UNIVERSITÀ DEGLI STUDI DI BOLOGNA

Dottorato di Ricerca in

Automatica e Ricerca Operativa

MAT/09

XXII Ciclo

Algorithms for Combinatorial Optimization Problems

Zahra Naji-Azimi

Il Coordinatore	Il Tutor
Prof. Claudio Melchiorri	Prof. Paolo Toth

A.A. 2007-2010

Contents

Acknowledgments

First, I have special thanks for my PhD supervisor, Professor Paolo Toth, who permitted me to be his PhD student. This opportunity is one of the honors in my life. I really appreciate his help and his scientific suggestions and notes for improving the research work. His ideas were always innovative and opening new ways of thinking. He has always been ready to answer my questions. His kind behavior and his help in different fields are very grateful for me. He is a model for me to follow for ever.

I want to thank other professors and PhDs in the Operations Research group of the University of Bologna. In particular, I want to thank Professors Daniele Vigo, Silvano Martello, Alberto Caprara, Andrea Lodi, Roberto Baldacci and Giovanni Marro for their lessons during this PhD period. I also want to thank the PhD coordinator, Professor Claudio Melchiorri.

In particular, I want to thank my colleague Dr. Laura Galli for having a common work. Special thanks go to PhD student Majid Salari for his help and contributions during our common projects. Many thanks go to Dr. Manuel Iori, Dr. Enrico Malaguti, Dr. Valentina Cacchiani, Dr. Andrea Tramontani and two PhD students Victor Vera Valdes and Felipe Navarro, who have been helpful from the beginning of my settle in Bologna until now. They have been very kind and always ready to help.

During my PhD I spent some months at the University of Maryland, College Park, USA. I want to thank Professors Bruce Golden and Raghu S. Raghavan for their contribution during my stay there. They extended my view to new Operations Research problems.

My PhD thesis has been supported by MIUR (Ministero Istruzione, Universitã e Ricerca), Italy, and my stay at the University of Maryland had an additional support called “Marcopolo Program (University of Bologna)”. These supports are gratefully acknowledged.

Bologna March 29, 2010

Zahra Naji-Azimi

List of Tables

List of Algorithms

List of Figures

Keywords

Combinatorial Optimization

Unicost Set Covering Problem

Electromagnetism Metaheuristic

Capacitated *m*-Ring-Star problem

Networks.

Minimum Spanning Tree Problem

Minimum Label Spanning Tree Problem

Mixed Integer Programming

Variable Neighborhood Search

Genetic Algorithm.

Covering Salesman Problem

Generalized Covering Salesman Problem

Generalized Traveling Salesman Problem

Heuristic Algorithms

Local Search

Preface

This PhD thesis concerns algorithms for Combinatorial Optimization Problems. In Combinatorial Optimization Problems the set of feasible solutions is discrete or can be reduced to a discrete one, and the goal is to find the best possible solution.

Specifically, in this research we consider four different problems in the field of Combinatorial Optimization including Set Covering Problem (SCP), Cost Constrained Minimum Label Spanning Tree Problem (CCMLST), Capacitated *m*-Ring Star Problem (C*m*RSP) and Generalized Covering Salesman Problem (GCSP). All of these problems are NP-Hard. For each problem we propose a heuristic algorithm and we compare our results with the best known results in the literature.

In chapter 1, we consider the Set Covering Problem, in which we are given m rows, n columns each with a specific positive cost, and an $(m \times n)$ sparse matrix of zero-one elements a_{ij}. We say that row i can be covered by column j if and only if a_{ij}=1. We want to cover each row (at least once) with a subset of columns of minimum global cost. In the case of *unicost SCP* all the costs are the same, so we should cover the rows using the minimum number of columns.

We propose a heuristic algorithm to solve the unicost version of the set covering problem based on the Electromagnetism Metaheuristic (EM). We add some new features to this heuristic, with respect to the standard electromagnetism scheme, applying a preprocessing procedure, imposing diverse and high quality solutions in the population, defining the core problem and applying mutation.

We test the performance of the proposed method on the existing datasets from the literature. All of the best known results for the classical benchmark instances, where the number of columns is larger than the number of rows, are found by the proposed method and 12 best known solutions are improved by the proposed algorithm. By using different parameter settings the algorithm improves 4 additional best known solutions. We also show the effectiveness of the EM approach in conducting the search in the solution space by using the related procedures in a Genetic Algorithm (GA) scheme. Although we improve the performance of this Modified GA method by adding a local search procedure, still the proposed algorithm EM performs better than the Modified GA method. Moreover, we modify the proposed algorithm EM for the general or non-unicost SCP. The modified algorithm can obtain all of the best known results from the literature.

The results of this approach will appear in the European Journal of Operational Research [59].

In Chapter 2, we focus on the Capacitated *m*-Ring Star Problem (C*m*RSP), in which the goal is to find *m* rings, each visiting a central depot, subset of customers and a subset of Steiner nodes, so that the total visiting and allocation cost is minimized. Moreover, in each feasible solution each node can not be visited or allocated to two rings simultaneously, and the total number of customers allocated or visited in a ring cannot be greater than the capacity *Q*. We propose a heuristic algorithm to solve this problem. In the proposed heuristic, after the construction of the initial solution, we apply an improvement method based on a set of *swap* and *Extraction-Assignment* moves, followed by the *Lin-Kernighan* TSP procedure to find a better order of the visited nodes. Moreover, the proposed heuristic incorporates some random aspects obtained by perturbing the current solution in the *shaking procedure*, which is applied whenever the algorithm remains in a local optimum.

We compare the proposed heuristic with the best state-of-the-art algorithms for the C*m*RSP on a set of benchmark instances from the literature. The results show the effectiveness of the proposed method. The proposed heuristic can obtain most of the optimal solutions, within a short computing time, and can improve most of the best known solutions for the instances whose optimal solution is not known.

A paper based on the results of this approach has been submitted to the European Journal of Operational Research [70].

In chapter 3, we consider the Cost Constrained Minimum Label Spanning Tree (CCMLST) Problem. Given a graph $G = (V, E)$, where each edge (i, j) has a label from the set L and an edge weight c_{ij}, and a positive budget B, the goal of the CCMLST problem is to find a spanning tree with the fewest number of labels whose weight does not exceed the budget B.

We propose a Variable Neighborhood Search (VNS) method for the CCMLST problem. Considering the VNS as a framework, we start by constructing an initial solution. We then improve upon this initial solution using *local search*. Then, the improvement of the incumbent solution (R) continues in a loop until the termination criterion is reached. This loop contains a *shaking phase* and a *local search phase*. The *shaking phase* considers a specially designed neighborhood and makes random changes to the current solution that enables us to explore neighborhoods farther away from the current solution. The *local search phase* considers a more restricted neighborhood set and attempts to improve upon the quality of a given solution. Besides, we adapt two local search methods and a Genetic Algorithm, proposed by Xiong et al. [82] for the Label Constrained Minimum Spanning Tree Problem, to the CCMLST problem by means of the bisection method.

To compare the results, we generate a set of small, medium-sized and large instances from the TSPLIB dataset. The VNS method performs very well for the CCMLST instances. Of the 191 instances, it provides the best solution in 189 instances. For all the 104 instances, where the optimal solution is known, the VNS method obtains the optimal solution. Furthermore, for the large instances, its running time is an order of magnitude smaller than those of the other heuristics.

The results of this approach will appear in Computers & Operations Research Journal [57].

Finally, in chapter 4 we define a generalization of the Covering Salesman Problem (CSP) in which 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 from a node on the tour. Considering the real world applications, sometimes satisfying the demand of a node and its neighbors by visiting it just once is not possible. In addition, in many applications there is a cost for visiting a node (e.g., cost of hotel for staying in a city for one night). So, we define the Generalized Covering Salesman Problem (GCSP) by specifying the coverage demand k_i which denotes the number of times a node *i* should be covered and by including the node visiting costs (for nodes on the tour). We divide this problem into three variants: Binary GCSP, Integer GCSP without overnight and Integer GCSP with overnight. In the Binary GCSP, 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. In the Integer GCSP without overnight, a node can be visited more than once, but overnight stay is not allowed. Therefore, to have a feasible solution, after visiting a node, the tour can return to this node, if necessary, after having visited at least one other node. Finally, the Integer GCSP with overnight is similar to the previous version, but overnight stay is allowed.

We design two local search heuristics, LS1 and LS2, for these variants. Overall, LS2 appears to be more robust in terms of its running time as well as its performance in terms of the number of times it finds the best solutions in the different variants. Since the Generalized Traveling Salesman Problem (GTSP) is a special case of the GCSP, we compare LS2 to 8 benchmark heuristics for the GTSP as well. The results show that LS2 performs quite well, finding high-quality solutions rapidly.

A paper based on the results of this approach has been submitted to the Informs Journal on Computing [68].

The results of the considered problems in this thesis have been presented in some international conferences such as INFORMS 2008 (Washington DC, US), 23rd Euro Conference on Operational Research (Bonn, Germany), MIC2009 (Hamburg, Germany) and AIRO 2009 (Siena, Italy).

Chapter 1:

An Electromagnetism Metaheuristic for the Unicost Set Covering Problem

Abstract

In this chapter we propose a new heuristic algorithm to solve the unicost version of the well known Set Covering Problem. The method is based on the Electromagnetism Metaheuristic approach which, after generating a pool of solutions to create the initial population, applies a fixed number of *local search* and *movement* iterations based on the “electromagnetism” theory. In addition to some *random* aspects, used in the *construction* and *local search* phases, we also apply *mutation* in order to further escape from local optima.

The proposed algorithm has been tested over 80 instances of the literature. On the classical benchmark instances, where the number of columns is larger than the number of rows, the algorithm, by using a fixed set of parameters, always found the best known solution, and for 12 instances it was able to improve the current best solution. By using different parameter settings the algorithm improved 4 additional best known solutions.

Moreover, we proved the effectiveness of the Electromagnetism Metaheuristic approach for the unicost Set Covering Problem by embedding the procedures of the proposed algorithm in a Genetic Algorithm scheme. The worse results obtained by the Genetic Algorithm show the impact of the Electromagnetism Metaheuristic approach in conducting the search of the solution space by applying the *movements* based on the electromagnetism theory. Finally, we report the results obtained by modifying the proposed Electromagnetism Metaheuristic algorithm for solving the non-unicost Set Covering Problem.

1.1. Introduction

In the general or non-unicost *Set Covering Problem* (SCP), we are given m rows and n columns, each with a specified positive cost c_j, and an $(m \times n)$ sparse matrix of zero-one elements a_{ij}. We say that row i can be covered by column j if and only if a_{ij}=1. We want to cover each row (at least once) with a subset of columns of minimum global cost. So we can formulate the problem through a binary linear programming model as follows:

$$\text{Min} \sum_{j \in J} c_j x_j \qquad (1.1)$$

$$\text{s.t} \quad \begin{cases} \sum_{j \in J} a_{ij} x_j \geq 1 & for\ i \in I \qquad (1.2) \\ x_j = 0 \quad or\ 1 & for\ j \in J \qquad (1.3) \end{cases}$$

where $I = \{1,...,m\}$ is the set of rows, $J = \{1,...,n\}$ the set of columns, and x_j ($j \in J$) a binary variable taking value 1 if and only if column j belongs to the optimal solution.

In the case of *unicost SCP* all the costs are the same (i.e. $c_j = 1, j \in J$), so we want to cover the rows using the minimum number of columns.

We denote the set of columns covering row i with $J_i (i = 1,...,m)$ and the set of rows covered by column j with $I_j (j = 1,...,n)$. Moreover, the number of ones in the binary matrix is denoted by $q = \sum_{i=1}^{m} |J_i| = \sum_{j=1}^{n} |I_j|$. By using this notation, in model (1.1)-(1.3) we can replace (1.2) with:

$$\sum_{j \in J_i} x_j \geq 1 \quad for\ i \in I \qquad (1.2')$$

The Set Covering Problem is known to be NP-hard [33]. It has been considered in the literature as a basic formulation for many real-world optimization problems, therefore it is well known for its numerous applications.

Crew scheduling in railway and mass-transit transportation companies is one of the most relevant applications of the SCP [13]. Delivery and routing, location, distribution, scheduling, manufacturing, service planning, information retrieval and job assignment are some other applications of the SCP. A survey of these applications is provided in Ceria et al. [17].

To challenge very large scale SCP instances, arising from crew scheduling in the Italian railway, Caprara et al. [14] designed a *Lagrangian based heuristic algorithm*, named CFT, which is one of the most effective techniques for the general SCP. Caprara et al. [15] compared different exact and heuristic algorithms and provided a complete survey of the existing literature.

Haouari and Chaouachi [37] reported that the performance of a *Probabilistic Greedy Search Method* is better than that of the static greedy procedures for combinatorial optimization problems, and specially to solve SCP instances. *Indirect Genetic Algorithms* and *Parallel Genetic Algorithms* are two variants of the well-known Genetic metaheuristic approach, proposed simultaneously by Aickelin [1] and Solar et al. [73] for the general SCP.

Different kinds of *Ant Colony Optimization* (ACO) algorithms for the SCP and a comparison of them were reported in Lessing et al. [48]. Yagiura et al. [84] proposed a *3-flip neighborhood local search* for the SCP, allowing their search to visit the infeasible region. They also used some information from the Lagrangian relaxation of model (1.1)-(1.3) to reduce the size of the problem.

A Tabu Search metaheuristic for large-scale set covering problems was presented by Caserta [16]. The author designed a dynamic primal-dual algorithm based on Tabu Search which progressively reduces the gap between the upper and the lower bound. Umetani and Yagiura [76] compared different relaxation heuristics for the SCP.

In the *GRASP method*, proposed by Bautista and Pereira [6], the unicost SCP is considered as a *maximum constraint satisfiability problem* (MAXSAT), and a *GRASP* algorithm to solve this new representation is considered. First they produce a feasible solution for the MAXSAT problem, by using a sequence of random selections from a *candidate list*, and then, as a local search procedure, they apply best flip or random flip to improve the current solution.

The *Metaheuristic for Randomized Priority Search* (Meta-RaPS) approach was proposed by Lan et al. [46] for both the general and the unicost SCP. By considering a candidate list, they construct an initial solution with a random selection between the best candidate and a member of the candidate list. After that, in the local search phase, some randomly chosen columns are removed and the corresponding partial SCP is solved by applying the constructive method. Preprocessing, random selection of the priority rules, definition of a core problem and penalization of the worst columns are characteristics of this method.

In this research, the *Electromagnetism Metaheuristic* (EM) approach [10] is considered as a framework of the proposed method (see Section 1.2). Some new ideas related to this heuristic, such as applying a preprocessing procedure, imposing diverse and high quality solutions in the population, defining the core problem and applying mutation, are presented.

The proposed EM algorithm, described in Section 1.3, is basically different from the *GRASP* and *Meta-RaPS* methods. However, in the construction of the solutions of the initial population, it is similar to the *GRASP* algorithm, although the *GRASP* method finds feasible solutions by solving an associated MAXSAT problem. Moreover, the ideas of preprocessing, core problem and removal of some columns in the local search phase make the proposed algorithm similar to the Caprara et al. [14], Haouari and Chaouachi [37] and Lan et al. [46] approaches, although the corresponding strategies are different.

Computational experiments on the benchmark instances of the literature, comparing the proposed EM algorithm with the most effective algorithms for the unicost SCP, are reported in Section 1.4.1. In addition, to investigate the effectiveness of the EM approach in conducting the search of the solution space, we embedded the proposed procedures in a Genetic Algorithm scheme [9]. The computational results, reported in Section 1.4.2 and comparing the proposed EM algorithm with the Genetic Algorithm, show the quality of the EM approach. Finally, we report the results obtained by modifying the proposed EM algorithm for solving the non-unicost SCP (see Section 1.4.3)

1.2. Electromagnetism Metaheuristic

The Electromagnetism Metaheuristic approach has been recently proposed by Birbil and Fang [10] to solve a class of optimization problems of the form:

$$Min\ \ f(x) \qquad (1.4)$$

$$\text{s.t.} \quad x \in [L,U] \qquad (1.5)$$

where $[L,U] = \{x \in \Re^n \mid L_k \leq x_k \leq U_k, k = 1,...,n\}$ and $x_1,...,x_n$ represent the decision variables. U_k , L_k and $f\,(x)$ represent, respectively, upper and lower bounds on the k-th variable (k =1, …, n) and the objective function value.

The principles of this algorithm are based on the real electromagnetism theory, so each solution is considered as a charged particle, whose charge depends on its objective function value. In the electromagnetic space all particles affect each other; in fact they attract or repel other particles according to their charge. In a similar way, all the forces exerted by other charged particles act upon each of them and determine the resultant force, according to which a particle is moved within its space.

Although the EM approach was introduced for continuous optimization problems, here we adapt it to solve a zero-one optimization problem.

The EM approach has been recently applied to solve several combinatorial optimization problems such as Examination Timetabling [67], Project Scheduling [27], Traveling Salesman [78], [41], Single Machine Scheduling [20], Nurse Scheduling [50], Flow Shop [26], [54], [64], and Job Shop [75].

The general structure of the EM algorithm is described in algorithm 1.1 [10].

Algorithm 1.1. The general Electromagnetism Approach

```
Algorithm EM (Pop_size, MAXITER, LSITER)
1:  Initialize (Pop_size)
2:  iteration :=1
3:  While iteration ≤ MAXITER do
4:      Local Search (LSITER)
5:      Calculate Forces
6:      Move
7:      iteration := iteration+1
8:  End while
```

The EM algorithm starts with the *Initialize* procedure, which produces a pool of *Pop_size* solutions representing the initial population. Then for a fixed number of iterations (*MAXITER*), three different procedures are applied: *Local Search*, *Calculate Forces* and *Move*.

The *Local Search* procedure is applied *LSITER* times and tries to improve each solution of the current population. Then, according to their electromagnetic state, all the particles (solutions) impose "forces" on each other and the total force exerted on each of them is calculated in the *Calculate Forces* procedure. The total force is determined by calculating the charge of each solution X^i, which depends on its objective function value $f(X^i)$, and on the objective value of the current best solution X^{best} in the population [10]. According to the EM algorithm a solution attracts those with higher charge and repels the others.

Finally in the *Move* procedure, by considering a random step length λ uniformly distributed between 0 and 1, each solution is moved in the direction of the resultant force to its new location in the solution space.

The specific formulas for the Set Covering Problem used to calculate charges, forces and the movement action of each solution will be described in Sections 1.3.5 and 1.3.6.

1.3. EM based heuristic for the unicost Set Covering Problem

The basic framework of our algorithm is based on the EM structure, with some modifications specific to deal with a 0-1 programming problem and other parts included to increase the overall performance.

1.3.1. Preprocessing Procedure

The first procedure we call is a standard *Preprocessing procedure* aimed at speeding up the algorithm by reducing the problem dimension. There are different methods for pre-processing an SCP instance [7]. What we do here is to delete a column that covers a subset of rows covered by another column, i.e. we remove the so called *dominated* columns. Moreover we insert in all the solutions those columns by which a row is covered exclusively, i.e. we insert the so called *essential* columns. Note that if column *j* is essential, we can define a "reduced" SCP problem, equivalent to the original one, by removing column *j* and all the rows covered by *j*, and by adding 1 to the value of the objective function.

According to our computational experiments, *preprocessing* does not have an important effect on the quality of the solutions, but it generally decreases the overall computing time of the algorithm.

1.3.2. Population Construction

After the *preprocessing* phase, we consider the remaining rows and columns and construct a pool of *Pop_size* feasible solutions (each consisting in a subset of columns covering all the rows) representing the initial population of the algorithm. Each solution X^i (*i* =1, …, *Pop_size*) is created by using a simple *construction procedure* consisting in a greedy search over a *candidate list* composed of the promising columns, i.e. those columns *j* for which the number of currently uncovered rows they cover (K_j) is greater than a threshold, defined as a given percentage α of the maximum number of uncovered rows covered by a column. At each iteration of the *construction procedure*, a column uniformly randomly chosen from the candidate list is added to the current solution, until we obtain a feasible solution (i.e. all the rows are covered). Each time a column is added to the solution, the values K_j of the columns *j* covering the uncovered rows covered by the added column are updated.

One of our modifications to the standard EM framework is to impose some *diversity* to the solution space. To achieve this we consider two values of parameter α in order to give many columns the possibility to enter the solution. In this manner we have an *elite* population (corresponding to the largest value of α) and a *diverse* one (corresponding to the smallest value of α), and they both make up the whole pool of solutions (see Table 1.3 of Section 1.4 for additional details).

Moreover, in order to reduce the computing time of the local search procedure (see Section 1.3.4), we construct a *core* problem *C* containing the columns of the candidate list. At each iteration of the *construction procedure*, *C* is updated to include the columns in the current candidate list.

After the *construction procedure*, *C* contains all the columns that were in one of the previously considered candidate lists. During the local search procedure we just consider the columns in *C*, in this way we deal with a smaller subset of columns and reduce the corresponding computing time.

By defining *r* (with $r \le \min\{m, n\}$) as the cardinality of the solution, the time complexity for constructing each solution X^i is O(*q*+*rn*).

1.3.3. Delete Redundant Columns Procedure

Since after the construction phase we may have some redundant columns, we define a procedure to delete them. A column is considered as *redundant*, with respect to a given solution, if after deleting it the solution remains feasible. Therefore, we check the columns of the solution to find possible removals. The time complexity of the corresponding procedure is O(*q*).

1.3.4. Local Search Procedure

The idea used in the *local search procedure* is a modification of the improvement phase presented for SCP by Caprara et al. [14] in their refining procedure, and by Lan et al. [46] in their local search phase. We apply this procedure for *LSITER* iterations. For each iteration of the procedure, we first remove a column subset $\overline{X}$ from the current solution *X*, so that we have a set I' of uncovered rows, and then we find a new subset of columns to cover the rows in I'. For each row $i \in I'$, let J'_i denote the subset of columns in the core problem *C* covering row *i*, i.e.:

$$J'_i = \{ j \in J_i \mid j \in C \} \qquad (1.6)$$

In addition, let J' denote the subset of columns of *C* covering the uncovered rows, i.e.

$$J' = \bigcup_{i \in I'} J'_i \qquad (1.7)$$

We can now consider the following *partial* unicost Set Covering Problem:

$$\text{Min} \sum_{j \in J'} x'_j \qquad (1.8)$$

$$\text{s.t} \begin{cases} \sum_{j \in J'_i} x'_j \ge 1 & i \in I' \qquad (1.9) \\ x'_j = 0 \ or \ 1 & j \in J' \qquad (1.10) \end{cases}$$

which is solved by using again the *construction procedure* (See Section 1.3.2). As previously said, the idea of this method is similar to that of the improvement phases presented in algorithms CFT [14] and *Meta-RaPS* [46]. In CFT, the “worst” columns of *X* are removed, while in *Meta-RaPS* the columns are removed completely randomly. Here we have a “mixed” rule in which we remove from

X the columns according to a probability inversely related to the corresponding number of covered rows:

$$P_j = 1 / |I_j| \qquad (1.11)$$

Moreover, of course, each algorithm has its own "constructive" procedure to solve the generated partial SCP.

The number ND of columns which are removed from the current solution X is determined as follows:

$$ND = |X| \times Search\_magnitude \qquad (1.12)$$

where *Search_magnitude* is a parameter which controls the number of the removed columns [46]. To remove a column, we generate a uniform random number r between 0 and 1, and choose a random column j from the current solution X. If r is less than or equal to $P_j / \overline{P}$ (where $\overline{P} = \sum_{j \in X} P_j$) then we remove the column, otherwise we continue with another random column until we find a suitable column to be removed.

Finally, after solving the partial SCP, we call the *delete redundant columns procedure*. Each time the *local search procedure* finds a better solution, the current best solution is updated. For each iteration, the time complexity of this procedure is $O(q+r|C|)$, i.e. in the worst case it is equal to $O(q+rn)$.

1.3.5. Force Calculation Procedure

After applying the *local search procedure* to each solution in the current population, the solutions must be moved towards promising regions in order to get closer to the optimal solution. As mentioned before, by using the main structure of the Electromagnetism approach, we have a way to shift the current solutions towards the best ones.

To adapt the EM approach to deal with binary variables we represent each solution X^i (i =1, …, *Pop_size*) as a binary vector x^i, whose j-th component x_j^i ($j \in J$) takes value one if and only if $j \in X^i$. To determine the new position of the solutions in the solution space, we calculate the total electromagnetic force exerted on each solution by the others according to their "charge". The charge q^i and the components F_j^i ($j \in J$) of the total force exerted on each solution X^i are obtained by adapting the equations proposed by Birbil and Fang (2003) to deal with binary variables.

$$F_j^i = \sum_{\substack{k=1,\\ k\neq i}}^{Pop\_size} \left\{ \begin{array}{ll} (x_j^k - x_j^i)\dfrac{q^i q^k}{\left\|x^k - x^i\right\|^2} & if \quad \left|X^k\right| < \left|X^i\right| \\ (x_j^i - x_j^k)\dfrac{q^i q^k}{\left\|x^k - x^i\right\|^2} & if \quad \left|X^k\right| \geq \left|X^i\right| \end{array} \right\}, i = 1,\dots,Pop\_size, j \in J \qquad (1.13)$$

where

$$q^i = \exp(-n\frac{|X^i| - |X^{best}|}{\sum_{k=1}^{Pop\_size}(|X^k| - |X^{best}|)}), \qquad i = 1,\dots,Pop\_size \qquad (1.14)$$

$$\left\|x^k - x^i\right\| = (\sum_{j\in J}(x_j^k - x_j^i)^2)^{1/2} \qquad (1.15)$$

and X^{best} is the current best solution in the population. The time complexity of this procedure is globally O($Pop\_size^2 \times n$).

1.3.6. Move Procedure

After receiving the effects of all the other solutions, each solution is *moved* according to the resultant force and a random step-length λ, uniformly distributed between 0 and 1 and used to increase the probability of searching the unvisited regions.

The formulation proposed by Birbil and Fang [10] to compute the new location of x^i is as follows:

$$x_j^i = x_j^i + \lambda\frac{F_j^i}{\left\|F^i\right\|}(RNG_j) \qquad i = 1,\dots,Pop\_size, \;\; j \in J \qquad (1.16)$$

where RNG_j denotes the amount of feasible movement toward the upper bound or the lower bound for the *j*-th component. Since here the upper and lower bounds for the variables are one and zero, respectively, the adaptation of equation (1.16) for the binary variables x_j^i gives the following formula:

$$x_j^i = \left\{ \begin{array}{ll} round(x_j^i + \lambda\dfrac{F_j^i}{\left\|F^i\right\|}(1 - x_j^i)) & if \quad F_j^i > 0 \\ round(x_j^i + \lambda\dfrac{F_j^i}{\left\|F^i\right\|}(x_j^i)) & if \quad F_j^i \leq 0 \end{array} \right\}, i = 1,\dots,Pop\_size, \;\; j \in J \qquad (1.17)$$

where $\left\|F^i\right\| = (\sum_{j\in J} {F_j^i}^2)^{1/2}$. The time complexity of the *move procedure* is O(*Pop_size*$\times n$).

It is important to notice that we do not modify the best solution X^{best} in the current population and apply the *move procedure* only to the other solutions.

Since after the *move procedure* the solutions may be no longer feasible, we consider the uncovered rows and the columns covering them and solve the corresponding partial SCP problem by using the *construction procedure*. Finally we apply the *delete redundant columns procedure* to remove the redundant columns.

To clarify the effect of the *force* and *movement* procedures on each solution, let us assume that we have a population with two members X^i and X^k and solution X^k has a smaller number of columns, so it should attract solution X^i. Since we consider each solution as a binary string, this means that solution X^k should try to modify the components of solution X^i to make this solution similar to itself. Let us consider the *j*-th component of both solutions (i.e. X^i_j and X^k_j). Now we have 4 cases depending on the value of the *j*-th component in these solutions (0 or 1). Let us consider first the two cases corresponding to $X^k_j = 1$. If X^i_j=0 the value of F^i_j will be a positive number (see (1.13)), so, from (1.16), the value of X^i_j will probably increase to 1, depending on the values of *force* F^i_j and λ. Otherwise, i.e. if X^i_j=1, from (1.13) we have F^i_j=0 and (1.16) does not change the value of X^i_j. So in both cases the *j*-th component of solution X^i is made similar to the corresponding component of solution X^k. Easy calculations show that when X^k_j is equal to 0, X^i_j will be set to 0 with a probability depending on the values of *force* F^i_j (which is negative in this case) and λ. In the general case, in which we have more solutions in the population, the trend of moving each solution toward the best solutions remains the same.

1.3.7. Mutation Procedure

Mutation is another way of perturbating the solutions and it generally helps to increase the diversity of the population. To achieve this aim, we apply the *mutation procedure* each time the difference between the values of the best and the worst solutions in the population is not larger than a given parameter denoted as *mutation_control*. The *mutation* consists in a *random flip*. A solution X^i of the population and a column *j* are selected randomly. If column *j* belongs to X^i, it is removed from X^i, otherwise it is added to X^i. In the case of removal, the new solution is not feasible, so the *construction procedure* is called to obtain a feasible solution that will be used at the next iteration. In this case the removed column is not allowed to be inserted in the new solution.

1.3.8. Overall algorithm

The procedures presented in the previous sections are applied iteratively, according to the scheme of Algorithm 1.2, until a given time limit is exceeded or a given number of iterations is reached. This stop criterion will be explained in details in Section 1.4.

Algorithm 1.2. Overall EM Metaheuristic for unicost SCP

```
Algorithm EM Metaheuristic
1:   Apply the Preprocessing Procedure
2:   For i=1, …, Pop_size
3:       Apply the Construction Procedure
4:       Apply the Delete Redundant Columns Procedure
5:   End For
6:   While stop criterion not satisfied
7:       For i=1, … , Pop_size
8:           Apply the Local  Search Procedure
9:       End For
10:      Apply the Force Calculation Procedure
11:      Apply the Move Procedure (and possibly the Construction Procedure)
12:      Let $X^{best}$ and $X^{worst}$ be, respectively, the best and the worst solution
         in the current population
13:      If $|X^{worst}| - |X^{best}| < mutation\_control$
14:          Then apply the Mutation Procedure (and possibly the Construction
             Procedure)
15:  End While
```

1.4. Computational Results

In order to test the effectiveness of the proposed algorithm (EM), we use the benchmark instances of the OR-Library introduced by Beasley [8]. All the algorithms considered in this section have been implemented in C. The computational experiments have been performed on a processor

Intel Core Duo CPU running at 1.7 GHz with 1 GB RAM. The characteristics of the instances are summarized in Tables 1.1 and 1.2. The *density* of an instance is defined as the ratio $q/(m \times n)$. Notice that those reported in Table 1.1 are sets of instances with the same features, whereas in Table 1.2 only single instances are considered.

Table 1.1. Characteristics of the instance sets

Data Set	Problem Type	Number of instances	Number of rows	Number of Columns	Density
4	Random	10	200	1000	2 %
5	Random	10	200	2000	2 %
6	Random	5	200	1000	5 %
A	Random	5	300	3000	2 %
B	Random	5	300	3000	5 %
C	Random	5	400	4000	2 %
D	Random	5	400	4000	5 %
E	Random	5	50	500	20 %
NRE	Random	5	500	5000	10 %
NRF	Random	5	500	5000	20 %
NRG	Random	5	1000	10000	2 %
NRH	Random	5	1000	10000	5 %

Table 1.2. Characteristics of the single instances

Instances	Problem Type	Number of rows	Number of columns	Density
CLR10	Combinatorial	511	210	12.3 %
CLR11	Combinatorial	1023	330	12.4 %
CLR12	Combinatorial	2047	495	12.5 %
CLR13	Combinatorial	4095	715	12.5 %
CYC06	Logical	240	192	2.1 %
CYC07	Logical	672	448	0.9 %
CYC08	Logical	1792	1024	0.4 %
CYC09	Logical	4608	2304	0.2 %
CYC10	Logical	11520	5120	0.8 %
CYC11	Logical	28160	11264	0.02 %

1.4.1. Computational results of algorithm EM for the unicost SCP

To compare the performance of algorithm EM with those of the most effective approaches proposed for the unicost SCP, we report in Tables 1.4 and 1.5 the results of algorithm *GRASP*, proposed by Bautista and Pereira [6], on all the benchmark instances (except data sets NRG and NRH), and in Table 1.5 the results of algorithm *Meta-RaPS*, proposed by Lan et al. [46], on the subset of instances considered by the authors (i.e. instances E, CLR and CYC). These methods are currently the best available ones for the unicost SCP. It is important to note that Bautista and Pereira [6] and Lan et al. [46] carried out their computational experiments on a 1.8 GHz and a 1.7 GHz PC, respectively, so we can compare the corresponding Computing times with those of algorithm EM with a good approximation.

Like for the other metaheuristic methods, the best values of the parameters to be used in the overall algorithm are obtained by performing extensive computational experiments on a set of benchmark instances. For algorithm EM the following parameters must be defined: *Pop_size*, α for "*elite population*" and α for "*diverse population*" (Section 1.3.2), *LSITER* and *Search_magnitude* (Section 1.3.4) and *Mutation_control* (Section 1.3.7). In addition, the *Stop criterion* to be used in the overall algorithm (Section 1.3.8) has to be chosen. For the experimental definition of the "best values" to be assigned to the parameters, we have considered the *classical* SCP benchmark instances, having the number of columns larger than the corresponding number of rows (see Table 1.1). For these instances no results (with the exception of the trivial data set E) are reported for algorithm *Meta-RaPS* [46], hence we have chosen the stop criterion by considering the results reported for algorithm *GRASP* [6]. In particular, we stop the main loop of algorithm EM when the global execution time reaches the CPU time spent by algorithm *GRASP* on the corresponding data set (time limit = largest multiple of 5 not larger than the minimum *GRASP* time for the considered data set). In any case, at most *MAXITER* (with *MAXITER*

Table 1.3. Parameters setting

Parameter	**Value**
Pop_size	5 (2 *elite* and 3 *diverse* solutions)
α **for *elite* population**	0.9
α **for *diverse* population**	0.8
Search_magnitude	0.3 if $\lvert X\rvert > 50$ or $d > 10$ 0.6 otherwise
LSITER	200 if $\lvert X\rvert > 50$ or $d > 10$ 400 otherwise
Mutation_control	5

= 100) iterations of the main loop are performed. The values of the parameters used in the execution of algorithm EM are given in Table 1.3.

All the times reported in the following tables are expressed in seconds. For all the instances, the *seed* of the pseudo random number generator used in the execution of EM has been fixed to 1.

Since some sets of instances are not considered in Bautista and Pereira [6] and Lan et al. [46], we divide the computational results into 3 tables.

Table 1.4 reports the results obtained by algorithms EM and *GRASP* [6] on the classical SCP data set 4, 5, 6, A, B, C, D, NRE and NRF. As already mentioned, no results are reported on these data sets for algorithm *Meta-RaPS* [46]. For each instance, the solution value and the corresponding *final time* are reported for the two considered algorithms; in addition, we report the time at which the best solution is found by algorithm EM. For the final time of EM, the mark "(G)" means that EM was stopped according to the *GRASP* final time.

Table 1.5 reports the results obtained by algorithms EM, *GRASP* [6] and *Meta-RaPS* [46] on data set E and on the "single instances" (see Table 1.2). In addition to the previous values, for algorithm *Meta-RaPS* [46] we report the solution value and the time at which this value was found (no information on the final time is given in [46]).

Table 1.6 reports the results obtained by algorithm EM on the remaining data sets NRG and NRH. No results on these data sets are reported in [6] and [46], so as stop criterion we just set the maximum number of iterations equal to 100.

In each table, the last two lines give the sum and the average of the values reported in the corresponding column.

Tables 1.4 and 1.5 show that algorithm EM performs better than algorithm *GRASP*, improving the solution values for 15 instances (marked with a * in the tables), and finding 12 new best known solutions (marked in bold in Table 1.4). Only for instance CYC11 algorithm EM finds a solution worse than that found by algorithm *GRASP*. Moreover, it can be seen that all the solution values found by algorithm EM are obtained in computing times generally much smaller than the corresponding *GRASP* final times. In particular, for datasets NRE and NRF the EM final times as well are always much smaller than the corresponding *GRASP* final times.

Lan et al. [46] reported the results of algorithm *Meta-RaPS* only for data sets E, CLR and CYC. Table 1.5 shows that, with respect to the solutions found by EM, algorithm *Meta-RaPS* obtains better solutions for 3 instances, while for the remaining instances the results are the same. In addition, the computing times of *Meta-RaPS* are smaller than those of EM. It has to be noted that these instances have a particular structure. Indeed, the 5 instances of data set E are much smaller than the other instances, and can be very easily solved to optimality by any greedy procedure. On

Table 1.4. Results of algorithms EM and *GRASP* [6]

Instance	**GRASP Sol**	**GRASP final time**	**EM Sol**	**EM Sol time**	**EM final time**
4.1	38	85	38	22.70	80 (G)
4.2	37	86	37	1.57	80 (G)
4.3	38	85	38	3.00	80 (G)
4.4	39	87	**38 ***	74.38	80 (G)
4.5	38	87	38	2.40	80 (G)
4.6	38	86	38	3.17	80 (G)
4.7	38	84	38	17.00	80 (G)
4.8	38	89	38	3.07	80 (G)
4.9	38	88	38	0.57	80 (G)
4.10	38	86	38	6.72	80 (G)
5.1	35	313	**34 ***	6.84	305 (G)
5.2	34	319	34	220.05	305 (G)
5.3	35	316	**34 ***	25.91	305 (G)
5.4	34	322	34	27.16	305 (G)
5.5	34	327	34	5.84	305 (G)
5.6	34	329	34	297.62	305 (G)
5.7	34	315	34	6.34	305 (G)
5.8	35	314	**34 ***	61.41	305 (G)
5.9	36	313	**35 ***	33.30	305 (G)
5.10	35	308	**34 ***	7.87	305 (G)
6.1	21	113	21	1.87	110 (G)
6.2	20	115	20	79.71	110 (G)
6.3	21	115	21	0.08	110 (G)
6.4	21	114	21	10.89	110 (G)
6.5	21	113	21	2.09	110 (G)
A1	39	549	39	28.85	540 (G)
A2	39	543	39	182.56	540 (G)
A3	39	544	39	140.21	540 (G)
A4	38	549	38	18.74	540 (G)
A5	39	542	**38 ***	7.81	540 (G)
B1	22	960	22	44.32	935 (G)
B2	22	955	22	7.39	935 (G)
B3	22	964	22	7.04	935 (G)
B4	22	942	22	29.04	935 (G)
B5	22	938	22	42.86	935 (G)
C1	43	1074	43	921.96	1030 (G)
C2	44	1079	**43 ***	1023.79	1030 (G)
C3	44	1076	**43 ***	918.46	1030 (G)
C4	44	1074	**43 ***	28.49	1030 (G)
C5	44	1032	**43 ***	1007.09	1030 (G)
D1	25	2468	25	49.60	1939.06
D2	25	2456	25	18.76	1939.06
D3	25	2358	25	73.53	1731.99
D4	25	2435	25	50.56	2329.78
D5	25	2446	25	212.62	2444.16
NRE1	17	20373	**16 ***	1305.72	5109.44
NRE2	17	20381	17	22.61	4770.71
NRE3	17	20372	17	50.67	5407.35
NRE4	17	20376	17	43.31	4617.45
NRE5	17	20368	17	10.78	4128.57
NRF1	10	41960	10	145.71	1353.87
NRF2	10	41521	10	1325.44	1501.82
NRF3	10	41856	10	1399.77	1694.32
NRF4	10	41450	10	120.20	1582.85
NRF5	10	42076	10	1651.14	1709.79
SUM	1613	340326	1601	11810.59	59185.22
AVG	29.33	6187.75	29.11	214.74	1076.09

Table 1.5. Results of algorithms EM, *GRASP* [6] and *Meta-RaPS* [46]

Instances	GRASP Sol	GRASP final time	EM Sol	EM Sol time	EM final time	Meta-RaPS Sol	Meta-RaPS Sol time
E1	5	54	5	0.01	4.77	5	0
E2	5	55	5	0.01	3.91	5	0.03
E3	5	51	5	0.01	2.32	5	0
E4	5	55	5	0.05	4.50	5	0.12
E5	5	56	5	0.02	2.62	5	0
CLR10	25	19	25	0.57	15 (G)	25	0.05
CLR11	23	250	23	15.53	250 (G)	23	3.03
CLR12	23	572	23	109.69	570 (G)	23	4.13
CLR13	23	4987	23	3539.45	4985 (G)	23	48.74
CYC6	60	6	60	0.08	5 (G)	60	0
CYC7	144	26	144	1.97	25 (G)	144	0
CYC8	348	645	344 *	303.40	645 (G)	344	38.91
CYC9	813	442	812 *	407.63	440 (G)	793	88.36
CYC10	1918	1922	1915 *	1892.06	1920 (G)	1826	80.56
CYC11	4268	42516	4272	12922.03	42515 (G)	4140	12656.75
SUM	7670	51656	7666	19192.51	51388.12	7426	12920.68
AVG	511.33	3443.73	511.06	1279.50	3425.87	495.07	861.38

Table 1.6. Results of algorithm EM on data sets NRG and NRH.

Set	EM Sol	EM Sol time	EM final time (100 iterations)
NRG1	63	101.75	2675.36
NRG2	63	127.87	3574.22
NRG3	63	124.47	3098.44
NRG4	63	117.15	4094.42
NRG5	63	32.38	2512.01
NRH1	34	755.72	14430.08
NRH2	34	464.40	12643.38
NRH3	34	1760.62	13201.37
NRH4	34	227.73	14334.96
NRH5	34	1912.47	11823.39
SUM	485	5624.56	82387.63
AVG	48.50	562.46	8238.76

the other hand, the 10 instances of data sets CLR and CYC derived from specific applications, and have the number of rows greater than the number of columns, which is not the case for the majority of the real world applications that can be formulated as a unicost Set Covering Problem. In addition, it is worth to note that the tuning of the parameters used in algorithm EM (see Table 1.3) has beencarried out by considering the classical unicost SCP instances whose results are reported in Table 1.4. No comparison between algorithms EM and *Meta-RaPS* can, of course, be performed on the other data sets.

By using parameters different from those reported in Table 1.3, and different seeds of the pseudo random number generator, algorithm EM was able to improve the best known solution values of the literature for 4 additional classical instances, and the solution values obtained with the fixed parameters setting for 5 instances of data sets CYC and NRG. The corresponding results are reported in column 4 of Table 1.7. We have to note that for instances CYC10 and CYC11 the solution values obtained by algorithm *Meta-RaPS* are still better than those obtained by algorithm EM.

Table 1.7. Improved solutions found by algorithm EM.

Instance	GRASP Sol	EM Sol	EM improved solution	Meta-RaPS solution
4.6	38	38	**37** *	…
4.8	38	38	**37** *	…
6.4	21	21	**20** *	…
D.1	25	25	**24** *	…
CYC9	813	812	793 *	793
CYC10	1918	1915	1892 *	1826
CYC11	4268	4272	4267 *	4140
NRG1	…	63	**62**	…
NRG2	…	63	**62**	…

1.4.2. The effectiveness of the EM approach for the unicost SCP

To investigate the effectiveness of the Electromagnetism Metaheuristic approach for the unicost SCP, we executed the procedures of algorithm EM within a different metaheuristic scheme. Since the main new ideas of the proposed algorithm are related to the *Force Calculation Procedure* and the *Move Procedure,* we eliminated these two procedures and embedded the other procedures in a different framework. In this way we can investigate if the high quality of the solutions found by algorithm EM is only related to the other main procedures (i.e. the construction and local search procedures) or if this is due to the EM approach that performs well in conducting the search in the solution space. Since algorithm EM is a population based algorithm, we embedded its procedures in a different population based metaheuristic approach: the Genetic Algorithm (GA) approach. In particular, we considered the GA method proposed by Beasley and Chu [9] for the non-unicost SCP. The main steps of the GA method are described in Algorithm 1.3 (for more details see [9]). Of course, to reach our aim we must apply this method by replacing the original Beasley and Chu procedures reported at Steps 1, 4 and 5 with the corresponding procedures used in algorithm EM, i.e. the *Construction Procedure*, the *Mutation Procedure* and the *Delete Redundant Columns Procedure*.

Although there is no local search procedure in the GA method proposed by Beasly and Chu [9], we cannot ignore the impact of this procedure in our method. Therefore in the Modified GA method we apply the *Local Search Procedure* after the construction of the initial population and after the *Mutation Procedure*.

Algorithm 1.3. GA method for SCP [9].

```
Algorithm GA for SCP
 1:  Generate an initial population of Pop_size random solutions. Set $t$:=0.
 2:  Randomly select two solutions $P1$ and $P2$ from the population by using the binary
     tournament selection.
 3:  Combine $P1$ and $P2$ to form a new solution $C$ by using the fusion crossover
     operator.
 4:  Mutate $k$ randomly selected columns in $C$, where $k$ is determined by the variable
     mutation schedule.
 5:  Make $C$ feasible and remove possible redundant columns from $C$ by applying the
     heuristic operator.
 6:  If $C$ is identical to any one of the solutions in the population, go to step 2;
     otherwise set $t$:=$t$+1.
 7:  Replace a randomly chosen solution having an above-average fitness in the
     population with $C$ (steady-state replacement method).
 8:  Repeat steps (2)-(7) until $t$=$M$ (i.e. $M$ non-duplicate solutions have been generated).
     The best solution found is the one with the smallest fitness in the population.
```

Moreover, before the execution of the Modified GA method, we apply the *Preprocessing Procedure*. In this way, in the Modified GA method, we consider all the procedures proposed in Section 1.3, except the *Force Calculation Procedure* and the *Move Procedure*, which are the native procedures of the EM approach. The main steps of the Modified GA method are reported in Algorithm 1.4.

After a lot of computational experiments we found that, for the Modified GA method, the best values of the parameters are those used for algorithm EM, except the size of the population which must be increased in the GA structure. Here the *Pop_size* parameter is equal to 100 (with 40 elite and 60 diverse solutions) i.e. the same value used by Beasley and Chu [9] in their paper.

Table 1.8 reports the results of algorithm EM and of the Modified GA method on the classical benchmark unicost SCP instances. The first four columns refer to algorithm EM and are taken from Table 1.4, the last three columns report the solution values, the solution times, and the

Algorithm 1.4. Modified GA with the EM procedures for the unicost SCP.

```
Algorithm Modified GA using the EM procedures
1:   Apply the Preprocessing Procedure.
2:   Generate an initial population of Pop_size solutions by using the
     Construction Procedure.
3:   Improve each solution of the initial population by using the Delete Redundant
     Columns Procedure and the Local Search Procedure.
4:   Randomly select two solutions P1 and P2 from the population by using the binary
     tournament selection.
5:   Combine P1 and P2 to form a new solution C by using the fusion crossover
     operator.
6:   Apply the Mutation Procedure on C according to the mutation _control parameter.
7:   Make C feasible and remove possible redundant columns in C by applying the
     Delete Redundant Columns Procedure.
8:   Improve C by applying the Local Search Procedure.
9:   If C is identical to any one of the solutions in the population, go to step (4);
10:  Replace a randomly chosen solution having an above-average fitness in the
     population with C (steady-state replacement method).
11:  Repeat steps (4)-(10) until the stop criterion is met.
     The best solution found is the one with the smallest fitness in the population.
```

final times of the Modified GA method. To have a fair comparison with algorithm EM the same stopping criterion has been considered. Therefore we stop the execution of the Modified GA method after 100 iterations of the main loop (Steps (4)-(10) in Algorithm 1.4) or when the *GRASP* final time [6] is reached (shown with mark “(G)” on the final time reported in the last column of

Table 1.8. Comparison of algorithm EM with the Modified GA method for the unicost SCP.

Instance	**EM Sol**	**EM Sol time**	**EM final time**	**Modified GA Sol**	**Modified GA Sol time**	**Modified GA final time**
4.1	38	22.70	80 (G)	38	36.17	80 (G)
4.2	37	1.57	80 (G)	37	2.04	80 (G)
4.3	38	3.00	80 (G)	38	4.49	80 (G)
4.4	38	74.38	80 (G)	38	75.41	80 (G)
4.5	38	2.40	80 (G)	**39**	1.49	80 (G)
4.6	38	3.17	80 (G)	37*	2.19	80 (G)
4.7	38	17.00	80 (G)	38	6.34	80 (G)
4.8	38	3.07	80 (G)	38	5.80	80 (G)
4.9	38	0.57	80 (G)	38	39.99	80 (G)
4.10	38	6.72	80 (G)	**39**	1.46	80 (G)
5.1	34	6.84	305 (G)	**35**	1.66	305 (G)
5.2	34	220.05	305 (G)	**35**	1.66	305 (G)
5.3	34	25.91	305 (G)	34	41.04	305 (G)
5.4	34	27.16	305 (G)	34	26.94	305 (G)
5.5	34	5.84	305 (G)	34	2.62	305 (G)
5.6	34	297.62	305 (G)	34	3.79	305 (G)
5.7	34	6.34	305 (G)	34	6.80	305 (G)
5.8	34	61.41	305 (G)	34	144.25	305 (G)
5.9	35	33.30	305 (G)	35	24.11	305 (G)
5.10	34	7.87	305 (G)	34	165.40	305 (G)
6.1	21	1.87	110 (G)	21	2.27	110 (G)
6.2	20	79.71	110 (G)	20	36.52	110 (G)
6.3	21	0.08	110 (G)	21	0.07	110 (G)
6.4	21	10.89	110 (G)	21	3.63	110 (G)
6.5	21	2.09	110 (G)	21	15.10	110 (G)
A1	39	28.85	540 (G)	39	291.50	540 (G)
A2	39	182.56	540 (G)	39	126.98	540 (G)
A3	39	140.21	540 (G)	39	296.24	540 (G)
A4	38	18.74	540 (G)	38	16.67	540 (G)
A5	38	7.81	540 (G)	**39**	57.73	540 (G)
B1	22	44.32	935 (G)	22	5.11	935 (G)
B2	22	7.39	935 (G)	22	8.20	935 (G)
B3	22	7.04	935 (G)	22	40.26	935 (G)
B4	22	29.04	935 (G)	22	78.85	935 (G)
B5	22	42.86	935 (G)	22	91.97	935 (G)
C1	43	921.96	1030 (G)	**44**	70.79	1030 (G)
C2	43	1023.79	1030 (G)	**44**	71.40	1030 (G)
C3	43	918.46	1030 (G)	**44**	114.34	1030 (G)
C4	43	28.49	1030 (G)	43	8.03	1030 (G)
C5	43	1007.09	1030 (G)	**44**	701.62	1030 (G)
D1	25	49.60	1939.06	25	1.17	1283.05
D2	25	18.76	1939.06	25	1.21	1311.68
D3	25	73.53	1731.99	25	28.62	1267.64
D4	25	50.56	2329.78	25	210.22	1264.45
D5	25	212.62	2444.16	25	721.09	1272.94
NRE1	16	1305.72	5109.44	**17**	81.77	2409.23
NRE2	17	22.61	4770.71	17	71.30	2465.47
NRE3	17	50.67	5407.35	17	6.61	2471.32
NRE4	17	43.31	4617.45	17	22.82	2517.28
NRE5	17	10.78	4128.57	17	384.18	2487.99
NRF1	10	145.71	1353.87	**11**	12.85	568.07
NRF2	10	1325.44	1501.82	**11**	13.10	563.29
NRF3	10	1399.77	1694.32	**11**	12.74	561.19
NRF4	10	120.20	1582.85	**11**	12.73	553.51
NRF5	10	1651.14	1709.79	**11**	12.71	560.36
SUM	1601	11810.59	59185.22	1615	4224.05	38482.47
AVG	29.11	214.74	1076.09	29.36	76.80	699.68

Table 1.8). The results of the Modified GA method which are better than those of algorithm EM are shown with a star mark, and the worse results are written in bold. The Modified GA method obtains a better result only in one case (instance 4.6), and is worse than algorithm EM for 15 instances. Note however that the result obtained by the Modified GA method for instance 4.6 has been achieved as well by algorithm EM, by using alternative parameters, as reported in Table 1.7. As for the computing times, Table 1.8 shows that the global average solution time of the Modified GA method is less than that of algorithm EM. However, to have a fair comparison we must consider only the instances whose solution values are the same for both methods. Accordingly, the sum and the average solution times of the EM method, over the 39 instances having the same solution values as the Modified GA method, are, respectively, 1744.32 and 44.73, while these quantities are, respectively, 3053.81 and 78.30 for the Modified GA method. Therefore, algorithm EM can be considered faster than the Modified GA method in finding the same solution values.

1.4.3. Adaptation of algorithm EM for the non-unicost SCP

We have adapted the proposed algorithm EM, designed for the unicost SCP, to the non-unicost SCP. In the unicost SCP only the number of rows covered by a column is considered as the fitness of a column, while in the non-unicost version the cost of each column should be considered as well.

To adapt algorithm EM to deal with the non-unicost SCP, we changed some parts of the proposed procedures. In particular, in the *Preprocessing Procedure* and *Delete Redundant Columns Procedure* (see Sections 1.3.1 and 1.3.3), where we try to remove *dominated* columns and *redundant* columns, respectively, we check the possible removal of a column by considering a subset of columns covering all the rows covered by the column and having a global cost not greater than the cost of the column. In the *Construction Procedure* (Section 1.3.2) we define the promising columns in the *candidate list* as those columns j for which the fitness value, defined as the ratio between the cost of column j (c_j) and the square of the number of currently uncovered rows covered by column j (K_j^2), is less than a threshold, defined as a given percentage ($1+\alpha$) of the minimum amount of the fitness values. Moreover, in the *Force Calculation Procedure* (Section 1.3.5), $f(X^k)$, $f(X^i)$ and $f(X^{best})$ substitute, respectively, $|X^k|$, $|X^i|$ and $|X^{best}|$ in (1.13) and (1.14), where $f(X)$ is the sum of the costs of the columns in solution X. We found as well that, for

Table 1.9. Parameters setting for the non-unicost SCP.

Parameter	Value
Pop_size	5 (2 *elite* and 3 *diverse* solutions)
α **for *elite* population**	0.1
α **for *diverse* population**	0.8
Search_magnitude	0.3
LSITER	400

Table 1.10. Comparison of algorithm EM (using seed 1) with algorithm *Meta-RaPS* [46] for the non-unicost SCP.

Instance	Meta-RaPS Sol	Meta-RaPS Sol time	EM Sol	EM Sol Time	EM final time
4.1	429	1.36	429	0.18	38.35
4.2	512	0.24	512	0.03	46.83
4.3	516	0.29	516	0.57	49.98
4.4	494	0.39	494	4.57	45.24
4.5	512	0.9	**514**	6.95	38.15
4.6	560	0.1	560	3.38	46.61
4.7	430	0.04	430	0.81	34.53
4.8	492	1.46	492	0.03	39.32
4.9	641	3.47	641	0.10	46.02
4.10	514	0.08	514	3.18	40.34
5.1	253	1.55	**254**	0.09	42.38
5.2	302	0.59	302	38.45	49.60
5.3	226	1.14	226	0.15	35.72
5.4	242	0.32	242	32.65	43.35
5.5	211	0.33	211	23.61	38.54
5.6	213	0.14	**214**	24.33	34.56
5.7	293	1.03	293	0.17	44.33
5.8	288	0.08	288	0.49	44.52
5.9	279	0.04	**280**	0.45	43.04
5.10	265	0.03	265	0.92	44.97
6.1	138	0.25	138	0.39	32.85
6.2	146	0.02	146	6.42	30.20
6.3	145	0.02	145	0.07	33.22
6.4	131	0.34	131	0.01	36.90
6.5	161	1.02	161	0.15	38.61
A1	253	6.22	253	23.58	78.56
A2	252	0.28	252	15.70	82.46
A3	232	16.94	**233**	14.21	66.00
A4	234	0.04	234	17.86	72.30
A5	236	9.37	**237**	1.90	73.14
B1	69	0.14	69	0.15	58.46
B2	76	0.53	76	0.32	79.29
B3	80	0.62	80	0.65	59.82
B4	79	2.25	79	4.60	58.21
B5	72	0	72	0.06	74.89
C1	227	0.43	227	59.46	111.05
C2	219	12.89	219	113.86	114.39
C3	243	26.24	**244**	0.60	108.51
C4	219	24.29	219	35.77	102.88
C5	215	1.79	215	10.46	120.21
D1	60	3.13	60	1.10	84.55
D2	66	13.59	66	0.73	75.14
D3	72	1.31	72	0.62	73.58

D4	62	0.2	62	0.12	95.65
D5	61	0.29	61	0.23	106.33
NRE1	29	0.73	29	0.18	130.05
NRE2	30	46.17	30	0.18	103.49
NRE3	27	5.95	27	0.74	111.21
NRE4	28	39.64	28	0.57	135.00
NRE5	28	0.81	28	0.20	180.69
NRF1	14	4.29	14	0.56	160.16
NRF2	15	3.8	15	0.45	167.57
NRF3	14	1.84	14	0.88	133.58
NRF4	14	5.44	14	0.65	144.54
NRF5	13	33.27	13	2.83	109.87
NRG1	176	289.97	176	63.44	434.94
NRG2	154	222.34	154	345.14	411.24
NRG3	166	21.56	**169**	6.24	361.31
NRG4	168	194.21	**170**	347.98	393.57
NRG5	168	47.57	**172**	1.18	362.17
NRH1	63	3917.08	**64**	3.16	308.67
NRH2	63	238.45	63	255.57	374.60
NRH3	59	783.2	**60**	5.27	348.47
NRH4	58	1358.28	58	154.71	321.04
NRH5	55	5.62	55	285.24	289.48
SUM	12762	7356.00	12781	1925.30	7695.23
AVG	196.34	113.17	196.63	29.62	118.39

Table 1.11. The improved solutions of algorithm EM for the non-unicost SCP using 5 additional seeds.

Instance	**Meta-RaPS Sol**	**Meta-RaPS Sol time**	**EM Sol**	**EM improved Sol**	**EM improved Sol time**	**EM improved final time**
4.5	512	0.9	514	512	38.55	221.43
5.1	253	1.55	254	253	63.57	257.12
5.6	213	0.14	214	213	103.17	210.65
5.9	279	0.04	280	279	87.12	255.83
A3	232	16.94	233	232	150.61	399.61
A5	236	9.37	237	236	166.82	435.22
C3	243	26.24	244	243	120.31	655.58
NRG3	166	21.56	169	166	1810.84	2172.65
NRG4	168	194.21	170	168	790.16	2368.93
NRG5	168	47.57	172	168	365.16	2178.73
NRH1	63	3917.08	64	63	1350.61	1846.54
NRH3	59	783.2	60	59	699.54	2097.69

the non-unicost SCP, removing columns completely randomly in the *Local Search Procedure* (Section 1.3.4) and applying the *Mutation Procedure* (Section 1.3.7) at each iteration of the algorithm increases the possibility of finding better solutions. The values of the parameters obtaining the best results for the non-unicost SCP are reported in Table 1.9.

We considered the 65 classical benchmark instances for the non-unicost SCP of the OR-Library [8], and executed the modified code for 100 iterations of the main loop, using again 1 as seed of the pseudo random number generator. Table 1.10 reports the solution values and the solution times of *Meta-RaPS* [46] and of the Modified EM algorithm. No information on the final times of *Meta-RaPS* is given in Lan et al. [46]. As it can be seen from Table 1.10, by using the standard seed 1, algorithm EM is able to obtain the best known solutions of the non-unicost version of SCP [14], [46] for 53 instances. For the remaining 12 instances the modified EM algorithm found worse solution values (shown in bold in Table 1.10). As for the computing time, by considering only the 53 instances whose solution values are the same for both methods, we have that the average solution times of algorithm EM and *Meta-RaPS* are, respectively, 28.55 and 44.10 seconds. By performing in sequence, for each instance, 5 additional executions of the algorithm, by using 5 different seeds and 100 iterations of the main loop for each seed, algorithm EM can obtain the remaining 12 best known solutions (see Table 1.11). Therefore, by executing 6 independent runs of the code, using 6 different seeds, all the best known solutions of the non-unicost SCP are obtained by algorithm EM. By considering all the 65 benchmark instances, the average computing time of algorithm EM for finding its best solution is 111.92 seconds, which is comparable with the corresponding computing time of *Meta-RaPS*.

1.5. Conclusions

We have proposed a new metaheuristic method for the unicost Set Covering Problem based on the Electromagnetism Metaheuristic approach. One of the new features with respect to the standard Electromagnetism scheme is the utilization of a *Preprocessing Procedure* to delete redundant columns, together with the definition of a *core problem* to speed up the algorithm. We construct the current population with both medium and high quality solutions to extend the diversity of the initial population, and apply a *Mutation Procedure* to enhance the possibility of visiting new regions of the solution space. The proposed algorithm is basically different from both the *GRASP* and the *Meta-RaPS* methods, because it uses *diverse* and *elite* solutions in the current population, instead of a single solution, and also because it applies the *Mutation Procedure*. Moreover, the proposed *Local Search Procedure* is able to explore a larger neighbourhood, with respect to the local search of algorithm *GRASP* [6], because it removes a subset of columns and tries to re-optimize over the corresponding partial SCP. Finally, the removal of the columns in the *Local Search Procedure* is not completely random, as in algorithm *Meta-RaPS* [46], but is based on the "quality" of the columns as well. Moreover all the columns have a chance to be removed, and this feature makes the proposed *Local Search Procedure* different from the refining method proposed in algorithm CFT [14].

On the classical benchmark instances from the literature, where the number of columns is larger than the number of rows, the proposed metaheuristic always found the best known solution and for 12 instances it was able to improve the current best solution by using a fixed set of parameters. By using different parameter settings the algorithm improved 4 additional best known solutions. We also reported computational results on 10 additional instances for which no results are known from the literature.

Moreover, to investigate the effectiveness of the EM approach in conducting the search in the solution space we embedded the procedures of the proposed algorithm (except *Force Calculation Procedure* and *Move Procedure* which are the native parts of algorithm EM) in a new scheme based on the Genetic Algorithm (GA) method proposed by Beasley and Chu [9] for the non-unicost SCP. Although we have improved the performance of this Modified GA method by adding the *Local Search Procedure*, still the proposed algorithm EM performs better than the Modified GA method.

Finally a modification of algorithm EM to the non-unicost SCP is presented. This modified algorithm obtains, for the classical SCP instances, all the best known solutions from the literature.

Chapter 2:

A heuristic procedure for the Capacitated *m*-Ring-Star Problem

Abstract:

In this chapter we propose a heuristic method to solve the *Capacitated m-Ring-Star Problem* which has many practical applications in communication networks. The problem consists of finding *m rings* (simple cycles) visiting a central depot, a subset of customers and a subset of potential (*Steiner*) nodes, while customers not belonging to any ring must be "allocated" to a visited (*customer* or *Steiner*) node. Moreover, the rings must be node-disjoint and the number of customers allocated or visited in a ring cannot be greater than the capacity *Q* given as an input parameter. The objective is to minimize the total visiting and allocation costs. The problem is a generalization of the Traveling Salesman Problem, hence it is NP-hard.

In the proposed heuristic, after the construction phase, a series of different local search procedures are applied iteratively. This method incorporates some random aspects by perturbing the current solution through a "shaking" procedure which is applied whenever the algorithm remains in a local optimum for a given number of iterations. Computational experiments on the benchmark instances of the literature show that the proposed heuristic is able to obtain, within a short computing time, most of the optimal solutions and can improve some of the best known results.

2.1 Introduction

The Capacitated *m*-Ring-Star Problem (C*m*RSP) has been introduced by Baldacci et al. [5] in 2007. In the C*m*RSP, we are given a mixed graph $G = (V, E \cup A)$, in which V is the set of nodes, $E = \{\{i, j\} : i, j \in V, i \neq j\}$ is the set of edges (undirected arcs) and A is the set of arcs. The node set V is defined as $V = \{0\} \cup U \cup W$ in which node 0 represents the *depot*, U is the set of *customers* and W is the set of *Steiner nodes*. Each customer $i \in U$ can be connected to a subset of nodes denoted by $C_i \subseteq U \cup W$, so the arc set A can be written as $A = \{(i, j) : i \in U, j \in C_i\}$. We consider a non negative *routing* cost c_e for each edge $e \in E$ and a non negative *allocation* cost d_{ij} for each arc $(i,j) \in A$. A *ring R* is a simple cycle visiting a subset of nodes including the depot. A customer *i* is *assigned* to a ring *R* if it is visited by the ring or allocated to a node on the ring. The number of rings, *m*, and the capacity of each ring, *Q*, are given as input parameters, and it is assumed that $mQ \geq |U|$. In each feasible solution of the C*m*RSP, each customer has to be assigned to exactly one ring, each Steiner node can be visited at most once, and the number of customers assigned to a ring cannot be greater than the capacity *Q*.

The goal of the C*m*RSP is to find *m* rings so that the global cost, given by the sum of the *routing* costs and of the *allocation* costs, is minimized. The C*m*RSP is NP-hard, since it generalizes the Symmetric Traveling Salesman Problem (TSP), arising when m=1, Q=|U|, $W = \phi$, $A = \phi$.

The C*m*RSP has many applications in telecommunication networks, in particular in the fiber optic communication networks (see, e.g. Baldacci et al. [5]).

Baldacci et al. [5] proposed two Integer Linear Programming (ILP) formulations and developed a Branch and Cut (*BC*) approach for the C*m*RSP. The algorithm has been tested on a large variety of problems, including real-world instances. The results show that the proposed algorithm is able to solve to optimality the small-sized instances in a reasonable computing time. Two heuristics are also proposed in [5]. The first one, H1, is an adaptation of the algorithm proposed by Baldacci and Dell'Amico [4] for the multi-depot C*m*RSP and is executed at the root node of the enumeration tree. The second heuristic, H2, takes advantage of the information obtained by the solution of the Linear Programming (LP) relaxation of the proposed ILP formulations to construct a C*m*RSP solution and is executed at a given set of nodes of the enumeration tree.

Mauttone et al. [52] proposed a hybrid metaheuristic approach for the C*m*RSP in 2007. In their approach a combination of *GRASP* and *Tabu Search* algorithms has been proposed for solving the problem.

Finally in 2008 Hoshino and de Souza [39] proposed an ILP formulation based on a Set Covering model and developed a *Branch-and-Price* (*BP*) algorithm for the C*m*RSP. Computational experiments on the two exact algorithms proposed in [5] and [39] show that these methods do not dominate each other, and that, for instances with more than 50 nodes; they fail in many cases in finding the optimal solutions, even employing up to two hours of computing time.

In this approach we propose a heuristic method for the C*m*RSP, which is able to obtain, within a short computing time, most of the optimal solutions and can improve some of the best known results proposed in the literature.

Variants of the C*m*RSP, studied in the literature and arising in telecommunication networks, are described in Baldacci et al. [5] and Labbé et al. [44], [45].

The rest of this chapter is organized as follows. The proposed heuristic is introduced in Section 2.2. Experimental results on the benchmark instances from the literature are presented in Section 2.3. Conclusions are given in Section 2.4.

2.2. Description of the proposed algorithm

This section presents a heuristic procedure developed for the C*m*RSP. In the proposed algorithm, we start with the *Initialization Procedure* which constructs a feasible solution.

Algorithm 2.1. Proposed heuristic for the C*m*RSP.

```
CurrentSolution := Initialization ();
BestCost := Cost (CurrentSolution)  and   BestSolution := CurrentSolution;
iter := 0;

While iter < Max_Iter do
    iter := iter + 1;
    While CurrentSolution can be improved do
        Improvement (CurrentSolution);
    End While;
    If (Cost (CurrentSolution) < BestCost ) Then
        For each ring, call the Lin-Kernighan procedure to improve the ring length;
        Update CurrentSolution, BestCost and BestSolution;
    Else if (Cost (CurrentSolution) > P* BestCost) then
        CurrentSolution := BestSolution;
    End If;
    Shaking (CurrentSolution);
End While.
```

The main body of the heuristic consists of two major phases: *Improvement Procedure* and *Shaking Procedure*, which are iteratively executed. In the *Improvement Procedure*, the goal is to improve the current solution locally, by using the different moves developed for this problem. Whenever we are not able to improve the quality of the current solution by applying the

Improvement Procedure, the algorithm tries to escape from the local optimum by perturbing the current solution through the execution of the *Shaking Procedure*.

Sometimes, by perturbing the current solution, the *Improvement Procedure* is not able to improve the best known solution found so far, and even to recover the current solution. In this situation, to enhance the performance of the algorithm, we use the *threshold accepting* idea, by accepting the worse solution as the current one if its cost is not greater than a given percentage *P* of the cost of the best known solution (where *P* is an input parameter). In addition, the Lin-Kernighan TSP procedure (see Lin and Kernighan [49] and Helsgaum [38]) is applied for each ring. The outline of the proposed heuristic is described in Algorithm 2.1. In the following subsections we give the details of each step. In the description of the algorithm the term "node" refers to a customer or a Steiner node.

2.2.1. Initialization Procedure

To construct the initial solution, we apply the *clustering* algorithm proposed by Fischetti et al. [31] for the Generalized Traveling Salesman Problem (GTSP). This algorithm first constructs a set of *m* rings by considering the depot and *m* customers as far as possible one from each other. To do so, the algorithm chooses the depot as the first node and then selects *m* customers in turn, each as far as possible from the previous ones. As soon as the *m* customers are selected, *m* rings are obtained by connecting each customer to the depot. The remaining customers are assigned (i.e.

Algorithm 2.2. *Clustering* algorithm for the *Initialization Procedure*.

```
Input: $m$, U, $c_{[v,w]}$ for $v, w \in \{0\} \cup U$;
Output: the depot and $m$ customers as far as possible one from each other;
Comment let $far(S) := \arg\max\{\min\{c_{[v,w]} : w \in S\} : v \in U \setminus S\}$ be the furthest customer from a given node subset $S \subseteq \{0\} \cup U$;

Begin
    1. $center_1 := far(\{0\})$;
    2. For $i$ = 2 to $m$+1 do
           $center_i := far(\{center_1,...,center_{i-1}\})$;
       End For;
    3. For $i \in U$ do
            If (customer $i$ is not visited or allocated) Then
                Assign (i.e. visit or allocate) $i$ to its best feasible position;
       End For;
End.
(In $\arg\max\{.\}$, ties are broken by choosing the smallest argument.)
```

visited or allocated) to their best feasible position, i.e., to the feasible (with respect to the capacity constraint) position that generates the minimum insertion cost with respect to the current rings. Note that no Steiner node is used in the initialization procedure. The outline of the clustering algorithm is given in Algorithm 2.2.

2.2.2. Improvement Procedure

In this phase, the algorithm tries to improve the current solution by applying the *Swap*, *Steiner-Node-Removal* and *Extraction-Assignment* procedures developed for the C*m*RSP. To do so, the algorithm iteratively applies the *Swap* procedure. As soon as the solution cannot be improved by using the moves of the *Swap* procedure, the algorithm continues by calling the *Steiner-Node-Removal* procedure and, iteratively, the *Extraction-Assignment* procedure. The outline of the *Improvement Procedure* is given in Algorithm 2.3. In the following subsections the details of the proposed procedures are given.

Algorithm 2.3. Improvement Procedure.

```
While the solution can be improved do
    Swap (CurrentSolution);
End While;
Steiner-Node-Removal (CurrentSolution);
While the solution can be improved do
    Extraction-Assignment (CurrentSolution);
End While.
```

2.2.2.1. Swap Procedure

In this procedure we start by randomly selecting a customer and testing all the possible ways to swap this customer with another visited or allocated node which is near to the selected one, starting from the first nearest node up to the T^{th} nearest one (where T is an input parameter). As soon as a feasible *swap* move leads to an improvement, the current solution is updated and the remaining possible swap moves with the other near nodes are not considered. The procedure continues with the next randomly selected customer of the current solution that has not been considered yet, and stops when all the customers of the current solution have been considered. While doing the *Swap* moves, regardless of the status of the swapped nodes (visited or allocated) the procedure follows the main idea of the *Swap* move, i.e., it switches the position of the two selected nodes. This means that if one of the two nodes is a customer/Steiner node with some allocated customers, after the *swap* move these customers are allocated, if the capacity constraint is satisfied, to the swapped node. The only exception to this rule is the case where both selected nodes are in the same ring, one of them, say *i*, which is a customer, is allocated and the other one, say *j*, which can be a customer or a Steiner node, is visited with some possible allocations. In this situation, since by changing the position of the two nodes the capacity of the ring remains the same, we apply a different rule. First we visit the allocated customer *i* in the current position of node *j*. The remaining nodes (i.e. node *j* and its

Figure 2.1. A special case of the *Swap Procedure* with a customer as a visited node.

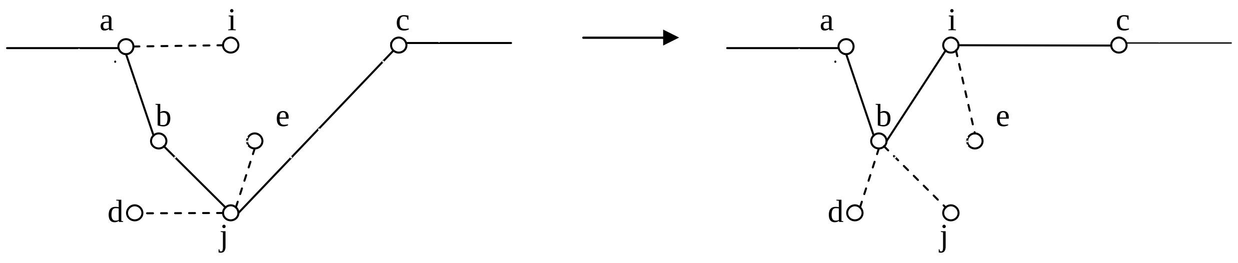
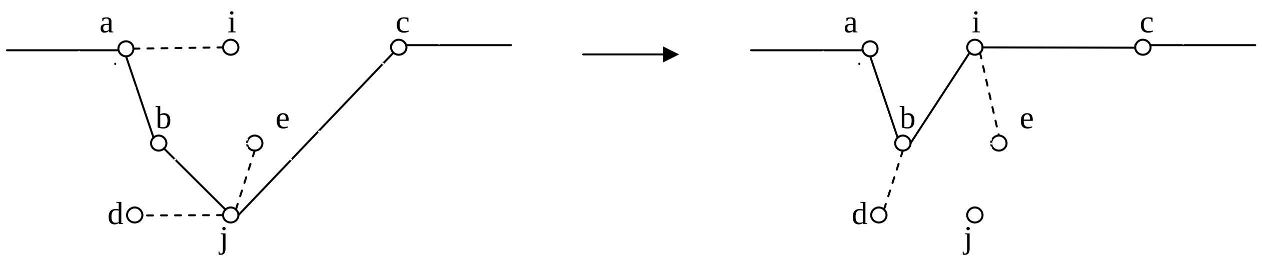


Figure 2.2. A special case in the *Swap Procedure* with a Steiner node as a visited node.



possible allocated customers) are first extracted from the current solution and then, in a random order and once at a time, are visited or allocated in their best feasible position, i.e, the position that generates the minimum insertion cost.

An example of this case is given in Figure 2.1. In this example (see the left side of Figure 2.1) customer *i* is allocated to node *a* and node *j* is a customer, with two allocated customers *d* and *e*. In the first step we visit customer *i* (between *b* and *c*). Then we search for the best position for visiting or allocating customers *d*, *j* and *e*. A possible positioning of these customers is given in the right side of Figure 2.1. Figure 2.2 shows a similar example, but in this case node *j* is a Steiner node. So after the swap it no longer belongs to the solution.

The framework of this procedure is given in Algorithm 2.4.

Algorithm 2.4. *Swap Procedure*.

```
Randomly order the customers;
For $i$ = 1,…, $|U|$ do
    For $l$ = 1,…, $T$ do
        $j$ := $l^{th}$ allocated or visited node nearest to customer $i$;
        If ($i$ and $j$ are in the same ring) and ($i$ is visited and $j$ is allocated, or viceversa)
            Then
            (suppose $i$ is the allocated customer and $j$ is the visited node)
            Construct NewSolution as follows:
                Visit customer $i$ in the ring in the current position of node $j$;
                Extract node $j$, along with its possible allocated customers, from the ring;
                Consider each of these nodes in a random order and, if it is a customer,
                allocate or visit it in its best feasible position in the ring;
        Else
            NewSolution := CurrentSolution with $i$ and $j$ swapped;
        End If;
        If (NewSolution is feasible) and (cost (NewSolution) < cost (CurrentSolution)) Then
            CurrentSolution : = NewSolution;
            Possibly update BestCost and BestSolution;
            Break;
        End If;
    End For;
End For.
```

2.2.2.2. Steiner-Node-Removal

In this procedure, we extract, in a random order, each visited Steiner node, with its allocated customers, from the current solution, and reassign the extracted customers to new feasible positions, so as to decrease the global cost. Starting from the first randomly selected Steiner node, we extract the node along with all its allocated customers. Then, we reinsert each of the extracted customers in its best feasible position, by considering its *T* nearest nodes. If the new solution is not improved with respect to the current one, all the extracted nodes are reinserted in their previous position. Otherwise we update the current solution (and possibly *BestSolution* and *BestCost*). We repeat this procedure until all the Steiner nodes of the current solution have been examined.

2.2.2.3. Extraction-Assignment Procedure

In this procedure, we extract, in a random order, each customer *i* from its current position, and reassign it to a possibly better feasible position by using some specific moves designed for the procedure.

Let us consider the case in which *i* is an allocated customer or a visited customer with no allocation. First we extract customer *i* from its current position. To speed up the search we consider a limited neighborhood of customer *i* containing only its *T* nearest nodes. Let us consider *j* as the l^{th} node nearest to *i*. If *j* is a visited customer or a visited/unvisited Steiner node, regardless of the status of customer *i* (visited with no allocation or allocated), we can consider three different possibilities for customer *i*. If *j* is a visited node, customer *i* can be allocated to node *j*, or visited

Algorithm 2.5. *Extraction-Assignment Procedure*.

```
Randomly order the customers;
For $i$ = 1,…, $|U|$ do
    If $i$ is an allocated customer or a visited customer with no allocation Then
        Extract $i$ from its current position;
        For $l$ = 1,…, $T$ do
            $j$ := $l^{th}$ node nearest to customer $i$;
            If ( $j$ is not an allocated customer ) Then
                State1. Consider the allocation of $i$ to the visited node $j$;
                State2. Consider the visit of $i$ before or after the visited node $j$;
                State3. If $j$ is an unvisited Steiner node, Then
                        consider the allocation of $i$ to $j$ and visit $j$ in its best feasible position
                        among its $T$ nearest visited nodes;
                    End If;
                Select the feasible State corresponding to the minimum cost and possibly
                update BestState;
            End If;
        End For;
    Else
        Extract customer $i$, along with all its allocated customers, from the current solution,
        and assign each of them to its best feasible position by considering its $T$ nearest
        visited nodes;
        Possibly update BestState based on the new positions of the customers;
    End If;
    NewSolution := CurrentSolution by forcing the BestState;
    If cost (NewSolution) < cost (CurrentSolution) Then
        CurrentSolution := NewSolution;
        Possibly update BestCost and BestSolution;
    End If;
End For.
```

before or after node *j*, depending on the position that yields the minimum insertion cost. Finally, if *j* is an unvisited Steiner node, customer *i* is allocated to node *j*, and *j* is visited in its best position among its *T* nearest visited nodes. After having considered all the possible *extraction-assignment* moves, by examining the *T* nodes nearest to *i*, we select the *BestState,* i.e. the move that yields the least total cost. Let us consider now the case in which the considered customer *i* is a visited one with some allocations, we follow the idea proposed in the *Steiner-Node-Removal Procedure*. This means that we extract customer *i*, along with all its allocated customers, from the current solution. Then, we reassign the extracted customers to their best feasible position and consider these assignments for defining the *BestState*.

In both cases the new solution, corresponding to the extraction and reassignment of customer *i*, is obtained from the current solution by forcing the *BestState*. In case of improvement of the total cost, we update the current solution. The outline of this procedure is given in Algorithm 2.5.

2.2.3. Shaking Procedure

Since the *Improvement Procedure* could fail in improving the current solution, the algorithm tries to escape from the local optimum by perturbing the current solution. In particular, we extract in a random order *I* nodes (where *I* is an input parameter), along with their possible allocated customers, from the current solution and construct a *restricted solution* by short cutting them. Then, starting from the first extracted customer, we examine all possible positions for allocating or visiting the customer in the current restricted solution and reassign it to its best feasible position (i.e., to the feasible position that produces the minimum extra cost) in the current restricted solution. We iterate this procedure until all the extracted customers are visited or allocated. The outline of the *Shaking Procedure* is given in Algorithm 2.6.

Algorithm 2.6. *Shaking Procedure*.

```
For $i$ = 1,…, $I$ do
    Randomly extract a node, along with all its possible allocated customers,
    from the current solution;
End For;
While all the extracted customers are not allocated or visited do
    Consider the next unvisited or not allocated customer and assign it to its
    best feasible position.
End While.
```

2.3. Computational Results

The performance of the proposed heuristic for the C*m*RSP has been evaluated by considering the benchmark instances proposed by Baldacci et al. [5] derived from the TSPLIB library defined by Reinelt [65].

The dataset used in [5] is divided into two classes (A and B) including instances from 26 to 101 nodes. The topology of the underlying graphs in both classes is the same, but they are different in the cost structure. In particular, in class A the *routing* and *allocation* costs corresponding to a pair of given nodes *i and j* are the same and equal to the Euclidean distance e_{ij}, computed according to the TSPLIB standard. In class B the *allocation* cost is smaller than the *routing* cost. In particular, $c_{ij} = \lceil 7e_{ij} \rceil$ and $d_{ij} = \lceil 3e_{ij} \rceil$, where c_{ij} and d_{ij} are the *routing* and *allocation* costs, respectively, corresponding to the pair of nodes *i* and *j*. The 6 additional real-world instances considered by Baldacci et al. [5] are not available.

The overall algorithm has been implemented in C and the computational experiments have been performed on an Intel processor with 1.66 GHz and 1 GB RAM. The performance of the proposed heuristic depends on the parameters *P* and *Max_Iter* (Section 2.2), *T* (Sections 2.2.2.1, 2.2.2.2 and 2.2.2.3) and *I* (Section 2.2.3). Like for other heuristics, extensive computational tests have been made to find a suitable set of parameters. The total number of iterations of the main loop of the algorithm, *Max_Iter*, is set to 2000. The other parameters were defined as: $P = 1.05$, $T = 0.2*|V|$ and $I = 0.5*|U|$. As it is customary in testing the performance of the randomized heuristic algorithms, we performed more independent executions of the algorithm. In particular, for each benchmark instance, 5 independent runs of the algorithm have been performed, with 5 different seeds for initializing the random number generator. The best, worst and average performance of the heuristic are provided in Tables 2.1 and 2.2. All the computing times are expressed in seconds. The first column gives the instance name, in which the numbers following *n* and *m* are the number of nodes and the number of rings, respectively. The second column gives the number of customers (|*U*|) and the third one shows the capacity (*Q*) of each ring. Columns 4 (H1) and 6 (H2) give the upper bounds obtained by using heuristics H1 and H2 proposed by Baldacci et al. [5]; their corresponding computing times are reported in columns 5 and 7, respectively. These results have been obtained by running the original codes (provided by Roberto Baldacci) on a Pentium IV computer with 3.4 GHz and 1 GB of RAM. Columns 8 and 9 report the optimal (or the best feasible) solution value obtained by the exact algorithm *BC* (Baldacci et al. [5]) and the corresponding computing time, respectively. These results (obtained on a Pentium IV computer with 2.2 GHz and 1 GB of RAM) are taken from Tables 5 and 6 of Baldacci et al. [5]. For the *BC* algorithm the termination criterion is 2 hours of computing time. For the instances whose computing time is 7200 seconds, the optimal solution value is not available (the best upper bound found within the time limit is reported). The last 5 columns in Tables 2.1 and 2.2 provide the results of the proposed heuristic. For each instance the columns labeled by “Best” and “Worst” report the best and the worst solution values, respectively, obtained during the five independent runs of the algorithm. The column labeled by “Avg.Gap” gives the average gap of the five solution values with respect to the best value found by the heuristic during the five runs. Finally the last two columns report, respectively, the average solution time, i.e., the average computing time at which the best solution has been obtained, and the average running time required to execute the 2000 iterations of the main loop of the proposed heuristic. In both tables, for each instance, the values which are equal to the best solution value, are written in bold. The last three lines of the tables give, respectively, the average values of the corresponding columns, the number of best solutions found by the considered algorithms, and the computer used in the computational experiments. Whenever the optimal solution is not known and the proposed heuristic improves the best known solution value, this is shown with a star mark.

Tables 2.1 and 2.2 show that for the instances solved to optimality by the *BC* algorithm, totally 63 out of 90 instances, the proposed heuristic is able to obtain 62 optimal solutions, by considering the best performance of the algorithm, while this value is 56 for its worst performance among the five independent runs. For the remaining 27 instances, whose optimal solution values are not available, the best performance of the heuristic improves the best known solution for 24 instances, there are 2 ties, and just for one instance the proposed heuristic finds a solution worse than the best known one. By considering its worst performance on these 27 instances, the proposed heuristic improves the best known solution for 21 instances, in 2 cases the results are the same and in 4 cases the results are worse. The tables also show that the proposed heuristic clearly outperforms the heuristic algorithms H1 and H2 proposed by Baldacci et al. [5].

In terms of the global running time, the proposed algorithm is faster than heuristics H1 and H2: the average running times of the proposed method are 1.1 and 2.0 seconds for Classes A and B, respectively, while the average solution times of H1 and H2 are, respectively, 5.4 and 29.7 seconds for class A, and 5.9 and 28.9 seconds for Class B. The exact algorithm *BC* has of course much larger computing times.

A comparison of the proposed heuristic with the *Hybrid Metaheuristic* approach presented by Mauttone et al. [52] is reported in Table 2.3. The performances of both methods are compared with those reported in Baldacci et al. [5], using the *GAP* and *Time Ratio* factors. For each instance, the *GAP* factor is calculated by using Equation (1), where Z^* is the best solution value found by

Table 2.1. Comparison of the proposed heuristic with algorithms H1, H2 and BC [5] for class A.

Instances	|U|	*Q*	H1	Time	H2	Time	BC	Time	**Heuristic Algorithm**				
									Best	Worst	Avg.Gap	Avg.Sol Time	Avg.Run Time
A01-n026-m03	12	5	**242**	0.3	**242**	0.1	**242**	0.1	**242**	**242**	0.00	0.0	0.1
A02-n026-m04	12	4	**261**	0.3	**261**	0.0	**261**	0.0	**261**	**261**	0.00	0.0	0.1
A03-n026-m05	12	3	**292**	0.3	**292**	0.0	**292**	0.0	**292**	**292**	0.00	0.0	0.1
A04-n026-m03	18	7	**301**	0.4	**301**	0.7	**301**	0.5	**301**	**301**	0.00	0.0	0.1
A05-n026-m04	18	5	**339**	0.4	**339**	0.4	**339**	0.3	**339**	**339**	0.00	0.0	0.1
A06-n026-m05	18	4	**375**	0.4	**375**	1.4	**375**	0.7	**375**	**375**	0.00	0.0	0.2
A07-n026-m03	25	10	333	0.8	333	1.7	**325**	3.8	**325**	**325**	0.00	0.0	0.2
A08-n026-m04	25	7	**362**	0.7	**362**	0.9	**362**	0.3	**362**	**362**	0.00	0.0	0.2
A09-n026-m05	25	6	**382**	0.6	**382**	0.6	**382**	0.2	**382**	**382**	0.00	0.0	0.2
A10-n051-m03	12	5	**242**	0.3	**242**	0.1	**242**	0.2	**242**	**242**	0.00	0.0	0.1
A11-n051-m04	12	4	**261**	0.2	**261**	0.1	**261**	0.4	**261**	**261**	0.00	0.0	0.1
A12-n051-m05	12	3	**286**	0.3	**286**	0.1	**286**	0.1	**286**	**286**	0.00	0.0	0.1
A13-n051-m03	25	10	331	0.8	**322**	1.0	**322**	2.1	**322**	**322**	0.00	0.0	0.3
A14-n051-m04	25	7	**360**	0.7	**360**	1.1	**360**	2.1	**360**	**360**	0.00	0.0	0.3
A15-n051-m05	25	6	**379**	0.6	**379**	1.7	**379**	2.3	**379**	**379**	0.00	0.0	0.4
A16-n051-m03	37	14	**373**	2.3	**373**	6.7	**373**	8.4	**373**	**373**	0.00	0.0	0.6
A17-n051-m04	37	11	408	1.6	408	7.6	**405**	41.7	**405**	**405**	0.00	0.1	0.6
A18-n051-m05	37	9	441	2.2	435	11.8	**432**	52.2	**432**	434	0.19	0.1	0.6
A19-n051-m03	50	19	459	4.8	469	14.1	**458**	182.8	**458**	459	0.04	0.1	0.8
A20-n051-m04	50	14	501	3.0	493	20.8	**490**	220.4	**490**	**490**	0.00	0.2	0.8
A21-n051-m05	50	12	521	5.3	521	19.2	**520**	6334.2	**520**	521	0.15	0.1	1.0
A22-n076-m03	18	7	**330**	0.7	**330**	2.9	**330**	48.3	**330**	**330**	0.00	0.0	0.3
A23-n076-m04	18	5	**385**	0.6	**385**	2.7	**385**	30.6	**385**	**385**	0.00	0.0	0.3
A24-n076-m05	18	4	**448**	0.8	**448**	4.2	**448**	63.7	**448**	**448**	0.00	0.0	0.4
A25-n076-m03	37	14	407	2.2	409	9.5	**402**	567.7	**402**	**402**	0.00	0.1	0.8
A26-n076-m04	37	11	462	2.3	461	16.5	460	7200.0	**457** *	458 *	0.18	0.1	0.9
A27-n076-m05	37	9	**479**	3.1	484	21.4	**479**	509.3	**479**	**479**	0.00	0.1	0.9
A28-n076-m03	56	21	475	7.3	478	38.9	**471**	1584.4	**471**	**471**	0.00	0.3	1.5
A29-n076-m04	56	16	523	7.1	524	50.5	523	7200.0	**519** *	520 *	0.15	0.1	1.4
A30-n076-m05	56	13	552	6.3	552	40.2	**545**	3221.3	**545**	549	0.55	0.2	1.5
A31-n076-m03	75	28	570	14.8	565	45.0	**564**	479.5	**564**	569	0.18	0.6	2.2
A32-n076-m04	75	21	617	15.3	628	57.4	606	7200.0	**602** *	607	0.37	0.4	2.3
A33-n076-m05	75	17	659	13.6	654	81.7	654	7200.0	**640** *	651 *	1.38	0.6	1.6
A34-n101-m03	25	10	**363**	0.9	**363**	3.2	**363**	8.7	**363**	**363**	0.00	0.0	0.5
A35-n101-m04	25	7	**415**	1.1	**415**	9.2	**415**	91.8	**415**	**415**	0.00	0.0	0.6
A36-n101-m05	25	6	**448**	1.5	**448**	10.8	**448**	680.4	**448**	**448**	0.00	0.0	0.7
A37-n101-m03	50	18	503	6.5	501	58.8	**500**	7200.0	**500**	**500**	0.00	0.0	1.3
A38-n101-m04	50	14	532	3.9	533	44.5	532	7200.0	**528** *	**528** *	0.00	0.1	1.5
A39-n101-m05	50	12	571	4.0	568	48.4	568	7200.0	**567** *	**567** *	0.00	0.1	1.4
A40-n101-m03	75	28	605	18.6	622	115.6	**595**	6690.1	**595**	**595**	0.00	0.5	2.6
A41-n101-m04	75	21	629	13.3	635	74.5	625	7200.0	**623** *	624 *	0.03	0.4	2.6
A42-n101-m05	75	17	663	11.5	665	120.5	662	7200.0	**657** *	661 *	0.24	1.0	2.4
A43-n101-m03	100	38	672	31.7	672	134.3	**646**	283.0	647	656	0.68	2.0	5.0
A44-n101-m04	100	28	702	26.5	704	109.0	680	7200.0	**679** *	683	0.27	1.7	4.7
A45-n101-m05	100	23	719	24.5	717	148.7	**700**	1310.8	**700**	**700**	0.00	2.3	4.5
Average			448.40	5.4	448.82	29.7	444.62	2098.3	**443.80**	444.89	0.10	0.2	1.1
# best			21		21		36		**44**	32			
			Pentium IV, 3.4 GHz				Pentium IV, 2.2 GHz		Intel, 1.66 GHz				

Table 2.2. Comparison of the proposed heuristic with algorithms H1, H2 and BC [5] for class B.

Instances	|U|	*Q*	H1	Time	H2	Time	BC	Time	**Heuristic Algorithm**				
									Best	Worst	Avg.Gap	Avg.Sol Time	Avg.Run Time
B01-n026-m03	12	5	**1684**	0.3	**1684**	0.1	**1684**	0.1	**1684**	**1684**	0.00	0.0	0.1
B02-n026-m04	12	4	**1827**	0.2	**1827**	0.1	**1827**	0.1	**1827**	**1827**	0.00	0.0	0.1
B03-n026-m05	12	3	**2041**	0.3	**2041**	0.0	**2041**	0.0	**2041**	**2041**	0.00	0.0	0.1
B04-n026-m03	18	7	**2104**	0.4	**2104**	0.6	**2104**	0.5	**2104**	**2104**	0.00	0.0	0.1
B05-n026-m04	18	5	**2370**	0.4	**2370**	1.5	**2370**	0.5	**2370**	**2370**	0.00	0.0	0.2
B06-n026-m05	18	4	**2615**	0.5	**2615**	2.2	**2615**	0.7	**2615**	**2615**	0.00	0.0	0.2
B07-n026-m03	25	10	2314	0.8	**2251**	1.6	**2251**	0.4	**2251**	**2251**	0.00	0.0	0.2
B08-n026-m04	25	7	**2510**	1.1	**2510**	1.2	**2510**	0.5	**2510**	**2510**	0.00	0.0	0.2
B09-n026-m05	25	6	**2674**	0.8	**2674**	2.9	**2674**	0.8	**2674**	**2674**	0.00	0.0	0.2
B10-n051-m03	12	5	**1681**	0.3	**1681**	0.4	**1681**	0.8	**1681**	**1681**	0.00	0.0	0.2
B11-n051-m04	12	4	**1821**	0.2	**1821**	0.6	**1821**	1.5	**1821**	**1821**	0.00	0.0	0.1
B12-n051-m05	12	3	**1972**	0.3	**1972**	0.2	**1972**	0.3	**1972**	**1972**	0.00	0.0	0.2
B13-n051-m03	25	10	**2176**	1.5	**2176**	1.6	**2176**	1.1	**2176**	**2176**	0.00	0.0	0.3
B14-n051-m04	25	7	2476	1.1	2495	4.1	**2470**	7.2	**2470**	**2470**	0.00	0.0	0.3
B15-n051-m05	25	6	2596	1.0	**2579**	2.4	**2579**	4.1	**2579**	**2579**	0.00	0.0	0.4
B16-n051-m03	37	14	2507	2.3	2599	9.4	**2490**	17.9	**2490**	**2490**	0.00	0.2	0.7
B17-n051-m04	37	11	2772	1.9	2811	10.5	**2721**	74.9	**2721**	**2721**	0.00	0.0	0.7
B18-n051-m05	37	9	2938	2.2	2937	14.2	**2908**	145.0	**2908**	2941	0.23	0.2	0.8
B19-n051-m03	50	19	3095	4.0	3071	17.4	**3015**	296.7	**3015**	**3015**	0.00	0.2	1.5
B20-n051-m04	50	14	3365	3.6	3298	18.1	**3260**	336.6	**3260**	**3260**	0.00	0.2	1.4
B21-n051-m05	50	12	3525	5.7	3516	18.9	**3404**	6470.7	**3404**	**3404**	0.00	0.6	1.5
B22-n076-m03	18	7	2260	0.7	2259	2.6	**2253**	105.5	**2253**	**2253**	0.00	0.1	0.4
B23-n076-m04	18	5	2625	0.5	**2620**	3.3	**2620**	29.5	**2620**	**2620**	0.00	0.0	0.4
B24-n076-m05	18	4	**3059**	0.9	**3059**	3.4	**3059**	85.3	**3059**	**3059**	0.00	0.0	0.4
B25-n076-m03	37	14	2742	3.1	**2720**	14.3	**2720**	1897.6	**2720**	**2720**	0.00	0.1	0.9
B26-n076-m04	37	11	3176	2.7	3138	17.5	3138	7200.0	**3100 ***	3123 *	0.49	0.6	1.1
B27-n076-m05	37	9	3339	3.0	3364	23.8	3311	7200.0	**3284 ***	**3284 ***	0.00	0.1	0.9
B28-n076-m03	56	21	3112	7.1	3146	31.4	3088	7200.0	**3044 ***	3064 *	0.52	0.9	2.6
B29-n076-m04	56	16	3447	5.1	3496	50.3	3447	7200.0	**3415 ***	3466	0.69	1.2	2.4
B30-n076-m05	56	13	3652	4.6	3703	35.5	3648	7200.0	**3636 ***	3645 *	0.17	0.9	2.5
B31-n076-m03	75	28	3786	14.1	3820	69.1	3740	7200.0	**3652 ***	3734 *	0.96	1.2	5.0
B32-n076-m04	75	21	4057	13.9	4084	78.8	4026	7200.0	**4003 ***	4011 *	0.12	1.8	4.4
B33-n076-m05	75	17	4442	15.9	4288	54.5	4288	7200.0	**4217 ***	**4217 ***	0.00	2.2	4.0
B34-n101-m03	25	10	2437	0.7	2439	4.3	**2434**	24.2	**2434**	**2434**	0.00	0.0	0.7
B35-n101-m04	25	7	**2782**	1.2	2819	9.5	**2782**	115.4	**2782**	**2782**	0.00	0.0	0.7
B36-n101-m05	25	6	3043	1.0	3012	4.8	**3009**	862.4	**3009**	**3009**	0.00	0.0	0.8
B37-n101-m03	50	18	3404	6.3	3387	37.2	3332	7200.0	**3322 ***	**3322 ***	0.00	0.3	1.7
B38-n101-m04	50	14	3593	4.3	3586	32.1	**3533**	7200.0	**3533**	**3533**	0.00	0.1	1.8
B39-n101-m05	50	12	3880	4.4	3872	33.2	3872	7200.0	**3834 ***	3841 *	0.15	1.4	2.0
B40-n101-m03	75	28	3935	26.0	3923	260.7	3923	7200.0	**3887 ***	3889 *	0.02	2.7	4.6
B41-n101-m04	75	21	4190	16.0	4202	63.5	4125	7200.0	**4082 ***	4114 *	0.16	2.0	4.5
B42-n101-m05	75	17	4486	14.2	4458	47.9	4458	7200.0	**4358 ***	**4358 ***	0.00	1.6	3.8
B43-n101-m03	100	38	4275	35.9	4155	103.0	**4110**	7200.0	4135	4193	0.38	9.3	13.4
B44-n101-m04	100	28	4583	28.0	4608	97.4	4506	7200.0	**4358 ***	4391 *	0.45	6.2	10.8
B45-n101-m05	100	23	4671	26.7	4639	114.6	4632	7200.0	**4567 ***	4597 *	0.21	6.4	9.3
Average			3023.09	5.9	3018.42	28.9	2991.71	2952.9	**2975.04**	2983.67	0.10	0.9	2.0
# best			14		17		30		**44**	32			
			Pentium IV, 3.4 GHz				Pentium IV, 2.2 GHz		Intel, 1.66 GHz				

Table 2.3. Comparison of the proposed method with *Hybrid Metaheuristic* [52].

Group			*Hybrid Metaheuristic*					*Heuristic Algorithm*				
			GAP(%)			*Time Ratio*		*GAP*(%)			*Time Ratio*	
Class	*N*	No. Instances	Best	Avg.	Worst	Min	Max	Best	Avg.	Worst	Min	Max
A	26	9	0.00	0.34	2.09	< 1	2	0.00	0.00	0.00	0.00	19.02
A	51	12	0.00	0.89	2.55	< 1	66	0.00	0.01	0.19	1.27	6598.39
A	76	12	-0.43	1.51	3.47	3	208	-2.14	-0.12	0.89	101.34	8461.20
A	101	12	0.28	2.04	4.71	< 1	194	-0.75	-0.06	1.55	24.44	5630.05
B	26	9	0.55	1.24	3.29	< 1	< 1	0.00	0.00	0.00	0.00	4.46
B	51	12	0.88	3.43	5.35	< 1	49	0.00	0.02	1.13	1.84	4397.75
B	76	12	1.42	3.19	5.99	4	121	-2.35	-0.53	0.55	81.55	7916.96
B	101	12	3.20	4.24	6.18	1	70	-3.28	-0.68	2.02	36.47	4205.84
Overall Avg.			0.74	2.11	4.20			-1.07	-0.17	0.79		
			PC, 2 GHz					Intel, 1.66 GHz				

$$GAP = 100 * \frac{(Z^{best} - Z^*)}{Z^*} \tag{2.1}$$

Baldacci et al. [5] and Z^{best} is the best solution value found during the different executions of the *Hybrid Metaheuristic* [52] or of the proposed heuristic. As mentioned before, we have executed 5 independent runs of the code, while 15 independent runs have been considered for the *Hybrid Metaheuristic* approach [52]. The other factor reported in Table 2.3, *Time Ratio*, is obtained by dividing, for each instance, the execution time of algorithm *BC* reported in [5] over the average computing time of the independent executions performed by each of the two heuristics. The computing times of the *Hybrid Metaheuristic* [52] refer to a PC computer with 2 GHz and 1 GB of RAM.

Considering the average performance of the two heuristics, Table 2.3 shows the superiority of the proposed heuristic for the considered instances. As it can be seen from this table, by considering the average performance of the methods, the overall average *GAP* is 2.11 for the *Hybrid Metaheuristic* and -0.17 for the heuristic method. Moreover, mainly for the larger instances, the proposed heuristic is much faster than the *Hybrid Metaheuristic*.

During its independent executions, the *Hybrid Metaheuristic* was able to obtain 2 new best solutions (instances A26 and A29) with respect to the solutions found by algorithm *BC* [5]. However the proposed heuristic not only was able to further improve these 2 new best solutions, but improved as well 22 additional best known solutions.

Finally, we executed the proposed heuristic with different tunings of parameters *P* and *T*, to investigate how the threshold accepting parameter (*P*) and the neighborhood size (*T*) affect the overall performance of the method. To do so, we considered all the instances proposed by Baldacci et al. [5] and fixed the maximum number of iterations.

Figure 2.3. Analyzing the effect of parameter *P* in total cost.

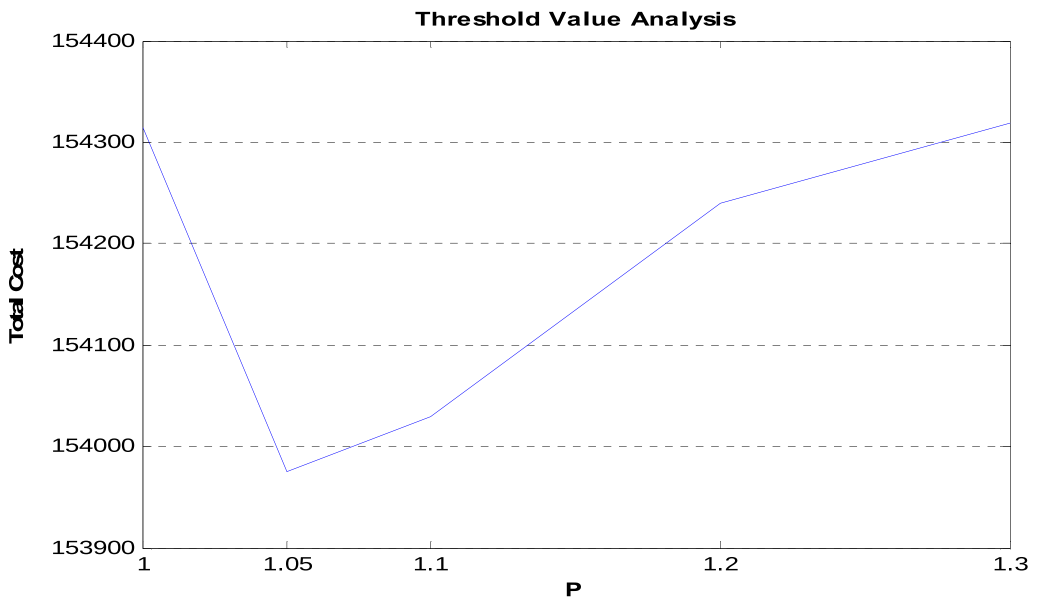
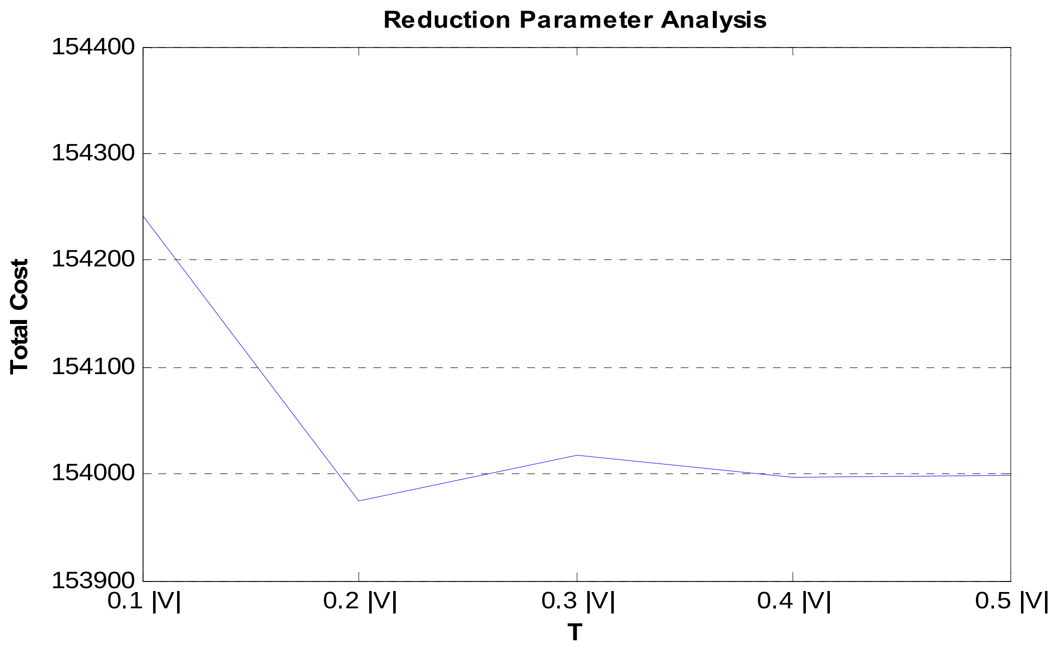


Figure 2.4. Analyzing the effect of parameter *T* in total cost.



Figures 2.3 and 2.4 represent the sensitivity of the heuristic method with respect to the different considered values. In these figures the vertical lines represent the total cost of the 90 benchmark instances for the different considered values of parameters P and T, as shown in the horizontal lines of Figures 2.3 and 2.4, respectively. As it can be seen from the figures, the minimum cost values occur for P=0.05 and T=0.2, which are the values considered to run the code.

2.4. Conclusion

We have proposed an effective heuristic approach for the Capacitated *m*-Ring-Star Problem (C*m*RSP). In the proposed heuristic, after the construction of the initial solution, we apply an improvement method based on a set of swap and Extraction-Assignment moves, followed by the Lin-Kernighan TSP procedure to find a better order of the visited nodes. The proposed heuristic incorporates some random aspects obtained by perturbing the current solution in the shaking procedure, which is applied whenever the algorithm remains in a local optimum.

We compared the proposed heuristic with the best state-of-the-art algorithms for the C*m*RSP on a set of benchmark instances from the literature. The results show the effectiveness of the proposed method. It turned out that in the considered instances the proposed heuristic can obtain most of the optimal solutions, within a short computing time, and can improve most of the upper bounds for the instances whose optimal solution is not known.

2.5. Acknowledgments

We would like to thank Roberto Baldacci who provided us the codes of algorithms H1 and H2 proposed in Baldacci et al. [5].

Chapter 3:

Variable Neighborhood Search For the Cost Constrained Minimum Label Spanning Tree Problem

Abstract:

Given an undirected graph whose edges are labeled or colored, edge weights indicating the cost of an edge, and a positive budget *B*, the goal of the Cost Constrained Minimum Label Spanning Tree (CCMLST) Problem is to find a spanning tree that uses the minimum number of labels while ensuring its cost does not exceed *B*. This problem is motivated from the design of telecommunication networks and is known to be NP-complete [82].

In this chapter, we present a Variable Neighborhood Search (VNS) algorithm for the CCMLST problem. We test the VNS algorithm on existing data sets as well as a large-scale dataset based on TSPLIB [65] instances ranging in size from 500 to 1000 nodes. For the CCMLST problem, the procedures suggested in [82] (for the Label constraint Minimum Spanning Tree Problem) can be applied by means of a binary search procedure. Consequently, we compared our VNS algorithm to the GA and two local search procedures suggested in [82]. The overall results demonstrate that the proposed VNS algorithm is of high quality and computes solutions rapidly. On our test datasets, it obtains the optimal solution in all instances for which the optimal solution is known. Further, it significantly outperforms the GA and two local search procedures described in [82].

3.1. Introduction

The Minimum Label Spanning Tree (MLST) problem was introduced by Chang and Leu [19]. In this problem, we are given an undirected graph $G = (V, E)$ with labeled edges; each edge has a single label from the set of labels L and different edges can have the same label. The objective is to find a spanning tree with the minimum number of distinct labels. The MLST is motivated from applications in the communications sector. Since communication networks sometimes include numerous different media such as fiber optics, cable, microwave or telephone lines and communication along each edge requires a specific media type, decreasing the number of different media types in the spanning tree reduces the complexity of the communication process. The MLST problem is known to be NP-complete [19]. Several researchers have studied the MLST problem including Brüggemann et al. [11], Cerulli et al. [18], Consoli et al. [22], Krumke and Wirth [42], Wan et al. [79], and Xiong et al. [80, 81, 83].

Recently Xiong et al. [82] introduced a more realistic version of the MLST problem called the Label Constrained Minimum Spanning Tree (LCMST) problem. In contrast to the MLST problem, which completely ignores edge costs, the LCMST problem takes into account the cost or weight of edges in the network (we use the term cost and weight interchangeably in this chapter). The objective of the LCMST problem is to find a minimum weight spanning tree that uses at most K labels (i.e., different types of communications media). Xiong et al. [82] describe two simple local search heuristics and a genetic algorithm for solving the LCMST problem. They also describe a Mixed Integer Programming (MIP) model to solve the problem exactly. However, the MIP models were unable to find solutions for problems with greater than 50 nodes due to excessive memory requirements.

The Cost Constrained Minimum Label Spanning Tree (CCMLST) problem is another realistic version of the MLST problem. The CCMLST problem was introduced by Xiong et al. [82]. In contrast to the LCMST problem, there is a threshold on the cost of the minimum spanning tree (MST) while minimizing the number of labels. Thus, given a graph $G = (V, E)$, where each edge

(i, j) has a label from the set L and an edge weight c_{ij}, and a positive budget B, the goal of the CCMLST problem is to find a spanning tree with the fewest number of labels whose weight does not exceed the budget B. The notion is to design a tree with the fewest number of labels while ensuring that the budget for the network design is not exceeded. (Notice that the objective function here is not the cost of the spanning tree, but rather the number of labels in the spanning tree). Xiong et al. [82] showed that both the LCMST and the CCMLST are NP-Complete. Thus, the resolution of these problems requires heuristics.

In this research, we focus on the CCMLST problem. We propose a Variable Neighborhood Search (VNS) method for the CCMLST problem. The VNS algorithm uses neighborhoods defined on the labels. We then compare the VNS method to the heuristics described by Xiong et al. [82]. In fact, we adapt the procedures of Xiong et al. [82] by embedding them in a binary search. To do so, we consider existing data sets and also design a set of nine Euclidean large-scale datasets, derived from TSPLIB instances [65]. The VNS method performs extremely well on the CCMLST problem, with respect to solution quality and computational running time.

The rest of this chapter is organized as follows. Section 3.2 describes the mathematical formulation proposed for the CCMLST problem. Section 3.3 describes the VNS method that we have proposed to solve the problem. Section 3.4 describes the procedures of Xiong et al. [82] for the LCMST problem, and explains the binary search procedure to apply them for the CCMLST problem. Section 3.5 reports on our computational experiments. Finally, Section 3.6 provides concluding remarks.

3.2. Mathematical Formulation

In this section, we provide a mixed integer programming (MIP) model for the CCMLST problem. It is based on a multicommodity network flow formulation, and is similar to the MIP model described in Xiong et al. [82] (though our notation is somewhat more compact). Further, we significantly strengthen the model by improving upon one of the constraints in their model.

The main idea in the multicommodity network flow model is to direct the spanning tree away from an arbitrarily selected root node, and to use flow variables to model the connectivity requirement. To help define the multicommodity network flow model, we define a bidirected network obtained by replacing each undirected edge $\{i, j\}$ by a pair of directed arcs (i, j) and (j, i). Let A denote the set of arcs, $L= \{1,2,...,l\}$ the set of labels, $V= \{1,2,...,n\}$ the set of nodes in the graph, and B the budget. Also, let A_k denote the set of all arcs with label k and c_{ij} be the cost of arc (i, j). Note that the cost and label of arcs (i, j) and (j, i) are identical to those of edge $\{i, j\}$. To model the fact that the spanning tree must be connected, we use the following well-known idea [51] and multicommodity network flow model. We pick node 1 as the root node (any node of the graph may be picked for this purpose). We then observe that the spanning tree on the nodes can be directed away from the root node. Consequently, we create commodities, where each commodity has the root node as its origin and the destination is one of the nodes in $\{2, 3,...., n\}$ (for a total of $n-1$ commodities). Each commodity has a supply of 1 unit of flow and a demand of 1 unit of flow. The variables in the multicommodity network flow formulation are defined as follows:

$$y_k = \begin{cases} 1 & \textit{if label } k \textit{ is selected} \\ 0 & \textit{otherwise} \end{cases}$$

$$x_{ij} = \begin{cases} 1 & \textit{if arc}\,(i,j)\,\textit{is used} \\ 0 & \textit{otherwise} \end{cases}$$

and

$$f_{ij}^{h} = \text{flow of commodity } h \text{ along arc } (i,j).$$

The MIP formulation based on the multicommodity flow (mcf) model is as follows:

(mcf) $$\min \sum_{k \in L} y_k \tag{3.1}$$

subject to $$\sum_{(i,j) \in A} x_{ij} = n-1 \tag{3.2}$$

$$\sum_{i:(i,h)\in A} f_{ih}^{h} - \sum_{l:(h,l)\in A} f_{hl}^{h} = 1 \quad \forall\ h \in V \setminus \{1\} \tag{3.3}$$

$$\sum_{i:(i,1)\in A} f_{i1}^{h} - \sum_{l:(1,l)\in A} f_{1l}^{h} = -1 \quad \forall\ h \in V \setminus \{1\} \tag{3.4}$$

$$\sum_{i:(i,j)\in A} f_{ij}^{h} - \sum_{l:(j,l)\in A} f_{jl}^{h} = 0 \quad \forall\ h \in V \setminus \{1\},\ \forall j \neq h \tag{3.5}$$

$$f_{ij}^{h} \le x_{ij} \qquad \forall (i,j) \in A,\ \forall h \in V \setminus \{1\} \tag{3.6}$$

$$x_{ij} + x_{ji} \le 1 \qquad \forall (i,j) \in E \tag{3.7}$$

$$\sum_{(i,j)\in A_k} x_{ij} \le (n-1) \cdot y_k \qquad \forall k \in L \tag{3.8}$$

$$\sum_{(i,j)\in A} c_{ij} x_{ij} \le B \tag{3.9}$$

$$x_{ij}, y_k \in \{0,1\} \qquad \forall (i,j) \in A,\ \forall k \in L \tag{3.10}$$

$$f_{ij}^{h} \ge 0 \qquad \forall (i,j) \in A,\ \forall h \in V \setminus \{1\}. \tag{3.11}$$

In the objective function (3.1), we want to minimize the total number of labels used in the solution. Constraint (3.2) ensures the tree has exactly (*n*-1) arcs. Constraints (3.3) to (3.5) represent the flow balance constraints for the commodity flows. Constraint set (3.6) is a forcing constraint set. These constraints enforce the condition that if flow is sent along an arc, the arc must be included in the directed tree. Constraint set (3.7) ensures that either arc (*i, j*) or arc (*j, i*) can be in the solution, but not both (recall the tree must be directed away from the root node). Constraint set (3.8) is a forcing constraint set between arcs and labels. It says that if an arc with label *k* is used, then this label must be selected. Constraint (3.9) imposes the budget on the tree cost. Finally, constraint (3.10) defines the arc and label variables as binary, and constraint (3.11) defines the flow variables as non-negative.

With the change in the objective function and constraint (3.9) the multicommodity flow formulation is virtually identical to Xiong et al. [82] (we have eliminated the edge variables in the Xiong et al. [82] model and thus our notation is somewhat more compact). However, this formulation can be considerable strengthened by using the technique of constraint disaggregation (see page 185 of [29]) on constraint set (3.8). We replace this constraint with the stronger:

$$x_{ij} \le y_k \qquad \forall k \in L, \quad \forall (i,j) \in A_k \; . \tag{3.8'}$$

In our computational work, we use the multicommodity flow model with constraint set (3.8') to obtain lower bounds and optimal solutions on our test instances. We found that it is considerably stronger than the multicommodity flow model proposed by Xiong et al. [82] that has constraint (3.8).

3.3. Variable Neighborhood Search for the CCMLST Problem

In this section, we develop our Variable Neighborhood Search algorithm for the CCMLST problem. Variable Neighborhood Search is a metaheuristic proposed by Mladenovic and Hansen [53], which explicitly applies a strategy based on dynamically changing neighborhood structures. The algorithm is very general and many degrees of freedom exist for designing variants.

The basic idea is to choose a set of neighborhood structures that vary in size. These neighborhoods can be arbitrarily chosen, but usually a sequence of neighborhoods with increasing cardinality is defined. In the VNS paradigm, an initial solution is generated, then the neighborhood index is initialized, and the algorithm iterates through the different neighborhood structures looking for improvements, until a stopping condition is met.

We consider VNS as a framework, and start by constructing an initial solution. We then improve upon this initial solution using *local search*. Then, the improvement of the incumbent solution (R) continues in a loop until the termination criterion is reached. This loop contains a *shaking phase* and a *local search phase*. The *shaking phase* follows the VNS paradigm. It considers a specially designed neighborhood and makes random changes to the current solution that enables us to explore neighborhoods farther away from the current solution. The *local search phase* considers a more restricted neighborhood set and attempts to improve upon the quality of a given solution.

We now make an important observation regarding the relationship between the selected labels and the associated solution. Given a set of labels $R \subseteq L$, the minimum cost solution on the labels R is the minimum spanning tree computed on the graph induced by the labels in R. We denote the minimum spanning tree on the graph induced by the labels in R as MST(R) and its cost by MSTCOST(R). These two can be computed rapidly using any of the well-known minimum spanning tree algorithms [43, 63]. Consequently, our search for a solution focuses on selecting labels (as opposed to edges), and our neighborhoods as such are neighborhoods on labels. Our solutions then are described in terms of the labels they contain (as opposed to the edges they contain). Furthermore, without loss of generality, we assume MSTCOST(L) $\leq B$, because if $B <$ MSTCOST(L) the problem is infeasible.

3.3.1. Initial Solution

Our procedure to construct an initial solution focuses on selecting a minimal set of labels that result in a connected graph. Let *Components*(R) denote the number of connected components in the graph induced by the labels in R. This can easily be computed using depth first search [74]. Our procedure adds labels to our solution in a greedy fashion. The label selected for addition to the current set of labels is the one (amongst all the labels that are not in the current set of labels) that when added results in the minimum number of connected components. Ties between labels are broken randomly. In other words, we choose a label for addition to the current set of labels R randomly from the set

$$S = \left\{ t \in (L \setminus R) : \min \ Components \ \left(R \cup \{t\}\right) \right\}. \tag{3.12}$$

This continues until the selected labels result in a single component.

In Figure 3.1, an example illustrating the initialization method is shown. Suppose there are three labels, namely *a*, *b*, and *c*, in the label set. Since the number of connected components after adding label *c* is less than for the two other labels, we add this label to the solution. However the graph is still not connected, so we go further by repeating this procedure with the remaining labels. Both labels *a* and *b* produce the same number of components, so we select one of them randomly (label *b*).

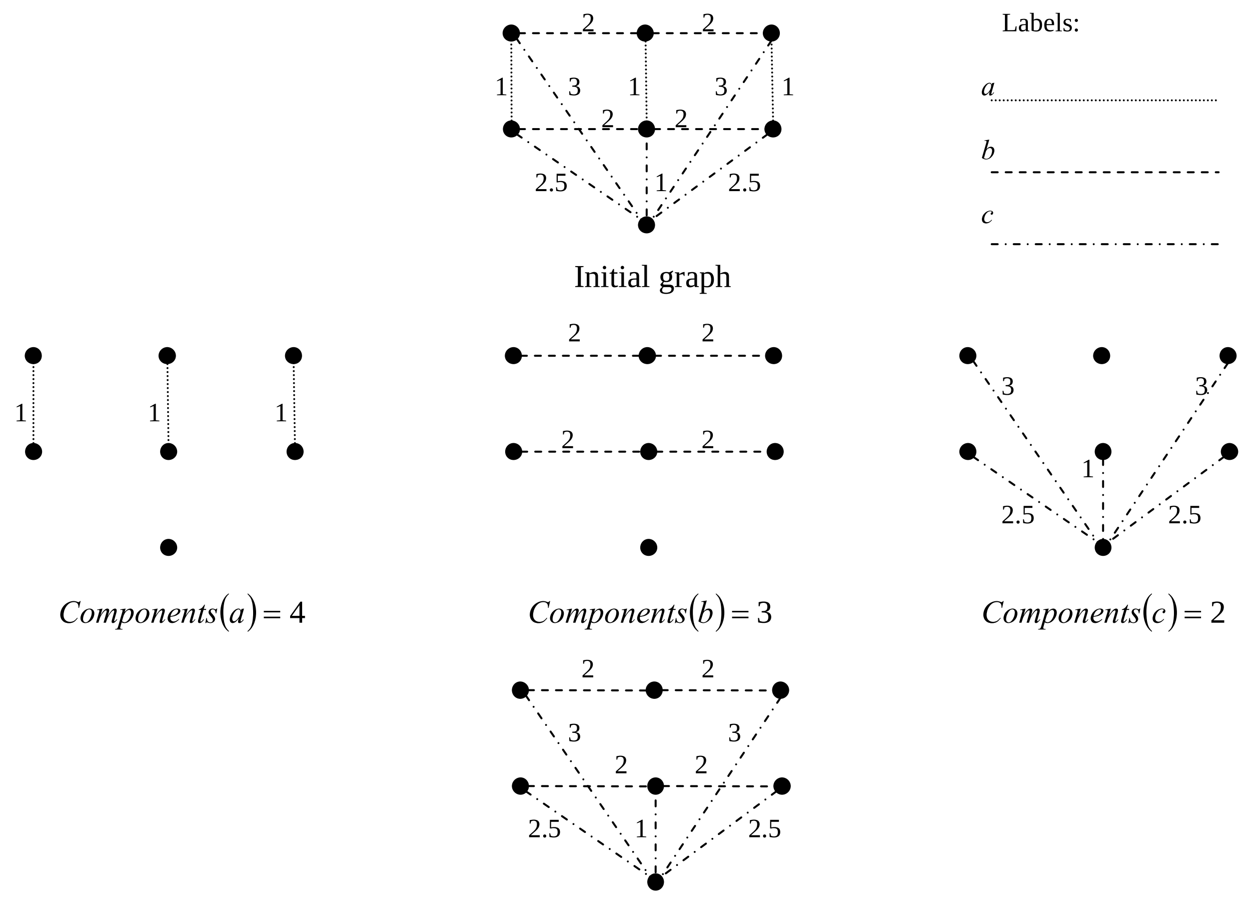
Before considering the cost constraint, which we need to satisfy, we have found it useful to try to improve the quality of the initial solution slightly. To this aim, we swap used and unused labels in a given solution in order to decrease the cost of a minimum spanning tree on the selected labels. We scan through the labels in the current solution. We iteratively consider all unused labels and attempt to swap a given label in the current solution with an unused label if it results in an improvement (i.e., the cost of the minimum spanning tree on the graph induced by the labels decreases). As soon as an improvement is found, it is implemented and the next used label in the current solution is examined. This is illustrated with an example in Figure 3.2. Consider $A = \{b, c\}$ we have MSTCOST(A) = 12 and MSTCOST($\{A \setminus b\} \cup a\}$) = 9. Therefore, we remove label b and add label a to the representation of our solution.

At this stage, it is possible that the set of labels in the current solution does not result in a tree that satisfies the budget constraint. To find a set of labels that does, we iteratively add labels to the current set of labels by choosing the label that, when added, results in the lowest cost minimum spanning tree. In other words the label to be added is selected from

$$S = \{t \in (L \setminus R) : \min \quad MSTCOST(R \cup \{t\})\}, \tag{3.13}$$

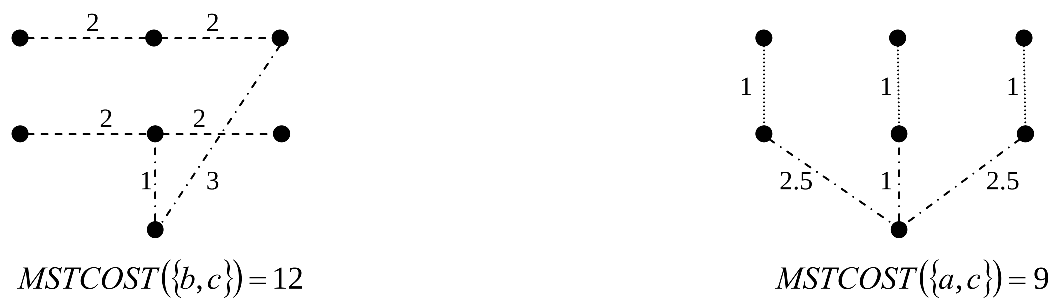
and ties are broken randomly. We continue adding labels to the current solution in this fashion until we obtain a minimum spanning tree satisfying the cost constraint. (Recall since $B \geq$ MSTCOST(L) a feasible solution exists and the initialization phase will find one).

Figure 3.1. An example illustrating the selection of labels for the initial connected subgraph



Connected graph with labels *b* and *c*.

Figure 3.2. An example for the swap of used and unused labels



3.3.2. Shaking Phase

The shaking phase follows the VNS paradigm and dynamically expands the neighborhood search area. Suppose *R* denotes the current solution (it really denotes the labels in the current solution, but as explained earlier it suffices to focus on labels). In this step, we use randomization to select a solution that is in the size k neighborhood of the solution R, i.e., $N_k(R)$. Specifically $N_k(R)$ is defined as the set of labels that can be obtained from R by performing a sequence of exactly k additions and/or deletions of labels. So $N_1(R)$ is the set of labels obtained from R by either adding exactly one label from R, or deleting exactly one label from R. $N_2(R)$ is the set of labels obtained from R by either adding exactly two labels, or deleting exactly two labels, or adding exactly one label and deleting exactly one label.

The shaking phase may result in the selection of labels that do not result in a connected graph, or result in a minimum cost spanning tree that does not meet the budget constraint. If the set of labels results in a graph that is not connected, we add labels that are not in the current solution one by one, at random until the graph is connected. If the minimum spanning tree on the selected labels does not meet the budget constraint, we iteratively add labels to the current set of labels by choosing the label that when added results in the lowest cost minimum spanning tree.

3.3.3. Local Search Phase

The local search phase consists of two parts. In the first part, the algorithm tries to swap each of the labels in the current solution with an unused one if it results in a lower minimum spanning tree cost. To this aim, it iteratively considers the labels in the solution and tests all possible exchanges of a given label with unused labels until it finds an exchange resulting in a lower MST cost. If we find such an exchange, we implement it (i.e., we ignore the remaining unused labels) and proceed to the next label in our solution. Obviously, a label remains in the solution if the algorithm cannot find a label swap resulting in an improvement.

The second part of the local search phase tries to improve the quality of the solution by removing labels from the current set of labels. It iteratively, tries to remove each label. If the resulting set of labels provides a minimum spanning tree whose cost does not exceed the

Algorithm 3.1. Variable Neighborhood Search Algorithm for the CCMLST Problem

```
VNS for CCMLST
  $R = \varphi$;
  Initialization Procedure(G,R);
  Local Search (G,R);
  While Termination criterion not met
      $k = 1$;
      While $k \leq 5$
          $\overline{R}$ = Shaking_Phase (G,k,R);
          While Components ($\overline{R}$) > 1
              Select at random a label $u \in L \setminus \overline{R}$ and add it to $\overline{R}$;
          End;
          While MSTCOST($\overline{R}$) > B
              $S = \{t \in (L \setminus \overline{R}) : \min\ MSTCOST(\overline{R} \cup \{t\})\}$;
              Select at random a label $u \in S$ and add it to $\overline{R}$;
          End;
          Local Search$(G, \overline{R})$;
          If $|\overline{R}| < |R|$ Then $R = \overline{R}$ and $k = 1$ Else $k = k+1$;
      End ;
  End.

Initialization Procedure(G,R)
  While Components (R) > 1
      $S = \{t \in (L \setminus R) : \min\ Components\ (R \cup \{t\})\}$;
      Select at random a label $u \in S$ and add it to R;
  End.
  Consider the labels $i \in R$ one by one;
      Swap the label i with the first unused label that strictly lowers the MST cost;
  End;
  While MSTCOST(R) > B
      $S = \{t \in (L \setminus R) : \min\ MSTCOST(R \cup \{t\})\}$;
      Select at random a label $u \in S$ and add it to R;
  End.

Shaking_Phase (G,k,R)
  For $i = 1,\ldots,k$
      $r$ = random(0,1) ;
      If $r \leq 0.5$ Then Delete at random a label from R Else Add at random a label to R;
  End;
  Return(R).

Local Search(G,R)
  Consider the labels $i \in R$ one by one;
      Swap the label i with the first unused label that strictly lowers the MST cost;
  End;
  Consider the labels $i \in R$ one by one;
      Delete label i from R, i.e. $R = R \setminus i$;
      If Components (R) > 1 or MSTCOST(R) > B Then $R = R \cup i$;
  End.
```

budget we permanently remove the label, and continue. Otherwise, the label remains in the solution. Our Variable Neighborhood Search algorithm is outlined in Algorithm 3.1.

3.3.4. Discussion of Algorithmic Parameter Choices

We now discuss the different components of our VNS algorithm and identify the rationale for the different choices made within the algorithm. In constructing the initial solution, we first chose labels to construct a connected graph. Our choice was to choose the label that resulted in the greatest decrease in the number of connected components. An alternative is to simply add labels randomly. We conducted experiments with these 2 variants for connectivity of the initial solution, and found that choosing to add labels so that they result in the greatest decrease in the number of connected components provided significantly better solutions. Table 3.1 identifies the different parts of the VNS algorithm for the CCMLST, the parameter or algorithmic choices within each part, and the choice that resulted in the best solution. For example, when comparing the use of swapping against no swapping in the construction of the initial solution, swapping labels provided the best results. In Table 3.1, the parameter cost constraint refers to the scenario where the cost of the solution is strictly greater than *B*. Here, in order to reduce the cost of the solution, Min Avg Cost considers the unused label with the smallest average cost to add to the solution. Feasibility post shaking phase refers to the part of the algorithm where feasibility of the solution obtained by the shaking phase is restored. As can be seen from Table 3.1, the best algorithmic choices within the

Table 3.1. Parameter/Algorithmic Choices within the VNS Procedure for the CCMLST Problem

Phase	Parameter Varied	Values	Best Value
Initial Solution	Connectivity	Random, Min Components	Min Componenets
	Swap	Yes/No	Yes
	Cost constraint	Random, Min Avg Cost, Min MST Cost	Min MST Cost
	Local search applied to Initial Solution	Yes/No	Yes
Feasibility post Shaking Phase	Connectivity	Random, Min Components	Random
	Cost Constraint	Min Avg Cost, Min MST Cost	Min MST Cost
Local Search	Swap	Yes/No	Yes

different parts of the VNS algorithm are the ones used in our VNS algorithm described in Algorithm 3.1.

Before we turn our attention to our computational experiments we must note the recent work by Consoli et al. [22] on the MLST problem that includes a VNS method for the MLST problem. We briefly compare the two VNS methods (albeit on different problems). Both of the methods follow the main structure of VNS. Consequently, at a high level they are somewhat similar, but they are different in several details. For example, in the initialization phase instead of generating a random solution as done in the paper by Consoli et al. [22] we use a more involved procedure to generate an initial solution by considering the set of labels that produces fewer components and smaller MST cost. Additionally, we use local search to improve the solution found in the initialization phase prior to applying VNS. Finally, since the objectives of the CCMLST problem is somewhat different from the MLST problem, our local search operator is quite different from that in Consoli et al. [22].

3.4. Applying Heuristics for the LCMST Problem to the CCMLST Problem by Means of Binary Search

We first describe the heuristics of Xiong et al. [82] for the LCMST problem. We then describe how any heuristic for the LCMST problem may be applied to the CCMLST problem by using binary search. This allows us to apply the heuristics of Xiong et al. [82] to the CCMLST problem.

The first proposed heuristic, LS1, by Xiong et al. [82] for the LCMST, begins with an arbitrary feasible solution, $A = \{a_1, a_2, ..., a_K\}$, of K labels. Then, it starts a replacement loop as follows. It first attempts to replace a_1 by the label in $L \setminus A$ that gives the greatest reduction in MST cost. In other words, it checks each label in $L \setminus A$ and selects the label $t \in L \setminus A$ that results in the lowest value of MSTCOST($\{A \setminus a_1\} \cup t$). If it finds an improvement (i.e., MSTCOST($\{A \setminus a_1\} \cup t$)<MSTCOST($A$)) it sets $a_1 = t$, otherwise it leaves a_1 unchanged. Next, it repeats the procedure with label a_2 attempting to replace it by the label in $L \setminus A$ that gives the greatest reduction in MST cost. It continues the replacement loop in this fashion until it considers replacing label a_K by the label in $L \setminus A$ that gives the greatest reduction in MST cost. In one replacement loop all of the K labels in A are considered for replacement. This continues until no cost improvement can be made between two consecutive replacement loops [82].

In the second heuristic, LS2, the algorithm starts with an arbitrary feasible solution $A = \{a_1, a_2, ..., a_K\}$ of K labels. Then, it attempts to find improvements as follows. It adds a label in $a_{K+1} \in L \setminus A$ to A. This results in a solution with K+1 labels, which is not feasible. Consequently, it chooses amongst these K+1 labels the label to delete that results in the smallest MST cost. To find the best possible improvement of this type, it searches among all possible additions of labels in $L \setminus A$ to A and selects the one that provides the greatest improvement in cost. The procedure continues until no further improvement is found [82].

In the GA proposed by Xiong et al. [82], a queen-bee crossover is applied. This approach has often been found to outperform more traditional crossover operators. In each generation, the best chromosome is declared the queen-bee and crossover is only allowed between the queen-bee and other chromosomes. For more details regarding crossover, mutation, etc., see [82].

Binary search is a popular algorithmic paradigm (see page 171 of [61]) that efficiently searches through an interval of integer values to find the smallest (or largest) integer that satisfies a specified property. It starts with an interval 1,…,l. In each step it checks whether the midpoint of the interval satisfies the specified property. If so, one may conclude the smallest integer that satisfies the specified property is in the lower half of the interval, otherwise the smallest integer that satisfies the specified property is in the upper half of the interval. It recursively searches through the interval in which the solution lies, until (after $\lceil \log_2 l \rceil$ steps) the interval contains only one integer value and the procedure terminates. We now describe how to use the binary search method to apply any algorithm for the LCMST problem to the CCMLST problem. Let ALG denote any heuristic for the LCMST problem. It takes as input a graph *G* and a threshold *K*. ALG attempts to find a minimum cost spanning tree that uses at most *K* labels. Either it returns a feasible solution, i.e., a set of at most *K* labels, or it indicates that no feasible solution has been found by sending back an empty set of labels. The cost of the solution can then be determined by MSTCOST(*R*) where *R* denotes the set of labels. Note that MSTCOST(*R*) is infinity if *R* is an empty set.

The details of the binary search method as applied to the CCMSLT problem are provided in Algorithm 3.2. The *lower* value for the number of labels is set to 1 and the *upper* value for the number of labels is set to *l* (the total number of labels). As is customary in binary search, ALG is executed setting the threshold on the number of labels to $\frac{lower + upper}{2}$. Note that since the number of labels must be integer, ALG rounds down any non-integral value of the threshold *K*. Essentially, anytime the cost of the tree found by ALG exceeds the budget *B*, we need more labels and increase the *lower* value to $\frac{lower + upper}{2}$. Anytime the cost of the tree found by ALG is within budget, we decrease the value of *upper* to $\frac{lower + upper}{2}$.

Algorithm 3.2. Binary Search Method for the CCMLST Problem

```
Begin
  Set $lower = 1$ and $upper = l$;
  While $(upper - lower) \geq 1$
      $mid = \frac{lower + upper}{2}$;
      $R$ = ALG($G,mid$);
      If $MSTCOST(R) > B$ Then $lower = mid$ Else $upper = mid$;
  End
  Output $R$ = ALG($G,upper$);
End
```

3.5. Computational Results

In this section, we report on an extensive set of computational experiments on the CCMLST problem. All heuristics have been tested on a Pentium IV machine with a 2.61 GHz processor and 2 GB RAM, under the Windows operating system. We also use ILOG CPLEX 10.2 to solve the MIP formulation.

The two parameters that are adjustable within the VNS procedure are the value of *k* (the size of the largest neighborhood $N_k(R)$ in the VNS method), and *Iter*, the number of iterations in which the algorithm is not able to improve the best known solution (which is the termination criterion). Increasing *k*, increases the size of the neighborhood but also increases the running time. We found that setting *k*=5 provides the best results without a significant increase in running time. Additionally, as the value of *Iter* is increased the running time of the algorithm is increase, though the quality of the solution improves. We found that setting *Iter*=10 provides the best results in a reasonable amount of running time.

We now describe how we generated our datasets, and then discuss our computational experience on these datasets for the CCMLST problem.

3.5.1. Datasets

Xiong et al. [82] created a set of test instances for the LCMST problem. These include 37 small instances with 50 nodes or less, 11 medium-sized instances that range from 100 to 200 nodes, and one large instance with 500 nodes. All of these instances are complete graphs. We adapted the instances created by Xiong et al. [82] to the CCMLST as follows. Essentially, for the labeled graph instance, we create a set of different budget values. These budget values are selected starting from slightly more than MSTCOST(*L*) with increments of approximately 500 units. The small instances in Xiong et al. were somewhat limited in the sense that the number of labels is equal to the number of nodes in the graph. Consequently, using Xiong et al.'s code, we generated additional labeled graphs where the number of labels was less than the number of nodes in the graph.

The main aim of using the small instances was to test the quality of the VNS method and the binary search method on instances where we can find the optimal solution. Consequently, we restricted our attention on these small instances to problems where we were able to compute the optimal solution using CPLEX on our MIP formulation. In this way, we created 104 small instances (with 10 to 50 nodes) where the optimal solution is known, and 60 medium-sized instances (with 100 to 200 nodes). We adapted the TSPLIB instances to create 27 large instances. We used the labeled graph generated from the TSPLIB instance, and created three instances from each labeled graph by varying the budget value. Specifically, we used budget values of 1.3, 1.6, and 1.9 times MSTCOST(L).

3.5.2. Results

The results on the 191 CCMLST instances are described in Tables 3.2 through 3.6 for the small instances, Tables 3.7 through 3.9 for the medium-sized instances, and Table 3.10 for the large instances.

On the 104 small instances, the VNS method found the optimal solution in all cases (recall that the optimal solution is known in all of these instances), while LS1, LS2, and GA generated the optimal solution 100, 100, and 102 times, respectively, out of the 104 instances. The average running time of the VNS method was 0.05 seconds, while LS1, LS2, and GA took 0.06, 0.07, and 0.18 seconds respectively. For the small and medium-sized instances, the termination criterion used was 10 iterations without an improvement. On the 60 medium-sized instances, the VNS method generated the best solution in 59 out of the 60 instances, while LS1, LS2, and GA generated the best solution 46, 50, and 50 times, respectively, out of the 60 instances. The average running time of the VNS method was 20.59 seconds, while LS1, LS2, and GA took 63.52, 68.75, and 62.19 seconds, respectively. This indicates that the VNS method finds better solutions in a greater number of instances much more rapidly than any of the three comparative procedures. For the large instances the termination criterion used was a specified running time which is shown in the tables with the

Table 3.2. VNS, GA, LS1, and LS2 for the CCMLST Problem on 10 nodes

# Labels	Cost Restriction	Exact method		LS1			LS2			GA			VNS		
		Labels	Time	Labels	Gap	Time	Labels	Gap	Time	Labels	Gap	Time	Labels	Gap	Time
10	7000	2	0.23	2	0%	0.00	2	0%	0.01	2	0%	0.00	2	0%	0.02
	6500	2	0.02	2	0%	0.00	2	0%	0.00	2	0%	0.02	2	0%	0.00
	6000	2	0.03	2	0%	0.00	2	0%	0.01	2	0%	0.00	2	0%	0.02
	5500	2	0.02	2	0%	0.01	2	0%	0.01	2	0%	0.02	2	0%	0.00
	5000	2	0.11	2	0%	0.00	2	0%	0.01	2	0%	0.00	2	0%	0.00
	4500	2	0.09	2	0%	0.00	2	0%	0.00	2	0%	0.00	2	0%	0.00
	4000	2	0.05	2	0%	0.00	2	0%	0.01	2	0%	0.00	2	0%	0.00
	3500	2	0.06	2	0%	0.00	2	0%	0.00	2	0%	0.02	2	0%	0.00
	3000	3	0.03	3	0%	0.00	3	0%	0.00	3	0%	0.00	3	0%	0.00
	2500	4	0.13	4	0%	0.01	4	0%	0.01	4	0%	0.02	4	0%	0.00

Table 3.3. VNS, GA, LS1, and LS2 for the CCMLST Problem on 20 nodes

# Labels	Cost Restriction	Exact method		LS1			LS2			GA			VNS		
		Labels	Time	Labels	Gap	Time	Labels	Gap	Time	Labels	Gap	Time	Labels	Gap	Time
10	7500	2	3.33	2	0%	0.01	2	0%	0.00	2	0%	0.02	2	0%	0.02
	7000	2	2.05	2	0%	0.01	2	0%	0.01	2	0%	0.02	2	0%	0.00
	6500	2	3.78	2	0%	0.01	2	0%	0.01	2	0%	0.00	2	0%	0.02
	6000	2	1.72	2	0%	0.01	2	0%	0.00	2	0%	0.00	2	0%	0.02
	5500	2	2.70	2	0%	0.01	2	0%	0.01	2	0%	0.00	2	0%	0.00
	5000	3	2.80	3	0%	0.01	3	0%	0.01	3	0%	0.02	3	0%	0.00
	4500	3	6.72	3	0%	0.01	3	0%	0.01	3	0%	0.00	3	0%	0.02
	4000	4	29.02	4	0%	0.01	4	0%	0.01	4	0%	0.02	4	0%	0.02
	3500	5	7.08	5	0%	0.01	5	0%	0.01	5	0%	0.02	5	0%	0.02
	3050	8	0.84	8	0%	0.01	8	0%	0.01	8	0%	0.03	8	0%	0.02
20	7500	2	0.69	2	0%	0.02	2	0%	0.02	2	0%	0.02	2	0%	0.02
	7000	2	0.34	2	0%	0.02	2	0%	0.02	2	0%	0.02	2	0%	0.00
	6500	2	1.14	2	0%	0.02	2	0%	0.02	2	0%	0.00	2	0%	0.00
	6000	3	2.98	3	0%	0.02	3	0%	0.02	3	0%	0.02	3	0%	0.00
	5500	3	15.98	3	0%	0.00	3	0%	0.02	3	0%	0.02	3	0%	0.02
	5000	4	4.30	4	0%	0.02	4	0%	0.02	4	0%	0.03	4	0%	0.02
	4500	5	18.16	5	0%	0.02	5	0%	0.02	5	0%	0.03	5	0%	0.02
	4000	6	12.28	6	0%	0.02	6	0%	0.02	6	0%	0.02	6	0%	0.02
	3500	8	3.83	8	0%	0.02	8	0%	0.02	8	0%	0.03	8	0%	0.02
	3050	11	0.88	11	0%	0.02	11	0%	0.02	11	0%	0.06	11	0%	0.02

Table 3.4. VNS, GA, LS1, and LS2 for the CCMLST Problem on 30 nodes

# Labels	Cost	Exact method		LS1			LS2			GA			VNS		
	Restriction	Labels	Time	Labels	Gap	Time	Labels	Gap	Time	Labels	Gap	Time	Labels	Gap	Time
10	8000	2	31.61	2	0%	0.01	2	0%	0.01	2	0%	0.03	2	0%	0.02
	7500	2	77.45	2	0%	0.01	2	0%	0.01	2	0%	0.02	2	0%	0.02
	7000	2	20.59	2	0%	0.01	2	0%	0.01	2	0%	0.02	2	0%	0.02
	6500	2	41.84	2	0%	0.01	2	0%	0.01	2	0%	0.03	2	0%	0.02
	6000	3	45.59	3	0%	0.01	3	0%	0.01	3	0%	0.03	3	0%	0.02
	5500	3	134.80	3	0%	0.01	3	0%	0.01	3	0%	0.03	3	0%	0.02
	5000	3	56.22	3	0%	0.01	3	0%	0.01	3	0%	0.03	3	0%	0.02
	4500	4	33.39	4	0%	0.02	4	0%	0.01	4	0%	0.03	4	0%	0.02
	4000	5	15.42	5	0%	0.01	5	0%	0.01	5	0%	0.03	5	0%	0.03
	3500	8	26.23	8	0%	0.02	8	0%	0.01	8	0%	0.06	8	0%	0.02
20	8000	3	69.92	3	0%	0.02	3	0%	0.02	3	0%	0.08	3	0%	0.02
	7500	3	132.47	3	0%	0.02	3	0%	0.02	3	0%	0.08	3	0%	0.02
	7000	3	9.69	3	0%	0.02	3	0%	0.03	3	0%	0.08	3	0%	0.02
	6500	4	31.55	4	0%	0.03	4	0%	0.02	4	0%	0.09	4	0%	0.02
	6000	4	76.02	4	0%	0.02	4	0%	0.02	4	0%	0.08	4	0%	0.03
	5500	5	111.28	5	0%	0.03	5	0%	0.03	5	0%	0.09	5	0%	0.03
	5000	5	74.59	5	0%	0.03	5	0%	0.03	5	0%	0.09	5	0%	0.04
	4500	6	46.27	6	0%	0.03	6	0%	0.03	6	0%	0.09	6	0%	0.03
	4000	8	22.90	8	0%	0.03	8	0%	0.03	8	0%	0.13	8	0%	0.04
	3500	14	177.91	14	0%	0.04	14	0%	0.04	14	0%	0.17	14	0%	0.03
30	8000	3	19.12	3	0%	0.03	3	0%	0.03	3	0%	0.06	3	0%	0.05
	7500	4	33.04	4	0%	0.03	4	0%	0.03	4	0%	0.08	4	0%	0.03
	7000	4	62.60	4	0%	0.05	4	0%	0.03	4	0%	0.08	4	0%	0.05
	6500	4	16.01	4	0%	0.03	4	0%	0.05	4	0%	0.08	4	0%	0.05
	6000	5	4.56	5	0%	0.03	5	0%	0.05	5	0%	0.08	5	0%	0.05
	5500	6	45.34	6	0%	0.06	6	0%	0.06	6	0%	0.13	6	0%	0.05
	5000	7	82.30	7	0%	0.05	7	0%	0.05	7	0%	0.13	7	0%	0.06
	4500	8	95.27	8	0%	0.06	8	0%	0.08	8	0%	0.13	8	0%	0.06
	4000	10	55.84	11	10%	0.09	11	10%	0.08	11	10%	0.13	10	0%	0.06
	3500	15	385.09	15	0%	0.09	15	0%	0.09	15	0%	0.30	15	0%	0.09

- The best solutions are in bold.

Table 3.5. VNS, GA, LS1, and LS2 for the CCMLST Problem on 40 nodes

# Labels	Cost Restriction	Exact method		LS1			LS2			GA			VNS		
		Labels	Time	Labels	Gap	Time	Labels	Gap	Time	Labels	Gap	Time	Labels	Gap	Time
10	9000	2	6125.77	2	0%	0.01	2	0%	0.01	2	0%	0.03	2	0%	0.03
	8500	2	24107.41	2	0%	0.01	2	0%	0.01	2	0%	0.03	2	0%	0.02
	8000	2	74551.36	2	0%	0.01	2	0%	0.01	2	0%	0.03	2	0%	0.03
	7500	3	12550.08	3	0%	0.01	3	0%	0.01	3	0%	0.06	3	0%	0.02
	7000	3	1734.30	3	0%	0.01	3	0%	0.01	3	0%	0.05	3	0%	0.03
	6500	3	1049.59	3	0%	0.02	3	0%	0.01	3	0%	0.05	3	0%	0.03
20	9000	3	3344.61	3	0%	0.03	3	0%	0.04	3	0%	0.09	3	0%	0.03
	8500	3	3709.23	3	0%	0.03	3	0%	0.04	3	0%	0.09	3	0%	0.03
	8000	4	1064.82	4	0%	0.04	4	0%	0.04	4	0%	0.09	4	0%	0.05
	7500	4	1475.95	4	0%	0.04	4	0%	0.04	4	0%	0.09	4	0%	0.05
40	9000	5	1514.57	5	0%	0.12	5	0%	0.11	5	0%	0.19	5	0%	0.09
	8500	5	2223.00	5	0%	0.14	5	0%	0.11	5	0%	0.20	5	0%	0.09
	8000	6	17008.94	6	0%	0.14	6	0%	0.12	6	0%	0.20	6	0%	0.09
	7500	6	541.14	7	17%	0.16	6	0%	0.12	6	0%	0.20	6	0%	0.11
	7000	7	424.28	7	0%	0.16	7	0%	0.16	7	0%	0.27	7	0%	0.13
	6500	8	1713.71	8	0%	0.20	8	0%	0.17	8	0%	0.31	8	0%	0.14
	6000	9	2519.92	9	0%	0.20	9	0%	0.20	9	0%	0.31	9	0%	0.16
	5500	11	1437.73	11	0%	0.22	11	0%	0.22	11	0%	0.41	11	0%	0.16
	5000	14	12402.04	14	0%	0.33	14	0%	0.27	14	0%	0.39	14	0%	0.19

- The best solutions are in bold.

Table 3.6. VNS, GA, LS1, and LS2 for the CCMLST Problem on 50 nodes

# Labels	Cost	Exact method		LS1			LS2			GA			VNS		
	Restriction	Labels	Time	Labels	Gap	Time	Labels	Gap	Time	Labels	Gap	Time	Labels	Gap	Time
10	9500	2	9798.88	2	0%	0.02	2	0%	0.02	2	0%	0.6	2	0%	0.02
	9000	2	8972.91	2	0%	0.02	2	0%	0.02	2	0%	0.6	2	0%	0.02
	8500	2	8984.60	2	0%	0.02	2	0%	0.02	2	0%	0.6	2	0%	0.02
	8000	3	1443.76	3	0%	0.02	3	0%	0.02	3	0%	0.6	3	0%	0.03
	7500	3	24729.7	3	0%	0.02	3	0%	0.02	3	0%	0.6	3	0%	0.03
	7000	3	28404.6	3	0%	0.02	3	0%	0.02	3	0%	0.8	3	0%	0.03
	6500	4	46791.0	4	0%	0.02	4	0%	0.02	4	0%	0.8	4	0%	0.05
	6000	4	6459.06	4	0%	0.02	4	0%	0.02	4	0%	0.9	4	0%	0.05
20	9500	3	9216.76	3	0%	0.07	3	0%	0.06	3	0%	0.16	3	0%	0.05
	9000	3	1880.70	3	0%	0.05	3	0%	0.06	3	0%	0.17	3	0%	0.06
	8500	4	7873.89	4	0%	0.06	4	0%	0.07	4	0%	0.19	4	0%	0.08
	8000	4	12730.2	4	0%	0.06	4	0%	0.07	4	0%	0.19	4	0%	0.08
	7500	5	7572.67	5	0%	0.12	5	0%	0.07	5	0%	0.19	5	0%	0.08
	7000	5	49912.1	5	0%	0.08	6	20%	0.07	5	0%	0.22	5	0%	0.08
	6500	6	22651.0	6	0%	0.07	6	0%	0.09	6	0%	0.28	6	0%	0.09
30	9500	5	11111.3	5	0%	0.12	5	0%	0.14	5	0%	0.31	5	0%	0.09
	9000	5	12638.7	5	0%	0.12	5	0%	0.14	5	0%	0.30	5	0%	0.09
	8500	5	15358.5	5	0%	0.12	5	0%	0.14	5	0%	0.30	5	0%	0.13
	8000	6	36336.0	6	0%	0.13	6	0%	0.15	6	0%	0.30	6	0%	0.13
50	9000	7	1322.60	7	0%	0.33	7	0%	0.37	7	0%	0.52	7	0%	0.22
	8500	8	10455.0	8	0%	0.31	8	0%	0.39	8	0%	0.59	8	0%	0.22
	8000	8	62948.1	9	13%	0.31	9	13%	0.44	8	0%	0.61	8	0%	0.27
	7500	9	30317.9	9	0%	0.33	9	0%	0.42	9	0%	0.55	9	0%	0.25
	7000	11	97384.1	11	0%	0.50	11	0%	0.53	11	0%	0.98	11	0%	0.27
	6500	12	119300.	13	8%	0.53	13	8%	0.58	13	8%	0.97	12	0%	0.31

- The best solutions are in bold.

Table 3.7. VNS, GA, LS1, and LS2 for the CCMLST Problem on 100 nodes

# Labels	Cost Restriction	LS1		LS2		GA		VNS	
		Labels	Time	Labels	Time	Labels	Time	Labels	Time
50	11500	**11**	2.13	**11**	2.33	**11**	3.61	**11**	1.05
	11000	**12**	2.28	**12**	2.39	**12**	3.61	**12**	1.09
	10500	**13**	3.32	**13**	3.43	**13**	4.53	**13**	1.22
	10000	**14**	3.48	**14**	3.29	**14**	4.31	**14**	1.25
	9500	16	3.60	16	3.98	**15**	4.17	**15**	1.36
	9000	18	3.41	**17**	3.84	**17**	5.22	**17**	1.47
	8500	**19**	4.23	20	4.56	**19**	5.88	**19**	1.61
	8000	**22**	3.82	**22**	4.35	23	6.89	**22**	2.70
	7500	**26**	3.63	27	4.71	27	9.17	**26**	2.61
	7000	**36**	3.78	**36**	3.65	**36**	9.13	**36**	1.36
100	11500	**16**	7.22	**16**	8.94	**16**	7.95	**16**	2.73
	11000	**17**	8.28	**17**	9.98	**17**	9.95	**17**	3.20
	10500	20	8.58	**19**	10.33	**19**	9.86	**19**	3.13
	10000	**21**	10.41	**21**	11.01	**21**	10.31	**21**	3.13
	9500	**23**	9.67	**23**	11.36	**23**	12.09	**23**	3.59
	9000	**25**	12.68	**25**	11.68	**25**	11.86	**25**	3.83
	8500	**28**	14.09	**28**	12.01	**28**	11.42	**28**	4.39
	8000	**32**	15.07	**32**	12.95	33	15.81	**32**	4.84
	7500	39	17.44	**38**	17.66	39	16.09	**38**	5.23
	7000	**50**	18.72	**50**	18.94	**50**	17.42	**50**	5.86

- The best solutions are in bold.

Table 3.8. VNS, GA, LS1, and LS2 for the CCMLST Problem on 150 nodes

# Labels	Cost Restriction	LS1		LS2		GA		VNS	
		Labels	Time	Labels	Time	Labels	Time	Labels	Time
75	13000	**17**	14.96	18	12.83	**17**	17.00	**17**	4.78
	12500	**18**	14.58	**18**	12.51	19	21.66	19	4.91
	12000	**20**	23.81	**20**	17.57	**20**	21.30	**20**	5.44
	11500	**22**	24.91	**22**	18.00	**22**	22.02	**22**	6.06
	11000	25	22.48	25	19.98	**24**	22.64	**24**	10.36
	10500	**27**	22.62	**27**	20.55	**27**	35.56	**27**	6.55
	10000	31	18.92	31	20.08	**30**	35.53	**30**	7.84
	9500	**35**	25.39	**35**	28.55	**35**	39.41	**35**	7.38
	9000	**41**	25.09	**41**	22.48	**41**	35.78	**41**	7.61
	8500	**55**	18.10	**55**	19.56	**55**	59.47	**55**	5.94
150	13000	**28**	48.99	**28**	50.86	**28**	36.38	**28**	12.80
	12500	**30**	52.69	**30**	61.85	**30**	36.41	**30**	13.92
	12000	**32**	59.57	**32**	51.53	**32**	36.17	**32**	16.52
	11500	**35**	62.60	**35**	53.18	**35**	38.17	**35**	16.92
	11000	**38**	62.58	**38**	66.05	**38**	42.80	**38**	18.20
	10500	42	75.69	42	85.19	43	43.36	**41**	37.38
	10000	47	73.85	**46**	86.46	**46**	45.80	**46**	33.22
	9500	**53**	75.80	**53**	90.01	**53**	51.80	**53**	23.42
	9000	**62**	78.06	**62**	80.87	**62**	58.53	**62**	25.27
	8500	**79**	93.13	**79**	98.56	**79**	79.47	**79**	26.19

- The best solutions are in bold.

Table 3.9. VNS, GA, LS1, and LS2 for the CCMLST Problem on 200 nodes

# Labels	Cost Restriction	LS1		LS2		GA		VNS	
		Labels	Time	Labels	Time	Labels	Time	Labels	Time
100	14000	**22**	37.67	**22**	46.45	**22**	50.78	**22**	13.05
	13500	**24**	39.91	**24**	46.96	**24**	45.77	**24**	14.41
	13000	**26**	55.61	**26**	63.99	**26**	59.70	**26**	15.33
	12500	**29**	59.08	**29**	60.53	**29**	64.97	**29**	16.34
	12000	**32**	66.69	**32**	69.67	**32**	62.81	**32**	17.47
	11500	36	54.23	**35**	60.70	**35**	61.25	**35**	19.44
	11000	**40**	70.93	**40**	80.11	**40**	83.45	**40**	20.30
	10500	**46**	72.70	**46**	75.86	**46**	123.86	**46**	39.14
	10000	**55**	81.17	**55**	87.80	**55**	138.72	**55**	22.73
	9500	**70**	73.51	**70**	65.46	**70**	194.38	**70**	18.88
200	14000	**35**	188.07	**35**	179.56	**35**	136.59	**35**	37.94
	13500	38	174.69	**37**	198.79	**37**	134.84	**37**	41.97
	13000	**40**	184.59	**40**	201.66	41	134.27	**40**	64.90
	12500	44	171.44	44	193.66	**43**	130.80	**43**	47.70
	12000	**48**	172.10	**48**	211.57	49	163.52	**48**	51.02
	11500	54	230.51	**53**	246.54	54	169.02	**53**	56.44
	11000	60	215.70	60	236.61	60	178.59	**59**	133.78
	10500	68	248.04	68	312.48	**67**	214.44	**67**	69.19
	10000	**80**	280.02	**80**	300.20	**80**	290.39	**80**	73.61
	9500	**101**	284.59	**101**	308.48	**101**	334.92	**101**	115.42

- The best solutions are in bold.

Table 3.10. VNS, GA, LS1, and LS2 for the CCMLST Problem on Large Datasets

# Nodes, # Labels	Cost Restriction	LS1		LS2		GA		VNS		
		Labels	Time	Labels	Time	Labels	Time	Labels	Time	Max time
532, 266	98800	**70**	5182	**70**	4890	**70**	3131	**70**	111	300
	121600	**44**	3252	**44**	3505	**44**	2500	**44**	103	300
	144400	**32**	2468	**32**	3013	**32**	2157	**32**	81	300
574, 287	42250	86	6830	86	7103	86	4234	**85**	278	300
	52000	57	4999	57	51333	57	2945	**56**	190	300
	61750	**42**	3796	**42**	3753	**42**	2600	**42**	86	300
575, 287	8190	79	6697	79	6575	79	4664	**78**	233	300
	10080	**47**	4371	**47**	4585	**47**	2971	**47**	133	300
	11970	**33**	3272	**33**	3897	**33**	2695	**33**	70	300
654, 327	39000	**54**	6839	55	7332	55	4279	**54**	203	700
	48000	36	5422	36	5622	36	3494	**35**	257	700
	57000	28	4373	28	4542	28	3117	**27**	199	700
657, 328	55900	**95**	15237	**95**	16660	**95**	8763	**95**	603	700
	68800	**61**	10451	**61**	10294	**61**	6158	**61**	249	700
	81700	**44**	8139	**44**	8458	**44**	4789	**44**	183	700
666, 333	3510	**75**	9763	**75**	10329	76	6466	**75**	477	700
	4320	49	6815	49	6894	49	4089	**48**	647	700
	5130	**35**	5208	**35**	5340	**35**	3515	**35**	232	700
724, 362	49400	102	18755	102	24650	102	13053	**101**	693	1000
	60800	**65**	15802	**65**	17318	66	8995	**65**	478	1000
	72200	**47**	12550	**47**	12856	**47**	7522	**47**	259	1000
783, 391	10660	**111**	26056	**111**	30572	**111**	16231	**111**	975	1000
	13120	**68**	19425	69	19754	69	10605	69	492	1000
	15580	**48**	17509	**48**	16581	49	9539	**48**	358	1000
1000, 500	20800000	141	156482	**140**	188704	141	76294	**140**	2923	4000
	25600000	**86**	134765	**86**	119253	**86**	49080	**86**	3920	4000
	30400000	**62**	101576	**62**	103167	**62**	45996	**62**	3587	4000

- The best solutions are in bold.

computational results. On the 27 large instances, the VNS method generated the best solution in 26 out of the 27 instances, while LS1, LS2, and GA generated the best solution 19, 18, and 14 times, respectively, out of the 27 instances. The average running time of the VNS method was 667 seconds, while LS1, LS2, and GA took 22,816, 25,814, and 11,477 seconds, respectively. This clearly shows the superiority of the VNS method, especially as the instances get larger. It finds the best solution in a greater number of instances (and for almost all instances) an order of magnitude faster than any of the three comparative procedures.

3.6. Conclusion

In this paper, we considered the CCMLST problem and we developed a VNS method for solving this problem. We compared the solutions obtained by the VNS method to optimal solutions for small instances and to solutions obtained by three heuristics LS1, LS2, and GA that were previously proposed for the LCMST problem (but can be easily adapted to the CCMLST problem as well). We generated small and medium-sized instances in a similar fashion to Xiong et al. [82], and generated a set of large instances from the TSPLIB dataset.

The VNS method was clearly the best heuristic for the CCMLST instances. Of the 191 instances, it provided the best solution in 189 instances. For all the 104 instances where the optimal solution was known, the VNS method obtained the optimal solution. Furthermore, for the large instances, its running time is an order of magnitude faster than those of the three other heuristics.

Chapter 4:

The Generalized Covering Salesman Problem

Abstract:

Given a graph $G = (N, E)$, the Covering Salesman Problem (CSP) is to identify the minimum length tour "covering" all the nodes. 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 from a node on the tour. In this chapter 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 it is not allowed to stay overnight (i.e. for revisiting 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 chapter, we develop two local search heuristics to find high-quality solutions to the three GCSP variants. For testing 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 these two problems as well. Overall the results show that our suggested heuristics find high-quality solutions very rapidly.

4.1. 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 [25]). In recent years, many new variants such as the TSP with profits [30], the Clustered TSP [21], the Generalized TSP [31], the Prize Collecting TSP [33], and the Selective TSP [47] have been introduced and studied. The recent monograph by Gutin and Punnen [36] has a nice discussion of different variations of the TSP and solution procedures.

In 1981 Current [23] 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 from 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 TSP (thus it is NP-hard). Current and Schilling [24] referred to several real world examples, such as routing of rural healthcare delivery teams where the assumption of visiting each city is not valid since it is sufficient for all cities to be near to some stops on the tour (the inhabitants of those cities which are not in the tour are expected to go to their nearest stop). Current and Schilling [24] 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 related Set Covering Problem, a zero-one $n \times n$ matrix, i.e. 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., $d_i >= c_{ij}$) 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 cost of columns of matrix *A* are all equal to one. Therefore a uni 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 of the nodes 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 (i.e., have an optimal TSP tour constructed over the nodes selected in the optimal SCP), and the best solution be selected. The algorithm is demonstrated on a sample problem, but no additional computational results are reported.

Arkin and Hassin [2] 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 from the nodes which are visited by the tour, in the geometric version it is sufficient for the tour to intersects the specific neighborhoods without visiting any specific node of the problem. Arkin and Hassin [2] 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.

Other than Current [23], Current and Schilling [24], and Arkin and Hassin [2] the CSP does not seem to have got 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. [35] 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 from 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. [35] proposed a heuristic that combines GENIUS, a high quality heuristic for the TSP [34], with PRIMAL1, a high quality heuristic for the SCP [3].

Vogt et al. [77] 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. [77] 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 [24], 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 variations that can arise in the GCSP (that deal with whether a node can be revisited or not). All these variants are strongly NP-Hard, since they contain the classical TSP as a special case.

The rest of this chapter is organized as follows. In Section 4.2, we formally define the generalized covering salesman problem, and describe three variants. We also describe a mathematical model for the problem. Section 4.3 describes two local search heuristics for the GCSP. Section 4.4 discusses our computational experience on the three different variants of the GCSP, as well as the CSP and the Generlized TSP (GTSP), which are special cases of the GCSP. Section 4.5 provides concluding remarks and discusses some possible extensions of the GCSP.

4.2. 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 *j*). Each node *i* can cover a subset of nodes D_i (note that $i \in D_i$, and when coverage is based on distance D_i can be computed easily from 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 overnight* and *Integer GCSP with overnight*. In the following we explain each of these variants.

4.2.1. 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.

4.2.2. Integer Generalized Covering Salesman Problem without Overnights

Here a node can be visited more than once, but overnight stay is not allowed. Therefore, to have a feasible solution, after visiting a node, the tour can return to this node, if necessary, 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.

4.2.3. Integer Generalized Covering Salesman Problem with Overnights

This version is similar to the previous one, but overnight stay at a node is 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., the number of nodes i 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.

4.2.4. Mathematical Formulation

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:

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

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

The integer programming model can now be stated as:

(**BinaryGCSP**) Min $\sum_{(i,j)\in A} c_{ij}x_{ij} + \sum_{i\in N} F_i w_i$ (4.1)

Subject to:

$$\sum_{j:(j,i)\in A} x_{ji} = \sum_{j:(i.j)\in A} x_{ij} = w_i \qquad \forall i \in N \tag{4.2}$$

$$\sum_{j\in N} a_{ij} w_j \geq k_i \qquad \forall \ i \in N \tag{4.3}$$

$$\sum_{l\in S}\sum_{k\in N\setminus S} x_{lk} + \sum_{k\in N\setminus S}\sum_{l\in S} x_{kl} \geq 2\ (w_i + w_j - 1) \qquad S \subset N,\ 2 \leq |S| \leq n-2\ ,\quad i \in S,\ j \in N \setminus S \tag{4.4}$$

$$x_{ij} \in \{0,1\} \qquad \forall\ (i,j) \in\ A \tag{4.5}$$

$$w_i \in \{0,1\} \qquad \forall\ i \in\ N \tag{4.6}$$

The objective is to minimize the sum of the tour costs and the node visiting costs. Constraint set (4.2) ensures that for each on-tour customer, we have one incoming and one outgoing arc. Constraint set (4.3) specifies that the demand of each node must be covered. Constraint set (4.4) is a connectivity constraint that ensures that there are no subtours. Note that there are an exponential number of connectivity constraints. Constraints (4.5) and (4.6) 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.

y_i : Number of times that node *i* is visited by the tour.

z_{ij} : Number of times arc (*i,j*) is traversed by the tour.

The integer programming model can now be stated as:

(IntegerGCSP) Min $$\sum_{(i,j)\in A} c_{ij} z_{ij} + \sum_{i\in N} F_i y_i \qquad (4.7)$$

Subject to:

$$\sum_{j:(j,i)\in A} z_{ji} = \sum_{j:(i,j)\in A} z_{ij} = y_i \qquad \forall i \in N \qquad (4.8)$$

$$\sum_{j\in N} a_{ij} y_j \geq k_i \qquad \forall\ i \in N \qquad (4.9)$$

$$y_i \leq L w_i \qquad \forall i \in N \qquad (4.10)$$

$$z_{ij} \leq L x_{ij} \qquad \forall (i,j) \in A \qquad (4.11)$$

$$\sum_{l\in S}\sum_{k\in N\setminus S} x_{lk} + \sum_{k\in N\setminus S}\sum_{l\in S} x_{kl} \geq 2\ (w_i + w_j - 1) \qquad S \subset N,\ 2 \leq |S| \leq n-2\ , \quad i \in S,\ j \in N \setminus S \qquad (4.12)$$

$$x_{ij} \in \{0,1\}, z_{ij} \in Z^+ \qquad \forall\ (i,j) \in\ A \qquad (4.13)$$

$$w_i \in \{0,1\}, y_i \in Z^+ \qquad \forall\ i \in\ N \qquad (4.14)$$

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 (4.8) ensures that if node i is visited y_i times, then we have y_i incoming and y_i outgoing arcs. Constraint set (4.9) specifies that the demand of each node must be covered. Constraint sets (4.10) and (4.11) 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 (4.12) 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 (4.13) and (4.14) 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. In this research, we focus on local search algorithms to develop high-quality solutions for the GCSP.

4.3. 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 4.2, 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 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 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>, <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 going away from a node and returning to revisit it.

4.3.1. 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:

$$P_k = C_k / \sum_{s \in S} C_s \qquad (4.15)$$

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). 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

$$I_k / N_k^2 = \min_{j \in P}(I_j / N_j^2) . \qquad (4.16)$$

Here I_k is the amount of increase in the tour cost by insering node *k* into its best position in the tour, while N_k is the number of uncovered nodes (or uncovered demand) which can be covered by node *k*. We update the calculation of N_k for all nodes in *P* and repeat the selection and insertion of nodes procedure 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 the case LS1 finds an improvement, i.e., the cost of S' is less than the cost of *S*, it tries to improve the tour length (and thus the overall cost) by applying the *Lin-Kernighan Procedure* [49] to the solution S'. We apply the Lin-Kernighan code LKH version 1.3 of Helsgaun [38] that is available for download on the web. Since the procedure is computationally expensive, we only apply it after *max_k* (a parameter) improvements over the solution *S*.

In order to get out 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 for 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 diversity to the search procedure, we allow downhill 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 better than (1+α) times the best solution found we keep it as the current solution (over which we try and 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 4.1. The parameters to be tuned for LS1 and their best values obtained in our computational testing are described in Table 4.1 (see Section 4.4).

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 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).

Algorithm 4.1. 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 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.

Feasibility Procedure ( $S'$ ):
   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.

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

Mutation Procedure (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).
```

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 lose their covering demands (meaning that their demand is no longer 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 (i.e., when the minimum insertion cost is negative) 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).

In the *Perturbation Phase*, 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 out of the current tour, while for the two other GCSP variants the nodes can be selected as well from those visited in the current tour) 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* [49], 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 4.2, and the parameters to be tuned for LS2 and their best values obtained in our computational testing are described in Table 4.2 (see Section 4.4).

Algorithm 4.2. 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
     For $j$ = 1, …, $J$ do
        While the solution can be improved do
           Improvement Procedure ($S$);
        End While
        If Cost of the current tour is greater than the $BestCost$ then
           update the solution with the best known solution;
        Perturbation Procedure ($S$);
     End For;
     If by applying the Improvement Procedure the best known solution has been improved then
        Call Lin-Kernighan Procedure($S$);
  End For;
End.

Improvement Procedure($S$):
Begin
   $r$ := 1;
   While $\left(r \le N\left(S\right)\right)$ do
      Extract the $r^{\text{th}}$ node of the tour;
      If the solution is feasible then
         Update the solution;
      Else
         Generate all sequences with cardinality one or two, by considering the $T$ nodes nearest
         to the extracted node;
         $Extra\_Cost$ := Extra cost (the cost generated by substituting sequence $s$ with the
                   extracted node) related to the sequence that has the minimum insertion cost;
         If $Extra\_Cost < 0$ then
            Update $BestCost$ and the current solution by substituting the new sequence with the
            extracted node;
         End If;
      End If;
      $r$ := $r$+1;
   End While;
End.

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

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 [24] 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 compare 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.

4.4.1. Test Problems

Since there are no test problems in the literature for the CSP (as well as the variants of the GCSP we introduce), we created datasets based on the TSP library instances [65]. In particular we constructed our datasets based on 16 Euclidean TSPLIB instances whose size ranged from 51 to 200 nodes.

In the datasets created, each node can cover its 7, 9 or 11 nearest nodes (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 datasets 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 for visiting a node can be different from node to node, for simplicity we consider the node visiting costs to be the same for all nodes in an instance. In fact, if we assign a high node visiting cost, the problem becomes a Set Covering Problem (as the node visiting costs dominate the routing cost) 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 overnight 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 to either extreme (Set Covering Problem or CSP), we fixed the node visiting cost value to 50 for all the instances (which turned out to be an appropriate amount for the different kinds of instances studied in this paper). In this fashion we constructed 48 datasets for our computational work.

After considerable experimentation on a set of small test instances, we determined the best values of the parameters to be used in both LS1 and LS2. Tables 4.1 and 4.2 show the different values that were tested for various parameters and the best value obtained for the parameters in LS1 and LS2. 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.

Table 4.1. Parameters for LS1

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

Table 4.2. Parameters for LS2

Parameters	Different values tested	Best value
J	{50, 100, 150, 200, 250, 300}	200
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)

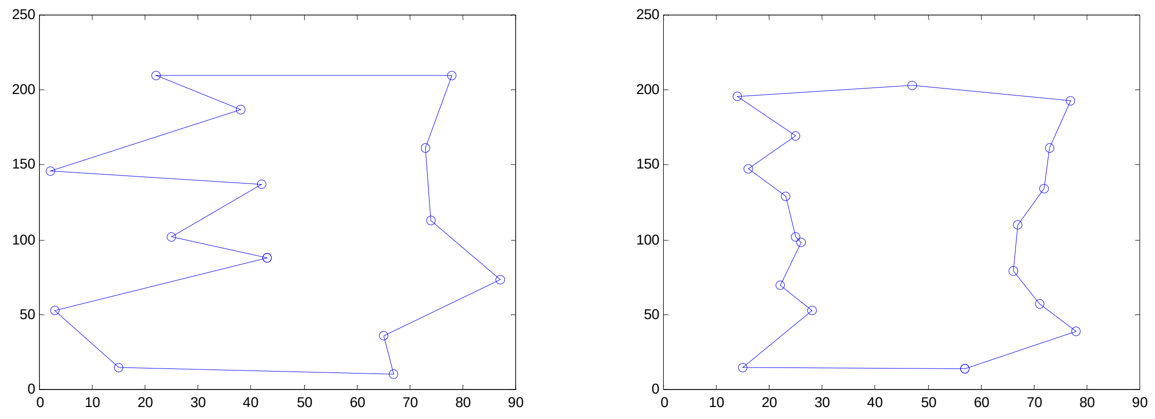
In all tables reporting the computational performance of the heuristics, the first column is related to the instance name which includes the number of nodes. The second column (NC) gives the number of nearest nodes that can be covered by each node. Moreover, for each method the best and the average cost, 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 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.

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

Since Current and Schilling [24] 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 4.1. 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 [40] 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* [49] 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 can be found in less than 10 minutes of running time.

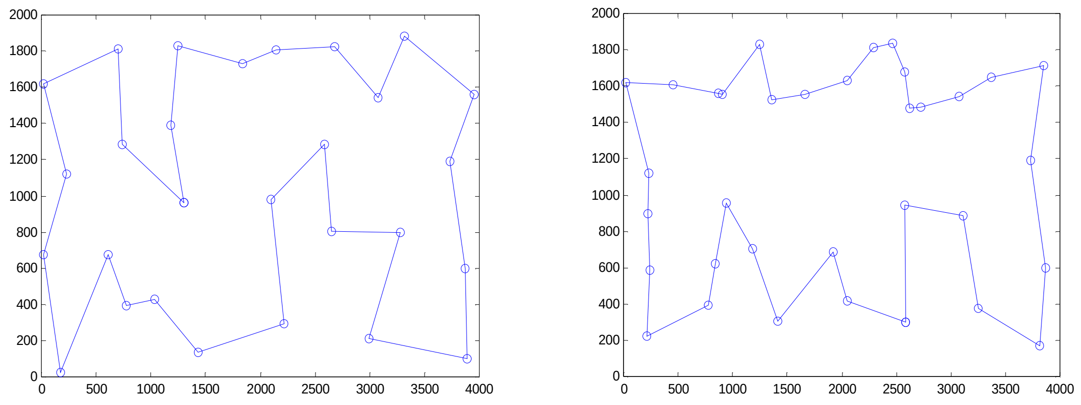
Table 4.3 reports the results obtained by LS1, LS2 and our implementation of Current and Schilling's method. In this table, the number of optimal solutions (NO) of the set covering problem is given. In Table 4.3 instances for which all the optimal solutions to the set covering problem cannot be obtained within the given time threshold are shown with an asterisk. As can be seen in Table 4.3 for the CSP, both LS1 and LS2 can obtain, in a few seconds, better solutions than Current and Schilling's method. The results of both the heuristics in all except one case (where they are tied with Current and Schilling's method) are better than 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 all 48 instances, while LS1 only obtains the best solution in 38 out of the 48 instances.

Figure 4.1. 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 4.2. 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

From Table 4.3 we can make the following counter-intuitive observation. Sometimes by selecting a set of nodes with a larger cardinality, 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. Figures 4.1 and 4.2 illustrate two examples of CSP (Rat99 and KroA200) in which, by increasing the number of nodes in the tour, the tour length is decreased.

4.4.3. Comparison of LS1 and LS2 on GCSP Variants

In Table 4.4 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. Although on average LS2 is a bit faster than LS1, in terms of the average cost, average time to best solution, and the number of best solutions found LS1 is better than LS2. Over the 48 instances, the two heuristics were tied in 22 instances. While, in 14 instances LS1 is strictly better than LS2, and in 12 instances LS2 is strictly better than LS1.

Table 4.5 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 12 instances, LS2 is strictly better than LS1 in 11 instances, while they are tied in 26 instances. Again the running time of both LS1 and LS2 is extremely small, taking no more than 20 seconds even for the largest instances.

Table 4.6 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). Here, over 48 test instances, LS1 is strictly better than LS2 in 8 instances, LS2 is strictly better than LS1 in 30 instances, and they are tied in 10 instances. However, the running time of LS1 increases significantly 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.

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 4.4 to 4.6. This is confirmed in our experiments.

4.4.4. 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 decrease compared to the instances of the GCSP without Steiner nodes. Table 4.7 confirms this observation. Here we compare LS1 and LS2 on the CSP with Steiner nodes. For each CSP instance (in Table 4.3) 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 LS1 was strictly better once, LS2 was strictly better 6 times, and the two methods were tied 41 times. Overall LS2 runs slightly faster than LS1. For brevity, we have limited the comparison to the CSP with Steiner nodes.

4.4.5. Analyzing the Quality of LS2 on the Generalized TSP

Overall, LS2 seems to be a better choice than LS1, in that it is more robust than LS1. It outperforms LS1 on the CSP and the integer GCSP with overnights, while it is tied with LS1 for the binary GCSP and integer GCSP without overnights. Further, the run time of LS2 remains fairly stable. However, 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 [31] 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 [31] 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 [31]) by first tuning its parameters. The tuned parameters of LS2 are configured as follows: *J*=300*, K*=10, *T*=10, *max_iter* = 50 and 10 independent runs of LS2 were performed. We compared LS2 to eight other heuristics in the literature that are described below.

1. MSA: A Multi-Start Heuristic by Cacchiani et al. [12],
2. mrOX: a Genetic Algorithm by Silberholz and Golden [71],
3. RACS: a Reinforcing Ant Colony System by Pintea et al. [62],
4. GA: a Genetic Algorithm by Snyder and Daskin [72],
5. GI^3 : a composite algorithm by Renaud and Boctor [66],
6. NN: a Nearest Neighbor approach by Noon [60],
7. *FST-Lagr* and *FST-root*: Two heuristics by Fischetti et al. [31].

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 [28]. The computer factors are shown in Table 4.8. 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. [12] 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 4.9 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 4.8) for all the methods but 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 4.9 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 4.9 indicates, although LS2 was not explicitly developed for the GTSP (but rather for a generalization of it), it performs quite creditably. On average it takes 2.2 seconds, finds solutions that are on average 0.08% from optimality, and found optimal solutions in 30 out of 41 benchmark GTSP instances.

4.5. Summary and Conclusions

In this chapter 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. Overall LS2 appears to be more robust in terms of its running time as well as its performance in terms of the number of times it found the best solutions in 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 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 be considered in future research. The earlier generalizations of the CSP (see [77]) included requirements 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.

Table 4.3. 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	0,20	1,48	**164**	164.0	10	0.04	0.77
	9	309	169	6	1.92	1.97	**159**	159.0	8	0.10	1.34	**159**	159.0	9	0.03	0.61
	11	282	167	5	0.59	1.70	**147**	147.0	7	0.04	1.22	**147**	147.0	8	0.03	0.55
Berlin 52	7	2769	4019	8	19.39	21.04	**3887**	3966.2	11	0.08	1.68	**3887**	3887.0	11	0.26	0.67
	9	11478	**3430**	7	26.08	94.14	**3430**	3435.8	7	0.10	1.41	**3430**	3430.0	7	0.04	0.62
	11	11	3742	5	0.22	0.26	**3262**	3262.0	6	0.02	1.60	**3262**	3262.0	6	0.02	0.34
St 70	7	32832	297	10	232.24	454.07	**288**	288.0	11	0.11	1.86	**288**	288.0	12	0.05	1.03
	9	18587	271	9	173.87	176.00	**259**	259.0	10	0.05	1.79	**259**	259.0	10	0.05	1.22
	11	1736	269	7	13.21	13.74	**247**	247.0	10	0.16	1.98	**247**	247.0	10	0.04	0.88
Eil 76	7	241	241	11	1.15	2.46	**207**	210.6	15	0.53	2.09	**207**	207.0	15	0.17	1.11
	9	1439	193	9	7.43	13.95	186	186.8	11	0.26	1.98	**185**	185.0	11	0.05	1.13
	11	7050	180	8	30.48	78.88	**170**	176.4	11	0.05	2.14	**170**	170.0	11	0.05	1.07
Pr 76	7	26710	53255	11	54.20	170.41	**50275**	51085.0	14	0.55	1.86	**50275**	50275.0	14	0.78	1.27
	9	326703*	45792	10	6743.66	9837.36	**45348**	45348.0	12	0.27	2.01	**45348**	45348.0	12	0.26	1.12
	11	20	45955	7	0.11	0.20	**43028**	43418.4	10	0.48	1.95	**43028**	43028.0	10	0.07	1.03
Rat 99	7	3968	572	14	22.74	32.75	**486**	486.4	18	0.08	2.20	**486**	486.0	18	0.16	1.77
	9	170366	462	12	1749.66	2729.67	**455**	455.6	15	0.67	2.38	**455**	455.0	15	0.11	1.92
	11	16301	456	10	88.87	140.18	**444**	444.8	12	0.43	2.25	**444**	444.0	12	0.09	1.75
KroA 100	7	208101*	10306	15	6303.03	6475.95	**9674**	9674.0	19	0.38	2.06	**9674**	9674.0	19	0.31	2.04
	9	95770	9573	12	524.49	1365.42	**9159**	9159.0	15	0.13	2.28	**9159**	9159.0	15	0.14	1.85
	11	33444	9460	10	409.47	433.97	**8901**	8912.2	13	0.19	2.56	**8901**	8901.0	13	0.13	1.62
KroB 100	7	4068	11123	14	45.62	48.35	**9537**	9537.0	20	0.39	1.99	**9537**	9537.0	20	0.33	1.93
	9	133396	9505	12	2112.57	2623.76	**9240**	9262.2	15	0.54	2.13	**9240**	9240.0	15	0.16	1.99
	11	90000*	9049	10	1056.27	2895.35	**8842**	8842.6	13	1.34	2.62	**8842**	8842.0	13	0.09	1.83
KroC 100	7	129545*	10367	15	3391.82	4212.98	9728	9728.6	18	0.72	2.46	**9723**	9723.0	17	0.17	1.97
	9	5028	9952	12	35.91	52.25	**9171**	9184.4	13	0.12	2.45	**9171**	9171.0	13	0.19	1.91
	11	75987*	9150	10	1389.84	2482.00	**8632**	8632.0	13	0.14	2.74	**8632**	8632.0	13	0.09	1.85
KroD 100	7	1392	11085	14	10.29	15.58	**9626**	9626.0	20	1.35	2.39	**9626**	9626.0	20	0.21	1.83
	9	700	10564	11	6.18	7.74	**8885**	8903.8	13	0.75	2.38	**8885**	8885.0	13	0.12	2.04
	11	85147*	9175	10	968.39	2761.51	**8725**	8730.4	13	0.48	2.83	**8725**	8725.0	13	0.13	1.89
KroE 100	7	92414*	11323	15	1971.32	3075.58	**10150**	10154.8	19	0.14	2.48	**10150**	10150.0	19	1.06	1.84
	9	85305*	9095	12	1918.72	2764.70	8992	8992.0	13	0.33	2.69	**8991**	8991.0	14	0.16	1.90
	11	70807*	8936	10	609.81	2335.43	**8450**	8450.0	13	0.36	2.89	**8450**	8450.0	13	0.08	1.97

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
Rd 100	7	2520	4105	14	24.43	4196.23	**3461**	3478.2	18	0.31	2.53	**3461**	3485.6	18	0.24	1.83
	9	95242*	3414	12	1798.14	3118.93	**3194**	3211.4	16	0.91	2.65	**3194**	3194.0	16	0.25	1.76
	11	1291	3453	10	8.60	22.11	2944	2944.0	12	0.44	3.1	**2922**	2922.0	13	0.14	1.54
KroA150	7	97785*	12367	22	2252.50	3499.43	11480	11548.8	27	0.89	2.68	**11423**	11481.0	28	1.97	2.91
	9	69377*	11955	17	2454.99	2477.69	10072	10072.0	23	0.71	2.78	**10056**	10056.0	26	1.91	2.75
	11	169846*	10564	15	5483.07	5518.26	**9439**	9439.0	21	1.06	2.82	**9439**	9439.0	21	0.39	2.68
KroB 150	7	14400	12876	21	196.85	270.94	11490	11517.0	30	1.39	2.58	**11457**	11463.6	30	1.66	3.08
	9	137763*	11774	18	2760.03	4572.81	**10121**	10173.4	24	1.19	2.77	**10121**	10121.0	24	0.64	2.78
	11	1431	10968	14	26.64	46.96	**9611**	9639.8	21	0.61	2.88	**9611**	9611.0	21	0.28	2.88
KroA 200	7	53686*	14667	28	537.60	1170.37	13293	13345.2	34	1.05	3.25	**13285**	13313.8	34	3.99	4.28
	9	64763*	12683	23	1504.07	1628.36	11710	11753.6	29	1.23	2.80	**11708**	11725.0	28	3.25	3.88
	11	29668*	12736	19	398.25	671.55	**10748**	10813.4	29	1.04	3.16	**10748**	10814.8	29	3.10	3.65
KroB 200	7	107208*	14952	29	365.08	3351.89	13280	13297.6	36	0.46	2.83	**13051**	13147.4	35	2.24	4.38
	9	38218*	13679	23	637.66	805.04	**11864**	11898.6	29	1.02	2.98	**11864**	11937.8	29	2.30	4.02
	11	67896*	12265	20	493.64	1410.60	**10644**	10714.0	29	0.98	2.81	**10644**	10650.4	29	3.10	3.72
Avg		55896	9808.02	13	1017.94	1626.68	9031.4	9070.3	17	0.52	2.35	9023.6	9031.5	17	0.65	1.95
No.Best			1				38					**48**				

Table 4.4. 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	1224	1224	20	0.10	1.65	**1190**	1191.6	19	0.24	1.08
	9	**991**	996.2	15	0.48	1.63	**991**	993.4	15	0.21	1.10
	11	**844**	869.4	13	0.27	1.49	**844**	849.4	13	0.19	1.14
Berlin 52	7	**5429**	5429.0	17	0.06	1.65	**5429**	5514.6	17	0.11	1.11
	9	**4807**	4818.8	14	0.09	1.49	**4807**	4834.0	14	0.08	1.08
	11	**4590**	4655.0	13	0.27	1.58	**4590**	4639.6	13	0.30	0.85
St 70	7	1836	1841.6	29	0.56	2.58	**1834**	1836.4	29	0.45	1.33
	9	1461	1468.8	22	0.35	1.76	**1460**	1460.0	22	0.42	1.38
	11	**1268**	1270.2	19	0.90	1.73	**1268**	1270.2	19	0.45	1.37
Eil 76	7	**1610**	1630.8	26	0.52	2.51	**1610**	1623.0	26	0.25	1.63
	9	**1270**	1319.8	20	0.37	1.83	1296	1301.2	21	0.83	1.64
	11	**1117**	1130.8	18	0.31	1.87	**1117**	1122.2	18	1.04	1.61
Pr 76	7	66789	66850.8	28	0.66	1.46	**66455**	66887.8	29	0.83	1.71
	9	**62907**	62916.0	23	0.18	1.71	63114	63203.6	25	0.83	1.68
	11	**52175**	52527.0	19	0.25	1.58	**52175**	52175.0	19	0.32	1.47
Rat 99	7	2341	2346.0	34	0.64	3.07	**2325**	2340.2	33	0.98	2.17
	9	**1936**	1940.4	27	0.24	1.97	**1936**	1941.2	27	1.01	2.43
	11	**1686**	1714.2	23	0.31	1.80	**1686**	1691.2	23	1.21	2.39
KroA 100	7	**14660**	14660	41	0.53	2.18	**14660**	14726.6	41	1.26	2.25
	9	**12974**	12974	33	0.13	1.65	**12974**	12987.2	33	0.47	2.38
	11	11970	11977.2	28	0.42	1.57	**11942**	11942.0	29	0.41	2.38
KroB 100	7	**14415**	14451.8	44	0.87	1.91	14459	14577.6	42	0.43	2.23
	9	12222	12296.4	34	0.86	2.17	**12194**	12247.0	33	2.18	2.27
	11	**11276**	11277.2	28	1.20	2.55	**11276**	11315.2	28	0.83	2.43
KroC 100	7	**13830**	13888.8	41	0.13	2.88	**13830**	13850.2	41	2.08	2.24
	9	**12149**	12190.2	33	0.64	2.12	**12149**	12189.6	33	1.45	2.21
	11	**11032**	11032	26	0.11	2.00	**11032**	11032.0	26	1.74	2.22
KroD 100	7	**13567**	13666.4	38	0.06	2.53	13704	13857.2	38	0.31	2.42
	9	**12409**	12448.6	32	1.03	1.92	12419	12479.8	31	2.08	2.48
	11	11486	11520.8	28	0.43	1.76	**11443**	11515.6	29	1.34	2.11
KroE 100	7	**15321**	15485.0	41	0.37	2.62	15471	15700.6	41	0.30	1.99
	9	**12482**	12482	32	0.19	1.64	**12482**	12482.0	32	0.40	2.33
	11	**11425**	11452.4	30	0.66	1.48	11456	11490.6	28	2.24	2.26
Rd 100	7	6209	6210.8	37	0.30	2.20	**6170**	6251.4	37	0.70	2.33
	9	**5469**	5595.0	29	0.23	2.10	**5469**	5477.2	29	1.06	2.44
	11	**4910**	4985.6	28	0.52	1.63	**4910**	4965.2	28	1.17	2.22
KroA150	7	**17258**	17274.6	55	0.96	4.52	17270	17425.8	54	2.11	3.63
	9	**15007**	15042.6	46	1.07	3.68	**15007**	15145.4	46	2.60	4.20
	11	**13666**	13755.6	40	1.20	2.93	13762	14010.8	41	2.50	3.75
KroB 150	7	**17639**	17745.8	60	2.94	4.16	**17639**	18141.4	60	2.18	3.56
	9	**15505**	15688.0	50	0.90	3.61	15506	15854.8	50	3.71	3.76
	11	13740	13899.0	42	1.82	2.97	**13719**	13836.4	40	2.82	3.69
KroA 200	7	21388	21553.8	74	3.23	8.11	**21346**	21543.8	76	3.71	4.73
	9	**17843**	17999.4	59	2.33	5.70	17893	18103.8	60	3.09	4.91
	11	16591	16702.4	54	1.84	4.94	**16380**	16580.4	55	3.80	4.85
KroB 200	7	**20736**	20960.0	79	2.30	8.71	20882	21117.6	79	3.88	4.52
	9	**18266**	18377.4	66	1.94	6.28	18269	18500.6	67	3.05	4.73
	11	**15961**	16428.8	55	2.37	4.71	16173	16372.0	55	3.57	4.65
Avg		13035.2	13103.6	35	0.79	2.72	13041.9	13137.4	35	1.40	2.49
No.Best		**36**					34				

Table 4.5. Comparison of LS1 and LS2 on Integer GCSP without overnight

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**	1199.8	19	1	0.28	4.19	**1185**	1187.0	19	1	0.96	3.83
	9	**991**	992.8	15	0	1.59	4.09	**991**	996.2	15	0	0.85	3.30
	11	**843**	845.0	13	1	1.27	4.77	**843**	843.0	13	1	1.12	3.22
Berlin 52	7	**5429**	5429.0	17	0	0.14	3.99	**5429**	5429.0	17	0	0.18	3.37
	9	**4785**	4796.8	15	1	0.02	4.17	**4785**	4807.6	15	1	0.12	2.69
	11	**4590**	4651.4	13	0	0.12	3.27	**4590**	4620.0	13	0	0.66	2.51
St 70	7	**1778**	1783.4	28	3	1.21	6.56	1782	1786.0	28	1	0.48	5.10
	9	1461	1497.8	22	0	0.23	5.27	**1460**	1461.2	22	0	1.46	4.56
	11	1268	1268.0	19	0	1.33	3.31	**1241**	1264.2	18	1	1.70	4.03
Eil 76	7	**1600**	1626.6	26	1	1.69	6.67	**1600**	1619.4	26	2	2.27	5.11
	9	**1270**	1291.6	20	0	1.26	5.10	**1270**	1294.0	20	0	1.93	4.79
	11	**1117**	1121.2	18	0	0.89	4.13	**1117**	1117.0	18	0	0.48	4.63
Pr 76	7	65990	66615.8	28	1	1.47	4.73	**64111**	65560.8	29	4	1.70	5.44
	9	57147	57945.2	29	1	1.14	5.17	**54907**	55862.4	29	6	1.37	4.94
	11	51587	51650.0	20	2	1.52	4.42	**49445**	49445.0	21	3	1.02	4.22
Rat 99	7	**2311**	2315.0	33	1	0.65	7.39	**2311**	2341.2	33	1	2.70	7.61
	9	**1936**	1937.8	27	0	1.53	5.85	**1936**	1949.4	28	0	1.99	7.19
	11	**1683**	1704.4	23	0	2.07	4.55	**1683**	1701.0	23	0	1.87	6.77
KroA 100	7	**14660**	14678.8	41	0	1.52	6.35	**14660**	14784.4	41	0	2.40	8.18
	9	**12974**	12974.0	33	0	0.53	4.98	**12974**	13090.0	33	0	2.27	7.49
	11	**11737**	11737.0	29	1	0.47	4.59	**11737**	11737.0	29	1	2.03	0.30
KroB 100	7	**14246**	14394.2	45	6	3.00	6.16	14297	14316.8	43	3	3.54	8.83
	9	12200	12348.6	34	3	2.39	4.91	**12189**	12197.8	33	2	1.56	7.35
	11	**11268**	11394.2	27	2	0.30	6.38	**11268**	11378.0	27	2	0.74	6.95
KroC 100	7	**13520**	13644.0	42	5	3.42	7.49	13792	13999.8	41	1	2.55	8.42
	9	**12119**	12209.0	33	1	1.22	6.76	**12119**	12119.0	33	1	1.18	7.22
	11	**11032**	11032.0	26	0	0.57	9.82	**11032**	11074.4	26	0	0.85	6.19
KroD 100	7	**13501**	13517.6	39	2	4.97	7.61	**13501**	13635.0	39	2	1.97	8.84
	9	12261	12303.2	31	1	0.62	6.20	**12257**	12279.6	31	1	1.98	7.71
	11	11452	11534.0	29	1	2.79	9.43	**11409**	11450.2	30	1	1.65	7.30
KroE 100	7	**15308**	15386.8	42	1	2.71	7.05	15471	15767.2	41	0	4.37	8.02
	9	**12482**	12541.8	32	0	0.42	5.72	**12482**	12485.0	32	0	1.29	7.18
	11	**11344**	11417.8	30	1	2.89	7.40	**11344**	11373.6	30	1	2.92	6.68
Rd 100	7	**6078**	6182.8	37	2	0.86	6.78	**6078**	6199.6	37	2	2.05	7.45
	9	**5384**	5501.0	30	2	3.54	6.39	**5384**	5418.8	30	2	2.60	7.15
	11	**4853**	4916.6	29	1	1.73	4.14	**4853**	4867.2	29	1	1.58	6.29
KroA150	7	**16947**	16974.0	57	4	3.29	13.36	16976	17143.0	56	3	7.22	12.50
	9	15007	15158.6	46	0	2.55	9.47	**15000**	15136.8	49	3	2.40	11.80
	11	**13580**	13709.4	40	2	1.92	7.47	13683	13791.8	41	2	4.76	10.60
KroB 150	7	**17621**	17776.8	59	1	5.55	11.46	17639	18136.2	60	0	3.15	12.15
	9	**15332**	15609.8	48	3	3.90	10.68	15383	15556.0	48	3	4.83	11.78
	11	**13554**	13582.0	41	2	2.76	7.66	**13554**	13670.8	41	2	4.96	10.82
KroA 200	7	21337	21415.2	79	3	10.35	20.03	**21120**	21294.8	78	6	10.09	16.92
	9	**17812**	17927.0	62	2	6.77	14.37	17832	18186.8	64	5	10.91	14.97
	11	**16290**	16517.8	54	4	9.14	12.56	16370	16485.0	53	4	10.41	14.48
KroB 200	7	**20628**	20808.2	78	2	5.91	20.45	20862	21027.4	77	2	6.62	16.01
	9	**18247**	18387.6	67	3	4.75	14.88	18260	18448.6	68	3	8.84	14.79
	11	15888	16150.0	56	3	4.03	11.73	**15688**	15968.8	56	4	8.01	13.80
Avg		12825.7	12925.0	35	1	2.36	7.50	12706.3	12839.7	35	2	2.97	7.74
No.Best		**37**						36					

Table 4.6. Comparison of LS1 and LS2 on Integer GCSP with overnight

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**	1146.0	19	10	0.60	10.21	**1146**	1146.0	19	10	0.09	2.81
	9	**958**	980.8	15	5	0.77	8.26	**958**	968.4	15	5	0.70	3.15
	11	842	866.8	13	5	1.66	7.85	**827**	829.4	13	4	0.33	2.90
Berlin 52	7	4969	4981.6	19	11	5.65	9.22	**4966**	4976.0	18	9	0.66	2.87
	9	**4272**	4301.2	16	8	3.00	6.74	**4272**	4324.0	16	8	0.58	2.35
	11	**3962**	4149.2	14	8	0.10	8.09	**3962**	3962.0	14	8	0.12	2.17
St 70	7	**1654**	1656.2	27	15	2.95	17.46	1655	1655.0	27	14	0.35	4.26
	9	1442	1453.0	23	12	1.07	11.21	**1416**	1438.6	22	9	0.40	3.79
	11	**1196**	1226.8	18	6	1.71	8.30	**1196**	1213.0	18	6	1.77	3.75
Eil 76	7	**1554**	1587.0	26	10	6.00	12.37	1562	1578.4	26	10	1.06	4.63
	9	**1268**	1307.2	21	7	4.23	9.54	**1268**	1298.2	21	7	1.00	4.22
	11	**1107**	1125.6	18	3	4.99	8.01	**1107**	1110.2	18	4	0.68	4.08
Pr 76	7	53270	56065.2	29	15	1.41	16.58	**53266**	54142.0	30	16	1.97	4.61
	9	47226	49028.4	26	15	7.40	12.84	**46912**	47245.8	27	17	0.72	3.99
	11	44036	46104.0	19	8	0.73	11.47	**44028**	44029.6	20	10	0.99	3.62
Rat 99	7	**2229**	2241.8	33	10	3.43	10.24	**2229**	2259.4	33	10	3.09	6.34
	9	**1908**	1940.6	27	5	5.58	10.05	1922	1947.0	28	8	3.17	6.19
	11	1673	1697.2	24	9	3.89	10.84	**1650**	1686.6	23	6	0.73	5.71
KroA 100	7	12474	12762.4	42	24	10.20	28.73	**12006**	12322.6	43	26	1.34	6.69
	9	11671	11733.4	34	21	16.59	22.31	**11218**	11245.2	35	21	1.00	6.28
	11	10886	10931.8	29	17	6.79	19.91	**10665**	10700.8	31	17	2.20	5.64
KroB 100	7	12728	12920.6	39	18	7.31	21.13	**12273**	12530.0	43	25	3.31	7.21
	9	11176	11232.0	34	19	3.11	17.40	**11128**	11133.2	35	21	3.65	6.59
	11	**10302**	10534.6	28	15	5.28	16.54	**10302**	10409.8	28	15	1.69	6.04
KroC 100	7	12202	12401.6	41	22	11.17	32.46	**12043**	12269.2	45	27	2.52	6.63
	9	11196	11374.8	33	16	3.63	17.75	**11031**	11141.0	35	20	1.95	5.71
	11	10445	10629.2	27	15	2.00	18.27	**10299**	10406.2	28	15	1.26	5.42
KroD 100	7	11868	12115.4	38	20	5.13	22.28	**11725**	11827.8	39	20	1.70	6.56
	9	11062	11287.0	31	16	6.55	15.75	**10742**	10869.6	35	20	2.56	6.11
	11	10523	10714.0	27	13	4.35	15.30	**10404**	10469.4	29	16	1.10	5.83
KroE 100	7	13101	13332.4	42	25	6.35	24.37	**12689**	12859.6	45	28	1.83	6.01
	9	**10821**	11193.2	34	20	6.03	24.89	**10821**	10905.0	34	20	1.52	5.61
	11	**10007**	10190.6	29	17	4.59	16.30	**10007**	10136.4	29	17	0.59	5.34
Rd 100	7	5626	5834.2	37	17	4.83	20.75	**5570**	5645.6	39	20	2.72	6.55
	9	**4950**	5129.4	32	18	4.24	13.02	5037	5093.2	30	13	0.81	5.89
	11	4541	4705.8	27	13	4.72	15.30	**4514**	4581.8	27	12	2.01	5.23
KroA150	7	**15341**	15483.2	59	32	19.02	41.95	15385	15644.6	60	37	3.34	9.56
	9	13475	13714.2	49	28	6.60	30.64	**12944**	13288.6	51	28	6.32	10.07
	11	**12151**	12399.4	43	25	4.31	21.24	12215	12407.4	44	25	3.31	8.92
KroB 150	7	15825	15964.0	58	31	8.91	32.67	**15252**	15774.2	61	32	4.47	10.49
	9	13198	13415.8	52	30	21.04	36.04	**13139**	13372.0	52	31	8.28	9.78
	11	12418	12933.0	40	24	6.91	28.11	**12174**	12561.6	44	26	3.46	9.06
KroA 200	7	18093	18186.4	76	39	27.64	47.57	**17873**	18431.8	82	49	6.32	13.74
	9	**15562**	15979.4	63	37	10.07	37.55	15782	16141.6	64	35	3.48	12.36
	11	14873	14933.0	55	30	20.95	30.29	**14629**	14835.0	56	32	4.41	11.39
KroB 200	7	18119	18436.6	82	46	36.42	56.66	**17701**	18108.4	85	51	8.09	13.98
	9	16289	16395.4	68	36	28.23	40.13	**15766**	16264.4	66	37	8.69	12.27
	11	**14217**	14705.8	54	26	13.26	29.64	14360	14490.8	56	27	7.99	11.37
Avg		11246.9	11529.7	35	18	7.74	20.50	11125.8	11284.9	36	19	2.51	6.54
No.Best		18						**40**					

Table 4.7. 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	0.02	1.29	**163**	163.0	9	0.03	0.69
	159	159.40	8	0.38	4.90	**159**	159.0	9	0.03	0.52
	147	147.0	7	0.10	2.08	**147**	147.0	8	0.03	0.57
Berlin 52	**3470**	3483.60	10	0.09	1.45	**3470**	3470.0	10	0.33	0.68
	3097	3097.0	7	0.98	4.45	**3097**	3097.0	7	0.03	0.51
	2956	2959.60	6	0.05	2.46	**2956**	2956.0	6	0.02	0.54
St 70	288	288.0	11	0.11	1.64	**287**	287.0	12	0.08	0.89
	259	259.0	9	0.04	4.53	**259**	259.0	10	0.06	1.07
	245	245.80	10	0.46	2.71	**245**	245.0	10	0.04	0.82
Eil 76	**207**	211.40	15	0.11	1.64	**207**	210.0	15	0.25	0.91
	186	186.40	11	0.76	4.39	**185**	185.0	10	0.04	0.99
	169	170.80	10	0.19	2.63	**169**	169.0	11	0.04	0.92
Pr 76	**49773**	50566.80	13	0.18	1.61	**49773**	49773.0	13	0.18	1.15
	44889	44889.0	12	0.47	4.65	**44889**	44889.0	12	0.12	1.08
	42950	43399.20	9	0.34	2.84	**42950**	42950.0	9	0.05	0.87
Rat 99	483	483.0	17	0.19	1.91	**482**	482.0	18	0.29	1.54
	454	454.0	14	0.56	4.38	**454**	454.0	14	0.09	1.67
	444	444.40	12	0.76	3.09	**444**	444.0	12	0.08	1.47
KroA 100	**9545**	9545.0	18	0.31	2.04	**9545**	9545.0	18	0.39	1.75
	9112	9112.0	15	0.09	1.72	**9112**	9112.0	15	0.35	1.55
	8833	8841.40	13	0.23	3.42	**8833**	8833.0	13	0.08	1.32
KroB 100	**9536**	9536.0	19	0.33	1.93	**9536**	9536.0	19	0.37	1.62
	9199	9205.80	15	0.66	1.53	**9199**	9199.0	15	0.09	1.64
	8763	8763.0	11	0.50	3.55	**8763**	8763.0	11	0.12	1.61
KroC 100	9591	9591.0	15	0.17	1.97	**9590**	9590.0	16	0.14	1.72
	9171	9171.0	13	0.70	1.79	**9171**	9171.0	13	0.27	1.52
	8632	8632.0	13	0.39	3.55	**8632**	8632.0	13	0.09	1.63
KroD 100	**9526**	9526.0	19	0.21	1.83	**9526**	9526.0	19	0.15	1.51
	8885	8885.40	13	0.78	1.87	**8885**	8885.0	13	0.15	1.83
	8725	8731.40	13	0.77	3.69	**8725**	8725.0	13	0.10	1.65
KroE 100	**9800**	9800.0	16	1.06	1.84	**9800**	9800.0	16	0.16	1.54
	8987	8987.0	13	0.29	2.02	**8986**	8986.0	14	0.11	1.56
	8450	8450.0	13	0.40	3.85	**8450**	8450.0	13	0.11	1.70
Rd 100	**3412**	3412.0	18	0.24	1.83	**3412**	3434.4	18	0.25	1.60
	3194	3206.80	16	0.43	1.87	**3194**	3194.0	16	0.31	1.52
	2761	2761.0	12	0.49	3.66	**2761**	2761.0	12	0.09	1.33
KroA150	**10939**	10939.0	27	1.97	2.91	**10939**	11099.6	27	0.85	2.45
	9808	9823.20	23	0.25	2.25	**9808**	9808.0	23	0.20	2.26
	9360	9382.60	20	1.13	3.46	**9360**	9360.0	20	0.29	2.30
KroB 150	**11225**	11288.6	30	1.66	3.08	**11225**	11240.4	30	1.08	2.48
	10121	10211.40	24	1.10	2.35	**10121**	10121.0	24	0.64	2.31
	9542	9556.60	20	0.86	3.54	**9542**	9542.0	20	0.19	2.55
KroA 200	**13042**	13042.0	32	3.99	4.28	13227	13268.0	35	1.12	3.62
	11392	11429.20	27	0.22	2.42	**11392**	11424.0	27	0.83	3.18
	10527	10615.80	24	0.51	4.02	**10525**	10673.8	26	0.79	2.95
KroB 200	**13020**	13160.20	34	2.24	4.38	**13020**	13092.0	34	1.77	3.41
	11712	11788.60	28	0.79	2.63	**11712**	11837.2	28	1.89	3.39
	10614	10769.40	28	1.62	3.60	**10614**	10734.8	28	0.89	2.99
Avg	8911.73	8953.56	16	0.63	2.82	8915.4	8930.9	16	0.33	1.65
No.Best	42					**47**				

Table 4.8. 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	-	1	mrOX
Pentium(R) IV, 3.4 Ghz	295	2.03	MSA
Our	145	1	LS2

Table 4.9. 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

Table 4.9. 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	gap
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

References

[1] Aickelin U. An indirect genetic algorithm for set covering problems. *Journal of the Operational Research Society* 53, 1118-1126 (2002).

[2] Arkin E.M. and Hassin R. Approximation algorithms for the geometric covering salesman Problem, *Discrete Applied Mathematics* 55(3), 197-218 (1994).

[3] Balas E. and Ho A. Set covering algorithms using cutting planes, Heuristics, and Subgradient Optimization: A Computational Study. *Math. Programming* 12, 37-60 (1980).

[4] Baldacci R., Dell'Amico M. Heuristic algorithms for the design of urban optical networks. *Technical Report*, 63, *Department of Sciences and Methods for Engineering, University of Modena and Reggio Emilia, Reggio Emilia, Italy*, (2004).

[5] Baldacci R., Dell'Amico M., and Salazar González J.J. The Capacitated *m*-Ring-Star Problem. *Operations Research* 55(6),1147-1162 (2007).

[6] Bautista J, Pereira J. A GRASP algorithm to solve the unicost set covering problem. *Computers & Operations Research* 34, 3162-3173 (2007).

[7] Beasley J.E. An algorithm for set covering problems. *European Journal of Operational Research* 31, 85-93 (1987).

[8] Beasley J.E. OR-Library: distributing test problems by electronic mail. *Journal of the Operational Research Society* 41, 1069-1072 (1990).

[9] Beasley J.E, Chu P.C. A genetic algorithm for the set covering problem. *European Journal of Operational Research* 94, 392-404 (1996).

[10] Birbil S.I, Fang S.C. An Electromagnetism-like mechanism for global optimization. *Journal of Global Optimization* 25, 263-282 (2003).

[11] Brüggemann T., Monnot J., Woeginger G.J. Local search for the minimum label spanning tree problem with bounded color classes. *Operations Research Letters* 31, 195-201 (2003).

[12] Cacchiani V., Fernandes Muritiba A.E., Negreiros M., and Toth P. A Multi-Start Heuristic For the Equality Generalized Traveling Salesman Problem, *Networks*, to appear (2010).

[13] Caprara A, Fischetti M, Toth P, Vigo D, Guida P.L. Algorithms for Railway Crew Management. *Mathematical Programming* 79, 125-141 (1997).

[14] Caprara A, Fischetti M, Toth P. A heuristic method for the set covering problem. *Operations Research* 47, 730-743 (1999).

[15] Caprara A, Fischetti M, Toth P. Algorithms for the set covering problem. *Annals of Operations Research* 98, 353-371 (2000).

[16] Caserta M, Tabu search-based metaheuristic algorithm for large-scale set covering problems. In: Doerner K.F, Gendreau M, Greistorfer P, Gutjahr W.J, Hartl R.F, Reimann M (Eds), Metaheuristics, *Progress in Complex Systems Optimization*, Springer,43-63 (2007).

[17] Ceria S, Nobili P, Sassano A. Set Covering Problem. In: Dell'Amico M, Maffioli F, Martello S (Eds), *Annotated Bibliographies in Combinatorial Optimization*, John Wiley and Sons USA: New York; 415-428 (1998).

[18] Cerulli R., Fink A., Gentili M., Voß S.: Metaheuristics comparison for the minimum labelling spanning tree problem. In: Golden B., Raghavan S., Wasil E. (Eds.), *The Next Wave in Computing, Optimization, and Decision Technologies,* Springer-Verlag, Berlin, 93-106 (2008).

[19] Chang R.-S., Leu S.-J. The minimum labeling spanning trees. *Information Processing Letters* 63(5), 277-282 (1997).

[20] Chen S.-H, Chang P.-C, Chan C.-L, Mani V. A hybrid electromagnetism-like algorithm for single machine scheduling problem. *Lecture Notes in Computer Science* 4682, 543-552 (2007).

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

[22] Consoli S., Darby-Dowman K., Mladenovic N., Moreno-Perez J.A. Greedy randomized adaptive search and variable neighbourhood search for the minimum labelling spanning tree problem. *European Journal of Operational Research* 196(2), 440-449 (2009).

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

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

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

[26] Davoudpour H, Hadji Molana M. Solving flow shop sequencing problem for deteriorating jobs by using Electro Magnetic algorithm. *Journal of Applied Sciences* 8(22), 4121-4128 (2008).

[27] Debels D, De Reyck B, Leus R, Vanhoucke M. A hybrid scatter search/electromagnetism metaheuristic for project scheduling*.* *European Journal of Operational Research* 169(2), 638-653 (2006).

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

[29] Eiselt H.A., Sandblom C.-L. Integer Programming and Network Models. Springer, (2000).

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

[31] Fischetti M., Salazar González J.J. , Toth P. A Branch-and-cut Algorithm for the Symmetric Generalized Traveling Salesman Problem, *Operations Research* 45, 378-394 (1997).

[32] Fischetti M., and Toth P. 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 (1988).

[33] Garey M.R, Johnson D.S. Computers and Intractability: A Guide to the Theory of NP-Completeness. Freeman, USA: San Francisco (1979).

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

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

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

[37] Haouari M, Chaouachi J.S. A probabilistic greedy search algorithm for combinatorial optimization with application to the set covering problem. *Journal of the Operational Research Society* 53, 792-799 (2002).

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

[39] Hoshino E. A. and de Souza C. C. Column Generation algorithms for the Capacitated *m*-Ring-Star Problem. *Computing and Combinatorics, 14th Annual International Conference, COCOON ,* Dalian, China *2008, Proceedings*, *LNCS*, 631-641 (2008).

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

[41] Javadian N, Gol Alikhani M, Tavakkoli-Moghaddam R. A discrete binary version of the electromagnetism-like heuristic for solving traveling salesman problem. Lecture Notes in Computer Science 5227; 123-130 (2008).

[42] Krumke S.O., Wirth H.C. On the minimum label spanning tree problem. *Information Processing Letters* 66(2), 81-85 (1998).

[43] Kruskal J.B. On the shortest spanning subset of a graph and the traveling salesman sroblem. In: *Proceedings of the American Mathematical Society* 7(1), 48-50 (1956).

[44] Labbé M., Laporte G., Martin I. R., and Salazar González J. J. The Ring Star Problem: Polyhedral Analysis and Exact Algorithm. *Networks* 43(3), 177-189 (2004).

[45] Labbé M., Laporte G., Martin I. R., and Salazar González J. J. Locating Median Cycles in Networks. *European Journal of Operations Research* 160, 457-470 (2005).

[46] Lan G, DePuy G.W, Whitehouse G.E. An effective and simple heuristic for the set covering problem. *European Journal of Operational Research* 176, 1387-1403 (2007).

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

[48] Lessing L, Dumitrescu I, Stützle T. A comparison between ACO algorithms for the set covering problem. *Lecture Notes in Computer Science* 3172, 1-12 (2004).

[49] Lin S. and Kernighan BW. An Effective Heuristic Algorithm for the Traveling Salesman Problem, *Operations Research* 20, 498-516 (1973).

[50] Maenhout B, Vanhoucke M. An electromagnetism metaheuristic for the nurse scheduling problem. *Journal of Heuristics* 13(4), 359-385 (2007).

[51] Magnanti T, Wolsey L. Optimal trees. In: Ball M, Magnanti T, Monma C, Nemhauser G, editors. Network models. Handbooks in operations research and management science. Amsterdam: North-Holland 7:503-615 (1996).

[52] Mauttone A., Nesmachnow S., Olivera A., Robledo F. A Hybrid Metaheuristic Algorithm to Solve the Capacitated m-Ring Star Problem. *International Network Optimization Conference* (2007).

[53] Mladenovic N., Hansen P. Variable neighborhood search. *Computers & Operations Research* 24, 1097-1100 (1997).

[54] Naderi B, Zandieh M, Khaleghi Ghoshe Balagh A, Roshanaei V. An improved simulated annealing for hybrid flowshops with sequence-dependent setup and transportation times to minimize total completion time and total tardiness. *Expert Systems with Applications* 36(6), 9625-9633 (2009).

[55] Naji-Azimi Z. Comparison of metaheuristic for Examination Timetabling problem. *Journal of Applied Mathematics and computing* 16(1-2), 337-354 (2004).

[56] Naji-Azimi Z. Hybrid Heuristic Methods for Examination Timetabling Problem. *Applied Mathematics and Computation Journal* 163, 705-733 (2005).

[57] Naji-Azimi Z., Salari M., Golden B., Raghavan S., Toth P. Variable Neighborhood Search for the Cost Constrained Minimum Label Spanning Tree and Label Constrained Minimum Spanning Tree Problems. *Computers & Operations Research*, To appear (2010).

[58] Naji-Azimi Z., Salari M., Toth P. An integer linear programming based heuristic approach for the Capacitated *m*-Ring Star Problem. Technical Report, DEIS, University of Bologna (2010).

[59] Naji-Azimi Z., Toth P., Galli L. An Electromagnetism Metaheuristic for the Unicost Set Covering Problem. *European Journal of Operational Research*, To appear (2010).

[60] Noon C. E. The generalized traveling salesman problem, Ph.D. Dissertation, University of Michigan (1988).

[61] Papadimitriou C. H., Steiglitz K. Combinatorial Optimization: Algorithms and Complexity. Prentice-Hall, New Jersey (1982).

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

[63] Prim R.C. Shortest connection networks and some generalizations. In: *Bell System Technical Journal* 36, 1389-1401 (1957).

[64] Rahmati K, Molavi M, Naderi B, Soltani M. A hybridization of simulated annealing and electromagnetism for flowshop problems with skipping probability. *Journal of Applied Sciences* 9(13), 2438-2444 (2009).

[65] Reinelt, G. A Traveling Salesman Problem library. *ORSA Journal on Computing* 3, 376-384 (1991).

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

[67] Salari M, Naji-Azimi Z. Introduction to Electromagnetism algorithm for the Examination Timetabling Problem and comparison of it with other metaheuristics. *Pacific Journal of Optimization* 2, 341-366 (2006).

[68] Salari M., Naji-Azimi Z., Golden B., Raghavan S., Toth P., *The Generalized Covering Salesman Problem.* Submitted to the Informs Journal on Computing.

[69] Salari M., Naji-Azimi Z., Toth P. A Variable Neighborhood Search and its application to a Ring Start Problem Generalization, International Symposium on Combinatorial Optimization, Hammamet, Tunisia March, 2010, Proceeding, *Electronic notes on Discrete Mathematics*, To appear (2010).

[70] Salari M., Naji-Azimi Z., Toth P., A heuristic Procedure for The Capacitated *m*-Ring Star Problem. Submitted to European Journal of Operational Reseach.

[71] Silberholz J. and Golden B. 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 (2007).

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

[73] Solar M, Parada V, Urrutia R. A parallel genetic algorithm to solve the set covering problem. *Computers & Operations Research* 29, 1221-1235 (2002).

[74] Tarjan R. Depth first search and linear graph algorithms. *SIAM Journal of Computing* 1(2), 215-225 (1972).

[75] Tavakkoli-Moghaddam R, Khalili M, Naderi B. A hybridization of simulated annealing and electromagnetism-like mechanism for job shop problems with machine availability and sequence-dependent setup times to minimize total weighted tardiness. *Soft Computing* 13(10), 995-1006 (2009).

[76] Umetani S, Yagiura M. Relaxation heuristics for the set covering problem. *Journal of the Operations Research Society of Japan* 50, 350-375 (2007).

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

[78] Wu P, Yang K.-J, Fang H.-C. A revised EM-like algorithm + K-OPT method for solving traveling salesman problem. First International Conference on Innovative Computing, Information and Control 2006, ICICIC'06;art no.1691858; 546-549 (2006).

[79] Wan Y., Chen G., Xu Y. A note on the minimum label spanning tree. *Information Processing Letters* 84, 99-101 (2002).

[80] Xiong Y., Golden B., Wasil E. A one-parameter genetic algorithm for the minimum labeling spanning tree problem. *IEEE Transactions on Evolutionary Computation* 9(1), 55-60 (2005).

[81] Xiong Y., Golden B., Wasil E. Worst-case behavior of the MVCA heuristic for the minimum labeling spanning tree problem. *Operations Research Letters* 33(1), 77-80 (2005).

[82] Xiong Y., Golden B., Wasil E., Chen S.: The label-constrained minimum spanning tree problem. In: Raghavan S., Golden B., Wasil E. (Eds.), Telecommunications Modeling, Policy, and Technology, Springer 39-58 (2008).

[83] Xiong Y., Golden B., Wasil E. Improved heuristics for the minimum label spanning tree problem*.* *IEEE Transactions on Evolutionary Computation* 10(6), 700-703 (2006).

[84] Yagiura M, Kishida M, Ibaraki T. A 3-flip neighborhood local search for the set covering problem. *European Journal of Operational Research* 172, 472-499 (2006).