7	**1778**	1778.0	28	3	3.36	5.05	**1778**	1779.6	28	3	0.34	7.13
	9	1461	1461.0	22	0	1.25	4.17	**1460**	1464.2	22	0	0.95	6.14
	11	**1241**	1251.8	18	1	0.55	3.27	**1241**	1253.8	18	1	1.16	5.58
Eil 76	7	**1600**	1600.6	26	1	3.46	4.73	**1600**	1627.4	26	1	1.72	6.49
	9	**1268**	1274.0	20	0	2.18	3.38	**1268**	1285.8	20	0	2.80	6.80
	11	**1117**	1117.0	18	0	0.06	3.65	**1117**	1128.2	18	0	0.66	6.39
Pr 76	7	65721	65902.4	28	1	1.61	7.07	**64111**	64286.0	29	4	0.39	7.70
	9	55129	57078.0	26	4	1.36	3.91	**54859**	55137.6	27	5	0.45	6.33
	11	49559	50478.2	21	1	1.25	3.09	**49445**	49589.0	21	3	1.82	5.62
Rat 99	7	**2311**	2314.4	33	1	4.80	6.24	**2311**	2335.8	33	1	2.41	10.10
	9	1937	1939.4	27	0	4.43	7.53	**1936**	1951.8	27	0	0.48	9.25
	11	**1683**	1700.4	23	0	2.02	3.73	1686	1704.0	23	0	0.40	9.28
KroA 100	7	**14660**	14660.0	41	0	1.68	10.20	**14660**	14721.8	41	0	0.60	11.73
	9	**12974**	12974.0	33	0	0.30	5.75	**12974**	12974.0	33	0	2.14	10.22
	11	**11737**	11737.0	31	1	2.44	5.87	11737	11737.0	29	1	0.69	9.51
KroB 100	7	**14203**	14284.8	43	3	4.92	7.41	14209	14288.8	43	2	2.79	12.77
	9	**12172**	12200.4	35	2	1.43	4.57	12200	12200.0	34	3	3.68	10.36
	11	**11268**	11268.0	27	2	1.99	4.63	**11268**	11468.4	27	2	4.07	9.90
KroC 100	7	**13525**	13671.4	42	5	3.74	6.53	13830	13938.0	41	0	0.76	12.53
	9	**12119**	12119.0	33	1	1.92	4.85	12165	12165.0	33	1	0.59	10.71
	11	**11032**	11032.0	26	0	1.01	5.46	**11032**	11042.6	26	0	1.12	8.53
KroD 100	7	**13501**	13501.0	39	2	1.70	6.51	**13501**	13621.8	39	2	2.36	13.06
	9	**12257**	12260.2	31	1	2.19	5.26	12261	12279.2	31	1	0.76	10.97
	11	11450	11467.0	30	1	2.59	4.69	**11409**	11434.4	30	1	2.42	9.98
KroE 100	7	**15308**	15327.4	43	1	3.38	6.68	15357	15633.0	40	0	3.12	11.44
	9	**12482**	12482.2	32	0	0.37	5.21	**12482**	12484.0	32	0	0.63	9.97
	11	11367	11401.2	28	1	2.27	4.14	**11344**	11462.0	30	1	1.89	9.41
Rd 100	7	**6078**	6104.2	37	2	3.55	6.91	6147	6256.4	36	1	1.09	10.87
	9	**5384**	5391.0	31	2	3.62	7.70	**5384**	5443.0	30	2	4.54	10.15
	11	**4853**	4864.8	29	2	2.74	4.07	**4853**	4901.6	29	1	0.33	8.74
KroA150	7	**16947**	16959.0	57	4	5.71	14.94	**16947**	17226.8	57	4	8.54	18.99
	9	**15007**	15025.8	46	0	5.61	11.25	**15007**	15080.8	46	0	2.96	18.09
	11	**13580**	13608.8	40	2	3.77	8.75	**13580**	13811.0	40	2	3.88	14.99
KroB 150	7	**17620**	17714.8	58	1	9.08	14.94	17621	17832.0	59	1	4.36	16.85
	9	**15332**	15332.0	49	3	7.47	11.51	**15332**	15493.4	48	3	6.90	16.71
	11	**13554**	13596.8	41	2	4.30	9.04	**13554**	13623.6	41	2	3.48	15.38
KroA 200	7	**20979**	21168.8	77	3	16.19	30.16	21115	21429.6	78	4	4.17	24.25
	9	**17657**	17901.4	62	2	12.22	21.73	17867	18291.4	63	4	9.68	22.23
	11	**16280**	16298.0	57	3	14.08	17.98	16453	16663.4	52	2	3.49	20.15
KroB 200	7	**20636**	20719.2	78	3	18.88	31.28	20817	21198.2	81	3	8.01	24.12
	9	18230	18324.6	66	1	17.42	23.12	**18124**	18251.4	66	6	7.86	22.22
	11	**15725**	15860.6	55	3	11.14	17.32	15801	16115.0	57	5	4.67	19.74
Avg		12719.7	12812.2	35	1	4.11	8.12	12701.4	12805.1	35	2	2.50	11.21
No.Best		**40**						33					

Table 6. Comparison of LS1 and LS2 on Integer GCSP with overnights

Instance	NC	LS1						LS2					
		Best Cost	Avg. Cost	NB	ON	Avg. TB	Avg. TT	Best Cost	Avg. Cost	NB	ON	Avg. TB	Avg. TT
Eil 51	7	**1146**	1846.0	19	10	0.39	6.84	**1146**	1146.0	19	10	0.53	4.05
	9	**958**	964.4	15	5	1.25	6.16	**958**	973.4	15	5	1.39	4.10
	11	**827**	827.0	13	4	2.08	5.44	**827**	835.0	13	4	1.41	3.86
Berlin 52	7	**4966**	4966.0	18	9	0.15	4.65	**4966**	4972.2	18	9	0.31	3.92
	9	**4272**	4274.2	17	8	2.09	4.79	**4272**	4295.4	16	7	0.28	3.41
	11	**3962**	3962.0	14	8	2.03	5.71	**3962**	3962.0	14	8	0.31	3.11
St 70	7	**1654**	1654.2	27	15	5.14	10.46	**1654**	1656.2	27	14	1.08	5.89
	9	1417	1437.0	22	10	0.70	9.48	**1416**	1439.2	22	9	1.74	5.37
	11	**1196**	1200.2	18	6	3.93	7.94	**1196**	1197.4	18	6	0.86	5.14
Eil 76	7	**1554**	1558.8	26	10	4.72	14.77	1561	1599.2	26	5	1.07	6.47
	9	**1284**	1296.4	21	7	6.27	11.83	**1284**	1314.0	21	4	1.02	5.71
	11	1108	1111.6	18	5	7.37	11.39	**1107**	1109.0	18	3	1.10	5.77
Pr 76	7	52427	53163.6	32	17	4.80	10.92	**52265**	53153.8	31	17	1.19	6.16
	9	**47092**	47764.0	29	17	3.02	7.11	47234	47609.4	28	16	1.56	5.60
	11	**44028**	44028.0	20	10	1.63	6.00	**44028**	44028.0	20	10	1.34	4.57
Rat 99	7	2233	2242.0	32	11	7.53	11.14	**2218**	2242.8	32	8	6.14	9.58
	9	**1895**	1901.6	27	8	7.98	11.64	1922	1942.8	28	8	2.62	8.59
	11	1673	1676.8	24	9	7.80	10.19	**1650**	1678.6	23	5	2.21	8.22
KroA 100	7	**12006**	12182.8	43	26	10.38	18.91	**12006**	12096.6	43	26	1.20	8.32
	9	11653	11733.4	33	18	10.33	17.68	**11185**	11259.4	36	23	0.83	7.84
	11	**10665**	10688.0	31	18	9.81	16.83	10701	10739.2	30	18	0.41	7.27
KroB 100	7	12373	12605.4	41	23	8.48	12.72	**12363**	12665.6	45	28	0.89	9.43
	9	11157	11200.2	35	20	5.01	11.65	**11128**	11320.0	35	21	2.17	9.12
	11	**10302**	10302.0	29	16	4.06	10.32	**10302**	10380.6	28	14	3.76	8.28
KroC 100	7	12223	12224.4	41	22	9.09	15.87	**11996**	12060.8	45	28	1.50	8.55
	9	11116	11160.8	33	17	7.54	15.09	**11084**	11142.2	35	20	2.06	7.69
	11	**10299**	10330.2	28	15	3.29	11.61	10421	10463.0	29	17	0.36	6.72
KroD 100	7	11725	11857.6	39	21	4.92	12.99	**11725**	11849.2	39	21	2.37	9.06
	9	11039	11108.8	30	17	7.09	13.19	**10787**	10787.0	34	21	2.45	7.80
	11	**10352**	10405.2	30	18	8.57	12.18	10426	10447.6	30	17	0.60	7.57
KroE 100	7	13273	13273.0	43	26	7.49	15.36	**12438**	12696.4	45	26	4.80	8.33
	9	**10821**	10926.2	34	20	8.08	14.79	**10821**	10906.2	34	20	1.53	7.13
	11	10010	10065.8	30	17	6.32	9.27	**10007**	10094.2	29	17	1.55	6.85
Rd 100	7	5634	5654.4	37	17	12.53	14.43	**5567**	5681.4	39	20	3.32	8.63
	9	**4950**	5034.4	32	18	9.68	9.78	4974	5059.4	32	16	0.71	7.79
	11	4514	4519.8	27	12	7.31	9.48	**4462**	4563.8	29	16	2.91	7.15
KroA150	7	15335	15381.4	57	28	19.20	25.73	**14936**	15293.2	60	34	4.47	14.61
	9	13484	13533.4	50	27	17.08	22.29	**13115**	13386.0	52	30	5.21	14.17
	11	**12112**	12300.4	41	18	10.51	17.64	12211	12384.2	44	22	2.91	11.89
KroB 150	7	15242	15747.4	59	29	21.14	27.39	**15162**	15353.8	62	33	6.49	14.79
	9	13242	13313.8	50	38	14.18	24.48	**13198**	13519.4	52	30	3.07	12.71
	11	12418	12418.0	43	24	15.26	20.85	**12183**	12379.4	46	25	5.21	11.95
KroA 200	7	17839	17994.0	77	37	40.08	54.89	**17683**	18222.0	80	44	8.23	19.80
	9	15563	15726.6	64	34	26.26	40.67	**15523**	15760.6	65	36	6.99	17.32
	11	14430	14667.8	57	34	20.14	33.16	**14302**	14518.4	59	37	6.09	15.02
KroB 200	7	18421	18488.8	80	47	55.55	68.05	**17873**	18117.6	84	49	4.98	19.71
	9	**15935**	16176.8	64	36	32.55	42.62	16069	16164.2	69	37	5.11	17.49
	11	14235	14323.6	54	28	22.79	31.49	**14064**	14455.2	58	32	6.79	15.72
Avg		11167.92	11275.38	36	19	10.49	16.83	11091.21	11227.52	37	19	2.61	8.92
No.Best		22						**39**					

Table 7. Comparison of LS1 and LS2 on Steiner CSP

Instance	LS1					LS2				
	Best Cost	Avg. Cost	NB	Avg. TB	Avg. TT	Best Cost	Avg. Cost	NB	Avg. TB	Avg. TT
Eil 51	**163**	163.0	8	0.03	0.79	**163**	163.0	8	0.05	1.04
	159	159.0	8	0.08	0.79	**159**	159.0	9	0.04	0.75
	147	147.0	7	0.06	0.79	**147**	147.0	7	0.04	0.87
Berlin 52	**3470**	3883.2	10	0.46	0.76	**3470**	3470.0	10	0.13	0.97
	3097	3097.4	7	0.27	0.89	**3097**	3097.0	7	0.04	0.80
	2956	2956.0	6	0.11	1.01	**2956**	2956.0	6	0.04	0.79
St 70	**287**	287.4	12	0.14	0.96	**287**	287.0	12	0.07	1.31
	259	259.0	10	0.01	0.89	**259**	259.0	9	0.06	1.60
	245	245.0	12	0.70	1.56	**245**	245.0	10	0.05	1.23
Eil 76	**207**	207.0	15	0.06	0.95	**207**	207.0	15	0.07	1.32
	185	185.4	10	0.35	1.06	**185**	185.0	12	0.06	1.51
	169	169.0	11	0.13	0.78	**169**	169.0	9	0.06	1.43
Pr 76	**49773**	49773.0	13	0.10	0.97	**49773**	49773.0	13	0.15	1.86
	44889	44889.0	12	0.11	1.04	**44889**	44889.0	12	0.09	1.65
	42950	42998.8	9	0.30	0.85	**42950**	42950.0	9	0.06	1.34
Rat 99	**482**	482.0	18	0.33	1.23	**482**	482.0	18	0.43	2.63
	454	454.0	14	0.34	1.18	**454**	454.0	14	0.15	2.63
	444	444.6	12	0.37	1.11	**444**	444.0	12	0.09	2.18
KroA 100	**9545**	9545.0	19	0.37	1.30	**9545**	9545.0	18	0.52	2.71
	9112	9112.0	16	0.44	1.03	**9112**	9112.0	15	0.12	2.30
	8833	8833.0	13	0.10	1.11	**8833**	8833.0	13	0.09	2.01
KroB 100	**9536**	9536.0	21	0.79	1.21	**9536**	9536.0	19	0.48	2.66
	9199	9230.6	17	0.51	0.95	**9199**	9199.0	15	0.10	2.49
	8763	8763.0	11	0.11	1.14	**8763**	8763.0	11	0.10	2.29
KroC 100	9591	9591.0	15	0.12	1.07	**9590**	9590.0	16	0.18	2.68
	9171	9171.0	13	0.34	1.05	**9171**	9171.0	13	0.19	2.35
	8632	8632.0	13	0.27	1.16	**8632**	8632.0	13	0.12	2.48
KroD 100	**9526**	9526.0	19	0.31	1.35	**9526**	9526.0	19	0.28	2.33
	8885	8885.0	16	0.30	0.98	**8885**	8885.0	13	0.12	2.76
	8725	8725.0	13	0.26	1.12	**8725**	8725.0	13	0.11	2.44
KroE 100	**9800**	9800.0	16	0.11	1.15	**9800**	9800.0	16	0.12	2.32
	8987	8987.0	13	0.09	0.98	**8986**	8986.0	14	0.10	2.35
	8450	8450.0	13	0.10	1.23	**8450**	8450.0	13	0.11	2.54
Rd 100	**3412**	3412.4	18	0.28	1.25	**3412**	3445.6	18	0.15	2.31
	3194	3194.0	17	0.32	0.93	**3194**	3194.0	16	0.32	2.24
	2761	2761.0	12	0.04	1.20	**2761**	2761.0	12	0.08	1.91
KroA150	**10939**	10939.6	28	0.68	1.84	**10939**	10999.0	27	1.00	3.86
	9808	9809.4	23	0.49	1.29	**9808**	9808.0	23	0.24	3.55
	9360	9367.2	20	1.33	1.92	**9360**	9360.0	21	0.38	3.52
KroB 150	11260	11299.8	29	1.41	1.73	**11225**	11231.6	30	0.94	4.06
	10121	10121.8	24	0.81	1.34	**10121**	10121.0	24	0.41	3.54
	9542	9562.8	20	0.79	1.81	**9542**	9542.0	20	0.20	3.84
KroA 200	13042	13085.4	34	1.30	2.71	**13041**	13120.4	32	1.69	5.38
	11392	11405.4	28	1.31	1.98	**11392**	11403.0	27	0.86	4.91
	10535	10535.6	26	1.03	1.68	**10528**	10838.2	26	0.75	4.65
KroB 200	13022	13078.4	36	2.09	2.86	**13020**	13304.2	34	0.46	5.45
	11713	11793.4	30	1.35	2.22	**11712**	11850.6	28	1.21	5.34
	10619	10563.8	30	1.22	1.82	**10614**	10679.8	28	1.00	4.74
Avg	8912.7	8927.4	17	0.47	1.27	8911.6	8932.2	16	0.29	2.54
No.Best	40					**48**				

Table 8. Comparison of LS1 and LS2 on large binary GCSP instances with varying node costs

Instance	Range	LS 1					LS2				
		Best Cost	Avg. Cost	NB	Avg. TB	Avg. TT	Best Cost	Avg. Cost	NB	Avg. TB	Avg. TT
ts225	A	**169425**	170633.6	65	11.7	17.8	169607	170402.8	64	1.7	6.0
	B	81470	81674.4	66	9.3	13.5	**80844**	82654.0	66	2.1	6.8
	C	72493	73388.8	67	9.8	13.5	**72343**	73266.4	70	1.4	6.6
a280	A	3497	3508.6	76	19.4	27.2	**3456**	3467.8	78	3.8	8.6
	B	1519	1538.0	79	12.2	22.9	**1505**	1533.6	81	5.8	9.1
	C	1237	1251.6	83	17.4	22.9	**1225**	1247.8	84	3.2	8.5
pr299	A	69521	69870.6	85	25.4	28.7	**68958**	69190.4	84	5.6	9.2
	B	29871	30193.6	85	19.1	26.4	**29855**	30216.6	82	4.0	9.7
	C	25100	25313.2	85	19.7	26.5	**25010**	25340.6	87	5.3	9.4
lin318	A	68151	68465.8	90	20.0	36.7	**67288**	67962.2	89	2.6	9.5
	B	**27839**	27885.4	90	19.1	32.2	27965	28214.8	93	4.3	9.6
	C	22600	22790.4	95	24.6	31.5	**22527**	22786.6	96	3.6	9.8
pr439	A	232359	233265.0	124	58.7	87.0	**228721**	229267.4	127	7.3	15.3
	B	82288	82658.6	128	41.9	83.5	**81039**	81303.4	128	6.9	15.0
	C	62084	62604.8	134	59.7	83.0	**61908**	62719.0	133	10.1	15.2
att532	A	170276	172554.8	142	86.1	150.9	**169357**	169963.0	143	8.0	19.0
	B	57937	58263.4	147	110.8	142.5	**56956**	57560.4	145	16.3	19.8
	C	43237	43709.6	150	106.0	149.0	**43124**	43582.4	149	9.5	20.5
ali535	A	4703	4754.8	145	92.6	164.0	**4684**	4694.0	145	10.5	20.3
	B	1546	1551.0	145	113.7	148.7	**1522**	1528.6	145	9.3	18.6
	C	1084	1136.4	152	112.8	149.0	**1082**	1094.8	158	10.4	17.8
u574	A	84623	85052.2	168	124.8	210.5	**83652**	83767.2	168	12.1	23.0
	B	27148	27224.0	171	108.6	190.2	**26792**	26852.6	176	14.0	23.0
	C	19448	19718.2	171	159.8	189.4	**19227**	19379.2	177	16.7	23.0
rat575	A	13016	13115.4	161	136.0	210.4	**12887**	12946.0	164	12.5	22.9
	B	4139	4198.2	159	142.7	189.1	**4119**	4152.6	158	16.7	21.8
	C	2953	3015.4	162	175.7	185.0	**2935**	3015.6	175	17.4	22.1
p654	A	114755	115350.0	180	174.2	306.9	**113355**	113694.0	182	21.3	30.8
	B	34146	34212.0	184	171.3	286.9	**33723**	33795.8	182	17.0	27.8
	C	25141	25184.8	193	129.7	282.3	**24858**	25170.8	195	14.4	25.5
d657	A	125916	127438.6	184	154.8	297.9	**125211**	125343.8	184	13.4	27.5
	B	37816	38356.8	181	90.9	277.7	**37434**	37926.6	184	19.7	26.9
	C	26378	26771.8	195	217.8	273.1	**26209**	26644.2	188	19.7	27.6
gr666	A	8338	8482.8	187	199.5	317.9	**8324**	8339.0	186	13.1	28.1
	B	2527	2537.8	187	273.3	314.9	**2492**	2495.6	189	20.3	25.0
	C	1679	1691.8	194	264.2	289.7	**1653**	1675.4	199	14.9	25.5
u724	A	101048	102655.0	197	92.3	397.2	**100284**	100547.0	202	12.5	30.8
	B	29358	29534.0	196	119.0	384.2	**29099**	29310.8	198	20.3	30.2
	C	20203	20788.2	197	299.9	362.0	**20163**	20558.4	210	14.7	31.0
rat783	A	21450	21699.6	211	398.5	518.5	**21272**	21307.6	213	15.6	35.2
	B	6085	6127.0	213	401.5	504.1	**6034**	6051.0	216	26.6	33.5
	C	4109	4160.6	218	320.0	479.9	**4054**	4089.4	229	20.2	33.4
dsj1000	A	68089542	68442158.0	280	580.1	693.3	**67653673**	67769758.6	283	49.6	72.8
	B	16728234	16954235.0	281	500.1	680.2	**16525509**	16629826.4	285	45.8	56.1
	C	10384875	10518746.0	292	413.5	605.5	**10265949**	10376224.8	295	47.5	57.5
Avg.		2158692.5	2174877.0	155	147.5	220.1	2141508.5	2149130.4	157	13.95	22.78
No. Best		2					**43**				

Table 9: Comparison of LS1 and LS2 on large GCSP without overnight instances with varying node costs

Instance	Range	LS1						LS2					
		Best Cost	Avg. Cost	NB	NR	Avg. TB	Avg. TT	Best Cost	Avg. Cost	NB	NR	Avg. TB	Avg. TT
ts225	A	170859	171026.8	64	4	15.0	27.8	**169887**	170929.8	64	4	7.8	17.7
	B	83290	83457.4	66	0	11.4	25.4	**83016**	84221.8	64	2	8.6	18.2
	C	72322	73054.4	67	0	15.0	25.6	**72242**	74070.0	68	1	19.8	19.4
a280	A	3532	3561.6	77	4	33.5	45.6	**3497**	3525.0	79	5	7.8	24.1
	B	1541	1567	78	0	27.2	41.1	**1533**	1552.2	78	1	16.2	25.4
	C	1242	1266.2	84	0	34.2	42.2	**1225**	1274.8	89	1	20.4	24.0
pr299	A	69803	70228.4	83	4	38.2	51.5	**69084**	69451.2	36	7	5.6	24.0
	B	30541	30714	83	1	27.1	46.9	**29975**	30442.0	85	1	6.7	26.3
	C	25320	25545.4	83	0	33.1	46.1	**25211**	25471.4	87	1	11.0	27.4
lin318	A	64081	64541	92	12	54.6	72.5	**63820**	64043.4	93	12	10.4	27.4
	B	27315	27499.6	92	5	33.1	56.5	**27109**	27449.2	98	14	12.6	26.9
	C	22481	22637.6	98	5	48.1	56.0	**22303**	22441.8	99	9	19.5	28.2
pr439	A	216792	217664.41	128	18	125.3	171.1	**215284**	215969.2	130	23	24.6	41.6
	B	79204	80295.4	129	5	112.3	159.1	**78779**	79161.8	129	11	23.0	40.4
	C	61520	62296.6	128	2	122.0	156.3	**60607**	61076.4	134	5	22.6	43.1
att532	A	169494	170010.41	142	18	221.3	303.1	**166189**	166651.2	144	12	25.3	53.8
	B	57427	57859.2	144	3	223.5	278.4	**57411**	57996.2	148	4	41.2	50.6
	C	42995	43717.8	146	3	218.2	282.5	**42551**	43167.2	156	6	26.5	53.0
ali535	A	4432	4456.6	144	24	248.4	319.1	**4396**	4431.0	147	30	76.0	76.0
	B	1527	1534.4	142	4	252.2	279.5	**1503**	1516.6	148	10	12.2	46.2
	C	1094	1107.6	153	10	202.6	261.2	**1070**	1084.6	204	35	39.8	50.1
u574	A	81485	82068.6	170	22	434.0	455.4	**80016**	80217.4	174	33	16.3	59.8
	B	26466	26618	169	8	329.5	398.8	**26305**	26461.4	177	9	40.0	60.7
	C	18885	19537.8	175	5	254.9	393.7	**18794**	19002.8	183	13	61.1	63.0
rat575	A	12683	12735	160	16	360.4	437.0	**12330**	12365.8	164	24	40.1	56.2
	B	4109	4145.2	162	3	352.9	391.4	**4086**	4119.0	162	2	35.3	55.5
	C	2993	3026.8	164	0	265.9	395.0	**2927**	2966.4	169	1	44.0	55.5
p654	A	110156	110240.8	187	31	715.4	888.4	**107189**	107689.4	187	43	98.6	104.9
	B	33702	33942.2	179	9	466.6	631.7	**33663**	33861.0	187	12	54.8	72.8
	C	**24260**	24629.2	194	6	517.2	646.7	24680	24780.8	199	2	58.7	69.8
d657	A	122409	123344.4	181	27	542.5	631.4	**118956**	119302.0	185	39	47.1	68.1
	B	37647	37867.8	182	2	529.8	602.1	**37078**	37505.2	184	1	41.4	68.9
	C	26544	27463	184	1	546.4	570.2	**26177**	26506.2	194	5	41.6	70.4
gr666	A	7999	8037.2	185	25	594.5	689.0	**7805**	7872.4	193	27	22.6	69.1
	B	2418	2447.8	188	11	566.8	670.9	**2417**	2434.8	190	10	46.5	64.5
	C	1660	1701.8	194	8	288.9	600.1	**1636**	1650.8	220	11	42.5	63.6
u724	A	102454	102945.8	204	27	822.1	929.2	**100132**	100690.0	205	30	33.9	79.0
	B	29643	29909.4	200	2	699.8	856.2	**29257**	29582.2	202	6	41.3	74.9
	C	20542	20633.2	203	5	558.5	809.9	**20390**	20656.0	208	8	61.6	77.4
rat783	A	20623	20805.8	192	29	924.9	1057.1	**20150**	20208.4	214	30	41.8	84.4
	B	6013	6084	212	8	814.2	1015.8	**5905**	5965.0	215	8	61.3	81.7
	C	4128	4153.4	225	5	719.0	950.5	**4030**	4053.8	229	4	57.9	82.8
dsj1000	A	64021453	64572154.2	276	58	1004.1	1378.4	**63550808**	63803980.8	288	61	206.2	235.1
	B	16801430	17051243.6	281	11	924.7	1242.3	**16500781**	16546597.2	278	12	96.3	145.5
	C	10524851	10689214	301	6	820.2	1151.3	**10395420**	10493207.0	299	7	111.8	137.2
Average		2072252.6	2093977.6	155	10	358.9	456.4	2051725.0	2060835.6	160	13	40.9	61.0
No. Best		1						**44**					

Table 10. Comparison of LS1 and LS2 on large GCSP with overnight instances with varying node costs

Instance	Range	LS1						LS2					
		Best Cost	Avg. Cost	NB	NR	Avg. TB	Avg. TT	Best Cost	Avg. Cost	NB	NR	Avg. TB	Avg. TT
ts225	A	144953	145012.4	67	28	16.1	24.2	**143495**	145716.8	71	37	2.3	15.0
	B	73412	75075.4	74	38	17.3	21.9	**73285**	74478.0	71	37	7.6	16.2
	C	63552	66184.2	76	44	15.1	23.5	**63075**	64390.8	73	43	7.9	16.3
a280	A	2839	2869.2	84	31	34.8	42.6	**2817**	2825.2	88	46	2.5	20.1
	B	1335	1369.8	81	28	22.9	32.4	**1329**	1336.4	89	43	6.6	21.8
	C	1091	1109.6	87	31	20.8	28.8	**1072**	1093.0	89	32	10.0	22.8
pr299	A	56842	57183.0	87	32	28.4	47.9	**56760**	57019.2	93	47	12.8	21.8
	B	25937	26078.2	93	42	33.6	41.3	**25774**	26478.6	97	52	11.6	23.6
	C	22301	22069.0	90	43	28.4	39.1	**22018**	22249.4	93	50	18.1	25.1
lin318	A	55557	55687.2	94	37	46.6	58.1	**55327**	55521.8	99	49	3.8	22.6
	B	23772	24193.0	97	49	31.9	53.3	**23431**	23785.4	100	56	8.9	24.1
	C	19844	20040.2	96	50	41.9	55.8	**19471**	19933.6	101	58	13.8	25.4
pr439	A	184831	185494.4	128	43	131.3	170.5	**183619**	184851.6	138	70	17.0	36.5
	B	69547	71507.0	130	57	114.9	145.7	**68774**	69588.2	140	75	25.0	39.6
	C	54641	56134.0	133	63	97.7	143.4	**54102**	55320.0	144	81	33.8	45.4
att532	A	136371	137644.0	147	45	185.1	226.2	**135280**	136056.2	150	54	30.8	47.1
	B	48943	49748.6	152	60	160.0	197.1	**48098**	48542.6	158	81	21.8	46.7
	C	**37104**	37892.2	161	68	133.8	186.0	37245	37551.4	170	100	43.8	53.4
ali535	A	3714	3848.6	144	39	150.3	226.4	**3672**	3680.6	154	64	19.6	45.3
	B	1338	1373.8	144	38	109.3	189.3	**1321**	1337.0	157	60	22.9	44.1
	C	987	1005.8	152	25	119.6	166.5	**979**	996.0	162	32	42.9	50.7
u574	A	64242	64451.8	171	64	195.8	322.7	**63405**	63747.6	183	89	15.3	49.8
	B	21935	23033.4	169	67	245.8	275.7	**21744**	21913.8	191	102	48.0	59.2
	C	16745	17193.8	179	86	184.9	265.8	**16441**	16638.2	190	108	58.6	62.5
rat575	A	10295	10392.4	161	46	218.9	297.5	**10196**	10239.2	180	82	30.0	53.3
	B	3508	3666.0	161	52	177.9	261.2	**3465**	3521.8	175	83	43.9	55.2
	C	2663	2707.6	175	64	169.3	244.4	**2600**	2619.4	194	96	44.5	58.9
p654	A	95019	100537.4	186	50	361.7	494.2	**94169**	94400.8	194	76	61.5	73.2
	B	30053	30539.6	190	59	170.3	438.0	**29718**	29850.0	198	92	45.1	69.2
	C	22965	23036.8	193	65	248.2	399.4	**22566**	22681.0	215	109	66.3	77.5
d657	A	97053	98678.0	181	58	301.5	424.1	**96632**	96926.6	189	84	46.9	66.4
	B	31322	33283.4	182	74	207.4	387.6	**31153**	31503.0	200	109	53.0	69.0
	C	23308	24737.2	184	75	258.8	375.4	**23120**	23479.4	218	120	65.8	75.1
gr666	A	6601	6771.0	189	49	292.6	436.9	**6505**	6524.8	206	89	36.6	68.3
	B	2128	2148.2	194	64	278.7	417.0	**2088**	2093.6	208	95	43.1	64.8
	C	1492	1533.4	198	41	262.8	353.2	**1465**	1492.4	227	65	45.2	72.7
u724	A	81276	81851.8	204	57	339.3	568.1	**81017**	81290.0	216	93	41.1	76.5
	B	25274	26464.0	198	62	186.3	496.5	**24932**	25082.4	228	117	50.3	72.3
	C	18289	19011.8	209	91	378.1	494.0	**18057**	18113.2	234	138	79.2	85.5
rat783	A	17337	17432.2	215	48	382.5	699.8	**17222**	17303.6	222	81	64.6	88.7
	B	5130	5168.8	213	56	362.6	642.4	**5106**	5159.0	241	112	48.1	83.7
	C	3650	3833.4	228	77	486.3	619.9	**3604**	3623.0	247	116	67.4	84.8
dsj1000	A	53016230	52924952.8	293	101	412.6	823.6	**52767955**	52811295.2	302	122	138.6	175.2
	B	13904578	14157851.0	305	101	500.5	803.5	**13714835**	13832973.2	309	151	115.8	154.1
	C	8992587	9038746.0	301	83	401.5	785.9	**8946964**	9010501.0	309	169	179.1	181.0
Average		1722724.2	1727989.8	160	55	190.3	298.8	1711686.7	1717016.09	171	81	41.1	58.7
No. Best		1						**44**					

Table 11. Comparison of computing times of GTSP methods

Computer	Mflops	r	Method
Gateway Profile 4MX	230	1.568	*GA*
Sun Sparc Station LX	4.6	0.032	GI^3, NN
HP 9000/720	2.3	0.016	FST-Lagr, FST-Root, B&C
Unknown	-	1	RACS
Dell Dimension 8400	290	2	mrOX
Pentium(R) IV, 3.4 Ghz	295	2.03	MSA
Ours	145	1	LS2

Table 12. Comparison of LS2 against 8 other heuristics on benchmark GTSP instances in the literature

Instances	LS2		MSA		mrOX		RACS	GA		GI^3		NN		FST-lagr		FST-Root		B&C
	gap	time	gap	time	gap	time	gap	gap	time	gap	time	gap	time	gap	time	gap	time	time
Att48	0	0.4	0	0	0	0.8	-	0	0	-	-	-	-	0	0	0	0	0.0
Gr48	0	0.4	0	0	0	0.6	-	0	0.8	-	-	-	-	0	0	0	0	0.0
Hk48	0	0.5	0	0	0	0.6	-	0	0.4	-	-	-	-	0	0	0	0	0.0
Eil51	0	0.5	0	0	0	0.6	0	0	0.2	0	0	0	0	0	0	0	0	0.0
Brazil58	0	0.6	0	0	0	1.6	-	0	0.4	-	-	-	-	0	0	0	0	0.0
St70	0	0.7	0	0	0	0.8	0	0	0.4	0	0	0	0	0	0	0	0.2	0.2
Eil76	0	0.8	0	0	0	0.8	0	0	0.4	0	0	0	0	0	0	0	0.2	0.2
Pr76	0	0.9	0	0	0	1.0	0	0	0.4	0	0	0	0	0	0	0	0.2	0.2
Rat99	0	1.2	0	0	0	1.0	0	0	1.0	0	0.2	0	0.2	0	0	0	0.8	0.8
KroA100	0	1.2	0	0	0	1.2	0	0	0.6	0	0.2	0	0.2	0	0	0	0.2	0.2
KroB100	0	1.3	0	0	0	1.2	0	0	0.6	0	0.2	0	0	0	0	0	0.4	0.4
KroC100	0	1.2	0	0	0	1.2	0	0	0.4	0	0.2	0	0.2	0	0	0	0.2	0.2
KroD100	0	1.2	0	0	0	1.4	0	0	0.6	0	0.2	0	0.	0	0	0	0.2	0.2
KroE100	0	1.3	0	0	0	1.2	0	0	1.2	0	0.2	0	0	0	0	0	0.2	0.2
Rd100	0	1.2	0	0	0	1.0	0	0	0.4	0.08	0.2	0.08	0.2	0.08	0	0	0.2	0.2
Eil101	0	1.1	0	0	0	1.0	0	0	0.4	0.4	0.2	0.4	0	0	0	0	0.4	0.4
Lin105	0	1.3	0	0	0	1.2	0	0	0.4	0	0.4	0	0.2	0	0	0	0.2	0.2
Pr107	0	1.2	0	0	0	1.0	0	0	0.6	0	0.2	0	0.2	0	0	0	0.2	0.2
Gr120	0	1.1	0	0	0	1.4	-	0	0.8	-	-	-	-	1.99	0	0	0.6	0.6
Pr124	0	1.5	0	0	0	1.4	0	0	1.0	0.43	0.4	0	0.4	0	0	0	0.4	0.4
Bier127	0.04	1.6	0	0	0	1.6	0	0	0.8	5.55	1.0	9.68	0.2	0	0.2	0	0.4	0.4
Pr136	0	1.8	0	0	0	1.6	0	0	0.8	1.28	0.4	5.54	0.2	0.82	0.2	0	0.6	0.6
Pr144	0	1.6	0	0	0	2.0	0	0	0.4	0	0.4	0	0.4	0	0	0	0.2	0.2
KroA150	0	2.1	0	0	0	2.0	0	0	2.0	0	0.6	0	0.6	0	0.2	0	1.4	1.5
KroB150	0	1.9	0	0	0	2.0	0	0	1.6	0	0.4	0	0.6	0	0.2	0	0.8	0.8
Pr152	0	2.0	0	0	0	2.0	0	0	2.4	0.47	0.6	1.8	0.4	0	0.2	0	0.8	1.5

Instances	LS2		MSA		mrOX		RACS	GA		GI^3		NN		FST-lagr		FST-Root		B&C
	gap	time	gap	time	gap	time	gap	gap	time	gap	time	gap	time	gap	time	gap	time	time
U159	0	2.2	0	0	0	2.0	0.01	0	1.0	2.6	0.6	2.79	0.8	0	0.2	0	2.0	2.0
Rat195	0	2.5	0	0.2	0	2.8	0	0	1.0	0	1.2	1.29	2.6	1.87	0.2	0	3.5	3.5
D198	0.32	3.4	0	0	0	3.2	0.01	0	1.8	0.6	1.8	0.6	3.6	0.48	0.2	0	10.8	10.8
KroA200	0	2.8	0	0	0	3.4	0.01	0	4.2	0	0.8	5.25	1.6	0	0.2	0	2.6	2.6
KroB200	0	2.7	0	0	0.05	3.2	0	0	2.2	0	1.0	0	4.0	0.05	0.2	0	3.9	3.9
Ts225	0	2.9	0	4.3	0.14	3.4	0.02	0	3.9	0.61	2.6	0	3.6	0.09	0.2	0.09	18.5	538.2
Pr226	0.09	2.5	0	0	0	3.0	0.03	0	1.6	0	0.8	2.17	2.0	0	0.2	0	1.4	1.4
Gil262	0.79	3.6	0	7.1	0.45	7.2	0.22	0.79	3.0	5.03	3.5	1.88	3.6	3.75	0.2	0.89	20.5	94.2
Pr264	0.59	3.6	0	0	0	4.8	0	0	2.0	0.36	2.0	5.73	4.5	0.33	0.4	0	4.9	4.9
Pr299	0.04	4.5	0	1.4	0.05	9.2	0.24	0.02	9.7	2.23	0.2	2.01	8.5	0	0.4	0	11.6	11.6
Lin318	0.01	4.4	0	0	0	16.2	0.12	0	5.5	4.59	6.2	4.92	9.7	0.36	0.8	0.36	12.0	23.8
Rd400	0.98	6.2	0	0.6	0.58	29.2	0.87	1.37	5.5	1.23	12.2	3.98	34.7	3.16	0.8	2.97	71.5	99.9
Fl417	0.01	5.8	0	0	0.04	16.4	0.57	0.07	3.8	0.48	12.8	1.07	40.8	0.13	1.0	0	237.5	237.5
Pr439	0.09	7.1	0	3.9	0	38.2	0.79	0.23	14.4	3.52	18.4	4.02	37.8	1.42	2.0	0	76.9	77.1
Pcb442	0.16	6.9	0	1.6	0.01	46.8	0.69	1.31	16.0	5.91	17.0	0.22	25.6	4.22	1.2	0.29	76.1	835.1
Average (36)	0.09	2.5	0	0.53	0.04	6.12	0.10	0.10	2.58	0.98	2.42	1.48	5.21	0.46	0.26	0.13	15.61	54.32
# Opt (36)	25		36		29		24	30		19		18		23		31		36
Average (41)	0.08	2.2	0	0.47	0.03	5.38		0.09	2.31					0.46	0.22	0.11	13.72	47.71
# Opt (41)	30		41		34			35						27		36		41