Studies on Bio-Inspired Hybrid Metaheuristics for k-Cardinality Tree Problems

(最小k部分木問題に対する 生物規範型ハイブリッドメタ戦略 に基づく近似解法の研究)

March 2014

QINGQIANG GUO

Department of System Cybernetics
Graduate School of Engineering, Hiroshima University, Japan

2013
Doctoral Dissertation

Studies on Bio-Inspired Hybrid Metaheuristics for k-Cardinality Tree Problems

Systems Optimization Laboratory Department of System Cybernetics
D116705 QINGQIANG GUO

Supervisor

Associate Professor Dr. Hedeki Katagiri

March 2014

Department of System Cybernetics
Graduate School of Engineering, Hiroshima University, Japan

This doctoral dissertation is submitted to the Department of Cybernetics, Graduate School of Engineering, Hiroshima University, Japan for partial fulfillment of requirements of Doctor of Engineering in Artificial Complex Systems Engineering.

QINGQIANG GUO

March 2014

To my family

Preface

The k-Cardinality Tree Problem (kCTP) is one of the famous combinatorial optimization problems, finding the best solution out of a very large, but finite, number of possible solutions. Accordingly, the goal of the kCTP is to find a subtree with exactly k edges in an undirected graph G, such that the sum of edges' weight is minimal. It is also a generalized version of the well known minimum spanning tree (MST) problem when k =|V|-1, where |V| is the number of vertices in the graph.

Owing to its outstanding combinatorial optimal properties for solving real-world decision making problems, kCTP has been applied in many fields, such as facility layout, matrix decomposition, telecommunication, and image processing. Additionally, since the kCTP is suggested to be *NP*-hard, i.e., hardly be solved in polynomial-time, it is also a challenging combinatorial optimization problem providing excellent benchmarks to estimate the efficiency of optimal approaches. As a result, the kCTP has attracted the attention of a large number of researchers and various of approaches have been proposed in last decades.

To solve the kCTP, exact methods (e.g., formulate it into an integer linear program with generalized circle elimination constraints, and solve it by branch and bound method) have been applied. However, as mentioned above, since it is *NP*-hard, exact methods could hardly solve complex instances, such as problems with large graphs, in a reasonable time. "Trial and error", called heuristics, are then the most reliable and efficient approach for finding possible answers of such complex optimization problems. Experimental results show that they are able to find "good" solution (i.e., low error from the real optimal solution) "quickly".

In 70s, new kinds of approximate algorithms called metaheuristics were emerged. They combine heurisictcs in high level frameworks aimed at efficiently and effectively

exploring the search space. Evolutionary Algorithm (EA), Tabu Search (TS), Simulated Anealing (SA), Ant Colony Optimization (ACO), and Iterated Local Search (ILS) are typical metaheuristics. Among those metaheuristics, EA and ACO can be classified as biological inspired computations, which act after principles that exist in natural systems. The approaches adopting such principles (e.g., using mechanism of pheromone evaporation,ACO could generates new solution with high precision and avoid the convergence to a local optimal solution), are enable to reach solutions with enhanced robustness and flexibility and are expected to solve complex optimization problems. Nowadays, it has been shown that a good combination of metaheuristics can lead to more efficient behavior and greater flexibility for solving combinatorial optimization problems. Such combination of metaheuristics are called hybrid metaheurisitcs. It is a new trend now to focus on hybrid metaheuristic rather than the scope of single metaheuristic.

In this dissertation, new hybrid metaheuristics combining bio-inspired algorithms (ACO, Immune algorithm, Memetic Algorithm) with TS and/or Dynamic Programming are proposed for the kCTP. Properties of metaheuristics and hybrid metaheuristics and the way to construct an efficient hybrid metaheuristic for kCTP are also discussed. Numerical results show that proposed algorithms are competitive to existing algorithms from the viewpoint of solution accuracy and computing time. More specifically, the proposed algorithms reached or updates almost of all of the best known solutions in the literature (benchmark instances proposed by Blum *et al.*). It also indicates that nothing else matches its balance of diversification strategy and centralization strategy in hybrid metaheuristics.

Acknowledgement

At the outset I express my deep sense of gratitude to my revered supervisor Associate Professor Dr. Hideki Katagiri of Systems Optimization Laboratory, Hiroshima University, for his key advice, valuable suggestions throughout the course of the study. Working under his supervision was a process of growing up. The development of hard-working and rigorous scholarship character is much more important than knowledge and will benefit me through my life.

I would like to thank Professor Dr. Ichiro Nishizaki, Professor Dr. Katsuhiko Takahashi and Assistant Professor Dr. Tomohiro Hayashida for offering valuable comments. Especially, I appreciate the help of Professor Dr. Ichiro Nishizaki who initially accepted me to Hiroshima University and has supported me in my study.

I also wish to express my sincere gratitude to Assistant Professor Dr. Takeshi Matsui for his sincere guidance and encouragement. I owe this work to his helpful advice and suggestions. Without his close supervision the research work might not be in the complete form.

I wish to thank all the students of the Systems Optimization Laboratory for their cordial cooperation and friendship.

Finally, I am grateful to my parents for their love, support and unattainable understanding and would like to thank my whole family members. I wish to express my deepest thanks to my wife Yaqin, my true friend. The research work would not have been completed without her love, help and encouragement.

Hiroshima University QINGQIANG GUO

March 2014

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Introduction and Historical Remarks on k-Cardinality Tree Problems

The k-cardinality tree problem (kCTP), also referred to as the k-minimum spanning tree problem, is a combinatorial optimization problem. It generalizes the well known Minimum Spanning Tree problem. Let $G = (V, E)$ be an undirected graph, which is made up by connecting a set of vertices V and edges E. Each edge $e \in E$ is attached with a nonnegative value w_e, called a weight. The goal of this problem is to find an acyclic and connected subset with exactly k ($k \leq |V|-1, |V| is the number of vertices$) edges of which the total weight is minimized [5]. The subset matching these conditions must form a tree, which we call a k-cardinality tree, denoted by T_k. The problem is mathematically formulated as follows:

$$\begin{aligned} &\text{minimize } w(T_k) = \sum_{(u,v)\in T_k} w_{(u,v)} \\ &\text{subject to } \quad T_k \in \mathcal{T}_k \end{aligned}$$

where $u \in V$, $v \in V$, $(u, v) \in E$, T_k is the edges set of a tree, and $\mathcal{T}_k$ is a set containing all feasible solutions in graph G. Figure 1.1 shows an example of a k-cardinality tree in a connected graph. The figures attached to edges are weights and the edges in the 4-cardinality tree are shaded. The total weight of the 4-cardinality tree is 11.

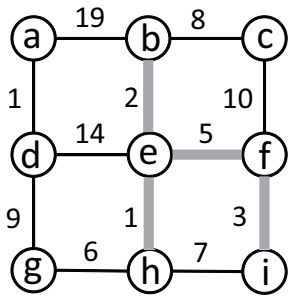

Figure 1.1: A 4-cardinality tree in a graph.

Owing to its outstanding combinatorial optimal properties for solving real-world decision making problems, kCTP has been applied in many fields, such as facility layout [6], matrix decomposition [23], telecommunication [18], and image processing [26].

The example introduced in Figure 1.1 can be easily solved even by enumerating all feasible solutions because the size of the graph is small. However, the problem becomes very complex when the graph is large. Figures 1.2 and 1.3 show a grid graph and a k-cardinality tree in that graph, respectively. It may cost a huge time to enumerate all the feasible solutions from such a large graph. In fact, previous researches have suggested that kCTP is an *NP*-hard problem [15]. Incidentally, this problem can be polynomially solved in two cases. One is that there are only two distinct weights in a graph [15], and the other is that a graph is given as a tree [38].

During the past few years, many algorithms have been proposed to solve kCTP [37]. The first exact algorithm was presented by Fischetti *et al.* [9], in which kCTP was formulated into an integer linear program with generalized circle elimination constraints. Then, Quintao *et al.* [43] [52] proposed two integer programming formulations, Multiflow Formulation and a formulation based on the Miller-Tucker-Zemlin constraints, for solving kCTP.

As has been pointed out that kCTP is NP-hard, it is extremely difficult to find an optimal solution in polynomial time by exact methods. In other words, though methods introduced above can reach optimal solutions, their computing time may

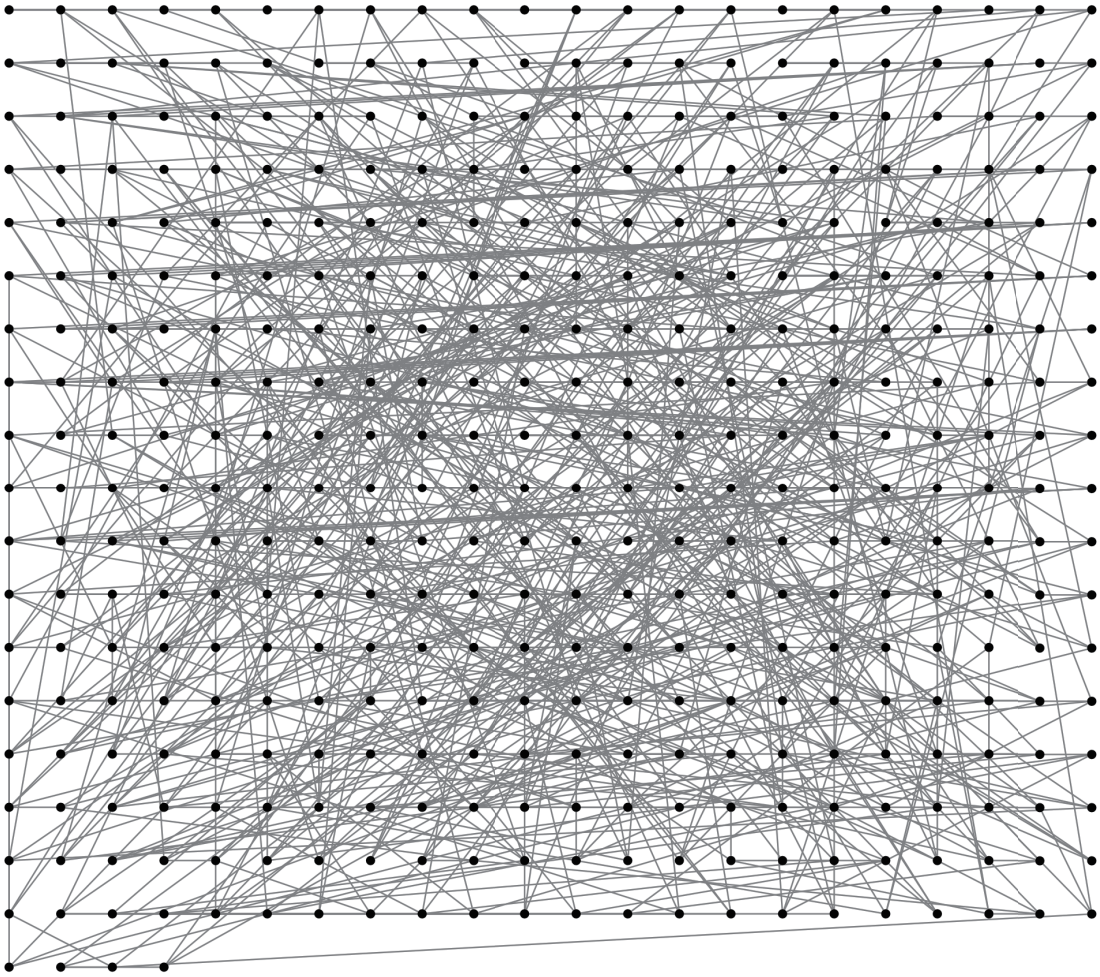

Figure 1.2: g400-4-01.g, a graph with 400 vertices and 800 edges.

be very large when solving complex problems. Therefore, a lot of approaches, called approximation algorithms, are proposed to find near-optimal solutions in polynomial time. At first, an $O(\sqrt{k})$-approximation algorithm for the vertex-weighted problem on grid graphs was proposed by Woeginger [6]. In 1995 [10], an $O(log^2 k)$ approximation was provided for finding the tree that spans k vertices in a graph. [12] proposed a constant-factor approximation for the problem in the plane. Later, a $2(\sqrt{k})$ approximation algorithm was obtained by Ravi *et al.* [15], and then a 3-approximation algorithm for a rooted case was proposed by Garg [14]. In [22], an algorithm with

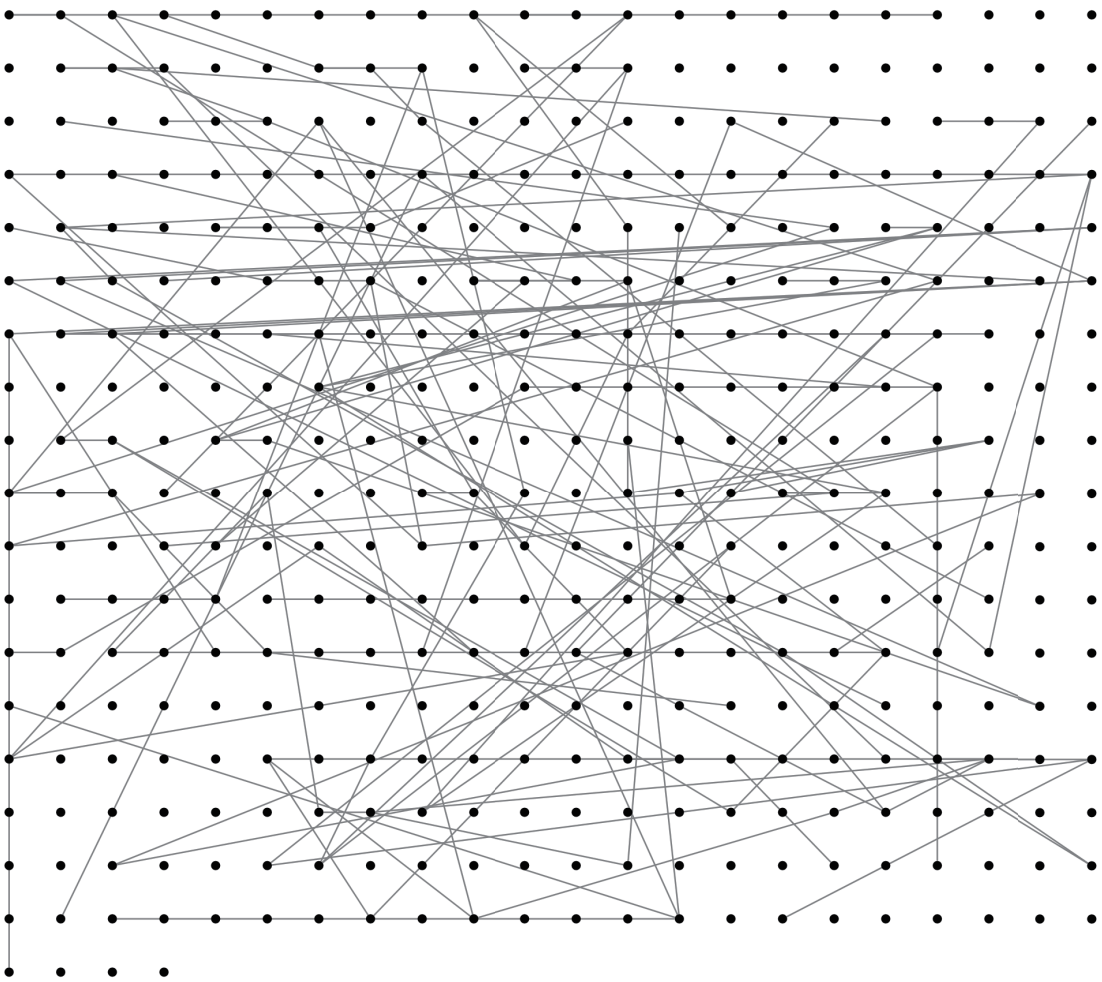

Figure 1.3: A cardinality tree of g400-4-01.g with k=160.

approximation factor of 2.5 was proposed by using a pruning technique for the case in which no root vertex is specified. Recently, a polynomial time 2-approximation algorithm for finding the minimum tour that visits k vertices was proposed by Garg [35].

However, in real life people usually need a solution with high precision which is obtained in a reasonable computing time. "Trail and error", called heuristics, are then the most reliable and efficient approach for finding possible answers of complex optimization problems. Various types of heuristic and metaheuristic methods have

been proposed in recent years. Among them Heuristics based on greedy strategy and Dynamic Programming (DP) were introduced by Ehrgott *et al.* [17]. Moreover, a heuristic based on Variable Neighborhood Decomposition Search, which has a good performance for problems of small size, was presented by Urosevic *et al.* [33].

In 2007, Blum [38] proposed an improved dynamic programming approach: after a minimum spanning tree for a given graph is obtained, DP is applied in order to obtain an optimal subtree with k edges from the minimum spanning tree. This algorithm has been proved to be efficient even for problems of large size. DP was also combined with the evolutionary computation paradigm for the application to kCTP in [36] [46].

Metaheuristic algorithms, such as Ant System [32] and ACO [29] based on pheromone have been proposed. More over, Tabu Search (TS), Evolutionary Computation and ACO for solving kCTP were studied in [34]. It showed that the performances of these metaheuristics depend on the characteristics of the tackled instances, such as the graph size, degree (number of edges that one vertex connects), and cardinality (the value of k).

Among those metaheuristics, EA and ACO can be classified as biological inspired computations, which act after principles that exist in natural systems. The approaches adopting such principles (e.g., using mechanism of pheromone evaporation,ACO could generates new solution with high precision and avoid the convergence to a local optimal solution), enable to reach solutions with enhanced robustness and flexibility and are expected to solve complex optimization problems.

Nowadays, it has been shown that a good combination of metaheuristics can lead to more efficient behavior and greater flexibility for solving combinatorial optimization problems. Such combination of metaheuristics is called hybrid metaheurisitc [53]. It is a new trend now to focus on hybrid metaheuristic rather than the scope of single metaheuristic. A hybrid metaheuristic based on TS and ACO was constructed by Katagiri *et al.*. Their experimental results using benchmark instances demonstrated that the hybrid metaheuristic provides a better performance with solution

accuracy over existing metaheuristics.

However, those metaheuristics mentioned above may not be effective in some cases, especially for the problems with large size graphs. That is because complexity of the problem increases significantly with size of the graph. There is still room for improvement in precision of solutions for those instances with large graphs or instances various in graph types or cardinalities (k).

In this dissertation, new hybrid metaheuristics combining bio-inspired algorithms (ACO, Immune Algorithm, Memetic Algorithm) with TS and/or DP are proposed for the kCTP. Properties of metaheuristics and hybrid metaheuristics for kCTP and how to construct an efficient hybrid metaheuristic are also discussed. Numerical results show that proposed algorithms are competitive to existing algorithms from the viewpoint of solution accuracy and computing time. Specifically, proposed algorithms updates almost all of the best known solutions of large benchmark instances proposed by Blum *et al.* (e.g., graphs with more than 5000 edges). It also indicates that nothing else matches its balance of diversity and centralization of solutions in hybrid metaheuristics.

1.2 Outline of the Thesis

New hybrid metaheuristics are considered for solving k-cardinality tree problems in this doctoral dissertation. The organization of each chapter is briefly summarized as follows.

In *Chapter* 2, the basic concepts and methods used in our study are introduced briefly. Firstly, combinatorial optimization concepts, such as NP-hard, benchmark problems, heuristic and metaheuristic, are outlined by introducing to metaheuristics and dynamic programming for solving kCTP. Results of experiments applying the above methods show that heuristics (i.e., Prim algorithm) and metaheuristics (i.e., Tabu Search) are not efficient, and that dynamic programming can hardly obtain

high quality solution to problems with large graphs. Secondly, we introduce the Biological Inspired Algorithms, which have attracted much attention in recent years. Finally, basic concepts of Hybrid Metaheuristics, promising methods to obtain high quality solutions to combinatorial optimization problems in a reasonable time, are provided.

In *Chapter* 3, a new hybrid metaheuristic based on TS and ACO is proposed. In the proposed tabu search, our neighborhood is different from the Blum-Blesa one. While only the leaf vertices can be selected in the transition of the Blum-Blesa's algorithm, all of vertices adjacent to the current tree can be selected in the proposed algorithm. This extension enables us to strengthen the intensification ability of local search. We also propose a diversification algorithm based on ACO by extending the Blum-Blesa's algorithm. One of the characteristics in the proposed algorithm is that our algorithm deposits pheromone on the edges selected in the local optimal solutions which were obtained by the TS-based local search algorithm. This procedure allows the proposed algorithm to explore a wider search space than the Tabu-Search-based local search method. To demonstrate efficiency of the proposed solution method, we have compared the performances of the proposed method with those of existing algorithms using the well-known benchmark problems. The numerical experimental results show that the proposed method has improved some of the best known solutions and values with very short computational time, and provides a better performance with the solution accuracy over existing algorithms.

Chapter 4 focuses on a Hybrid Metaheuristic based on TS and Immune Algorithm. Since TS stops when the length of tabu list reaches its limitation, Immune Algorithm is applied to enlarge the search area by generating a new initial solution for TS. In other words, Immune algorithm acts as a diversitification strategy for TS, and thus the algorithm can search the solution space with a large step size. The proposed immune algorithm is inspired by immune systems, especially the mechanism of keeping diversity of the immune cells. More specifically, population of antibodies are constructed with a variety of infeasible solutions. At each step the population-based algorithm deals with a set of infeasible solutions rather than with a single

one, providing a natural and intrinsic way to explore the search space. Experimental results show that Immune Algorithm improves the solution accuracy significantly. Some best known solutions are also updated by the proposed algorithm. We arrive at a conclusion that a well-designed hybrid metaheuristic algorithm is efficient for solving the kCTP.

In *Chapter* 5, we develop two Hybrid Metaheuristics, both of which are based on Memeteic Algorithm and TS. In the first one, a Memetic Algorithm based on TS-based local search is proposed. It has both merits of Evolutionary Computation and Local Search. Note that a configuration is a list of vertices of a feasible solution. To enlarge the search area, a crossover operator is applied to combine all vertices of two configurations and returns a feasible solution with a good objective function value. Moreover, to find the optimal solution, TS with short-term memory is applied to each feasible solution generated by crossover. To enhance the quality of initial population, one configuration of initial population is generated by DP. Experimental results show that the proposed algorithm has a high solution precision and a short computing time. That is because the crossover in memetic algorithm enlarges search area effectively, and local search improves the solution greatly. The second one is a TS with Memetic Algorithm, which acts as a powerful diversitification strategy. In addition, the TS with dynamic tabu list improved the precision of solution significantly. Experimental results show that the new hybrid metaheuristic is dramatically superior to exiting algorithms in precision. Specifically, some of our proposed algorithms reach or update almost all of the best known solutions of large benchmark instances proposed by Blum *et al.* (e.g., graphs with more than 5000 edges). It also indicates that nothing else matches its balance of diversity and centralization of solutions in hybrid metaheuristics.

Chapter 6 concludes the doctoral dissertation and briefly summarizes this research.

1.3 List of Publications

[A-1] H. Katagiri, T. Hayashida, I. Nishizaki, Q. Guo, A hybrid algorithm based on tabu search and ant colony optimization for k-minimum spanning tree problems, *Expert Systems with Applications*, 39, pp. 5681-5686, 2012

[A-2] Q. Guo, H. Katagiri, A Memetic Algorithm based on Tabu Search for k-cardinality tree problems, *Scientiae Mathematicae Japonicae* e-2013, pp. 609-619, 2013.

[B-1] Q. Guo, H. Katagiri, I. Nishizaki, T. Hayashida, A Hybrid Algorithm Based on Memetic Algorithm and Tabu Search for k-Minimum Spanning Tree Problems, *International Multiconference of Engineers and Computer Scientists 2012*, pp. 1611-1616, 2012.

[B-2] Q. Guo, H. Katagiri, A Hybrid Approach Based on Exact Methods andMeta-heuristics for k-Minimum Spanning Tree Problems, *15th Czech-Japan Seminar on Data Analysis and Decision Making under Uncertainty*, pp. 192-199, 2012.

[B-3] Q. Guo, H. Katagiri, A Hybrid Algorithm Based on Tabu Search and Immune Algorithm for k-Cardinality Tree Problems, *SCIS-ISIS 2012*, pp. 346-351, 2012.

[C-1] H. Katagiri, Q. Guo, A Hybrid-Heuristics Algorithm for k-Minimum Spanning Tree Problems, *Lecture Notes in Electrical Engineering*, Volume 186, pp. 167-180, 2013

Chapter 2

Basic Concepts and Methods

In this chapter, the basic concepts and methods used in our study are introduced briefly. Firstly, combinatorial optimization concepts, such as NP-hard, benchmark problems, heuristic and metaheuristic, are outlined by introducing metaheuristics and dynamic programming for solving k-Cardinality Tree Problems(kCTP). Results of experiments applying the above methods show that heuristics (i.e., Prim algorithm) and metaheuristics (i.e., Tabu Search) are not efficient, and that dynamic programming can hardly obtain high quality solution to problems with large graphs. Secondly, we introduce the Biological Inspired Algorithms, which have attracted much attention in recent years. Finally, basic concepts of Hybrid Metaheuristics, promising methods to obtain high quality solutions to combinatorial optimization problems in a reasonable time, are provided.

2.1 A Review of Heuristics and Dynamic Programming for k-Cardinality Tree Problems

There are generally three principles to find a solution in an optimization problem by Heuristic. They are *Constructive Heuristic*, *Improvement Heurisics*, and *Metaheuristics* which are usually based on the former two heuristics with strategies. To solve the kCTP, the most often used constructive heurisitc is Prim algorithm, a *greedy heuristic*. The edge with the lowest weight is chosen at each construction

step to extend the current partial solution. In case of improvement heurisitics, local search starts from an initial solution and tries to find a better solution from the neighborhood of the current solution. Vertex swap or edge exchange are the commonly used for solving kCTPs. Combining basic heuristics in high level frameworks, *metaheuristic* could find a better solution efficiently and effectively. For example, TS is one of well-known metaheuristics for solving combinatorial optimization problems. It has *memory* of solution spaces that have been searched, and prevents from searching those areas again for a while. This mechanism helps local search work much more efficiently.

Although Dynamic programming is classified to $Exact\ Methods$, we also introduce it here since the minimum spanning tree is generated by heuristic in case of solving kCTPs. In the following parts, we will explain these classical methods introduced above for solving kCTPs.

2.1.1 Prim algorithm

Prim algorithm is usually used to construct a solution for the well known minimum spanning tree problems. It can also quickly grow a k-cardinality tree by adding one edge at a time. To be more specific, let T be a subset of a k-cardinality tree T_k. We call an edge a $safe\ edge$ if T is still a tree after being added with it. More specifically, it is an edge, one of its vertices belongs to tree T and the other does not. Firstly, a vertex is selected randomly to be the first component of tree T. Then in each step, one safe edge should be added to T until there are k edges in the tree T. The pseudo-code is shown as follows:

Growing a k-cardinality tree

T$\Leftarrow$ select one vertex randomly

while k-cardinality tree is not completed **do**

List ← generate list of safe edges

(u, v) ← an edge with minimal weight in the List of safe edges

T← T ∪ (u, v)

Update the List of safe edges

end while

2.1.2 Tabu Search

Metaheuristics are usually developed for solving complex combinatorial optimization problems, e.g. the kCTP. TS, offering global search strategy, is one of the most notable Metaheuristics.

Local Search, an improvement heuristic, is usually used to improve an initial solution. A class of modifications to the current solution is constructed and a better one or the best one is selected to be new current solution. It stops when there is no better solution in the neighborhood of the current solution. To solve kCTPs, *vetex exchange* and *edge exchange* are basic modifications. The first one is to remove a vertex from the current solution and add another vertex to the tree. Edges connected these vertices are also changed accordingly. In the later one, edges are removed from the solution and replaced by new edges, leading to a new k-cardinality tree. Since the later one can only change the leaf edges, it is hardly able to improve the current solution significantly. In our research, *vertex exchange* is applied in local search. Pseudo-codes of local search are shown in as follows:

Algorithm 1 LocalSearch

Input: T_k^{cur}

$V_{add} \leftarrow V_{NH}(T_k^{cur})$

$T_k^{NH_{best}} \leftarrow T_k^{cur}$

while $V_{add} \neq \emptyset$ **do**
$v_{add} \leftarrow \arg\min \{V_{add}\}$
$V_{add} \leftarrow V_{add} \setminus v_{add}$
$E_{add_1} \leftarrow \{(v, v_{add}) | v \in V(T_k^{cur})\}$
$T_{k+1}^{NH} \leftarrow$Construct_k-plus-one_MinimumSpanningTree(v_{add})
$V_{del} \leftarrow V_{NH}(T_k^{cur})$
while $V_{del} \neq \emptyset$ **do**
$v_{del} \leftarrow \arg\min \{V_{del}\}$
$V_{del} \leftarrow V_{del} \setminus v_{del}$
$T_k^{NH} \leftarrow$Construct_k_MinimumSpanningTree(T_{k+1}^{NH}, v_{del})
if $T_k^{NH} \neq \emptyset$ **and** $f(T_k^{NH_{best}}) > f(T_k^{NH})$ **and** can translate to T_k^{NH} **then**
$T_k^{NH_{best}} \leftarrow T_k^{NH}$
end if
end while
end while
Output: $T_k^{NH_{best}}$

Algorithm 2 Construct_k-plus-one_MinimumSpanningTree
Input: v_{add}
$e_{min_1} \leftarrow \arg\min_{e \in E_{add_1}} \{w(e)\}$
$T_{k+1}^{NH} \leftarrow (V(T_k^{cur}) \cup v_{add}, E(T_k^{cur}) \cup e_{min_1})$
$E_{add_1} \leftarrow E_{add_1} \setminus e_{min_1}$
while $E_{add_1} \neq \emptyset$ **do**
$e_{min_1} \leftarrow \arg\min_{e \in E_{add_1}} \{w(e)\}$
$T_{k+1}^{NH} \leftarrow T_{k+1}^{NH} \cup e_{min_1}$
$E_{add_1} \leftarrow E_{add_1} \setminus e_{min_1}$
$E_{loop} \leftarrow$ edges set of loop in T_{k+1}^{NH}
$e_{max} \leftarrow \arg\max_{e \in E_{loop}} \{w(e)\}$
$T_{k+1}^{NH} \leftarrow T_{k+1}^{NH} \setminus e_{max}$
if $f(T_{k+1}^{NH_{best}}) > f(T_{k+1}^{NH})$ **then**
$T_{k+1}^{NH_{best}} \leftarrow T_{k+1}^{NH}$

end if
end while
return T_{k+1}^{NH}

Algorithm 3 Construct_k_MinimumSpanningTree

Input: T_{k+1}^{NH}, v_{del}
$e_{min}^{del} \leftarrow \arg\min_{e \in (v_{del}, v)} \{w(e) | v' \in T_k^{cur}\}$
if $f(T_k^{NH_{best}}) < (\sum_{e \in E(T_{k+1}^{NH})} w(e)) - w(e_{min}^{del})$ **then**
return $\emptyset$
end if
$T_k^{NH} \leftarrow T_{k+1}^{NH} \setminus v_{del}$
take subtrees which are generated by deleting v_{del} from T_{k+1}^{NH} as hyper vertices S_r, $r = 0, 1, 2 \ldots$
$E_{add_2} \leftarrow \{(v_i, v_j) | v_i \in S_k, v_j \in S_l, k \neq l\}$
repeat
$e_{min_2} \leftarrow \arg\min_{e \in E_{add_2}} \{w(e)\}$
if $f(T_k^{NH_{best}}) < (\sum_{e \in E(T_k^{NH})} w(e)) + w(e_{min_2})$ **then**
return $\emptyset$
end if
if there is no loop in $e_{min_2} \cup T_k^{NH}$ **then**
$E(T_k^{NH}) \leftarrow E(T_k^{NH}) \cup e_{\min_2}$
end if
$E_{add_2} \leftarrow E_{add_2} \setminus e_{\min_2}$
until T_k^{NH} is a tree
return T_k^{NH}

Tabu Search, firstly proposed by Glover *et al.* [2] [19], is one of the mostly used metaheuristics for solving combinatorial optimization problems. TS enhances its search ability based on local search. The most important characteristic of TS is that it uses a concept of *memory* to control movements via a dynamic list

of forbidden movements. To be more specific, the solutions which have been searched will be "tabu" (forbidden) from visiting for a while. This mechanism allows TS to intensify or diversify its search procedure in order to escape from local optima. Incidentally, TS has also been proved to be effective in solving kCTPs [34]. The search starts from an initial solution. Parameters and other details of TS will be proved in Chapter 3. Pseudo-codes of TS are shown as follows:

Algorithm 4 TabuSearch

Input: a problem instance (G, w, k)

$T_k^{cur} \leftarrow$ GenerateInitialSolution()
$T_k^{gb} \leftarrow T_k^{cur}$
while $tl \leq tl_{max}$ **do**
  Initialize($InList$, $OutList$, tl, γ_e)
  $T_k^{lb} \leftarrow T_k^{cur}$
  $T_k^{NH_{best}} \leftarrow$LocalSearch($T_k^{cur}$)
  $T_k^{cur} \leftarrow T_k^{NH_{best}}$
  if $T_k^{cur} \neq \emptyset$ **then**
    if $f(T_k^{lb}) > f(T_k^{cur})$ **then**
      $T_k^{lb} \leftarrow T_k^{cur}$, $nic \leftarrow 0$
      if $f(T_k^{gb}) > f(T_k^{cur})$ **then**
        $T_k^{gb} \leftarrow T_k^{cur}$
      end if
    else
      $nic \leftarrow nic + 1$
    end if
    if $nic > nic_{max}$ **then**
      $tl \leftarrow tl + tl_{inc}$
    end if
  else

PerformRestart()
$nic \leftarrow 0$
end if
end while
Output: T_k^{gb}

2.1.3 Dynamic Programming

The Dynamic Programming (DP) [11] is a technique based on a very simple idea, solving the current problem with results of subproblem that has been already solved. In this chapter DP is applied to the T^{SP}, a minimum spanning tree, for finding out the best k-cardinality tree. We revisit the DP algorithm of Blum [38] for finding the best k-cardinality tree in an edge weighted graph that is itself a tree. This algorithm has polynomial running time. Firstly, a spanning tree is constructed by the Prim algorithm. Then DP is applied to obtain the best k-cardinality tree in the constructed spanning tree.

Algorithms of the proposed method are shown as follows.

Algorithm 5 Dynamic programming algorithm for solving kCTP
Input: Graph G, cardinality k
Construct a minimum spanning tree T^{SP}
DP for finding optimal k-cardinality tree in tree T^{SP}
Retrieval of a k-cardinality tree
Output: The k-cardinality tree T

Algorithm 6 Constructing a Minimum spanning tree T^{SP}
Input: Graph G
New vertex : a new vertex selected randomly
for there is vertex (vertices) has not been spanned **do**

for all vertices have not yet been spanned **do**
calculate their costs (edge weights) to connect T
end for
Span the vertex by the edge with smallest cost, and make it as the new vertex
end for
Output: The minimum spanning tree T^{SP}

Algorithm 7 DP for finding optimal k-cardinality trees in trees[38]
Input: A rooted tree T with root node v_{root}, and a maximum cardinality $k \leq |E(T)|$
Color all nodes white
for all leaf nodes v_{leaf} of T **do**
Solve subproblem $(T(v_{leaf}), 0)$
Color v_{leaf} black
end for
while T contains a white node v whose children are all black **do**
Solve subproblems $(T(v), l)$ for $l = 0, \ldots, \min\{k, |E(T(v))|\}$
Color v black
end while
Output: The values of the best l-cardinality trees in $T(0 \leq l \leq k)$, and the manipulated data structures for their efficient retrieval

where, $(T(v), l)$ is the subproblem of l-cardinality tree problem of subtree $T(v)$ rooted at v.

- $f_{-}(v, l)$ is the minimum objective function value of l-cardinality tree which does not contain v in subtree $T(v)$,
- $f_{+}(v, l)$is the minimum objective function value of l-cardinality tree which contains v in subtree $T(v)$.

For the leaf vertex, $f_-(v_{leaf}, 0) \leftarrow \infty$, $f_+(v_{leaf}, 0) \leftarrow w(v_{leaf})$. Otherwise,

$$f_-(v, l) \leftarrow \min\{f(v_i^c, l) | i = 1, \ldots, r\} \tag{2.1}$$

$$f_+(v, l) \leftarrow \min\left\{\sum_{i=1}^{r}(\delta(\alpha_i)(w(v, v_i^c) + f_+(v_i^c, \alpha_i) \mid \sum_{i=1}^{r}\alpha_i = l - r, \alpha_i \geq -1\right\} \tag{2.2}$$

where v^c means the child vertex of vertex v. Since it costs a lot of time to calculate $f_+(v, l)$, we solve it efficiently by calculating the following values:

$f_+(v, l) \leftarrow \min\left\{\sum_{i=1}^{2}(\delta(\alpha_i)(w(v, v_i^c) + f_+(v_i^c, \alpha_i))| \sum_{i=1}^{2}\alpha_i = l - r, \alpha_i \geq -1\right\}$
for $i = 3$ **to** r **do**
  for $l = 0$ **to** $\min\{k, \sum_{j=1}^{i}|E(T(v_j^c))| + 1\}$ **do**
    $f_+^{new}(v, l) \leftarrow \min\{f_+(v, \gamma) + (\delta(\beta_i)(w(v, v_i^c) + f_+(v_i^c, \beta_i)))|\gamma + \beta_i = l - 1, \gamma \geq 0, \beta_i \geq -1\}$
  end for
  $f_+(v, l) \leftarrow f_+^{new}(v, l)$
end for

Although the objective function value of kCTP can be calculated by $f = \min\{f_-(v_{root}, k), f_+(v_{root}, k)\}$, we calculate the following variables to find the tree itself.

- Boolean value $s(v, l)$ is TRUE if vertex v is the root of l-cardinality tree, else FALSE.
- Pointer $n(v, l)$ indicates which child of v is to move if $s(v, l)$ is FALSE.
- A set $c(v, l)$ of tuples of the form (v', t). v' is the child of v, and $t \leq |E(T(v'))|$ is an integer number that denotes the size of the subtree to be collected in $T(v')$.

In Function 2.2 , if $f_-(v, l) < f_+(v, l)$ we set $s(v, l) \leftarrow$ FALSE, $n(v, l) \leftarrow v_j^c$. Otherwise, we set $s(v, l) \leftarrow$ TRUE, $c(v, l) \leftarrow \{(v_i^c, \alpha_i^*)\}$. Let α_i^* be the values of α_i with which the minimum in Function 2.2 can be obtained.

Algorithm 8 Retrieval of an k-cardinality tree

Input: A rooted tree T with root node v_{root}, a cardinality k, and the manipulated data structures for tree retrieval

$v_c \leftarrow v_{root}$

$T_k \leftarrow$ empty tree

while $s(v_c, k) =$ FALSE **do**

$v_c \leftarrow n(v_c, k)$

end while

AddNext(v_c, k)

Output: The optimal k-cardinality tree T_k in T

AddNext()

Input: The current node v_c, and an integer number t indicating the size of the subtree to be retrieved from $T(v_c)$

for all tuples (v, r) in $c(v_c, t)$ **do**

AddNext(v, r)

Add v and (v_c, v) to T_k

end for

2.1.4 Experimental results and analysis

We use C as the programming language and compile programs with C-Compiler: Microsoft Visual C++ 2010 Express. Each algorithm above was tested 30 runs on a PC with Intel Core i7 2.8 GHz CPU and 6 GB RAM under Microsoft Windows 7. The aim of the experiment is to analyze the properties of those algorithms. Each algorithm is applied only once at each run.

We use two different kinds of graphs out of the well-known benchmark instances [1]. They are the grid graph with $|V| = 1089, |E| = 2112$ and the 4-regular graph with $|V| = 1000, |E| = 2000$. Both of them are middle size of benchmark instances in [1]. Each graph is applied with 5 cardinalities (k), accordingly we get 10 different instances. Since the range of k is from 1 to $|V| - 1$, cardinalities are chosen as about

20, 40, 60, 80 and 90 percents of $|V|$. That is because we want to study the possibly changing of each algorithm's performance over the whole range of cardinalities. On the other hand, by these different instances we can get results with much stronger persuasion.

In order to analyze the solution quality of each algorithm, the best, mean and worst objective function values of each algorithm in every run are obtained. The average computing time when the algorithm reaches the best solution is also recorded.

Based on these results in Tables 2.1 and 2.2, we provide much more intuitive figures (Figure 2.1) for analysis. We also conclude properties of each algorithm mainly based on the following 4 aspects.

Best objective function values Since the aim of each algorithm is to find the best solution of kCTP, best objective function value of each algorithm is compared. We introduce *relative error*, calculated by $(BS - BKS)/BKS$, to evaluate the error of each algorithm to a problem, where BS indicates the best objective function values of each algorithm, BKS indicates the value of best known solution given out in [1]. The *average* relative error of each algorithm to various problems is shown in Figure 2.1 [1]. The horizontal axis shows each algorithm and vertical axis shows the relative error. The smaller the relative error is, the higher precision an algorithm has.

Mean objective function values To evaluate the quality of solutions of each algorithm much more precisely, we also compared the mean objective function values of 30 runs. Similarly as described above, we use relative error, calculated by $(AS-BKS)/BKS$ to evaluate the solution quality of each algorithm. AS indicates the average objective function values of each instance obtained by that algorithm. The *mean* relative error of benchmark instances for each algorithm is shown in Figure 2.1 [2].

Standard deviation of objective function values Figure 2.1 [3] shows the average relative standard deviations of each algorithm to instances. It is calcu-

lated by $(DEV - BKS)/BKS$, where DEV means the standard deviation. If the standard deviation of an algorithm is small, algorithm has strong robustness. On the contrary, we can say that the algorithm has a character of high diversity if its deviation is large.

Mean computing time One of the most important pointers of an algorithm's performance is computing time. In our experiments, we recorded the mean computing time of 30 runs. Figure 2.1 [4] shows the mean computing time of each algorithm. The speed of each algorithm can be judged from this figure.

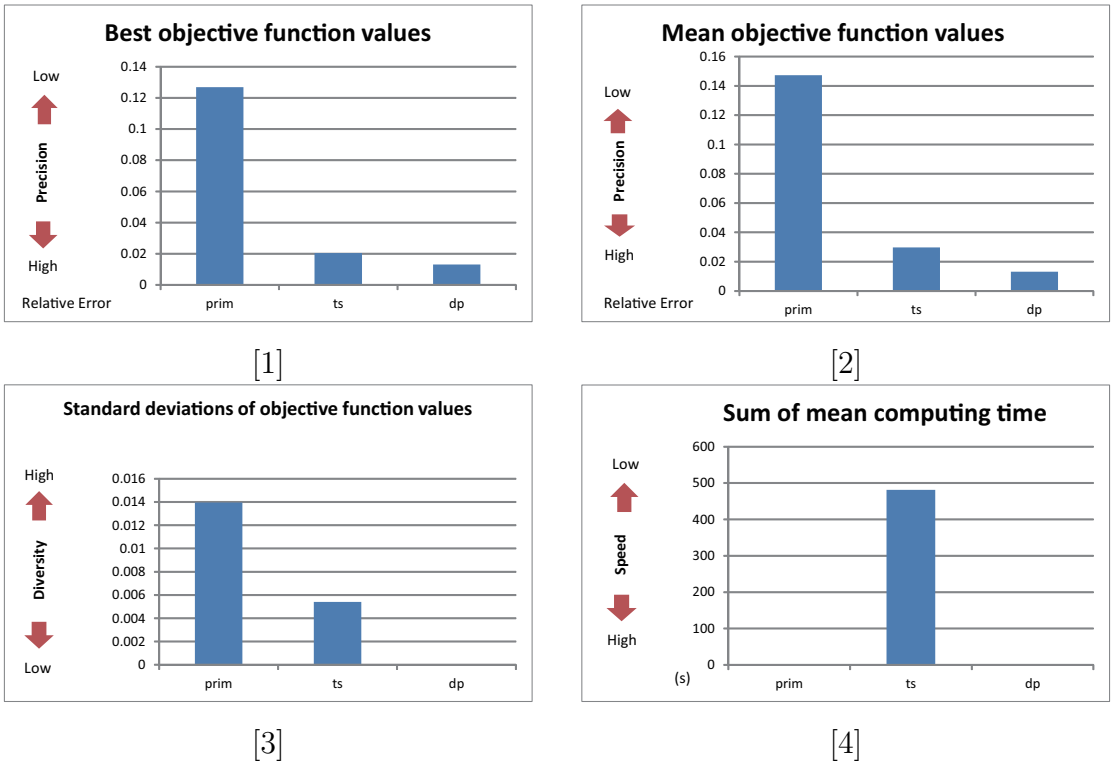

Figure 2.1: Results

Properties of each single algorithm are given as follows.

Prim's algorithm At first glance of Figure 2.1, the performance of Prim's algorithm (PRIM) is the worst one among three algorithms. It has the worst

precision and the largest standard deviation. However, the computing time of PRIM is very short. It shows that constructive heuristic is very fast, but it often returns solutions of low quality compared with the other two algorithms. Another merit of PRIM is that the implementation is very easy.

TS Figure 2.1[1][2] shows that TS is of medium solution quality among three algorithms, with relative error of 2% to the best know solution. We take it for granted that local search updates the initial solution generated by PRIM remarkably. The standard deviation of objective function values in 2.1[3] indicates TS could generates various solutions. It seems that if TS is given enough time, the possibility to reach the best solution is very high. However, we can see from Figure 2.1 [4] that TS has the longest computing time. To solve real-life problems, it is often desired to reach a high quality solution in a short time. We conclude that TS has a strong search ability but not efficient.

DP From Figure 2.1 [1] and [2], we can see that the solution quality of DP is the best among three algorithms, especially about average objective function values. Its high performance comes from the ability of exploring a large search space. The minimum spanning tree is constructed by connecting all vertices in the graph with edges which have small weights. Furthermore, Figure 2.1 [4] indicates that computing time of the DP is very short. That's due to little computing time of generating a minimum spanning tree, and DP has a very good performance. Figure 2.1 [3] shows that DP has the least robustness, and its standard deviation is nearly 0.

Strong robustness is usually considered as a good property for an algorithm. However, it may also be a disadvantage, especially when the algorithm can not reach the best known solutions at the first time. The solution may not be updated anymore no matter how long the search continues. To testify our hypothesis, we applied the following experiment. We applied PRIM to a benchmark problem, the 4-regular graph ($|V| = 1000, |E| = 2000$) with $k = 600$, to construct a k-cardinality trees starting at each vertex. Accordingly, we get 1000 trees and the frequencies of vertices and edges being used in each

tree. Comparing to the results of the best known solution, we found there are 6 edges that have never been used in those 1000 trees. In other words, no best known solution can be reached from these spanning trees. We conclude that DP can hardly reach the best known solution unless we get a spanning tree contains all edges contained in best known solution or apply a local search that could reach to a solution contains such edges.

Table 2.1: Results for instances with Grid graph[1]

Graph	k	BKS		PRIM	TS	DP
$\|V\| = 1089$	200	3303	Best	4245	3506	3400
$\|E\| = 2112$			Mean	4488.4	3595.2	3400
$d(v) = 3.87$			Worst	4793	3784	3400
(bb33x33-1.gg)			Time	0.005	9.639	0.042
	400	7070	Best	8778	7419	7276
			Mean	9031	7500.5	7276
			Worst	9843	7586	7276
			Time	0.093	72.197	0.109
	600	11579	Best	13931	12005	11798
			Mean	14020.0	12141.5	11798
			Worst	14309	12214	11798
			Time	0.016	64.424	0.080
	800	17393	Best	19187	17481	17436
			Mean	19240.1	17570.6	17436
			Worst	19453	17592	17436
			Time	0.019	91.552	0.148
	900	20919	Best	22292	20947	20926
			Mean	22452.8	21154.4	20926
			Worst	22519	21175	20926
			Time	0.019	49.721	0.100

Table 2.2: Results for instances with Regular graph [1]

Graph	k	BKS		PRIM	TS	DP
$\|V\| = 1000$	200	3308	Best	3817	3426	3432
$\|E\| = 2000$			Mean	3955.6	3477.3	3432.1
$d(v) = 4$			Worst	4192	3540	3433
(1000-4-01.g)			Time	0.002	8.569	0.024
	400	7581	Best	8364	7698	7653
			Mean	8543.5	7773.8	7657.7
			Worst	8712	7854	7658
			Time	0.004	42.450	0.046
	600	12708	Best	13670	12775	12789
			Mean	13780.2	12783.8	12789
			Worst	14016	12794	12789
			Time	0.007	52.932	0.066
	800	19023	Best	19566	19020	19076
			Mean	19595.5	19028.9	19076
			Worst	19639	19051	19076
			Time	0.010	80.054	0.083
	900	22827	Best	23160	22827	22830
			Mean	23160.2	22827	22830
			Worst	23167	22827	22830
			Time	0.013	34.078	0.091

2.2 Introduction to Bio-inspired algorithms

Biological inspired computations after principles that exist in natural systems. The approaches adopting such principles (e.g., using mechanism of pheromone evaporation,ACO could generates new solution with high precision and avoid the convergence to a local optimal solution) enable to reach solutions with enhanced robustness and flexibility and are expected to solve complex optimization problems efficiently.

In this section, we review some metaheuristics which are classified to Bio-inspired algorithms, such as evolutionary algorithms, ant colony optimization, and artificial immune systems. Bio-inspired algorithms can also be considered as heuristics reconstructed based on strategy learned from nature.

2.3 Basic concepts of Hybrid Metaheuristic

Referring to a dictionary, a hybrid is an animal or a plant that has been bred from two different species of animals or plants. In case of Hybrid Metaheuristic, it means a combination of metaheuristics or a metaheuristic with other operation research technique. For example, TS can be viewed as the hybrid of construction heuristic and Local Search based on strategy of "tabu". As we can see from the properties of algorithms introduced above, each algorithm has its advantage in solution quality or computing time. Moreover, the heuristic and metaheuristic are easier to implement than classical gradient-based techniques. By combining metaheuristic and other optimization technique based on their advantages, it is desirable to find good solutions in a significantly reduced amount of time. In other words, hybrid metaheuristics are good combination of metaheuristics. It has been proved to be much more efficient than a sole metaheuristic in recent years, and has higher flexibility when dealing with real-world and large-scale problems [39]. In the following chapters, we will introduce three hybrid metaheuristics proposed in this research. Experimental results show that those approaches are competitive with state-of-the-art methods on Blum's bench mark instances [1].

Chapter 3

Hybrid Metaheuristic Based on Tabu Search and Ant Colony Optimization

3.1 Introduction

In this chapter, a new hybrid metaheuristic based on Tabu Search (TS) and Ant Colony Optimization (ACO) is presented. The idea is based on approximate solution methods of Blum and Blesa [34]. They have proposed several metaheuristics such as evolutionary computation, ACO and TS for solving k cardinality tree problems (kCTP). They compared their performances through benchmark instances [1] and suggested that an ACO approach is the best one for relatively small ks, whereas a TS-based approach has an advantage for large ks with respect to solution accuracy.

In the proposed TS, the neighborhood structure is different from the Blum-Blesa one. While only the leaf vertices can be selected in the transition of the Blum-Blesa's algorithm, all of vertices adjacent to the current tree can be selected in the proposed algorithm. This extension enables us to strengthen the intensification ability of local search.

We also propose a diversification algorithm based on ACO by extending the Blum-Blesa algorithm. One of the characteristics in the proposed algorithm is that our algorithm deposits pheromone on the edges selected in the local optimal solutions which were obtained by the Tabu-Search-based local algorithm. This procedure

allows the proposed hybrid algorithm to explore a wider search space than the TS-based local search method. It generates new and diversified initial solution for TS.

To demonstrate efficiency of the proposed solution method, we compare the performances of the proposed method with those of existing algorithms using the well-known benchmark instances [1] that are easily accessible through the internet. The numerical experimental results imply that the proposed method has improved some of the best known solutions in very short computational time, and provides a better performance with the solution accuracy over existing algorithms.

3.2 Hybrid Metaheuristic Based on Tabu Search and Ant Colony Optimization

Since TS is an extension of local search, its intensification ability, which means the ability of robustly finding a very good local optimal solution in relatively narrow search space, is very high. However, the diversification capability of TS, which means the ability of exploring a wide solution space and covering the whole region to be searched, is relatively lower. This characteristic of tabu search reflects the fact that the TS-based algorithm by Blum and Blesa is the best for solving the benchmark instances of kCTP in the case of large ks.

On the other hand, ACO-based algorithm by Blum and Blesa can seek a very good solution for kCTP with small ks, which we think is caused by its high diversification capability. In fact, as will be described later in the experimental results, it is observed that ACO by Blum and Blesa often finds better "best" values than their TS method, and that some of the "best" values are even better than the proposed method. However, the objective function values obtained through ACO may quite-variable due to its lower intensification ability than TS. Realizing that TS and ACO are considered to be complementary to each other, we propose a hybrid algorithm that achieves a balance between the diversification and the intensification by incorporating the ideas of ACO into a TS algorithm.

The outline of the proposed algorithm is as follows:

Step 1 (Generation of an initial solution) For a vertex selected at random, the application of Prim method is continued until a k-subtree is constructed. Let the obtained k-subtree be an initial solution and the current solution T_k^{cur}.

Step 2 (Initialization of parameters) Initialize the tabu lists and the values of parameters such as tabu tenure tl_{ten} and aspiration criterion levels.

Step 3 (Tabu search-based local search procedure) Search the neighborhood based TS, and store a set of local minimum solutions. If the current tabu tenure tl_{ten} is greater than tt_{max}, go to Step 4. Otherwise, return to Step 2.

Step 4 (Ant colony optimization-based diversification procedure) Expand the exploration area based on ACO to increase the diversity of the solutions.

Step 5 (Terminal condition) If the current computational time is greater than $TimeLimit$, terminate the algorithm. Otherwise, return to Step 2.

Let T_k^{cur}, T_k^{gb} and T_k^{lb} be the current solution, the best found solution and local optimum solution, respectively. Then, we describe the details on the procedures in Steps 3 and 4.

3.2.1 Tabu search-based local search

In this section, we describe the details on the TS-based local search algorithm performed in Step 3.

For a set $V(T_k)$ of vertices included in k-subtree T_k, we define

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

Let T_k^{NH} be a local minimum solution of k-subtree obtained by adding $v_{in} \in V_{NH}(T_k)$ to T_k and deleting $v_{out} \in V(T_k)$. Then, the neighborhood of T_k denoted by $NH(T_k)$ is defined as a whole set of possible T_k^{NH} in G.

In the proposed local search algorithm, the next solution through transition is selected as the k-subtree that has the best objective function value of all solutions $T_k^{NH} \in N(T_k^{cur})$ as follows:

$$T_k^{NH_{best}} := \arg \min_{T_k^{NH} \in NH(T_k^{cur})} \{f(T_k^{NH})\}.$$

It should be stressed here that our neighborhood is different from the Blum-Blesa one. While only the leaf vertices can be selected in the transition of the Blum-Blesa's algorithm, all of vertices adjacent to the current tree can be selected in the proposed algorithm. This extension enables us to strengthen the intensification ability of local search, but the computational time for finding the best neighborhood solution may be longer. One of the promising approaches to decreasing the computational time is to incorporate the solution algorithm for minimum spanning tree (MST) problems, which has an advantage that it is solved in a polynomial time. However, a direct application of the MST algorithms by Prim and Kruskal is not efficient even if it is a polynomial-time algorithm because so large number of applying the MST algorithm is needed. Realizing such difficulty, we employ a more efficient method of obtaining the best neighborhood solution without applying the MST algorithm, which will be described later in the details of the algorithm. When using a local search algorithm, there is a problem of how to go out of a local optimal solution or how to avoid cycling among a set of some solutions. In order to resolve such a problem, we use two tabu lists *InList* and *OutList*, which keep the induces of removed edges and added edges, respectively. A tabu tenure, denoted by θ, is a period for which it forbids edges in the tabu lists from deleting or adding. In details, at the beginning, we set an initial value of the tabu tenure tl_{ten} to tt_{min} which is the minimum tabu tenure defined as

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

Let nic_{int} be the period of the best found solution T_k^{gb} not being updated. If $nic_{int} > nic_{max}$, then tabu tenure is updated as $tl_{ten} \leftarrow tl_{ten} + tt_{inc}$, where

$$nic_{max} := \max\{tt_{inc}, 100\}, \qquad tt_{inc} := \left\lfloor \frac{tt_{max} - tt_{min}}{10} \right\rfloor + 1.$$

If the current tabu tenure tl_{ten} is greater than tt_{max} defined as

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

the local search algorithm is terminated, and diversification strategy based on ACO is performed.

When checking whether the transition from the current solution to some solution in T_k^{NH} is acceptable, if an edge e in $InList$ or $OutList$, which is related to the transition as the added edge or deleted edge, satisfies the condition $\gamma_e > f(T_k^{NH})$, then the transition is permitted. The parameter γ_e called *aspiration criterion level* is given to all of edges and is initially set to

$$\gamma_e = \begin{cases} f(T_k^{cur}), & e \in E(T_k^{cur}) \\ \infty, & e \notin E(T_k^{cur}). \end{cases} \tag{3.1}$$

In each explored solution T_k, γ_e is updated as $\gamma_e \leftarrow f(T_k)$ for every $e \in E(T_k)$. The following are the details on the proposed local search algorithm.

[Tabu search-based local search algorithm]

Step 1 (Initialization of the list of a deleted vertex) Let $V_{in} \leftarrow V_{NH}(T_k^{cur})$.

Step 2 (Decision of a deleted vertex) If $V_{in} = \emptyset$, terminate the algorithm. Otherwise, go to Step 2-1.

Step 2-1 Find

$$v_{in} := \arg\min_{v \in V_{in}} \left\{ \frac{\sum_{v' \in V(T_k^{cur})} w(e)}{d(v)} \middle| e = (v, v') \right\}$$

and set $V_{in} \leftarrow V_{in} \backslash v_{in}$, where $d(v)$ is the number of edges existing between $v \in V$ and T_k^{cur}. Go to Step 2-2.

Step 2-2 Find $E_{in_1} := \{(v, v_{in}) \mid v \in V(T_k^{cur})\}$ (see Fig. 3.1) and $e_{\min_1} := \arg\min_{e \in E_{in_1}} \{w(e)\}$. Set $T_{k+1}^{NH} \leftarrow (V(T_k^{cur}) \cup v_{in}, E(T_k^{cur}) \cup e_{min_1})$ and $E_{in_1} \leftarrow E_{in_1} \backslash e_{\min_1}$ (see Fig. 3.2), and go to Step 2-3.

Step 2-3 Find $e_{\min_1} := \arg\min_{e \in E_{in_1}} \{w(e)\}$, and set $T_{k+1}^{NH} \leftarrow T_{k+1}^{NH} \cup e_{\min_1}$ and $E_{in_1} \leftarrow E_{in_1} \backslash e_{\min_1}$ (see Fig. 3.3). Go to Step 2-4.

Step 2-4 For a set E_{loop} of edges which compose a loop in Step 2-3, find $e_{\max} := \arg\max_{e \in E_{loop}} \{w(e)\}$ and set $T_{k+1}^{NH} \leftarrow T_{k+1}^{NH} \backslash e_{\max}$ (see Fig. 3.3).

Step 2-5 If $E_{in_1} = \{\emptyset\}$, then set $V_{out} \leftarrow V(T_k^{cur})$ and go to Step 3. Otherwise, return to Step 2-4.

Step 3 (Decision of an added vertex for constructing T_k^{NH}) If $V_{out} = \{\emptyset\}$, then return to Step 2. Otherwise, go to Step 3-1.

Step 3-1 Find

$$v_{out} := \arg \max_{v \in V_{out}} \left\{ \frac{\sum_{v' \in V(T_k^{cur})} w(e)}{d(v)} \middle| e = (v, v') \right\}$$

and set $V_{out} \leftarrow V_{out} \backslash v_{out}$. Go to Step 3-2.

Step 3-2 Find $e_{\min}^{out} := \arg\min_{e \in \{(v_{out}, v')\}} \{w(e) \mid v' \in T_k^{cur}\}$. If $f(T_k^{NH_{best}}) < \left(\sum_{e \in E(T_{k+1}^{NH})} w(e)\right) - w(e_{\min}^{out})$ for $e_{\min}^{out}$, then return to Step 3. Otherwise, go to Step 3-3.

Step 3-3 For a set of super-vertices S_r, $r = 0, 1, 2 \cdots$, each of which is a connected component obtained by deleting v_{in} from T_k^{NH}, find $E_{in_2} := \{(v_i, v_j) \mid v_i \in S_k, v_j \in S_l, k \neq l\}$ (see Fig. 3.4). Go to Step 3-4.

Step 3-4 Find $e_{\min_2} := \arg\min_{e \in E_{in_2}} \{w(e)\}$. If $f(T_k^{NH_{best}}) < w(e_{\min_2}) + \sum_{e \in E(T_k^{NH})} w(e)$ for $e_{\min_2}$, then return to Step 3. Otherwise, go to Step 3-5.

Step 3-5 If there is no loop in $e_{\min_2} \cup T_k^{NH}$, then set $E(T_k^{NH}) \leftarrow E(T_k^{NH}) \cup e_{\min_2}$ and $E_{in_2} \leftarrow E_{in_2} \backslash e_{\min_2}$. Otherwise, set $E_{in_2} \leftarrow E_{in_2} \backslash e_{\min_2}$. Go to Step 3-6.

Step 3-6 If T_k^{NH} is a tree (see Fig. 3.5), then set $f(T_k^{NH_{best}}) \leftarrow f(T_k^{cur})$ and return to Step 3. Otherwise, return to Step 3-4.

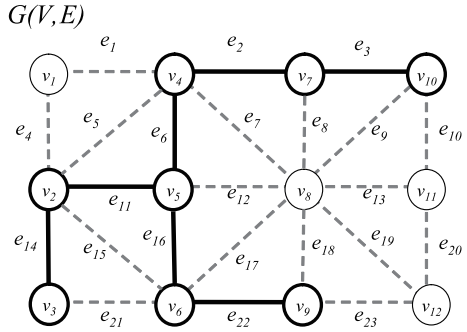

Figure 3.1: Current solution (when $v_{in} = v_8$ and $E_{in_1} = \{e_7, e_8, e_9, e_{12}, e_{17}, e_{18}\}$)

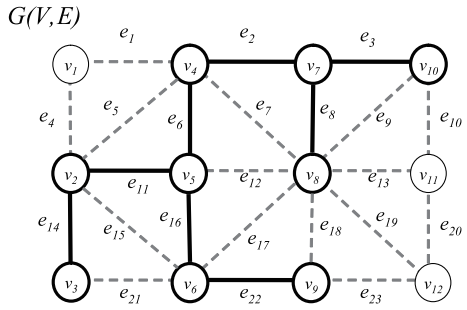

Figure 3.2: $k+1$-subtree T_{k+1}^{NH} (e_8 is added to the current solution)

As mentioned before, the most important feature of the proposed tabu search-based local search algorithm is to obtain a minimum spanning tree for every subgraph without applying the MST algorithm. This means that every solution obtained in each iteration must be a local optimal solution of kCTP because the minimum spanning tree for each subgraph obtained is exactly the best solution among neighborhood. Step 2 suggests how to select a deleted vertex, and Step 3 demonstrates how to choose an added vertex. Through the combination of Steps 2 and 3, a minimum spanning tree for each subgraph, which is the best neighborhood solution, can be obtained without using a MST algorithm iteratively.

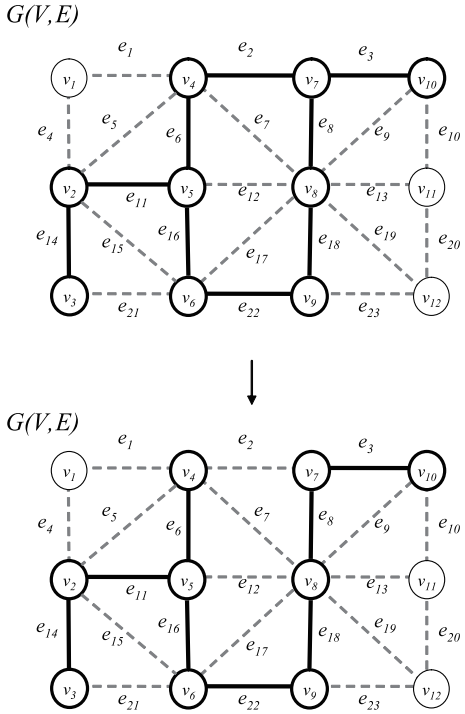

Figure 3.3: Improvement of $k+1$-subtree T_{k+1}^{NH} (e_{18} is added, and then e_2 is deleted so that a new $k+1$-subtree T_{k+1}^{NH} is constructed)

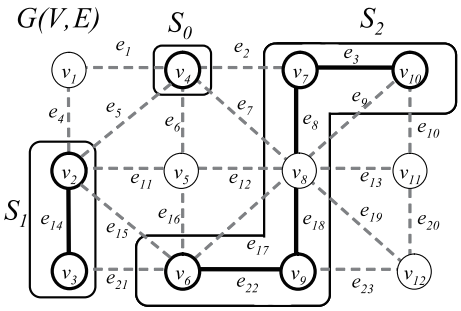

Figure 3.4: Set of super-vertices

3.2.2 Ant colony optimization-based diversification procedure

This section devotes describing the details on the ACO-based diversification procedure performed in Step 4.

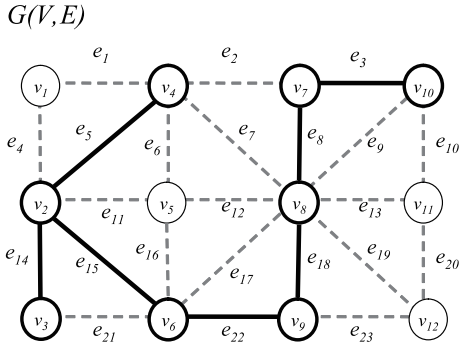

Figure 3.5: Solution T_k^{NH} in neighborhood $NH(T_k)$

ACO which was initiated by Dorigo *et al.* [13, 16] is a metaheuristic approach for solving hard combinatorial optimization problems. The behavior is based on a cooperative interaction which leads to the emergence of shortest paths by depositing a substance, called *pheromone*, on the ground so as to minimize the length of the path between nest and food source. Blum and Blesa proposed an ACO-based algorithm for solving kCTP and showed that the performance of the ACO is better than those of TS and EC in the case of small k. When the cardinality k is small, the candidate for the optimal solution exist in wider solution space compared to the case of large k. With this observation in mind, in this paper, we propose a diversification algorithm based on ACO by extending the Blum-Blesa algorithm [34]. One of the characteristics in the proposed algorithm is that our algorithm deposits pheromone on the edges selected in the local optimal solutions which were obtained by the TS-based local algorithm. This procedure allows the proposed hybrid algorithm to explore a wider search space than the TS-based local search method.

Diversification algorithm based on ant colony optimization

Step 1 (Setting of learning rate) Set the learning rate of each solution in E_{lb} to the value defined by

$$\rho = \begin{cases} 0.15, & cf < 0.7 \\ 0.1, & 0.7 \leq cf \leq 0.95 \\ 0.05, & cf > 0.95, \end{cases}$$

where cf is a convergence factor defined by

$$cf \leftarrow \frac{\sum_{e \in E_{lb}} \tau_e}{|E_{lb}| \cdot \tau_{max}}.$$

Step 2 (Update of pheromone) Update the amount of pheromone assigned to each edge e as follows:

$$\tau_e = f_{mmas}(\tau_e + \rho(\delta_e - \tau_e))$$

where

$$f_{mmas}(x) = \begin{cases} \tau_{min}, & x < \tau_{min} \\ x, & \tau_{min} \leq x \leq \tau_{max} \\ \tau_{max}, & x > \tau_{max} \end{cases}, \quad \delta_e = \begin{cases} 1, & e \in E_{lb} \\ 0, & e \notin E_{lb}. \end{cases}$$

Step 3 (Generation of a k-subtree) Replace the weight attached to each edge e in G by $w_d(e)$ defined as

$$w_d(e) \leftarrow \frac{w(e)}{\tau_e}.$$

Starting from a randomly selected vertex, a k-subtree T_k is constructed by applying the Prim method. After that, replace $w_d(e)$ by the original weight $w(e)$, and construct a k-subtree T_k^{cur} by applying the Prim method again to the subgraph of which vertices and edges are $V(T_k)$ and $E(T_k)$, respectively.

In this paper, we set the initial values of τ_e, the values of τ_{min} and τ_{max} to 0.5, 0.001 and 0.999, respectively.

3.3 Numerical experiments

In order to compare the performances of our method with two of existing solution algorithms proposed by Blum and Blesa [34]. 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.

The parameter settings as well as the source code used for all the experiments in this paper are just the same ones provided by Blum and Blesa, which was downloaded from their web-site page [1] through the internet. [1] Tables 3.1-3.4 and **??** show the results for several existing instances [1] and our new instances, respectively. Bold face means that it is the best obtained value among the three algorithms to be compared. In Tables 3.1-3.4, BKS denotes the best known values which have been obtained by Blum and Blesa through their tremendous experiment for several months. The values with $*$ denotes new best known values that are updated by the proposed algorithm. TSACO, TSB and ACOB represent the proposed algorithm, TS algorithm [34] and ACO algorithm [34] by Blum and Blesa, respectively. We executed each method for 30 runs under the condition that $TimeLimit = 300(s)$ and computed the *Best*, *Mean* and *Worst* objective function values for each method. We describe '$-$' in the tables when the algorithms do not derive solutions within the given time limit.

Tables 3.1-3.4 show that the performance of the proposed method is better than those of the existing algorithms by Blum and Blesa, especially in the case of high cardinality k and high degree $\bar{d}(v)$. This is due to the effect of the subroutine through which a minimum spanning tree (a local optimal solution for a subgraph) is obtained and updated for a new subgraph constructed by swapping a pair of vertices. As a special case, when $k = |V| - 1$ which is the maximum value of k available, the corresponding $(|V| - 1)$-minimum spanning tree problem is equivalent to a conventional (ordinal) minimum spanning tree problem that can be exactly solved by the proposed algorithm. In the case of very large ks, even when k is not the maximum value $(= |V| - 1)$, the performance of the proposed algorithm is also very high. In addition, the effect of finding a MST for every subgraph increases

[1] Although the source code for Blum-Blesa's algorithm could be freely downloaded from their web-site page before, it can be now obtained only by directly contacting with the authors by e-mail.

with increasing the graph density. Table **??** shows that the proposed method is the best for instances of graphs with higher degrees than existing ones. From the aspect of the real applications of kCTP, we can suggest that if the number of sites (or facilities) to be connected in the problem of telecommunications (or facility layout) is large, the proposed algorithm is recommended. In addition, if the number of lines available to be used for connecting sites or facilities is large, the proposed algorithm is also promising.

On the other hand, when k is small, the performance of the proposed algorithm is not so good, and the performance of the ACOB is the best. This is because the candidates of an optimal solution exist over a wide rage when k is small. Since the effect of TS is stronger than that of ACO in the proposed hybrid algorithm, the diversification ability of the proposed method is lower than (non-hybrid) ACO-based algorithm such as ACOB.

It should be stressed here that as shown in Table 3.1, the proposed method updates some of best known values despite of very short computational time limit (300s), while the time limits in the experiments by Blum and Blesa are fairly large, at most several hours. From these experimental results, we can conclude that the proposed algorithm is considerably promising for solving kCTPs.

3.4 Conclusion

In this chapter, a new hybrid approximate solution algorithm for kCTP by combining tabu search and ant colony optimization is proposed. Through numerical experiments for several benchmark instances, we have shown that the performances of the proposed method are better than those of existing methods. Furthermore, it has been shown that the proposed method updates some of the best known values.

Table 3.1: Results for instances with Grid graphs [1]

Graph	k	BKS		TSACO	TSB	ACOB
$\|V\| = 225$	40	695	Best	**695**	696	**695**
$\|E\| = 400$			Mean	**695**	696	695.4
$\bar{d}(v) = 3.55$			Worst	**695**	696	696
(bb45x5_1.gg)	80	*1552	Best	**1552**	1579	1572
		(1568)	Mean	**1565.1**	1592.7	1581.2
			Worst	**1572**	1615	1593
	120	*2444	Best	**2444**	2546	2457
		(2450)	Mean	**2457.9**	2558.5	2520.3
			Worst	**2465**	2575	2601
	160	*3688	Best	**3688**	3724	3700
		(3702)	Mean	**3688**	3724.9	3704.7
			Worst	**3688**	3729	3720
	200	5461	Best	**5461**	5462	**5461**
			Mean	**5461**	5462.4	5469
			Worst	**5461**	5463	5485
$\|V\| = 225$	40	654	Best	**654**	**654**	**654**
$\|E\| = 400$			Mean	**654**	**654**	**654**
$\bar{d}(v) = 3.55$			Worst	**654**	**654**	**654**
(bb45x_5_2.gg)	80	1617	Best	**1617**	**1617**	**1617**
			Mean	1619.1	**1617.1**	1626.9
			Worst	1620	**1619**	1659
	120	*2632	Best	**2632**	2651	2637
		(2633)	Mean	**2641.3**	2677.9	2664.6
			Worst	**2648**	2719	2706
	160	3757	Best	**3757**	3815	**3757**
			Mean	**3764.3**	3815.0	3797.6
			Worst	**3779**	3815	3846
	200	5262	Best	**5262**	**5262**	**5262**
			Mean	**5262**	5268.6	5272
			Worst	**5262**	5296	5288

Table 3.2: Results for instances with Regular graphs [1]

Graph	k	BKS		TSACO	TSB	ACOB
$\|V\| = 1000$	200	3308	Best	3393	3438	**3312**
$\|E\| = 2000$			Mean	3453.1	3461.4	**3344.1**
$d(v) = 4$			Worst	3517	3517	**3379**
(1000-4-01.g)	400	7581	Best	**7659**	7712	7661
			Mean	7764	7780.2	**7703**
			Worst	7819	7825	**7751**
	600	12708	Best	**12785**	12801	12989
			Mean	12836.6	**12821.8**	13115.6
			Worst	13048	**12869**	13199
	800	19023	Best	19099	**19093**	19581
			Mean	**19101.1**	19112.6	19718.7
			Worst	**19128**	19135	19846
	900	22827	Best	**22827**	22843	23487
			Mean	**22827**	22859.2	23643
			Worst	**22827**	22886	23739
$\|V\| = 1000$	200	3620	Best	3667	3692	**3632**
$\|E\| = 2000$			Mean	3697.5	3722.0	**3670.1**
$d(v) = 4$			Worst	3738	3751	**3710**
(g400-4-05.g)	400	8206	Best	**8323**	8358	8376
			Mean	**8357.1**	8385.6	8408.3
			Worst	**8424**	8415	8442
	600	13584	Best	13807	**13735**	14085
			Mean	13824.3	**13759.4**	14164.5
			Worst	13900	**13820**	14235
	800	20076	Best	**20110**	20130	20661
			Mean	**20129.9**	20142.9	20811.3
			Worst	**20143**	20155	20940
	900	24029	Best	**24035**	24044	24782
			Mean	**24035**	24052.6	24916
			Worst	**24035**	24064	25037

Table 3.3: Results for instances constructed from Steiner tree problems [1]

Graph	k	BKS		TSACO	TSB	ACOB
$\|V\| = 1000$	200	1018	Best	**1034**	1036	1036
$\|E\| = 5000$			Mean	1048.6	1047.3	**1045.9**
$\bar{d}(v) = 10.0$			Worst	1063	**1056**	**1056**
(steind15.g)	400	2446	Best	**2469**	2493	2665
			Mean	**2480.7**	2502.5	2806.6
			Worst	**2492**	2524	2928
	600	4420	Best	**4426**	4442	5028
			Mean	**4433**	4454.6	5398.4
			Worst	**4451**	4490	5602
	800	7236	Best	**7236**	7252	8457
			Mean	**7236.9**	7272.8	8839.6
			Worst	**7237**	7308	9006
	900	9248	Best	**9256**	9283	10873
			Mean	**9256**	9294.2	11166.3
			Worst	**9256**	9304	11423

Table 3.4: Results for instances constructed from graph coloring problems [1]

Graph	k	BKS		TSACO	TSB	ACOB
$\|V\| = 450$	90	135	Best	**135**	**135**	**135**
$\|E\| = 8168$			Mean	**135.1**	135.3	135.7
$\bar{d}(v) = 36.30$			Worst	137	**136**	137
(le450_15a.g)	180	336	Best	**336**	337	352
			Mean	**337**	337.1	374.4
			Worst	**337**	338	419
	270	630	Best	**630**	630	696
			Mean	**630.1**	630.3	839
			Worst	**631**	633	913
	360	1060	Best	**1060**	**1060**	1267
			Mean	**1060**	1064.1	1461.2
			Worst	**1060**	1118	1566
	405	1388	Best	**1388**	**1388**	1767
			Mean	**1388**	1391.1	1888.7
			Worst	**1388**	1392	2015

Table 3.5: Results for new instances

Graph	k		TSACO	TSB	ACOB
$\|V\| = 500$	100	Best	**1943**	1954	**1943**
$\|E\| = 15000$		Mean	**1950.5**	1990.9	2022.3
$\bar{d}(v) = 60$		Worst	**1966**	2023	2241
	200	Best	**5037**	5063	5517
		Mean	**5047.3**	5080.4	7444.4
		Worst	**5066**	5221	9859
	300	Best	**9758**	9821	-
		Mean	**9769.6**	9922.6	-
		Worst	**9795**	11696	-
	400	Best	**16351**	16373	-
		Mean	**16363.8**	16488	-
		Worst	**16368**	17953	-
	450	Best	**20929**	20934	-
		Mean	**20929**	20945.2	-
		Worst	**20929**	20992	-
$\|V\| = 500$	100	Best	**1294**	1319	1398
$\|E\| = 30000$		Mean	**1303.7**	1352.8	1743.5
$\bar{d}(v) = 120$		Worst	**1321**	1385	2479
	200	Best	**3064**	3150	4013
		Mean	**3097.1**	3934.4	6861.4
		Worst	**3127**	6032	9623
	300	Best	**5312**	5380	-
		Mean	**5312.5**	6471.9	-
		Worst	**5318**	8308	-
	400	Best	**8582**	8586	-
		Mean	**8582**	9540	-
		Worst	**8582**	11485	-
	450	Best	**10881**	10882	-
		Mean	**10881**	11300.4	-
		Worst	**10881**	13570	-

Chapter 4

Hybrid Metaheuristic Combining Tabu Search with Immune Algorithm

4.1 Introduction

In this chapter, we focus on a Hybrid Metaheuristic based on Tabu Search (TS) and Immune Algorithm(IA). Since TS stops when the length of tabu list reaches its limitation, IA is applied to enlarge the search area by generating a new initial solution for TS. In other words, IA acts as a diversitification strategy for TS, and thus the algorithm can search the solution space with a large step size.

The proposed immune algorithm is inspired by immune systems, especially the mechanism of keeping diversity of the immune cells. More specifically, population of antibodies are constructed with a variety of infeasible solutions. At each step the population-based algorithm deals with a set of infeasible solutions rather than with a single one, providing a natural and intrinsic way to explore the search space.

Experimental results show that IA improves the solution accuracy significantly. Some best known solutions are updated by the proposed algorithm. Another feature of proposed algorithm is that it's speed is relatively higher than existing algorithms. We arrive at a conclusion that a well-designed hybrid metaheuristic algorithm is efficient for solving the k-Cardinality Tree Problem.

4.2 Tabu search incorporated with Immune Algorithm

Recall that TS, firstly proposed by Glover *et al.* [2], is one of the mostly used metaheuristics for solving combinatorial optimization problems. In short, TS aims to improve a given solution by modified local search. The most important characteristic of TS is that it uses a concept of *memory* to control movements via a dynamic list of forbidden movements. To be more specific, the solutions which have been searched will be "tabu" (forbidden) from visiting for a while. This mechanism allows TS to intensify or diversify its search procedure in order to escape from local optima. However, when the size of the problem is very large, the length of tabu list can not be increased unlimitedly. The computation time will increase sharply with the length of tabu list. Therefore, in this paper the maximum length of tabu list is limited. TS stops when the length of tabu list reaches the limitation. Then IA is applied to generate a new initial solution to enlarge the search area. Without loss of generality, the flowchart of the proposed algorithm is shown in Figure 4.1, where T^{lb} means local solution and T^{cur} means current solution.

4.2.1 Generate the Initial solution

Given a connected graph $G = (V, E)$ with a weight function $w : E \to R$, we wish to find a k-cardinality tree from G as the initial solution. The algorithm we consider here uses a greedy heuristic (Prim's algorithm) to generate a minimum spanning tree from G. To be specific, all the vertices of V in the graph are connected to be a spanning tree T^{SP} by Prim's algorithm. Then dynamic programming algorithm (DP), originally introduced in [38], is applied to find out the best k-cardinality tree from the T^{SP}. DP has been proved to be able to construct an initial solution with a small objective function value in a short computing time even for a problem with a large size graph [38]. Experimental results show that solutions generated DP have high precision.

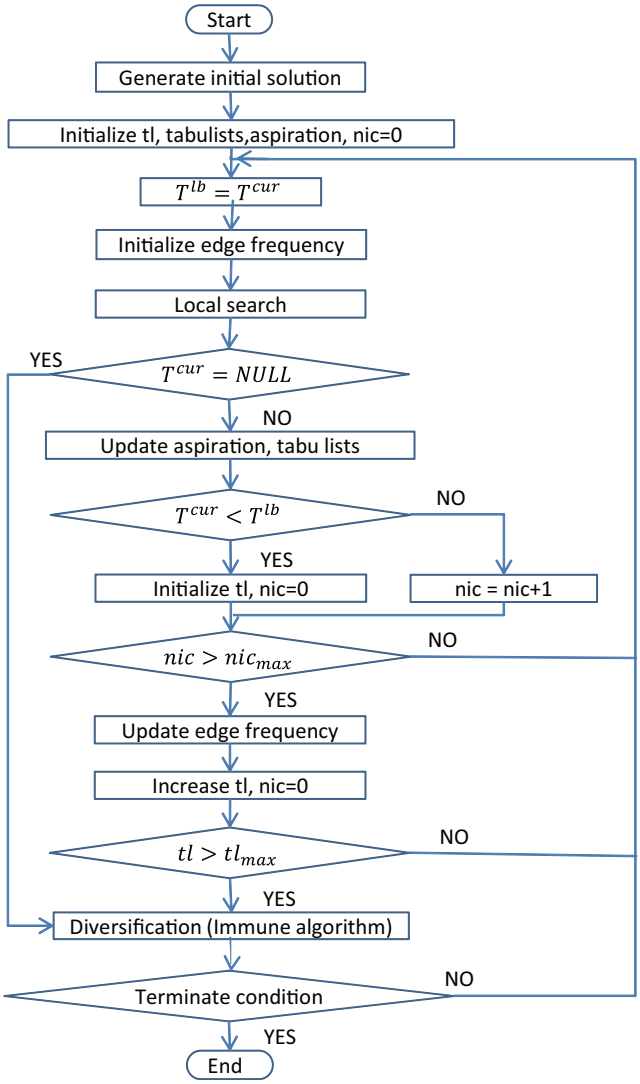

Figure 4.1: Proposed Algorithm TS

4.2.2 Length of Tabu List

The core procedure of TS is to forbid some moves based on memory in order to enlarge the search area. In the proposed algorithm, the "tabu" (forbiddance) is applied to edges that have been added to or deleted from the k-cardinality tree recently. Tabu lists are used as a memory to record edges that should be forbidden to be added or deleted. $InList$ and $OutList$ are adopted to keep the records of removed edges and added edges, respectively. Tabu tenure, which generally depends on the

length of tabu lists, is a period during which it forbids edges in the tabu lists from being added or deleted. The length of tabu lists is dynamically changed in the proposed algorithm. By adjusting the length of tabu list local search can implement intensification and diversification strategies. If the best solution in an 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 $InList$ or $OutList$ reaches tl_{max}. Some parameters can be referred to Chapter 3.

4.2.3 Aspiration Criterion

The "tabu" mechanism, which forbids some of the moves to be employed, helps the algorithm avoid falling into local optima. However, this mechanism may also forbid a move that may reach the best solution. In order to avoid such a situation, a procedure called *aspiration criterion* is used in the proposed algorithm. That is, if $f(T_k^{new}) < \gamma_e$ is satisfied, the movement will be acceptable even if edge e is included in $InList$ or $OutList$. Parameters γ_e called *aspiration level criterion* are given to all of edges and are initially set to be:

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

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

4.2.4 Local Search

The basic ingredient of TS is local search. Local search is often conducted via some move operators. A move from the current solution to the candidate solution is only performed when the objective function value is improved. In this study, we propose an efficient local search for solving the kCTP. The basic idea is to translate the current solution T_k to a new one, by exchanging one vertex in T_k with a vertex not in it. Correspondingly, the edges which connect those vertices in T_k should also be updated.

In detail, we define that $V(T_k)$ denotes the vertex set of tree T_k. Neighbourhood vertex set of T_k was defined as: $V_{ADD}(T_k) := \{v|(v,u) \in E, v \notin V(T_k), u \in V(T_k)\}$. Edge (v,u) is called a connecting edge. $V_{RMV}(T_k)$ denotes the the vertex set of tree T_k, which would be removed from T_k after constructing a T_{k+1}.

The procedure of the local search is showed as follows:

while $V_{ADD}(T_k) \neq \emptyset$ **do**

Growing T_{k+1}

while $V_{RMV}(T_k) \neq \emptyset$ **do**

Reconstructing T_k^{new}

Updating $T_k^{localbest}$

end while

end while

Updating T_k,

where T_k^{new} is the newly constructed k-cardinality tree.

Growing T_{k+1} : A tree T_{k+1} is constructed by adding vertex $v_{add} \in V_{ADD}(T_k)$ as well as its least weight edge to tree T_k. At the same time, v_{add} is deleted from the neighbourhood vertex set $V_{ADD}(T_k)$.

Reconstructing T_k^{new} : To reconstruct a new k-cardinality tree T_k^{new}, one vertex called v_{rmv} should be removed from T_{k+1}. Correspondingly, the edge (or edges) connecting to v_{rmv} is (are) also removed from T_{k+1}. If the v_{rmv} is a leaf vertex (it connects to the tree by only one edge), we obtain a new k-cardinality tree without further procedure. If a set of the remaining vertices and edges becomes a forest, Kruskal's Aglorithm would be applied to connect the forest into a tree T_k^{new}. Then, v_{rmv} will be deleted from the neighbourhood vertex

set $V_{RMV}(T_k)$.

Updating $T_k^{localbest}$: If T_k^{new} could be constructed successfully and is better than the local best k-cardinality tree $T_k^{localbest}$, the latter should be updated by T_k^{new}.

Updating T_k : If the objective function value of local best tree $T_k^{localbest}$ is better than current tree T_k, the latter should be updated by $T_k^{localbest}$.

Some parameters and how to select the "necessary" vertices can be referred to chapter 3.

4.3 Details of Immune Algorithm

As mentioned above, for a problem with a large size graph, the length of tabu list may become very long in order to enlarge its search area. Accordingly, the computing time may expand significantly with size of the graph. IA is adopted as a diversification strategy for further search when the length of tabu list reaches tl_{max}.

Immune system has a powerful ability to protect our bodies 24 hours a day against attacks from antigens (i.e., viruses and bacteria). Firstly, it recognizes and eliminates invading antigens by producing antibodies. For antigens which have been eliminated before, long lasting cells (also called memory cells) remember the first exposure and respond very quickly to the second exposure. Secondly, for an antigen firstly met, the system can eliminate it with antibodies secreted by a new immune cell which is usually generated by mutation. Finally, one of the beautiful things about immune system is that immune cells are able to produce antibodies against hundreds of thousands of different pathogens that the body will come into contact with in a lifetime. It benefits from a mechanism that uses suppressor cells to restrain the excessive proliferation of antibodies and maintains the diversity of immune cells.

The proposed immune algorithm is inspired by immune systems, especially the mechanism for keeping diversity of the immune cells. More specifically, population

of antibodies are constructed with a variety of infeasible solutions. At each step the population-based algorithm deals with a set of infeasible solution rather than with a single one, providing a natural and intrinsic way to explore the search space. Representations of the biological immune system are shown as follows: antigens are defined as kCTPs; antibodies are considered as potential solutions to kCTPs. The pseudo-code of the proposed IA is shown as follows:

Outline of the Immune Algorithm

Input: set A (antibodies) and set S (suppressors), fq (edge-use-frequency)

if $A = \emptyset$ **then**

$A \leftarrow$Generate Initial Population(N_A)

end if

for i=1 to 10 **do**

for each a $\in$ A **do**

$c =$ Concentration (a)

if $c > \theta$ **then**

Update Suppressor(S, a)

end if

end for

for each a $\in$ A, s $\in$ S **do**

if $a = s$ **then**

$A \leftarrow (A \setminus \{a\}) \cup$ Generate Solution()

end if

end for

$A' \leftarrow \emptyset$

while $|A'| < N_A$**do**

$a_1 \leftarrow$ Roulette Selection(A)

$a_2 \leftarrow$ Roulette Selection(A)

$A' \leftarrow A' \cup \{$ Apply Crossover$(a_1, a_2)\}$

end while

$A \leftarrow A'$

end for

Output: T_k^{gb}

where the parameters is defined as follows:

number of antibodies: $N_A = 100$

number of suppressor: $N_S = 5$

threshold value: $\theta = 0.5$

crossover probability: $p_c = 0.5$.

Generate initial population Initial population consists of feasible solutions (antibodies) constructed by a greedy heuristic algorithm. To generate a k-cardinality tree, one vertex is selected randomly, then edges and vertices are added to following Prim's algorithm until the number of edges reaches k. The size of the population is N_A.

Concentration(a) The concentration is used to measure the proportion of a kind of antibody in population. The concentration is calculated as follows:

$$Concentration(a) = \frac{\sum_{a_i \in A} \delta_{a_i,a}}{N_A}$$

$$\delta_{a_i,a} = \begin{cases} 1, \text{ if } a_i = a \\ 0, \text{ if } a_i \neq a. \end{cases}$$

Update suppressor If the concentration of an antibody is larger than threshold θ, we say this kind of antibody has been over proliferated. To keep the diversity of antibodies in population, we add the proliferated antibody to a set S of suppressor. Antibodies in S should be restrained. Incidentally, if $|S| = N_S$, $\arg\min(\text{Hamming\_distance}(a, s_i))$, $s_i \in S$ is used to determine which antibody should be added to S.

Generate Solution The antibody in suppressor set S should removed from the population. The space should be filled up with new antibodies generated in the same way as generating initial population.

Roulette selection As mentioned above, new antibodies are generated by crossover in the proposed algorithm. Antibodies are selected with probability p_a from population A for generating offspring. In particular, p_a of each antibody is calculated as follows:

$$p_a = \frac{\prod_{s \in S} \delta_{a,s}}{1 + f_a}$$

$$\delta_{a,s} = \begin{cases} 0.1, & a \in G_{NH}(s) \\ 1, & \text{otherwise} \end{cases}$$

$$G_{NH}(s) = \{g \in B^{|E|} | \text{Hamming\_distance}(g, s) < \lfloor |E|/50 \rfloor \}.$$

As we can see from these functions, an infeasible solution with a smaller object function value and "far away from" suppressors has a higher probability to be selected.

Apply Crossover *Crossover* is applied to generate new immune cells which secrete necessary antibodies. In consideration of both continuity and diversity, antibodies selected by *Rouletteselection* are applied with crossover with probability p_c for generating offspring. With probability $(1 - p_c)$, the selected antibodies are regarded as offspring directly.

The crossover is the same as the one introduced by Blum *et.al* for EC approach in [34]. It applies two complementary heuristically guided crossover operators, Union-crossover and Intersection-crossover, to the crossover partners T_k and T_k^p. We present the crossover here only for the sake of completeness. The pseudo-code of crossover is shown in the followings:

Framework for U-crossover and I-crossover[34]

Input: Two k-cardinality trees T_k and T_k^p

$E_\cup \leftarrow E(T_k) \cup E(T_k^p)$

$E_\cap \leftarrow E(T_k) \cap E(T_k^p)$

$t \leftarrow 1$

$E(T_t^{child}) \leftarrow \{\arg\min\{w(e = \{v, v'\})|e \in E_\cap\}\}$

$V(T_t^{child}) \leftarrow \{v, v'\}$

repeat

Choose set E_C in follow ways:

U-crossover: $E_C \leftarrow E_{NH}(T_t^{child}) \cap (E_\cup \setminus E_\cap)$;

I-crossover: $E_C \leftarrow E_{NH}(T_t^{child}) \cap E_\cap$;

if $E_C = \emptyset$ **then**

$e = \{v, v'\} \leftarrow$

$\arg\min\{w(e)|e \in E_{NH}(T_t^{child}) \cap E_\cup\}$

else

$e = \{v, v'\} \leftarrow \arg\min\{w(e)|e \in E_C\}$

end if

$E(T_t^{child}) \leftarrow E(T_t^{child}) \cup \{e\}$

$V(T_t^{child}) \leftarrow V(T_t^{child}) \cup \{v, v'\}$

$t \leftarrow t + 1$

until $|E(T_k^{child})| = k$

Output: T_k^{child}

where the $E_{NH}(T)$ is defined as follows:

$$E_{NH}(T) = \{e = (v, v') \in E(G) | v \in V(T), v' \notin V(T).\} \tag{4.1}$$

Centralization strategy based on edge frequency Besides the diversitification strategy of antibodies, centralization strategy is also considered. The basic idea is that edges that have been selected to construct antibodies at high frequencies are considered as good elements for generating the best k-cardinality tree. With this consideration in mind, the frequency of each edge f_e is used as a parameter when generating new solutions. "New weight" $\bar{w}_e$ for each edge is fixed based on f_e as follows: $\bar{w}_e = \frac{w_e}{f_e}$. "New weights" are applied in generate population and crossover. In particular, the initial value of f_e is defined as $f \leftarrow 0.5$, and then updated in each local search as follows:

$$f_e \leftarrow f_e + \rho(\delta - f_e)$$
$$\delta = \begin{cases} 1, & e \in E(T_k^{cur}) \\ 0, & e \notin E(T_k^{cur}) \end{cases}$$

where $E(T_k^{cur})$ is the edge set of the current solution. The value of ρ is given as 0.1 in the proposed algorithm.

4.4 Experiments

Two sets of experiments have been carried out for evaluating the efficiency of the proposed hybrid metaheuristic based on TS and IA (TSIA). Firstly we measured the efficiency of IA by comparing the results of algorithms with and without IA. Secondly, TSIA and three state-of-the-art existing algorithms were also comprehensively compared.

4.4.1 Efficiency of Immune Algorithm

To investigate the efficiency of IA, we compared TSIA with TS in experimental results. We use C as the programming language and compile the program with C-Compiler: Microsoft Visual C++ 2010 Express. The experiment was carried out in famous benchmark instances (bb15x15_1.gg, bb45x5_1.gg, g400-4-01.g, steinc5.g, steinc15.g, le450_15a.g, bb33x33_1.gg, bb100x10_1.gg, steind5.g, steind15.g, bb50x50_1.gg, steine5.g) [1]. Sizes of those instances vary from $|V| = 225$, $|E| = 400$ to $|V| = 2500, |E| = 4900$, where $|V|$ means the number of vertices and $|E|$ means the number of edges. Different instances based on each graph are generated by changing capacities (the value of k). Cardinalities are chosen as about 10, 20, 30, 40, 50, 60, 70, 80, 90 and 95 percents of $|V|$, so that we obtain 121 different instances. We expect results with much stronger persuasion by these differential instances. The limited running time for graphs with less than 1000 , between 1000 and 2000 and more than 2000 vertices is 100s, 300s and 900s, respectively.

Each instance is tested 30 runs on a PC with Intel Core i7 2.93 GHz CPU (the multiprocessor did not process in parallel) and 4 GB RAM under Microsoft Windows 7. The best, mean and worst objective function values of each algorithm in every run are obtained. Furthermore, the average computing time that an algorithm costs until the best solution is reached is also recorded.

Due to space limitation we only present the summary of results. The number of cases that an algorithm beats or equals to the other one is shown in Table 4.1. We can see from this table that the accuracy of TS is improved significantly by incorporating IA, considering best objective function values (headed “Best obj.”) and mean objective function values (headed “Mean obj.”). On the other hand, the worst objective function values (headed “Worst obj.”) and average computing time (headed “Mean time”) of the proposed algorithm do not become worse after adding IA. We can say that the TSIA improves the precision of TS significantly.

Further more, a diagrammatic analysis of the efficiencies of TSIA and TS is given

Table 4.1: The number of cases that an algorithm beats or equals to the other one

	TSIA	TS
Best obj.	**114**	82
Mean obj.	**102**	69
Worst obj.	**107**	106
Mean time	**62**	59

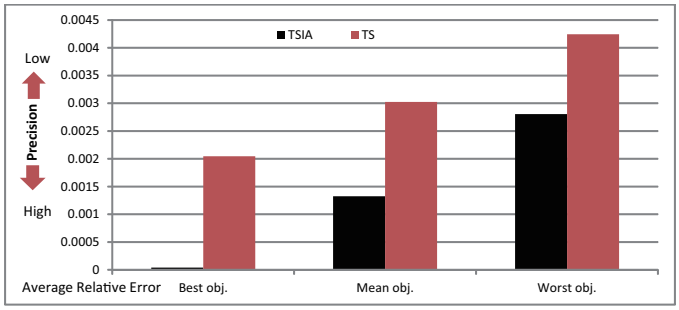

Figure 4.2: Average Relative Error of each instance

by Figure 4.2. Since the differences among objective function values are extremely small, we introduce *Relative Error* to evaluate the performance of each algorithm. It is calculated by $(S-B)/B$, where S indicates the objective function values of each algorithm, and B indicates the better objective function value between TSIA and TS. *Average Relative Error* of each instance is shown in this figure. The smaller the average relative error is, the higher the precision an algorithm has. From this figure we can say that the performance of TS is significantly improved by applying IA as a diverstification strategy.

4.4.2 Comparing TSIA to Existing Algorithms

To evaluate the efficiency of the proposed algorithm TSIA, we also compared it with three state-of-the-art existing algorithms. One algorithm is a Hybrid algorithm (TSACO) based on the TS and ACO, introduced in Chapter 3. The other two algorithms are tabu search algorithm (TSB) and ant colony optimization (ACOB),

both of which are introduced by Blum *et al.* in [34].

The proposed algorithm was coded in C language and compiled with C-Compiler: Microsoft Visual C++ 7.1. All of the experiments were performed in the following computation environment: CPU: Core 2 Duo 2.10 GHz, RAM: 2 GB, the multiprocessor did not process in parallel. For TSB and ACOB, the parameter settings as well as the source code used for all the experiments in this paper are just the same as those provided by Blum and Blesa [1]. Each algorithm was executed for 30 runs under the terminate condition TimeLimit = 300 (s) on benchmark instances [1].

Tables 4.2–4.8 show some of the results of these experiments. $|V|$, $|E|$, and $\bar{d}(v)$ indicate the number of vertices, the number of edges and the average number of edges a vertex connecting in a graph, respectively. k denotes the cardinality of kCTP. BKS means the best known solutions which have been obtained by Blum and Blesa through their tremendous experiments [1]. The rows headed "Best", "Mean" and "Worst" provide the best, average, and the worst objective function values, respectively. Results highlighted in bold mean that this algorithm beats others. The values marked by $*$ denote that the best known solutions were updated by that algorithm. In addition, columns headed "time" provide the average computing time that an algorithm costs until the best solution is reached.

We can see from Tables 4.2–4.8, that the performance of the proposed algorithm is significantly better than existing algorithms considering "Best value", "Mean value" and "Worst value". Furthermore, the proposed algorithm updates 24 best known solutions out of 63 cases and it's average computing time is 26% of HybridK's, 19% of TSB's, 12% of ACOB's. In conclusion, we can say that the proposed algorithm has a good performance both in terms of solution precision and computing time. We believe that the diversification strategy based on IA enlarges the search area and leads to a better solution.

4.5 Conclusion

We proposed a new hybrid metaheuristic based on TS and IA for solving kCTPs in this chapter. Computational tests indicate that IA is a good diversity strategy for TS, because it improves the precision significantly. That is because IA has an mechanism to generate new but high precision solutions and keeps the diversity of those solutions. It can be also observed that a proper combination of metaheuristics is efficient for solving kCTPs.

Table 4.2: Results for instances with Grid graph

Graph	k	BKS		TSIA	TSACO	TSB	ACOB
$\|V\| = 225$	40	695	Best	**695**	**695**	696	**695**
$\|E\| = 400$			Mean	**695.0**	**695.0**	696.0	**695.0**
$\bar{d}(v) = 3.55$			Worst	**695**	**695**	696	**695**
(bb45x5_1.gg)			Time	**0.0**	2.2	3.3	45.0
	60	1115	Best	***1107**	**1107**	1115	**1107**
			Mean	**1107.0**	1107.2	1115.0	1117.0
			Worst	**1107**	1114	1115	1129
			Time	14.8	55.8	**7.0**	135.4
	80	1568	Best	***1552**	1553	1579	1572
			Mean	**1553.5**	1557.8	1592.6	1572.3
			Worst	**1558**	**1558**	1615	1579
			Time	87.8	**61.0**	92.6	118.1
	100	1979	Best	***1963**	1974	2048	1974
			Mean	**1971.4**	1974.0	2048.6	1979.2
			Worst	**1972**	1974	2050	1990
			Time	**9.7**	36.8	126.7	160.6
	120	2450	Best	***2444**	2462	2546	**2444**
			Mean	**2448.2**	2468.7	2553.2	2474.6
			Worst	**2455**	2470	2575	2516
			Time	39.2	**25.3**	106.1	173.5
	140	3028	Best	***3024**	3025	3060	**3024**
			Mean	**3024.0**	3025.1	3062.1	3024.3
			Worst	**3024**	3026	3080	3026
			Time	63.3	**24.9**	115.8	98.0
	160	3702	Best	***3688**	**3688**	3724	**3688**
			Mean	**3688.0**	**3688.0**	3724.6	3698.8
			Worst	**3688**	**3688**	3729	3700
			Time	**10.5**	19.2	141.1	89.4
	180	4474	Best	***4472**	**4472**	4501	**4472**
			Mean	**4472.0**	**4472.0**	4505.1	4472.7
			Worst	**4472**	**4472**	4526	4490
			Time	**0.0**	0.6	139.3	100.7
	200	5461	Best	**5461**	**5461**	5462	**5461**
			Mean	**5461.0**	**5461.0**	5462.4	**5461.0**
			Worst	**5461**	**5461**	5463	**5461**
			Time	**0.0**	**0.0**	74.5	65.8

Table 4.3: Results for instances with Grid graph

Graph	k	BKS		TSIA	TSACO	TSB	ACOB
$\|V\| = 1089$	200	3255	Best	3331	**3325**	3431	3463
$\|E\| = 2112$			Mean	**3339.6**	3388.1	3527.2	3430.9
$\bar{d}(v) = 3.87$			Worst	**3347**	3554	3596	3472
(bb33x33_2.gg)			Time	**10.0**	142.7	122.6	233.2
	300	5185	Best	**5276**	5378	5432	5307
			Mean	**5373.5**	5471.3	5540.7	5460.0
			Worst	**5397**	5623	5630	5558
			Time	**97.2**	159.3	167.2	231.7
	400	7252	Best	**7372**	7525	7548	7503
			Mean	**7431.5**	7660.3	7696.7	7589.1
			Worst	**7507**	7929	7953	7844
			Time	**106.8**	152.5	180.7	292.8
	500	9465	Best	**9527**	9722	9939	9825
			Mean	**9539.6**	9901.1	10021.6	9845.2
			Worst	**9540**	9954	10144	10015
			Time	**45.2**	129.2	176.2	273.6
	600	11856	Best	**11874**	12060	12316	12648
			Mean	**11914.6**	12107.5	12425.8	12489.1
			Worst	**11941**	12266	12515	12706
			Time	**15.9**	202.8	162.5	253.3
	700	14509	Test	***14492**	14811	14984	15274
			Mean	**14492.8**	14898.9	15059.4	15311.0
			Worst	**14493**	14949	15108	15478
			Time	**18.4**	166.3	168.9	261.4
	800	17542	Best	***17511**	17708	17834	18487
			Mean	**17512.5**	17790.5	17897.1	18389.5
			Worst	**17517**	17832	17981	18622
			Time	**24.0**	279.0	153.0	261.8
	900	20993	Best	***20992**	21010	21052	21940
			Mean	**20992.0**	21011.1	21079.1	21932.1
			Worst	**20992**	21026	21131	22196
			Time	**0.4**	151.3	151.3	260.3
	1000	25273	Best	**25273**	25274	25307	26321
			Mean	**25273.8**	25274.9	25340.9	26446.4
			Worst	**25274**	25275	25361	26580
			Time	**36.7**	55.2	194.7	273.0

Table 4.4: Results for instances with Grid graph.

Graph	k	BKS		TSIA	TSACO	TSB	ACOB
$\|V\| = 1000$	100	1661	Best	1668	1668	1687	**1661**
$\|E\| = 1890$			Mean	1668.0	1668.9	1690.8	**1667.8**
$\bar{d}(v) = 3.78$			Worst	**1668**	1676	1700	**1668**
(bb100x10_ 2.gg)			Time	**0.0**	127.1	154.1	25.6
	200	3618	Best	**3624**	3668	3746	3648
			Mean	**3624.0**	3695.7	3754.2	3707.7
			Worst	**3624**	3724	3764	3942
			Time	**0.2**	159.5	143.9	224.3
	300	5435	Best	***5424**	5585	5692	5890
			Mean	**5426.3**	5785.9	5747.5	5954.8
			Worst	**5431**	6019	5860	6024
			Time	**1.1**	162.0	128.3	243.1
	400	7531	Best	***7517**	7733	7819	7732
			Mean	**7517.0**	8063.8	7966.6	8410.2
			Worst	**7517**	8240	8270	8624
			Time	**0.8**	233.1	166.3	273.3
	500	9861	Best	***9850**	10071	10021	10086
			Mean	**9850.0**	10306.8	10168.3	10292.2
			Worst	**9850**	10407	10628	10536
			Time	**1.4**	224.0	177.4	265.3
	600	12481	Best	**12512**	12597	12733	12716
			Mean	**12522.6**	12698.6	12821.2	12871.8
			Worst	**12523**	12821	12861	13224
			Time	**3.3**	207.4	161.1	273.5
	700	15599	Best	**15602**	15692	15848	15862
			Mean	**15605.0**	15758.7	15919.5	15969.1
			Worst	**15608**	15906	15954	16101
			Time	**0.8**	149.4	168.1	260.7
	800	19188	Best	***19177**	19212	19323	19522
			Mean	**19182.7**	19224.0	19456.0	19730.6
			Worst	**19187**	19277	19507	19833
			Time	**15.5**	113.9	159.9	260.7
	900	23481	Best	***23476**	23476	23489	23969
			Mean	**23476.0**	23476.0	23494.3	24102.3
			Worst	**23476**	23476	23519	24258
			Time	**0.4**	15.2	156.3	272.9

Table 4.5: Results for instances with Grid graph

Graph	k	BKS		TSIA	TSACO	TSB	ACOB
$\|V\| = 2500$	250	3988	Best	4054	4072	4160	**3950**
$\|E\| = 4900$			Mean	4075.3	4172.5	4229.9	**4055.8**
$\bar{d}(v) = 3.78$			Worst	**4076**	4276	4314	4097
(bb50x50_1.gg)			Time	**2.9**	193.1	181.8	192.8
	500	8150	Best	**8392**	8869	8842	8601
			Mean	**8392.0**	9050.9	8964.6	8815.0
			Worst	**8392**	9105	9174	9004
			Time	**3.4**	192.1	160.9	250.7
	750	12551	Best	**12776**	13727	13559	13876
			Mean	**12778.7**	13952.6	13796.3	14179.5
			Worst	**12789**	14311	14091	14456
			Time	**37.7**	265.0	163.9	293.2
	1000	17437	Best	**17657**	19236	18515	19461
			Mean	**17668.0**	19449.2	18956.6	19927.8
			Worst	**17701**	19895	19456	20351
			Time	**89.2**	273.5	205.6	341.5
	1250	22823	Best	**22862**	24581	24048	25638
			Mean	**22886.6**	24778.1	24418.0	26092.4
			Worst	**22900**	25537	24666	26622
			Time	**95.6**	264.9	275.7	310.6
	1500	28683	Best	***28621**	30985	30526	31675
			Mean	**28627.9**	31366.3	30856.9	32215.2
			Worst	**28636**	31474	31178	32853
			Time	**23.9**	258.7	294.2	470.2
	1750	35534	Best	***35427**	37321	37560	39154
			Mean	**35432.8**	37646.5	37851.2	39610.2
			Worst	**35448**	37845	38029	40520
			Time	**206.1**	274.5	293.0	329.0
	2000	43627	Best	***43574**	44899	45103	46663
			Mean	**43574.0**	44991.9	45256.1	47356.0
			Worst	**43574**	45267	45509	47917
			Time	**16.7**	255.3	294.2	397.1
	2250	53426	Best	***53409**	53682	53951	56562
			Mean	**53411.1**	53683.6	54086.9	57269.2
			Worst	**53413**	53725	54242	57633
			Time	**7.7**	284.8	225.4	341.6

Table 4.6: Results for instances with Regular graph

Graph	k	BKS		TSIA	TSACO	TSB	ACOB
$\|V\| = 1000$	100	1652	Best	*1651	**1649.0**	1662	**1649**
$\|E\| = 2000$			Mean	**1655.4**	1657.2	1664.2	1656.3
$\bar{d}(v) = 4$			Worst	**1660**	1664.0	1669	1661
(g1000-4-05.g)			Time	130.5	129.0	**120.9**	170.6
	200	3620	Best	**3633**	3666	3692	3642
			Mean	**3655.7**	3683.9	3720.4	3661.6
			Worst	**3681**	3706	3749	3688
			Time	161.1	**125.9**	140.5	236.7
	300	5801	Best	**5833**	5866	5931	5850
			Mean	**5851.7**	5922.7	5947.5	5890.0
			Worst	**5860**	5968	5967	5922
			Time	**5.2**	58.0	172.7	280.1
	400	8206	Best	**8230**	8333	8357	8327
			Mean	**8232.2**	8365.9	8382.7	8398.8
			Worst	**8234**	8392	8411	8461
			Time	**2.9**	150.7	148.3	278.5
	500	10793	Best	**10798**	10888	10964	11078
			Mean	**10798.0**	10897.5	10989.8	11165.1
			Worst	**10798**	10944	11029	11242
			Time	**0.9**	116.3	165.4	274.1
	600	13584	Best	**13587**	13665	13733	14005
			Mean	**13587.4**	13745.4	13753.3	14125.6
			Worst	**13588**	13806	13790	14253
			Time	**0.4**	99.0	158.1	256.2
	700	16682	Best	**16682**	16710	16800	17162
			Mean	**16683.9**	16721.2	16830.1	17311.3
			Worst	**16690**	16777	16861	17417
			Time	**0.5**	177.2	148.2	251.3
	800	20076	Best	***20074**	20078	20127	20682
			Mean	**20074.7**	20092.6	20143.4	20797.5
			Worst	**20079**	20105	20156	20912
			Time	**23.5**	87.9	164.6	255.6
	900	24029	Best	**24029**	**24029**	24042	24754
			Mean	**24029.0**	**24029.0**	24054.5	24861.6
			Worst	**24029**	**24029**	24063	25012
			Time	**2.7**	32.0	164.9	281.4

Table 4.7: Results for instances constructed from Steiner tree problems

Graph	k	BKS		TSIA	TSACO	TSB	ACOB
$\|V\| = 1000$	100	455	Best	***454**	462	456	455
$\|E\| = 5000$			Mean	**455.8**	466.0	458.4	455.8
$\bar{d}(v) = 10.0$			Worst	**457**	471	461	461
(steind15.g)			Time	**37.2**	132.5	144.3	153.9
	200	1018	Best	**1029**	1035	1042	1033
			Mean	**1037.1**	1043.0	1048.4	1043.3
			Worst	**1044**	1049	1055	1057
			Time	**41.1**	101.5	163.8	283.8
	300	1674	Best	**1686**	1692	1710	1733
			Mean	**1689.6**	1700.3	1720.0	1804.0
			Worst	**1690**	1710	1733	1866
			Time	**8.1**	160.3	150.9	254.4
	400	2446	Best	**2458**	2474	2482	2691
			Mean	**2458.0**	2483.4	2499.0	2832.3
			Worst	**2459**	2488	2512	2978
			Time	**0.8**	115.5	162.4	239.1
	500	3365	Best	**3372**	3390	3397	3802
			Mean	**3372.1**	3399.1	3409.0	4033.8
			Worst	**3375**	3407	3428	4163
			Time	**0.2**	70.9	164.6	259.6
	600	4420	Best	**4422**	**4422**	4439	5142
			Mean	**4423.6**	4426.5	4453.6	5397.6
			Worst	**4424**	4433	4483	5573
			Time	**62.2**	153.0	152.3	254.5
	700	5685	Best	**5687**	5695	5713	6595
			Mean	**5689.7**	5697.9	5732.3	6942.1
			Worst	**5690**	5698	5751	7163
			Time	**17.9**	64.5	143.0	265.0
	800	7236	Best	**7236**	7237	7254	8551
			Mean	**7236.0**	7237.3	7271.6	8804.2
			Worst	**7236**	7246	7305	9001
			Time	**0.4**	86.8	192.8	275.3
	900	9248	Best	**9248**	9256	9283	10826
			Mean	**9248.0**	9256.0	9292.2	11117.2
			Worst	**9248**	9256	9307	11352
			Time	**0.3**	6.1	149.5	228.6

Table 4.8: Results for instances constructed from Steiner tree problems

Graph	k	BKS		TSIA	TSACO	TSB	ACOB
$\|V\| = 2500$	250	3883	Best	**3963**	4147	4191	3954
$\|E\| = 3125$			Mean	**3963.0**	4216.9	4246.7	4001.2
$\bar{d}(v) = 2.5$			Worst	**3963**	4289	4307	4068
(steine5.g)			Time	**0.3**	131.9	177.5	262.3
	500	9306	Best	**9382**	9836	9875	9825
			Mean	**9383.4**	10020.2	10011.6	9974.1
			Worst	**9384**	10268	10239	10173
			Time	**0.8**	141.6	177.4	253.9
	750	15818	Best	**15854**	16532	16460	16854
			Mean	**15862.5**	16757.5	16872.1	17067.7
			Worst	**15891**	17042	17129	17229
			Time	**1.6**	247.5	170.9	284.4
	1000	23528	Best	**23553**	24604	24131	24948
			Mean	**23553.8**	24810.0	24513.8	25281.6
			Worst	**23555**	24933	24861	25589
			Time	**3.0**	279.5	211.8	296.4
	1250	32493	Best	**32607**	33925	33375	34209
			Mean	**32608.0**	33975.2	33534.7	34590.5
			Worst	**32608**	34141	33871	34923
			Time	**2.8**	273.0	287.0	315.5
	1500	42769	Best	**42782**	44160	43983	44864
			Mean	**42782.7**	44405.4	44157.2	45170.0
			Worst	**42785**	44583	44377	45458
			Time	**4.5**	288.3	278.2	402.7
	1750	54763	Best	***54762**	55978	56004	56627
			Mean	**54765.7**	56327.8	56115.4	57061.6
			Worst	**54769**	56475	56236	57476
			Time	**27.5**	287.2	296.6	448.2
	2000	68622	Best	**68637**	69691	69554	70450
			Mean	**68637.0**	69704.7	69742.2	70905.5
			Worst	**68637**	69876	69912	71159
			Time	**4.2**	269.1	296.6	434.9
	2250	85366	Best	***85361**	85883	85848	87071
			Mean	**85361.0**	85883.0	86080.2	87378.2
			Worst	**85361**	85883	86544	87745
			Time	**5.3**	272.6	266.6	332.2

Chapter 5

Hybrid Metaheuristics Based on Memeteic Algorithm and Tabu Search

5.1 Introduction

In this chapter, we develop two Hybrid Metaheuristics, both of which are based on Memeteic Algorithm (MA) and Tabu Search (TS).

In the first one, a MA based on TS is proposed. It has both merits of EC and Local Search. Note that a configuration is a list of vertices of a feasible solution. To enlarge the search area, a crossover operator is applied to combine all vertices of two configurations and returns a feasible solution with a good objective function value. Moreover, to find the optimal solution, TS with short-term memory is applied to each feasible solution generated by crossover. To enhance the quality of initial population, one configuration of initial population is generated by DP. Experimental results show that the proposed algorithm has a high solution precision and a short computing time. This is because the crossover in memetic algorithm enlarges search area effectively, and local search improves the solution greatly.

The second one is a TS with MA, which acts as a powerful diversitification strategy. In addition, the TS with dynamic tabu list improved the precision of solution significantly. Experimental results show that the new hybrid metaheuristic is dramatically superior to exiting algorithms in precision. Specifically, some of

our proposed algorithm updates all the best known solutions of large benchmark instances proposed by Blum *et al.* (e.g., graphs with more than 5000 edges). It also indicates that nothing else matches its balance of diversity and centralization of solutions in hybrid metaheuristics.

5.2 Memetic Algorithm

Traditional Evolutionary Computation (eg. Genetic Algorithm) has been applied widely to solve optimization problems because of their good search abilities. On the other hand, they may not be efficient to some problems that contain many local optima. For example, it seems difficult to reach the best solution x^* by Evolutionary Computation directly for a minimization problem in Figure 5.1. However, it is easy to find the best solution by local search if the search starts from B. In our study, MA is used as a diversification strategy to reach B easily. Then TS is applied to find optimal solution efficiently.

MA was firstly introduced by Moscato in 1989 [3]. It has both merits of Evolutionary Computation and local search. In this section, we present a new MA based on TS for solving the kCTP. One of the core ideas is that vertices in a feasible solution (k-cardinality tree) with a good objective function value are usually good components for constructing an optimal solution. Note that a configuration is a list of vertices of a feasible solution. Firstly, we pay attention to the diversity of configurations in each generation. When generating a new generation, repeated configurations should be gotten rid of from the population and the space would be filled up with new configurations. Secondly, to enlarge the search area, crossover is applied to combine all vertices of two configurations and returns a feasible solution with a good objective function value. Finally, in order to find the optimal solution, TS is applied to each feasible solution generated by crossover. Moreover, to enhance the quality of initial population, one configuration of initial population is generated by Dynamic Programming [38].

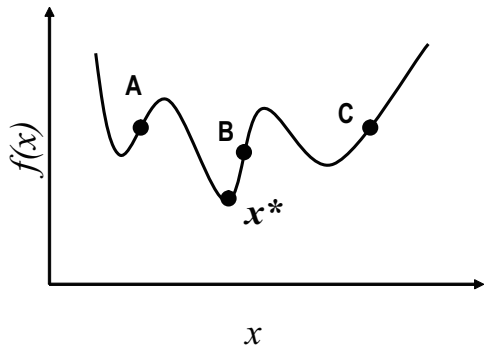

Figure 5.1: A Minimization Problem

The pseudocode of proposed MA is shown in the following:

Memetic Algorithm

P:= Generating Initial Population

while stop criterion not satisfied **do**

P' := Crossover (P)

P' := Renew (P')

P'' := Tabu Search (P')

P := Renew (P'')

end while

return the best k-cardinality tree in P,

where P means the population of configurations. Renew $(P) := P/\ P_{repeated} \cup P_{new}$. $P_{repeated}$ and P_{new} are repeated configurations and configurations generated in the way of Section 5.2.1, respectively.

Generating Initial Population According to numeral experiments beforehand, we find that the performance of the algorithm is high if there is a configuration with high precision in the initial population. The first configuration is obtained

by DP, originally introduced in [38]. It has a good objective function value and is considered to be able to improve the quality of the population P. In order to keep the diversity of configuration in the population, others are generated under the procedure described in Section 5.2.1. Additionally, in order to make sure that the structures of all the individuals are not the same, we compared the newly generated configuration (a list of vertices) with the existing ones. If they are reduplicate, new configuration should be regenerated with $p' := 1 - p'$ momentarily.

Stopping criteria of Memetic Algorithm We define a generation as an *idle generation* if the best objective function value is not improved in that generation. MA stops if idle generation occurs continuously for several times. In this way, the algorithm makes sure there is enough chance for the improvement of solution.

Crossover In our study, a genetic operation, *crossover*, is applied for generating new configuration. *Crossover* has been adopted in many Evolutionary Computations. It enlarges the explored domain, so that the search can escape from local optima easily. The crossover operator is completed by the following two procedures:

Step one, generate spanning tree based on two configurations. Two individuals in P are considered to be parents T_k^C and T_k. If T_k and its cross partner T_k^C have at least one common vertex, a vertex set $V(G^C)$ is defined as: $V(G^C) = V(T_k) \bigcup V(T_k^C)$. Otherwise, edges and vertices should be added to T_k until at least one common vertex is found with $p' := 1 - p'$ momentarily. A spanning tree T^{SP}, which contains all vertices of $V(G^C)$, is constructed under the procedure we introduced in Section 5.2.1.

Step two, generate k-cardinality tree from T^{SP}. DP [38] is applied to the T^{SP} for finding out the best k-cardinality tree. Since DP is very efficient, the crossover operator will help us get a feasible solution with a good objective function value in a very short computing time.

5.2.1 Growing a k-cardinality tree

The algorithm we consider here uses a semi-greedy approach to solve problems. It grows a k-cardinality tree one edge at a time. Let T be a subset of a k-cardinality tree T_k. We call an edge a *safe edge* if T is still a tree after being added with it. Firstly, a vertex is selected randomly to be the first component of tree T. Then in each step, one safe edge should be added to T until there are k edges in the tree T.

To obtain various k-cardinality trees, a real number $p \in (0, 1)$ is generated randomly at each step. If $p \leq p'$, the edge with smallest weight in safe edges will be selected and added to T, else one edge would be selected randomly from safe edges. The value p' determines the range of the heuristic bias. In an extreme case of $p' = 1$, at each step the edge added to T is the best edge in safe edges, thus the construction would be equivalent to Prim's algorithm. It tends to reach k-cardinality trees with a smaller objective function value, but these trees tend to be less diversified. On the contrary, in case $p' = 0$, a k-cardinality tree would be constructed randomly. In this case, it would not be a good initial solution for further searching. It is expected to attain a good balance between the goodness of initial solutions and their diversity by a proper value p'. In this research, we determine the value $p' = 0.85$ due to the results of preliminary numeral experiments.

The pseudo-code is shown in the following:

Growing a k-cardinality tree

T$\Leftarrow$ select one vertex randomly

while k-cardinality tree is not completed **do**

List $\leftarrow$ generate list of safe edges

$p \leftarrow$ generate a value randomly in (0,1)

if $p \leq p'$

$(u, v) \leftarrow$ an edge with minimal weight in List of safe edges

T$\leftarrow$ T $\cup$ (u, v)

else

$(u, v) \leftarrow$ an edge randomly selected form List of safe edges

T$\leftarrow$ T $\cup$ (u, v)

Update the List of safe edges

end while

5.2.2 Details of Tabu Search with short-term memory

After crossover, each offspring should be further improved by local search.In the proposed algorithm, we found TS-based local search effective. That is because the most important characteristic of TS is that it uses a concept of *memory* to control movements via a dynamic list of forbidden movements. To be more specific, the solutions which have been searched will be "tabu" (forbidden) from visiting for a while. This mechanism allows TS to intensify or diversify its search procedure in order to escape from local optima. Incidentally, TS has also been proved to be effective in solving kCTPs [34].

The basic ingredient of TS is local search. Local search is often conducted via some move operators. A move from the current solution to the candidate solution is only performed when the objective function value is improved. Local search we used here is the same with the one introduced in Chapter 3.

Length of Tabu List The core procedure of TS is to forbid some moves based on memory in order to enlarge the search area. In the proposed algorithm, the "tabu" (forbiddance) is applied to edges that have been added to or deleted from the k-cardinality tree recently. Tabu lists are used as a memory to record

edges that should be forbidden to be added or deleted. *InList* and *OutList* are adopted to keep the records of removed edges and added edges, respectively. Tabu tenure, which generally depends on the length of tabu lists, is a period during which it forbids edges in the tabu lists from being added or deleted. The lengths of tabu lists are not dynamically changed in the proposed algorithm, since the computing time will explode as the length increases. The length of tabu list (tl) is defined as follows:

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

where $|V|$ is the number of vertices in G, k is the value of cardinality.

Aspiration Criterion The "tabu" mechanism, which forbids some of the moves to be employed, helps the algorithm avoid falling into local optima. However, this mechanism may also forbid a move that may reach the best solution. In order to avoid such a situation, a procedure called *aspiration criterion* is used in the proposed algorithm. That is, if $f(T_k^{new}) < \gamma_e$ is satisfied, the movement will be acceptable even if edge e is included in *InList* or *OutList*. Parameters γ_e called *aspiration level criterion* are given to all of edges and are initially set to be:

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

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

5.2.3 Experimental study

To evaluate the efficiency of MA, we compared the proposed method (MA) with three state-of-the-art existing algorithms. One algorithm is a Hybrid algorithm (TSACO) based on the TS and ACO, introduced in Chapter 3. The other two algorithms are tabu search algorithm (TSB) and ant colony optimization (ACOB), both of which are introduced by Blum *et al.* in [34].

We use C as the programming language and compile the program with C-Compiler:

Microsoft Visual C++ 2010 Express. MA is tested 10 runs on a PC with Intel Core i7 2.8 GHz CPU (the multi-processor did not process in parallel) and 8 GB RAM under Microsoft Windows 7. The best, mean and worst objective function values and computing time are obtained. Accordingly, the results of existing algorithms are referred to Chapter 3. They executed each method for 30 runs under the condition that TimeLimit = 300 (s). All the metaheuristic approaches were tested on a PC with Celeron 3.06 GHz CPU and RAM 1 GB under Microsoft Windows XP.

The experiments were applied to several famous instances [1] and instances proposed in Chapter 3, respectively. Tables 5.1-5.6 show the results of these experiments. $|V|$, $|E|$, and $\bar{d}(v)$ indicate the number of vertices, the number of edges and the average number of edges a vertex connecting in a graph, respectively. k denotes the cardinality of kCTP. BKS means the best known solutions which have been obtained by Blum and Blesa through their tremendous experiments [1]. The rows headed "Best", "Mean" and "Worst" provide the best, average, and worst objective function values, respectively. "–" indicates that the algorithm does not derive solutions within the given time limit. Results highlighted in bold mean that this algorithm beats others. The values marked by $*$ denote that the best known solutions were updated by that algorithm. In addition, columns headed "time" provide the average computing time to reach the best solution.

From Table 5.1 we can see that the precision of MA is not so good, even worse than ACOB in cases of the best objective function values when the graph size is small. However, comparing Tables 5.1 with 5.2 and 5.3, we find that as the size of graph becomes large the performance of MA establishes total supremacy to rivals considering "Best value", "Mean value" and "Worst value". We believe that the diversification strategy based on MA enlarges the search area and leads to a better solution, especially when the graph size is large.

The $\bar{d}(v)$s of instances in Tables 5.4, 5.5, and 5.6 are larger than those of instances in Tables 5.1, 5.2 and 5.3. In these instances with large $\bar{d}(v)$s, the performance of MA is also outstanding, especially considering the mean objective function values.

Furthermore, the MA's deviation between "Best", "Mean" and "Worst" is relatively smaller than those of other algorithms. It can be thought that the MA has a strong robustness.

In consideration of terminate condition is 300 (s) for other algorithms, the computing time of the proposed algorithm is relatively short (most of them are less than 1 second). We can conclude that the proposed MA is good in solution precision and is high-speed.

5.3 Tabu Search combined with Memeteic Algorithm

Though the proposed MA is powerful in diversification and is better in precision for some benchmark problems than existing algorithms, its precision is poor in some case compared to the best known solutions in literature [1]. One possible reason is that graphs in those benchmark instances are different in size, degree and weights. It has been pointed out that characteristics of a problem instance as well as the size of the cardinality have a high influence on the behavior of algorithm. To solve this problem, hybrid algorithm are considered. The basic idea is that different algorithms have their superiorities for some kinds of benchmark instances and have different features in searching. Different algorithms may be complementary in a hybrid algorithm.

As we have summarized before, if there is a solution with high precision in initial population, the performance will be improved significantly. On the other hand, MA also maintains the diversity of individuals in the population. Among the metaheuristics proposed for solving kCTP, we find TS has both the search ability to generate a solution with high precision and short coming in keeping diversity of solutions. Based on these considerations, we propose a hybrid metaheuristic in which MA acts as diversification strategy for TS. It is expected to be a metaheuristic which has the highest precision for almost all of the benchmark instances in [1].

Pseudo-codes of hybrid algorithm proposed are shown in the following:

Input a problem instance (G, w, k)

$s^* \leftarrow \infty$

$s \leftarrow$ Dynamic Programming()

$s^* \leftarrow s$ **if** $f(s) < f(s^*)$

$s' \leftarrow$ Tabu Search(s^*)

$s^* \leftarrow s'$ **if** $f(s') < f(s^*)$

while Terminate condition not met **do**

$s'' \leftarrow$ Memetic Algorithm($s^*, s_1, s_2, ..., s_n$)

$s^* \leftarrow s''$ **if** $f(s'') < f(s^*)$

$s''' \leftarrow$ Tabu Search(s^*)

$s^* \leftarrow s''$ **if** $f(s'') < f(s^*)$

end while

Output s^* (or T_k),

where s, s', s'', s''' indicate feasible solutions (or a k-tree T_k). s^* means the solution with global best objective function value. Function $f(\bullet)$ computes the objective function value. $s_1, s_2, ..., s_n$ indicates the population of solutions in MA.

As we have given experiment results of TS in Figure 2.1 in Chapter 2, the computing time of TS is significantly large compared with DP. In which the initial solution for TS is generated by Prim algorithm which is poor in precision (Figure 2.1). Referring to Algorithm 4 in Chapter 2, the length of tabu list in proposed TS increases when the solution is not improved for several iterations until the determined length. If the initial solution is poor in precision, process of increasing tabu length may slow, and the number of times of local search becomes large. Accordingly the computing

time of TS becomes long. In order to make the computing time shorter, we use the solution generated by DP, which gives a solution with high precision fleetly, as the initial solution. In other words, a good initial solution helps TS converge in a shorter computing time. On the other hand, according to the results in Table 2.1 and 2.2 in Chapter 2, the results obtained by DP are better when cardinality (k) are small. Hybrid algorithm is considered to achieve expectable solutions of varied cardinalities.

As we have introduced above, MA acts by updating population of solutions. Varied individuals (solutions) in the population and the crossover operator between two individuals keep the diversification of solutions. Knowing this feature, the hybrid algorithm applies the MA after TS. The MA makes the solution generated by one of the individuals and newly generates other individuals under the way we introduced in Chapter 5.2.1. To explore the search area thoroughly, TS is applied once again. In this way, MA acts as the diversification strategy for TS.

In the proposed algorithm, the procedure combined MA and TS terminates if the solution have not been improved for 5 iterations or two times of iterations in which solution has been improved. In such way the algorithm make sure there is enough chance for the solution improving when a problem is complex.

Details of the TS and MA can be referred to the chapter 3 and this chapter, respectively. Since TS and MA can generate "good" solution for each other, combing these two different search methods to solve the kCTP, it is desirable to reach a high performance.

5.3.1 Experiments and results

In order to evaluate the performance of the proposed hybrid metaheuristic algorith, we did experiments of the proposed algorithm. Looking for a hybrid algorithm of the highest precision, we compare the results of proposed hybrid metaheuristic (HyTSMA) with the best known solutions (BKS) given in literature [1]. We select the most kinds of benchmark instances in [1]. They are instances with regular graph,

grid graph and instances constructed from Steiner problems. Since the complexity of instance increases as the size of graph becomes large, we choose the largest graph of each kind of instance. The cardinalities are nearly 20%, 40%, 60%, 80% and 90 % of the number of vertices of the graph. We use C as the programming language and compile the program with C-Compiler: Microsoft Visual C++ 2010 Express. MA is tested 30 runs on a PC with Intel Core i7 2.93 GHz CPU (the multi-processor did not process in parallel) and 4 GB RAM under Microsoft Windows 7. The best, mean and worst objective function values and computing time are obtained. For an instance, if the number of edges is not more than 2000 the limited time condition is set to be 900s. For a graph is larger, the limited time condition is set to be 7200s.

Tables 5.7 to 5.10 show the results of experiments. $|V|$ indicates the number of vertices in a graph, and k denotes the cardinality of kCTP. BKS means the best known solutions which have been obtained by Blum and Blesa through their tremendous experiments [1]. The rows headed "Best", "Mean" and "Worst" provide the best, mean, and worst objective function values, respectively. The values marked by $*$ denote that the best known solutions were equal to or updated by that algorithm.

We can see from the results that, the precision of HyTSMA establishes total supremacy to rivals considering "Best value", "Mean value" and "Worst value". Considering the precision is hard to be improved for an instance with large graph (eg. 2500 vertices and 4900 edges), the proposed algorithm improved the BKS significantly. Furthermore, the hybrid algorithm is robust since the instances include almost all kinds of instances in [1]. Based on those results, we can say that the proposed algorithm has the highest precision among approximate algorithms.

5.4 Conclusion

In this chapter we proposed a MA and a hybrid algorithm based on MA and TS for kCTP. The proposed algorithm enhances the diversity of the configurations in each generation, which helps search escape from local optima even the size of the

graph is large. The experiments applied in existing benchmark instances show that MA is able to find optimal (near-optimal) solutions for larger instances within short running time. The high precision of proposed hybrid algorithm can be observed that a proper combination of metaheuristics is efficient for solving kCTPs.

Table 5.1: Results on grid graph.

Graph	k	BKS		MA	time (s)	TSACO	TSB	ACOB
$\|V\| = 225$	40	695	Best	**695**	0.008	**695**	696	**695**
$\|E\| = 400$			Mean	699.6	0.029	**695.0**	696.0	695.4
$\bar{d}(v) = 3.55$			Worst	728	0.052	**695**	696.0	696.0
(bb45x5_1.gg)	80	*1552	Best	1618	0.013	**1552**	1579	1572
		(1568)	Mean	1636.9	0.029	**1565.1**	1592.7	1581.2
			Worst	1639	0.104	**1572**	1615	1593
	120	*2444	Best	2456	0.038	**2444**	2546	2457
		(2450)	Mean	2468.7	0.092	**2457.9**	2558.5	2520.3
			Worst	2477	0.154	**2465**	2575	2601
	160	*3688	Best	3701	0.027	**3688**	3724	3700
		(3702)	Mean	3714.1	0.085	**3688.0**	3724.9	3704.7
			Worst	3724	0.260	**3688**	3729	3720
	200	5461	Best	**5461**	0.032	**5461**	5462	**5461**
			Mean	**5461.0**	0.055	**5461.0**	5462.4	5469.0
			Worst	**5461**	0.154	**5461**	5463	5485

Table 5.2: Results on regular graph

Graph	k	BKS		MA	time (s)	TSACO	TSB	ACOB
$\|V\| = 1000$	200	3308	Best	3421	0.186	3393	3438	**3312**
$\|E\| = 2000$			Mean	3423.7	0.202	3453.1	3461.4	**3344.1**
$\bar{d}(v) = 4$			Worst	3424	0.226	3517	3517	**3379**
(g1000-4-01.g)	400	7581	Best	**7600**	1.021	7659	7712	7661
			Mean	**7621.8**	1.926	7764.0	7780.2	7703.0
			Worst	**7636**	3.220	7819	7825	7751
	600	12708	Best	**12733**	0.982	12785	12801	12989
			Mean	**12746**	2.035	12836.6	12821.8	13115.6
			Worst	**12759**	4.001	13048	12869	13199
	800	19023	Best	**19033**	1.496	19099	19093	19581
			Mean	**19047.1**	3.682	19101.1	19112.6	19718.7
			Worst	**19060**	12.872	19128	19135	19846
	900	22827	Best	**22827**	0.072	**22827**	22843	23487
			Mean	22829.7	0.176	**22827.0**	22859.2	23643
			Worst	22830	1.006	**22827**	22886	23739

Table 5.3: Results on regular graph

Graph	k	BKS		MA	time (s)	TSACO	TSB	ACOB
$\|V\| = 1000$	200	1018	Best	**1018**	0.166	1034	1036	1036
$\|E\| = 5000$			Mean	**1024.2**	0.451	1048.6	1047.3	1045.9
$\bar{d}(v) = 10.0$			Worst	**1036**	0.849	1063	1056	1056
(steind15.g)	400	2446	Best	**2448**	0.883	2469	2493	2665
			Mean	**2452.4**	1.331	2480.7	2502.5	2806.6
			Worst	**2458**	2.206	2492	2524	2928
	600	4420	Best	**4420**	0.553	4426	4442	5028
			Mean	**4420.7**	0.934	4433.0	4454.6	5398.4
			Worst	**4423**	1.772	4451	4490	5602
	800	7236	Best	**7236**	1.736	**7236**	7252	8457
			Mean	7237.8	2.356	**7237.0**	7272.8	8839.6
			Worst	7239	3.741	**7237**	7308	9006
	900	9248	Best	**9248**	0.068	9256	9283	10873
			Mean	**9248**	0.080	9256.0	9294.2	11166.3
			Worst	**9248**	0.097	9256	9304	11423

Table 5.4: Results on instances constructed from graph coloring problems

Graph	k	BKS		MA	time (s)	TSACO	TSB	ACOB
$\|V\| = 450$	90	135	Best	**135**	0.006	**135**	**135**	**135**
$\|E\| = 8168$			Mean	**135**	0.010	135.1	135.3	135.7
$\bar{d}(v) = 36.30$			Worst	**135**	0.034	137	136	137
(le450_15a.g)	180	336	Best	**336**	0.008	**336**	337	352
			Mean	**336.5**	0.071	337	337.1	374.4
			Worst	**337**	0.268	**337**	338	419
	270	630	Best	**630**	0.163	**630**	**630**	696
			Mean	**630**	0.175	630.1	630.3	839.0
			Worst	**630**	0.196	631	633	913
	360	1060	Best	**1060**	0.014	**1060**	**1060**	1267
			Mean	**1060**	0.020	**1060.0**	1064.1	1461.2
			Worst	**1060**	0.060	**1060**	1118	1566
	405	1388	Best	**1388**	0.014	**1388**	**1388**	1767
			Mean	**1388**	0.018	**1388**	1391.1	1888.7
			Worst	**1388**	0.030	**1388**	1392	2015

Table 5.5: Results on new instances (1)

Graph	k		MA	time (s)	TSACO	TSB	ACOB
$\|V\| = 500$	100	best	**1943**	0	**1943**	1954	**1943**
$\|E\| = 15000$		mean	**1943**	0.008	1950.5	1990.9	2022.3
$\bar{d}(v) = 60$		worst	**1943**	0.016	1966	2023	2241
	200	best	5062	0	**5037**	5063	5517
		mean	5062.8	0.066	**5047.3**	5080.4	7444.4
		worst	**5063**	0.312	5066	5221	9859
	300	best	9760	0.577	**9758**	9821	-
		mean	**9761.7**	0.716	9769.6	9922.6	-
		worst	**9763**	0.998	9795	11696	-
	400	best	**16351**	0.015	**16351**	16373	-
		mean	**16351.0**	0.017	16363.8	16488	-
		worst	**16351**	0.031	16368	17953	-
	450	best	**20929**	0.015	**20929**	20934	-
		mean	**20929**	0.019	**20929**	20945.2	-
		worst	**20929**	0.032	**20929**	20992	-

Table 5.6: Results on new instances (2)

Graph	k		MA	time (s)	TSACO	TSB	ACOB
$\|V\| = 500$	100	best	1306	0.171	**1294**	1319	1398
$\|E\| = 30000$		mean	1319	0.278	**1303.7**	1352.8	1743.5
$\bar{d}(v) = 120$		worst	1322	0.780	**1321**	1385	2479
	200	best	**3007**	0.250	3064	3150	4013
		mean	**3012.4**	0.700	3097.1	3934.4	6861.4
		worst	**3020**	1.154	3127	6032	9623
	300	best	**5304**	0.452	5312	5380	-
		mean	**5304.0**	0.479	5312.5	6471.9	-
		worst	**5304**	0.515	5318	8308	-
	400	best	**8582**	0.015	**8582**	8586	-
		mean	**8582**	0.017	**8582**	9540	-
		worst	**8582**	0.031	**8582**	11485	-
	450	best	**10881**	0.015	**10881**	10882	-
		mean	**10881**	0.020	**10881**	11300.4	-
		worst	**10881**	0.032	**10881**	13570	-

Table 5.7: Results for instances with Regular graph [1]

Graph	k	BKS		HyTSMA
$\|V\| = 1000$	200	3308	Best	*3308
$\|E\| = 2000$			Mean	3308.4
$d(v) = 4$			Worst	3318
(g1000-4-01.g)	400	7581	Best	7586
			Mean	7589.1
			Worst	7596
	600	12708	Best	*12705
			Mean	*12705.8
			Worst	12709
	800	19023	Best	*19015
			Mean	*19015.6
			Worst	*19017
	900	22827	Best	*22827
			Mean	*22827
			Worst	*22827

Table 5.8: Results for instances with Regular graph [1]

Graph	k	BKS		HyTSMA
$\|V\| = 1000$	200	3620	Best	*3618
$\|E\| = 2000$			Mean	3621.9
$d(v) = 4$			Worst	3635
(g1000-4-05.g)	400	8206	Best	*8197
			Mean	*8200.6
			Worst	*8206
	600	13584	Best	*13580
			Mean	*13580.1
			Worst	*13582
	800	20076	Best	*20074
			Mean	*20074
			Worst	*20074
	900	24029	Best	*24029
			Mean	*24029
			Worst	*24029

Table 5.9: Results for instances with Grid graph

Graph	k	BKS		HyTSMA
$\|V\| = 2500$	500	8150	best	*8088
$\|E\| = 4900$			mean	*8127.4
$\bar{d}(v) = 3.78$			worst	*8135
(bb50x50_1.gg)	1000	17437	best	*17381
			mean	*17424.8
			worst	17480
	1500	28683	best	*28502
			mean	*28517.3
			worst	*28528
	2000	43627	best	*43541
			mean	*43549.6
			worst	*43558
	2250	53426	best	*53407
			mean	*53407.3
			worst	*53409

Table 5.10: Results for instances constructed from Steiner problems

Graph	k	BKS		HyTSMA
$\|V\| = 2500$	500	9306	Best	*9299
$\|E\| = 3125$			Mean	*9299.7
$\bar{d}(v) = 2.5$			Worst	9312
(steine5.g)	1000	23528	Best	*23500
			Mean	*23501.8
			Worst	*23511
	1500	42769	Best	*42735
			Mean	*42737.0
			Worst	*42746
	2000	68622	Best	*68618
			Mean	*68618
			Worst	*68618
	2250	85366	Best	*85360
			Mean	*85360.2
			Worst	*85361

Chapter 6

Conclusion

In this doctoral dissertation we have proposed new hybrid metaheuristics combining bio-inspired algorithms with Tabu Search (TS) and/or Dynamic Programming (DP) for the k-Cardinality Tree Problem (kCTP) . Properties of each sole algorithm as well as those hybrid metaheuristics have been analyzed. Furthermore, how to construct an efficient hybrid metaheuristic is also discussed.

To be more precise, in Chapter 3, a new hybrid algorithm for kCTP by combining TS and ACO was proposed. We have shown that the performances of the proposed method are better than those of existing methods. Furthermore, it has been shown that the proposed method updates some of the best known values. In Chapter 4, we proposed a new hybrid metaheuristic based on TS and Immune Algorithm (IA) for solving kCTPs. It indicates that IA is a good diversity strategy for TS, since it improves the precision significantly. In Chapter 5, we proposed a MA based on TS for kCTP. The proposed algorithm enhances the diversity of the configurations in each generation, which helps search escape from local optima even the size of the graph is large. It can be also observed that a proper combination of metaheuristics is efficient for solving kCTPs.

Numerical results show that proposed algorithms are competitive to existing algorithms from the viewpoint of solution accuracy and computing time. Specifically, some of our proposed algorithm updates all the best known solutions of large benchmark instances proposed by Blum *et al.* (e.g., graphs with more than 5000 edges). Based on large amounts of experiments and their results of proposed hybrid meta-

heuristics, we conclude the following points, which should be paid attention to, for solving kCTP. Hoping these advices would be helpful in developing hybrid metaheuristics for other combinatorial optimal problems.

Bio-inspired techniques provide powerful strategies. Using ACO as diversification strategy, the algorithm restarts and further improves the solution. In IA, the mechanism that maintains the diversity of individuals helps the algorithm avoid falling into these traps of local optimal. Additionally, MA effectively enlarges the search area by using population of individuals, crossover and tabu search with short-term memory.

To visit new solutions, worse solution should also be admitted. Since local search always take the better or the best solution in neighborhood as the current solution, in the proposed algorithm even a worse solution is used as an initial solution when restarting the TS. Those "worse" initial solutions, generated by ACO or IA, may have very different structures comparing to the current solution.

Crossover operator is efficient to enlarge the search area. Crossover plays an important role both in IA and MA. The idea of Crossover is that a good offspring is often generated by on good parents. Further more, two individuals far away in the search area may generate an offspring located between them, as a result of enlarging the search area.

Hybrid algorithm doubles powerfulness of sole metaheuristic. For a metaheuristic algorithm with defects in structure, e.g. the computing cost of TS may extremely large if the length of tabu list is very long, hybrid algorithm is one of the best ways of filling that gap by combing with MA or IA. The results of experiments show that the hybrid algorithm beats state-of-the-art metaheuristics in precision of solution. It also indicates that nothing else matches its balance of diversity and centralization of solutions in hybrid metaheuristics.

References

[1] KCTLIB. http://iridia.ulb.ac.be/~cblum/kctlib/, 2003.

[2] F. Glover, Future paths for integer programming and links to artificial intelligence. *Computers and Operations Research* 5, pp. 533–549, 1986.

[3] P. Moscato. On Evolution, Search, Optimization, Genetic Algorithms and Martial Arts: Towards Memetic Algorithms. *Technical Report Caltech Concurrent Computation Program*, Report. 826, California Institute of Technology, Pasadena, California, USA, 1989.

[4] F. Glover, Tabu Search-Part I, Operations Research Society of America, *Journal on Computing*, Vol. 1, No. 3, Summer 1989.

[5] H.W. Hamacher, K. Jornsten, F. Maffioli, Weighted k-cardinality trees, *Technical Report 91.023*, Politecnico di Milano, Dipartimento di Elettronica, Italy, 1991.

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

[7] J. Freitag, Minimal k-cardinality trees, *Master thesis, Department of Mathematics*, University of Kaiserslautern, Germany, 1993.

[8] S.Y. Cheung, A. Kumar, Efficient quorum-cast routing algorithms, *Proceedings of INFOCOM*, Los Alamitos, USA, Silver Spring, MD: IEEE Society Press, 1994.

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

[10] B. Awerbuch, Y. Azar, A. Blum, S.Vempala, Improved approximation guar-

antees for minimum-weight k-trees and prize-collecting salesmen, *STOC '95 Proceedings of the twenty-seventh annual ACM symposium on Theory of computing*, Pp.277–283, 1995.

[11] D.P. Bertsekas, *Dynamic Programming and Optimal Control*, vol. 1, Athena Scientific, Belmont, MA, 1995.

[12] A. Blum, P. Chalasani, S. Vempala, A constant factor approximation algorithm for the k-MST problem, *STOC '95 Proceedings of the twenty-seventh annual ACM symposium on the Theory of Computing* , pp. 294–302, 1995.

[13] M. Dorigo, V. Maniezzo, A. Colorni, Ant system: optimization by a colony of cooperating agents, *IEEE Transactionson Systems, Man and Cybernetics: Part B* Vol. 26, No. 1, pp. 29–41, 1996.

[14] 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, pp. 302–309, 1996.

[15] R. Ravi , R. Sundaram , M.V. Marathe , D. J. Rosenkrantz , S. S. Ravi, Spanning trees short or small, *SIAM Journal on Discrete Mathematics*, Vol. 9, No.2, pp. 178–200, 1996.

[16] M. Dorigo, L. M. Gambardella. Ant colony system: a cooperative learning approach to the traveling salesman problem, *IEEE Transactions on Evolutionary Computation* Vol. 1, No. 1, pp. 53–66, 1997.

[17] 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, pp. 87–114, 1997.

[18] 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.

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

1997.

[20] J. Jornsten, A. Lokketangen, Tabu search for weighted k-cardinality trees, *Asia-Pacific Journal of Operational Research*, Vol. 14, No.2, pp. 9–26, 1997.

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

[22] S. Arya and H. Ramesh, A 2.5 factor approximation algorithm for the k-MST problem, *Information Processing Letters*, 65(3), pp. 117–118, 1998.

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

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

[25] S. Arora, G. Karakostas, A $2 + \varepsilon$ approximation algorithm for the k-MST problem, in:*Proceedings of the SIAM Symposium on Discrete Algorithm,*2000, SIAM, Philadelphia, pp. 754-759, 2000.

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

[27] M. J. Blesa, F. Xhafa, A C++ Implementation of Tabu Search for k-Cardinality Tree Problem Based on Generic Programming and Component Reuse, *GCSE Young Researchers Workshop*, pp. 9–12, 2000.

[28] C. Blum, Ant Colony Optimization for the Edge-Weighted k-Cardinality Tree Problem, *GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 27–34, 2002.

[29] C. Blum, M. Ehrgott, Local search algorithms for the k-cardinality tree problem, *Discrete Applied Mathematics*, Volume 128, Issues 2-3, Pp. 511–540, 2003.

[30] C. Blum, A. Roli, Metaheuristics in combinatorial optimization: Overview and conceptual comparison, *ACM COMPUTING SURVEYS*, pp. 268–308, 2003.

[31] P. Moscato and C. Cotta, A gentle introduction to memetic algorithms. *Handbook of Metaheuristics, F. Glover and G.Kochenberger, Eds. Boston*, pp. 105–44, 2003.

[32] T. N. Bui and G. Sundarraj, Ant System for the k-Cardinality Tree Problem, *Genetic and Evolutionary Computation – GECCO 2004 Lecture Notes in Computer Science*, Volume 3102, pp. 36–47, 2004.

[33] D. Urosevic, J. Brimberg, N. Mladenovic, Variable neighbourhood decomposition search for the edge weighted k-cardinality tree problem,*Computers & Operations Research* 31, pp. 1205–1213, 2004.

[34] C. Blum, M. Blesa, New metaheuristic approaches for the edge-weighted k-cardinality tree problem, *Computers & Operations Research* 32, pp. 1355-1377, 2005.

[35] N. Garg, Saving an epsilon: a 2-approximation for the k-MST problem in graphs, *STOC '05 Proceedings of the thirty-seventh annual ACM symposium on Theory of computing*, pp. 396–402, 2005.

[36] C. Blum, A new hybrid evolutionary algorithm for the huge k-cardinality tree problem, *GECCO '06 Proceedings of the 8th annual conference on Genetic and evolutionary computation*, pp. 515-522, 2006.

[37] M. Bruglieri, M. Ehrgott, H. W. Hamacher, F. Maffioli, An annotated bibliography of combinatorial optimization problems with fixed cardinality constraints, *Discrete Applied Mathematics*, 154, pp. 1344–1357, 2006.

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

[39] C. Blum, M. J. B. Aguilera, A. Roli, M. Sampels, *Hybrid Metaheuristics: An Emerging Approach to Optimization*, Studies in Computational Intelligence 114,

Springer, 2008.

[40] J.-S. Chen, J. C.-H. Pan, C.-K. Wu, Hybrid tabu search for re-entrant permutation flow-shop scheduling problem, *Expert Systems with Applications*, Vol. 34, pp. 1924–1930, 2008.

[41] S.-Y. Tseng, C.-C. Lin, Y.-M. Huang, Ant colony-based algorithm for constructing broadcasting tree with degree and delay constraints, *Expert Systems with Applications*, Vol. 35, pp. 1473–1481, 2008.

[42] T. Oncan, J.-F. Cordeau, G. Laporte, A tabu search heuristic for the generalized minimum spanning tree problem, *European Journal of Operational Research*, Vol. 191, pp. 306–319, 2008.

[43] F. P. Quintao, A.S. da Cunha, G.R. Mateus, Integer Programming Formulations for the k-Cardinality Tree Problem, *Electronic Notes in Discrete Mathematics*, 30, pp. 225–230, 2008.

[44] H. Abdallah, H.M. Emara, H.T. Dorrah, A. Bahgat, Using ant colony optimization algorithm for solving project management problems, *Expert Systems with Applications*, Vol. 36, pp. 10004–10015, 2009.

[45] C.H. Aladag, G. Hocaoglu, M.A. Basaran, The effect of neighborhood structures on tabu search algorithm in solving course timetabling problem, *Expert Systems with Applications*, Vol. 36, pp. 12349–12356, 2009.

[46] C. Blum and M. J. Blesa, Solving the KCT problem: large-scale neighborhood search and solution merging. *Optimization Techniques for Solving Complex Problems*, pp. 407-421, 2009.

[47] F. Liu, G. Zeng, Study of genetic algorithm with reinforcement learning to solve the TSP, *Expert Systems with Applications*, Vol. 36, No. 2, pp. 6995–7001, 2009.

[48] H. M. Naimi, N. Taherinejad, New robust and efficient ant colony algorithms: Using new interpretation of local updating process, *Expert Systems with Applications*, Vol. 36, No. 1, pp. 481–488, 2009.

[49] K. Tang, Y. Mei, X. Yao, Memetic Algorithm With Extended Neighborhood Search for Capacitated Arc Routing Problems, *IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION*, VOL. 13, NO. 5, pp. 1151–1166, 2009.

[50] S.-C. Li, Q.-S. Zhu, Z. Yan, H.-L. Yan, Ant colony optimization for nonlinear AVO inversion of network traffic allocation optimization, *Expert Systems with Applications*, Vol. 37, pp. 8343–8347, 2010.

[51] G. Gutin, D. Karapetyan, A memetic algorithm for the generalized traveling salesman problem, *Natural Computing: an international journal*, Volume 9 Issue 1, pp. 47 - 60, 2010.

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

[53] F. B. Pereira, J. Tavares, Bio-inspired Algorithms for the Vehicle Routing Problem, *Studies in Computational Intelligence*, Volume 161, Springer, 2009.