A Tabu Search Based Approximate Solution Algorithm for $k$-minimum Spanning Tree Problems

Hideki Katagiri, Ichiro Nishizaki, Tomohiro Hayashida and Jun Ishimatsu

Abstract—This paper considers $k$-minimum spanning tree problems. An existing solution algorithm based on tabu search includes an iterative procedure of applying an exact solution method to obtain a local optimal solution. This article provides a new tabu search based approximate solution method that does not iteratively solve minimum spanning tree problems. Results of numerical experiments show that the proposed method provides a good performance in terms of accuracy over those of existing methods.

Keywords: $k$-minimum spanning tree, tabu search, $k$-cardinality, approximation solution method

1 Introduction

A $k$-minimum spanning tree ($k$-MST) problem is one of combinatorial optimization problems formulated in networks, and the objective of the problem is to find a subtree with exactly $k$ edges, called $k$-subtree, such that the sum of the weights attached to edges is minimal. The $k$-MST problem is a generalized version of MST problems: when $k = |V| - 1$ where $V$ is a cardinality of vertices in a graph, $k$-MST problem corresponds to the MST problem. The wide varieties of decision making problems in the real world can be formulated as $k$-MST problems, and many articles discuss the applications of $k$-MST problems to various decision making problems: telecommunications [8], facility layout [6], open pit mining [17], oil-field leasing [10], matrix decomposition [2, 3] and quorum-cast routing [4].

The $k$-MST problem was firstly introduced by Hamacher et al. [10] in 1991. Since the $k$-MST problem is NP-hard [5, 14], it is difficult to solve large-scale problems within a practically acceptable time. Therefore, it is very important to construct solving methods which quickly obtain a near optimal solution.

As for existing approximate solution methods for $k$-MST problems, Blum et al. [1] proposed several metaheuris-

tic approaches which includes evolutionary computation, ant colony optimization and tabu search. Urošević et al. [18] provided Variable Neighborhood Search (VNS). Recently, Katagiri et al. [12] developed a solution method which uses a combination of tabu search and exact solution method. They showed that their method provides a better performance than existing methods for dense graphs with high cardinality $k$ through some numerical experiments.

In this paper, we propose a new tabu search based approximate solution method which is combined use of tabu search and exact solution algorithm for local search. Our local search algorithm obtains local optimal solutions of $k$-MST problem without solving MST problems iteratively unlike the existing method by Katagiri et al. In order to demonstrate efficiency of the proposed solution method, we compare the performances of the proposed method with those of several existing methods.

2 Problem formulation

Given that a graph $G = (V, E)$ where $V$ is a set of vertices and $E$ is a set of edges, $k$-subtree $T_k$ is defined as

$$T_k \in G, \ k \leq |V| - 1.$$ 

Then a $k$-minimum spanning tree problem is formulated as

$$\text{minimize } \sum_{e \in E(T_k)} w(e)$$

subject to

$$T_k \in T_k,$$

where $T_k$ is the set of all $k$-subtree $T_k$ in $G$, $E(T_k)$ denotes the edges of $T_k$ and $w(e)$ is a weight attached to an edge $e$. The aim of this problem is to seek a $k$-subtree with minimum sum of weights. If the problem size is small the problem can be easily solved by finding the $k$-subtree with the minimum sum of weights after enumerating all possible $k$-subtrees in a given graph.

Even if the size of problem is not so large, it can be solved by some exact solution algorithm. As for exact solution algorithms for $k$-MST problems, a branch and bound method [4] and a branch and cut algorithm [7] have been developed and implemented.
However, it has been shown that the $k$-MST problem is NP-hard even if the edge weight is in $\{1, 2, 3\}$ for all edges, or if a graph is fully connected. The problem is also NP-hard for planar graphs and also for points in the plane [14]. Therefore, it is impossible to solve large-scale problems within a practically acceptable time even if efficient exact solution methods are applied.

Therefore, it is important to construct not only exact solutions methods but also efficient approximate solution methods. Metaheuristic approaches such as genetic algorithms are useful for getting an approximate optimal solution. Blum et al. [1] proposed three metaheuristic approaches to $k$-MST problems, namely, evolutionary computation, tabu search and ant colony optimization. They compared their performances through benchmark instances [13] and showed that the performance of their metaheuristics depends not only on the instances but also on the cardinality $k$. For example, an ant colony optimization approach is the best for relatively small $k$s, whereas a tabu search approach has an advantage for large $k$s in terms of accuracy.

Recently, Katagiri et al. [12] proposed a tabu search based approximate solution method which includes a procedure of iteratively solving MST problems for variable subgraphs. They showed that their algorithm has a better performance in terms of accuracy in comparison with those of existing methods for dense graph with large $k$s.

3 Summary of tabu search

Tabu search (TS) [9] is one of metaheuristics which is the extended exploration method based on local search. In order to make this paper self-contained, this section devotes describing the outline of tabu search.

Let $x^c$ be a current solution on search space. Local search generally finds a better solution $x' \in N(x^c)$ where $N(x^c)$ is a neighborhood of $x^c$. For simplicity, suppose that $x^c$ is a local minimum solution and that the next solution $x'$ is selected as the best solution among $N(x^c)$. If the local search is applied for $x'$, then $x'$ is moved back to $x^c$ because $x^c$ is the best solution among a neighborhood $N(x^c)$. In this way, cycling among solutions easily occurs around local minima. In order to avoid such cycling, TS algorithms use a short-term memory. The short-term memory is implemented as a set of tabu lists that store solution attributes. Attributes usually refer to components of solutions, moves, or differences between two solutions. Tabu lists prevent the exploration from returning to lately visited solutions or search domain.

Aspiration criteria permit a part of moves in the tabu list to cancel any tabu status. The typical aspiration criterion is to accept a tabu move if it updates the current best objective function value. The outline of TS is as follows:

**Step 1** Generate an initial solution $x$ and initialize a set of tabu lists $TL$.

**Step 2** Find the best solution $x' \in N(x)$ such that $x' \notin TL$, and set $x := x'$.

**Step 3** Stop the algorithm if termination conditions are satisfied. If not, then update $TL$ and return to Step 2.

In Step 2, tabu lists generally memorize solution attributes. A tabu tenure, which is the period of time when each of tabu moves is in the tabu lists, affects the behaviors and performances of the algorithm.

In Step 3, it is checked whether the algorithm satisfies termination conditions. The termination condition is usually related to the iteration number of the algorithm and/or the iteration number of not updating the current best solution. If the termination conditions are not satisfied, then tabu lists $TL$ are updated; new solution attributes are added in the lists, and some solution attributes are deleted if the iteration number that the attributes are tabu move is beyond their tabu tenure.

4 Proposed algorithm

In this section, we briefly explain our tabu search based algorithm. The most important feature of the proposed algorithm is that it does not include a MST algorithm iteratively for a subgraph with exactly $k + 1$ vertices unlike the solution method by Katagiri et al. [12]. When minimum spanning tree algorithms is applied for a fixed subgraph in the original graph, the obtained solution is regarded as a local minimum of $k$-MST problem. In this sense, MST algorithms is worth using for local search. However, there are many cases where it does not need to use MST algorithms in order to find a local optimal solution. For example, when the number of candidates for deleting and adding edges is only one, applying MST algorithms is too much cost for obtaining a local minimum. Therefore, in this paper, we consider a new local search algorithm which moves the current solution to a local minimum solution without using MST algorithms.

Fig. 1 shows the flowchart of our algorithm.

Our algorithm begins at selecting a node at random. Then, continue to apply the Prim method, which is one of the exact solution algorithms for MST problems, until a $k$-subtree is constructed.

4.1 Local search

Let $V(T_k)$ be a set of vertices which compose a $k$-subtree $T_k$.

One of the most important key to success is that we efficiently construct a neighborhood of $T_k$, denoted by
Figure 1: Flowchart of our algorithm

$N(T_k)$. Since the algorithm is a little complex, we explain the algorithm together with a simple example (see Fig. 2).

The algorithm for constructing elements in neighborhood

**Step 1** For a current solution (see Fig. 1), construct $E_{\text{add}}^1(v) := \{(v, v') \in E(G) \mid v \in V(T_k), v' \notin V(T_k)\}$ and find $c_{\text{min}}^1 = \arg\min \{w(e) \mid e \in E_{\text{add}}^1(v)\}$. Construct $T_{k+1}^{\text{ex}} := (V(T_k) \cup v_{\text{add}}, E(T_k) \cup c_{\text{min}}^1)$ such that $c_{\text{min}}^1 = (v, v_{\text{add}}), v \in V(T_k), v_{\text{add}} \notin V(T_k)$. Go to Step 2.

**Step 2** Update $F_{\text{add}}^1 := F_{\text{add}}^1 \setminus c_{\text{min}}^1$. If $F_{\text{add}}^1 = \{\emptyset\}$, then go to Step 4. Otherwise, find $c_{\text{min}}^1 := \arg\min \{w(e) \mid e \in E_{\text{add}}^1\}$ and update $T_{k+1}^{\text{ex}} := (V(T_k) \cup v_{\text{add}}, E(T_k) \cup c_{\text{min}}^1)$ such that $c_{\text{min}}^1 = (v, v_{\text{add}}), v \in V(T_k), v_{\text{add}} \notin V(T_k)$. Go to Step 3.

**Step 3** Let $E_{\text{loop}}$ be a set of edges which compose a loop. Find $c_{\text{max}} := \arg\max \{w(e) \mid e \in E_{\text{loop}}\}$ and update $E(T_{k+1}^{\text{ex}}) := E(T_{k+1}^{\text{ex}}) \setminus c_{\text{max}}$ (see Fig. 3). Go to Step 4.

**Step 4** Delete a node $v_{\text{del}} \in V(T_{k+1}^{\text{ex}}) \setminus v_{\text{add}}$ (see Fig. 4). Update $V(T_k^{\text{new}}) := V(T_{k+1}^{\text{ex}}) \setminus v_{\text{del}}$ and $E(T_k^{\text{new}}) := E(T_{k+1}^{\text{ex}}) \setminus \{(v, v_{\text{del}}) \mid v \in V(T_{k+1}^{\text{ex}})\}$. Go to Step 5.

**Step 5** Construct $E_{\text{add}}^2 := \{(v_i, v_j) \in E(G) \mid v_i, v_j \in V(T_k^{\text{new}}), (v_i, v_j) \notin E(T_k^{\text{new}}), i \neq j\}$, and go to Step 6.

**Step 6** Find $c_{\text{min}}^2 := \arg\min \{w(e) \mid e \in E_{\text{add}}^2\}$ to the current solution. Update $E(T_k^{\text{new}}) := E(T_k^{\text{new}}) \cup c_{\text{min}}^2$ and $E_{\text{add}} := E_{\text{add}} \setminus c_{\text{min}}^1$. Go to Step 7.

**Step 7** If $T_k^{\text{new}}$ is a tree, then terminate the algorithm. Otherwise, return to Step 6.

The local search finds a $k$-subtree whose objective function value is the best, namely,

$$T_k^{NH_{best}} = \arg\min \{f(T_k^{NH}) \mid T_k^{NH} \in N(T_k)\}.$$

If the pair of two nodes which are candidates for being exchanged are not in tabu lists, we regard such transitions as feasible transitions. Otherwise, we search $N(T_k)/T_k^{NH_{best}}$, etc. so as to find a feasible transition. This procedure is continued until a feasible transition is obtained.

**Figure 2:** Example of $|V| = 12$, $|E| = 23$, $k = 7$ (Bold lines are edges which form the current solution).

**Figure 3:** Example of deleting the maximum weighted edge $c_{\text{max}}$ in a loop $(v_{\text{add}} = v_9)$.
In this paper, we use two tabu lists InList and OutList, which keep the induces of removed edges and to store that of added edges, respectively. Tabu tenure, denoted by \( \theta \), is a period for which it forbids edges in the tabu lists from deleting or adding. At the beginning, InList and OutList are set empty. A parameter \( \theta \) is set the default value. Let \( n_{c_{\text{max}}} \) and \( \theta_{\text{inc}} \) be given parameters. If the current best solution is not updated \( n_{c_{\text{max}}} \) times, then we regards this situation as cycling and increase tabu tenure using (1).

\[
\theta \leftarrow \theta + \theta_{\text{inc}}
\]  

(1)

If the period of being tabu move is beyond \( \theta_{\text{max}} \), terminate the local search algorithm.

The tabu status of an attribute can be revoked if it leads to a solution with smaller cost than that of the best solution identified having that attribute. The aspiration level \( \gamma_{e} \) of an attribute is initially set equal to the cost of the initial solution \( T^{\text{ini}} \) if edge \( e \) belongs to this solution, and to \( \infty \) otherwise. At every iteration, the aspiration level of each attribute \( e \in E(T_k) \) of the current solution is updated to \( \min\{\gamma_{e}, f(T_k)\} \), where \( f(T_k) \) stands for the cost value of solution \( T_k \).

4.2 Diversification strategy and stopping condition

A diversification procedure, using the residence frequency memory function, will lead to the exploration of region of the solution space not previously visited. The residence frequency memory records the number of times a specific element has been a part of the solution.

Frequency-based memory is one of the long-term memories and consists of gathering pertinent information about the search process so far. In our algorithm, we use residence frequency memory, which keeps track of the number of iterations where vertices have been explored.

The diversification procedure begins at selecting a node at random. Then, we continue to apply the Prim method, which is one of the solution algorithm for minimum spanning tree problems, until \( k \)-subtree is obtained.

\[
w_{d}(e) \leftarrow \frac{\text{Freq}(e)}{t} \times w(e)
\]

(2)

where \( t \) is the current iteration number, and \( \text{Freq}(e) \) is the number of searching \( e \).

In this paper, we give the stopping condition as a very simple one; if the iteration number is beyond a given value, then terminate the algorithm.

5 Numerical experiments

In order to compare the performances of our method with those of representative existing solution algorithms, we solve the benchmark instances provided by Blum [13] and our own instances. Tables 1 and 2 show the results for instances by Blum [13] and our new instances, respectively.

We use C as the programming language and compiled all software with C-Compiler: Microsoft Visual C++ 7.1. All the metaheuristic approaches were tested on a PC with Celeron 3.06GHz CPU and RAM 1GB under Microsoft Windows XP. In the tables shown, RTSK, TSK and TSB represent tabu search approaches by this paper, Katagiri et al. and Blum et al., respectively. We executed each method in 30 runs and computed the best, mean and worst objective function values for each method.

Table 1 shows that the performance of the proposed method is clearly better than that of the existing method by Katagiri et al.. Also, our algorithm provides better performance than the method by Blum et al., for high cardinality \( k \), whereas the performance of the method by Blum et al. is the best for low cardinality \( k \).

Table 2 shows the results for new instances which are more dense than the existing benchmark instances. It is observed from Table 1 that our algorithm provides better performance than the methods by Katagiri et al and Blum et al. except for the best objective value of one instance.
Table 1: Experimental results for existing benchmark instances

<table>
<thead>
<tr>
<th>graph</th>
<th>$k$</th>
<th>Objective value</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>RTSK</td>
<td>TSK</td>
</tr>
<tr>
<td>$</td>
<td>N</td>
<td>= 1000$</td>
<td>200 best</td>
</tr>
<tr>
<td>$</td>
<td>E</td>
<td>= 1200$</td>
<td>mean</td>
</tr>
<tr>
<td>$d = 2.5$</td>
<td>worst</td>
<td>3933</td>
<td>4587</td>
</tr>
<tr>
<td>$\sigma(d) = 1.57$</td>
<td>400 best</td>
<td>9291</td>
<td>9936</td>
</tr>
<tr>
<td></td>
<td>mean</td>
<td>9512.7</td>
<td>10088.9</td>
</tr>
<tr>
<td></td>
<td>worst</td>
<td>9610</td>
<td>10202</td>
</tr>
<tr>
<td>$</td>
<td>N</td>
<td>= 600$</td>
<td>162333</td>
</tr>
<tr>
<td>$</td>
<td>E</td>
<td>= 1200$</td>
<td>mean</td>
</tr>
<tr>
<td>$d = 2.5$</td>
<td>worst</td>
<td>16267</td>
<td>17330</td>
</tr>
<tr>
<td>$\sigma(d) = 1.57$</td>
<td>800 best</td>
<td>26516</td>
<td>27170</td>
</tr>
<tr>
<td></td>
<td>mean</td>
<td>26525.1</td>
<td>27172.9</td>
</tr>
<tr>
<td></td>
<td>worst</td>
<td>26573</td>
<td>27173</td>
</tr>
<tr>
<td>$</td>
<td>N</td>
<td>= 400$</td>
<td>32981</td>
</tr>
<tr>
<td>$</td>
<td>E</td>
<td>= 800$</td>
<td>mean</td>
</tr>
<tr>
<td>$d = 4.00$</td>
<td>worst</td>
<td>32981</td>
<td>33459</td>
</tr>
<tr>
<td>$\sigma(d) = 0.00$</td>
<td>160 best</td>
<td>1473</td>
<td>1478</td>
</tr>
<tr>
<td></td>
<td>mean</td>
<td>1488.7</td>
<td>1562.4</td>
</tr>
<tr>
<td></td>
<td>worst</td>
<td>1512</td>
<td>1627</td>
</tr>
<tr>
<td>$</td>
<td>N</td>
<td>= 500$</td>
<td>32417</td>
</tr>
<tr>
<td>$</td>
<td>E</td>
<td>= 1000$</td>
<td>mean</td>
</tr>
<tr>
<td>$d = 10.0$</td>
<td>worst</td>
<td>3261</td>
<td>3449</td>
</tr>
<tr>
<td>$\sigma(d) = 3.22$</td>
<td>240 best</td>
<td>5212</td>
<td>5270</td>
</tr>
<tr>
<td></td>
<td>mean</td>
<td>5213.8</td>
<td>5497.5</td>
</tr>
<tr>
<td></td>
<td>worst</td>
<td>5224</td>
<td>5531</td>
</tr>
<tr>
<td>$</td>
<td>N</td>
<td>= 320$</td>
<td>7682</td>
</tr>
<tr>
<td>$</td>
<td>E</td>
<td>= 5000$</td>
<td>mean</td>
</tr>
<tr>
<td>$d = 10.0$</td>
<td>worst</td>
<td>7691</td>
<td>7719</td>
</tr>
<tr>
<td>$\sigma(d) = 3.22$</td>
<td>360 best</td>
<td>9249</td>
<td>9256</td>
</tr>
<tr>
<td></td>
<td>mean</td>
<td>9249</td>
<td>9257.8</td>
</tr>
<tr>
<td></td>
<td>worst</td>
<td>9249</td>
<td>9259</td>
</tr>
</tbody>
</table>

Table 2: Experimental results for new benchmark instances

<table>
<thead>
<tr>
<th>graph</th>
<th>$k$</th>
<th>Objective value</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>RTSK</td>
<td>TSK</td>
</tr>
<tr>
<td>$</td>
<td>N</td>
<td>= 300$</td>
<td>60 best</td>
</tr>
<tr>
<td>$</td>
<td>E</td>
<td>= 5000$</td>
<td>mean</td>
</tr>
<tr>
<td>$d = 133.3$</td>
<td>worst</td>
<td>553</td>
<td>566</td>
</tr>
<tr>
<td>$\sigma(d) = 36.57$</td>
<td>120 best</td>
<td>1218</td>
<td>1267</td>
</tr>
<tr>
<td></td>
<td>mean</td>
<td>1228.3</td>
<td>1304.4</td>
</tr>
<tr>
<td></td>
<td>worst</td>
<td>1240</td>
<td>1384</td>
</tr>
<tr>
<td>$</td>
<td>N</td>
<td>= 180$</td>
<td>21666</td>
</tr>
<tr>
<td>$</td>
<td>E</td>
<td>= 4000$</td>
<td>mean</td>
</tr>
<tr>
<td>$d = 200.06$</td>
<td>worst</td>
<td>21955</td>
<td>2264</td>
</tr>
<tr>
<td>$\sigma(d) = 38.99$</td>
<td>240 best</td>
<td>3564</td>
<td>3564</td>
</tr>
<tr>
<td></td>
<td>mean</td>
<td>3564.0</td>
<td>3568.8</td>
</tr>
<tr>
<td></td>
<td>worst</td>
<td>3564</td>
<td>3581</td>
</tr>
<tr>
<td>$</td>
<td>N</td>
<td>= 270$</td>
<td>4000</td>
</tr>
<tr>
<td>$</td>
<td>E</td>
<td>= 10000$</td>
<td>mean</td>
</tr>
<tr>
<td>$d = 200.06$</td>
<td>worst</td>
<td>4090</td>
<td>4090</td>
</tr>
<tr>
<td>$\sigma(d) = 38.99$</td>
<td>120 best</td>
<td>869</td>
<td>897</td>
</tr>
<tr>
<td></td>
<td>mean</td>
<td>875.6</td>
<td>912.4</td>
</tr>
<tr>
<td></td>
<td>worst</td>
<td>898</td>
<td>922</td>
</tr>
<tr>
<td>$</td>
<td>N</td>
<td>= 180$</td>
<td>1645</td>
</tr>
<tr>
<td>$</td>
<td>E</td>
<td>= 40000$</td>
<td>mean</td>
</tr>
<tr>
<td>$d = 206.67$</td>
<td>worst</td>
<td>1652</td>
<td>1784</td>
</tr>
<tr>
<td>$\sigma(d) = 24.61$</td>
<td>240 best</td>
<td>27377</td>
<td>27357</td>
</tr>
<tr>
<td></td>
<td>mean</td>
<td>27377.0</td>
<td>27533.0</td>
</tr>
<tr>
<td></td>
<td>worst</td>
<td>27377</td>
<td>2765</td>
</tr>
<tr>
<td>$</td>
<td>N</td>
<td>= 270$</td>
<td>3491</td>
</tr>
<tr>
<td>$</td>
<td>E</td>
<td>= 100000$</td>
<td>mean</td>
</tr>
<tr>
<td>$d = 206.67$</td>
<td>worst</td>
<td>3491</td>
<td>3491</td>
</tr>
<tr>
<td>$\sigma(d) = 24.61$</td>
<td>60 best</td>
<td>232</td>
<td>224</td>
</tr>
<tr>
<td></td>
<td>mean</td>
<td>234.4</td>
<td>238.1</td>
</tr>
<tr>
<td></td>
<td>worst</td>
<td>238</td>
<td>257</td>
</tr>
<tr>
<td>$</td>
<td>N</td>
<td>= 180$</td>
<td>541</td>
</tr>
<tr>
<td>$</td>
<td>E</td>
<td>= 100000$</td>
<td>mean</td>
</tr>
<tr>
<td>$d = 246.10$</td>
<td>worst</td>
<td>557</td>
<td>589</td>
</tr>
<tr>
<td>$\sigma(d) = 24.61$</td>
<td>240 best</td>
<td>1635</td>
<td>1656</td>
</tr>
<tr>
<td></td>
<td>mean</td>
<td>1633.9</td>
<td>1659.3</td>
</tr>
<tr>
<td></td>
<td>worst</td>
<td>1634</td>
<td>1676</td>
</tr>
<tr>
<td>$</td>
<td>N</td>
<td>= 270$</td>
<td>2106</td>
</tr>
<tr>
<td>$</td>
<td>E</td>
<td>= 1000000$</td>
<td>mean</td>
</tr>
<tr>
<td>$d = 206.67$</td>
<td>worst</td>
<td>2106</td>
<td>2109</td>
</tr>
</tbody>
</table>

6 Conclusion

In this paper, we have proposed a new solution method based on tabu search for $k$-minimum spanning tree prob-

lems and compared the performance of the proposed method with those of existing methods through numerical experiments for several benchmark instances. It has been shown that the proposed method has provided better performances than the existing methods. In the near future, we will provide additional benchmark instances such as random graphs, geometric graphs or small-world graphs, and execute more numerical experiments to clarify the advantage of our method.

References

[1] Blum, C., Blesa, M.J., "New metaheuristic approaches for the edge-weighted $k$-cardinality tree


