Proceedings of the International MultiConference of Engineers and Computer Scientists 2012 Vol II,
IMECS 2012, March 14 - 16, 2012, Hong Kong

A Hybrid Algorithm Based on Memetic Algorithm and Tabu Search for k-Minimum Spanning Tree Problems

Qingqiang Guo, Hideki Katagiri, Ichiro Nishizaki, and Tomohiro Hayashida

***Abstract*—A combinatorial optimization problem, namely *k*-Minimum Spanning Tree Problem, is to find a subtree with exactly *k* edges in an undirected graph *G* , such that the sum of edges' weights is minimal. where graph *G* is made up of a set of vertices and weight attached edges. Since this problem is NP-hard, heuristic and metaheuristic methods are widely adopted for solving large instances. In this paper, we propose a new hybrid algorithm to solve this problem, by using Memetic Algorithm as a diversification strategy for Tabu Search. Especially, the genetic operator in our Memetic Algorithm is based on dynamic programming algorithm, which finds the optimal subtree in a given tree efficiently. The proposed algorithm is tested on the *k*–Minimum Spanning Tree problems with other exiting algorithms, The experimental results show that the proposed algorithm is superior to all the exiting algorithms in precision and updates some of the best known results.**

***Index Terms*—k-minimum spanning tree, memetic algorithm, tabu search, combinatorial optimization, hybrid algorithm.**

I. Introduction

THE k-minimum spanning tree problem also referred to the k-cardinality tree problem, is a combinatorial optimization problem. Let $G = (V, E)$ be an undirected graph, which is made up by connecting a set of vertices V and edges E. For each vertex $v \in V$, it can connect to any vertex $v' \in V$, through one edge or edges. Each edge e is attached with a nonnegative value w_e, called weight. The goal is to find a tree with exactly $k(k \leq \mid V \mid -1)$ edges, so that the sum of edges' weights $f(T_k)$ is minimal. It can also be considered in another view point, that is to connect exactly $k+1$ vertices by k edges without cycle, such that the sum of edges' weights is minimal. The mathematical program is shown as below:

$$\begin{aligned} \text{minmize} \quad & f(T_k) = \sum w_e * x_e \\ \text{subject to} \quad & T_k \in \mathcal{T}_k \\ & \sum x_e = k \\ & x_e = \begin{cases} 1 & e \in E(T_k) \\ 0 & otherwise \end{cases} \end{aligned}$$

Manuscript received August 26, 2011; revised October 16, 2011.

Q. Guo is with the Department of System Cybernetics, Graduate School of Engineering, Hiroshima University e-mail:guoqingqiang@hiroshima-u.ac.jp
H. Katagiri is with the Department of System Cybernetics, Graduate School of Engineering, Hiroshima University e-mail:katagiri-h@hiroshima-u.ac.jp
I. Nishizaki is with the Department of System Cybernetics, Graduate School of Engineering, Hiroshima University e-mail:nisizaki@hiroshima-u.ac.jp
T. Hayashida is with the Department of System Cybernetics, Graduate School of Engineering, Hiroshima University e-mail:hayashida@hiroshima-u.ac.jp

where $E(T_k)$ is the edges set of tree T_k, and $\mathcal{T}_k$ is the set containing all feasible solutions with k edges in the graph G.

The k-minimum spanning tree problem was firstly raised by Hamacher $et\ al.$ [1], and has been applied very broadly, such as oil-field leasing [2], facility layout [3], open pit mining [4], matrix decomposition [5], telecommunication [6], and image processing [7].

This problem has been proved to be an *NP*-hard problem by several researchers [1] [8] [9]. In two cases the problem can be polynomially solved. One case is that there are only two distinct weights in the graph [9], the other case is that the graph is a tree itself [10].

Many solving methods were proposed for dealing with the k-minimum spanning tree problem, due to various applications of this problem. The first exact algorithm was presented by Fischetti $et\ al.$ [8]. In which, the k-minimum spanning tree problem was formulated into an integer linear program (ILP) based on generalized subtour elimination constraints. Recently, Quintao $et\ al.$ [11] also proposed two integer programming formulations, a Multiflow Formulation and a formulation based on the Miller-Tucker-Zemlin constraints, for the k-Cardinality Tree Problem. In which, the comparison of the two formulations and a Lagrangian heuristic based on the first reformulation were also implemented.

Besides the exact methods, a lot of approximation algorithms were also proposed for the k-minimum spanning tree problem. At first an $O(\sqrt{k})$-approximation algorithm for the vertex-weighted problem on grid graphs was proposed by Woeginger [12]. Later, a $2(\sqrt{k})$ approximation was obtained by Marathe $et\ al.$ [9], and then a 3-approximation algorithm for a rooted case was proposed by Garg [13]. Based on the [13], a $(2 + \varepsilon)$ approximation algorithm with complexity $n^{O(1/\varepsilon)}$ for any $\varepsilon > 0$ was obtained by Arora $et\ al.$ [14]. These papers represent ongoing improvement algorithms until a constant approximation factor could be obtained.

Since the problem is NP-hard, heuristic and metaheuristic methods were also widely proposed. Heuristics based on greedy strategy and dynamic programming were proposed by Ehrgott $et\ al.$ [15]. And a heuristic based on variable neighbourhood decomposition search (VNDS) which has a good performance for small size problems, was proposed by Urosevic $et\ al.$ [16].

Then Blum [17] proposed a dynamic programming approach for finding an optimal subtree in a graph. Firstly all the vertices in the graph is connected to be a spanning tree, then the optimal subtree is generated from the spanning tree. This algorithm was proved to be efficient even for large size problems.

ISBN: 978-988-19251-9-0
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

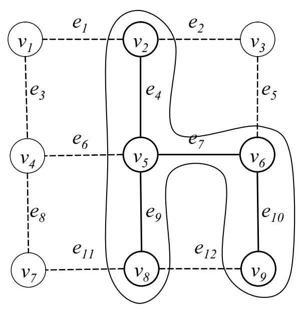

Fig. 1. A k-spanning tree (k=4, $\{v_2, v_5, v_6, v_8, v_9\}, \{e_4, e_7, e_9, e_{10}\}$)

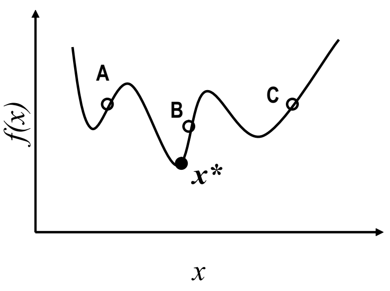

Fig. 2. A Minimization Problem

Concerning the metaheuristics, three metaheuristic approaches, namely Tabu Search, Evolutionary Computation and Ant Colony Optimization approach, for the edge-weighted k-cardinality tree problem were introduced in [18]. It also showed that the performances of three metaheuristics depend on the characteristics of the tackled instances, as well as the cardinalities. Recently a hybrid algorithm based Tabu Search and Ant Colony Optimization was proposed by Katagiri $et\ al.$ [19] and the experimental results showed that their hybrid algorithm provided a better performance with solution accuracy over existing algorithms. It can be observed that metaheuristic are attracting more and more interests recently for solving the k-minimum spanning tree problem.

The remainder of this article is organized as follows. Section II introduces preliminary background of this paper, including k-spanning tree structure, Local search, Tabu Search and Memetic Algorithm. We will describe the proposed hybrid algorithm in Section III. Our new experimental results and analysis are described in Section IV. Finally, Section V gives the conclusions.

II. Background

In this section, we start from the solution representation and then briefly describe ultimate principles of Local Search, Tabu Search and Memetic Algorithm.

A. k-spanning tree structure

Recall that a k-spanning tree is constructed by connecting $k+1$ vertices with k edges into a tree in graph G. Like other existing data structures of trees, we represent the solution of the k-minimum spanning tree problem as an ordered list of vertices and a list of edges. One example is shown in figure 1.

B. Local Search

Before proceeding to the description of metheuristic methods, we firstly focus on local search. A local search is often conducted via some move operators. To be exactly, it translates a solution s to a new one $s'(s' \in N(s))$, where $N(s)$ is a set of the neighbourhood solutions. In other words, the solution is improved by replacing it with a new one. If a movement is carried out once a new solution with a smaller objective function value is met, we call it $first\ improvement$. If a movement does not carry out until finding a solution with the smallest objective function value from its neighbourhood solutions, we call it $best\ improvement$. Though the solution can be improved based on the move operators smoothly at first, it may not reach to the best solution in most cases. The local search may repeatedly search some areas where had been searched, becoming a cycle in the worst case if it meets a local optimal solution. The search may fall into local optimum easily, in other words.

C. Tabu Search

Tabu Search, firstly proposed by Glover $et\ al.$ [22] [23], is one of the most usually used metaheuristics for solving combinatorial optimization problems. It is considered as a variety of iterative local search strategies for discrete optimization.The most important characteristic of tabu search is that it uses a concept of $memory$ to control movements via a dynamic list of forbidden moves. To be more specifically, the area that has been searched will be “tabu” (prohibited) to visit for a while. This mechanism allows Tabu Search to intensify or diversify its search procedure in order to escape from local optima. Many experiments showed that Tabu Search can lead to a significant improvement in terms of solution quality, and may even accelerate convergence of the algorithm. Tabu Search has also been proved to be effective in solving k-minimum spanning tree problem [18].

D. Memetic Algorithm

As a rising field of Evolutionary Computation, Memetic Algorithm was first introduced by Moscato in 1989 [24]. Traditional Evolutionary Computation (eg. Genetic Algorithm) have been applied widely to solve optimization problems because of their good search abilities. However, they may not be efficient to some problems which contain many local optima. For example, for a minimization problem showed in Figure 2, it seems difficult to reach the best solution x^* by Evolutionary Computation directly. But it is easy to find the best solution by local search if the search starts from solution B. Usually solution B can be easily generated by evolutionary computation operators (eg. crossover or mutation). As a matter of fact, an efficient method, calling Memetic Algorithm, is constructed by combining local search with Evolutionary Computation.

For a k-spanning tree problem with a small graph, the best solution may be found by Tabu Search easily. Because in Tabu Search, tabu list is applied to prevent falling into local optima, However, for the problem with a large graph, the length of tabu list may become very long in order to enlarge the search area. Accordingly, the computing cost would be expended rapidly with the increasing of the length.

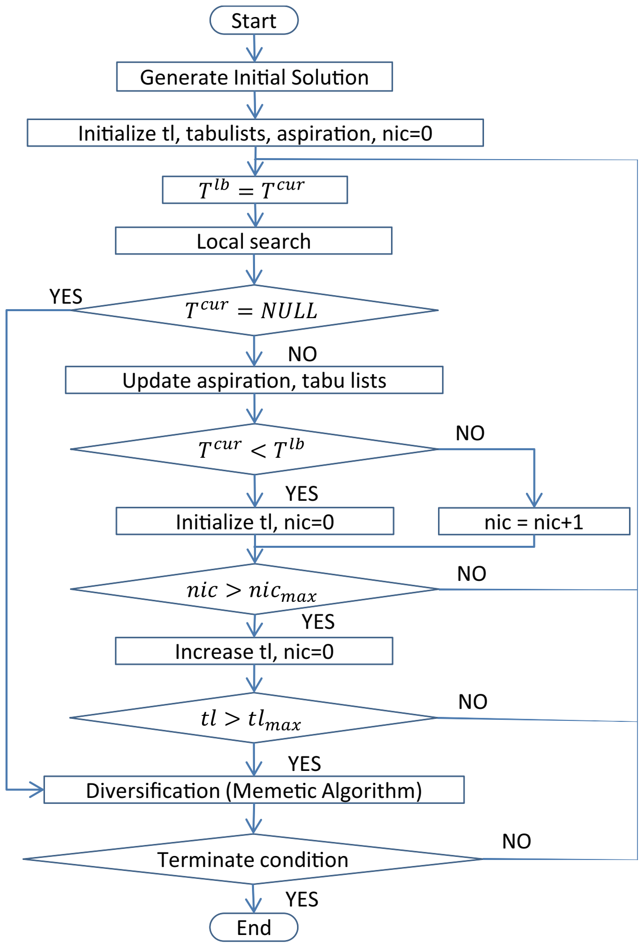

Fig. 3. Proposed Algorithm

As a result, Tabu Search may no longer be so efficient for the problem with a large graph. Memetic Algorithm, with evolutionary computation operators, is expected to help Tabu Search overcome this shortcoming. The proposed hybrid algorithm will be described in the next section.

III. Proposed Algorithm

As described in previous section, Tabu Search and Memetic Algorithm have their own advantages on solving optimization problems respectively. In this section, we would like to describe the proposed hybrid algorithm, in which Memetic Algorithm is used as a diversification strategy for Tabu Search. Without loss of generality, the flowchart of the proposed algorithm is shown in figure 3.

A. Generate Initial Solution

Let e_i denote an edge, one of its vertices belongs to k-spanning tree T_k and the other one does not. The set of all the e_i for a k-spanning tree T_k is denoted by $E_{NH}(T_k)$, which means a neighbor edges set for T_k. To generate a k-spanning tree, firstly a vertex is selected as the initial vertex for T_k at random. Then edges will be added to T_k one by one until a k-spanning tree is completely constructed. The added edge e is selected as $e := argmin\{w_e | e \in E_{NH}(T_k)\}$ under a probability p, or selected randomly from $E_{NH}(T_k)$ under probability $1 - p$, otherwise. In an extreme case, k-spanning tree would be constructed in a greedy strategy if the probability $p = 1$, or be constructed randomly if the probability $p = 0$. A good balance will be built up between generating good initial solutions and maintaining the diversity of initial solutions, if a proper probability p is found. In this research, we determine the probability $p = 0.85$ due to numeral experiments beforehand.

B. Tabu Search

The Tabu Search used in this paper is similar to the method in [20], in which aspiration criterion and dynamic length of tabu lists were used. In this section, the details on Tabu Search will be described.

1) Local Search: In local search, the current tree is considered to be translated into a new one, by exchanging one vertex contained in current tree with one that not be contained. Firstly, the set including all the vertices of k-spanning tree T_k, is denoted by $V(T_k)$. And we define the neighbourhood vertex set of T_k as follows:

$$V_{NH}(T_k) := \{v | (v, v') \in E, v \notin V(T_k), v' \in V(T_k)\}. \quad (1)$$

Then let T_k^{NH} be a new k-spanning tree obtained by adding one vertex $v_{add} \in V_{NH}(T_k)$ to T_k and deleting another vertex $v_{del} \in V(T_k)$. And the set of neighbourhood k-spanning trees T_k, denoted by $NH(T_k)$, contains all the possible k-spanning trees T_k^{NH} in G.

In order to obtain a better solution, *best improvement* is used in the proposed local search algorithm. The k-spanning tree $T^{NH_{best}}$ which has the smallest objective function value in the neighbourhood k-spanning tree set of T_k is selected as follows:

$$T^{NH_{best}} := argmin\{f(T_k^{NH}) | T_k^{NH} \in NH(T_k)\}. \quad (2)$$

However, in traditional *best improvement*, constructing and evaluating of all the $NH(T_k)$ would cost a lot of computing time. In order to make the local search more efficient, we construct the k-spanning trees by adding or deleting only the “necessary” vertices, so that we can find the $T^{NH_{best}}$ early before all the $NH(T_k)$ be constructed. The added vertex is selected as follows:

$$v_{add} := argmin\left\{\frac{\sum w(e)}{degree(v)} \middle| e = (v, v')\right\} \quad (3)$$

where degree(v)is the number of edges that can be connected to current tree T_k^{cur}, and $v \in V_{NH}(T_k), v' \in V(T_k^{cur})$. The deleted vertex is selected as follows:

$$v_{del} := argmax\left\{\frac{\sum w(e)}{degree(v)} \middle| e = (v, v')\right\} \quad (4)$$

where degree(v)is the number of edges that now contained in the current tree T_k^{cur}, and $v \in V(T_k^{cur}), v' \in V(T_k^{cur})$.

2) Parameters: The core procedure of Tabu Search is to forbid some movements based on *memory*. In the proposed algorithm, the“tabu” (forbiddance) is applied to edges that has added or deleted to the k-spanning tree recently, and tabu lists are used as memory to record edges that should be forbidden. To be concretely, *InList* and *OutList* are adopted keep the records of removed edges and added edges respectively. Tabu tenure, determined by the length of tabu lists, is a period for which it forbids edges in the tabu lists from adding or deleting. The lengths of tabu lists are dynamic in proposed algorithm, which help the local search implement centralization and diversification strategies. If the best solution in this iteration has not been updated for nic_{max} movements, the length of tabu lists will be increased by tl_{inc}. The search stops if the length of either of tabu lists is larger than tl_{max}. Some parameters are defined as follows:

$$tl_{min} := \min \left\{ \left\lfloor \frac{|V|}{20} \right\rfloor, \frac{|V|-k}{4}, \frac{k}{4} \right\}$$

$$tl_{max} := \left\lfloor \frac{|V|}{5} \right\rfloor$$

$$tl_{inc} := \left\lfloor \frac{tl_{max} - tl_{min}}{10} \right\rfloor + 1$$

$$nic_{max} := \max \{tl_{inc}, 100\}$$

where $|V|$ is the number of vertices in G, k is the cardinality of k-spanning tree, tl_{min} is the initial length and tl_{max} is the max length of tabu list.

3) Aspiration Criterion : The "tabu" mechanism that forbids some of the movements, helps the algorithm avoid falling to local optimal. However, it may also cause a loss of some information that the best solution may contained in. In order to avoid losing information, a procedure called $aspiration\ criterion$ is used in the proposed algorithm. That is, if $\gamma_e > f(T_k^{NH})$ is satisfied, the movement will be acceptable even e is included in $InList$ or $OutList$. The parameter γ_e called $aspiration\ criterion\ level$ are given to all of edges and are initially set to be:

$$\gamma_e = \begin{cases} f(T_k^{cur}) & e \in E(T_k^{cur}) \\ \infty & e \notin E(T_k^{cur}) \end{cases} \quad (5)$$

For each explored solution T_k, γ_e is updated as $\gamma_e := f(T_k)$ for each $e \in E(T_k)$.

C. Memetic Algorithm

If a solution could not be improved any more by Tabu Search, we would like to try Memetic Algorithm as a diversification strategy to improve the solution once more. This procedure is used to make the algorithm escape from local optimum by evolutionary computation operators. The proposed Memetic Algorithm is shown in the following:

Memetic Algorithm

```
Step 1 $P := Initialize(P)$
$while\ stop\ criterion\ not\ satisfied\ do$
Step 2 $P' := Genetic\ Operations(P)$
Step 3 $P' := UpdatingPopulation(P')$
Step 4 $P' := Replace(P \cup P')$
Step 5 $P'' := Tabu\ Search\ based\ Local\ Search(P')$
Step 6 $P'' := UpdatingPopulation(P'')$
Step 7 $P := Replace(P \cup P'')$
$endwhile$
Step 8 $Return(P)$
```

where P means the population of individuals now, P' means the individuals which are generated from the Genetic Operations, and P'' denotes the individuals improved by Tabu Search based Local Search.

1) Generating Initial Population: Firstly, the size of P should be determined. Obviously the larger the size is, the more new individuals would be generated. However, considering the computing cost for large size problems, we select the size of P as 4 in this study. The initial population P includes one individual obtained by Tabu Search and individuals generated under the procedure described in $section\ 3.A$. Additionally, in order to make sure that the structures of all the individuals are not the same, we replace the reduplicate individuals with new generated k-spanning trees.

2) Genetic Operation: In this paper, $crossover$, usually adopted in Evolutionary Computation, is used as the Genetic Operation. We use crossover operator to enlarge the explored domain, so that the algorithm can escape from local optima easily. The crossover operator is completed by two procedures:

Firstly, each individual except T_k itself in P is considered to be a cross partner T_k^C for T_k. More concretely, if T_k and its cross partner T_k^C have at least one common vertex, then a vertex set $V(G^C)$ is defined as: $V(G^C) = V(T_k) \bigcup V(T_k^C)$. Otherwise, edges and vertices and edges should be added to T_k until at least one common vertex is found, by the procedure we described in $section\ 3.A$ under probability $(1-p)$. Then a spanning tree T^{SP}, which contains all the vertices of $V(G^C)$, is constructed under the procedure we introduced in $section\ 3.A$.

Then, a Dynamic Programming algorithm, originally introduced in [17], is applied to the spanning tree T^{SP} for finding out the best k-spanning tree. Since the Dynamic Programming algorithm is very efficient, the crossover operator will help us get a good solution in a very short time.

3) Updating Population: Since there are so many improvements for individuals under the operators above, some of the individuals may be the same in offspring populations. In order to avoid redundancy searching, an updating operator is applied to the offspring populations. Concretely, the updating operator is to replace the repeated individuals with k-spanning trees constructed by the procedure introduced in $Section\ 3.A$ under probability $(1-p)$.

4) Tabu Search based Local Search: After Genetic Operation, each offspring will be further improved by local search. In the previous section, we introduced the procedures and parameters of Tabu Search, which is considered to be very useful. Now we want to make use of these concepts to conduct local search effectively. Consequently, the local search here used is the same as the one described in $Section 3.B$, excepting tabu list length set to be definite numbers.

5) Stopping criteria of Memetic Algorithm: We define a generation as an $idle\ generation$ if the objective function value is not improved in that generation. If idle generation occurs continuously for several times, the Memetic Algorithm stops. In order to have a complete searching, the Memetic Algorithm will not end until the number of idle generations is twice as much as the largest idle generations happened before. The stopping criteria function is shown as below:

$$i := \arg\max\{i_s, 2 * i_{max}\} \quad (6)$$

where i is the number of continuously occurred idle generations, i_s is the smallest number of idle generations which is determined in advance, and i_{max} is the largest idle generation happened before.

D. Centralization strategy

Centralization strategy is also concerned in the proposed algorithm. We would not finish the algorithm after doing the Memetic Algorithm instantly. A deeper search is needed, because the best solution may exist near to where Memetic Algorithm just searched. More concretely, the best solution generated from Memetic Algorithm is regarded as the initial solution for Tabu Search to restart the algorithm.

E. Stopping criteria

The stopping criteria of the proposed algorithm is as same as that we described in $Section\ 3.C.5$. The algorithm ends if the objective function value is no longer improved for several iterations.

IV. Experimental study

In order to evaluate the efficiency of the proposed algorithm, numeral experiments have been carried out. Beside the proposed method (Hybrid Algorithm Based on Memetic Algorithm and Tabu Search: HMATS), three state-of-the-art existing algorithms were also experimented. One is a Hybrid algorithm (Hybr.K) based on the Tabu Search and Ant Colony Optimization, originally proposed by Katageri $et.al$ [19]. The other two algorithms are Tabu Search algorithm (TSB) and Ant colony Optimization algorithm (ACO), both introduced by Blum $et\ al.$ in [18]. The proposed algorithm was coded in C language and compiled with C-Compiler: Microsoft Visual C++ 7.1. All the algorithms were carried out under the following compute environment: CPU: Pentium 4 3.06 GHz, RAM: 1 GB, OS: Microsoft Windows XP. The parameter settings as well as the source code used for all the experiments in this study were as same as the those provided by Katageri $et\ al.$ and Blum $et\ al.$. The k-spanning tree problem benchmark instances were downloaded from [21]. All of the 75 instances were created based on 15 different graphs, by changing the cardinality (k). In particular, for small instances, the number of edges in the graph are less then or equal to 2000, the $Limit\ Time$ was set to be 300 (s), and experiments were executed for 30 independent runs. For larger instances, the number of edges in the graph are larger than 2000, the $Limit\ Time$ was set to be 6000 (s), and experiments were executed for 10 independent runs. Part of experiment results are reported in this paper.

Tables I-X show the results of experiments for several benchmark instances. $|V|$, $|E|$ indicate the number of vertices and edges of graph G and k denotes the number of edges that the k-spanning tree contains separately. BKS means the best known solutions which have been obtained by Blum $et\ al.$ through their tremendous experiments. The rows headed “Best value”, “Mean value” and “Worst value” provide the best results, average results, and the worst results of each experiment. Results highlighted in bold means that the algorithm achieved the best solution among the compared methods for an instance. The values marked by $*$ denote that the best known solutions were updated by the proposed algorithm. Table XI summarizes the outperformed instances among the compared methods. For example, a row headed “Best value” indicates that the number of instances reached the best results among four methods out of all the benchmark instances.

Firstly, we can see from tables I-V, showing the results for the instances with small graphs ($|E| \leq 2000$), that performance of the proposed method is better than existing algorithms considering “Best value”. Then, tables VI-X, showing the results for the instances with large graphs ($|E| > 2000$), show that the performance of the proposed method is better than existing algorithms considering “Best value”, “Mean value” and “Worst value” . The better performance is due to the effect of the diversification strategy based on Memetic Algorithm, which enlarges the search area through genetic operators. Considering the performance of the proposed method, another feature is that the larger the graphs are, the better the results are than other existing methods. Finally, from the row headed “Best value” in table XI, it can be found that the proposed method obtained the best solutions out of 75 instances, which is significantly better than all the other algorithms. Furthermore, the proposed algorithm reaches 47 best known solutions out of 75 cases. We can also judge that the proposed algorithm is robust, since it has the most instances (59 cases) in the row headed “Mean value”. Thus it can be concluded that the proposed method is superior in terms of solution quality for all kinds of graphs.

V. Conclusion

In this paper we proposed a hybrid algorithm based on Memetic Algorithm and Tabu Search for k-minimum spanning tree problems. The experiments applied in existing benchmark instances show that the proposed algorithm is powerful in searching and has a stronger robust than other algorithms. It can be observed that a proper combination of metaheuristics is efficient for solving k-spanning tree problems.

We will do some experiments with much larger size of benchmark problems to show the effectiveness of the proposed algorithm in future.

References

[1] H.W. Hamacher, K. Jorsten, F. Maffioli, Weighted k-cardinality trees, *Technical Report*, Politecnico di Milano, Dipartimento di Elettronica, Italy, 1991.

[2] H.W. Hamacher, K. Jorsten, Optimal relinquishment according to the Norwegian petrol law: a combinatorial optimization approach. *Technical Report*, No. 7/93, Norwegian School of Economics and Business Administration, Bergen, Norway, 1993.

[3] L. R. Foulds, H.W. Hamacher, J. Wilson, Integer programming approaches to facilities layout models with forbidden areas. *Annals of Operations Research*,81:405-417, 1998.

[4] A. B. Philpott, N.C. Wormald, On the optimal extraction of ore from an open-cast mine, New Zealand: University of Auckland, 1997.

[5] R. Borndorfer, C. Ferreira, A. Martin, Decomposing matrices into blocks, *SIAM Journal on Optimization*, Vol. 9, No. 1, pp. 236-269, 1998.

[6] N. Garg, D. Hochbaum, An $O(\log k)$ approximation algorithm for the k minimum spanning tree problem in the plane, *Algorithmica*, Vol. 18, No.1, pp. 111-121, 1997.

[7] B. Ma, A. Hero, J. Gorman, O. Michel, Image registration with minimum spanning tree algorithm, *IEEE International Conference on Image Processing*, 2000.

[8] M. Fischetti, H.W. Hamacher, K. Jornsten, F. Maffioli, Weighted k-cardinality trees: complexity and polyhedral structure, *Networks*, Vol. 24, pp. 11-21, 1994.

[9] M. Marathe, D. Ravi, S.S. Ravi, D. Rosenkrantz, R. Sundaram, Spanning trees short or small, *SIAM J. Discrete Math*. 9 (2), 178-200, 1996.

TABLE I
$|V|$:400 $|E|$: 800 k:160 BKS:3062

Obejective function	HMATS	Hybr.K	TSB	ACOB
Best value	**3062**	3067	3070	3063
Mean value	3067.1	3069.2	3071.8	**3065.6**
Worst value	3087	**3072**	3076	3069

TABLE II
$|V|$:400 $|E|$:800 k:240 BKS:5224

Obejective function	HMATS	Hybr.K	TSB	ACOB
Best value	**5224**	5230	5238	5228
Mean value	**5226.5**	5235.3	5247.2	5240.1
Worst value	5241	**5238**	5257	5261

TABLE III
$|V|$:1000 $|E|$:1250 k:200 BKS:3456

Obejective function	HMATS	Hybr.K	TSB	ACOB
Best value	**3456**	3553	3588	3460
Mean value	3505.2	3600.1	3608.3	**3471.3**
Worst value	3569	3644	3636	**3487**

TABLE IV
$|V|$:1000 $|E|$:1250 k:400 BKS:8691

Obejective function	HMATS	Hybr.K	TSB	ACOB
Best value	**8692**	8857	8941	8802
Mean value	**8739.3**	8933.3	8997.4	8872.9
Worst value	**8853**	9010	9059	8936

[10] F. Maffioli, Finding a best subtree of a tree, *Technical Report* 91.041, Politecnico di Milano, Dipartimento di Elettronica e Informazione, 1991.

[11] F. P. Quintaoa, A.S. da Cunha, G.R. Mateus, and A. Lucena, The k-Cardinality Tree Problem: Reformulations and Lagrangian Relaxation, *Discrete Applied Mathematics* 158, 1305-1314, 2010.

[12] G. J. Woeginger, Computing maximum valued regions, *Acta Cybernet.* 10 (4), 303-315, 1992.

[13] N. Garg, A 3-approximation for the minimum tree spanning k vertices, in: *Proceedings of the 37th Annual IEEE Symposium on Foundations of Computer Science*, IEEE Society Press, Los Alamitos, 302-309, 1996.

[14] S. Arora, G. Karakostas, A 2 + ε approximation algorithm for the k-MST problem, in:*Proceedings of the SIAM Symposium on Discrete Algorithm, (SODA)* 2000, SIAM, Philadelphia, 402-408, 2000.

[15] M. Ehrgott, J. Freitag, H.W. Hamacher, F. MaLoli, Heuristics for the k-cardinality tree and subgraph problem. *Asia-Pacifc Journal of Operational Research* 14(1):87-114, 1997.

[16] D. Urosevic, J. Brimberg, N. Mladenovic, Variable neighbourhood decomposition search for the edge weighted k-cardinality tree problem, *Comput. Operat. Res.* 31, 1205-1213, 2004.

[17] C. Blum. Revisiting dynamic programming for finding optimal subtrees in trees. *European Journal of Operational Research*, 177, 102-115, 2007.

[18] C. Blum, M. Blesa, Newmetaheuristic approaches for the edge-weighted k-cardinality tree problem, *Comput. Oper. Res.* 32, 1355-1377, 2005.

[19] H. Katagiri, T. Hayashida, I. Nishizaki and J. Ishimatsu, A hybrid algorithm based on tabu search and ant colony optimization for k-minimum spanning tree problems, *Modeling Decisions for Artificial Intelligence, Torra, Narukawa and Inuiguchi (eds.), MDAI 2009* , Lecture Notes in Artificial Intelligence 5861, pp. 315-326, Springer, Berlin/Heidelberg, 2009.

[20] H. Katagiri, T. Hayashida, I. Nishizaki and J. Ishimatsu, An approximate solution method based on tabu search for k-minimum spanning tree problems, *International Journal of Knowledge Engineering and Soft Data Paradigms* , Volume 2 Issue 3, 263-274, 2010.

[21] A Library for the Edge-Weighted K-Cardinality Tree Problem, http://iridia.ulb.ac.be/blum/kctlib/, 2003. (access: 2010/ 6/ 31)

[22] F. Glover, Future paths for integer programming and links to artificial intelligence. *Computers and Operations Research* 5:533-49, 1986.

[23] F. Glover, M. Laguna, Tabu search. Dordrecht: Kluwer Academic Publishers; 1997.

[24] P. Moscato, On evolution, search, optimization, genetic algorithms and martial arts: Toward memetic algorithms, Caltech Concurrent Comput. Program, CalTech, Pasadena, CA, Rep. 826, 1989.

TABLE V
$|V|$:1000 $|E|$:1250 k:600 BKS:15916

Obejective function	HMATS	Hybr.K	TSB	ACOB
Best value	**15917**	16041	16132	16186
Mean value	**15955.8**	16084.9	16162.3	16285.7
Worst value	**15998**	16111	16218	16384

TABLE VI
$|V|$: 2500 $|E|$:4900 k:500 BKS:8150

Obejective function	HMATS	Hybr.K	TSB	ACOB
Best value	**∗8146**	8621	8722	8406
Mean value	**8259.4**	8673.9	8824.5	8481
Worst value	**8356**	8811	8874	8605

TABLE VII
$|V|$:2500 $|E|$:4900 k:1000 BKS:17437

Obejective function	HMATS	Hybr.K	TSB	ACOB
Best value	**17508**	18343	18374	17973
Mean value	**17810.2**	18663.8	18465.2	18155.5
Worst value	**18242**	19031	18624	18358

TABLE VIII
$|V|$:2500 $|E|$: 4900 k:1500 BKS:28683

Obejective function	HMATS	Hybr.K	TSB	ACOB
Best value	**∗28549**	29771	29878	30410
Mean value	**28748**	29938	30097.4	30518.2
Worst value	**29353**	30061	30404	30722

TABLE IX
$|V|$:2500 $|E|$:4900 k:2000 BKS:43627

Obejective function	HMATS	Hybr.K	TSB	ACOB
Best value	**∗43558**	44101	43955	45520
Mean value	**43597.7**	44411.3	44059.7	45697.7
Worst value	**43646**	44876	44151	45940

TABLE X
$|V|$:2500 $|E|$:4900 k:2250 BKS:53426

Obejective function	HMATS	Hybr.K	TSB	ACOB
Best value	**∗53407**	53494	53825	55780
Mean value	**53411.4**	53494	53907.8	55959.3
Worst value	**53418**	53494	53988	56081

TABLE XI
PERFORMANCES FOR EACH ALGORITHMS

Obejective function	HMATS	Hybr.K	TSB	ACOB
Best value	**75**(instances)	30	15	18
Mean value	**59**	27	4	11
Worst value	**47**	35	5	13
BKS	**47**	30	15	17