Knowledge Guided Tabu Search for the Prize Collecting Steiner Tree Problem in Graphs

Zhang-Hua Fu, Jin-Kao Hao* (corresponding author)

LERIA, Université d'Angers, 2 Boulevard Lavoisier, 49045 Angers Cedex 01, France
{fu,hao}@info.univ-angers.fr

Abstract. Given an undirected graph with prizes associated with its vertices and costs associated with its edges, the *prize-collecting Steiner tree problem in graphs (PCSPG)* consists of finding a subtree of this graph, so as to minimize the sum of the costs of its edges plus the prizes of the vertices not spanned. In this paper, we propose a knowledge guided tabu search (named K-TS) algorithm for the PCSPG which integrates several new (and important) ingredients, including two path-based move operators for generating neighboring solutions, a tabu search procedure for local optimization, two knowledge guided perturbation operators for escaping from local optimum, as well as a knowledge updating mechanism. Specially, for instances with special structures, we implement an innovative swap-vertex move operator which is shown to be significantly effective. Experiments results based on a subset of representative PCSPG benchmarks show that, the proposed K-TS algorithm is overall highly effective, especially on several groups of special instances which are extremely difficult for the existing algorithms. K-TS also produces a number of remarkable results on the rooted version of PCSPG and the classical SPG.

Keywords: Network design, prize-collecting Steiner tree problem, knowledge guided tabu search, swap-vertex move operator.

1 Introduction

Given an undirected graph $G = (V, E)$ with a set V ($|V| = n$) of vertices and a set E ($|E| = m$) of edges. Each vertex $i \in V$ is associated with a real-valued prize $p_i \geq 0$ (vertex v is a customer vertex if $p_i > 0$, and a non-customer or Steiner vertex otherwise), and each edge $e \in E$ is associated with a real-valued cost $c_e \geq 0$. Then the *prize-collecting Steiner tree problem in graphs (PCSPG)* aims to find a subtree T (with vertices set V_T and edges set E_T respectively) of G, so as to minimize the sum of the costs of its edges plus the prizes of the vertices not spanned by T, i.e.:

$$Minimize\ f(T) = \sum_{e \in E_T} c_e + \sum_{i \notin V_T} p_i. \tag{1}$$

Specifically, if an additional source vertex is chosen as the root which must be part of any feasible solution, we get a rooted version of the PCSPG (named RPCST for short).

Moreover, the classical Steiner tree problem in graphs (SPG) [1] is a particular case of the PCSPG, if each terminal is associated with a high enough prize (corresponding to a customer vertex of the PCSPG, to ensure that every customer vertex must be spanned by any optimal PCSPG solution), and each Steiner vertex is associated with a prize equals to zero. Given that the classical SPG is NP-hard [2], the PCSPG is at least as difficult as the SPG in the general case.

The PCSPG is relevant to model a number of network design problems, e.g., optic fibers, electricity, transportation, distribution (water, gas, heat) supply, etc. Due to its theoretical importance and wide practical applications, the PCSPG has been extensively investigated since it was proposed [3, 4] (where it firstly appeared as the so called *node weighted Steiner tree problem*). Many approaches have been proposed for solving the PCSPG, which could be mainly classified into three categories: approximation algorithms, exact algorithms and heuristics. Specifically, as a preprocessing technique, reduction tests like those introduced in [5] are used to transform the original graph to an equivalent graph of reduced size.

Among the approximation algorithms, a factor 3 approximation algorithm was first proposed in [6]. Later, Goemans and Williamson [7] used a primal-dual scheme to derive a $(2 - \frac{1}{n-1})$-approximation for the rooted PCSPG within complexity $O(n^2 \log n)$. By trying all possible choices for the root, they obtained a $(2 - \frac{1}{n-1})$-approximation for the un-rooted PCSPG within a complexity of $O(n^3 \log n)$ [8]. Furthermore, a $(2 - \frac{1}{n-1})$-approximation algorithm within $O(n^2 \log n)$ running time was introduced in [9] for the un-rooted PCSPG, which was subsequently extended to the rooted PCSPG in [10]. The new algorithm in [11] achieves an approximation ratio of $(2 - \frac{2}{n})$ within $O(n^2 \log n)$ time. Recently, Archer et al. [12, 13] reported the best approximation ratio below 1.9672.

Several exact approaches based on different integer programming formulations are studied for the PCSPG, which aim to find provably optimal solutions

(or improved lower bounds). Respectively, Lucena and Resende [14] presented a polyhedral cutting plane based algorithm, which yields 96 optimal solutions out of the 114 classical test instances (with up to 1000 vertices and 25000 edges). Ljubić et al. [15] re-formulated the PCSPG in a directed formulation and implemented a branch-and-cut algorithm which yields outstanding results (with all the 189 test instances solved to optimality, including 35 real-world instances with up to 1825 vertices and 214095 edges). Cunha et al. [16] used a *Lagrangian non delayed relax and cut (NDRC)* algorithm to generate primal and dual bounds to the problem. The proposed algorithm incorporates ingredients such as a new reduction test, a Lagrangian heuristic and a modification in the NDRC framework. Experimental results on a new group of difficult instances showed its competitiveness for solving the PCSPG.

As for heuristics, Canuto et al. [17] developed a multi-start local search algorithm for the PCSPG, based on a primal-dual algorithm for solution initialization, and a path-relinking procedure for post-optimization. Klau et al. [18] developed a new algorithmic framework consisting of an extensive preprocessing phase, a memetic search framework incorporated with an ILP-based exact subroutine. Goldbarg et al. [19] developed a trans-genetic algorithm hybridized with path-relinking. Biazzo et al. [20] studied the behavior of an algorithm derived from the cavity method, based on the zero temperature limit of the cavity equations. Notice that even the current best heuristic is not able to attain the known optimal values for all the existing benchmarks, meaning that there is still space for further improvement.

In this work, we propose several effective strategies for solving different types of PCSPG instances accordingly. Basically, for the most general instances, we develop a knowledge based tabu search (K-TS) algorithm, which incorporates four basic move operators (including two newly designed path-based move operators) for generating neighboring solutions, a tabu search procedure for local optimization, and two knowledge guided perturbation strategies for escaping from local optimum, as well as a learning based mechanism which aims to fully utilize the historical information discovered during the search process. Based on these ingredients, for some special instances, we implement for the first time a powerful swap-vertex based move operator, and combine it with the four basic move operators to form an enhanced algorithm. Experimental results on a subset of selected PCSPG challenging benchmarks show that, K-TS produces quite competitive results within reasonable time, especially for the special instances which have shown to be extremely difficult for the existing algorithms, indicating the importance of the swap-vertex based move operator. Finally, K-TS (with very slight adaption) also produces remarkable results on the rooted PCSPG and the classical SPG.

The remainder of this paper is organized as follows: Section 2 presents the main ideas of the proposed K-TS algorithm. Section 3 reports the results based on a number of selected challenging benchmarks. Finally, Section 4 concludes this paper.

2 Proposed Approach for the PCSPG

Algorithm 1 Outline of the proposed approach for the PCSPG

```
1: Input: Graph G(V, E)
2: Output: The best found solution
3:
4: Identify the type of the input graph G
5:
6: if G belongs to large type then
7:    Call a simple iterated local search to solve G
8: else
9:    if G belongs to Special type then
10:      Call an enhanced iterated tabu search to solve G
11:   else
12:      Call a knowledge guided iterated tabu search to solve G
13:   end if
14: end if
15:
16: return The overall best found solution
```

The proposed algorithm is outlined in Algorithm 1. After reading the input graph, we at first attempt to identify its type, and then accordingly call a subroutine to solve it. More details are described below.

2.1 Solution Representation

According to the problem definition, the optimal solution of the PCSPG must be a minimum spanning tree (MST) over the spanned vertices, indicating that a given solution could be uniquely represented by its spanned vertices. However, for the sake of efficient local search, we adopt the solution representation method used in [21, 22]. Given a specified root vertex, we uniquely represent each feasible solution by a one-dimensional vector $T = \{t_i, i \in V\}$, where t_i denotes the parent vertex of vertex i if $i \in V_T$ (excluding the root vertex, V_T denotes the vertices set of T), or $t_i = Null$ otherwise. Note that in the broadly defined PCSPG, there is no vertex which should be necessarily spanned by any feasible solution. In the case that the current root vertex is deleted, we re-choose the customer vertex spanned by T with the highest prize as the new root vertex (all the t_i values should be renewed subsequently).

2.2 Reduction Tests

For the sake of simplicity, we do not use reduction test in the proposed algorithm, although reduction tests have shown to be rather powerful in many cases [5]. Definitively, this would be a direction for further improvement.

2.3 Identify Instance Type

After reading the instance, we at first try to identify some special instances. For this purpose, we define as follows a normalized edge costs deviation σ.

$$\sigma = \frac{1}{|E|} \sum_{e \in E} \frac{|c_e - \overline{c_e}|}{\overline{c_e}}. \tag{2}$$

where $\overline{c_e} = \frac{\sum_{e \in E} c_e}{|E|}$ is the average cost of all the edges $e \in E$. Intuitively, a lower value of σ generally indicates lower differences of all the edge costs. Specifically, if $\sigma = 0$, it means all the edges have an uniform cost. This information may be useful to guide the search. For example, in the case $\sigma = 0$, once the set of customers are fixed, the solution cost completely depends on the number of Steiner vertices. Inspired by this observation, we identify the instances with $\sigma < 0.05$ as special instances and call the remaining instances as general ones.

For the general instances, we develop a knowledge guided iterated tabu search algorithm (detailed in Section 2.7). Based on it, in order to take advantage of the information of special instances, we develop a new swap-vertex move operator and combine it with the conventional techniques to tackle the special instances (detailed in Section 2.8).

The above algorithms, in their current form, are not suitable to solve large instances with more than 5000 vertices. Alternatively, we use a simple local search to find feasible solutions within reasonable time (detailed in Section 2.9).

In the following subsections, we first describe several commonly used basic elements, and then present the techniques developed for solving the general

and the special instances respectively. Finally we show how we deal with large instances with more than 5000 vertices.

2.4 Preparation

For small or mid-sized instances (with $n \leq 5000$), before running the algorithm, we first calculate and store the costs of the shortest path between any pair of vertices. Using Dijkstra's algorithm (with the aid of binary heap), this could be achieved within a complexity of $O(m\ n\ \log\ n)$. These values are stored in a $n \times n$ table and can be fetched directly during the search process, instead of recalculating them repeatedly.

2.5 Solution Initialization

In order to provide an initial solution, starting from a randomly chosen customer vertex i (let i be the original root vertex), we iteratively connect an enough good customer (while guaranteeing that the objective value after insertion would not increase), using the shortest path between the selected customer vertex and the incumbent partial solution. If there are more than one such customers available at an iteration, one of them is randomly chosen for connection. This process is repeated until no customer could be further connected. After that, in order to further improve the obtained solution, we use Kruskal's algorithm to re-construct a MST over the spanned vertices, to obtain a feasible solution that serves as the starting point of following algorithms.

This initialization method is uniform for all instances. The only slight difference is for large instances, the needed shortest paths are calculated temporarily, instead of previously calculating them in a pre-processing step (Section 2.4).

2.6 Basic Move Operators

The move operators are the basic elements of local search based approaches. Therefore, we first introduce several basic move operators for generating neighboring solutions, including two conventional move operators developed in [17], and two path-based move operators newly developed in this paper. These four move operators are effective for solving general instances. In Section 2.8, we will introduce a new swap-vertex move operator specifically designed for the special instances.

As mentioned in Section 2.1, any optimal solution of the PCSPG could be uniquely characterized by its spanned vertices set V_T. Consequently, given a solution $T = (V_T, E_T)$, if we insert a vertex $i \notin V_T$ to (respectively, remove a vertex $i \in V_T$ from) V_T, the resulting minimum spanning tree (MST) is a neighboring solution of T (the unfeasible moves should be discarded), denoted by $\text{MST}(V_T \cup \{i\})$ (respectively, $\text{MST}(V_T \backslash \{i\})$).

Fig. 1 shows an example of applying insert vertex or delete vertex moves, where (b) is a sub-tree of the original graph (a), while (c) and (d) are improved

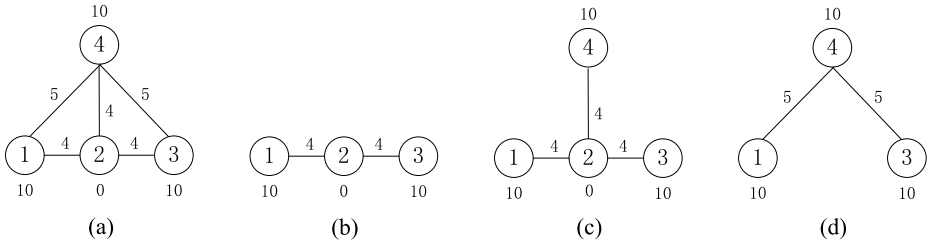

Fig. 1. Example of applying insert vertex or delete vertex moves

neighboring solutions obtained by inserting vertex 4 and deleting vertex 2 subsequently.

Corresponding to these two conventional move operators [17], two sub-neighborhoods $N1(T)$, $N2(T)$ could be defined as follows:

$$\begin{aligned} N1(T) &= \{\text{MST}(V_T \cup \{i\}),\ \forall\ i \notin V_T\}, \\ N2(T) &= \{\text{MST}(V_T \backslash \{i\}),\ \forall\ i \in V_T\}. \end{aligned} \tag{3}$$

Using novel dynamic data structures slightly adapted from [23], at each iteration of local search, all the neighboring solutions belonging to $N1(T)$ (respectively, $N2(T)$) could be evaluated in $O(m \log n)$.

In addition to these two conventional move operators, we implement (for the first time to our knowledge) as follows two other move operators specifically designed for the PCSPG, which mainly focus on customer vertices.

1. *Connect a Customer Vertex*: Add a path to connect a customer vertex $i \notin V_T$, using the shortest path between vertex i and the incumbent solution T. The resulting neighboring solution is denoted by $T \oplus \text{Connect\_Customer}(T, i)$, whose objective value would be increased by the total cost on the inserted path minus the total prize of the inserted vertices.
2. *Disconnect a Customer Vertex*: Disconnect a customer vertex $i \in V_T$ (for convenience, we only consider leaf customer vertex here), by deleting vertex i associated with the edges which become useless (see [21] for more details about how to disconnect a leaf customer vertex). The corresponding neighboring solution is denoted by $T \oplus \text{Disconnect\_Customer}(T, i)$. The objective value would be decreased by the total cost of the deleted edges minus the total prizes of deleted vertices.

For example, as shown in Fig. 2, starting from a sub-tree (b) of graph (a), it could be improved to (c) and (d) by connecting customer 3 and disconnecting customer 9 sequentially.

Similarly, corresponding to these two new move operators, two sub-neighborhoods ($N3(T)$ and $N4(T)$ respectively) of the incumbent solution T are defined as follows:

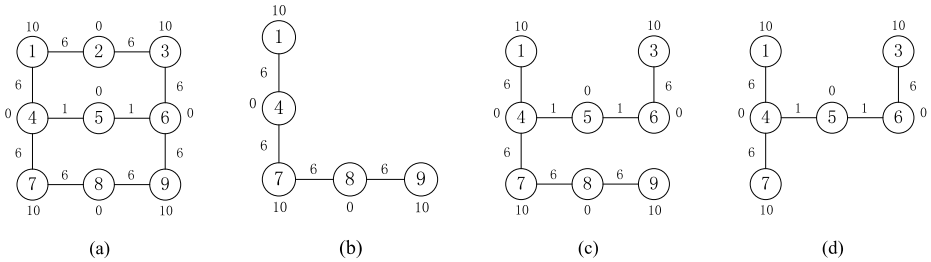

Fig. 2. Example of applying connect customer or disconnect customer moves

$$\begin{aligned} N3(T) &= \{T \oplus \text{Connect\_Customer}(T, i), \forall\ i \notin V_T, p_i > 0\}, \\ N4(T) &= \{T \oplus \text{Disconnect\_Customer}(T, i), \forall \text{ leaf vertex } i \in V_T, p_i > 0\}. \end{aligned} \tag{4}$$

Given these four sub-neighborhoods $N1$-$N4$, we define the composed neighborhood $N(T)$ as:

$$N(T) = N1(T) \cup N2(T) \cup N3(T) \cup N4(T). \tag{5}$$

2.7 Algorithm for Solving General Instances

We use a knowledge guided iterated tabu search algorithm to solve the general instances (outlined in Algorithm 2). For preparation, we use a one-dimensional vector Z (with all values initialized to zeros) to record the frequency information (knowledge) of each vertex appearing in historically visited local optima. Accordingly, the number Cnt of visited local optima is also initialized to 0. After that, the algorithm repeatedly generates (randomly reconstruct from scratch or perturb from the incumbent solution) an initial solution and then calls a tabu search to improve the solution to a local optimum. The obtained solution is subsequently used to update vector Z, which would be used in the perturbation process. Once the stop condition is met, the best found solution is returned as the final solution.

During the first ten times (pre-learning phase), the initial solutions are randomly constructed (to reinforce diversity) using the method described in Section 2.5. After that, the search alternates between weak perturbations and strong perturbations to generate initial solutions. The search terminates once no further improvement is gained after a given number of (100 in this paper) consecutive rounds of solution initialization followed by tabu search. More details are given below.

Tabu Search for Local Optimization Typically, starting from a given initial solution T, the tabu-search procedure evaluates the neighboring solutions of $N(T) = N1(T) \cup N2(T) \cup N3(T) \cup N4(T)$ and iteratively replaces the incumbent

Algorithm 2 Algorithm for solving **General** instances

Input: Graph $G(V, E)$)
Output: The best found solution

Let $Z = \{Z_1, Z_2, \ldots, Z_n\}$ be a one-dimensional vector recording the frequency information (knowledge) of each vertex appearing in historically visited local optima
for Each vertex indexed i **do**
    $Z_i \leftarrow 0$
end for
Let T denote the incumbent solution
$Cnt \leftarrow 0$
while The stop condition is not met **do**
    if $Cnt < 10$ **then**
        $T \leftarrow Randomly\_Construct\_Initial\_Solution(G)$
    else
        if Cnt Mod 10=0 **then**
            $T \leftarrow Knowledge\_Guided\_Strongly\_Perturb(G, Z, Cnt)$
        else
            $T \leftarrow Knowledge\_Guided\_Weakly\_Perturb(G, Z, Cnt)$
        end if
    end if
    $T \leftarrow Tabu\_Search(T)$
    $Z \leftarrow Update\_Vector(Z, T)$
    $Cnt \leftarrow Cnt + 1$
end while
return The best found solution

solution with a best neighboring solution. Nevertheless, using this local optimization procedure, local cycling may occur in some special cases. To address this drawback, we introduce as follows some memory-based mechanisms, to form a tabu search subroutine [24]. For each vertex i (customer of non-customer), we save in an array the last iteration I_i when vertex i is included into or removed from the current solution. Then, before applying any one of above four move operators (corresponding to vertex i), we check at first whether the current iteration index is larger than $I_i + l_{in}$ (if $i \notin V_T$) or $I_i + l_{out}$ (if $i \in V_T$), where l_{in} and l_{out} are parameters indicating the length of the prohibition, i.e., the tabu tenures [24] (in this paper, l_{in} and l_{out} are randomly distributed within $[3, 3 + \frac{|V|-|V_T|}{5}]$ and $[3, 3 + \frac{|V_T|}{5}]$ respectively). If this is not the case, the corresponding move is marked tabu and is thus prohibited, otherwise it is declared non-tabu that can be applied freely. This mechanism aims to avoid the inclusion of a recently removed vertex or the removal of a recently included vertex, unless the move meets the aspiration criterion, i.e., leading to a solution better than the overall best found solution. Guided by the above information, the tabu search subroutine examines all the non-tabu legal moves of $N(T)$ and iteratively applies the best legal move to the incumbent solution (no matter it leads to an improved solution or not), until the incumbent solution could not be improved after a given number M (fixed to 30 in this paper) of consecutive iterations. At this point, the best met solution T is returned as the solution found of one run of tabu search.

Knowledge Maintaining In order to maintain the knowledge learned during the search process, after each run of tabu search, we use the obtained solution T to update the frequency vector Z as follows: for each vertex i, if $i \in V_T$, we increase Z_i by 1; otherwise we keep Z_i unchanged. Synchronously, variable Cnt is also increased by 1. After that, the search enters into the next round of solution perturbation followed by tabu search. This process is repeated, until some specifically designed stop condition (no further improvement is gained after 100 consecutive rounds) is met.

Perturbation Strategies After the pre-learning phase (first ten rounds), once tabu-search reaches a local optimum T, we try to perturb it to a new solution. For the tradeoff between intensification and diversification, we develop respectively a weak perturbation and a strong perturbation as follows.

For weak perturbation, we try to flip the status (insert an un-spanned vertex or remove a spanned vertex) of a number of vertices. More precisely, after randomly choosing a vertex i, if $i \in V_T$, we flip its status with probability $\frac{Cnt-Z_i+1}{Cnt+1}$. Otherwise, we flip its status with probability $\frac{Z_i+1}{Cnt+1}$ ("+1" is to ensure that the resulting probability belongs to $(0, 1)$) (unfeasible flips should be discarded). This process is repeated until a given number (randomly chosen between $[1, |V_T|]$) of flips have been applied, resulting a new solution different from T.

In addition to the weak perturbation, we also implement a strong perturbation operator which reconstructs a solution after every ten rounds. For this, we

first select a subset of promising customer vertices for connection, guided by the knowledge stored in the frequency vector Z. More precisely, for each customer vertex i, we add it to a candidate customer vertices set CV with probability $\frac{Z_i+1}{Cnt+1}$. If no customer vertex is selected, a randomly chosen customer vertex is added into CV, to ensure that CV is not empty. After that, starting from a customer vertex i randomly chosen from CV, we iteratively connect the nearest customer vertex of CV, using the shortest path between the selected customer vertex and the incumbent partial solution, until all the customer vertices of CV are connected. After that, we re-construct a minimum spanning tree (MST) over the spanned vertices, to obtain a new feasible solution that serves as the starting point of the next round of tabu search.

2.8 Algorithm for Solving Special Instances

Algorithm 3 Algorithm for solving **Special** instances

Input: Graph $G(V, E)$)
Output: The best found solution
Let T denote the incumbent solution
$Cnt \leftarrow 0$
while The stop condition is not met **do**
  if $Cnt < 1$ **then**
    $T \leftarrow Randomly\_Construct\_Initial\_Solution(G)$
  else
    if Cnt Mod 10=0 **then**
      $T \leftarrow Swap\_Based\_Strongly\_Perturb(G)$
    else
      $T \leftarrow Swap\_Based\_Weakly\_Perturb(G)$
    end if
  end if

  $T \leftarrow Tabu\_Search(T)$
  $T \leftarrow Enhanced\_Local\_Search(T)$

  $Cnt \leftarrow Cnt + 1$
end while
return The best found solution

The knowledge based iterated tabu search described above is quite effective for the general instances, however, they do not perform very well for the special instances (with $\sigma < 0.05$), possibly due to their irregular landscapes. To clarify this point, we consider an instance with uniform edge cost ($c_e = 1,\ \forall e \in E$) and highly enough prize of each customer for example. Clearly, in this case, all the customers should be connected, thus the objective value completely depends on the number of used Steiner vertices. It means optimizing the objective value

is essentially equivalent to reducing the Steiner vertices. More precisely, adding a Steiner vertex (if feasible) would definitively increase the solution cost by 1, leading to a worse solution. On the contrast, although deleting a Steiner decreases the objective value by 1, the search process is very easy to get stuck into a local optimum where no feasible deleting move is possible. At this point, it seems very difficult to escape from the incumbent local optimum by applying only the above four basic move operators, even with the aid of tabu search. In order to overcome this drawback, we implement a new swap-vertex based move operator and combine it with the four basic move operators in a variable neighborhood mode.

As outlined in Algorithm 3, starting from a randomly constructed solution, we first try to improve it by the tabu search procedure described above, based on the four basic move operators. After that, the search enters into an enhanced local search phase, which combines the four basic move operators and the new swap-vertex move operator. Once no improvement is possible, the incumbent solution is perturbed to a new solution (using strong perturbation or weak perturbation), and then the search enters into a new round of tabu search followed by enhanced local search again. This process is repeated until the incumbent solution could not be further improved after a given number of (100 in this paper) consecutive rounds of solution initialization followed by tabu search and enhanced local search. More details are described below.

Swap-Vertex Based Move Operator Typically, the swap-vertex based move operator adds a vertex $i \notin T$ to T and removes another vertex $j \in T$ from T, leading to a new solution (the unfeasible moves should be discarded). This idea is natural, but to our best knowledge, until now the swap operator has not been well implemented in the field of Steiner tree problems, possibly due to its unreasonably high computational complexity. Indeed, at each iteration there are a total of $O(|V_T|) \times O(|V| - |V_T|) \leq O(n^2)$ possible swap moves. If we reconstruct a MST (using kruskal's algorithm with the aid of Fibonacci heap) after applying each swap-vertex move, the overall complexity would be $O(n^2) \times O(m + n\log\, n)$, being unaffordable for a local search based approach.

Fortunately, for the special instance with σ equals to zero (or nearly zero), the computations could be much simplified. Without loss of generality, we consider the case with uniform edge costs ($\sigma = 0$) at first. Obviously, in this case, if swapping vertex $i \notin T$ and vertex $j \in T$ leads to a feasible solution, the objective value would definitively decrease by $r_i - r_j$, because the consumed cost remains unchanged and the collected prize varies by $r_i - r_j$. Specifically, swapping two Steiner vertices would never change the objective value. Obviously, these computations are much easier, and could be realised in $O(n^2)$ for all $O(n^2)$ possible swap-vertex moves.

However, we have to consider three additional questions: (1) How to verify the solution feasibility after applying a swap-vertex move? (2) How to distinguish the feasible moves corresponding to the same objective difference (denoted by

Δ hereafter)? and (3) How to deal with the case if σ does not strictly equal to 0 (the edge costs slightly differ from each other)?

For the first question, using some novel data structures such as union find set and leftist heap (the detailed techniques are somewhat complex and omitted here, due to page limitation), the feasibility of all the $O(n^2)$ possible swap-vertex moves could be examined within an overall complexity of $O(m\log\, n) + O(m \times |V_T|) \leq O(m \times n)$ (unfeasible moves are discarded directly). This complexity seems high, but still remains affordable for sparse graphs (with $O(m) \approx O(n)$), especially when the solution is of small size ($|V_T| \ll n$).

The second question is much more difficult because there are usually a large number of swap-vertex moves corresponding to the same Δ value (e.g., swapping any two Steiner vertices would lead to a $\Delta = 0$). Using the objective value as the evaluation function will not be able to guide the search since this evaluation function cannot distinguish neighboring solutions of the same objective value even if they can lead to different search trajectories and thus different local optima. In order to guide the search towards more promising search regions, relative to each feasible solution T, we define as follows an auxiliary function.

Definition 1. For each vertex $i \in V_T$, its *special degree* sd_i is defined as the number of vertices belonging to V_T which are directly reachable from i. Precisely, we say a vertex $j \neq i$ is directly reachable from vertex i if $c_{ij} < \infty$.

According to this definition, we observe that if $sd_i = 1$, vertex i is only directly reachable from another vertex $j \in V_T$, implying that once vertex j is removed from T, the resulting solution would become unfeasible.

Definition 2. The *special degree sd(T)* of a feasible solution T is defined as:

$$sd(T) = \sum_{i \in V_T, sd_i = 1} 1. \tag{6}$$

Intuitively, the lower the value of $sd(T)$, the larger opportunity to feasibly delete a vertex (thus improve the solution). Inspired by this idea, during the search process, we use the objective value $f(T)$ (Eq. (1)) as the main evaluation criterion, while adopt the special degree $sd(T)$ as an auxiliary evaluation criterion. We say $T1$ dominates $T2$ if $f(T1) < f(T2)$ or $f(T1) = f(T2), sd(T1) < sd(T2)$, so as to distinguish the neighboring solutions with the same objective value.

Fig. 3 shows the benefits of combining swap-vertex move operator with basic move operators, where sub-figure (a) is the input graph and (b) is an initial solution with total collected prize 30 and total consumed cost 4. The corresponding special degree of solution (b) is 2 (both vertex 1 and 3 have a special degree $sd_1 = sd_3 = 1$). Notice that solution (b) could not be further improved by the four basic move operators. However, if swap vertex 2 and 6 to get a new solution (c), although the objective value does not vary, the special degree of solution (c) decreases to 1 (only vertex 3 has a special degree $sd_3 = 1$). From this solution, we could feasibly delete vertex 4, to get an improved solution (d).

The final question related to the swap-vertex move operator is how to adapt it to instances with $\sigma > 0$. The main difficulty here is that the Δ value after

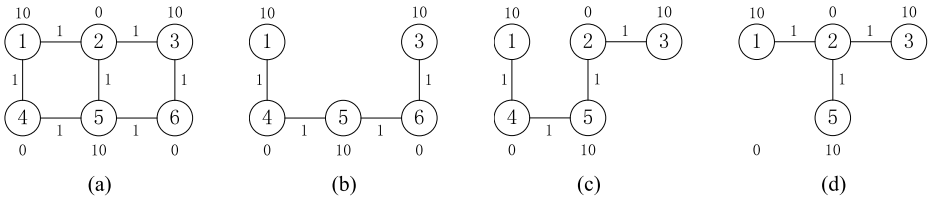

Fig. 3. Example of combining swap-vertex move operator with the basic operators.

swapping vertices i and j no longer strictly equals to $r_i - r_j$ (the total consumed cost may vary). If we choose to calculate Δ exactly after swapping each pair of vertices, the complexity would become unaffordable again. Fortunately, we do not need to do so if σ is small enough (e.g., $\sigma < 0.05$), because in this case swapping two vertices generally only slightly changes the total consumed cost, being almost negligible compared to the overall objective value. This implies that once the customers to connect are fixed, the overall objective value still mainly depends on the number of used Steiner vertices. For this reason, we decide to solve the special instances in two different phases. The first phase is the tabu search procedure (regarding exactly the cost of each edge) described in Section 2.7, using the four basic move operators. The second phase is an enhanced local search procedure (described below), which combines the four basic move operators and the swap-vertex move operator, regardless the edge cost differences. These two phases complement each other.

Enhanced Local Search Once the basic tabu search terminates, the search enter into an enhanced local search phase, which relies on a larger neighborhood by combining the four basic move operators and the swap-vertex based move operator in a variable mode. More precisely, at each iteration, the search process examines at first the neighboring solutions generated by the four basic move operators and accepts the first met improving solution, and then continues to the next iteration. If no improving is found, the search process examines again the neighboring solutions generated by swap-vertex move operator, and accepts the first met improving solution. To identify an improving solution, the objective value $f(T)$ is used as the main criterion, while the special degree $sd(T)$ is used as an auxiliary criterion to break ties.

For the sake of simplicity, we do not adopt tabu mechanism here, thus once no improving solution is found in both neighborhoods, this enhanced local search process terminates. After that, the search turns into a perturbation phase, which is detailed below.

Perturbation Strategies In order to exploit a reasonable tradeoff between intensification and diversification, we develop again a weak perturbation operator and a strong perturbation operator respectively for the special instances. For weak perturbation, we randomly choose an vertex $i \notin V_T$ to insert into T at

first, and then randomly swap $|V_T|$ pairs of Steiner vertices, while guaranteeing feasibility after each step. Adding a vertex before swapping is to increase the number of feasible swap moves, thus increase the opportunity to escape from the incumbent local optimum. The strong perturbation operator works almost the same. The only difference is several (randomly distributed between [3,6]) vertices are added before swapping, to enhance diversity. Note that we no longer use history information to guide the perturbation process, thus no frequency vector Z is needed here.

2.9 Algorithm for Solving Large Instances

Algorithm 4 Algorithm for solving **Large** instances

1: **Input:** Graph $G(V, E)$
2: **Output:** The best found solution
3: Let T denote the incumbent solution
4: **while** The stop condition is not met **do**
5: $T \leftarrow Randomly\_Construct\_Initial\_Solution(G)$
6: $T \leftarrow Simple\_Local\_Search(T)$
7: **end while**
8:
9: **return** The best found solution

Unfortunately, due to our initial implementation choice, the above two algorithms in their present form are not able to solve large instances with more than 5000 vertices (our programs maintain several $n \times n$ matrices, leading to memory excess for large instances). In order to tackle the large instances, we implement alternatively a weaken version of iterated local search algorithm, which is able to find a feasible solution within reasonable time.

As outlined in Algorithm 4, starting from an initial solution randomly constructed by the method described in Section 2.5, the search calls a simple local search procedure to improve it to a local optimum, which examines the feasible neighboring solution (only considers insert vertex move and delete vertex move) in random order and iteratively accepts the first met improving solution, until no such solution exists in the whole neighborhood. After that, the search enters into a new round of solution initialization followed by local search again, until the stop criterion is met (e.g., the allowed time is elapsed).

3 Experimental Results

In this section, we report results obtained by K-TS on PCSPG, RPCST and SPG respectively. The proposed algorithm is coded in C language and executed on a machine with i3-2120 CPU (3.30 GHz) and 3.2 GB RAM (the score obtained after running the test script on our machine is 227.863466).

3.1 Results on the PCSPG

Currently there are 8 groups of PCSPG instances included by the 11th DIMACS challenge, among which 32 instances (4 instances from each group) are selected for the challenge. For each of these 32 instances, we independently run K-TS multiple times, until the allowed time (3600s for each instance) is elapsed. The results are reported in Table 1, where the first three columns indicate the instance name, the number of vertices and edges of the graph. The following column indicates the optimal results (if applicable). The last three columns report the results obtained by our K-TS algorithm, including the historically found overall best solution (regardless of computation time, in order to indicate the discover ability of K-TS), the best solution found within 3600s for each instance, as well as the CPU time (in seconds) to reach the best (found within 3600s) solution.

Table 1. Results obtained by K-TS on 32 PCSPG instances (selected for the challenge)

Instance			Optimal	K-TS		
Name	$\|V\|$	$\|E\|$		Overall Best	Best within 3600s	Time to Best (s)
C13-A	500	2500	236	*236*	*236*	1.19
C19-B	500	12500	146	*146*	*146*	2.62
D03-A	1000	1250	1509	*1509*	*1509*	2.94
D20-A	1000	25000	536	*536*	*536*	11.14
P400.3	400	1175	2951725	*2951725*	*2951725*	40.78
P400.4	400	1144	2852956	*2852956*	*2852956*	10.33
K400.7	400	1442	474466	*474466*	*474466*	1.71
K400.10	400	1507	394191	*394191*	*394191*	66.71
hc10p	1024	5120	N/A	59732	59883	1075.87
hc11u	2048	11264	N/A	1115	1117	1559.02
hc12p	4096	24576	N/A	236478	237043	3226.98
hc12u	4096	24576	N/A	2220	2222	3447.69
bip52nu	2200	7997	222	*222*	223	482.61
bip62nu	1200	10002	214	*214*	*214*	14.84
cc3-12nu	1728	28512	N/A	95	95	28.07
cc12-2nu	4096	24574	N/A	567	570	706.56
i640-001	640	960	2932	*2932*	*2932*	0.01
i640-221	640	204480	8400	*8400*	*8400*	22.75
i640-321	640	204480	N/A	28787	28787	2.97
i640-341	640	40896	N/A	29666	29668	2227.30
a2000.2	2000	16062	1483.8368	1486.337077	1489.495826	1794.66
a4000.3	4000	32025	3406.61873	3411.926814	3411.926814	177.81
a8000.1.2	8000	64373	4719.96527	4757.798841	4757.798841	3070.84
a14000.1.5	1400	112228	9475.59356	9576.048590	9576.048590	773.99
handsd04	42500	84475	N/A	737.715995	737.715995	1365.39
handbd13	169800	338551	N/A	13.212800	13.212800	1308.16
handsi03	39600	78704	56.149422	56.201404	56.201404	83.49
handbi07	158400	315808	150.974258	150.985112	150.985112	633.84
drosophila001	5226	93394	8273.98263	8276.536854	8278.526979	2498.04
HCMV	3863	29293	7371.53637	*7371.536373*	*7371.536373*	17.37
lymphoma	2034	7756	3341.89024	*3341.890237*	*3341.890237*	45.48
metabol_expr_mice	3523	4345	11346.9272	11349.164341	11349.164341	1441.11

As shown in Table 1, for most of the instances (except the very large instances) with known optimal results, K-TS is able to reach quickly an optimal or near-optimal solution. Specifically, for the *hc*, *bip* and *cc* instances with special structure (with $\sigma < 0.05$) which are known to be extremely difficult for the existing algorithms, K-TS produces remarkable results compared to other competitors's results (not shown in the Table). Actually, the main contribution belongs to the swap-vertex based move operator. Indeed, if we remove the swap-vertex based move operator from the proposed algorithm, the results would become much worse, indicating its importance for solving the special instances.

Encouraged by this observation, we decide to do more experiments on all the remaining instances of special structures (*hc*, *bip* and *cc* instances). The detailed results are listed in Table 2, where the meaning of each column keeps in accordance with Table 1 (column "optimal" is removed from the table, because to our knowledge for almost all these instances the optimal results remain unknown).

Table 2. Results obtained by K-TS on 38 special PCSPG instances

Instance			K-TS		
$Name$	$\|V\|$	$\|E\|$	Overall Best	Best within 3600s	Time to Best (s)
hc6p.stp	64	192	3908	3908	1.89
hc6u.stp	64	192	36	36	0.03
hc7p.stp	128	448	7721	7721	1.58
hc7u.stp	128	448	72	72	0.95
hc8p.stp	256	1024	15213	15213	225.63
hc8u.stp	256	1024	143	143	1.84
hc9p.stp	512	2304	30082	30082	3227.84
hc9u.stp	512	2304	283	283	46.55
hc10u.stp	1024	5120	559	559	464.09
hc11p.stp	2048	11264	118981	119029	2855.94
hc6p2.stp	64	192	3923	3923	1.64
hc6u2.stp	64	192	20	20	0.01
hc7p2.stp	128	448	7711	7711	94.39
hc7u2.stp	128	448	47	47	0.03
hc8p2.stp	256	1024	15243	15243	35.08
hc8u2.stp	256	1024	97	97	0.56
hc9p2.stp	512	2304	30240	30240	410.56
hc9u2.stp	512	2304	190	190	4.33
hc10p2.stp	1024	5120	59766	59766	169.83
hc10u2.stp	1024	5120	380	380	237.97
hc11p2.stp	2048	11264	118922	118922	2292.92
hc11u2.stp	2048	11264	751	752	429.24
hc12p2.stp	4096	24576	236872	237144	3048.38
hc12u2.stp	4096	24576	1493	1494	1239.00
bip42nu.stp	1200	3982	227	227	44.50
bipa2nu.stp	3300	18073	325	325	414.89
bipe2nu.stp	550	5013	53	53	0.76
cc3-4nu.stp	64	288	10	10	0.01
cc3-5nu.stp	125	750	17	17	0.03
cc3-10nu.stp	1000	13500	61	61	2.81
cc3-11nu.stp	1331	19965	79	79	15.19
cc5-3nu.stp	243	1215	36	36	10.08
cc6-2nu.stp	64	192	15	15	0.01
cc6-3nu.stp	729	4368	95	95	59.00
cc7-3nu.stp	2187	15308	272	272	120.31
cc9-2nu.stp	512	2304	83	83	948.94
cc10-2nu.stp	1024	5120	168	168	100.81
cc11-2nu.stp	2048	11263	305	305	957.63

3.2 Results on the RPCST

Our K-TS algorithm could also be used to solve the rooted version of PCSPG (RPCST), just after very slight adaption (by assigning the fixed root vertex a high enough prize, to make sure it will always be spanned). Similarly, we test our K-TS algorithm on 29 RPCST instances selected by the 11th DIMACS challenge, using the same termination criterion for each instance. The obtained results are listed in Table 3, where the meaning of each column is the same as in Table 1.

Table 3. Results obtained by K-TS on 29 RPCST instances (selected for the challenge)

Instance			Optimal	K-TS		
Name	$\|V\|$	$\|E\|$		Overall Best	Best within 3600s	Time to Best (s)
Cologne1-i01M1	748	6332	109271.503	*109271.5028*	*109271.5028*	0.01
Cologne1-i01M2	748	6332	315925.31	*315925.3105*	*315925.3105*	1.61
Cologne1-i01M3	748	6332	355625.409	*355625.4089*	*355625.4089*	3.70
Cologne1-i02M1	749	6343	104065.801	*104065.8012*	*104065.8012*	0.01
Cologne1-i02M2	749	6343	352538.819	*352538.8189*	*352538.8189*	1.18
Cologne1-i02M3	749	6343	454365.927	*454365.9275*	*454365.9275*	23.35
Cologne1-i03M1	751	6343	139749.407	*139749.4074*	*139749.4074*	0.01
Cologne1-i03M2	751	6343	407834.228	*407834.2279*	*407834.2279*	1.92
Cologne1-i03M3	751	6343	456125.488	*456125.4880*	*456125.4880*	8.97
Cologne1-i04M2	741	6293	89920.8353	*89920.8354*	*89920.8354*	0.01
Cologne1-i04M3	741	6293	97148.789	*97148.7891*	*97148.7891*	0.64
Cologne1-i05M1	741	6296	26717.2025	*26717.2025*	*26717.2025*	0.01
Cologne1-i05M2	741	6296	100269.619	*100269.6186*	*100269.6186*	0.01
Cologne1-i05M3	741	6296	110351.163	*110351.1633*	*110351.1633*	3.32
Cologne2-i01M2	1803	16743	355467.684	*355467.6844*	*355467.6844*	0.03
Cologne2-i01M3	1803	16743	628833.614	*628833.6143*	*628833.6143*	44.04
Cologne2-i01M4	1803	16743	773398.303	*773398.3026*	*773398.3026*	688.65
Cologne2-i02M2	1804	16740	288946.832	*288946.8318*	*288946.8318*	4.54
Cologne2-i02M3	1804	16740	419184.159	*419184.1590*	*419184.1590*	7.26
Cologne2-i02M4	1804	16740	430034.264	*430034.2639*	*430034.2639*	7.15
Cologne2-i03M2	1809	16762	459894.776	*459894.7765*	*459894.7765*	4.57
Cologne2-i03M3	1809	16762	643062.02	*643062.0196*	*643062.0196*	72.88
Cologne2-i03M4	1809	16762	677733.067	*677733.0673*	*677733.0673*	788.36
Cologne2-i04M2	1801	16719	161700.545	*161700.5453*	*161700.5453*	0.03
Cologne2-i04M3	1801	16719	245287.203	*245287.2026*	*245287.2026*	6.31
Cologne2-i04M4	1801	16719	245287.203	*245287.2026*	*245287.2026*	5.75
Cologne2-i05M2	1810	16794	571031.415	*571031.4154*	*571031.4154*	126.30
Cologne2-i05M3	1810	16794	672403.143	*672403.1432*	*672403.1432*	6.30
Cologne2-i05M4	1810	16794	713973.623	*713973.6228*	*713973.6228*	5.84

As shown in Table 3, K-TS is able to quickly reach the optimal solution on each of these 29 instances, indicating its effectiveness and efficiency for solving the RPCST.

3.3 Results on the SPG

Our K-TS algorithm could also be used to solve the SPG, just with very slight adaption (by assigning each customer a large enough profit). Although we do not intend to participate in the competition about the SPG (due to time limit), we would like to report several remarkable results on the SPG (listed in Table 4), which are better than the best upper bounds known by 01/August/2014, and being at least no worse than the upper bounds released by 12/09/2014.

Table 4. Results obtained by K-TS on 4 difficult SPG instances

Instance			UB by 01/08/2014	UB by 12/09/2014	UB by K-TS
Name	$\|V\|$	$\|E\|$			
hc9p	512	2304	30258	**30242**	*30242*
hc10p	1024	5120	60494	59797	**59732**
hc10u	1024	5120	581	**575**	*575*
hc12p	4096	24576	236949	236949	**236899**

4 Conclusion

The *prize-collecting Steiner tree problem in graphs (PCSPG)* is theoretically important and has many practical applications in network design. In this paper, we propose a knowledge based tabu search (K-TS) for the PCSPG, which incorporates several innovative ingredients and produces highly competitive results not only on the PCSPG, but also on two other closely related problems (the rooted version of PCSPG and the classical SPG).

Acknowledgments

The work was supported by the following projects: RaDaPop and LigeRO (2009-2013, Pays de la Loire Region) and PGMO (2014-2015, Jacques Hadamard Mathematical Foundation).

References

1. Hakimi, S. L. 1971. Steiner's problem in graphs. *Networks*, 1, 113-133.
2. Karp, R. M. 1972. Reducibility among combinatorial problems. Miller, R. E., Thatcher, J. W., eds. *Complexity of Computer Computations*, New York.
3. Segev, A. 1987. The Node Weighted Steiner Tree Problem. *Networks*, 17, 1-17.
4. Engevall, S., Göthe-Lundgren, M., Värbrand, P. 1998. A strong lower bound for the node weighted Steiner tree problem. *Networks*, 31(1998), 11-17.
5. Uchoa, E. 2006. Reduction tests for the prize-collecting Steiner problem. Operations Research Letters, 34, 437-444.
6. Bienstock, D., Goemans, M. X., Simchi-Levi, D., Williamson, D.P. 1993. A note on the prize collecting travelling Salesman problem. *Mathematical Programming*, 59, 413-420.
7. Goemans, M. X., Williamson, D. P. 1995. A general approximation technique for constrained forest problems. *SIAM Journal on Computing*, 24(2), 296-317.
8. Goemans, M. X., Williamson, D. P. 1997. The primal-dual method for approximation algorithms and its application to network design problems. In D. S. Hochbaum editor, Approximation Algorithms for NP-Hard Problems, PWS Publiching Company, Boston, 144-191.
9. Johnson, D., Minkoff, M., Phillips, S. 2000. The prize collecting steiner tree problem: theory and practice. Proceedings of the eleventh annual ACM-SIAM symposium on Discrete algorithms, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 760-769.

10. Minkoff, M. 2000. The prize collecting Steiner tree problem. *Master's thesis*, Departament of Electrical Engineering and Computer Science, Massachusetts Insitute of Technology, USA.
11. Feofiloff, P., Fernandes, C. G., Ferreira, C. E., Pina, J. C. D. 2007. Primal-dual approximation algorithms for the Prize-Collecting Steiner Tree Problem. *Information Processing Letters*, 103, 195-202.
12. Archer, A., Bateni, M., Hajiaghayi, M., Howard Karloff, H. 2011. Improved Approximation Algorithms for Prize-Collecting Steiner Tree and TSP. 2009. Proceedings of the 2009 50th Annual IEEE Symposium on Foundations of Computer Science, 427-436.
13. Archer, A., Bateni, M., Hajiaghayi, M., Howard Karloff, H. 2011. Improved Approximation Algorithms for Prize-Collecting Steiner Tree and TSP. *SIAM Journal on Computing*, 40(2), 309-332.
14. Lucena, A., Resende M. G. C. 2004. Strong lower bounds for the prize collecting Steiner problem in graphs. *Discrete Applied Mathmatics*, 141(1-3), 277-294.
15. Ljubić, I.,Weiskircher, R., Pferschy, U., Klau, G., Mutzel, P., Fischetti, M. 2006. An algorithmic framework for the exact solution of the prize-collecting Steiner tree problem. *Mathmatical Programming: Series B*, 105, 427-449.
16. Salles da Cunha, A., Lucena, A., Maculan, N., Resende, M. G. C. 2009. A relax-and-cut algorithm for the prize-collecting Steiner problem in graphs. *Discrete Applied Mathematics*, 157, 1198-1217.
17. Canuto, S. A., Resende, M. G. C., Ribeiro, C. C. 2001. Local search with perturbations for the prize-collecting Steiner tree problem in graphs. *Networks*, 38, 50-58.
18. Klau, W., Ljubić I., Moser, A., Mutzel, P., Neuner, P., Pferschy, U., Raidl, G., Weiskircher, R. 2004. Combining a memetic algorithm with integer programming to solve the prize collecting Steiner tree problem. *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2004), Lecture Notes in Computer Science*, 3102, 1304, 1315.
19. Goldbarg, E. F. G., Goldbarg, M. C., Schmidt, C. C. 2008. A hybrid transgenetic algorithm for the prize collecting Steiner tree problem. *Journal of Universal Computer Science*, 14(15), 2491-2511.
20. Biazzo, I., Braunstein, A., Zecchina, R. On the performance of a cavity method based algorithm for the Prize-Collecting Steiner Tree Problem on graphs. *Physical Review: E*, 86, 026706.
21. Fu, Z. H., Hao, J. K. 2014. Breakout local search for the Steiner tree problem with revenue, budget and hop constraints. *European Journal of Operational Research*, 232(1), 209-220.
22. Fu, Z. H., Hao, J. K. 2014. Dynamic programming driven memetic search for the Steiner tree problem with revenue, budget and hop constraints. *INFORMS Journal on Computing*, accepted, forthcoming.
23. Uchoa, E., Werneck, R. F. F. 2010. Fast local search for Steiner trees in graphs. *In Blelloch, G.E., Halperin, D., eds.: ALENEX, SIAM*, 1-10.
24. Glover, F., & Laguna, M., 1997. Tabu search. Kluwer Academic Publishers.
25. Polzin, T., Daneshmand, S.V. 2001. Improved algorithms for the Steiner problem in networks. *Discrete Applied Mathematics*, 112, 263-300.
26. Beasley, J.E., 1990. OR-Library: Distributing test problems by electornic mail. *Journal of the Operational Research Society*, 41, 1069-1072.