Variable Neighborhood Search for the Cost Constrained Minimum Label Spanning Tree and Label Constrained Minimum Spanning Tree Problems

Majid Salari* Zahra Naji-Azimi* Bruce Golden† S. Raghavan†
Paolo Toth*

*DEIS, University of Bologna
Viale Risorgimento 2, 40136, Bologna, Italy
{zahra.najiazimi2, majid.salari2, paolo.toth}@unibo.it

†Robert H. Smith, School of Business, University of Maryland
College Park, US
raghavan@umd.edu, bgolden@rhsmith.umd.edu.

1 Introduction

The Minimum Label Spanning Tree (MLST) problem was introduced by Chang and Leu [2].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 [2]. Several researchers have studied the MLST problem including Brüggemann et al. [1], Cerulli et al. [3], Krumke and Wirth [4], Wan et al. [10], and Xiong et al. [13, 14, 15].
Recently Xiong et al. [12] introduced a more realistic adaptation 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. 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. [12] describe two simple local search heuristics and a genetic algorithm for solving the LCMST problem. They also describe a Mixed Integer Programming (MIP) model, but 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 the dual of the LCMST problem and is another realistic adaptation of the MLST problem. The CCMLST problem was introduced by Xiong et al. [12]. 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. Xiong et al. [12] show that both the LCMST and the CCMLST are NP-Complete.
In this paper, we first focus on the CCMLST problem. We propose a Variable Neighborhood Search (VNS) method for the CCMLST problem. We also adapt the VNS method that we develop to the LCMST problem. We then compare the VNS method to the heuristics described by Xiong et al. [12]. To do so, we consider existing datasets and also design a set of nine Euclidean large-scale datasets, derived from TSPLIB instances [8]. On the LCMST problem we directly compare our VNS method with the heuristic procedures of Xiong et al. [12]. On the CCMLST, we adapt the procedures of Xiong et al. [12] by embedding them in a bisection search. The VNS methods perform extremely well on both the CCMLST and LCMST problems.

2 Variable Neighborhood Search for the CCMLST Problem

In this section, we develop our Variable Neighborhood Search algorithm for the CCMLST problem. We also describe how it can be applied to the LCMST problem. Variable Neighborhood Search is a metaheuristic procedure proposed by Mladenovic and Hansen [6], 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, and 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 enable 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 [5, 8]. 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) = B, because if $B \leq MSTCOST(L)$ the problem is infeasible.

2.1 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 [9]. 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

$$S = \{t \in (L - R) : min\ Components(R \cup \{t\})\} \qquad (2.1.1)$$

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

$$S = \{t \in (L - R) : min\ MSTCOST(R \cup \{t\})\} \qquad (2.1.2)$$

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).

2.2 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.

2.3 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 immediately 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 budget we permanently remove the label, and continue. Otherwise, the label remains in the solution.

2.4 A Bisection Method to Apply Heuristics for the LCMST Problem to the CCMLST Problem

We now describe how any heuristic for the LCMST problem may be applied to the CCMLST problem by using bisection search. Observe that the LCMST problem is also a problem that essentially involves selecting a set of labels for the tree. Given a choice of labels, the cost of the tree is simply the minimum spanning tree cost on the given choice of labels. Thus, we would like to choose a set of at most K labels that minimizes the cost of the spanning tree. 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 bisection search method are provided in Algorithm 1. 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 bisection search, ALG is executed setting the threshold on the number of labels to $(lower + upper)/2$. 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 $(lower + upper)/2$. Anytime the cost of the tree found by ALG is within budget, we decrease the value of upper to $(lower + upper)/2$ The heuristics we use as ALG in the bisection search are the Genetic Algorithm (GA), and the two local search heuristics denoted by LS1 and LS2 that were described by Xiong et al. [12] for the LCMST problem.

Algorithm 1: The Bisection Method

Begin

Set $lower = 1$ *and* $upper = l$

If $(upper - lower) \geq 1$ *then*

$mid = (upper + lower)/2$

$R = ALG(G, mid)$

If $MSTCOST(R) > B$ *then* $lower = mid$

Else $upper = mid$

Endif

Endif

Output $R = ALG(G, upper)$

End

3 Variable Neighborhood Search for the LCMST Problem

We now adapt our VNS procedure for the CCMLST problem to the LCMST problem. Recall, the objective of the LCMST problem is to find a minimum weight spanning tree that uses at most K labels. Consequently, heuristics for the LCMST problem also essentially search for a set of K or fewer labels that minimize the cost of the spanning tree. Our framework for the VNS method for the LCMST problem is essentially the same as the CCMLST problem with a few minor changes that we now explain.

Initialization: As before, we first choose a set of labels in a greedy fashion to ensure that the resulting graph is connected. However, it is easy to observe that if R is a subset of labels, then the cost of an MST on any superset T of R is less than or equal to the cost of the MST on R. In other words, if $R \subseteq T$ then MSTCOST(R) = MSTCOST(T). Consequently, in order to try to minimize the cost, we iteratively add labels to the initial set of labels until we have K labels. The label to be added in any iteration is selected from the set $S = \{t \in (L - R) : min\ MSTCOST(R \cup \{t\})\}$, with ties broken randomly.

Shaking Phase: The shaking phase is identical to the shaking phase for the VNS method for the CCMLST problem. It may result in the selection of labels that do not result in a connected

graph, or result in a set with more than K labels. 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 set of labels exceeds K, we iteratively delete labels from the current set of labels by choosing the label that when deleted results in the lowest cost minimum spanning tree.

Local Search: The local search procedure first adds labels to a given solution until it has K labels. The additional labels are selected iteratively, each time selecting a label that provides the greatest decrease in the cost of the minimum spanning tree, until we have K labels. The local search procedure then 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. This is done in an analogous fashion to the local search procedure in the VNS method for the CCMSLT problem.

4 Computational Results

In this section, we report on an extensive set of computational experiments on both the CCMLST and LCMST problems. 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. We first describe how we generated our datasets, and then discuss our computational experience on these datasets for both CCMLST and LCMST problems.

4.1 Datasets

Xiong et al. [12] 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 used Xiong et al.'s small and medium-sized instances for our tests on the LCMST problem. For these problems, the optimal solutions are known for the small instances, but not for any of the medium-sized instances. We also generated a set of 18 large instances that range from 500 to 1000 nodes. These were created from nine large TSP instances in TSPLIB and considered to be Euclidean. To produce a labeled graph from a TSPLIB instance, we construct a complete graph using the coordinates of the nodes in the TSPLIB instance. The number of labels in the instance is one half of the total number of nodes, and the labels are randomly assigned. For the LCMST problem, we set the threshold on the maximum number of labels as 75 and 150, creating two instances for each labeled graph created from a TSPLIB instance.

We adapted the instances created by Xiong et al. [12] 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 the small instances was to test the quality of the VNS method and the bisection search method on instances where the optimal solution is known. 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) 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).

4.2 Results for CCMLST and LCMST Instances

We first discuss our results on the 191 CCMLST 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). This is in comparison to LS1, LS2, and GA which generated the optimal solution 100, 100, and 102 times, respectively, out of the 104 instances. The average running time of the all the methods was less than 0.2 seconds. For the small and medium-sized instances, the VNS method generated the best solution in 59 out of the 60 instances. This is in comparison to LS1, LS2, and GA which generated the best solution 46, 50, and 50 times, respectively, out of the 60 instances. On the 27 large instances, the VNS method generated the best solution in 26 out of the 27 instances. This is in comparison to LS1, LS2, and GA which generated the best solution 19, 18, and 14 times. As for the LCMST problem, The results clearly show 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.

5 Conclusion

In this paper, we considered both the CCMLST and LCMST problems. We developed a VNS method for both the CCMLST and LCMST problems. We compared the VNS method to optimal solutions for small instances and to 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. [12], 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. Furthermore, for the large instances, its running time is an order of magnitude faster than those of the three other heuristics. For all the 104 instances where the optimal solution was known, the VNS method obtained the optimal solution. On the LCMST instances, the VNS method was also by far the best heuristic. It provided the best solution in all of the 48 small and medium-sized instances. On the large instances, it was by far the best, providing the best solution in 12 out of the 18 instances. For all of the 37 instances where the optimal solution was known, the VNS method obtained the optimal solution. It also ran significantly faster than the three heuristics LS1, LS2, and GA.

References

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

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

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

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

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

[6] N. Mladenovic, and P. Hansen. Variable neighborhood search. *Computers & Operations Research*, 24: 1097–1100, 1997.

[7] R.C. Prim. Shortest connection networks and some generalizations. *Bell System Technical Journal*, 36: 1389–1401, 1957.

[8] G. Reinelt. TSPLIB, A Traveling Salesman Problem Library. *ORSA Journal on Computing*, 3: 376–384 , 1991.

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

[10] Y. Wan, G. Chen, and Y. Xu. A note on the minimum label spanning tree. *Information Processing Letters*, 84: 99–101, 2002.

[11] A. Wolsey Laurence. *Integer programming.* Wiley-Interscience , 1998.

[12] Y. Xiong, B. Golden, E. Wasil, and S. Chen. The label-constrained minimum spanning tree problem. In S. Raghavan, B. Golden, E. Wasil, editors, *Telecommunications Modeling, Policy, and Technology*, Springer, 2008.

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

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

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