Chapter 9

COMPARISON OF HEURISTICS FOR SOLVING THE GMLST PROBLEM

Yiwei Chen[†], Namrata Cornick[‡], Andrew O. Hall[§], Ritvik Shajpal[#], John Silberholz[§], Inbal Yahav[§], Bruce L. Golden[§*]

[†] *Department of Electrical Engineering, Stanford University, Stanford, CA, 94305*

[‡] *Department of Applied Mathematics, University of Maryland, College Park, MD, 20742*

[§] *R. H. Smith School of Business, University of Maryland, College Park, MD, 20742*

[#] *Department of Geography, University of Maryland, College Park, MD, 20742*

Abstract Given a graph G whose edges are labeled with one or more labels, the Generalized Minimum Label Spanning Tree problem seeks the spanning tree over this graph that uses the least number of labels. We provide a mathematical model for this problem and propose effective greedy heuristics and metaheuristics. We finally compare the results of these algorithms with benchmark heuristics for the related Minimum Label Spanning Tree problem.

Keywords: Combinatorial optimization; computational comparison; genetic algorithm; greedy heuristic; metaheuristic; minimum label spanning tree.

1. Introduction

The Generalized Minimum Label Spanning Tree (GMLST) problem is a variant of the Minimum Label Spanning Tree (MLST) problem. The problem takes as input an undirected graph $G = (V, E, L)$, where G is defined to have V as the node set, E as the edge set, and L as the label set, with $n = |V|$ and $m = |L|$. While in the MLST problem, each edge is colored by exactly one

[*] Corresponding author. E-mail address *bgolden@rhsmith.umd.edu*

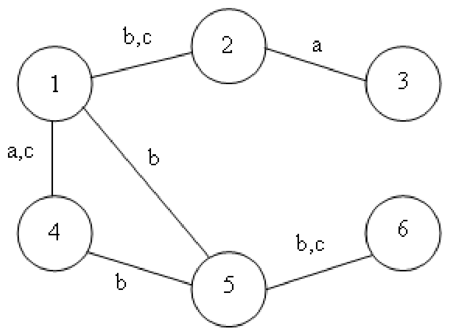

Figure 9.1. Sample GMLST problem graph

label from the set L, in the GMLST problem, each edge has an associated label set, which is a subset of L. In this manner, edge $e \in E$ has an associated label set $l(e) \subset L$. The optimal solution to the GMLST problem is a minimum label spanning tree, T, such that each edge $e \in T$ has been colored by a label in $l(e)$ and T uses the smallest number of distinct labels.

Consider the sample graph found in Figure 9.1. One feasible spanning tree connects nodes 2 and 3 with label a, nodes 1 and 2 with label c, nodes 1 and 4 with label c, nodes 4 and 5 with label b, and nodes 5 and 6 with label c. As labels a, b, and c are used in the solution, this solution uses 3 total labels. Another spanning tree connects nodes 2 and 3 with label a, nodes 1 and 2 with label b, nodes 1 and 5 with label b, nodes 4 and 5 with label b, and nodes 5 and 6 with label b. As this spanning tree uses only two labels, a and b, it is a superior solution to the first.

The MLST problem was motivated by problems in computer network design. Different types of media are available for computer network construction. It is often considered optimal to minimize the different types of media used within the network. In a typical residential community today, it is common to find cable, optic fiber, and telephone line all connecting computer users to the Internet. Such diversity may not be optimal in planning a computer or telecommunications network, as shown by Patterson and Rolland, 2002. In this example, each type of medium would be represented with a different label, and the MLST would be a spanning tree that uses the minimum number of different medium types. Since it is reasonable that more than one type of medium could connect the same two locations, meaning this network could not be modeled as a MLST problem instance, the GMLST problem also has applications in computer network design. The MLST problem has been shown to be NP-hard by Chang and Leu, 1997, making this problem difficult to solve to optimality in reasonable runtimes for larger datasets.

We will explore several new algorithms for finding good GMLST problem solutions for several classes of graphs and compare these algorithms with Chang and Leu's maximum vertex covering algorithm (MVCA), a benchmark heuristic, and with the modified genetic algorithm (MGA) due to Xiong et al., 2005a; Xiong et al., 2006, an effective metaheuristic for the MLST problem. In Section 2, we further describe the GMLST problem and its general solution strategies. In Section 3, we provide the details of existing MLST problem heuristics and the new algorithms we developed. In Section 4, we describe the results from each of the algorithms solving a series of test problems. Section 5 provides conclusions and directions for future research.

2. The GMLST problem

2.1 Comparing the MLST and the GMLST Problems

For 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, the subgraph induced by the label set $C \subset L$ is $G' = (V, E', C)$, with $E' = \{e \in E | l(e) \cap C \neq \emptyset\}$, where $l(e)$ is the label set associated with edge e.

Both the MLST and the GMLST problems require the search for a minimum-label set for which the induced subgraph is connected. The potential solutions for the MLST and GMLST problems can be represented by similar data structures. The structure of the problem motivates the decomposition of the problem into storing label sets as potential solutions and determining the feasibility of those solutions by checking if the subgraphs induced by the label sets on the graph considered are connected and span all nodes in V. The optimal solution would be represented by the minimum-cardinality feasible label set.

2.2 Optimal Solutions

The GMLST problem can be modeled as a mixed integer program and results can be obtained in reasonable runtimes for small graphs.[1] The integer formulation seeks to minimize the total number of labels needed subject to constraints of having a connected directed graph. Formally, let e_{ijk} be an indicator of the existence of an edge of label k between nodes i and j. Since the graphs we consider are undirected, $e_{ijk} = e_{jik}$. Let M be a very large number (for our purposes: $M = |E|$). We define x_{ijk} as a boolean number that determines whether there is a directed connection from node i to node j with label k in the solution. We use the boolean variable l_k to denote whether label k is in the solution. Finally, we obtain connectivity by defining y_i as a dummy variable associated with the node i. The GMLST problem is formulated as follows.

[1]Up to 50 nodes when using ILOG CPLEX 3.6.1

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

s.t.

$$\begin{aligned}
Connectivity: \quad & \forall i \in V, \sum_{j \in V, k \in L} x_{ijk} \geq 1 && (9.1)\\
& \forall i \in V \setminus \{1\}, j \in V : y_i - y_j + n \sum_{k \in L} x_{ijk} \leq n - 1 \\
Feasibility: \quad & \forall i, j \in V, k \in L : x_{ijk} \leq e_{ijk} && (9.2)\\
Labels: \quad & \forall k \in L : \sum_{i,j \in V} x_{ijk} \leq M * l_k && (9.3)\\
Variables: \quad & \forall i, j \in V, k \in L : x_{ijk} \in \{0, 1\} && (9.4)\\
& \forall k \in L : l_k \in \{0, 1\} && (9.5)
\end{aligned}$$

2.3 Heuristic Algorithms

Ideally, an exact method like solving the model provided with integer programming software would be desired for solving this combinatorial optimization problem because this method can return the optimal solution to any problem instance. However, exact solutions often require a prohibitively long runtime, making them impractical approaches for solving large problem instances of NP-complete problems. For instance, the backtrack search mentioned in Xiong et al., 2005a, which is an exact algorithm, could not be used to analyze datasets containing more than 50 nodes and 50 labels due to its exponential runtime. This shortcoming demonstrates the need for heuristics that return approximate results in much quicker runtimes. Examples of some heuristics for the MLST problem include the genetic algorithm due to Nummela and Julstrom, 2006, the genetic algorithm due to Xiong et al., 2005a, and the tabu search procedure due to Cerulli et al., 2005.

In this paper, we present two greedy heuristics for the GMLST problem. The first heuristic, the maximum vertex covering algorithm (MVCA) first posed in Chang and Leu, 1997, is a benchmark heuristic for the MLST problem. We show the MVCA can be used without modification on the GMLST problem in Section 3. In addition to this benchmark heuristic, we propose a new heuristic algorithm, the rarest insertion (RI) heuristic.

In addition to the greedy heuristics, we present a number of metaheuristics. These metaheuristics allow for randomness. This allows these algorithms to search more of the solution space. We present the Modified Genetic Algorithm (MGA), a heuristic proposed in Xiong et al., 2005a. We also propose two new heuristics: the Increasing Diverse Population Genetic Algorithm (IDP) and the Iterative Perturbation and Correction Heuristic (IPC). Each algorithm is detailed in full in Section 3.

3. Algorithms for the GMLST Problem

3.1 MVCA

An effective heuristic for the MLST problem, the MVCA was proposed in Chang and Leu, 1997. Bruggemann et al., 2003 proved that the MVCA can be used to provide approximate solutions to the MLST problem in polynomial time. Xiong et al., 2005b, using the harmonic numbers $H_b = \sum_{i=1}^{b} \frac{1}{i}$, where b is the maximum frequency of any label in the graph, showed that the number of labels in the MVCA heuristic solution is no worse than H_b times optimal and demonstrated that this bound is tight.

Though the MVCA was developed for the MLST problem, it can be used without modification for the GMLST problem because its data representation is a label set. The MVCA begins with an empty label set. Hence, in this initial state, the subgraph induced by the label set on a graph G has n components. Each of them is composed of a single vertex. At each stage, the MVCA chooses labels to add to the partial label set such that the number of components in the subgraph induced by the new label set is minimum. Pseudocode for the algorithm is shown below.

```
0 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 $C \leftarrow \{\}$ be the set of added labels.
2 Do while the subgraph induced by $C$ on $G$ has more than one component
    (a) minnumcomp$\leftarrow \min($ the number of components of $C \cup \{k\}$ in-
        duced on $G)$ for any $k \in L \setminus C$
    (b) possiblelbls$\leftarrow \{k \in L \setminus C$ : the number of components of $C \cup \{k\}$
        induced on $G =$ minnumcomp$\}$
    (c) Randomly select label $f$ from possiblelbls
    (d) $C \leftarrow C \cup \{f\}$
3 Report $C$
```

3.2 Rarest Insertion

We developed the Rarest Insertion (RI) algorithm to decrease the number of labels considered for removal by the MVCA. The RI heuristic maintains the components of the subgraph induced by the current label set on the graph considered. Each iteration, the algorithm selects the component that has the fewest number of labels linking its member nodes to any other node in the graph. From that set of labels, the RI heuristic selects the label that, when added to the current label set, results in the label set whose induced subgraph on the

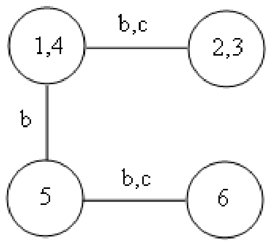

Figure 9.2. Sample graph after one iteration of the RI algorithm

graph considered has the least number of components. That label is added to the heuristic solution's label set and the components are recalculated. The RI algorithm continues adding labels in this manner until the induced subgraph is connected and spans all nodes in V.

To illustrate this procedure, consider the RI algorithm performed on the graph shown in Figure 9.1. As no labels have been selected yet, each node is its own component. Next, the rarest component (node) must be found. Node 1 is connected to other nodes by all 3 labels, node 2 is connected to other nodes by all 3 labels, node 3 is connected to other nodes by only 1 label (a), node 4 is connected to other nodes by all 3 labels, node 5 is connected to other nodes by 2 labels (b and c), and node 6 is connected to other nodes by 2 labels (b and c). Hence, the component containing node 3 is selected as the rarest component, and label a is added to the solution set. After recomputing the components, the result is the graph shown in Figure 9.2. Now, the rarest component is again selected. Each of the four components pictured is connected to other components by all 2 remaining labels (b and c), so the rarest component is selected randomly from these choices. Suppose the component containing only node 5 is selected. As it is connected to other components by both labels b and c, a decision must be made about which of these labels to use. Since adding b would result in 1 remaining component (containing a feasible spanning tree) but adding c would result in 2 remaining components (one containing nodes 1, 2, 3, and 4 and the other containing nodes 5 and 6), b is added and the procedure terminates, having found the solution set $\{a, b\}$.

Pseudocode for this algorithm is shown next.

```
0 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 $C \leftarrow \{\}$, the set of added labels.

2 $N \leftarrow \{\{1\}, \{2\}, ..., \{n\}\}$, the components of the subgraph induced by
  $C$ on $G$
```

```
3 Do while $|N| > 1$
    (a) *minnumlbl*$\leftarrow \min(|\bigcup_{a\in P,b\in V\setminus P} l(\overline{ab})|)$ for any $P \in N$, where $l(\overline{ab})$ is the label set of the arc between node $a$ and node $b$
    (b) *possiblecomp*$\leftarrow \{P \in N : |\bigcup_{a\in P,b\in V\setminus P} l(\overline{ab})| =$*minnumlbl*$\}$
    (c) Randomly select $S$ from *possiblecomp*
    (d) $T \leftarrow \bigcup_{a\in S,b\in V\setminus S} l(\overline{ab})$, the set of labels connected to the selected component $S$
    (e) *minnumcomp*$\leftarrow \min($ the number of components of the subgraph induced by $C \cup \{k\}$ on $G)$ for any $k \in T$
    (f) $possiblelblset \leftarrow \{k \in T$ : the number of components of the subgraph induced by $C \cup \{k\}$ on $G = minnumcomp\}$
    (g) Randomly select $f$ from *possiblelblset*, the selected label to be added
    (h) $C \leftarrow C \cup \{f\}$
    (i) Do while $\exists J, K \in N, J \neq K$ s.t. $\exists j \in J, k \in K$ s.t. $\{f\} \subset l(\overline{jk})$
        i $N \leftarrow N \setminus \{J, K\} \cup \{J \cup K\}$ /* $|N| \leftarrow |N| - 1$ */
4 Report $C$
```

3.3 Iterative Perturbation and Correction Heuristic

The Iterative Perturbation and Correction (IPC) heuristic is an algorithm aimed at overcoming the drawbacks of hill climbing methods. Hill climbing methods perform a local search for increases in fitness. However, they often get trapped at local optimal solutions, rarely discovering a solution near the global optimal solution. The IPC allows deteriorating moves along with iterative improvement to broaden the search space and to provide better future solutions. The final IPC solution is less dependant upon the starting point than the hill climbing final solution, making the initial solution a less important consideration in the IPC. Parameters for this metaheuristic are found in Section 4.

Algorithm Structure. To generate the initial feasible solution used in our IPC heuristic, we start with an empty label set. We then add a label using the weighted selection technique described later in this section. We keep adding unadded labels in this manner until the subgraph induced by the label set on the graph considered is connected and spans all nodes in V, meaning the solution is feasible. If this initial label set has cardinality one, we return it as the final solution. If not, one at a time we remove all labels from the label set that are not needed to maintain the connectivity of the label set induced on the graph and then add a label using the weighted selection technique described below.

The algorithm continues until the best solution found has not improved for T iterations. Pseudocode for the algorithm is found below.

```
0 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; a set $f$ of the frequencies of each label in the graph, where $f_l$ is the frequency of label $l$; a value $F$ that is the maximum frequency of any label in the graph; and parameters $T$ and $\lambda$.
1 $C \leftarrow \{\}$, the set of used labels.
2 Do while the subgraph induced by $C$ on $G$ has more than one component
    (a) $C \leftarrow C \cup \{$ a randomly selected $l \in L \setminus C\}$, with the probability of selecting $l$ as $\frac{e^{-\lambda(\frac{F-f_l}{F})}}{\sum_{k \in L \setminus C} e^{-\lambda(\frac{F-f_k}{F})}}$ for any $l \in L \setminus C$.
3 If $|C| = 1$ then return final solution $C$ and terminate
4 *numstagnant*$\leftarrow 0$, the number of iterations with no improvement in the best solution
5 Do while *numstagnant*$< T$
    (a) *possibleremove*$\leftarrow \{l \in C :$ the subgraph induced by $C \setminus \{l\}$ on $G$ is connected and spans all nodes in $V\}$ /* the set of all labels we can remove while maintaining a feasible solution */
    (b) Do while $|$textitpossibleremove$| > 0$
        i Let $rem$ be the label in *possibleremove* added to $C$ the longest time ago
        ii $C \leftarrow C \setminus \{rem\}$ /* $|C| \leftarrow |C| - 1$ */
        iii *possibleremove*$\leftarrow \{l \in C :$ the subgraph induced by $C \setminus \{l\}$ on $G$ is connected and spans all nodes in $V\}$ /* As the cardinality of $C$ decreases, the cardinality of *possibleremove* will approach 0 */
    (c) If $|C|$ is the smallest yet encountered by the heuristic, then *numstagnant*$\leftarrow 0$
    (d) $C \leftarrow C \cup \{$a randomly selected $l \in L \setminus C\}$, with the probability of selecting $l$ as $\frac{e^{-\lambda(\frac{F-f_l}{F})}}{\sum_{k \in L \setminus C} e^{-\lambda(\frac{F-f_k}{F})}}$ for any $l \in L \setminus C$
    (e) *numstagnant*$\leftarrow$ *numstagnant*$+1$
6 Report the lowest-cardinality $C$ ever encountered
```

Exponentially Weighted Selection. Selections of labels are made based on an exponential distribution, where labels with a higher frequency in the graph are given preference. A parameter λ is used to govern how strongly more frequent labels are favored. The preference given to a label $l \in L \setminus C$ is modeled by $e^{-\lambda(\frac{F-f_l}{F})}$, where f_l is the number of times a label l is present in the graph, F is the maximum frequency of any label in the graph, L is the set of all labels, and C is the current label set. The probability of a label $l \in L \setminus C$ being selected in the weighted selection is given by that label's selection preference divided by the sum of the selection preferences of all labels not in C, or $\frac{e^{-\lambda(\frac{F-f_l}{F})}}{\sum_{k \in L \setminus C} e^{-\lambda(\frac{F-f_k}{F})}}$. Clearly, if λ is set to a high value, labels with a higher frequency will gain more of an advantage in selection over those with a lower frequency, making the IPC more greedy in nature. From our data, the IPC solution quality decreased when λ was set too high or too low.

3.4 Modified Genetic Algorithm

We implemented the MGA due to Xiong et al., 2005a; Xiong et al., 2006. Though this heuristic was designed for the MLST problem, no modifications to the metaheuristic were necessary, as its data representation is a label list. This genetic algorithm uses a single parameter to set the number of generations equal to the population size. A large parameter value is associated with longer runtimes.

3.5 An Increasing Diverse Population Genetic Algorithm

The MGA sacrifices population diversity for intensive local optimization. Therefore, we were motivated to develop a genetic algorithm that involves less local optimization and a greater focus on population diversity to yield better solutions for datasets. This prompted the creation of the IDP.

The IDP stores candidate solutions in data structures called chromosomes. As suggested in Xiong et al., 2005a, we use a list of the labels to store the candidate solutions. The parameters used for this heuristic in data collection are discussed in Section 4.

Initial Chromosome Generation. The initial chromosome begins as an empty label set. An initial chromosome is generated iteratively by randomly selecting *initlabelselect* labels not in the current label set and adding the one that minimizes the number of components of the graph when all edges containing that label are added, continuing until the subgraph induced on the graph by the label set is connected and spans all nodes in V. Though for large *initlabelselect* this process requires more runtime than the random method of initial chromosome generation used in Xiong et al., 2005a, it results in a fitter initial population. If an element of the initial population has a cardinality of one, the IDP terminates after this stage.

Another initial chromosome generation technique we considered but decided not to use involved using domain-specific knowledge to create an overlap matrix to create initial chromosomes of better fitness than those produced by a fully random procedure like the one used by the MGA. The overlap matrix is a $m \times m$ matrix that stores in each entry the number of edges in the graph whose label sets contain both the label represented by the row and the label represented by the column. An initial chromosome is generated by iteratively adding the label with the least overlap with the labels already selected to be included in the chromosome and breaking ties randomly.

The procedure can be made fast by precomputing the overlap matrix. However, we noted that this method did not produce as fit initial chromosomes as other algorithms. This could be due to the fact that some overlapped edges may be counted multiple times as we build up the chromosome.

Crossover and Mutation. A key component of any genetic algorithm is the crossover operation, which combines the genetic information from two parents into a child that is similar to both. In this genetic algorithm, the crossover begins by first maintaining all the labels in the label sets of both parents. Next, it adds random unused labels from either of the parents' label sets. The crossover continues this process until the subgraph induced by the label set on the graph is connected and spans all nodes in V. A feasible label set is guaranteed to exist, since both parents are feasible solutions, meaning the union of their label sets would also induce a connected subgraph.

To maintain population diversity, *nummutate* chromosomes in the population are randomly selected each generation. Each of these chromosomes is mutated. The mutation operator is very simple: a label $l \notin C$ is selected at random and added to the label set C of the chromosome.

Local Search. As discussed in Michalewicz, 1996, genetic algorithms for network problems will often perform poorly without a unary local search procedure to iteratively improve chromosomes in a population. For IDP, a non-intensive local search procedure removes the first label it finds that can be removed from the label set while maintaining that label set's feasibility. The IDP also uses a second type of local search, an intensive local search. The intensive local search iterates through all of the labels in the label set, removing each that is not necessary for the feasibility of the label set. Because the non-intensive local search can only remove one unneeded label while the intensive local search can remove multiple labels, the non-intensive local search allows for quicker runtimes but is a weaker local search operator.

Generation Structure. Each generation, *numreplace* of the *popsize* chromosomes in a population are selected for replacement by new chromosomes. These chromosomes are selected based on a probability distribution, in which the probability of selecting a chromosome P from the population Q for replacement is $\frac{|P|-|best|+1}{\sum_{A \in Q} |A|-|best|+1}$, where *best* is the fittest chromosome in the population. Therefore, chromosomes with a larger number of associated labels, and hence a worse fitness, are more likely to be replaced, simulating an evolutionary process.

The *popsize* − *numreplace* chromosomes not selected to be replaced are maintained until the next population by replication. Next, the *numreplace* new chromosomes for the next generation are generated by the crossover operator described earlier in this section. The unique parents for each crossover operation are also generated using a probability distribution. For this distribution, the probability of selecting a chromosome P from the population Q is $\frac{|worst|-|P|+1}{\sum_{A \in Q} |worst|-|A|+1}$, where *worst* is the least fit chromosome in the population. The chromosomes with the smaller number of labels, and hence a greater fitness, are more likely to be selected as parents, again simulating an evolutionary process.

The *numreplace* new chromosomes are placed into the population for the next generation. After this replacement, mutation is carried out as described above. Next, *numlocaloptimize* random selections of chromosomes are made. Each time a chromosome is selected, local optimization is applied to it as long as the chromosome has not had exhaustive local search performed on it in previous generations, in which case no action is taken after selection. The type of local search used will be discussed during the explanation of the population structure.

Population Structure. In an effort to prevent premature population convergence, a check is done for concurrent label sets. Two label sets are defined to be concurrent if one is a subset of the other. If two chromosomes have concurrent label sets, then only one needs to be maintained in the population. If two concurrent label sets are found, then the one with fewer labels, P, is maintained, with ties broken randomly. The other chromosome is mutated *numalterations*(P) times, where *numalterations* is a function found in Section 4. Next, the non-intensive local search is performed until no more improvements can be made, but no more than *numalterations*(P) times.

Finally, to avoid having one population converge to a poor solution, *numisolated* isolated populations are maintained in a structure similar to that used in Silberholz and Golden, 2007. Each population contains *popsize* chromosomes and the populations are maintained until the best chromosome found by any of the populations has not improved for *numgensisolated* generations. To maintain shorter runtimes, the non-intensive local search is used during the local search phase for these isolated populations. After this time of isolated evolution, the isolated populations are combined into a final population of *popsize* chromosomes using a probability distribution identical to the one used for parent selection in the reproduction phase of the genetic algorithm. The single combined population is maintained until the best chromosome produced by the population has not improved for *numgensfinal* generations. The intensive local search procedure is used instead of the non-intensive version for this final single population. Additionally, since the MVCA runs so quickly, MVCA solutions are incorporated into this part of the GA. *numinitmvca* of the chromosomes in the initial combined population are generated by the MVCA heuristic, and each generation, *nummvcaeachgen* of the chromosomes are generated by the MVCA.

4. Computational Results

4.1 Small-World Dataset Generation

The first type of datafile generation technique that we used is based on the Small-World datafile generator proposed in Watts and Strogatz, 1998. Essentially, the Small-World generator works in two steps as follows: for a selected number of nodes n and density d (where $d = \frac{|E|}{|V|^2}$):

1 Create a regular lattice-like network, i.e., n nodes connected to form a circle, with each node linked to its $\frac{|E|}{|V|}$ neighbors.

2 For each arc, with probability p, rewrite the arc's end node such that the graph remains connected.

3 Repeat the second step for i iterations.

To account for arcs labels, we expanded the Watts and Strogatz algorithm by labeling each arc such that the arc has a maximum of l labels (*LabelsPerArc* $\sim$ *Uniform*$[0, l]$)

In our generator we set the rewriting probability p to be 0.5, the number of iteration $i = 3$, and the maximum labels per arc $l = 5$.[2]

4.2 TSPLib-Based Dataset Generation

In addition to creating datasets based on the Small-World datafile generator as discussed above, we also generated datasets based on the TSPLib datasets from Reinelt, 1990. In the following sections, we will describe the deterministic algorithms used to generate these datasets so that others may compare results

[2]The graph generator is available at http://www.rhsmith.umd.edu/faculty/phd/inbal/

with those presented in this paper by downloading TSPLib datasets from the Internet and implementing our labeling algorithms.[3] Our algorithms for creating GMLST problem graphs from TSPLib datasets were divided into two steps. First, we generated a frequency distribution of the number of labels between every pair of nodes. We then determined which labels would be used for each arc. For each of these steps, we generated two algorithms.

Algorithms for Generating Frequency Distributions. Algorithms for generating frequency distributions were based on three parameters. The first, *maxlabelsperarc*, is the maximum number of labels associated with an individual arc in the graph. This value was set to be 5. The next parameter, *totnumlabels*, is the sum of the number of labels associated with each arc over all arcs in the graph. This parameter was set at $\lfloor \binom{n}{2} * \mathit{density} * \mathit{maxlabelsperarc} \rfloor$, where *density* is a value that varied in experimentation between .005 and .1. The final parameter, m, or the number of labels for the dataset, was determined by the equation $m = \lfloor \frac{n^2}{100} \rfloor$, where n is the number of nodes specified in the TSPLib dataset. Since the values of m and *totnumlabels* both varied with degree 2 in relation to n, the number of labels in datasets with the same density stayed approximately constant regardless of the size of that dataset.

Two algorithms were used for generating the frequency distributions. The first, random frequency, uses the distance matrix of the TSPLib dataset to generate a distribution that is relatively pattern-free. Pseudocode for this algorithm is provided in Appendix A. The other algorithm, length-based frequency, is based on the idea that in many applications of the GMLST problem, most labels will be found on shorter arcs. For instance, in the example of a telecommunications network, it is likely that a provider would connect two distant cities through a series of shorter connections rather than a single direct connection. The algorithm assigns each arc a number of labels proportional to the inverse of the length of that arc. Pseudocode for this algorithm is provided in Appendix A.

Algorithms for Determining Labels for Each Arc. Two algorithms were developed for determining the labels for each arc in the graph, given the number of labels associated with that arc determined by one of the two algorithms presented above. The first, random label selection, again uses the distance matrix of the TSPLib dataset to make label selections that are generally pattern-free. Pseudocode for this algorithm is provided in Appendix A. The other algorithm, clustered label selection, stems from the concept of localization of services in the real-world applications of the GMLST problem. A local company would only provide services to a small geographical range, meaning its labels would touch a small number of nodes. This effect is approximated

[3] At http://www.rhsmith.umd.edu/faculty/bgolden/, the graph generator and problem instances can be found

in the clustered labels algorithm by assigning each label a central node. The algorithm then labels arcs based on proximity to node centers, finally evening the distribution so that there is less variation in the frequency of labels in the graph. Pseudocode for this algorithm is provided in Appendix A.

4.3 Parameters for Computational Experiments

Parameters for the IPC heuristic were determined based on preliminary modeling runs and were selected to be values that returned good results. The value 3000 was used for T, the number of iterations without improvement. The value 6 was used for λ, the affinity to add high-frequency labels. Preliminary testing showed that as T increased the solution quality became less sensitive to the λ value.

Parameters for the IDP were also selected based on preliminary modeling runs. Parameters for the structure of the GA were also based on suggested values from Michalewicz, 1996. The number of labels considered before choosing the most advantageous in the initial chromosome creation, *initlabelselect*, was selected to be 10. The number of chromosomes mutated in each population, *nummutate*, was selected to be 7. The size of each population, *popsize*, was set to be 40 chromosomes. The number of chromosomes to be replaced by crossover each generation, *numreplace*, was selected to be 25. The number of chromosomes selected to have local optimization performed each generation, *numlocaloptimize*, was selected to be 40. The number of times a chromosome was mutated and then locally optimized if it was found to have a concurrent label set with another chromosome, *numalterations*(P), was set to be $\lfloor \frac{|P|}{10} \rfloor + 3$, where $|P|$ is the number of labels in the candidate chromosome P. The number of isolated populations maintained at the beginning of the IDP heuristic, *numisolated*, was set to be 6. The number of generations without improvement that isolated populations are maintained, *numgensisolated*, was set to be 7. The number of generations without improvement that the final population is maintained, *numgensfinal*, was set to be 30. The number of MVCA solutions introduced to the population after the initial combination of isolated populations, *numinitmvca*, was set to be 3. Finally, the number of MVCA solutions introduced to the final population each generation, *nummvcaeachgen*, was set to be 1.

4.4 Computational Experiments

Computational experiments were performed on 52 datasets generated using the Small-World generation methods described above. We considered densities of 0.04, 0.06, 0.08, and 0.1 for datafiles with 50, 100, 200, and 400 nodes. For datafiles with 800 nodes, we considered densities of 0.04 and 0.06. For each density used, except 0.06 for 800-node problems, we tested a datafile with 10

labels, a datafile with 55 labels, and a datafile with 100 labels. For the remaining datasets, we used 10 labels only. For each dataset, 5 instances were generated with different random seeds and each of those instances was tested, with the average results reported.

Computational experiments were also performed on 64 datafil-es generated using the TSPLib-based methods described above. We chose 4 TSPLib datafil-es, $eil51$, $pr152$, $tsp225$, and $rd400$, for experimentation. Using the formula provided earlier in this section, the datafiles generated had 26, 231, 506, and 1600 labels, respectively. For datafiles generated from $eil51$, densities of 0.025, 0.05, and 0.1 were used. For datafiles generated from $pr152$ and $tsp225$, densities of 0.01, 0.025, 0.05, and 0.1 were used. Finally, for datafiles generated from $rd400$, densities of 0.005, 0.01, 0.025, 0.05, and 0.1 were used. For each density used, we generated a datafile with a length-based frequency distribution and random labeling, a datafile with a length-based frequency distribution and clustered labeling, a datafile with a random frequency distribution and random labeling, and a datafile with a random frequency distribution and clustered labeling.

Computational experiments were performed on a Systemax Venture H524 computer with 512 MB RAM and a 3.06 GHz processor using code programmed in C and C++. Computational experiments considered the MGA, the IDP, and the IPC. Additionally, we tested the repeated MVCA (RMVCA), which is the MVCA repeated 100 times with the best result returned, and the repeated RI (RRI), which is the RI procedure repeated 100 times with the best result returned. For each of the executions of these heuristics, a different random seed was used. Because both the MVCA and RI use random selection to break ties, the solutions returned by these heuristics often varied between executions. For each of the TSPLib-based datasets considered, 5 modeling runs were performed with each heuristic with a different random seed used each time. For each of the Small-World datasets considered, 5 modeling runs were performed with each heuristic on each of the 5 instances for that datafile, resulting in 25 total modeling runs performed by each heuristic for each of those datafiles. Multiple trials are necessary because every heuristic considered is nondeterministic, meaning results will vary between modeling runs. The average solutions and runtimes for the modeling runs are provided in Appendix 9.B.1 so direct runtime comparisons can be made with the results of this paper.

4.5 Results

In our computational experiments, we counted the number of datasets for which each heuristic performed better than each other heuristic. In Table 9.1, each heuristic is compared with all of the other heuristics on the smallest datasets, which are all 36 datasets with $n \leq 100$. This includes both Small-

Heuristic	IDP	RMVCA	RRI	MGA	IPC	Sum
IDP	0	23	18	16	4	61
RMVCA	0	0	5	1	0	6
RRI	0	14	0	2	0	16
MGA	1	22	16	0	1	40
IPC	5	23	17	15	0	60
Sum	6	82	56	34	5	183

Table 9.1. Comparison over 36 small dataset instances. Entries represent the number of datasets for which the algorithm in the row heading returned fewer labels on average than the algorithm in the column heading.

Heuristic	IDP	RMVCA	RRI	MGA	IPC	Sum
IDP	0	28	29	13	14	84
RMVCA	3	0	10	3	8	24
RRI	4	21	0	1	9	35
MGA	18	30	27	0	20	95
IPC	15	28	24	11	0	78
Sum	40	107	90	28	51	316

Table 9.2. Comparison over 44 medium dataset instances. Entries represent the number of datasets for which the algorithm in the row heading returned fewer labels on average than the algorithm in the column heading.

Heuristic	IDP	RMVCA	RRI	MGA	IPC	Sum
IDP	0	17	19	13	17	66
RMVCA	3	0	7	7	12	29
RRI	2	11	0	8	10	31
MGA	10	17	17	0	20	64
IPC	8	14	15	5	0	42
Sum	23	59	58	33	59	232

Table 9.3. Comparison over 36 large dataset instances. Entries represent the number of datasets for which the algorithm in the row heading returned fewer labels on average than the algorithm in the column heading.

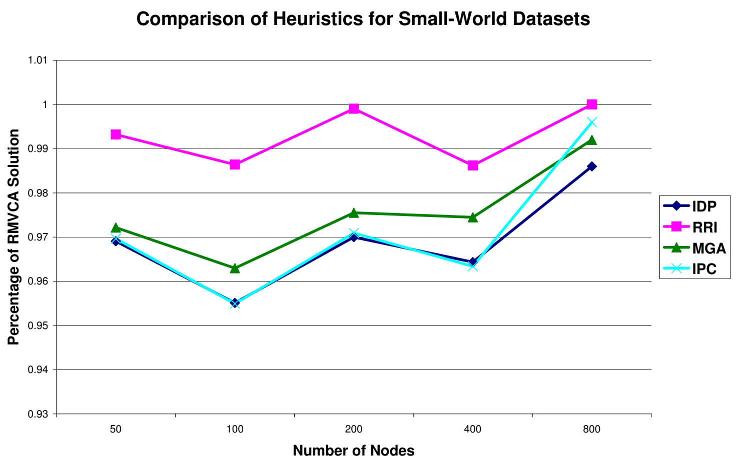

Figure 9.3. Effects of Small-World dataset size on comparative solution qualities of heuristics tested

World datasets and TSPLib-based datasets. Each entry in the table is the number of datasets considered for which the heuristic in the row label had a lower average number of labels returned than the heuristic in the column label. The column sum shows the number of times the heuristic was outperformed by any other heuristic, and the row sum is the number of times the heuristic outperformed any other heuristic. The row and column sums provide good summary statistics for how well each heuristic performed on the datasets considered. Successful heuristics had high row sums and low column sums. Table 9.2 has the same format, but considers all 44 datasets with $100 < n \leq 225$. Table 9.3 considers all 36 datasets with $n > 225$.

To illustrate the solution quality and runtime trends, we considered the effects of certain datafile attributes of the comparative solutions of the heuristics. In Figures 9.3 and 9.4 we consider how the size of the dataset affects the comparative solution qualities for both Small-World and TSPLib-based datasets. In Figure 9.5, we demonstrate how the density of a TSPLib-based dataset affects comparative solution qualities for the heuristics.

The RRI performed better than the RMVCA in modeling runs, averaging 0.79% fewer labels than the benchmark greedy heuristic. Of the 68 datafiles for which the two heuristics did not return the same average solutions, the RRI out-

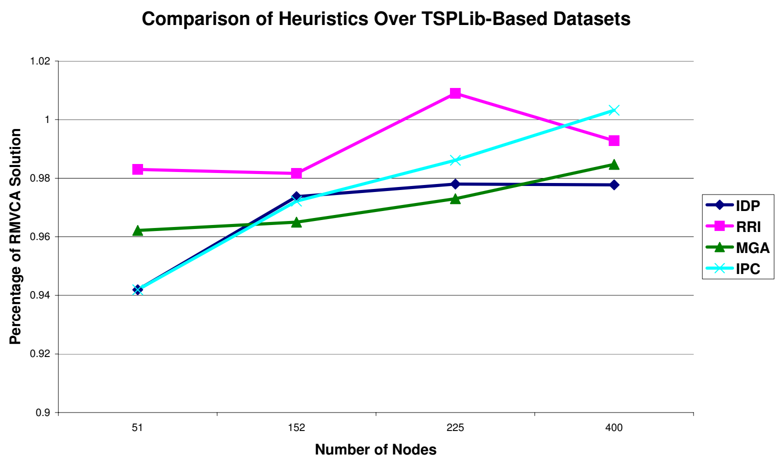

Figure 9.4. Effects of TSPLib-based dataset size on comparative solution qualities of heuristics tested

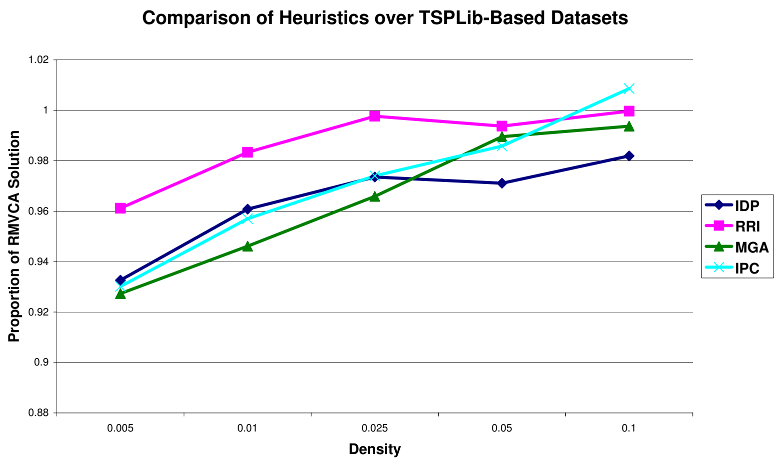

Figure 9.5. Effects of TSPLib-based dataset density on comparative solution qualities of heuristics tested

performed the RMVCA in 46 instances. The comparison between the RRI and RMVCA seems to be highly dependent on the particular dataset considered. In TSPLib-based datasets using clustered label selection, the RRI averaged 2.47% fewer labels than the RMVCA, while the heuristic actually performed worse than the RMVCA on TSPLib-based datasets using random label selection, averaging 0.91% more labels than the benchmark. This difference may be due to the fact that the RRI chooses between fewer labels than the RMVCA in an even label distribution like the one found in a dataset generated through clustered label selection, causing the RRI to make more informed label selections that help it perform better than the benchmark. In addition to outperforming the RMVCA in solution quality, the RRI performs faster than the RMVCA, completing all the modeling runs in 56.71% the runtime of the RMVCA.

The IDP and MGA consistently outperformed both of the greedy heuristics, validating both as useful metaheuristics. On average, the IDP returned solutions with 3.16% fewer labels than the RMVCA solution and 2.28% fewer labels than the RRI solution, while the MGA on average returned solutions with 2.73% fewer labels than the RMVCA solution and 1.84% fewer labels than the RRI solution. The computational results of the two metaheuristics are similar. Over all modeling runs, the IDP outperformed the MGA on 42 datasets and the MGA outperformed the IDP on 29 datasets. On average, the IDP solutions had 0.41% fewer labels than the MGA solutions. Though the IDP performed better than the MGA in solution quality comparisons, this may be due to the parameters for the IDP being set to favor solution quality over runtime. Though the IDP performed in reasonable runtimes, averaging 76.83 seconds per modeling run with the longest average runtime for a datafile slightly more than 15 minutes, the MGA executed all datasets considered in about a third the IDP runtime.

While the RMVCA and RRI solution comparison was most affected by the label selection technique for TSPLib datafiles, the comparison between the IDP and MGA is most affected by a more subtle attribute of datafiles: the cardinality of good solution sets. For this analysis, we introduce b, the best solution encountered by any of the heuristics tested for a given dataset. On datasets with low-cardinality good solutions, specifically those with $b \leq 5$, the IDP averaged solutions with 0.77% fewer labels than the MGA's results. Likewise, on datasets with medium-cardinality good solutions, specifically those with $5 < b \leq 20$, the IDP averaged solutions with 1.12% fewer labels than the MGA's results. However, on the solutions with the high-cardinality good solutions, specifically those with $b > 20$, the MGA averaged solutions with 0.93% fewer labels than the IDP's solutions. Indeed, this trend extends to comparisons with other heuristics. On datasets with medium-cardinality good solutions, the IDP dominates all other heuristics considered, averaging 3.53% fewer labels than the RMVCA, 2.67% fewer labels than the RRI, and 0.87% fewer labels than the IPC. Likewise, on datasets with high-cardinality good solutions, the

MGA dominates all other heuristics considered, averaging 3.62% fewer labels than the RMVCA, 2.64% fewer labels than the RRI, and 1.54% fewer labels than the IPC. Though this trend exists based on the cardinality of good solutions for datasets, the trend does not seem to extend to the size of datasets. The IDP actually outperformed the MGA on large datafiles, averaging 0.60% fewer labels than the MGA on 400-node TSPLib-based datasets and 1.10% fewer labels than the MGA on 400-node Small-World datasets.

Over the Small-World datasets, the IPC produced solutions of good quality. Over these datasets, the IPC and the IDP had nearly identical results, with the IPC averaging 0.09% more labels than the IDP, and the IPC outperformed all other heuristics considered in solution quality. Over the Small-World datasets, the IPC averaged 0.61% fewer labels than the MGA, 3.29% fewer labels than the RMVCA, and 2.44% fewer labels than the RRI. In runtime comparisons, the IPC was outperformed by all other heuristics and used 49% more runtime than the IDP, the closest heuristic in runtime, over all problem instances. However, the IPC runtimes were fast; the IPC averaged 8.89 seconds of runtime per Small-World dataset instance and did not average over one minute of runtime for any Small-World dataset.

On TSPLib-based datasets, the IPC performed better on datasets with a low density. With density 0.005, the IPC used 0.30% more labels than the MGA and 6.98% fewer labels than the RMVCA. However, on the highest density TSPLib graphs, with density 0.1, the IPC performed the worst of all the algorithms considered, averaging 1.46% more labels than the MGA and 0.86% more labels than the RMVCA. The IPC results seemed much less affected by the generation technique used for the TSPLib-based dataset considered. In the comparison between the IPC and MGA for different generation techniques, we averaged the number of labels for each heuristic over all datasets of that type and then compared those averages. The MGA averaged 1.5% fewer labels than the IPC on datasets with length-based frequency distribution and clustered label section, 0.8% fewer labels on datasets with length-based frequency distribution and random label selection, 1.05% fewer labels on datasets with random frequency distribution and clustered label selection, and 0.7% fewer labels on datasets with random frequency distribution and random label selection. On TSPLib-based datasets, the IPC performed better on graphs with fewer nodes. The IPC averaged 2.01% fewer labels than the MGA on TSPLib-based datasets with 51 labels, but the MGA outperformed the IPC as the number of nodes increased. For TSPLib-based datasets with 400 nodes, the MGA averaged 1.77% fewer labels than the IPC. For TSPLib-based datasets with 51 nodes, the IPC performed slower than the other heuristics, but for 400-node TSPLib-based graphs the IPC runtime was similar to that of other heuristics except for the IDP, which the IPC outperformed significantly on those datasets in terms of runtime.

From the computational results, the advantages of each type of dataset also became evident. The Small-World datasets are easier to generate, as the procedure for generation is simple compared to that of the TSPLib-based datasets. However, the TSPLib-based datasets seemed to produce more difficult datafiles. The average number of labels returned by any heuristic for the TSPLib-based datasets was 36.67, while that average for the Small-World datasets was only 5.65. Additionally, the average spread between the best and worst heuristic results for the TSPLib-based datafiles was 6.68%, while the average spread for Small-World datafiles was 4.49%. A final consideration should be that the generation of TSPLib-based datafiles requires much less runtime than the generation of a Small-World datafile.

5. Conclusions and Future Research

In this paper we presented the GMLST problem and discussed effective heuristics for generating approximate solutions to the problem. We presented the IDP, a genetic algorithm that performed as well as the MGA, a genetic algorithm in the literature for the MLST problem, and we also proposed the IPC, a fast metaheuristic that performed well in testing runs. The IPC is also a simple concept that is easy to implement, a further benefit of this approach to the GMLST problem. We also developed the RI heuristic, an effective greedy heuristic that produces solutions better than those of the MVCA, a benchmark approach for the MLST problem, in significantly less runtime.

Much can be done to extend the results published in this paper. Though the datafiles considered were interesting from a computation standpoint, creating datasets based on real-world GMLST problems would further strengthen the analysis of the heuristics presented.

Additionally, mathematical analysis of the runtime and solution quality performance of the RI heuristic is needed. For example, it might be possible to derive an upper bound on the solution error.

Finally, we must analyze how the methods we used to solve the GMLST problem perform on similar problems, like the MLST problem. It is likely that some of the methods we developed for the GMLST problem can be applied to other network optimization problems.

Appendix: A: Datafile Generation Pseudocodes

1. Datafile Generation Pseudocodes

As the pseudocode provided in this section is meant to help programmers to reimplement the dataset generation procedures described in this paper, throughout this appendix nodes and labels will be numbered between 0 and $n-1$ and between 0 and $m-1$ repectively to emulate common programming techniques.

1.1 Random Frequency Distribution

```
0 Input: A distance matrix $dist$ for TSPLib graph $G = (V, E)$, $totnumlabels$, and $maxlabelsperarc$.
1 $frequencypos \leftarrow 1$
2 for $labeladdcount = 0$ to $totnumlabels - 1$
    (a) Add a link between $a = \lfloor \frac{frequencypos}{n} \rfloor$ and $b = frequencypos \bmod n$
    (b) $frequencypos \leftarrow (frequencypos + dist_{ab} + labeladdcount) \bmod n^2$
    (c) While the arc $\overline{ab}$ associated with $frequencypos$ as calculated in step 2a already has $maxlabelsperarc$ associated with it or both nodes in that arc are the same, $frequencypos \leftarrow (frequencypos + 1) \bmod n^2$
```

1.2 Length-Based Frequency Distribution

```
0 Input: A distance matrix $dist$ for TSPLib graph $G = (V, E)$, $totnumlabels$, and $maxlabelsperarc$.
1 $currval, currfreq, node_1, node_2 \leftarrow 0$
2 for $edgecount = 0$ to $\binom{n}{2} - 1$
    (a) $node_2 \leftarrow node_2 + 1$
    (b) if $node_2 = n$ then
        i $node_1 \leftarrow node_1 + 1$
        ii $node_2 \leftarrow node_1 + 1$
    (c) $freqarray_{edgecount} \leftarrow currval + \frac{1}{dist_{node_1 node_2} + 1}$
    (d) $currval \leftarrow freqarray_{edgecount}$
3 for $edgecount = 0$ to $\binom{n}{2} - 1$
    (a) Using $node_1, node_2$ associated with $edgecount$ as assigned in steps 2a through 2b, assign $\lfloor \frac{freqarray_{edgecount} * totnumlabels}{currval} \rfloor - currfreq$ labels to the arc between $node_1$ and $node_2$
    (b) if more than $maxlabelsperarc$ were assigned, assign $maxlabelsperarc$ instead
    (c) $currfreq \leftarrow currfreq$ + the number of labels just added
4 $edgecount \leftarrow 0$
5 do while $currfreq < totnumlabels$, meaning not all labels have been distributed
    (a) using $node_1, node_2$ associated with $edgecount$ as assigned in steps 2a through 2b, increase the number of labels between $node_1$ and $node_2$ up to either $maxlabelsperarc$ or the $totnumlabels - currfreq$ + the previous number of labels assigned to the arc, whichever is less
    (b) $currfreq \leftarrow currfreq$ + the number of labels just added
    (c) $edgecount \leftarrow edgecount + 1$
```

1.3 Random Label Selection

```
0 Input: A distance matrix $dist$ for TSPLib graph $G = (V, E)$, a matrix $numlabels$ of the
  number of labels for any given arc.
1 for $node_1 = 0$ to $n - 1$, $node_2 = node_1 + 1$ to $n - 1$
    (a) for $labelcount = 0$ to $numlabels_{node_1 node_2} - 1$
        i   $currlbl \leftarrow (node_1 + node_2 + (labelcount + 1) * dist_{node_1 node_2}) \bmod m$
        ii  while the label $currlbl$ is already used between $node_1$ and $node_2$, $currlbl$
            $\leftarrow (currlbl + 1) \bmod m$
        iii assign $currlbl$ to the arc between $node_1$ and $node_2$
```

1.4 Clustered Label Selection

```
0 Input: A distance matrix $dist$ for TSPLib graph $G = (V, E)$, a matrix $numlabels$ of the
  number of labels for any given arc, $totnumlabels$ as provided to earlier algorithms
1 for $ctrcnt = 0$ to $m - 1$
    (a) $center_{ctrcnt} \leftarrow ctrcnt \bmod n$, the node used as the center of label $ctrcnt$'s cluster
    (b) $labelfreq_{ctrcnt} \leftarrow 0$, the number of label $ctrcnt$ added to the graph
2 for $node_1 = 0$ to $n - 1$, for $node_2 = 0$ to $node_1 - 1$, for $lblcnt = 1$ to $numlabels_{node_1 node_2}$
    (a) $lblselect \leftarrow -1$, the next label selected to be added ($-1$ means none yet selected)
    (b) for $lbl = 0$ to $m - 1$, if $lbl$ has not been added to the arc between $node_1$
        and $node_2$ then if $lblselect = -1$ or $dist_{node_1 center_{lbl}} + dist_{node_2 center_{lbl}} <$
        $dist_{node_1 center_{lblselect}} + dist_{node_2 center_{lblselect}}$ then $lblselect \leftarrow lbl$
    (c) assign $lblselect$ to the arc between $node_1$ and $node_2$
    (d) $labelfreq_{lblselect} \leftarrow labelfreq_{lblselect} + 1$
3 $minlbl \leftarrow \lfloor \frac{totnumlabels}{m} \rfloor$, the minimum number of arcs to be assigned to any label
4 for $node_1 = 0$ to $n - 1$, $node_2 = 0$ to $node_1 - 1$, $lbl = 0$ to $m - 1$
    (a) if $lbl$ is assigned to the arc between $node_1$ and $node_2$ and either $labelfreq_{lbl} >$
        $minlbl + 1$ or ( $labelfreq_{lbl} = minlbl + 1$ and $\exists l, 0 \leq l \leq m - 1$, s.t. $labelfreq_l <$
        $minlbl$ ) then
        i   $bestreplace \leftarrow -1$, the label to replace $lbl$ on the arc ($-1$ means none found)
        ii  for $lbl_2 = 0$ to $m - 1$, if $lbl_2$ is not on the arc between $node_1$ and $node_2$ and
            ( either $labelfreq_{lbl_2} < minlbl$ or ( $labelfreq_{lbl_2} = minlbl$ and $\forall l, 0 \leq l \leq$
            $m - 1, labelfreq_l \geq minlbl$ )) and ($bestreplace = -1$ or $dist_{node_1 center_{lbl_2}} +$
            $dist_{node_2 center_{lbl_2}} < dist_{node_1 center_{bestreplace}} + dist_{node_2 center_{bestreplace}}$)
            then $bestreplace \leftarrow lbl_2$
        iii if $bestreplace \neq -1$ then
            A $labelfreq_{lbl} \leftarrow labelfreq_{lbl} - 1$
            B $labelfreq_{bestreplace} \leftarrow labelfreq_{bestreplace} + 1$
            C remove $lbl$ from the arc between $node_1$ and $node_2$ and add $bestreplace$
              to the arc
```

Appendix: B: Detailed Computational Results

1. Detailed Computational Results

In this section, we provide the average results of each heuristic tested over all of the datasets to provide a basis for comparison with the results presented in this paper. In each of the tables, n is the number of nodes in a dataset, m is the number of labels, d is the density, and $type$ describes the type of dataset. For TSPLib-based datasets, LC implies the length-based frequency distribution was used with clustered label selection, LR means the length-based frequency distribution was used with random label selection, RC means the random frequency distribution was used with clustered label selection, and RR means the random frequency distribution was used with random label selection. A $type$ of SW means a Small-World dataset was used. Beneath each heuristic name, *lbl* is the average number of labels returned over the modeling runs for each dataset and sec is the average runtime in seconds for the modeling runs. Bolded entries indicate the best solution for a given dataset.

Table 9.B.1 displays the results of the heuristics (IDP, RMVCA, RRI, MGA, and IPC) on TSPLib-based datasets, and Table 9.B.2 displays the results of the heuristics on Small-World datasets.

References

Bruggemann, Tobia, Monnot, Jerome, and Woeginger, Gerhard J. (2003). Local search for the minimum label spanning tree problem with bounded color classes. *Operations Research Letters*, 31(3):195–201.

Cerulli, Raffaele, Fink, Andreas, Gentili, Monica, and Voss, Stefan (2005). Metaheuristics comparison for the minimum labelling spanning tree problem. In Golden, B, Raghavan, S, and Wasil, E, editors, *The Next Wave in Computing, Optimization, and Decision Technologies*, pages 93–106.

Chang, RS and Leu, Shing-Jiuan (1997). The minimum labeling spanning trees. *Information Processing Letters*, 63(5):277–282.

Michalewicz, Z (1996). *Genetic Algorithms+ Data Structures= Evolution Programs*. Springer.

Nummela, Jeremiah and Julstrom, Bryant A. (2006). An effective genetic algorithm for the minimum-label spanning tree problem. In *GECCO '06: Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation*, pages 553–558, New York, NY, USA. ACM Press.

Patterson, RA and Rolland, Erik (2002). Hybrid fiber coaxial network design. *Operations Research*, 50(3):538–551.

Reinelt, G (1990). *TSPLIB-A Traveling Salesman Problem Library*. Inst. für Mathematik.

Silberholz, John and Golden, Bruce (2007). The generalized traveling salesman problem: A new genetic algorithm approach. In Baker, E, Joseph, A, Mehrotra, A, and Trick, M, editors, *Extending the Horizons: Advances in Computing, Optimization, and Decision Technology*, pages 165–181.

Watts, Duncan J. and Strogatz, Steven H. (1998). Collective dynamics of 'small-world' networks. *Nature*, 393(6684):409–410.

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

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

Datafile				IDP		RMVCA		RRI		MGA		IPC	
d	n	m	type	lbl	sec	lbl	sec	lbl	sec	lbl	sec	lbl	sec
0.025	51	26	LC	**10**	0.85	**10**	0.05	**10**	0.07	**10**	0.14	**10**	1.03
0.025	51	26	LR	**8**	0.60	9	0.05	**8**	0.06	**8**	0.13	**8**	0.85
0.025	51	26	RC	**9**	0.91	10	0.05	10	0.07	**9**	0.15	**9**	1.00
0.025	51	26	RR	**7**	0.65	**7**	0.04	**7**	0.06	**7**	0.11	**7**	0.84
0.05	51	26	LC	**6**	0.57	7	0.05	**6**	0.05	6.8	0.11	**6**	1.03
0.05	51	26	LR	**4**	0.44	**4**	0.04	5	0.05	4.4	0.07	**4**	0.79
0.05	51	26	RC	**6**	0.58	7	0.06	7	0.05	6.2	0.12	**6**	1.14
0.05	51	26	RR	**5**	0.38	**5**	0.04	**5**	0.05	**5**	0.08	**5**	0.94
0.1	51	26	LC	**4**	0.49	5	0.05	**4**	0.05	**4**	0.09	**4**	1.32
0.1	51	26	LR	**3**	0.32	**3**	0.04	**3**	0.04	**3**	0.06	**3**	1.17
0.1	51	26	RC	**4**	0.42	**4**	0.05	**4**	0.05	**4**	0.07	**4**	1.30
0.1	51	26	RR	**3**	0.31	**3**	0.04	**3**	0.05	**3**	0.06	**3**	1.17
0.01	152	231	LC	60	27.34	63.8	3.48	60.8	2.76	**58**	15.34	59.4	16.77
0.01	152	231	LR	**33**	20.84	33.6	2.10	33.2	1.66	**33**	6.56	**33**	7.85
0.01	152	231	RC	57	27.33	61.2	3.48	58.2	2.66	**55.6**	14.90	56.8	14.16
0.01	152	231	RR	37.2	18.71	38.4	2.32	38	1.83	37.4	8.03	**37**	9.99
0.025	152	231	LC	28.4	23.36	30.2	2.67	28.6	1.55	**28.2**	7.63	**28.2**	12.28
0.025	152	231	LR	**17**	9.76	**17**	1.71	18	1.08	17.4	3.48	17.2	6.57
0.025	152	231	RC	28.4	24.05	29.4	2.74	29	1.57	**27.4**	7.99	27.6	10.31
0.025	152	231	RR	**17.6**	8.25	18	1.87	18	1.08	18	3.71	**17.6**	5.80
0.05	152	231	LC	18.2	12.46	18.4	2.49	**18**	1.16	**18**	5.49	18.2	10.12
0.05	152	231	LR	**11**	5.91	12	1.78	**11**	0.88	**11**	2.61	**11**	5.60
0.05	152	231	RC	17.4	11.59	17.2	2.66	17	1.13	**16.4**	5.12	17.4	11.03
0.05	152	231	RR	10.6	5.90	11	1.73	**10**	0.89	11	2.41	10.4	5.93
0.1	152	231	LC	11	7.48	11	2.38	11	0.92	**10.4**	3.09	10.8	10.37
0.1	152	231	LR	7	3.79	7	1.85	7	0.85	**6.6**	1.81	**6.6**	8.17
0.1	152	231	RC	10.2	6.90	**10**	2.65	10.4	0.96	10.4	3.48	11	8.80
0.1	152	231	RR	**6.8**	4.03	7	1.98	7	0.94	7	1.68	7	6.44
0.01	225	506	LC	80.4	76.31	83.8	14.84	80.6	10.04	**77.4**	46.15	78	45.95
0.01	225	506	LR	43	66.10	43.8	7.81	43.6	5.75	**42.8**	17.80	43	20.22
0.01	225	506	RC	76.6	85.52	81.6	14.93	77.6	9.68	**75.2**	44.06	75.8	36.30
0.01	225	506	RR	41	75.33	42.2	7.70	43.8	5.71	**40.6**	17.16	41.2	19.18
0.025	225	506	LC	39.2	84.69	41	11.60	40.4	5.77	**38.4**	22.44	40	23.32
0.025	225	506	LR	21.6	36.93	21.6	6.28	24	3.56	21.4	9.34	**21.2**	14.93
0.025	225	506	RC	38.6	69.64	40	11.67	39.2	5.69	**37.8**	22.60	38.4	27.07
0.025	225	506	RR	**22**	40.10	22.4	6.64	**22**	3.48	**22**	9.96	22.4	14.51
0.05	225	506	LC	24.8	35.29	24.2	10.56	24	4.08	**23.8**	15.34	24.8	25.49
0.05	225	506	LR	**13.8**	18.42	14	6.40	14	2.67	14	6.91	14.2	13.99
0.05	225	506	RC	23.8	35.06	23.8	11.03	24	4.14	**23.4**	15.23	24.2	23.74
0.05	225	506	RR	**13.6**	20.32	14	6.82	14	2.96	**13.6**	6.57	14	11.53
0.1	225	506	LC	**14.8**	24.09	**14.8**	10.16	15	3.18	**14.8**	9.57	15	23.66
0.1	225	506	LR	9	12.47	9	6.56	10	2.59	**8.8**	4.58	**8.8**	16.61
0.1	225	506	RC	**14.8**	20.02	**14.8**	11.32	15	3.49	15	9.85	15.4	25.19
0.1	225	506	RR	**8.2**	12.39	9	6.94	9	2.88	9	4.45	8.6	15.66
0.005	400	1600	LC	208	850.06	227.4	197.38	215	158.54	**205**	351.47	205.4	173.08
0.005	400	1600	LR	93.2	634.82	97.8	83.38	96	73.69	92.8	133.58	**92.6**	94.12
0.005	400	1600	RC	215.4	911.50	234.8	202.72	218.2	160.72	**214**	382.73	**214**	187.73
0.005	400	1600	RR	**97.2**	584.54	102.8	88.88	101.6	78.11	97.4	137.81	98.6	88.22
0.01	400	1600	LC	136.2	680.22	144.2	164.80	138.2	106.07	**134.2**	256.08	135	175.40
0.01	400	1600	LR	58.4	531.47	59	66.06	61.6	49.67	**57**	77.76	59.4	64.93
0.01	400	1600	RC	135.6	623.13	143.2	165.41	138	106.02	**134.2**	256.83	135.2	136.65
0.01	400	1600	RR	60.8	517.35	62.4	69.92	62.8	50.19	**60**	86.63	60.8	76.98
0.025	400	1600	LC	68.6	396.31	69	124.49	68.6	58.43	**66.2**	125.77	69	111.35
0.025	400	1600	LR	31.2	219.89	31	55.23	31.4	28.53	**30**	46.27	31.4	52.32
0.025	400	1600	RC	67.8	384.34	67.8	124.21	67.6	57.77	**67.4**	128.47	67.6	106.38
0.025	400	1600	RR	**32**	232.73	**32**	57.62	32.6	29.47	**32**	47.94	32.2	52.83
0.05	400	1600	LC	**40.8**	206.09	41	113.74	41	39.38	41.8	86.25	42	92.62
0.05	400	1600	LR	**18**	94.46	**18**	50.57	**18**	19.90	18.8	28.77	19	46.20
0.05	400	1600	RC	40.2	233.07	**40**	115.06	**40**	39.48	41	84.65	41.2	96.07
0.05	400	1600	RR	**19.2**	112.46	20	55.32	20	21.76	20	31.15	20.6	52.82
0.1	400	1600	LC	25	162.27	**24.8**	108.91	25	28.86	25.8	56.68	27.6	79.57
0.1	400	1600	LR	**11**	55.13	**11**	48.59	**11**	16.72	11.8	18.82	11.6	48.34
0.1	400	1600	RC	**24**	179.11	**24**	111.24	24.2	29.53	25.2	54.15	26.6	88.78
0.1	400	1600	RR	**12**	55.07	**12**	57.27	**12**	19.75	12.2	20.04	12.6	56.63
Average runtime				134.29 sec.		35.16 sec.		19.86 sec.		43.26 sec.		37.52 sec.	

Table 9.B.1. Heuristic results on TSPLib-based datasets. n is the number of nodes, m is the number of labels, d is the density, lbl is the average number of labels returned by a heuristic, bst is the best number of labels returned by a heuristic over the 5 modeling runs, and sec is the average runtime in seconds of a heuristic.

Datafile				IDP		RMVCA		RRI		MGA		IPC	
d	n	m	type	lbl	sec	lbl	sec	lbl	sec	lbl	sec	lbl	sec
0.04	50	10	SW	**6.4**	0.73	6.8	0.02	**6.4**	0.04	**6.4**	0.08	**6.4**	1.23
0.04	50	55	SW	**18.6**	1.44	18.76	0.13	**18.6**	0.14	18.64	0.27	**18.6**	1.68
0.04	50	100	SW	23.52	1.68	23.6	0.28	23.6	0.23	23.6	0.87	**23.48**	1.93
0.06	50	10	SW	**3.6**	0.44	**3.6**	0.01	**3.6**	0.02	**3.6**	0.05	**3.6**	0.96
0.06	50	55	SW	**10.8**	0.76	11	0.10	11.16	0.10	10.96	0.16	10.88	1.23
0.06	50	100	SW	14.32	0.98	14.52	0.20	14.4	0.16	**14.24**	0.50	**14.24**	1.37
0.08	50	10	SW	**2.6**	0.36	2.8	0.01	**2.6**	0.02	**2.6**	0.04	**2.6**	0.94
0.08	50	55	SW	**7.8**	0.54	**7.8**	0.09	7.92	0.08	**7.8**	0.11	**7.8**	1.02
0.08	50	100	SW	**11**	0.80	11.4	0.17	11.2	0.14	11.12	0.39	11.08	1.26
0.1	50	10	SW	**2.2**	0.31	2.4	0.02	2.6	0.02	**2.2**	0.04	**2.2**	1.04
0.1	50	55	SW	**6.08**	0.50	6.6	0.08	6.4	0.07	6.16	0.10	**6.08**	0.98
0.1	50	100	SW	**9.4**	0.72	**9.4**	0.16	**9.4**	0.13	**9.4**	0.31	**9.4**	1.15
0.04	100	10	SW	**3**	1.22	**3**	0.05	**3**	0.05	**3**	0.31	**3**	2.08
0.04	100	55	SW	**10**	2.36	10.96	0.23	10.2	0.20	10.2	0.94	10.12	2.64
0.04	100	100	SW	**14**	3.28	14.88	0.42	14.2	0.34	14.04	1.32	14.2	3.19
0.06	100	10	SW	**2.4**	0.96	2.6	0.05	**2.4**	0.05	**2.4**	0.29	**2.4**	2.52
0.06	100	55	SW	**7**	1.66	7.4	0.23	7.8	0.18	7.08	0.72	**7**	2.35
0.06	100	100	SW	**10.08**	2.13	10.4	0.40	10.24	0.28	10.16	1.00	**10.08**	2.67
0.08	100	10	SW	**2**	0.93	**2**	0.05	**2**	0.06	**2**	0.26	**2**	2.92
0.08	100	55	SW	5.24	1.45	6	0.23	6	0.17	5.32	0.55	**5.2**	2.37
0.08	100	100	SW	**8**	1.89	8.2	0.40	8.12	0.27	8.04	0.85	**8**	2.63
0.1	100	10	SW	**1.8**	0.70	**1.8**	0.06	**1.8**	0.06	**1.8**	0.26	**1.8**	3.41
0.1	100	55	SW	4.64	1.23	4.8	0.23	4.8	0.17	4.76	0.51	**4.6**	2.54
0.1	100	100	SW	6.68	1.63	7	0.40	7	0.26	6.76	0.73	**6.6**	2.71
0.04	200	10	SW	**2**	3.51	**2**	0.19	**2**	0.16	**2**	0.95	**2**	6.15
0.04	200	55	SW	**6.6**	5.49	7	0.81	6.8	0.43	6.76	2.53	6.68	5.70
0.04	200	100	SW	**10.04**	8.14	10.6	1.39	10.4	0.67	10.2	3.74	10.16	6.45
0.06	200	10	SW	**1.8**	2.36	**1.8**	0.20	**1.8**	0.20	**1.8**	0.92	**1.8**	8.57
0.06	200	55	SW	**5**	4.37	5.2	0.86	**5**	0.46	**5**	1.96	**5**	5.85
0.06	200	100	SW	7.04	6.24	7.4	1.41	7.8	0.67	7.16	2.75	**7**	7.09
0.08	200	10	SW	**1**	0.01	**1**	0.21	**1**	0.20	**1**	0.47	**1**	5.98
0.08	200	55	SW	**4**	3.86	**4**	0.91	**4**	0.55	**4**	1.66	**4**	6.43
0.08	200	100	SW	5.96	4.79	6	1.49	6	0.71	5.96	2.28	**5.92**	6.95
0.1	200	10	SW	**1**	0.01	**1**	0.21	**1**	0.23	**1**	0.48	**1**	7.53
0.1	200	55	SW	**3**	3.42	3.4	0.93	3.6	0.63	3.04	1.39	**3**	7.53
0.1	200	100	SW	**5**	4.42	5.2	1.53	**5**	0.84	**5**	2.04	**5**	7.53
0.04	400	10	SW	**1**	0.04	**1**	0.76	**1**	0.71	**1**	1.75	**1**	13.50
0.04	400	55	SW	4.88	15.37	5	3.55	5	1.74	4.96	7.73	**4.8**	17.73
0.04	400	100	SW	**7**	22.58	**7**	5.95	7.2	2.20	**7**	10.51	7.04	18.62
0.06	400	10	SW	**1**	0.03	**1**	0.80	**1**	0.83	**1**	1.85	**1**	0.05
0.06	400	55	SW	**3.08**	12.71	3.6	3.74	3.4	2.34	3.28	6.02	3.12	23.46
0.06	400	100	SW	**5**	17.13	5.2	6.30	**5**	2.90	**5**	8.09	5.04	22.36
0.08	400	10	SW	**1**	0.03	**1**	0.81	**1**	0.90	**1**	1.84	**1**	0.06
0.08	400	55	SW	**3**	11.48	**3**	3.86	**3**	2.89	**3**	5.46	**3**	27.28
0.08	400	100	SW	**4**	15.49	**4**	6.44	**4**	3.61	**4**	6.88	**4**	27.20
0.1	400	10	SW	**1**	0.04	**1**	0.89	**1**	1.01	**1**	1.98	**1**	0.08
0.1	400	55	SW	**2**	11.25	2.2	4.05	2.2	3.24	**2**	4.08	**2**	31.68
0.1	400	100	SW	3.48	14.15	4	6.72	3.6	4.26	3.68	6.84	**3.4**	35.00
0.04	800	10	SW	**1**	0.13	**1**	3.23	**1**	3.36	**1**	7.21	**1**	0.11
0.04	800	55	SW	**3**	49.93	**3**	15.56	**3**	10.22	**3**	20.93	**3**	58.23
0.04	800	100	SW	**4.72**	65.00	5	26.83	5	13.18	4.84	32.32	4.92	58.23
0.06	800	10	SW	**1**	0.14	**1**	3.60	**1**	3.85	**1**	8.07	**1**	0.16
Average runtime				5.98 seconds		2.06 seconds		1.27 seconds		3.14 seconds		8.89 seconds	

Table 9.B.2. Heuristics results on Small-World datasets. n is the number of nodes, m is the number of labels, d is the density, lbl is the average number of labels returned by a heuristic, bst is the best number of labels returned by a heuristic over the 5 modeling runs for each of the 5 instances considered, and sec is the average runtime in seconds of a heuristic.

Xiong, Yupei, Golden, Bruce, and Wasil, Edward (2006). Improved heuristics for the minimum label spanning tree problem. *IEEE Transactions on Evolutionary Computation*, 10(1):700–703.