An Effective Genetic Algorithm for the Minimum-Label Spanning Tree Problem

Jeremiah Nummela
Program in Bioinformatics
University of Michigan
Ann Arbor, MI 48109 USA
jnummela@umich.edu

Bryant A. Julstrom
Department of Computer Science
St. Cloud State University
St. Cloud, MN 56301 USA
julstrom@stcloudstate.edu

ABSTRACT

Given a connected, undirected graph G with labeled edges, the minimum-label spanning tree problem seeks a spanning tree on G to whose edges are attached the smallest possible number of labels. A greedy heuristic for this NP-hard problem greedily chooses labels so as to reduce the number of components in the subgraphs they induce as quickly as possible. A genetic algorithm for the problem encodes candidate solutions as permutations of the labels; an initial segment of such a chromosome lists the labels that appear on the edges in the chromosome's tree. Three versions of the GA apply generic or heuristic crossover and mutation operators and a local search step. In tests on 27 randomly-generated instances of the minimum-label spanning tree problem, versions of the GA that apply generic operators, with and without the local search step, perform less well than the greedy heuristic, but a version that applies the local search step and operators tailored to the problem returns solutions that require on average 10% fewer labels than the heuristic's.

Categories and Subject Descriptors

G.2.1 [**Mathematics of Computing**]: Discrete Mathematics—*Combinatorics*; I.2.8 [**Problem Solving, Control Methods, and Search**]: Heuristic Methods

General Terms

Algorithms

Keywords

Labeled spanning trees, spanning subgraphs, greedy heuristics, genetic algorithms, local search

1. INTRODUCTION

Given a connected, undirected graph G with labeled edges, the minimum-label spanning tree problem seeks a spanning tree on G to whose edges are attached the smallest possible number of distinct labels. This problem arises in the design of "uniform" networks that use as few different communications media, or links owned by as few parties, as possible.

A genetic algorithm for this problem encodes candidate solutions as permutations of the available labels. An initial segment of such a chromosome lists the edges in a spanning subgraph of the target graph. Every permutation represents a valid solution, so operators that generate permutations from parent permutations can be used. Better results are possible, however, when the GA's operators are tailored to the problem. Three versions of the GA, then, apply generic and heuristic operators as well as a local search step that removes redundant labels.

In tests on 27 randomly-generated instances of the minimum-label spanning tree problem of up to 400 vertices, 2000 edges, and 600 available labels, the three versions of the GA were compared to each other and to an effective greedy heuristic for the problem. The versions with generic operators, with or without the local search step, are less effective than the greedy heuristic, but with heuristic operators and the local search step, the GA returns solutions that require on average about 10% fewer labels than do the greedy heuristic's solutions, though the GA uses more time.

Section 2 below describes the minimum-label spanning tree problem in more detail, and Section 3 describes the greedy heuristic. Section 4 presents the coding, operators, structure, and parameter values of the genetic algorithm, and Section 5 presents the tests that compared the GA versions and the greedy heuristic.

2. THE PROBLEM

Let G= (V, E) be a connected, undirected graph with vertices V and edges E in which each edge is *labeled* or *colored* with a label from a finite set L. Edges' labels are not in general unique; several edges may share a label. The graph G in Figure 1 is such a graph.

For a subgraph H of G, let $\ell(H)$ be the set of labels associated with H's edges. A minimum-label spanning tree (MLST) of G is then a spanning tree T of G for which the number of labels in $\ell(T)$ is a minimum; that is, a MLST's edges are labeled with the smallest possible number of distinct labels. the minimum-label spanning tree problem seeks a MLST on a labeled graph G. Graphs J and K in Figure 1 are candidate labeled trees on G. K is a MLST on G; it uses only three of the four labels that appear in G.

An alternate formulation of the problem, which we adopt,

GECCO'06, July 8–12, 2006, Seattle, Washington, USA.
Copyright 2006 ACM 1-59593-186-4/06/0007 ...$5.00.

seeks a smallest set of labels whose edges span G; any spanning tree within such a spanning subgraph is necessarily a MLST.

The MLST problem was first described by Chang and Leu [2], who presented it as a variant of the minimum spanning tree problem and demonstrated that it is NP-hard, even when G is complete. They described an exact algorithm based on A* search as well as two heuristics for the problem. The second of these, a greedy heuristic that they called the Maximum Vertex Covering Algorithm (MVCA) and which Section 3 describes, was particularly effective.

Krumke and Wirth [5] established bounds on both the time and the performance of the MVCA. In particular, they showed that it returned results within a factor of $(2 \ln n + 1)$ of optimum, a result that Wan, Cheng, and Xu [7] improved to $(\ln(n-1)+1)$. Xiong, Golden, and Wasil [9] showed that the MVCA always returns a number of labels within H_b of optimum, where $H_b = \sum_{i=1}^{b}(1/i)$ is the b^{th} harmonic number, and that this bound is tight.

Xiong, Golden, and Wasil [8] also described a genetic algorithm for the MLST problem. It encoded candidate solutions as sets of labels whose edges span the target graph. Its crossover operator selected the offspring's labels from the union of the parents' labels, and its mutation added one label and removed any labels rendered redundant by the addition. The removal of redundant labels also constituted a local search step. On problem instances of up to 200 vertices and 250 labels, the GA in general returned spanning trees with fewer labels than those identified by the MVCA heuristic.

3. A GREEDY HEURISTIC

The Maximum Vertex Covering Algorithm of Chang and Leu [2] is classically greedy. Beginning with an empty set of labels and the subgraph of the target graph comprised of its isolated vertices, it builds a solution—a selection of labels and thus of the edges the labels indicate—one label at a time. At each step, the label added to the solution minimizes the number of components in the subgraph induced by the labels selected so far. When the number of components falls to one—when the induced subgraph becomes connected—the heuristic returns an arbitrary spanning tree of the subgraph. The following sketch summarizes the MVCA. In it, H is the subgraph induced by the selected labels.

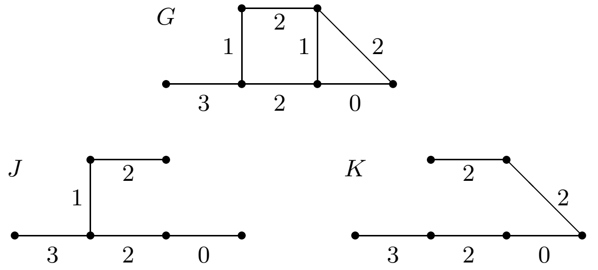

Figure 1: An instance G of the minimum-label spanning tree problem with six vertices, seven edges, and four distinct labels. J and K are candidate solutions to this instance, and K is an optimum solution.

```
$H \leftarrow (V, \emptyset)$;
while $H$ is not connected
    $l \leftarrow$ a label whose inclusion most reduces
        the number of components in $H$;
    add to $H$ the edges $l$ labels;
return a spanning tree on $H$;
```

Chang and Leu observed that the time the MVCA requires is $O(\ell mn)$, where $n = |V|$, $m = |E|$, and $\ell = |L|$ is the number of available labels.

4. A GENETIC ALGORITHM

Call a set of labels whose induced subgraph spans the target graph *feasible*. The subgraph a feasible set of labels induces will in general contain many spanning trees. Like the genetic algorithm of Xiong, Golden, and Wasil [8], the present GA searches the space of feasible sets of labels rather than the larger space of spanning trees. This section describes the GA's coding and operators, then its structure and its parameter values in the tests of Section 5.

4.1 Coding and Evaluation

Candidate solutions are sets of labels that induce spanning subgraphs of the target graph. Chromosomes are permutations of the available labels; a prefix of a chromosome lists the labels in the feasible set that the chromosome represents.

Chromosomes are decoded and evaluated by an algorithm similar to the MVCA heuristic. To identify the set of labels that a chromosome represents, this algorithm scans the chromosome, beginning with its first label. When the labels scanned so far induce a connected subgraph of the target graph, the process halts and the number of labels is returned as the chromosome's fitness. Every permutation represents a solution; there are no invalid chromosomes. Also, the coding is highly redundant: rearranging the labels in a chromosome's feasible set may not change it, and the labels that follow the feasible set can appear in any order.

The following sketch summarizes the evaluation algorithm. In it, H is a subgraph of the target graph, and `c[·]` is the chromosome being evaluated.

```
$H \leftarrow (V, \emptyset)$;
$i \leftarrow 0$;
while $H$ is not connected
    add to $H$ the edges c[i] labels;
    $i \leftarrow i + 1$;
return $i$;
```

For example, the permutation 2310 encodes a candidate solution for the graph G in Figure 1. Evaluating this chromosome induces subgraphs that contain progressively fewer components. The edges labeled 2, 3, and 1 induce a connected subgraph of G, so the fitness of the chromosome is three. The last entry in the chromosome is not examined. Evaluation's time is $O(\ell m)$.

4.2 Operators

Researchers have developed a variety of crossover operators for permutations [4, pp.52–56]. Among them is Davis' Order Crossover (OX) [3]. OX constructs an offspring permutation from two parents by (1) choosing two cut points at random within the parents; (2) copying one parent into the offspring between the cut points; and (3) copying the remaining values from the second parent in order from its first value.

A crossover more appropriate to the MLST problem and the coding of its candidate solutions builds an offspring by alternating values from the two parents, then eliminating duplicates; each value's first appearance survives. For example, the alternation of 03214 and 13024 is 0133201244; removing duplicates yields 01324. This operator keeps labels from the parents' feasible sets near the beginning of the offspring, where they are more likely to appear in the solution the offspring represents. It is therefore likely to be more effective than OX on the MLST problem.

An obvious mutation operator for permutations swaps the values at two random positions in one parent chromosome, though if both positions fall within the chromosome's feasible set, or both fall outside it, the subgraph the chromosome represents may not be changed, as the previous section observed. In particular, if the number of available labels—thus the length of the chromosome—is ℓ and the chromosome's fitness is f, then the probability that this operator swaps a currently-used label for one not currently used is

$$\frac{f}{\ell}\frac{\ell - f}{\ell} + \frac{\ell - f}{\ell}\frac{f}{\ell} = \frac{2f(\ell - f)}{\ell^2}.$$

When, for example, $\ell = 100$ and $f = 25$, this probability is only 0.375.

A mutation variant thus swaps a random label from the parent's feasible set with a random label not in the set; that is, it randomly selects one label from the first f positions of the chromosome, and the other from the remaining $\ell - f$ positions. Restricting the labels that may be swapped increases the likelihood that the offspring's solution will differ from its parent's. We call this operation heuristic mutation.

4.3 Local Search

The evaluation function counts the labels a chromosome lists until the subgraph the labels induce becomes connected. It is possible that labels listed later render earlier labels redundant; that is, earlier labels can be removed without disconnecting the induced subgraph.

An additional step, then, can follow evaluation. This step reexamines in order the labels a chromosome lists and removes from the feasible set those whose removal does not disconnect the induced subgraph. It swaps each such label with the last label in the feasible set and decrements the chromosome's fitness—the count of necessary labels—by one. This step can be thought of as directed local search around the chromosome's original solution.

The following sketch summarizes the local search step, whose time, like that of evaluation, is $O(\ell m)$. Again, H is the subgraph that the chromosome c[·] represents, and f is c[·]'s fitness.

```
$H \leftarrow$ the subgraph c[·] represents;
$f \leftarrow$ the number of labels in $H$;
$i \leftarrow 0$;
while ( $i < f$ )
    remove from $H$ the edges c[$i$] labels;
    if ( $H$ is connected )
        $f \leftarrow f - 1$;
        swap(c[$i$],c[$f$]);
    else
        return to $H$ the edges c[$i$] labels;
        $i \leftarrow i + 1$;
```

Table 1: The operators implemented by the three versions of the genetic algorithm for the minimum-label spanning tree problem.

	GA version		
Operator	GA1	GA2	GA3
Crossover	Order Crossover	Order Crossover	Alternation
Mutation	Unrestricted swapping	Unrestricted swapping	Heuristic swapping
Local search?	No	Yes	Yes

For example, the permutation 2013 encodes a candidate solution to the problem instance G in Figure 1. As the chromosome is decoded, all four labels must be added before the vertices of G are connected. Reconsidering the labels discovers that removing 2 disconnects the subgraph, but removing 0 will not, so 0 is swapped with 3, and the chromosome's fitness is reduced by one. Removing either 3 or 1 disconnects the graph, so the chromosome resulting from this step is 2310, with fitness three.

An alternate version of this operator moves each redundant label to the end of the feasible set and shifts all the remaining feasible labels down one position, thereby preserving their order. Swapping was implemented because it is faster: $O(1)$ rather than $O(\ell)$.

4.4 Structure

The coding of candidate label sets and the variation and local search operators for it were implemented in a generational genetic algorithm. The algorithm initializes its population with random permutations of the available labels. It chooses chromosomes to be parents in k-tournaments without replacement, and it generates each offspring by applying either crossover or mutation to the appropriate number of parents. The GA is 1-elitist, preserving the best chromosome from the current generation into the next, and it runs through a fixed number G of generations.

Three versions of the GA differ in their variation operators and whether or not they apply the local search of Section 4.3. The first version, called GA1, applies Order Crossover and mutation by unrestricted swapping, and it does not use local search. The second version, GA2, augments GA1 with local search. The third version, GA3, applies alternating crossover, heuristic mutation, and local search. Table 1 summarizes these choices.

In the tests the next section describes, the GA's parameters were unchanged across its three versions. Its population contained 100 chromosomes, the size of its selection tournaments was $k = 2$, the probability that crossover generated each offspring was 70% (so the probability of mutation was 30%), and it ran through $G = 100$ generations.

5. TESTS

The Maximum Vertex Covering Algorithm of Chang and Leu and the three versions of the genetic algorithm were compared on 27 randomly-generated instances of the minimum-label spanning tree problem. These instances ranged in size from 200 vertices, 1000 edges, and 100 available labels to 400 vertices, 2000 edges, and 600 available labels. The leftmost column of Table 2 lists the parameters of the test instances.

Table 2: Results of the MVCA heuristic of Chang and Leu and of the sets of trials of the three GA variants on the test MLSTP instances. For each instance, the table lists its numbers of vertices, edges, and labels. For the greedy heuristic, it lists the number of labels in the heuristic's solution and the time the algorithm required. For each set of 30 trials of each GA variant, it lists the number of labels in the best solution found, the mean number of labels, the standard deviation of the 30 results, and the average time the GA required.

Instance															
$\lvert V\rvert$				GA1				GA2				GA3			
$\lvert E\rvert$		MVCA		Labels			Mean	Labels			Mean	Labels			Mean
$\lvert L\rvert$	Num	Labels	Time	Best	Mean	StdDev	time	Best	Mean	StdDev	time	Best	Mean	StdDev	time
200	1	20	0.05	19	22.53	1.41	5.50	18	19.30	0.60	12.63	18	18.03	0.18	14.78
1000	2	21	0.06	21	22.87	1.17	5.29	19	19.67	0.61	12.46	18	18.07	0.25	15.18
100	3	20	0.06	21	23.43	1.33	5.77	18	19.57	0.82	12.42	18	18.00	0.00	14.73
200	1	32	0.16	36	39.40	1.81	8.29	30	31.40	0.81	18.40	28	28.00	0.00	21.03
1000	2	31	0.16	37	40.20	1.92	8.97	30	31.47	0.94	19.04	28	28.40	0.50	22.11
200	3	30	0.16	35	39.23	2.42	7.54	28	30.30	1.12	17.95	27	27.00	0.00	19.99
200	1	41	0.32	45	52.40	2.59	11.56	38	40.87	1.22	25.19	25	35.73	0.45	27.09
1000	2	37	0.29	47	51.97	2.70	11.48	36	39.27	1.53	23.54	34	34.00	0.00	26.31
300	3	39	0.29	47	54.03	2.41	11.46	39	42.07	1.60	24.34	36	36.00	0.00	25.35
400	1	78	0.47	83	90.57	4.29	27.20	68	70.03	1.00	54.03	68	68.00	0.00	59.25
1000	2	72	0.43	79	86.50	3.75	24.08	65	66.20	0.71	49.82	63	63.77	0.43	55.02
200	3	77	0.48	83	91.47	4.31	24.15	68	69.87	0.86	51.09	67	67.90	0.31	56.36
400	1	116	1.46	159	185.53	13.74	51.19	104	106.87	1.76	80.04	100	100.07	0.25	92.15
1000	2	105	1.32	147	174.23	13.16	46.98	99	103.47	1.87	78.28	95	95.33	0.48	84.67
400	3	113	1.44	166	185.90	14.41	53.87	104	106.77	1.50	82.37	99	99.80	0.66	90.40
400	1	145	2.77	260	290.00	22.66	78.87	137	142.77	1.85	106.82	131	131.63	0.56	112.37
1000	2	134	2.58	241	290.20	22.18	83.03	131	136.27	2.27	103.79	122	123.07	0.52	111.62
600	3	133	2.59	253	285.27	21.32	67.39	127	131.13	2.33	100.45	119	120.00	0.74	108.10
400	1	40	0.43	45	49.40	2.08	24.63	39	40.97	1.13	53.72	36	37.03	0.41	59.65
2000	2	43	0.46	47	50.70	1.99	23.50	40	41.13	0.82	52.85	37	37.17	0.38	58.76
200	3	38	0.41	45	49.10	1.86	24.75	39	40.17	0.95	53.62	35	35.93	0.58	59.05
400	1	66	1.43	81	86.03	2.53	42.73	63	66.23	1.74	86.15	56	57.23	0.57	91.18
2000	2	61	1.33	78	84.10	3.36	47.07	62	65.60	1.81	87.55	55	55.60	0.56	90.43
400	3	68	1.48	80	85.73	3.03	44.21	64	68.33	1.88	87.34	58	58.50	0.51	90.95
400	1	78	2.54	110	121.77	5.59	62.86	81	84.00	2.23	108.29	69	70.03	0.61	105.86
2000	2	80	2.60	109	125.17	7.48	67.74	82	86.37	2.34	111.55	70	71.30	0.75	112.75
600	3	79	2.58	114	123.47	4.88	55.35	84	86.70	1.90	108.04	71	71.87	0.57	110.73

A MLSTP instance is a connected graph with labeled edges. A collection of random edges on n vertices will not in general form a connected graph, so construction of each instance began with a random spanning tree on the vertices, identified by decoding a random Prüfer number [6] [1, p.48] of length $n-2$. This tree was augmented with random edges up to the required number, and each edge was assigned a random label. There was no mechanism to ensure the use of every label, so the values of $|L|$ in Table 2 are upper bounds on the numbers of labels that actually appear. Optimum solutions to the instances are not known.

The MVCA was run once, and each version of the GA was run 30 independent times, on each test instance, on a Pentium 4 processor with 1Gb of memory running at 2.53 GHz under Red Hat Linux 9.0. Table 2 summarizes the results of these trials. For each instance, the table lists the number of labels in the subgraph the MVCA found and the wall-clock time it required and, for each version of the GA, the number of labels in the best solution found, the mean number of labels, the standard deviation of the 30 results, and the average time that the GA required.

The values in the table show that GA1, which applies Order Crossover and unrestricted swapping and which does not use local search, is not competitive with the heuristic of Chang and Leu. Despite taking far more time, its results are similar to the MVCA's only on the smallest instances. It rapidly falls behind as the numbers of vertices, edges, and labels increase. Over all 27 instances, the solutions GA1 identifies are on average 42.2% greater than those of the MVCA.

GA2 augments GA1 with the local search step of Section 4.3 and does much better. It identifies subgraphs with fewer labels on average than those of the MVCA on the majority of the smaller problem instances. On the largest instances, with $|V| = 400$ vertices, $|E| = 2000$ edges, and $|L| = 400$ or 600 available labels, the MVCA retains a decisive advantage of up to 9.7% over the average of GA2's results. However, GA2's local search step imposes a significant time penalty; it takes between two and three times as long as does GA1.

GA3 replaces the naive variation operators of GA1 and GA2 with the alternating crossover and heuristic mutation of Section 4.2, and it retains the local search step. This combination enables GA3 to outperform the earlier versions of the GA and the MVCA on all the problem instances. On average, its solutions require about 10% fewer labels than the solutions returned by either GA2 or the MVCA.

GA3 requires slightly more time than does GA2, but it converges to good solutions in fewer generations, as Figure 2 illustrates. The figure plots the average performances of the three GA versions and the value returned by the MVCA on the first instance with $|V| = 400$, $|E| = 2000$, and $|L| = 200$. It illustrates that GA1 cannot compete with the other algorithms, that GA2 spends a lot of time approaching the

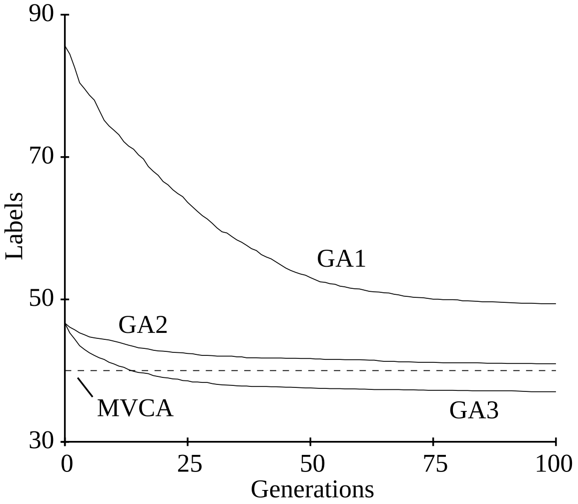

Figure 2: The average run-time behavior of the three versions of the genetic algorithm on the first MLSTP instance with $|V| = 400$ vertices, $|E| = 2000$ edges, and $|L| = 200$ available labels. The dotted line indicates the number of labels in the subgraph identified by the MVCA heuristic.

MVCA's result, and that GA3 identifies better solutions quickly. Figure 2 also indicates that, with the local search step appended to evaluation, the initial populations of both GA2 and GA3 begin with much better candidate solutions than does GA1.

Each modification that transforms GA1 into GA3 contributes to the latter's relative success. The local search oeprator, by removing redundant labels, explicitly improves candidate solutions and modifies chromosomes to represent and preserve those improvements. Each label's position in a chromosome determines its contribution, if any, to the solution the chromosome represents. Alternating crossover tends to preserve the positions and order of labels from the parent chromosomes. Restricting swapping in mutation to a label in the parent chromosome's feasible set and one not in the feasible set makes it likely that the offspring chromosome will represent a different selection of labels, so that search is more effective than with unrestricted swapping.

6. CONCLUSION

Given a connected, undirected graph G with labeled edges, the minimum-label spanning tree problem seeks a spanning tree on G with the minimum number of distinct labels. The Maximum Vertex Covering Algorithm of Chang and Leu greedily chooses labels so as to reduce the number of components in the subgraphs they induce as quickly as possible. A genetic algorithm for the problem represents candidate subgraphs as permutations of the available labels; a prefix of each chromosome lists the labels in the subgraph it represents.

A version of the GA with generic operators for permutations cannot compete with the MVCA, but augmenting it with a local search step that removes redundant labels improves its performance considerably. A third version applies operators tailored to the minimum-label problem and retains the local search step; though it takes much more time than the MVCA, its results on a selection of test instances are on average about 10% better.

This inquiry can be extended in several ways. The speed with which GA3 converges suggests that it might achieve yet better results with reduced selection pressure or with a more aggressive mutation operator, so as to maintain diversity in the population. At the cost of additional time, both mutation and the local search operator could shift segments of chromosomes, thus better conserving labels' orders, rather than swapping labels.

A more thorough investigation of the operators' properties would test them individually and in each possible combination. The present inquiry omits, for example, a GA with order crossover but heuristic mutation and one with alternating crossover and heuristic mutation but without local search.

The versions of the present GA both resemble and differ from the algorithm of Xiong, Golden, and Wasil [9]. The latter algorithm should be included in the comparisons.

The LCMST problem seeks a set of labels that induces a connected subgraph of the target graph and is as small as possible. The apparent success of GA3 suggests that the permutation coding and the operators tailored to it might be effective in EAs for other problems of subset selection such as knapsack and facility location problems.

7. REFERENCES

[1] Peter J. Cameron. *Combinatorics: Topics, Techniques, Algorithms*. Cambridge University Press, Cambridge, 1994.

[2] Ruay-Shiung Chang and Shing-Jiuan Leu. The minimum labeling spanning trees. *Information Processing Letters*, 63:277–282, 1997.

[3] Lawrence Davis. Applying adaptive algorithms to epistatic domains. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 162–164, 1985.

[4] A. E. Eiben and J. E. Smith. *Introduction to Evolutionary Computing*. Springer-Verlag, Berlin, 2003.

[5] Sven O. Krumke and Hans-Christoph Wirth. On the minimum label spanning tree problem. *Information Processing Letters*, 66:81–85, 1998.

[6] H. Prüfer. Neuer beweis eines satzes über permutationen. *Archives of Mathematical Physics*, 27:742–744, 1918.

[7] Yingyu Wan, Guoliang Chen, and Yinlong Xu. A note on the minimum label spanning tree. *Information Processing Letters*, 84:99–101, 2002.

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

[9] Yupei Xiong, Bruce Golden, and Edward Wasil. Worst-case behavior of the MVCA heuristic for the minimum labeling spanning tree problem. *Operations Research Letters*, 33(1):77–80, 2005.