Chapter 3

THE LABEL-CONSTRAINED MINIMUM SPANNING TREE PROBLEM

Yupei Xiong
Sysmind LLC
38 Washington Road, Princeton Junction, NJ 08550, USA
yupei72@yahoo.com

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

Edward Wasil
Kogod School of Business, American University
Washington, DC 20016, USA
ewasil@american.edu

Si Chen
College of Business and Public Affairs, Murray State University
Murray, KY 42071, USA
si.chen@murraystate.edu

Abstract Given a positive integer K and a connected, undirected graph G whose edges are labeled (or colored) and have weights, the label-constrained minimum spanning tree (LCMST) problem seeks a minimum weight spanning tree with at most K distinct labels (or colors). In this paper, we prove that the LCMST problem is NP-complete. Next, we introduce two local search methods to solve the problem. Then, we present a genetic algorithm which gets comparable results, but is much faster. In addition, we present two mixed integer programming for-

mulations for the LCMST problem. We compare these on some small problem instances. Finally, we introduce a dual problem.

Keywords: Local search; genetic algorithm; NP-complete; spanning trees; mixed integer programming.

1. Introduction

Computing a minimum weight spanning tree (MST) is one of the fundamental and classic problems in graph theory. Given an undirected graph G with a nonnegative weight on each edge, the MST of G is the spanning tree of G with the minimum total edge weight among all possible spanning trees [1]. This problem and many variants such as the k shortest spanning tree problem [3], the problem of updating a minimum spanning tree [7], the minimum diameter spanning tree problem [5], and the most and least uniform spanning trees problem [1, 4] have been studied extensively. Spanning tree problems have applications in many areas, including network design, VLSI, and geometric optimization.

The minimum label spanning tree (MLST) problem was defined in [2]. In the MLST problem, we are given an undirected graph with labeled (or colored) edges as input. Each edge has a single label (or color) and different edges can have the same label (or color). The goal is to find a spanning tree with the minimum number of distinct labels. In other words, in the MLST problem, we seek to construct a spanning tree whose edges are as similar as possible. The MVCA (Maximum Vertex Covering Algorithm) heuristic was developed to solve the MLST problem [6] and its worst-case performance has been examined [6, 8, 10]. In addition, two effective genetic algorithms were used to solve the MLST problem [9, 11].

In communications networks, there may be many types of communications media including fiber optics, cable, microwave, and telephone lines. Communication along each edge requires a pre-specified media type. If we can reduce the number of different media types in the spanning tree, we reduce the complexity of the communications process. On the other hand, there is also a weight (or distance) associated with each edge. In the label-constrained minimum spanning tree (LCMST) problem, we are trying to find a spanning tree that has minimum total weight while using at most K distinct types of communications media. This is somewhat more realistic than the MLST problem, since the LCMST problem considers the total weight as well as the number of labels in the spanning tree. We now formally define the problem.

Definition (LCMST problem). Given an undirected labeled graph $G = (V, E, L)$, where V is the set of nodes, E is the set of edges, and L is the set of labels, a weight function $w(e)$ for all $e \in E$, and a positive integer K,

find a spanning tree T of G such that the total weight $W_T = \sum_{e \in T} w(e)$ is minimized and the number of distinct labels of T, denoted by $|L_T|$, is no more than K.

In the next section, we show that the LCMST problem is NP-complete. In Section 3, two local search methods are developed and applied to solve the LCMST problem. In Section 4, a genetic algorithm is proposed. In Section 5, computational results are presented and analyzed. In Section 6, a dual problem is discussed. Concluding observations and remarks are provided in Section 7.

2. NP-completeness

Theorem 1. The Label-Constrained Minimum Spanning Tree (LCMST) problem is NP-complete.

Proof. We first show that LCMST belongs to NP. Given an instance of the problem, we consider an arbitrary spanning tree T. The verification algorithm checks that T contains at most K labels and that the total cost of T is no more than a given positive number C. This process can certainly be done in polynomial time.

To prove that LCMST is NP-complete, we show that MLST $\leq_P$ LCMST, which means that MLST is polynomial-time reducible to the LCMST problem. The MLST problem is known to be NP-complete [2]. Let $G = (V, E, L)$ be an instance of the MLST problem. It has a minimum label spanning tree T with K labels. We construct an instance of the LCMST problem as follows. We extend G to the complete graph $G' = (V, E', L')$, where E' contains E and L' contains L. For each $e \in E$, we define its weight $w(e)$ to be 0. For each $e \notin E$, we define its weight $w(e)$ to be 1. For each $e \notin E$, we also define its label to be different from all other labels in G'.

We now show that graph G has a minimum label spanning tree with K labels if and only if graph G' has a minimum weight spanning tree with K labels. Suppose that graph G has a minimum label spanning tree T with K labels. Then, each edge in T has weight 0 in G'. Thus, T has total weight of 0. This tree T is the minimum weight spanning tree in G'. Conversely, suppose that graph G' has a minimum weight spanning tree T with K labels and total weight 0, then all the edges in T are contained in E. Thus, T is also a spanning tree in G with K labels.

Therefore, LCMST is NP-complete. ■

3. Local Search Methods

In this section, two local search methods are introduced. Given a labeled graph G with a weight for each edge, G has a minimum label spanning tree T_1 and a minimum weight spanning tree T_2. Suppose T_1 has n_1 distinct labels

and T_2 has n_2 distinct labels. In the LCMST problem, a positive number K is given. If $K < n_1$, then the LCMST problem has no solution. If $K > n_2$, then the optimal solution to the LCMST problem is T_2, which can be obtained in polynomial time. So, in this paper, we will assume that $n_1 \leq K \leq n_2$.

3.1 Encoding

The solution to the LCMST problem is a spanning tree. Each spanning tree has a label set. Conversely, given any label set $A \in 2^L$, we can obtain a subgraph G_A of G induced by A. If the subgraph G_A is connected and spans all the nodes in V, a minimum weight spanning tree can be found by Prim's algorithm in polynomial time. From this, we can determine the total weight of the spanning tree. In the subgraph G_A, the minimum weight spanning tree may not be unique, but the total weight is unique. Thus, we can build a map $f : 2^L \rightarrow \mathbb{R}$ as follows. For any $A \in 2^L$, if G_A is connected and spans V, then $f(A)$ is the total weight of a minimum weight spanning tree of G_A; otherwise, $f(A) = \infty$. In Figure 3.1, an example which illustrates how the function f works for different label sets in 2^L is presented. In particular, an input graph and three label sets with their corresponding f values are shown. Let $L_K \subset 2^L$ represent the collection of all label sets with K labels. We restrict f to L_K. Then the goal of the LCMST problem is to find $A^* \in L_K$, such that $f(A^*) = \min_{A \in L_K} f(A)$. By this well-defined map f, it is sufficient to consider a label set $A \in L_K$ as a solution to the LCMST problem.

3.2 Local Search 1 (LS1)

In local search 1 (LS1), we begin with an arbitrary label set $A \in L_K$. Suppose $A = \{a_1, a_2, \ldots, a_K\}$. Then, we start a replacement loop as follows. We first replace a_1 by some label $b_1 \in L - A$. To do this, we check each label in $L - A$. If we can make an improvement, then we find b_1 such that $f(A - \{a_1\} + \{b_1\}) = \min_{b \in L-A} f(A - \{a_1\} + \{b\})$. Then, we set $A = A - \{a_1\} + \{b_1\}$. Otherwise, we set $b_1 = a_1$. Next, we seek to replace a_2 by some label $b_2 \in L - A$. To do this, we check each label in $L - A$ and, if we can make an improvement, we obtain $f(A - \{a_2\} + \{b_2\}) = \min_{b \in L-A} f(A - \{a_2\} + \{b\})$. Then, we set $A = A - \{a_2\} + \{b_2\}$. Otherwise, we set $b_2 = a_2$. We continue with this replacement routine until we replace a_K by some suitable b_K (possibly, $b_K = a_K$). After one replacement loop, we improve the label set to $A = \{b_1, b_2, \ldots, b_K\}$. We repeat until no improvement can be made between two consecutive replacement loops.

3.3 Local Search 2 (LS2)

In local search 2 (LS2), we begin with an arbitrary label set $A \in L_K$. Suppose $A = \{a_1, a_2, \ldots, a_K\}$. Then, we start a replacement loop as follows.

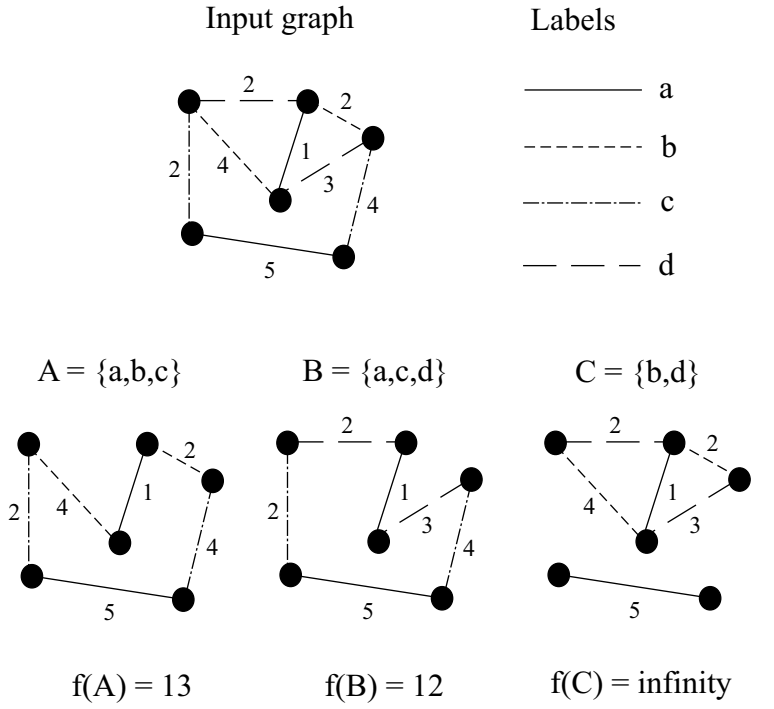

Figure 3.1. An example of encoding

For each $b \in L - A$, we first set $A = A + \{b\}$ and let $a_{K+1} = b$. So, A now has $K + 1$ labels. To maintain K labels, we have to remove one label from A. We check all the labels a_i for $1 \leq i \leq K + 1$ and find a_j such that $f(A-\{a_j\}) = \min_{1\leq i\leq K+1} f(A - \{a_i\})$. Then, we set $A = A-\{a_j\}$. After one replacement loop, we observe, hopefully, some improvement in the label set A. We repeat until no improvement can be made between two consecutive replacement loops.

If we compare the two methods of local search, we obtain the following result.

Theorem 2. A label set A cannot be improved by LS1 if and only if A cannot be improved by LS2.

Proof. Suppose a label set A cannot be improved by LS1. Then, for any label $a \in A$ and $b \notin A$, we know $f(A - \{a\} + \{b\}) \geq f(A)$. If we apply LS2 to the label set A, we first add a label $b \notin A$, and then remove some label $a \in A$. We obtain the label set $A + \{b\} - \{a\} = A - \{a\} + \{b\}$. We still have $f(A + \{b\} - \{a\}) = f(A - \{a\} + \{b\}) \geq f(A)$. So, the label A cannot

be improved by LS2. Conversely, if a label set A cannot be improved by LS2, from the above logic, since $A + \{b\} - \{a\} = A - \{a\} + \{b\}$, we also know that A cannot be improved by LS1. ■

3.4 Running Time Analysis

In the LCMST problem, we are given a labeled graph $G = (V, E, L)$ and a positive integer K. Let $|V| = n$, $|E| = m$, and $|L| = \ell$. We consider only complete graphs here. So, $m = O(n^2)$. For each $A \in 2^L$, we use Prim's algorithm to find a minimum weight spanning tree in the subgraph G_A. Prim's algorithm requires $O(m + n \lg n) = O(n^2)$ running time. So, $f(A)$ requires $O(n^2)$ running time. Both LS1 and LS2 run $f(A)$ at most $K|L| = K\ell$ times in each replacement loop. Thus, the running time of each replacement loop in LS1 and LS2 is $O(K\ell n^2)$.

4. Genetic Algorithm

Given previous success in applying genetic algorithms to the MLST problem [9, 11], we thought a genetic algorithm might also work well for the LCMST problem. In this section, we introduce a genetic algorithm (GA) to solve the LCMST problem. The encoding of the GA is the same as that in local search. Each chromosome is a label set in L_K. Each label is a gene.

4.1 Crossover

Given two parent chromosomes P and Q, one child C is created by the crossover operation. First, we set $R = P \bigcup Q$ and we get a subgraph G_R induced by R. We also set $C = \phi$. Second, we apply Prim's algorithm to the subgraph G_R. In Prim's algorithm, we begin with a random node $v \in V$. Then, we grow a tree by adding the least-weight edge, one at a time, until the tree spans all the nodes in G_R. Each time we add edge e, e has a label c. If $c \notin C$, we set $C = C \bigcup \{c\}$. So, as the tree is growing, so is the label set C. Once $|C|$ reaches K, the label set is restricted to C and Prim's algorithm is applied over the induced subgraph G_C. Finally, we obtain the output, child C, and the minimum weight spanning tree in the subgraph G_C. Of course, C may not be a feasible solution. If an infeasible solution is obtained, we select another start node and run Prim's algorithm again to find another child. If each start node results in infeasibility, $f(C) = \infty$. If K is large, the child C is more likely to be feasible. Crossover is illustrated in Figure 3.2. In this figure, the two parents are $P = \{a, b, c\}$ and $Q = \{a, d, e\}$ and $K = 3$. Note that $f(P) = 23$, $f(Q) = 21$. The union of the two parents is $R = \{a, b, c, d, e\}$. In Figure 3.3, Prim's algorithm starts at the right-most node. The output child is a feasible solution $C = \{b, c, e\}$ and $f(C) = 17$. In Figure 3.4, Prim's algorithm starts

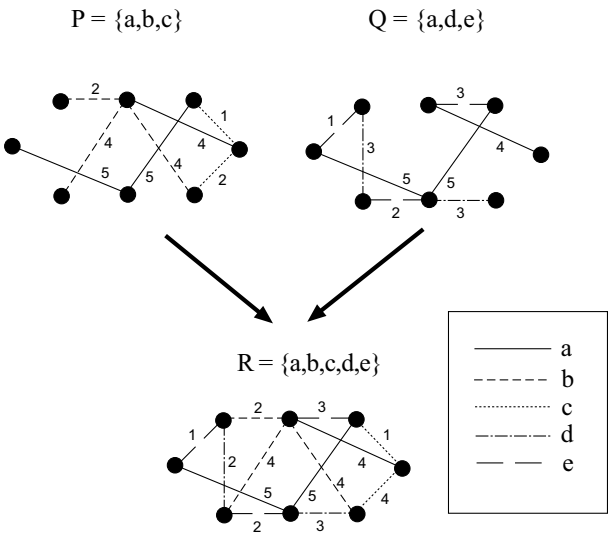

Figure 3.2. An example of crossover: $f(P) = 23$, $f(Q) = 21$.

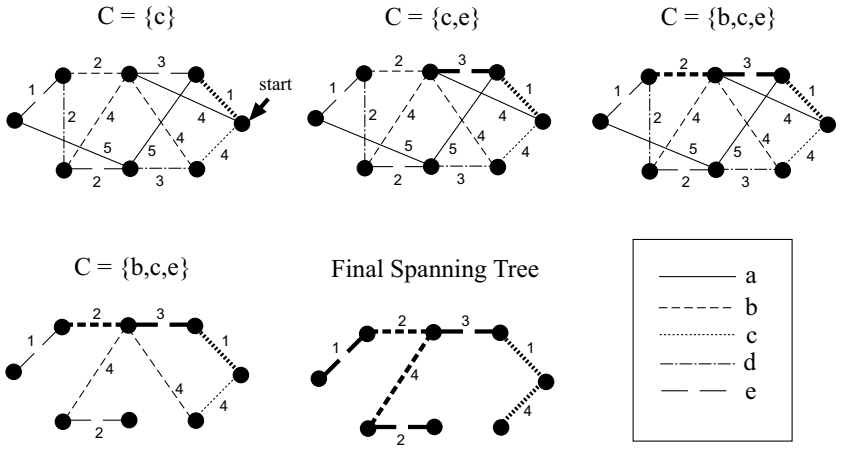

Figure 3.3. Prim's algorithm to get a feasible solution C and $f(C) = 17$, where $K = 3$.

at the left-most node. The output child is an infeasible solution $C = \{b, d, e\}$ and $f(C) = \infty$. The crossover operation is very fast with a running time that is the same as that of Prim's algorithm.

To make the procedure more efficient, we implement the queen-bee crossover in our GA. In each generation, we find the best chromosome QB and designate it the queen-bee chromosome. Then, we only allow crossover between QB and other chromosomes.

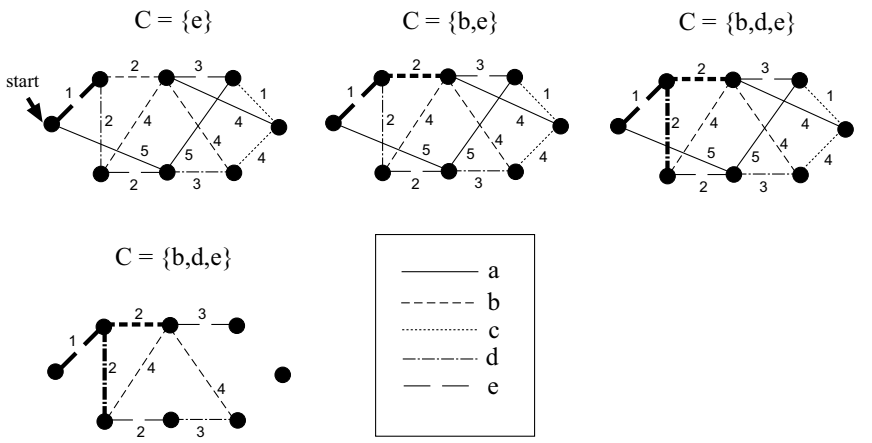

Figure 3.4. Prim's algorithm to get an infeasible solution C and $f(C) = \infty$, where $K = 3$.

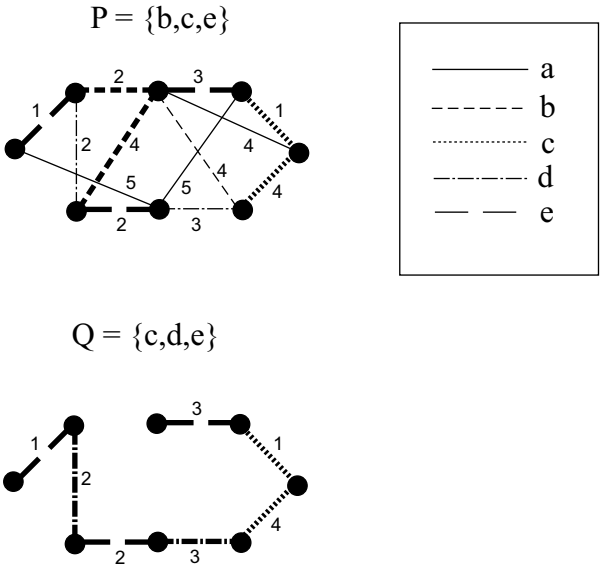

Figure 3.5. An example of mutation: $f(Q) = 16$.

4.2 Mutation

Given one chromosome P, the mutation operation creates a new chromosome Q. Suppose $P = \{p_1, p_2, \ldots, p_K\}$. First, we randomly select $b \in L - P$. Second, we check each p_i for $1 \leq i \leq K$ to see if the replacement of p_i by b obtains a better solution. If it does for some p_i, then we set $Q = P - \{p_i\} + \{b\}$.

If it does not, then we set $Q = P$. We take the child from Figure 3.3 and denote it as the input P in Figure 3.5. We note that $f(P) = 17$. In Figure 3.5, we illustrate mutation. Here, label b is replaced by label d and we obtain an improved solution $Q = \{c, d, e\}$ with $f(Q) = 16$. The mutation operation requires that Prim's algorithm is run at most K times. It is, therefore, slower than the crossover operation.

4.3 Building Each Generation

Suppose the population size is p (we set $p = 20$ in our computational tests). It is easy to randomly generate p chromosomes as the initial generation. To be more specific, we randomly generate K labels and check to see if the induced subgraph is connected and spans V. If it is, then we have a feasible chromosome. If it is not, then we try again. This process continues until we obtain p feasible chromosomes. (Randomness seems to ensure a sufficient level of diversity in the starting population.) To build the next generation, we find the queen-bee chromosome in the current generation. Then, we perform crossover between QB and the $p - 1$ other chromosomes. The crossover of QB and another chromosome P outputs a solution Q. This solution may be infeasible since Prim's algorithm may select the "wrong" labels. For example, in Figure 3.4, labels e, b, and then d are chosen in Prim's algorithm. Since the induced subgraph G_C where $C = \{b, d, e\}$ does not span V, $f(C) = \infty$. The better solution of P and Q becomes the child, which is feasible. Then, we perform mutation for each of the children and obtain $p - 1$ "mutated" children. These children, along with QB, comprise the p chromosomes for the next generation. We stop building generations if the queen-bee chromosome of three consecutive generations does not change. After the last generation, we make a final improvement. We find the queen-bee chromosome QB and execute one replacement loop of LS1 on QB. From the above discussion, we observe that the population size p is the only parameter of GA.

5. Computational Results

5.1 Small Cases

In this section, we report on computational results for small problem instances. For each of these instances, we solve a Mixed Integer Program (MIP) to obtain the optimal solution. This MIP is presented in detail in Appendix A. The computational results for $n = 20$, 30, 40, and 50 are presented in Tables 3.1 to 3.4. The coordinates of the nodes and the label matrix for the instance examined in Table 3.1 are provided in Appendix B.

Table 3.1. Computational results for a graph with $n = \ell = 20$

K	MIP Value	MIP Time(sec)	LS1 Gap(%)	LS2 Gap(%)	GA Gap(%)
2	6491.35	3.30	0.00	0.00	0.00
3	5013.51	7.70	0.00	0.00	0.00
4	4534.67	81.20	0.00	0.00	0.00
5	4142.57	277.56	0.00	0.00	0.00
6	3846.50	216.00	0.00	0.00	0.00
7	3598.05	97.97	0.00	0.00	0.00
8	3436.57	65.56	0.00	0.00	0.00
9	3281.05	74.36	0.00	0.00	0.00
10	3152.05	30.34	0.00	0.00	0.00
11	3034.01	9.77	0.00	0.00	0.00

LS1, LS2, and GA are implemented on a Pentium 4 PC at 1.80 GHz with 256 MB RAM. The MIP formulation is implemented using OPL Studio 3.7 on a Pentium 3 PC at 993 MHz with 512 MB RAM.

For each instance, we test all possible K values. For each combination of instance and K value, we report the MIP (optimal) solution value and the percent gap (i.e., percent above optimality) for LS1, LS2, and GA. We point out that the gap tends to be larger when K is small. This is because there are a relatively small number of feasible solutions for small K. For larger K, all three heuristics work very well. They obtain the optimal solution or the gap is about 1%. For each input, we run LS1 five times and output the best result. The same is true for LS2. Both LS1 and LS2 are very fast on small instances. They each require about one second, on average, for each input (i.e., five runs). The GA is faster; it requires less than half a second for each input (i.e., a single run). The MIP running times are provided in Tables 3.1 to 3.4. When $n = \ell = 50$ (in Table 3.4), these running times are already quite large (we stopped at $K = 8$ because for $K > 8$ the MIP could not find the optimal solutions within 72 hours). We are unable to solve the MIP for instances with more than 50 nodes.

5.2 Large Cases

In this section, we report on computational results for large problem instances ($n \geq 100$). We consider at most two values of K for each combination of n and ℓ. In particular, we use $K = 20$ and $K = 40$. For each instance, we run LS1 and LS2 five times and output the best result. The results are presented in Table 3.5. The GA gap is the percent above the better of LS1 and LS2 for each problem. Running times are presented in Table 3.6. From these two tables, we observe that GA is much faster than LS1 and LS2. Although the solution quality of GA is not as high as that of LS1 and LS2, the gap is, typically, under 1%.

Table 3.2. Computational results for a graph with $n = \ell = 30$

K	MIP Value	MIP Time(sec)	LS1 Gap(%)	LS2 Gap(%)	GA Gap(%)
3	7901.81	20.83	0.00	0.00	0.00
4	6431.58	38.89	0.00	0.00	0.00
5	5597.36	30.47	0.00	0.00	0.00
6	5106.94	174.20	0.00	0.00	0.00
7	4751.00	464.30	0.00	0.46	0.00
8	4473.11	1229.48	0.00	0.00	0.00
9	4196.71	579.63	0.00	0.00	0.00
10	3980.99	931.83	0.52	0.52	0.00
11	3827.23	279.73	0.00	0.00	1.41
12	3702.08	297.36	0.00	0.23	0.00
13	3585.42	90.78	0.00	0.00	0.00

Table 3.3. Computational results for a graph with $n = \ell = 40$

K	MIP Value	MIP Time(sec)	LS1 Gap(%)	LS2 Gap(%)	GA Gap(%)
3	11578.61	9651	0.00	0.00	0.00
4	9265.42	5305	1.72	2.56	1.72
5	8091.45	5841	0.00	0.00	0.75
6	7167.27	1844	0.00	0.00	2.53
7	6653.23	2904	0.13	0.00	0.13
8	6221.63	10744	0.00	0.00	0.00
9	5833.39	16487	0.00	0.00	0.48
10	5547.08	8830	0.00	0.00	0.00
11	5315.92	13641	0.00	0.00	0.00
12	5164.14	109284	0.00	0.00	0.00

Table 3.4. Computational results for a graph with $n = \ell = 50$

K	MIP Value	MIP Time(sec)	LS1 Gap(%)	LS2 Gap(%)	GA Gap(%)
3	14857.09	3285	0.00	0.00	3.08
4	12040.89	5894	0.00	0.00	0.00
5	10183.95	2750	0.00	0.00	0.00
6	9343.69	11820	0.00	0.00	0.00
7	8594.36	74138	0.00	0.00	1.51
8	7965.52	303389	0.00	0.00	0.00

Table 3.5. Computational results for large cases (best solutions in bold)

Cases	LS1	LS2	GA	GA Gap(%)
$n = 100, \ell = 50, K = 20$	**8308.68**	**8308.68**	8335.75	0.33
$n = 100, \ell = 100, K = 20$	**10055.85**	**10055.85**	10138.27	0.82
$n = 100, \ell = 100, K = 40$	7344.72	**7335.61**	**7335.61**	0.00
$n = 150, \ell = 75, K = 20$	11882.62	**11846.80**	11854.17	0.06
$n = 150, \ell = 75, K = 40$	**9046.71**	**9046.71**	9047.22	0.01
$n = 150, \ell = 150, K = 20$	15427.54	**15398.42**	15688.78	1.89
$n = 150, \ell = 150, K = 40$	**10618.58**	10627.36	10728.93	1.04
$n = 200, \ell = 100, K = 20$	**14365.95**	**14365.95**	14382.65	0.12
$n = 200, \ell = 100, K = 40$	**10970.94**	**10970.94**	**10970.94**	0.00
$n = 200, \ell = 200, K = 20$	18951.05	18959.37	**18900.25**	**-0.27**
$n = 200, \ell = 200, K = 40$	**12931.46**	12941.85	12987.29	0.43
$n = 500, \ell = 500, K = 40$	34320.00	**34138.61**	35117.61	2.87

Table 3.6. Running times for large cases (in seconds)

Cases	LS1	LS2	GA
$n = 100, \ell = 50, K = 20$	8	8	7
$n = 100, \ell = 100, K = 20$	13	16	3
$n = 100, \ell = 100, K = 40$	24	26	5
$n = 150, \ell = 75, K = 20$	30	37	12
$n = 150, \ell = 75, K = 40$	52	57	41
$n = 150, \ell = 150, K = 20$	65	56	7
$n = 150, \ell = 150, K = 40$	140	145	13
$n = 200, \ell = 100, K = 20$	72	63	30
$n = 200, \ell = 100, K = 40$	176	222	35
$n = 200, \ell = 200, K = 20$	125	166	29
$n = 200, \ell = 200, K = 40$	342	333	35
$n = 500, \ell = 500, K = 40$	9063	7440	672

In one case, GA outperforms both LS1 and LS2. The largest instance studied has $n = \ell = 500$ and $K = 40$. In this case, the GA solution is 2.87% higher than the LS2 solution. However, LS1 and LS2 require more than 10 times as much running time as GA. We point out that if LS1 and LS2 were run a single time (rather than five), then they would underperform GA in the vast majority of cases. This point is illustrated in Table 3.7.

Additional computational results are presented in Appendix C. These tests also demonstrate the ability of LS1, LS2, and GA to obtain optimal or near-optimal solutions to large problem instances of the LCMST problem.

6. A Related Problem

In the LCMST problem, the primary concern is with the total cost and there is an upper bound on the number of labels. In a related problem, the primary

Table 3.7. Computational results for a graph with $n = \ell = 30$, with a single run of LS1 and LS2

K	MIP	Time(sec)	LS1 Gap(%)	LS2 Gap(%)	GA Gap(%)
3	7901.81	20.83	7.42	0.67	0.00
8	4473.11	1229.48	0.03	0.03	0.00
10	3980.99	931.83	1.61	0.52	0.00
12	3702.08	297.36	0.23	0.23	0.00

concern is with the number of labels and there is an upper bound on the total cost. We refer to this problem as the cost-constrained minimum label spanning tree (CCMLST) problem. Given a labeled graph $G = (V, E, L)$ and a positive number C, the goal of the CCMLST problem is to find a spanning tree with the smallest number of labels and a total cost of no more than C.

There is an easy way to obtain good solutions to the CCMLST problem by using algorithms to solve the LCMST problem. Let ALG represent an algorithm (e.g., LS1, LS2, or GA) to solve the LCMST problem. The input for ALG is a labeled graph G and a positive integer K and the output is a label set X. Let k represent the number of labels in a minimum weight spanning tree of G. Then, we can use a bisection search method to find the solution to the CCMLST problem. To begin, the lower bound is $lower = 1$ and the upper bound is $upper = k$. We run ALG for $mid = \frac{lower+upper}{2}$ labels. If the solution has a cost greater than C, we need more labels and we set $lower = mid$. If the solution has a cost less than C, then we need fewer labels and we set $upper = mid$. A detailed description of the method is shown next.

Input: A labeled graph G and a positive number C.

CCMLST(G,C)

1 Set $lower = 1$ and $upper = k$

2 *if* $(upper - lower) \geq 1$ *do*

3 $\quad mid = \frac{lower+upper}{2}$

4 $\quad X$=ALG(G,mid)

5 $\quad$ *if* $f(X) > C$ *do* $lower = mid$

6 $\quad$ *else do* $upper = mid$

7 *end do*

8 Output X

7. Conclusions

In this paper, we introduced and motivated the LCMST problem. We proved that the LCMST problem is NP-complete. Next, we proposed three heuristics (two local search methods and a genetic algorithm) to solve the LCMST problem and we formulated the problem as a MIP. Extensive computational experiments were performed. In all of these tests, the three heuristics generated high-quality solutions. Finally, a related problem was introduced and a solution approach was presented.

References

[1] P.M. Camerini, F. Maffinoli, S. Martello, and P. Toth. Most and least uniform spanning tree. *Discrete Appl. Math*, 15:181–197, 1986.

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

[3] D. Eppstein. Finding the k smallest spanning trees. *BIT*, 32:237–248, 1992.

[4] Z. Galil and B. Schieber. On finding most uniform spanning trees. *Discrete Appl. Math.*, 20:173–175, 1988.

[5] J.-H. Ho, D.T. Lee, C.-H. Chang, and C.K. Wong. Minimum diameter spanning trees and related problems. *SIAM J. Comput.*, 20:987–997, 1991.

[6] S.O. Krumke and H.-C. Wirth. On the minimum label spanning tree problem. *Inform. Process. Lett.*, 66(2):81–85, 1998.

[7] X. Shen and W. Liang. A parallel algorithm for multiple edge updates of minimum spanning trees. *Proc. 7th Internet. Parallel Processing Symp.*, pages 310–317, 1993.

[8] Y. Wan, G. Chen, and Y. Xu. A note on the minimum label spanning tree. *Inform. Process. Lett.*, 84:99–101, 2002.

[9] 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, 2005.

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

[11] 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.

Appendix: A: MIP Formulations for the LCMST Problem

In this section, we provide two mixed integer programming formulations for the LCMST problem and compare their performances. The first one uses the single-commodity flow model while the second uses the multi-commodity flow model. For small instances of the LCMST problem, we are able to obtain optimal solutions from both formulations. This enables us to compare solution values found using LS1, LS2, and GA with optimal solution values.

Notation and Variables

In the LCMST problem, we assume that L is the set of all labels, E is the set of all edges, $V = \{1, 2, \cdots, n\}$ is the set of all nodes, $|V| = n$ is the total number of nodes, and K is the

maximum number of labels allowed in a solution. Let E_k be the set of all the edges with color k. Additionally, let A be the set of all arcs. In particular, for every edge (i, j) in E, there are two corresponding arcs $< i, j >$ and $< j, i >$ in A. For any $1 \leq i, j \leq n$ and $i \neq j$, we define c_{ij} to be the cost or weight of edge (i, j). We define the variables e_{ij}, x_{ij}, and y_k as follows:

$$e_{ij} = \begin{cases} 1 & \text{if edge } (i,j) \text{ is used} \\ 0 & \text{otherwise} \end{cases}$$
$$x_{ij} = \begin{cases} 1 & \text{if arc } < i,j > \text{ is used} \\ 0 & \text{otherwise} \end{cases}$$
$$y_k = \begin{cases} 1 & \text{if label } k \text{ is selected} \\ 0 & \text{otherwise.} \end{cases}$$

MIP Formulation using the Single-Commodity Flow Model (F1)

In this subsection, we formulate the LCMST problem as a single-commodity flow problem.

$$\min \sum_{(i,j)\in E} c_{ij} e_{ij} \tag{3.A.1}$$

$$\text{subject to} \sum_{(i,j)\in E} e_{ij} = n - 1 \tag{3.A.2}$$

$$\sum_{i:(i,j)\in A} f_{ij} - \sum_{l:(j,l)\in A} f_{jl} = 1 \quad \forall j \in V - \{1\} \tag{3.A.3}$$

$$\sum_{i:(i,1)\in A} f_{i1} - \sum_{l:(1,l)\in A} f_{1l} = -(n-1) \tag{3.A.4}$$

$$f_{ij} + f_{ji} \leq (n-1) \cdot e_{ij} \quad \forall (i,j) \in E \tag{3.A.5}$$

$$\sum_{(i,j)\in E_k} e_{ij} \leq (n-1) \cdot y_k \quad \forall k \in L \tag{3.A.6}$$

$$\sum_{k\in L} y_k \leq K \tag{3.A.7}$$

$$e_{ij},\ y_k \in \{0,1\} \quad \forall (i,j) \in E \quad \forall k \in L \tag{3.A.8}$$

$$f_{ij} \geq 0 \quad \forall (i,j) \in A. \tag{3.A.9}$$

The objective function (3.A.1) seeks to minimize the total cost or weight of the spanning tree. Constraint (3.A.2) ensures that the tree contains $n - 1$ edges. The flow variables f_{ij} are included to guarantee connectivity. They work as follows: select one node (say, node 1) out of the set V and think of it as the root node. Create a supply of $n - 1$ units of flow at this node. For all other nodes, create a demand of one unit of flow. Send one unit of flow from the root node to all other nodes in order to satisfy their demands. The above notion of connectivity is represented by the flow balance constraints (3.A.3) and (3.A.4). Constraints (3.A.5) ensure that if flow is sent along an edge, then that edge must be in the tree. Constraints (3.A.6) ensure that if an edge with label k is used, then this label must be counted. Finally, an upper bound on the number of labels is imposed in constraint (3.A.7).

Suppose we want to formulate the CCMLST as an MIP. The objective function would become

$$\min \sum_{k\in L} y_k$$

and the constraints would include (3.A.2) to (3.A.6), (3.A.8), (3.A.9), and the new constraint

$$\sum_{(i,j)\in E} c_{ij} e_{ij} \leq C. \tag{3.A.7$'$}$$

MIP Formulation using the Multi-Commodity Flow Model (F2)

Alternatively, we can formulate the LCMST problem as a multi-commodity flow problem as follows.

$$\min \sum_{(i,j)\in E} c_{ij}e_{ij} \tag{3.A.10}$$

$$\text{subject to} \sum_{(i,j)\in E} e_{ij} = n-1 \tag{3.A.11}$$

$$\sum_{i:(i,h)\in A} f_{ih}^h - \sum_{l:(h,l)\in A} f_{hl}^h = 1 \quad \forall h \in V-\{1\} \tag{3.A.12}$$

$$\sum_{i:(i,1)\in A} f_{i1}^h - \sum_{l:(1,l)\in A} f_{1l}^h = -1 \quad \forall h \in V-\{1\} \tag{3.A.13}$$

$$\sum_{i:(i,j)\in A} f_{ij}^h - \sum_{l:(j,l)\in A} f_{jl}^h = 0 \quad \forall h \in V-\{1\}, \forall j \neq h \tag{3.A.14}$$

$$f_{ij}^h \leq x_{ij} \quad \forall (i,j) \in A, \forall h \in V-\{1\} \tag{3.A.15}$$

$$x_{ij} + x_{ji} \leq e_{ij} \quad \forall (i,j) \in E \tag{3.A.16}$$

$$\sum_{(i,j)\in E_k} e_{ij} \leq (n-1)\cdot y_k \quad \forall k \in L \tag{3.A.17}$$

$$\sum_{k\in L} y_k \leq K \tag{3.A.18}$$

$$e_{ij},\ y_k \in \{0,1\} \quad \forall (i,j) \in E \quad \forall k \in L \tag{3.A.19}$$

$$x_{ij} \in \{0,1\} \quad \forall (i,j) \in A \tag{3.A.20}$$

$$f_{ij}^h \geq 0 \quad \forall (i,j) \in A, \forall h \in V-\{1\}. \tag{3.A.21}$$

The objective function is the same as in (3.A.1). Constraint (3.A.11) ensures that the tree contains $n-1$ edges. The flow variables f_{ij}^h are included to guarantee connectivity. They work as follows: select one node (say, node 1) out of the set V and think of it as the root node. Create a supply of 1 unit of flow h for every node h ($h \in V-\{1\}$) at node 1. For every node h, create a demand of one unit of flow h. Send one unit of flow h from the root node to every h node in order to satisfy their demands. The above notion of connectivity is represented by the flow balance constraints (3.A.12), (3.A.13), and (3.A.14). Constraints (3.A.15) and (3.A.16) ensure that if flow is sent along an edge, then that edge must be selected in the tree. Constraints (3.A.17) ensure that if an edge with label k is used, then this label must be counted. Finally, an upper bound on the number of labels is imposed in constraint (3.A.18).

Suppose we want to formulate the cost-constrained minimum label spanning tree (CCMLST) as an MIP. The objective function would become

$$\min \sum_{k\in L} y_k$$

and the constraints would include (3.A.9) to (3.A.17), (3.A.19) to (3.A.21), and the new constraint

$$\sum_{(i,j)\in E} c_{ij}e_{ij} \leq C. \tag{3.A.18$'$}$$

Table 3.A.1. Comparison of two MIP formulations based on a graph with $n = \ell = 50$

K	F1 Time (sec)	F2 Time (sec)
4	12040.89	NA
5	10183.95	67473.37
6	9343.69	186148.83
7	8594.36	59783.95
8	7965.52	15869.49
9	NA	6380.30
10	NA	3418.46
11	NA	5133.17
12	NA	20199.94

Table 3.B.1. Coordinates of nodes for the graph with $n = \ell = 20$

node	1	2	3	4	5	6	7	8	9	10
x	41	334	169	478	962	705	281	961	995	827
y	467	500	724	358	464	145	827	491	942	436
node	11	12	13	14	15	16	17	18	19	20
x	391	902	292	421	718	447	771	869	667	35
y	604	153	382	716	895	726	538	912	299	894

Comparison of MIP Models

We compare the performance of F1 and F2 based on a graph with 50 nodes. The formulations are implemented using OPL Studio 3.7 on a Pentium III PC with 993MHz and 512MB RAM. The results are shown in Table 3.A.1, where the time to reach optimality is recorded for each model. NA indicates no optimal solution can be found after 72 hours. As we can see, F2 seems to work better for $K > 8$, while F1 does better for $K \leq 8$. We also conduct experiments on larger examples (graphs with 100 nodes) using F2. However, it fails to find solutions due to a lack of memory.

We point out that constraints (3.A.6) and (3.A.17) can be written in disaggregate form. This increases the number of constraints. In our experiments, the disaggregated formulations did not solve as quickly as F1 and F2.

Appendix: B: A Small Sample Graph for the LCMST Problem

We give a small instance of the LCMST problem. This instance is a complete graph of 20 nodes and 20 labels, where $V = \{1, 2, \ldots, 19, 20\}$ and $L = \{1, 2, \ldots, 19, 20\}$. For each node, its (x, y) coordinates are randomly selected integers from $[0, 999]$. The label matrix of the graph is symmetric. Each entry is randomly selected from L. Computational results for this problem are shown in Table 3.1. In Table 3.B.1, the (x, y) coordinates are provided and the label matrix is given in Table 3.B.2.

Appendix: C: A Special Family of Graphs

We construct a special family of graphs for which we know a reasonably good solution to the LCMST problem. When we run LS1, LS2, and GA on these graphs, we can then compare the solution values to the reasonably good (known) solution values.

Table 3.B.2. The label matrix for the graph with $n = \ell = 20$

–	4	12	3	14	14	5	2	12	14	9	8	5	3	18	18	20	4	2	10	
4	–	19	17	16	11	3	9	7	1	3	5	9	7	6	11	10	11	11	7	
12	19	–	2	14	9	10	4	5	15	17	1	7	17	12	9	5	20	7	4	
3	17	2	–	18	19	19	3	10	2	14	16	20	19	5	11	18	7	14	7	
14	16	14	18	–	2	6	5	13	11	10	18	14	18	13	7	11	2	17	16	
14	11	9	19	2	–	8	16	15	12	13	11	11	2	5	7	11	8	12	8	
5	3	10	19	6	8	–	18	18	8	14	4	6	10	10	19	2	9	3	7	
2	9	4	3	5	16	18	–	7	11	14	9	1	12	3	16	11	20	5	18	
12	7	5	10	13	15	18	7	–	9	4	16	2	3	11	12	17	15	1	17	
14	1	15	2	11	12	8	11	9	–	2	9	20	9	5	2	15	14	20	19	
9	3	17	14	10	13	14	14	4	2	–	19	1	9	8	8	7	14	9	4	
8	5	1	16	18	11	4	9	16	9	19	–	8	2	11	18	14	15	10	17	
5	9	7	20	14	11	6	1	2	20	1	8	–	16	12	1	10	20	17	19	
3	7	17	19	18	2	10	12	3	9	9	2	16	–	4	5	9	5	10	10	
18	6	12	5	13	5	10	3	11	5	8	11	12	4	–	3	16	6	14	4	
18	11	9	11	7	7	19	16	12	2	8	18	1	5	3	–	4	8	15	4	
20	10	5	18	11	11	2	11	17	15	9	14	10	9	16	4	–	9	1	19	
4	11	20	7	2	8	9	20	15	14	14	15	20	5	6	8	9	–	19	1	
2	11	7	14	17	12	3	5	1	20	9	10	17	10	14	15	1	19	–	17	
10	7	4	7	16	8	7	18	17	19	4	17	19	10	4	4	19	1	17	–	

Table 3.C.1. Computational results for a special family of graphs with $r = 80\%$ (best solutions in bold)

Cases	T	LS1	LS2	GA	GA Gap(%)
$n = \ell = 100, K = 40$	8433.69	7418.82	**7418.37**	7418.82	0.01
$n = \ell = 120, K = 40$	9545.52	**8424.78**	**8424.78**	8449.90	0.30
$n = \ell = 150, K = 40$	10120.65	**9682.18**	**9682.18**	**9682.18**	0.00
$n = \ell = 180, K = 40$	10910.71	**10539.14**	10599.39	10549.64	0.10
$n = \ell = 200, K = 40$	11453.89	**11298.40**	**11298.40**	11339.51	0.36

Table 3.C.2. Computational results for a special family of graphs with $r = 50\%$ (best solutions in bold)

Cases	T	LS1	LS2	GA	GA Gap (%)
$n = \ell = 100, K = 40$	7784.84	**7234.45**	**7234.45**	7244.37	0.14
$n = \ell = 120, K = 40$	8846.22	**8253.57**	**8253.57**	**8253.57**	0.00
$n = \ell = 150, K = 40$	9300.71	9037.11	**9032.54**	9045.35	0.14
$n = \ell = 180, K = 40$	10177.16	10131.91	**10127.83**	10150.17	0.22
$n = \ell = 200, K = 40$	10541.98	**10463.57**	**10463.57**	**10463.57**	0.00

Table 3.C.3. Computational results for a special family of graphs with $r = 0\%$ (best solutions in bold)

Cases	T	LS1	LS2	GA	GA Gap(%)
$n = \ell = 100, K = 40$	6870.59	**6870.59**	**6870.59**	**6870.59**	0.00
$n = \ell = 120, K = 40$	7528.71	**7528.71**	**7528.71**	**7528.71**	0.00
$n = \ell = 150, K = 40$	8343.89	**8343.89**	**8343.89**	**8343.89**	0.00
$n = \ell = 180, K = 40$	9006.14	**9006.14**	**9006.14**	**9006.14**	0.00
$n = \ell = 200, K = 40$	9336.30	**9336.30**	**9336.30**	**9336.30**	0.00

We construct a complete graph G as follows. First, we randomly generate n nodes in the square $[0, 999] \times [0, 999]$. The cost of each edge (i, j) is the Euclidean distance between i and j. Then, we sort all the edges by cost, from lowest to highest. A spanning tree requires $n - 1$ edges. We know that Kruskal's algorithm obtains a minimum weight spanning tree by selecting the $n - 1$ shortest edges, without forming a cycle. In order to find a reasonably good solution T to the LCMST problem, we set a parameter r with $0 \leq r < 1$. Next, we select $r(n - 1)$ edges randomly from the $2n$ shortest edges, without forming a cycle. Then, we select the other $(1 - r)(n - 1)$ edges by applying Kruskal's algorithm to the complete graph. Thus, we have tree T. Next, we need to assign a label to each edge. To do this, we randomly select K distinct labels from L and call this label subset L_1. Let $L_2 = L - L_1$. For each edge in T, we assign it a label from L_1, at random; for each edge not in T, we assign it a label in L_2, at random. As a result, we obtain a labeled graph G with a reasonably good spanning tree T which has K distinct labels.

Computational results for $r = 80\%$, $r = 50\%$, and $r = 0\%$ are presented in Tables 3.C.1, 3.C.2 and 3.C.3, respectively. The GA gap is the percentage gap between the GA solution value and the better solution value of LS1 and LS2. From the three tables, we observe that all three heuristics obtain a solution that is no worse than T. In most cases, they are better than T. The

GA gap is always less than 0.4%. Table 3.C.3 gives the computational results for $r = 0\%$. In this case, T is the minimum weight spanning tree. This is because T is generated entirely from Kruskal's algorithm. We observe that all three heuristics obtain the optimal solution. In fact, given the increasing reliance on Kruskal's algorithm as we move from Table 3.C.1 to Table 3.C.2 to Table 3.C.3, we expect T to be a very good solution in Table 3.C.2 and a good solution in Table 3.C.1. Tables 3.C.1, 3.C.2 and 3.C.3 reinforce the conclusion that the three heuristics obtain very high-quality solutions to the LCMST problem.