Joint Conference of 1996 International Computer Symposium
December 19~21, Kaohsiung, Taiwan, R.O.C.

The Minimum Labeling Spanning Trees

Ruay-Shiung Chang and Shing-Jiuan Leu

Department of Information Management
National Taiwan Institute of Technology
No. 43, Sec. 4, Keelung Road, Taipei, Taiwan, ROC
email: rschang@cs.ntit.edu.tw

November 8, 1996

Abstract

One of the fundamental problems in graph theory is to compute a minimum weight spanning tree. In this paper, a variant of spanning trees, called the minimum labeling spanning tree, is studied. The purpose is to find a spanning tree that tries to use edges that are as similar as possible. Giving each edge a label, the minimum labeling spanning tree is to find a spanning tree whose edge set consists of the smallest possible number of labels. This problem is shown to be NP-complete even for complete graphs. Two heuristic algorithms and an exact algorithm, based on $A^* - algorithm$, are presented. According to the experimental results, one of the heuristic algorithms is very effective and the exact algorithm is very efficient.

Keywords: Graph Theory, Spanning Trees, NP-Complete, Analysis of Algorithms
AMS subject classifications: 05C05, 05C85, 68Q20, 68Q25, 68R10

1 Introduction

Computing a *minimum weight spanning tree* (MWST) is one of the fundamental and classic problems in graph theory. Given an undirected graph G with nonnegative weight on each edge, the MWST of G is the tree spanning G having the minimum total edge weight among all possible spanning trees [1]. This problem and its related problems, k smallest spanning tree [5], edge update of minimum spanning tree [10], minimum diameter spanning tree [8], min-max spanning tree [2], most and least uniform spanning trees [3, 7] and so on, have been intensely studied. Minimum weight spanning trees have applications in many areas, including network design, VLSI, and geometric optimization [4, 11].

In this paper, the minimum labeling spanning tree is defined and studied. The purpose is to construct a spanning tree using edges which are as similar as possible. For example, in communication networks, there are many different types of communication medium, such as optic fiber, cable, microwave, telephone line and so on [12]. A communication node may communicate with different nodes by choosing different types of communication medium. Given a set of communication networks nodes, the problem we are interested is to find a spanning tree (a connected communication network) that uses as few types of communication lines as possible. This spanning tree will reduce the construction cost and complexity of the network. This problem can be formulated as a graph problem. Given a graph $G = (V, E)$ and a labeling function $L(e)$ for all edges $e \in E$ where vertices represent communication nodes. Edges and their labelings represent the communication lines and their types respectively. The objective is to find a spanning tree which uses the smallest number of different types of edges labels. Define L_T to be the set of different labels in edges for a spanning tree T. The *minimum labeling spanning tree* (MLST) problem is formally defined as follows.

Problem (MLST problem). Given a graph $G = (V, E)$ and a labeling function $L(e)$ for all $e \in E$, find a spanning tree T of G such that $|L_T|$ is minimized.

Reducing from the *minimum covering* problem [6], it is shown that the MLST problem is NP-complete even when restricted to complete graphs. Two heuristic algorithms and an algorithm to obtain an optimal solution are proposed. The optimal solution algorithm is based on the $A^* - algorithm$ [9] concept. Experimental results indicate that the $A^* - algorithm$ is quite efficient and one of the heuristics gives quite

good answers.

The rest of this paper are organized as follows. In Section 2, The MLST problem is shown to be NP-complete even when restricted to complete graphs. In Section 3, two heuristic algorithms for the MLST problem and one algorithm that will produce an optimal solution are proposed. Experimental results for the algorithms are given in Section 4. Finally, conclusions and some open problems are discussed in the last section.

2 NP-Complete Proof

First, a decision version, called *bounded labeling spanning tree* (BLST) problem, for MLST problem is defined and shown to be NP-complete.

Problem (BLST problem). Given a graph $G = (V, E)$ and a labeling function $L(e)$ for all $e \in E$ and a positive integers K, is there a spnning tree T for G such that $|L_T| \leq K$.

Lemma 1. BLST is NP-complete.

Proof : It is easy to see that BLST$\in$NP since a nondeterministic algorithm need only guess a subset of edges and check in polynomial time whether these edges connect all vertices and $|L_T|$ is the appropriate size.

It is proved by transforming the minimum covering problem to the BLST problem. Let $S = \{a_1, a_2, a_3, ..., a_n\}$ and $C_1 = \{a_{i_1}, a_{j_1}, a_{k_1}\}$,$C_2 = \{a_{i_2}, a_{j_2}, a_{k_2}\}$, $C_3 = \{a_{i_3}, a_{j_3}, a_{k_3}\}$,..., and $C_m = \{a_{i_m}, a_{j_m}, a_{k_m}\}$ be subsets of S. The NP-complete minimum covering problem is to find a minimum number of subsets to cover all the element in S. From S and $C_i's$, we will construct a graph $G = (V, E)$ such that there is a cover for S with $K-1$ subsets if and only if G has MLST with K labels.

The construction of G is as follows (Fig. 1). G contains the following vertices: the element nodes $\{a_1, a_2, a_3, ..., a_n\}$, the subset nodes $\{C_1, C_2, C_3, ..., C_m\}$ and a special node s. The edges set contains the special edges $\{(s, C_i) | i = 1 \text{ to } m\}$ and the covering edges $\{(C_p, a_{l_p}) | p = 1 \text{ to } m,\ l = i, j, \text{ or } k,$ where a_{l_p} is in the C_p subset $\}$. All the special edges have label s^*. The covering edges have label C_p^* for $p = 1$ to m depending on which C_p this edge connects. It is easy to see that the construction can be accomplished in polynomial time. All that remain to be shown are that G has a MLST with K labels if and only if a minimum covering of size $K-1$ exists.

First, suppose there is a spanning tree T of G such that $|L_T| = K$. First note that $s^* \in L_T$. To construct a covering from L_T, for each label C_i^* in $L_T - s^*$, includes C_i in the cover. Since T is a spanning tree, this covering will cover all elements in S as required. Therefore, if there is a spanning tree T for G with $|L_T| = K$, then there is a covering for S with size of $K-1$.

Conversely, suppose a minimum covering with K subsets exists. Construct a spanning tree T of G according to this cover as follows.

(1) For each C_i in this cover, include the edges between C_i and its covering elements into T. If an element has more than one edge, delete the edges randomly until one is left.

(2) Include all the special edges into T. It is obvious T is a spanning tree with $K+1$ labels.

Hence, minimum covering is polynomially reducible to BLST. □

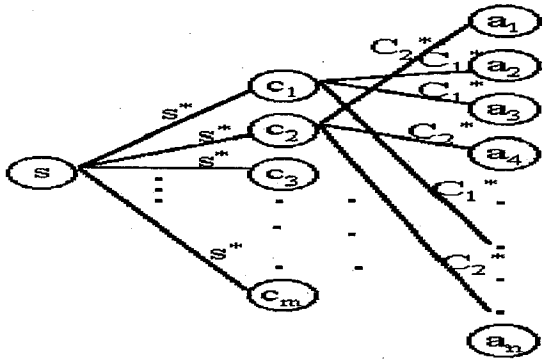


Figure 1 : *Construction of G in proof of Lemma* 1.

Corollary 2. The MLST problem is NP-complete for complete graphs.

Proof: The proof is similar to that of Lemma 1. Except in Fig. 1, all missing edges are added to make it complete. Each newly added edge is assigned a unique label. Now if a minimum covering with K subsets exists, a spanning tree with $K+1$ labels can be obtained with the same reasoning as in Lemma 1.

Conversely, suppose there is a spanning tree T with $|L_T|$ labels. If s^* is not in $|L_T|$, add (s, C_1) to T will result in a cycle. This cycle will include an edge (s, a_i) for some i, $1 \leq i \leq n$. Since the label of (s, a_i) is unique, (s, a_i) can be replaced with (s, C_1) without increasing the number of labels. After s^* is included into T, (s, C_i), $2 \leq i \leq m$, can all be added with another edge removed. For other edges in T, if it is (a_i, a_j), replace it with (C_x, a_i) or (C_y, a_j) where $a_i \in C_x$ or $a_j \in C_y$ as long as a cycle is not created. Similarly, for edge (C_i, a_j) where a_j is not in C_i, replace it with (C_x, a_j) where $a_j \in C_x$. Afterwards, by following the method in the proof of Lemma 1, a covering for S with size $K-1$ can be found. □

3 Algorithms for Minimum Labeling Spanning Tree

Given a graph $G = (V, E)$, with a label on each edge, the MLST problem is to find a spanning tree which uses the smallest number of different types of edge labels. As shown in Section 2, the MLST problem is NP-complete. In this section, two heuristic algorithms and an exact algorithm for finding an optimal solution for the MLST problem are proposed.

3.1 Heuristic Algorithms

The first heuristic algorithm is based on the edge replacement concept. At first, find an arbitrary spanning tree. Then, in order to reduce the spanning tree's total number of edge labels, for each nontree edge, test if the tree after this nontree edge is included and a tree edge deleted will have the possibility of obtaining smaller number of labelings. The detail of this heuristic algorithm is described as follows.

Algorithm 1 : Edge Replacement Algorithm

Input : A labeling graph $G = (V, E)$, with $|V| = n$, $|E| = m$ and $|L| = l$, where L is the set of possible labels for all edges.

Output : A spanning tree.

Step 1. Find an arbitrary spanning tree of G.

Step 2. For every nontree edge i

Step 3. Let $label(i)$ be the label of edge i. If $label(i)$ has notappeared in the spanning tree, go to *Step 7*.

Step 4. Find the cycle, say C, which is created by adding the edge i.

Step 5. Let *current_label_count* be the number of times that $label(i)$ appears in C.
Let *minimum_label* be the label with the least number ofappearings in C.
Let *minimum_count* be the number of appearings of*minimum_label*.

Step 6. If *current_label_count* > *minimum_count* and $label(i) = \neq$ *minimum_label*, then put edge i into spanning tree and remove a tree edge, whose label is equal to *minimum_label*

Step 7. End (for i).

Before analyzing the time complexity, we give an example. In Fig. 2-(a), an arbitrary spanning tree shown in Fig. 2-(b) is found to have 5 labels, which are a, b, c, d, and e. Next, we start trying edge replacements. First, consider the nontree edge $\{B, E\}$. $label(B, E) = a$ and a appears in labels of the spanning tree. The cycle created by adding $\{B, E\}$ is $\{B, A, E\}$. The edge $\{B, E\}$ will not be put into the spanning tree, because *current_label_count* = 1 is not greater than *minimum_count* = 1 in the cycle. Second, the nontree edge $\{C, D\}$ is tested. $label(C, D) = b$, which appears in labels of the spanning tree. The cycle created by adding $\{C, D\}$ is $\{C, B, D\}$. The edge $\{C, D\}$ will be put into the spanning tree, because *current_label_count* = 2 is greater than *minimum_count* = 1 in the cycle, where *minimum_label* = a. And the edge to be removed is $\{B, C\}$ with label a. The result is shown in Fig. 3-(a). Next, the nontree edge $\{C, E\}$ is considered. $label(C, E) = b$, which appears in labels of the spanning tree. The cycle created by adding $\{C, E\}$ is $\{C, D, B, A, E\}$. The edge $\{C, E\}$ will be put into the spanning tree, because *current_label_count* = 3 is greater than *minimum_count* = 1 in the cycle, where *minimum_label* = c. And the edge to be removed is $\{A, E\}$ with label c. The result is shown in Fig. 3-(b). Then, the nontree edge $\{D, E\}$ or $\{F, B\}$ will not change the spanning tree any more. Finally, in Fig. 3-(b), we get a spanning tree with labels $\{b, d, e\}$. The optimal spanning tree is in Fig. 3-(c) with labels $\{b, e\}$, which indicates that the algorithm may not always obtain an optimal solution.

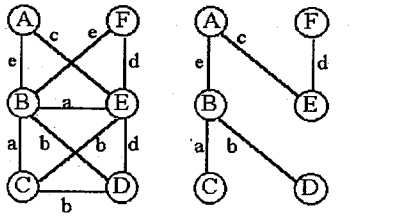


Figure 2 : *Graph G and its spanning tree.*

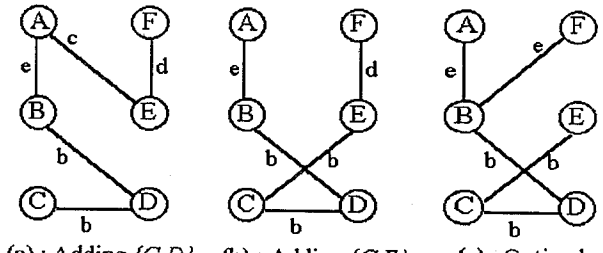


Figure 3 : *Two spanning tree of G.*

Theorem 3. The time complexity of Algorithm 1 is $O(mn)$.

Proof : At *Step 2*, there are $m - (n - 1)$ nontree edges. We then spend $O(n)$ to find a cycle at *Step 4*. Therefore, this algorithm takes $O(mn)$. □

The second heuristic algorithm tries to construct the spanning tree gradually, each time selecting a label such that edges with this label cover as many uncovered vertices as possible. We repeat this procedure until all vertices are covered. The detail of this algorithm is described as follows.

Algorithm 2 : Maximum Vertex Covering Algorithm

Input : A labeling graph $G = (V, E)$, with $|V| = n$, $|E| = m$ and $|L| = l$, where L is the set of possible labels for all edges.

Output : A spanning tree.

Step 1. Let $H = (V, \phi)$ be the subgraph of G, which hasn't any edge.

Step 2. While H is not connected

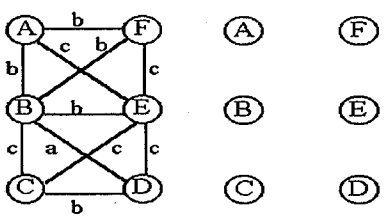
Step 3. Find a unused label l such that edges with label l cover as many uncovered vertices as possible. If there are more than one candidate, select one randomly.

Step 4. Add edges whose labels are l into the subgraph H.

Step 5. End (for *while*).

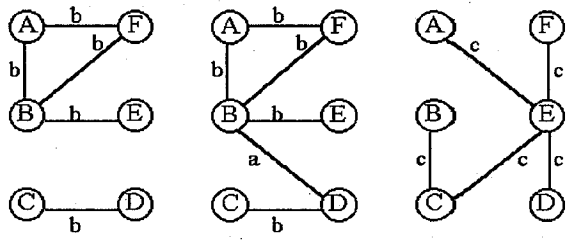
Step 6. Find an arbitrary spanning tree of H.

To give an example, let Fig. 4-(a) be the input graph. Initially, the subgraph H, shown in Fig. 4-(b), contains all vertices of G, but no edges. First, since edges with label b and label c cover the largest number of vertices, which are 6, we select label b. $\{A, B, C, D, E, F\}$ are covered. Adding those edges with label b, which are $\{A, B\}$, $\{A, F\}$ $\{B, E\}$ $\{B, F\}$ and $\{C, D\}$, into H, we get the new subgraph in Fig. 5-(a). Next, since all vertices are covered but H is not connected, we select one randomly from $\{a, c\}$. Assume a is selected. Adding the edge with label a, which is $\{B, D\}$, into H, we get the new subgraph in Fig. 5-(b). From this subgraph, a spanning tree with labels $\{a, b\}$ can be obtained, but the optimal solution is a spanning with one label $\{c\}$ in Fig. 5-(c).



(a) : Graph G (b) : The initial subgraph H

Figure 4 : *Graph G and its subgraph H.*



(a) :Adding lebel $\{b\}$ (b) :Adding label $\{a\}$ (c) :Optimal

Figure 5 : *After selecting* $\{b, a\}$ *and the Optimal.*

Please also note that the solution obtained may not be optimal.

Theorem 4. The time complexity of Agorithm 2 is $O(lmn)$, where l is the total number of different labels.

Proof: At most, the *while* loop will take $O(n)$ times. We spend $O(lm)$ to find the maximum covering label at *Step 3*. Since *Step 2* is the dominating step, the time complexity of this algorithm is $O(lmn)$. □

3.2 An Optimal Solution Algorithm

The algorithm is based on the $A^* - algorithm$ [9]. Basically, an $A^* - algorithm$ is a tree-searching algorithm. It always selects a least cost node to expand for minimization problems. We first explain how to evaluate the node-cost-estimation function f. For any node x, $f(x)$ consists of two parts, $g(x)$ and $h(x)$, where $g(x)$ = the cost of the current search path from root node r to x with $g(r) = 0$ and $h(x)$ = an estimate of the cost of the best path from x to a goal node, where $h(goal\ node) = 0$.

$A^* - algorithm$ states that if $h(x)$ is an underestimate, then the first goal node reached will be an optimal solution. In our problem, $g(x)$ is just the number of labels that have been used so far. $h(x)$ is calculated as follows.

(1) Let $l_1, l_2, ..., l_k$ be the unused labels and e_i be the number of edges with label l_i for $1 \leq i \leq k$. Without loss of generality, assume $e_1 \geq e_2 \geq ... \geq e_k$.

(2) Let $H = (V_c, E_c)$, where E_c consists of the edges whose labels have been selected so far and V_c consists of the vertices covered by edges in E_c.

(3) Let H_1 be the subgraph of H with an arbitrary edge of each fundamental cycle in H removed. (H_1 is a forest.)

(4) Let $edge\_needed = (n-1) - (\#\ of\ edges\ in\ H_1)$.

(5) $h(x)$ = the smallest j such that $\sum_{i=1}^{j} e_i \geq edge\_needed$.

It is obvious that $h(x)$ is an underestimate and can be computed in polynomial time.

Having described the cost function, we next present the $A^* - algorithm$.

Algorithm 3 : An Exact Algorithm for the MLST Problem

Input : A graph $G = (V, E)$ where each edge has a label in L and $|V| = n$, $|E| = m$, and $|L| = l$.

Output : A spanning tree with minimum number of tree edge labels.

Step 1. Put the root node r on OPEN. /* OPEN is the storage place for all generated but unexpanded nodes. */

Step 2. If OPEN is empty, exit with failure. /* Since a solution always exists, this will never be executed. */

Step 3. Remove from OPEN and place on CLOSED a node n for which f is minimum. If there are more than one node with the same minimum f value, the latest generated node will be selected. /*CLOSED is the storage place for the expanded nodes. */

Step 4. If n is a goal node (a spanning subgraph is formed), goto *Step 8.*

Step 5. Otherwise expand n. If there are k unselected labels, then n has k children, one for each unselected label.

Step 6. For each child n' of n:
If n' is not already on OPEN or CLOSE, calculate $h(n')$ and $f(n') = g(n') + h(n')$ where $g(n') = g(n) + 1$ and $g(r) = 0$. Put n into OPEN.

Step 7. Goto *Step 2.*

Step 8. Find a spanning tree of the subgraph.

Step 9. End.

Applying the algorithm to the graph in Fig. 2-(a), the whole expanded tree is as shown in Fig. 6.

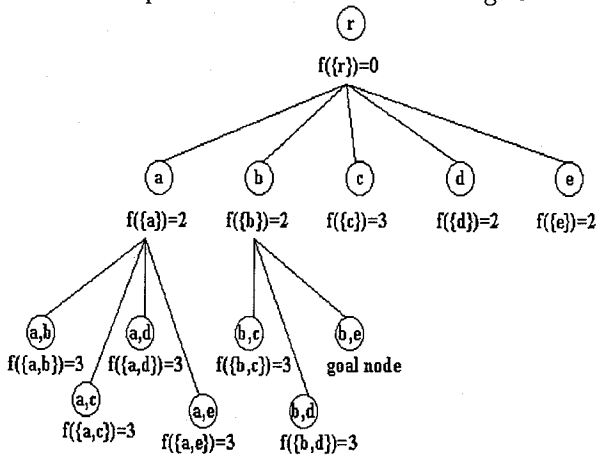


Figure 6 : *The whole tree searched by $A^* - algorithm$ for graph in Fig. 5(a).*

Let's take node (b) below root as an example. It means that all edges with label b are included into the solution set. Since one label b is used, $g(\{b\}) = 1$. The subgraph formed by edges with label b is shown in Fig. 7. Therefore, 2 more edges are needed to construct a spanning tree. Since there are two edges with the unused label e, we have $h(\{b\}) = 1$ and $f(\{b\}) = 2$

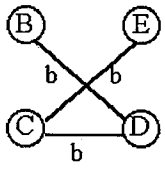


Figure 7 : *The subgraph of label b.*

Similarly for node (c) below root, $g(\{c\}) = 1$. The subgraph formed by edges with label c is shown in Fig 8.

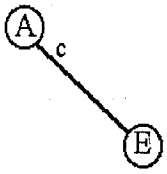


Figure 8 : *The subgraph of label c.*

Therefore, 4 more edges are needed. Since there are 3 edges with label b and 2 edges with label e, we have $h(\{c\}) = 2$ and $f(\{c\}) = g(\{c\}) + h(\{c\}) = 3$.

To test the effectiveness of the heuristic algorithms and the efficiency of the exact algorithm, we have implemented those three algorithms. The test results are shown in the next section.

4 Experimental Results

In this section, we study the performance of the proposed heuristic algorithms, Algorithm 1 and Algorithm 2, and the MLST algorithm, Algorithm 3. These algorithms are implemented using the C language and run in SUN $sparc - 20$ machine. The graph $G = (V, D, L)$ used in simulations has three parameters. V is the set of vertices with $|V| = n$, D is the density of this graph used in generating edges in the graph, and L is the set of labels with $|L| = l$.

The algorithm to generate a graph $G = (V, D, L)$ is described as follows:

Algorithm 4 : A Graph Generating Algorithm

Input : For a graph $G = (V, D, L)$, $|V| = n$ is the number of vertices, D is the density; and $|L| = l$ is the number of labels.

Output : A graph's adjacency matrix

Step 1. Let $edge = n(n-1)/2$ be the total number of edges of the complete graph, and the array $str[n][n]$ be the graph's adja- cency matrix.

Step 2. for $i = 0$ to $n-1$.

Step 3. for $j = i+1$ to $n-1$.

Step 4. Let $random\_number = rand()\ mod\ edge$, and $label = (rand()\ mod\ l) + 1$. /* $rand()\ mod\ l$ creates an integer between 0 and $l-1$ */

Step 5. if $float(random\_number/edge) < D$, then $str[i][j] = label$ and $str[j][i] = label$.

Step 6. else $str[i][j] = 0$ and $str[j][i] = 0$.

Figures 9 and 10 show the performance of the exact MSLT algorithm, Algorithm 5. Figure 12 shows the relation of the CPU time used in seconds for various number of vertices for small density graphs. The number of labels is equal to the number of vertices. We also draw the curve $y = 0.1n^2$ for comparison. Figure 10 shows the same simulation for large density graphs. The graph with density=1 is a complete graph. Those curves are always under the curve $y = 0.1n + 15$ and the CPU time needed is never more than 30 seconds.

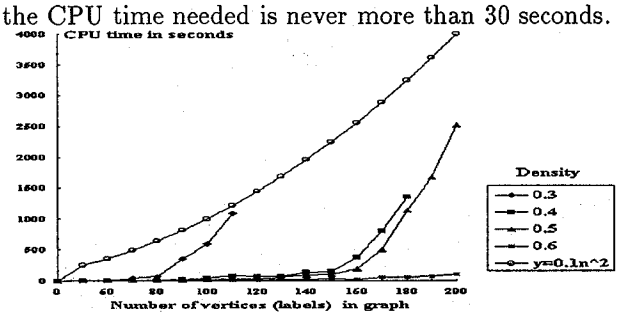


Figure 9 : *The MLST algorithm's CPU time used in the graph with density 0.3, 0.4, 0.5 and 0.6., and the curve* $y = 0.1n^2$

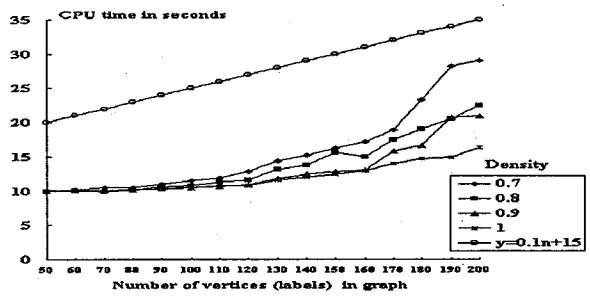


Figure 10 : *The MLST algorithm's CPU time needed in the graph with density 0.7, 0.8, 0.9 and 1 and the curve of* $y = 0.1n + 15$.

The next eight figures show the performance of Algorithms 1, 2 and 3. In those figures Algorithm 1 is dubbed as "Heu1", Algorithm 2 as "Heu2" and the exact MLST algorithm as "Opt". The density in this simulation is from 0.3 to 1 and the number of vertices are from 50 to 200. Because of the memory constraint, the exact algorithm doesn't run over 110 vertices (labels) at density 0.3 and not over 180 vertices (labels) at density 0.4. From those figures, it can be seen that the solution of Algorithm 2 is getting closer to that of the exact algorithm when the density gets larger.

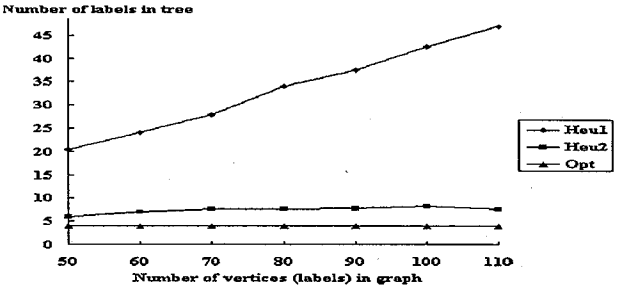


Figure 11 : *Performance comparison with graphs of density 0.3.*

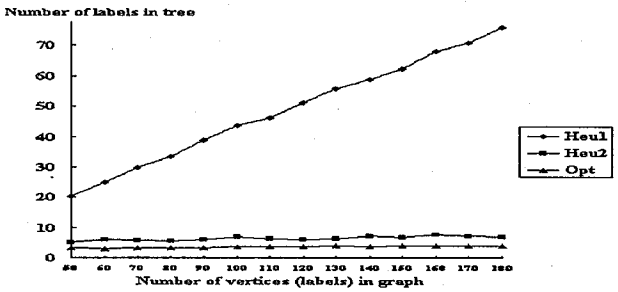


Figure 12 : *Performance comparison with graphs of density 0.4.*

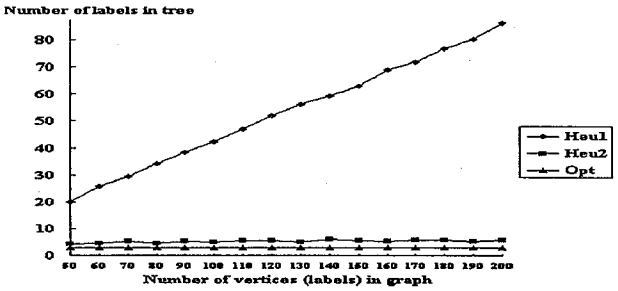


Figure 13 : *Performance comparison with graphs of density 0.5.*

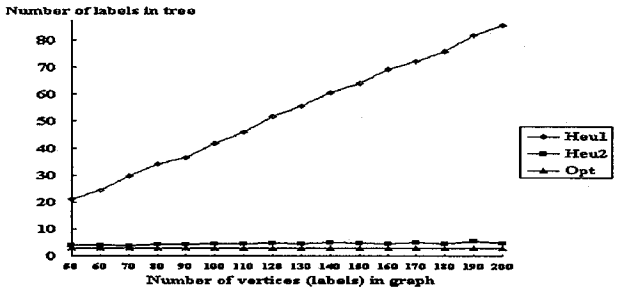


Figure 14 : *Performance comparison with graphs of density 0.6.*

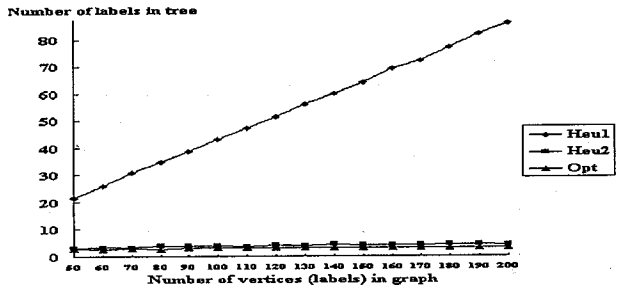


Figure 15 : *Performance comparison with graphs of density 0.7.*

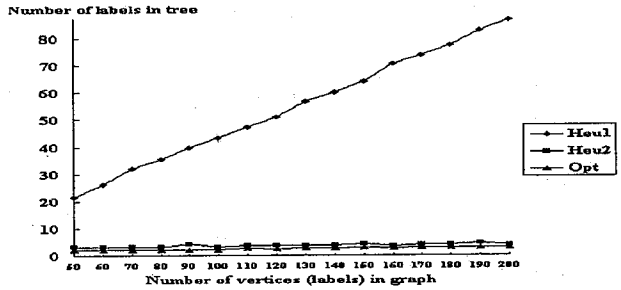


Figure 16 : *Performance comparison with graphs of density 0.8.*

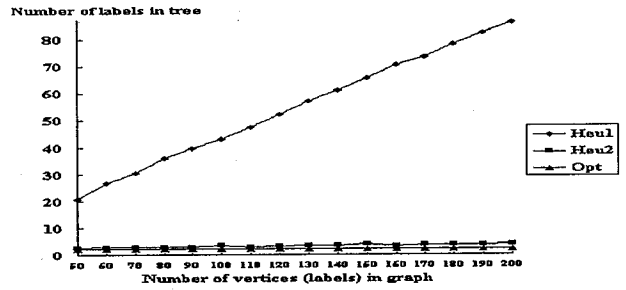


Figure 17 : *Performance comparison with graphs of density 0.9.*

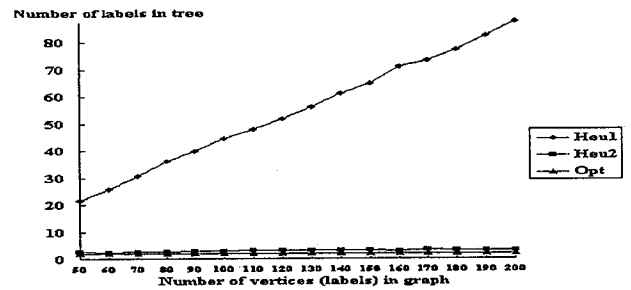


Figure 18 : *Performance comparison with graphs of density 1.*

Figures 19 and 20 show the effect of changing the number of labels to choose in the 80 vertices graph with density 0.3 and 0.7 respectively.

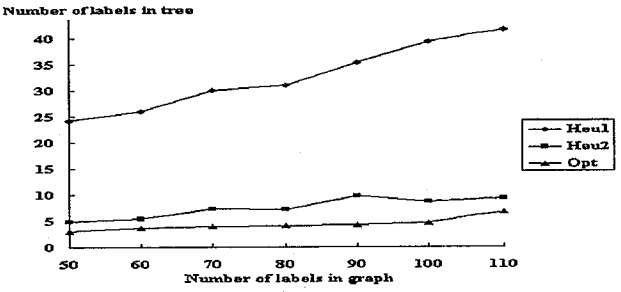


Figure 19 : *Performance comparison with different number of labels under a graph with 80 vertices, density 0.3.*

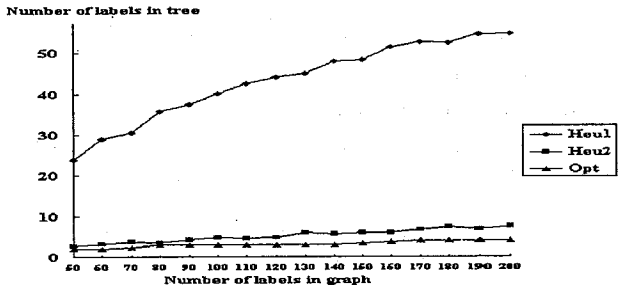


Figure 20 : *Performance comparison with different number of labels under a graph with 80 vertices, density 0.7.*

Figures 21 and 22 show the effect of changing the number of vertices in graph with 80 labels to choose and with density 0.3 and 0.7 respectively.

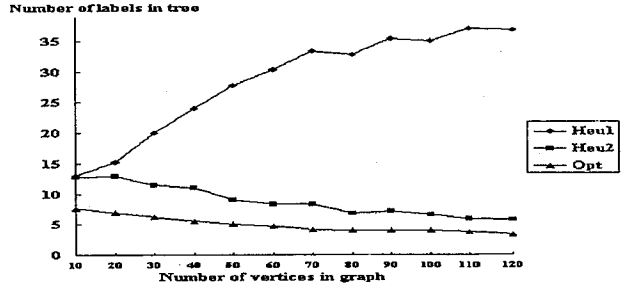


figure 21 : *Performance comparison with different number of vertices under a graph with 80 labels to choose, density 0.3.*

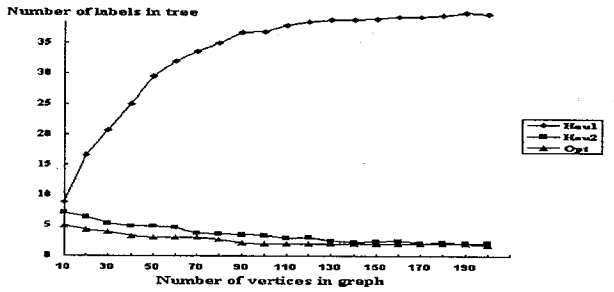


Figure 22 : *Performance comparison with different number of vertices under a graph with 80 labels to choose, density 0.7.*

5 Conclusion

The concept of the minimum labeling spanning tree is to find a spanning tree such that the spanning tree edges are as similar as possible. It is shown that the MLST problem is NP-complete, and two heuristic algorithms and an exact algorithm based on $A^* - algorithm$ are proposed. Possible future researches are to apply the minimum labelings definitions to other weighted optimization problems. For example, the minimum labeling shortest path problem, the minimum labeling maximum matching problem, the minium labeling cut set problem and so on.

References

[1] Brassard, Billes and Paul Bratley, *Algorithmics Theory and Practice*, Prentice-Hall, New Jersey, 1988.

[2] Camerini, P.M., The min-max spanning tree problem and some extensions, *Information Processing Letters*, vol. 7, no. 1, pp. 10-14, 1978.

[3] Camerini, P.M., F. Maffioli, S. Martello, and P. Toth, Most and least uniform spanning trees, *Discrete Applied Math.*, 15 (1986), pp. 181-197.

[4] Chen, Wen-Tsuen and Nen-Fu Huang, The strongly connecting problem on multihop packet radio networks, *IEEE Trans. Commun.*, vol. 37, no. 3, pp. 293-295.

[5] Eppstein, David Finding the k smallest spanning trees, *Bit*, 32 (1992), pp. 237-248.

[6] Garey, Michael R. and David S. Johnson, *Computer and Intractability A Guide to the Theory of NP-Completeness*, Freeman, San Francisco, 1979.

[7] Galil, Z. and B. Schieber, On finding most uniform spanning trees, *Discrete Applied Math.*, 20 (1988), pp. 173-175.

[8] Ho, Jan-Hing, D. T. Lee, Chia-Hsiang Chang, and C. K. Wong, Minimum diameter spanning trees and related problems, *SIAM J. Comput.*, 20 (1991), pp. 987-997.

[9] Nillson, Nils J., *Principles of Artificial Intelligence*, Tioga Publishing company, California, 1981.

[10] Shen, Xiaojun and Weifa Liang, A parallel algorithm for multiple edge undates of minimum spanning trees, *Proceedings of Seventh International Parallel Processing Symposium 1993*, pp. 310-317.

[11] Simha, Rahul and Bhagirath Norahari, Single path routing with delay considerations, *Computer Network and ISDN System,*, 24 (1992) pp. 405-419.

[12] Tanenbeum Andrew S., *Computer Networks*, Prentice-Hall, 1989.