A One-Parameter Genetic Algorithm for the Minimum Labeling Spanning Tree Problem

Yupei Xiong, Bruce Golden, and Edward Wasil

***Abstract*—Given a connected, undirected graph G whose edges are labeled (or colored), the minimum labeling spanning tree (MLST) problem seeks a spanning tree on G with the minimum number of distinct labels (or colors). In recent work, the MLST problem has been shown to be NP-hard and an effective heuristic [maximum vertex covering algorithm (MVCA)] has been proposed and analyzed. In this paper, we use a one-parameter genetic algorithm (GA) to solve the problem. In computational tests, the GA clearly outperforms MVCA.**

***Index Terms*—Genetic algorithm (GA), NP-hard, spanning trees.**

I. Introduction

In THE minimum labeling spanning tree (MLST) problem, we are given an undirected graph with labeled edges as input. Each edge has a single label and different edges can have the same label. We can think of each label as a unique color. The goal is to find a spanning tree with the minimum number of labels. The problem was first introduced by Chang and Leu [2], motivated by applications in communications network design. In this paper, we describe an effective genetic algorithm (GA) to address the MLST problem.

Chang and Leu [2] proved that the MLST problem is NP-hard and they proposed two heuristics. Their computational experiments showed that the maximum vertex covering algorithm (MVCA) was a very effective heuristic. Krumke and Wirth [5] proposed a modification to MVCA and proved that MVCA can yield a solution no greater than $2\ln n+1$ times optimal, where n is the number of nodes. Wan *et al.* [7] obtained a better bound. They showed that MVCA can yield a solution no greater than $1+\ln(n-1)$ times optimal. In more recent work on the MVCA, Xiong *et al.* [8] improved the performance guarantee to H_b for any graph with label frequency bounded by b (i.e., no label occurs more than b times in G), where $H_b = \sum_{i=1}^{b}(1/i)$ is the bth harmonic number. Next, they presented a worst case family for which the MVCA solution is exactly H_b times the optimal solution. Brüggemann *et al.* [1] used a different approach; they applied local search techniques based on the concept of j-switch neighborhoods to a restricted version of the MLST problem. In addition, they proved a number of complexity results and showed that if each label appears at most twice in the input graph, the MLST problem is solvable in polynomial time.

Manuscript received July 25, 2003; revised March 1, 2004.

Y. Xiong is with the Department of Mathematics, University of Maryland, College Park, MD 20742 USA.

B. Golden is with the R. H. Smith School of Business, University of Maryland, College Park, MD 20742 USA (e-mail: bgolden@rhsmith.umd.edu).

E. Wasil is with the Kogod School of Business, American University, Washington, DC 20016 USA.

Digital Object Identifier 10.1109/TEVC.2004.840145

Since the MVCA heuristic is the most popular and best studied MLST heuristic in the literature, we will use it as a benchmark. MVCA begins with the nodes of G and an empty set of edges as an initial graph. At each iteration, one label is selected and the edges with this label are added to the graph. The goal is to reduce the number of connected components by as many as possible. The procedure continues until the graph is connected. A more detailed description is provided below.

Algorithm: Modified version of MVCA for MLST [5]
Input: A 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.

1) Let $C \leftarrow \emptyset$ be the set of used labels.
2) **repeat**
3) Let H be the subgraph of G restricted to V and edges with labels from C.
4) **for all** $i \in L-C$ **do**
5) Determine the number of connected components when inserting all edges with label i in H.
6) **end for**
7) Choose label i with the smallest resulting number of components and do: $C \leftarrow C \bigcup \{i\}$.
8) **until** H is connected.

The MVCA heuristic has to check all the labels before it selects a specific label which reduces the number of components by the largest amount. It makes sense, but it takes time. MVCA can, and often does, select unnecessary labels. The local-1 search (due to Brüggemann *et al.* [1]) can remove unnecessary labels one by one, but it does not guarantee an optimal solution. In our genetic algorithm, we apply ideas from MVCA in the crossover operation, without checking all the labels at each step. We apply local-1 search in the mutation operation in order to remove unnecessary labels. Based on our computational results, the genetic algorithm beats MVCA in most cases.

In recent years, a wide variety of network design problems has been addressed using genetic algorithms. For example, see Chou *et al.* [3], Palmer and Kershenbaum [6], and Golden *et al.* [4].

II. Genetic Algorithm

During the last three decades, there has been a growing interest in algorithms that rely on analogies to natural processes. The emergence of massively parallel computers, and faster computers in general, has made these algorithms of practical interest. The genetic algorithm (GA) is one of the best known algorithms in this class. It is based on the principle of evolution, operations such as crossover and mutation, and the concept of fitness. In the MLST problem, fitness is the number of distinct labels in the candidate solution. After some number of generations, the algorithm converges and the best individual, we hope, represents a near-optimal solution.

In the MLST problem, we are given a 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. Let $|V| = n$, $|E| = m$, and $|L| = \ell$. An individual (or a chromosome) in a population is a feasible solution, which is defined as a subset C of L such that all the edges with labels in C construct a connected subgraph of G and span all the nodes in G. Each label in C can be viewed as a gene. A random individual can be generated by adding random labels to an empty set until a feasible solution emerges. So, it is not difficult to build a population of individuals.

After we get the initial population, we can apply crossover and mutation operations in order to build one generation from the last. The details of the GA are discussed next.

A. Encoding

Ultimately, a solution to the MLST problem is a spanning tree. However, it is somewhat inconvenient to encode a spanning tree and much easier to think in terms of a feasible solution (as defined above). We, therefore, encode a feasible solution (i.e., a list of labels). Two simple observations follow.

Observation 1. If C is a feasible solution and G_C is the subgraph induced by C, then any spanning tree of G_C has at most $|C|$ labels.

Observation 2. If C is an optimal solution and G_C is the subgraph induced by C, then any spanning tree of G_C is a minimum labeling spanning tree of G.

From the above observations, to solve the MLST problem, we seek a feasible solution with the least number of labels. Then, we want an arbitrary spanning tree from the subgraph induced by this feasible solution. Thus, encoding a feasible solution enables us to solve the MLST problem. An illustration of encoding is provided in Fig. 1.

B. Crossover

Crossover builds one offspring from two parents. It begins by forming the union of the two parents, then sorts the labels in the union in descending order of their frequencies. The operator adds labels in their sorted order to the initially empty offspring until the offspring represents a feasible solution. The following sketch summarizes the operator. Fig. 2 presents an example of its application.

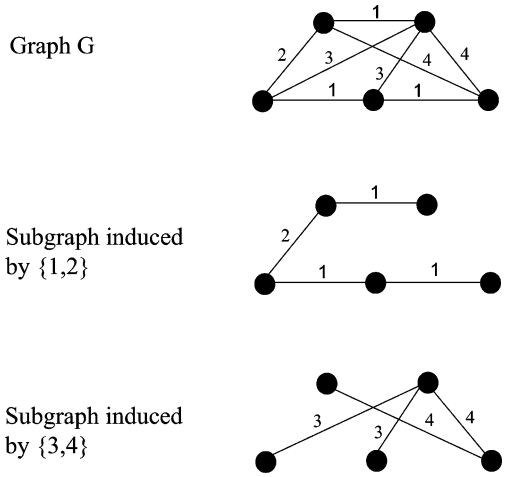


Fig. 1. Encoding feasible solutions: {1,2} and {3,4} are two feasible solutions.

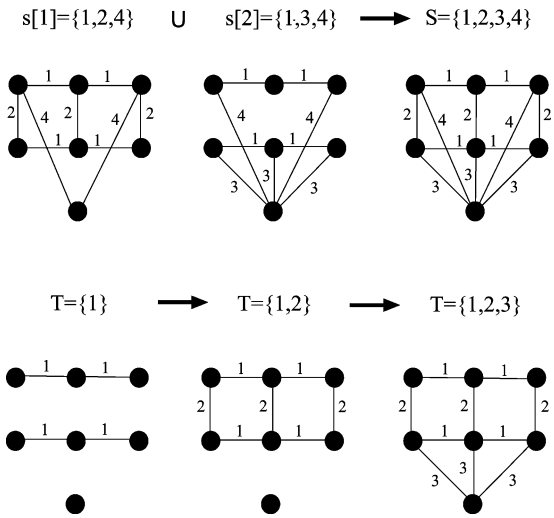


Fig. 2. Example of crossover.

Crossover(s[1],s[2])

1) Let $S = s[1] \bigcup s[2]$ and T be an empty set.
2) Sort S in decreasing order of the frequency of labels in G.
3) Add labels of S, from the first to the last, to T until T represents a feasible solution.
4) Output T.

C. Mutation

Given a solution S, a new solution $T = \text{mutation}(S)$ can be built using our mutation operation. First, a new label is added to S. Next, labels are removed (i.e., the associated edges), from the least frequently occurring label to the most, as long as S remains feasible. A more detailed description of the mutation operation follows. An example is presented in Fig. 3.

In Fig. 3, after adding a label, we remove label 4 first. Next, we try to remove label 3, but cannot without violating feasibility. Likewise, label 2 cannot be removed. Finally, label 1 is removed to obtain the mutation T.

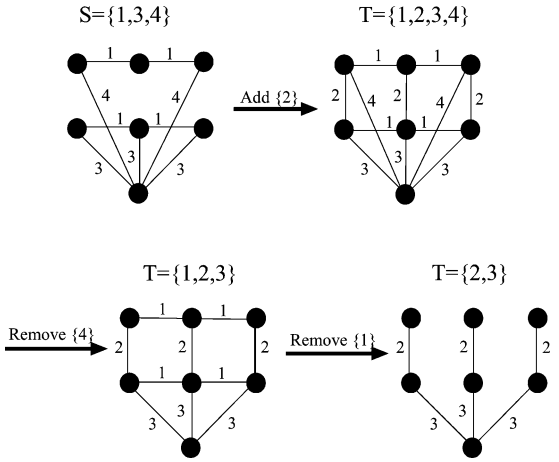


Fig. 3. Example of mutation.

Mutation*(S)*

1) Randomly select c not in S and let $T = S \bigcup \{c\}$.
2) Sort T in decreasing order of the frequency of labels in G.
3) From the last label of the above list to the first, try to remove one label from T and keep T as a feasible solution.
4) Repeat 3) until no labels can be removed.
5) Output T.

D. Building Each Generation

In our genetic algorithm, we use only one parameter: the population size. In each generation, the probability of crossover is 100% and the probability of mutation is 100%. There is no biased random selection from one generation to the next and there is no fine-tuning of numerous parameters required in order to determine an effective combination of parameter values. The genetic algorithm is designed to be simple and easy to replicate.

In many applications of metaheuristics (e.g., tabu search, genetic algorithms, ant colony optimization, etc.) to combinatorial optimization, performance is highly dependent on and sensitive to the parameter values. If a metaheuristic performs well with only a single parameter, then, in our view, the procedure is especially well-suited to solve the problem under study.

Suppose the initial generation consists of p individuals. These individuals are denoted by $s[0], s[1], \cdots, s[p-1]$. We can build generation k from generation $k-1$ as follows, where $1 \leq k < p$. In all, there are p generations (i.e., $0, 1, \cdots, p-1$). Note that the number of generations is equal to the initial population. An example of generation building (with $p = 4$) is shown in Fig. 4.

1) **For** each j from 0 to $p-1$, do
2) $t[j] = \text{crossover}(s[j], s[(j+k) \bmod p])$
3) $t[j] = \text{mutation}(t[j])$
4) $s[j]$ = the better solution of $s[j]$ and $t[j]$
5) **End For**.

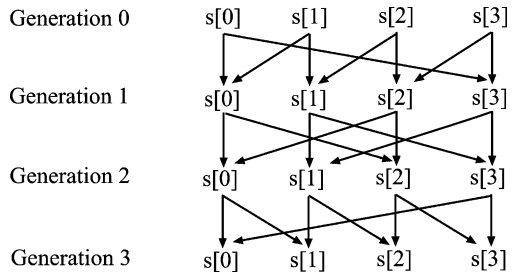


Fig. 4. Build generations with $p = 4$.

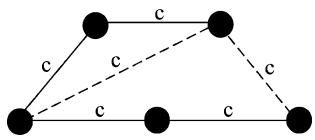


Fig. 5. General frequency of label c is 6. There are two cycle edges (dashed lines). So, the net frequency of label c is 4.

E. Running Time Analysis

Let us begin with an analysis of the crossover operation. Given a subset C of L, we can use depth-first search (DFS) to determine whether the subgraph of G restricted to V and edges with labels from C is connected. The running time of DFS is $O(m + n)$. In each crossover operation, a feasible solution has $O(\ell)$ labels. Merge sort is used to combine two feasible solutions and this requires $O(\ell)$ computations. In step 3, we can add at most $O(\ell)$ labels and we use DFS after each addition to determine whether a feasible solution has been obtained. Thus, the worst case running time of a crossover operation is $O(\ell(m + n))$.

Similarly, the worst case running time of a mutation operation is $O(\ell(m+n))$. Each generation requires p crossovers and p mutations, and there are p generations in all. Therefore, the worst case running time of GA is $O(p^2\ell(m + n))$. A running time analysis of MVCA is provided by Krumke and Wirth [5].

III. Computational Results

The one-parameter GA and the MVCA heuristic were compared on 78 cases of the MLST problem. This section describes these comparisons. Throughout, $d = (m/\binom{n}{2}) = (2m/n(n-1))$ is the density of the graph.

The frequency of labels plays an important role in the GA, but for a label c, we may have two different definitions of its frequency. One definition is the number of appearances of c in G. This is called the general frequency. The other definition is the number of appearances in the subgraph induced by c after removing cycle edges. This is called the net frequency. An example is shown in Fig. 5. So, we have two GAs to report on. GA1 is described in Section II. GA1 represents the GA using general frequency and GA2 is a minor variant which uses the concept of net frequency. In other words, the concept of net frequency is based on the fact that a minimum labeling spanning tree will contain no cycles and GA2 tries to take advantage of this fact.

We use OPT to represent the optimal MLST solution. We obtain the optimal solution via backtrack search. Given $0 < k \leq \ell$, backtrack search tries all possible subsets of L with k labels

TABLE I
COMPUTATIONAL RESULTS FOR GROUP I

	OPT	GA1	GA2	MVCA
$n = \ell = 20, d = 0.8$	2.40	2.40	2.40	2.50
$n = \ell = 20, d = 0.5$	3.20	3.25	3.25	3.40
$n = \ell = 20, d = 0.2$	6.85	6.85	6.85	7.00
$n = \ell = 30, d = 0.8$	2.45	2.45	2.45	2.65
$n = \ell = 30, d = 0.5$	3.70	3.70	3.70	3.80
$n = \ell = 30, d = 0.2$	7.25	7.30	7.30	7.70
$n = \ell = 40, d = 0.8$	2.95	2.95	2.95	2.95
$n = \ell = 40, d = 0.5$	3.85	3.95	3.95	3.95
$n = \ell = 40, d = 0.2$	7.20	7.20	7.25	7.85
$n = \ell = 50, d = 0.8$	3.00	3.00	3.00	3.00
$n = \ell = 50, d = 0.5$	4.00	4.05	4.05	4.20
$n = \ell = 50, d = 0.2$	7.45	7.65	7.65	8.25

and determines whether a feasible solution exists. We can sort all the labels by their frequencies. If the sum of the frequencies of the k labels is less than $n-1$, we have an infeasible solution. We increase k until a feasible solution is found or a running time limit is exceeded.

The running time of backtrack search grows exponentially, but if the problem size is small or the optimal solution is small, the running time is reasonable. In our experiments, the optimal solution is reported unless a single instance requires more than 20 min of CPU time. In such a case, we report not found (NF).

We use n, ℓ, d as input. For each input combination, we construct 20 random graphs and we apply MVCA, GA1, GA2, and OPT to each one. Each procedure is run on each instance once only. The output is the average number of labels for each solution procedure. In the GAs, the population size is set to 20 if n, $\ell \leq 100$, and 30 if $n > 100$. We divide all the computational tests into three groups as described next.

A. Group I

Group I consists of small graphs, with $n = \ell \leq 50$. There are three density levels: high ($d = 0.8$), medium ($d = 0.5$), and low ($d = 0.2$). We can generate optimal solutions here without difficulty. The average number of labels is reported for 12 input combinations (or cases) in Table I. GA1 beats MVCA in nine of these cases and ties MVCA in three cases. GA1 beats GA2 in one case and ties GA2 in 11 cases.

B. Group II

Group II consists of a specially constructed family of graphs. In addition, the graphs are larger than the Group I graphs. For each member in this family, each label appears exactly b times and we know an optimal solution has $\lceil (n-1)/b \rceil$ labels (see Appendix A for details). This family is constructed in order to challenge the label selection strategy of MVCA. At each step, many labels are available for selection since each one reduces the number of components equally. Thus, MVCA has a greater chance of selecting the labels incorrectly. The results for 21 input combinations of n, ℓ, and b are presented in Table II. The optimal solution value is also included. GA1 beats MVCA in all of these cases (often by a wide margin). GA1 beats GA2 in three cases and GA2 beats GA1 in two cases. There are 16 ties. Therefore, both GAs clearly outperform MVCA for Group II graphs.

TABLE II
COMPUTATIONAL RESULTS FOR GROUP II

	Optimal	GA1	GA2	MVCA
$n = \ell = 50,\ b = 4$	13	13.00	13.00	14.25
$n = \ell = 50,\ b = 5$	10	10.00	10.00	11.55
$n = \ell = 50,\ b = 10$	5	5.90	5.90	6.30
$n = \ell = 50,\ b = 15$	4	4.00	4.00	4.65
$n = \ell = 50,\ b = 20$	3	3.00	3.00	3.60
$n = \ell = 75,\ b = 4$	19	19.00	19.00	21.30
$n = \ell = 75,\ b = 7$	11	11.40	11.40	12.90
$n = \ell = 75,\ b = 15$	5	6.00	6.00	6.65
$n = \ell = 75,\ b = 25$	3	4.00	4.00	4.50
$n = \ell = 100,\ b = 4$	25	25.20	25.20	28.95
$n = \ell = 100,\ b = 10$	10	11.30	11.25	12.60
$n = \ell = 100,\ b = 20$	5	6.75	6.75	7.00
$n = \ell = 100,\ b = 30$	4	5.00	4.95	5.05
$n = \ell = 150,\ b = 4$	38	38.00	38.00	42.65
$n = \ell = 150,\ b = 15$	10	12.10	12.15	13.10
$n = \ell = 150,\ b = 30$	5	7.00	7.00	7.30
$n = \ell = 150,\ b = 45$	4	5.00	5.00	5.10
$n = \ell = 200,\ b = 4$	50	50.95	50.95	57.60
$n = \ell = 200,\ b = 20$	10	13.10	13.10	13.75
$n = \ell = 200,\ b = 40$	5	7.45	7.50	7.90
$n = \ell = 200,\ b = 60$	4	5.40	5.45	5.55

TABLE III
COMPUTATIONAL RESULTS FOR GROUP III WITH $n = 50$

	OPT	GA1	GA2	MVCA
$n = 50,\ \ell = 12,\ d = 0.8$	1.10	1.10	1.10	1.10
$n = 50,\ \ell = 12,\ d = 0.5$	1.95	1.95	1.95	1.95
$n = 50,\ \ell = 12,\ d = 0.2$	3.20	3.30	3.30	3.45
$n = 50,\ \ell = 25,\ d = 0.8$	2.00	2.00	2.00	2.00
$n = 50,\ \ell = 25,\ d = 0.5$	2.95	2.95	2.95	2.95
$n = 50,\ \ell = 25,\ d = 0.2$	5.00	5.10	5.10	5.45
$n = 50,\ \ell = 50,\ d = 0.8$	3.00	3.00	3.00	3.00
$n = 50,\ \ell = 50,\ d = 0.5$	4.00	4.05	4.05	4.20
$n = 50,\ \ell = 50,\ d = 0.2$	7.45	7.65	7.65	8.25
$n = 50,\ \ell = 62,\ d = 0.8$	3.00	3.10	3.10	3.40
$n = 50,\ \ell = 62,\ d = 0.5$	4.15	4.45	4.50	4.75
$n = 50,\ \ell = 62,\ d = 0.2$	NF	8.55	8.55	9.20

C. Group III

Group III consists of graphs with a range of sizes and different values of ℓ for each n. In particular, ℓ is set to $0.25n$, $0.5n$, n, and $1.25n$ (approximately). In Tables III–VI, results from 48 input combinations (cases) are presented. GA1 beats MVCA in 30 cases. MVCA beats GA1 in three cases and there are 15 ties. GA1 beats GA2 in four cases and GA2 beats GA1 in five cases. There are 39 ties.

To build a Group I or Group III graph, we are given n, ℓ, and d. So, the number of edges should be $m = dn(n-1)/2$. We begin with a graph that consists of the n nodes only. Next, we randomly generate m distinct edges. For each edge e, we label it with a random label from L. We check to see if the graph is connected. If yes, we stop. If not, we continue to generate random edges with random labels and add them until the graph is connected. Thus, the density of the graph might actually be a little higher than d. This is most likely to happen when $n < 10$ and $d = 0.2$ and is unlikely to happen when $n \geq 20$.

TABLE IV
COMPUTATIONAL RESULTS FOR GROUP III WITH $n = 100$

	OPT	GA1	GA2	MVCA
$n = 100, \ell = 25, d = 0.8$	1.45	1.50	1.45	1.45
$n = 100, \ell = 25, d = 0.5$	2.00	2.00	2.00	2.00
$n = 100, \ell = 25, d = 0.2$	3.95	3.95	3.95	4.00
$n = 100, \ell = 50, d = 0.8$	2.00	2.00	2.00	2.00
$n = 100, \ell = 50, d = 0.5$	3.00	3.00	3.00	3.05
$n = 100, \ell = 50, d = 0.2$	NF	5.95	5.95	6.15
$n = 100, \ell = 100, d = 0.8$	3.00	3.20	3.15	3.60
$n = 100, \ell = 100, d = 0.5$	NF	4.95	4.95	4.90
$n = 100, \ell = 100, d = 0.2$	NF	8.90	8.90	9.40
$n = 100, \ell = 125, d = 0.8$	3.95	4.00	4.00	4.05
$n = 100, \ell = 125, d = 0.5$	NF	5.40	5.40	5.65
$n = 100, \ell = 125, d = 0.2$	NF	10.15	10.15	10.95

TABLE V
COMPUTATIONAL RESULTS FOR GROUP III WITH $n = 150$

	OPT	GA1	GA2	MVCA
$n = 150, \ell = 37, d = 0.8$	1.85	1.85	1.85	1.85
$n = 150, \ell = 37, d = 0.5$	2.00	2.00	2.00	2.00
$n = 150, \ell = 37, d = 0.2$	4.00	4.00	4.00	4.10
$n = 150, \ell = 75, d = 0.8$	2.00	2.10	2.10	2.30
$n = 150, \ell = 75, d = 0.5$	3.00	3.05	3.00	3.35
$n = 150, \ell = 75, d = 0.2$	NF	6.30	6.30	6.60
$n = 150, \ell = 150, d = 0.8$	NF	3.95	3.95	3.95
$n = 150, \ell = 150, d = 0.5$	NF	5.05	5.05	5.10
$n = 150, \ell = 150, d = 0.2$	NF	9.80	9.80	10.35
$n = 150, \ell = 187, d = 0.8$	NF	4.05	4.10	4.35
$n = 150, \ell = 187, d = 0.5$	NF	6.00	6.00	6.00
$n = 150, \ell = 187, d = 0.2$	NF	11.45	11.45	11.85

TABLE VI
COMPUTATIONAL RESULTS FOR GROUP III WITH $n = 200$

	OPT	GA1	GA2	MVCA
$n = 200, \ell = 50, d = 0.8$	2.00	2.00	2.00	2.00
$n = 200, \ell = 50, d = 0.5$	2.00	2.00	2.00	2.00
$n = 200, \ell = 50, d = 0.2$	4.00	4.00	4.00	4.30
$n = 200, \ell = 100, d = 0.8$	2.20	2.35	2.25	2.65
$n = 200, \ell = 100, d = 0.5$	3.00	3.30	3.30	3.75
$n = 200, \ell = 100, d = 0.2$	NF	6.80	6.80	6.90
$n = 200, \ell = 200, d = 0.8$	NF	4.00	4.00	4.00
$n = 200, \ell = 200, d = 0.5$	NF	5.75	5.85	5.80
$n = 200, \ell = 200, d = 0.2$	NF	10.85	10.85	10.90
$n = 200, \ell = 250, d = 0.8$	NF	4.80	4.85	4.75
$n = 200, \ell = 250, d = 0.5$	NF	6.50	6.50	6.50
$n = 200, \ell = 250, d = 0.2$	NF	12.25	12.15	12.55

D. Summary

In total, we tested 78 input combinations (there are three duplicated combinations when $n = \ell = 50$ in Tables I and III). GA1 beats MVCA in 58 cases. MVCA beats GA1 in three cases and there are 17 ties. GA2 beats MVCA in 57 cases. MVCA beats GA2 in three cases and there are 18 ties. In Table I and Tables III–VI, we applied backtrack search successfully in 740 instances (there are 60 duplicated instances when $n = \ell = 50$ in these tables). GA1 obtains an optimal solution in 701 or 94.73% of the instances. GA2 obtains an optimal solution in 704 or 95.14% of the instances. Although one might have predicted superior performance from GA2, the computational results indicate otherwise. In fact, GA1 and GA2 are essentially the same in quality of performance, but GA2 is slightly more time consuming. GA2 requires $O(\ell(m+n))$ additional computations to find the net frequency for each label. From the point of view of running time, GA1 is slightly slower than MVCA. This is not a major issue, however, since the longest running time for GA1 to solve a single MLST problem instance is under 7 s of CPU time on a Pentium 4 PC with 1.80 GHz and 256 MB RAM, while GA2 takes at most 8 s. Thus, the GAs are very fast, they contain a single parameter, and they typically outperform MVCA. Based on these results, we recommend GA1 as a fast and effective solution procedure for the MLST problem. We point out that if $p = 20$ for all n and ℓ, then GA1 and GA2 perform only slightly worse for $n > 100$. Similarly, if $p = 30$ for all n and ℓ, GA1 and GA2 perform slightly better for n, $\ell \leq 100$.

As mentioned, in 20 of the 78 cases, MVCA ties or beats GA1. We ran one final experiment in which we included the MVCA solution in the initial population and ran GA1. In five of the cases, a slight improvement resulted.

TABLE VII
GROUP II GRAPH WITH $n = \ell = 10$ AND $b = 3$

Labels	Edges
1	(8,2) (2,6) (3,8)
2	(7,2) (5,3) (3,2)
3	(7,9) (6,8) (10,6)
4	(5,4) (2,4) (9,8)
5	**(10,9) (9,3) (3,4)**
6	**(4,6) (6,7) (7,8)**
7	(10,4) (2,10) (6,1)
8	**(2,5) (5,1) (1,10)**
9	(7,1) (5,7) (9,5)
10	(10,7) (4,8) (3,10)

IV. CONCLUDING REMARKS

In this paper, we present a one-parameter genetic algorithm for the minimum labeling spanning tree problem which is an NP-hard problem. The GA is simple, fast, and effective. We compare the GA with the MVCA, the most popular MLST heuristic in the literature. Based on extensive computational tests, the GA clearly outperforms MVCA. This is a nice example of the ability of GAs to successfully solve difficult NP-hard problems. A final point of interest is as follows: for the worst case family of graphs (presented by Xiong *et al.* [8]), mentioned in Section I, MVCA can perform poorly, but GA1 obtains the optimal solution in each case.

APPENDIX A
CONSTRUCTING GRAPHS FOR WHICH THE OPTIMAL SOLUTION IS KNOWN

Given inputs $n = \ell$ and b, we can build a Group II graph as follows. Suppose the graph is $G = (V, E, L)$, where $V = 1, 2, \ldots, n$. First, we randomly select $\text{opt} = \lceil (n-1)/b \rceil$ different labels from L. Second, for each of these labels, we assign it to b edges for a total of $b \cdot \text{opt}$ edges. We must ensure that the $b \cdot \text{opt}$ edges construct a connected subgraph of G and span all nodes in G. Thus, the $\lceil (n-1)/b \rceil$ labels

construct an optimal solution, since at least $b \cdot \mathrm{opt}$ edges are required for a feasible solution. Third, for each of the other labels in L, we assign it to b randomly selected edges. We now have the graph. An example is shown in Table VII with $n = \ell = 10$ and $b = 3$. In this example, the optimal solution is {5,6,8}. It is indicated in bold.

REFERENCES

[1] T. Brüggemann, J. Monnot, and G. J. Woeginger, "Local search for the minimum label spanning tree problem with bounded color classes," *Oper. Res. Lett.*, vol. 31, pp. 195–201, 2003.

[2] R.-S. Chang and S.-J. Leu, "The minimum labeling spanning trees," *Inf. Process. Lett.*, vol. 63, no. 5, pp. 277–282, 1997.

[3] H. Chou, G. Premkumar, and C.-H. Chu, "Genetic algorithms for communications network design—an empirical study of the factors that influence performance," *IEEE Trans. Evol. Comput.*, vol. 5, no. 3, pp. 236–249, 2001.

[4] B. Golden, S. Raghavan, and D. Stanojević, "Heuristic search for the generalized minimum spanning tree problem," *INFORMS J. Comput.*, 2004, to be published.

[5] S. O. Krumke and H.-C. Wirth, "On the minimum label spanning tree problem," *Inf. Process. Lett.*, vol. 66, no. 2, pp. 81–85, 1998.

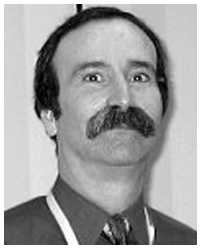
[6] C. C. Palmer and A. Kershenbaum, "An approach to a problem in network design using genetic algorithms," *Networks*, vol. 26, no. 3, pp. 151–163, 1995.

[7] Y. Wan, G. Chen, and Y. Xu, "A note on the minimum label spanning tree," *Inf. Process. Lett.*, vol. 84, pp. 99–101, 2002.

[8] Y. Xiong, B. Golden, and E. Wasil, "Worst case behavior of the MVCA heuristic for the minimum labeling spanning tree problem," *Oper. Res. Lett.*, vol. 33, no. 1, pp. 77–80, 2005.

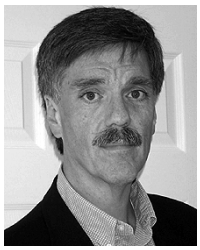
Yupei Xiong received the B.S. and M.S. degrees in mathematical science from Peking University, Beijing, China. He is currently working towards the Ph.D. degree in applied math at the University of Maryland, College Park.

His areas of expertise include discrete applied math and computer science.



Bruce Golden is the France-Merrick Chair in Management Science in the Robert H. Smith School of Business, University of Maryland, College Park. His research interests include heuristic search, combinatorial optimization, and applied OR.

Dr. Golden has received numerous awards, including the Thomas L. Saaty Prize (1994), the University of Maryland Distinguished Scholar–Teacher Award (2000), and the INFORMS Award for the Teaching of OR/MS Practice (2003). In 2004, he was named an INFORMS Fellow. He is currently Editor-in-Chief of *Networks* and was previously Editor-in-Chief of the *INFORMS Journal on Computing*.



Edward Wasil is the Tarek Omar Chair of Organizational Transformation in the Kogod School of Business, American University, Washington, DC, where he has taught courses in managerial statistics, production and operations management, and operations research for 19 years. He serves as the Feature Article Editor of the *INFORMS Journal on Computing*. He conducts research in applied data mining and network optimization.