J Heuristics (2008) 14: 69–93
DOI 10.1007/s10732-007-9027-1

The prize-collecting generalized minimum spanning tree problem

Bruce Golden · S. Raghavan · Daliborka Stanojević

Received: 31 December 2005 / Revised: 12 October 2006 /
Accepted: 16 October 2006 / Published online: 15 May 2007

Abstract We introduce the prize-collecting generalized minimum spanning tree problem. In this problem a network of node clusters needs to be connected via a tree architecture using exactly one node per cluster. Nodes in each cluster compete by offering a payment for selection. This problem is NP-hard, and we describe several heuristic strategies, including local search and a genetic algorithm. Further, we present a simple and computationally efficient branch-and-cut algorithm. Our computational study indicates that our branch-and-cut algorithm finds optimal solutions for networks with up to 200 nodes within two hours of CPU time, while the heuristic search procedures rapidly find near-optimal solutions for all of the test instances.

Keywords Networks · Heuristics · Local search · Genetic algorithms · Branch-and-cut

In the prize-collecting generalized minimum spanning tree (PCGMST) problem, which arises in the design of regional telecommunications networks, a set of regions needs to be connected by a minimum cost tree structure and, for that purpose, one gateway site needs to be selected out of a set of candidate sites from each region. The competing sites in each region offer a monetary compensation, or a "prize," if selected as the gateway node for their region. The objective is to minimize the total cost of links used to connect the regions offset by the total sum of prizes collected from gateway sites selected for the design.

Examples of providing a monetary compensation for selection into a telecommunication network arise in many real-world contexts. For example, in the design of

B. Golden · S. Raghavan (✉) · D. Stanojević
The Robert H. Smith School of Business, University of Maryland, College Park,
MD 20742-1815, USA
e-mail: raghavan@umd.edu

undersea cable networks connecting different continents, not all countries or cities en-route can be directly connected to the undersea cable network. This is due to the very significant cost of connecting a location to a deep sea cable network. Consequently, planners of these undersea cable networks usually designate that one location will be selected from each of a specified set of regions that the network traverses. Given the potential monetary benefits associated with being a location that is directly connected to a transcontinental fiber-optic network with significant economic benefits it is not uncommon for cities or countries to vie against each other for selection as a location on this network. These monetary incentives are usually in the form of tax credits and rebates to the builder or operator of the telecommunications network.

In mathematical terms we are given an undirected graph $G = (V, E)$, with node set V and edge set E, with a cost vector $c \in R_+^{|E|}$ defined on the set of edges E, and a prize vector $p \in R_+^{|V|}$ defined on the set of nodes V. We are also given a partition of the node set $V_1, \ldots, V_K$ (i.e., $V_i \cap V_j = \emptyset$, if $i \neq j$ and $\bigcup_{k=1}^{K} V_k = V$). We need to find a minimum cost tree spanning exactly one node from each set in the partition, where the cost of the tree is defined as the total cost of the edges used for the tree minus the sum of the prizes corresponding to the nodes selected for the tree.

When all the prizes are equal to zero (or equivalently are exactly equal to each other within a node set in the partition), this problem corresponds to the generalized minimum spanning tree problem, that has been studied recently by several groups of researchers (see Feremans et al. 2004; Golden et al. 2005; Pop 2004). Since the GMST problem is NP-hard, it implies (by restriction) that the PCGMST problem is also NP-hard.

In this paper we present several polynomial-time heuristics for the PCGMST problem, and discuss two improvement strategies that significantly enhance the performance of these algorithms. We also adapt two heuristic search procedures—local search and a genetic algorithm—that we designed for the GMST problem (see Golden et al. 2005) to the PCGMST problem. These heuristic search procedures can be used to obtain high-quality solutions in large networks.

We also present a simple and computationally efficient branch-and-cut solution procedure for this problem. This procedure is very easy to implement and utilizes simple depth-first search at the integer incumbent nodes of the branch-and-bound tree to identify violated cuts. We compare its performance with two different variations of an exact procedure proposed by Pop (2004) for the GMST problem, and show that our algorithm provides better bounds for the problem. We show on a large set of instances that this procedure can be used to find optimal solutions in networks with random edge costs with up to 200 nodes and up to 40 clusters (for the rest of this paper we will refer to a node set in the partition as a cluster) within two hours of CPU time. On networks with Euclidean edge costs this procedure can be used to find optimal solutions in networks with up to 125 nodes within two hours of CPU time.

Our computational testing indicates that this branch-and-cut algorithm is sensitive to the relative values of edge costs and node prizes. When the node prizes have a smaller contribution to the objective function (compared to the contribution of edge costs), our branch-and-cut procedure finds optimal solutions in 130 out of 169 test instances within a two hour CPU time limit. On the other hand, when the node prizes have a higher contribution to the objective function, our branch-and-cut algorithm finds the optimal solutions in 166 out of 169 test instances within a two hour time limit. The performance of the heuristic search procedures is quite remarkable. Specifically, both local search and the genetic algorithm find the optimal solution in all of the 296 test instances where the optimal solution is known (from the branch-and-cut procedure)!

The rest of this paper is organized as follows. In Sect. 1 we discuss related work on the GMST problem. In Sect. 2, we discuss heuristic strategies for the PCGMST problem. We propose a lower bounding procedure and several polynomial-time heuristics for the PCGMST problem. We show that tailored repetitions of these heuristics provide a significant improvement in the quality of their solutions. In Sect. 2, we also adapt two heuristic procedures that we previously developed—local search and a genetic algorithm to the PCGMST problem. In Sect. 3, we review the mathematical formulation proposed by Pop (2004) for the GMST problem and discuss some important properties of this formulation that are relevant to our branch-and-cut algorithm. Section 4 discusses two exact solution procedures based on the mathematical formulation presented in Sect. 3. First, Sect. 4.1 explains the rooting procedure proposed in Pop (2004), and then compares the performance of two different variations of the rooting procedure. Next, in Sect. 4.2, we propose a new branch-and-cut algorithm for the PCGMST problem. We discuss specific choices that need to be made in this algorithm and computationally compare the performance of two different versions of this procedure. In Sect. 5, we compare our two heuristic search procedures with the branch-and-cut algorithm. Finally, in Sect. 6, we provide concluding remarks.

1 Literature review

To our knowledge, we are the first researchers to consider the prize collecting variant of the GMST problem. We will discuss some of the past work done on the GMST problem.

Several variants of the GMST problem have been studied in the literature. The version studied in this paper was introduced by Myung et al. (1995), who have shown that the GMST problem is NP-hard. Myung et al. (1995) have also developed a dual-ascent based branch-and-bound procedure that was used to solve problems in networks with up to 100 nodes and 4,500 edges.

Feremans (2001) and Feremans et al. (2002, 2004) presented several different formulations for the GMST problem and proposed a specialized branch-and-cut algorithm. In the computational study performed in Feremans (2001), this algorithm was used to find optimal solutions for the GMST problem in networks with random edge costs with up to 200 nodes. The same procedure provided optimal solutions for the GMST problem in the networks with edge costs satisfying the triangle inequality and with up to 160 nodes. Pop (2004) proposed a new mathematical formulation for the GMST problem and used it to develop a new exact procedure that can be viewed as a special form of delayed row and column generation. We will discuss this formulation and exact solution procedure in Sects. 3 and 4. Golden et al. (2005) proposed two fast, high-quality metaheuristic procedures for the GMST problem. Their computational study shows that these procedures provide optimal solutions for most of the problems in the large set of test instances.

Other versions of the GMST problem found in the literature are closely related to the more extensively studied Group Steiner Problem (GSP). In this sense, the first version of the GMST problem was introduced by Cockayne and Melzak (1968). The GSP studied by Cockayne and Melzak (1968) requires the design of a tree structure spanning *at least* one node from each cluster of nodes. Additionally, there may exist a set of Steiner nodes that does not belong to any of the clusters, but can be used for the tree design.

Recently, Duin and Voß (2004) have pointed out that when a specific GMST problem fits the framework of the GSP, one can use the transformation of the GSP to the well-studied undirected Steiner problem in graphs (SPG) to model the GMST problem. In their computational study, Duin and Voß (2004) found that two specialized SPG heuristics, Pilot-Rush and Pilot-Drop (originally defined in Duin and Voß (1999)), provide good results for this special case of the GMST problem. On a set of problems with Euclidean, random, and rectilinear distances in networks with 100 to 400 nodes, the Pilot-Drop procedure provided solutions that were on average less than 0.3% from optimality and with a maximum gap of 3.4% from optimality.

The *at least* version of the GMST problem (which is similar to the GSP except there are no Steiner nodes) is identical to the GMST problem, except that the tree must span at least one (instead of exactly one) node from each cluster. For these problems Duin and Voß showed that an exact SPG solver similar to the one developed by Duin (1993) significantly outperformed a genetic algorithm (GA) developed by Dror et al. (2000), both in terms of the solution quality and computational effort. On a set of 20 problems, the GA developed by Dror et al. (2000) provided solutions that were on average 6.53% from optimality, while the exact SPG procedure provided optimal solutions for all test instances within very short CPU times. Shyu et al. (2003) developed an ant colony approach that provides comparable results to the GA developed by Dror et al. It also takes less CPU time than the GA developed by Dror et al. Recently, Haouari et al. (2005) proposed an exact branch-and-bound algorithm for the *at least* version of the GMST problem. The proposed algorithm was combined with a specialized preprocessing algorithm, and was used to find optimal solutions for problems in networks with up to 250 nodes, 1000 edges, and 25 clusters within three hours of CPU time.

2 Heuristic procedures for the PCGMST problem

Given the fact that the PCGMST problem is a generalization of the polynomially solvable minimum spanning tree (MST) problem, it is natural to raise a question regarding the application of existing MST algorithms, as heuristics, to the PCGMST problem. In our previous work (Golden et al. 2005) we addressed this question for the GMST problem, and presented a spanning tree lower bound and three heuristic procedures based on three well-known MST algorithms—Kruskal's, Prim's, and Sollin's (see the text by Ahuja et al. 1993, for a nice description of the three algorithms, their implementations, and complexity).

The procedures for the GMST may be applied to the PCGMST, by ignoring the node prizes, and subtracting them ex-post from the solutions (i.e., by subtracting the node prizes for the nodes in the tree solution). However, if the prizes are considered ex-post, the three heuristic procedures provide solutions of extremely poor quality. In particular, on a set of TSPLIB instances (these instances are described in Sect. 5) for which the optimal solution is known, we found that the heuristic solutions were on average 40% from optimality. On the other hand when the node prizes are zero, these three heuristic procedures provided solutions that were on average 15% from optimality. This suggests that for the PCGMST problem, the node prizes should be accounted for *within* the heuristic procedures instead of ex-post.

In this section, we show how to account for node prizes, and describe a spanning tree lower bound procedure. We also describe how to adapt Kruskal's, Prim's, and Sollin's algorithm to the PCGMST problem while accounting for the node prizes. We then discuss a repetition strategy that significantly improves the performance of the proposed heuristics while maintaining the polynomial running time. Finally, we describe two fast and efficient metaheuristic procedures—local search and a genetic algorithm—for the PCGMST problem.

2.1 Spanning tree lower bound

A very simple lower bound for a given PCGMST problem can be obtained through the straightforward application of Kruskal's algorithm for the MST. This procedure solves the MST problem on a modified graph where each cluster is contracted into a single node (the prize of this node is set to the highest prize offered by nodes from the same cluster) and multiple edges between pairs of clusters are replaced by the one of minimum cost. To calculate this lower bound, it is not necessary to contract the graph. Instead, it is fairly easy to modify Kruskal's algorithm to accomplish the same. The running time of the lower bound algorithm is identical to Kruskal's and is $\mathcal{O}(|E| + |V| \log |V|)$. (Observe, that if we ignore the node prizes or node prizes are zero, we obtain the spanning tree lower bound for the GMST problem (Golden et al. 2005).)

2.2 Polynomial-time heuristics

We now describe how to adapt Kruskal's, Prim's, and Sollin's algorithm for the MST problem as heuristics to the PCGMST problem while taking into account the node prizes within the algorithm. We then consider a heuristic polynomial-time repetition framework called the *pilot method* that significantly improves the performance of these adaptations.

The adaptation of Kruskal's algorithm builds the PCGMST in a similar fashion to Kruskal's algorithm for the MST, with two exceptions. First, we need to make sure that exactly one node in each cluster is selected. And, second, we need to take into account prizes offered by the selected nodes.

To take into account the prizes we apply the following strategy in all three adaptations. Initially, we modify the cost of all edges by subtracting the prizes of the end points of an edge from its actual cost (note that the edge cost defined this way more precisely reflects the impact of addition of a particular edge on the objective function). In the subsequent steps of our algorithms, we update the edge costs whenever a new *node* is added to the tree that is being built. Once a node is included in the tree, we *add* the weight of this node to the cost of all edges that have this node as one of its end points. This modification of edge costs ensures that weights of the nodes already in the tree are not considered in the edge selection process (note that once the node is selected for the generalized spanning tree (GST), it is only the actual edge cost that makes a difference in the objective function).

In other words, in each iteration of Kruskal's adaptation (we call this adaptation PCKH) for the PCGMST problem, it finds the minimum cost edge (observe the edge costs can change in each iteration) such that its addition to the tree does not create a cycle among the clusters, and that at most one node from each cluster is selected for the final network design. This is also the bottleneck step in Kruskal's adaptation. This step takes $\mathcal{O}(|E|)$ time, and, since we add $K - 1$ edges, the running time of PCKH is $\mathcal{O}(|E|K)$.

The adaptation of Prim's algorithm for the PCGMST problem (referred to as PCPH) requires the selection of a starting node from which the tree is grown. Once the starting node is selected, we proceed in a straightforward manner identical to Prim's algorithm for the MST problem while taking care that exactly one node is selected from each cluster. That is, in each step we add the minimum cost edge from the nodes in the tree being grown to the nodes not in the tree, taking care that exactly one node is selected from each cluster. Additionally, we need to make sure that the node prizes are taken into account (as explained above). Observe that the solution provided by Prim's adaptation may depend on the node selected as the starting node (we refer to this node as the root node). In our implementation of this algorithm we select the root node randomly.

The running time of Prim's adaptation is identical to Prim's algorithm for the MST plus the time needed to make updates of edge costs. Since we will update an edge cost only once, this takes $\mathcal{O}(|E|)$ time. So, the running time of Prim's adaptation remains $\mathcal{O}(|E| + |V| \log |V|)$.

Sollin's algorithm for the MST starts with each node representing a tree. In each iteration of Sollin's algorithm, it identifies, for each tree in the partial solution, the minimum cost edge emanating from the tree. It then adds these edges to the partial solution (thus merging trees to build larger trees, and reducing the number of trees in the partial solution). The iterations of the algorithm are repeated until a spanning tree is obtained. We adapt Sollin's algorithm to the PCGMST as follows (we refer to this adaptation as PCSH). In each iteration of the algorithm, for each tree (or cluster, if no edge in the forest constructed so far is incident to any node in the cluster) select as a candidate edge the minimum cost edge out of the tree (or cluster) whose addition is feasible (i.e., adding the edge will not result in multiple nodes from a cluster in the partial solution). Ties between edges, for selection as candidate edge, are broken by choosing the edge that appears first in the sorted order. Consider the selected edges in sorted order and add an edge to the partial solution if its addition is feasible. (Note that, although the edges were feasible when selected, once we start adding edges to the partial solution, a selected edge may no longer be feasible for addition to the partial solution.) We repeat these iterations, updating edge costs as described earlier, until a feasible generalized spanning tree is obtained.

Each iteration of Sollin's adaptation takes $\mathcal{O}(|E| + K^2 \log K)$ time as we go through the list of $|E|$ edges to select the minimum cost feasible edge out of each tree (or cluster), sort them, and then consider at most $K - 1$ edges to add in an iteration. The time for making edge updates at the end of the iteration is $\mathcal{O}(|E|)$. In each iteration at least one edge is added, since it is always feasible to add the first selected edge. Thus there are at most $K - 1$ iterations. Consequently, the overall running time of Sollin's adaptation is $\mathcal{O}(|E|K + K^2 \log K)$.

Note that the proposed heuristics are guaranteed to return a feasible solution only when the underlying graph is complete (in terms of edges between all pairs of nodes). Ideally, we would like to add an edge in our adaptations only if the addition of the edge does not cause the heuristic to fail (i.e., make the problem infeasible). In order to do this, we need to answer the following question. Given a graph, clusters, and edges, does it contain a feasible GST? (When we add an edge, we have selected a node in a cluster. Thus, we can delete all other nodes in the cluster and edges emanating from them, and ask the question: does the modified graph contain a feasible GST?) Unfortunately, in general, from the transformation given by Myung et al. (1995), even the recognition of whether a graph contains a generalized spanning tree is NP-complete. Consequently, we handle infeasibility as follows.

In the case of Kruskal's adaptation, if the algorithm results in an infeasible solution, we propose that the algorithm remove the most expensive edge added to the tree from the problem and run the heuristic again. We repeat this procedure until either a feasible GST is obtained or one cluster has no edges out of it. In the case of Prim's and Sollin's adaptations, if the algorithms result in an infeasible solution, we delete the edge that was most recently added to the tree, and run the algorithm again.

When feasibility is an issue, a crude running-time bound for Kruskal's adaptation is $\mathcal{O}(|E|^2 K)$ (since we run Kruskal's at most $|E|$ times). However, this bound is somewhat misleading, because when the graph is dense (i.e., $|E|$ is large), infeasibility is very unlikely. Furthermore, observe that in Kruskal's algorithm the most expensive edge in the tree constructed is the last edge that was added to it. Thus, instead of building a tree from scratch, we may simply delete the last edge added and continue to build the tree from there. Similarly, arguing as for Kruskal's adaptation, a crude running-time bound for Prim's adaptation, when feasibility is an issue, is $\mathcal{O}(|E|^2 + |E||V| \log |V|)$. Again, for the very same reason as in the case of Kruskal's adaptation, this is somewhat misleading. Also, as in the case for Kruskal's adaptation, we can delete the most recently added edge and continue to rebuild a tree from there. Arguing similarly, when feasibility is an issue, a crude running-time bound for Sollin's adaptation is $\mathcal{O}(|E|^2 K + |E|K^2 \log K)$.

The quality of the solutions provided by the three adaptations PCKH, PCPH, and PCSH, are shown in Table 1. Table 1(a) shows results for the PCGMST problem with node weights set to zero (so, these problems correspond to the GMST problem), and Table 1(b) shows results for the PCGMST problem with the integer node weights randomly selected from the interval [0, 10]. Note, the number of instances in the two tables differs because they only include results for problems where the optimal solution is known. The results in Table 1(b) indicate that accounting for the prizes within the adaptations (instead of ex-post) significantly improves the quality of results (recall that at the start of this section we found that accounting for the prizes ex-post results in average gaps on the order of 40%). On the other hand, the average gap of

Table 1 Comparison of Kruskal's, Prim's, and Sollin's adaptation and the spanning tree lower bound for the PCGMST problem: (a) Zero node prizes; (b) Integer node prizes randomly selected in the range [0, 10]

Clustering type	Number instances	Spanning tree Lower bound	Upper bound procedures		
			PCKH	PCPH	PCSH
		Avg error	Avg error	Avg error	Avg error
(a)					
Center	37	37.13%	8.27%	13.10%	7.99%
Grid, $\mu = 3$	28	22.22%	5.05%	7.94%	5.22%
Grid, $\mu = 5$	28	32.95%	10.65%	12.41%	10.70%
Grid, $\mu = 7$	28	42.34%	15.87%	17.18%	15.64%
Grid, $\mu = 10$	29	44.39%	15.90%	18.56%	15.67%
Overall	150	35.94%	11.00%	13.83%	10.89%
(b)					
Center	33	74.18%	12.06%	25.42%	17.64%
Grid, $\mu = 3$	22	47.84%	7.95%	13.71%	10.89%
Grid, $\mu = 5$	22	68.68%	12.59%	22.16%	14.21%
Grid, $\mu = 7$	26	101.10%	21.33%	32.12%	23.28%
Grid, $\mu = 10$	27	74.39%	18.98%	23.76%	19.93%
Overall	130	74.22%	14.75%	23.88%	17.52%

the PCKH, PCPH, and PCSH is still quite high and probably too crude to be used for practical purposes. The same holds for our simple spanning tree lower bound for which we also report results in Table 1. This lower bound is very weak, deteriorates with non-zero node weights, and is probably not a good choice for obtaining lower bounds for the PCGMST problem.

The poor performance of the proposed upper bound heuristics is not surprising, given the fact that edge selection is used to direct the search. This kind of search can be inefficient for the PCGMST problem since when we select an edge, we automatically select ("fix") nodes that will be used for the two clusters incident to the edge selected. This motivates the question of whether a different type of search that would take into account node selection, would perform better.

We now describe an extensive search strategy that significantly improves our upper bound heuristics by progressively fixing *all* the nodes in the GST. First, let T be a set of nodes that are fixed for the initial GST design, such that $1 \leq |T| \leq K$. Next, let S_T^t represent a solution obtained by fixing *all* the nodes from the set T ($t = |T|$) for the GST design and then applying one of the three upper bound heuristics. (The steps for finding a solution S_T^t for a given set T are specific to each of our upper bound heuristics, and will be explained later in this section.) Also, let T^* indicate a set of nodes selected for the *final* GST design, and V^* indicate a set of all nodes that belong to the same clusters as the nodes in T^*. The search is started with empty sets T and T^*, and one of our upper bound heuristics is run $|V|$ times, each time starting with a different node selected for the design. The node i^* ($i \in V$), that provides the minimum cost solution is then fixed for the final GST (i.e., it is added to the set T^*). In order to add another node to the set T^*, we first set $T = T^*$ and then run the same upper bound heuristic $|V \setminus V^*|$ times, each time with a different node i ($i \in V \setminus V^*$)

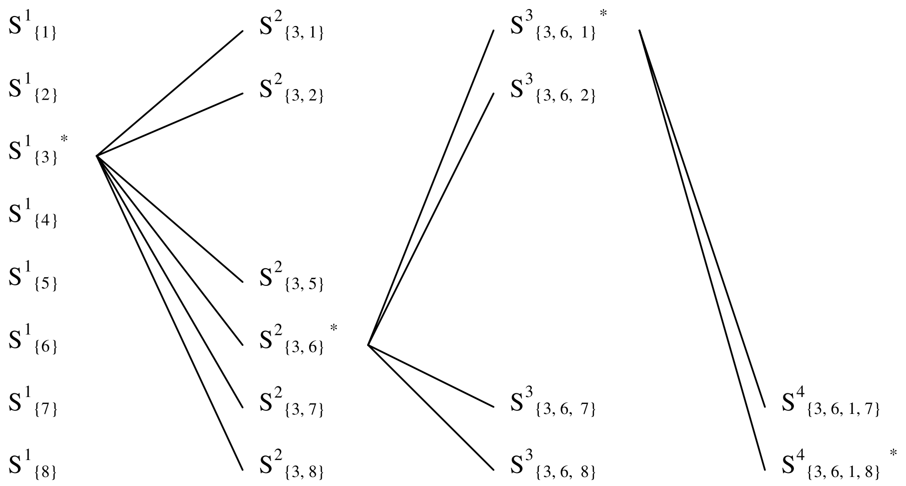

Fig. 1 Pilot method for the upper bound heuristics for the PCGMST problem

added to the set T (i.e., each time we set $T = T^* \cup \{i\}$, fix the nodes in the set T for the GST design, and apply the same upper bound heuristic). We continue this procedure until K nodes are added to the set T^*. This search paradigm is called the *pilot method* (see Duin and Voß 1999).

An example of the steps of our implementation of the pilot method for the proposed upper bound heuristics is illustrated in Fig. 1. In this example, we are given an 8-node network with 4 clusters, each containing 2 nodes. In the first iteration of the pilot method we run one of our upper bound heuristics $|V|$ times. The node that provides the best solution (node 3 in this example) is then added to set T^*, i.e., it is fixed for the final GST design. In the next iteration of the pilot method the same upper bound heuristic is run $|V \setminus V^*|$ times. As in the previous iteration, we select the node that provides the best solution (in this case node 6) and add it to the set T^*. The pilot method is continued until K nodes are fixed for the GST design.

The procedure used to obtain solutions S_T^t at any given iteration of the pilot method is as follows. In the case of Kruskal's and Sollin's adaptation, we simply fix the nodes from the set T for the GST design and apply either PCKH or PCSH respectively. In other words, we eliminate all nodes belonging to the same clusters as nodes that are fixed for the design and run our upper bound heuristics without any modifications. However, since each step of Prim's algorithm adds an edge to a partially completed tree, we need to implement a slightly different strategy. Specifically, we want to make sure that the new node added to the set T^* at the end of a given iteration of the pilot method is connected to the tree that is being built (in other words we want to make sure it is feasible to construct a tree on the set T^* in each step of the pilot method). To ensure this, we first connect the candidate node $i \in V \setminus V^*$ that is being considered to the tree on T^* using the least-cost edge available before proceeding with Prim's adaptation.

The results of the pilot method applied to each of our upper bound heuristics are shown in Table 2. We can see that solutions obtained using the pilot method provide

Table 2 Comparison of Pilot method for PCKH, PCPH, and PCSH. (a) Zero node prizes. (b) Integer node prizes randomly selected in the range [0, 10]

Clustering type	Number instances	PM-PCKH Average error	PM-PCPH Average error	PM-PCSH Average error
(a)				
Center	37	0.80%	1.53%	0.83%
Grid, $\mu = 3$	28	0.42%	0.83%	0.48%
Grid, $\mu = 5$	28	0.71%	1.30%	0.65%
Grid, $\mu = 7$	28	1.56%	1.42%	1.74%
Grid, $\mu = 10$	29	1.49%	1.42%	1.71%
Overall	150	0.99%	1.31%	1.07%
(b)				
Center	33	1.75%	1.18%	3.97%
Grid, $\mu = 3$	22	1.21%	0.62%	2.30%
Grid, $\mu = 5$	22	1.43%	1.85%	1.59%
Grid, $\mu = 7$	26	0.91%	1.63%	1.27%
Grid, $\mu = 10$	27	1.25%	1.99%	1.70%
Overall	130	1.33%	1.46%	2.27%

upper bounds that are less than 2% from optimality in the case of zero node weights, and less than 3% from optimality in the case of non-zero node weights. This is a significant improvement when compared to the quality of the upper bounds obtained by the initial adaptations (PCKH, PCPH, and PCSH) of the MST algorithms. Also, an interesting observation is that the pilot method for our upper bound heuristics runs in polynomial time (a very crude running time of the pilot method for each of our upper bound heuristics is $|V|K$ times their running times). The actual average CPU times of the PM-PCKH, PM-PCPH, and PM-PCSH procedures were 0.84, 0.34, and 66.29 seconds respectively in the case of zero node weights, and 101.22, 0.23, and 49.30 seconds respectively in the case of non-zero weights. (We note that the longer CPU times of the PM-PCKH and PM-PCSH procedures compared to the PM-PCPH procedure are due to our code. We believe that the running times for these procedures can be improved with more efficient codes.)

2.3 Metaheuristic procedures

We now describe two fast and efficient metaheuristic procedures for the PCGMST problem—local search and a genetic algorithm.

The local search (LS) procedure we developed in Golden et al. (2005) for the GMST problem may be directly applied to the PCGMST problem. It differs only in that an additional objective function term for node prizes needs to be taken into account here. Briefly, our local search procedure works as follows. It is an iterative 1-opt procedure. It visits clusters in a wraparound fashion following a randomly defined order. In each cluster visit, the neighborhood of a feasible generalized spanning tree is explored by examining all feasible trees obtained by replacing the node (in the tree) from the current cluster. In other words, a GST of least cost (the cost of

```
Begin
  $t \longleftarrow 0$
  initialize $P(t)$
  while (not $t <$ NUMGENS) do
    $t \longleftarrow t + 1$
    Generate $\alpha P(t-1)$ offspring using local search enhanced
      crossover operator
    Generate $\beta P(t-1)$ offspring using random mutation
      operator
    Select new generation $P(t)$, with population size equal
      to $\theta(1 + \alpha + \beta)P(t-1)$
  end
end
```

Fig. 2 Steps of our Genetic Algorithm for the PCGMST problem

the tree is the sum of the edge costs minus the rewards on the nodes) is found by trying to use every node from that cluster, while fixing nodes in other clusters. The local search procedure continues with visiting clusters until no further improvement is possible. The procedure is applied to a pre-specified number of starting solutions (denoted by t).

We now present a genetic algorithm (GA) for the PCGMST problem that is similar to the one we developed in Golden et al. (2005), with a few differences in the initial population and genetic operators applied. Figure 2 shows an outline of our genetic algorithm. The initial population is created by randomly generating a pre-specified number of feasible GSTs. Before adding a new chromosome to the population $P(0)$, we apply local search and add the resulting chromosome as a new population member. Within each generation t, new chromosomes are created from population $P(t-1)$ using two genetic operators: local search enhanced crossover and random mutation. The total number of offspring created using these operators is equal to the number of chromosomes in the population $P(t-1)$, with $\alpha P(t-1)$ offspring created using crossover, and $\beta P(t-1)$ offspring created using mutation (fractions α and β are experimentally determined). Once the pre-specified number of offspring is generated, a subset of chromosomes is selected to be carried over to the next generation. The algorithm terminates when the termination condition, a pre-specified number of generations that we denote NUMGENS, is met.

We now provide some more details of the genetic algorithm. A chromosome is represented by an array of size K, so that the gene values correspond to the nodes selected for the generalized spanning tree. The initial population is generated by random selection of nodes for each cluster. If possible, a minimum spanning tree is built over the selected nodes. Otherwise, the chromosome is discarded, since it represents an infeasible solution. Each feasible minimum spanning tree built in this way is then used as input for the local search procedure. The resulting solution is then added to the initial population as a new chromosome. We apply a standard one-point crossover operation (see Fig. 3 for an example). As in the initial population, only the feasible solutions are accepted. Each child chromosome created using this operator is used as input to the local search procedure, and the resulting chromosome is added to the population. A random mutation operator randomly selects a cluster to be modified and replaces its current node by another, randomly selected, node

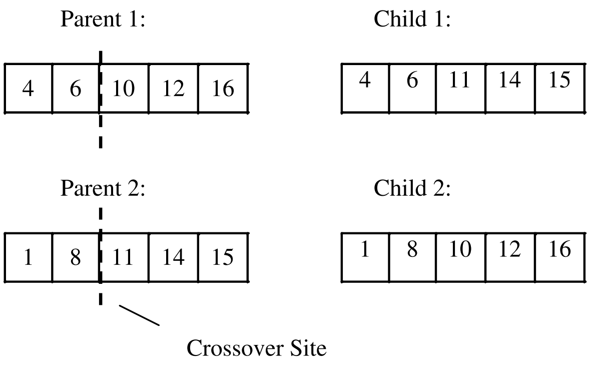

Fig. 3 An example of one-point crossover operator

from the same cluster. The new chromosome is accepted if it results in a feasible GST. In order to maintain diversity of the population, we do not apply local search to new chromosomes created by random mutation. At the end of each generation, a fraction, θ, of the current population is selected to be carried to the next generation, while the remaining chromosomes are discarded. This selection is a combination of elitism and rank-based selection, where the top 10% of the current population is selected using elitism and the remaining 90% is selected using rank-based selection (see Michalewicz 1996).

Our two metaheuristic procedures performed outstandingly. On a set of 296 test problems where the optimal solution is known both LS and GA found the optimal solution. We will elaborate on these results in Sect. 5 when we compare them with an exact solution procedure for the PCGMST problem.

3 A compact formulation

The exact procedures that we develop in this paper are based on a compact (i.e., polynomial size) mathematical formulation for the GMST problem originally proposed by Pop (2004). We will first introduce and motivate this formulation. We will then describe some important properties associated with this formulation that we later use to develop a simple and efficient branch-and-cut algorithm. While the formulation was introduced in the context of the GMST problem, we adapt and present it in the context of the PCGMST problem. Since the formulation has node variables to indicate whether a particular node is selected, this is easily done by modifying the objective function in the formulation.

To motivate the formulation for the PCGMST problem, it is best to first look at the well-known subtour elimination formulation for the minimum spanning tree (MST) problem from which this formulation is derived. This (*submst*) formulation can be stated as follows:

$$\text{Minimize} \quad \sum_{(i,j)\in E} c_{ij}u_{ij} \tag{1}$$

subject to:

$$\sum_{\forall(i,j)\in E} u_{ij} = n-1, \tag{2}$$

$$\sum_{\forall(i,j)\in S} u_{ij} \le |S|-1, \quad S \subset V,\ |S|>1, \tag{3}$$

$$u_{ij} \in B^1 \quad \forall (i,j)\in E. \tag{4}$$

The linear relaxation of this formulation describes the convex hull of the integer feasible region (see Magnanti and Wolsey 1995), but has an exponential number of subtour elimination constraints (3). It is well known that the separation problem for the identification of violated subtour elimination constraints (SECs) is a polynomially solvable min-cut problem that can be solved by either using one of the well-known combinatorial algorithms for the min-cut problem or by solving a linear program.

Martin (1991) shows that when the separation problem can be formulated as a polynomial-size linear program, we can use the dual of the separation problem to obtain an equivalent (in terms of the linear relaxation) compact (i.e., polynomial sized) formulation. In the case of the *submst* formulation, use of this approach leads to a new formulation (*ref-submst*) that replaces constraint (3) by the following three constraints:

$$w_{kij} + w_{kji} = u_{ij} \quad \forall (i,j)\in E,\ \forall k \in V, \tag{5}$$

$$\sum_j w_{kij} \le 1 \quad \forall k \in V,\ \forall i \in V \setminus k, \tag{6}$$

$$\sum_j w_{kkj} = 0 \quad \forall k \in V. \tag{7}$$

Martin (1991) formally establishes the equivalence of the *submst* and *ref-submst* formulations. Note, it is easy to see that the *ref-submst* formulation is valid. In terms of the MST problem, each set of w_{kij} variables for a given k represents a set of variables defining a directed tree rooted at node k, where an individual variable w_{kij} is equal to 1 if node j is predecessor of node i in the directed tree rooted at node k. Constraints (6) and (7), together with (2) and (5), guarantee that each node in the directed tree rooted at node k has at most one predecessor node, while the node k, by constraint (7), has no predecessors at all. (Note, constraint (6) may also be written with an $=$ sign, instead of a $\le$ sign.) Constraints (5) through (7) guarantee that there are no cycles, since if there was a cycle it would mean that for some root node k, and the corresponding set of variables w_{kij}, there is a node with either two predecessors, or a root node has a predecessor, which is not valid by the definition of these constraints.

We now discuss the properties of a special type of relaxation of the *ref-submst* formulation where only a subset of nodes W ($W \subset V$) is selected to be used as the root nodes of directed trees. In other words, instead of having variables w_{kij} defined for each $k \in V$, only the w_{kij} variables for $k \in W$ are included in the formulation. We will refer to this type of relaxation of the *ref-submst* formulation as a *p-root* relaxation, where p indicates the number of nodes used as roots in a given relaxation.

Lemma 1 *The linear relaxation of the* ref-submst formulation *with w_{kij} variables defined only for $k \in W \subset V$ $(1 \leq |W| \leq |V| - 1)$, provides an optimal solution for a given MST problem if the solution is integer feasible to the* ref-submst formulation.

Proof It is easy to see that Lemma 1 is true, as any relaxation of the *ref-submst* not including a set of constraints (5–7) is actually equivalent to the *submst* formulation without a set of subtour elimination constraints. And, for the latter relaxation, an integer solution without cycles always represents the optimal solution. □

Lemma 1 implies that an integer solution of the *p-root* relaxation of the *ref-submst* formulation always represents either a tree, or a structure with at least one cycle (note that if the solution is integer but does not define a tree structure, then, by constraints (3), the solution must contain at least one cycle). So, one way to solve the original problem is to start with a *p-root* relaxation, and then add violated constraints (5–7) (which means adding root nodes to the relaxation). Another approach to solve the original problem is to start with a *p-root* relaxation, and then use violated subtour elimination constraints (3) to eliminate any cycles that may be present in the solution of the *p-root* relaxation. While the violated SECs can be identified in polynomial time using a min-cut algorithm, identification of the violated constraints (5–7) may not be immediately obvious. However, we show that when the solution of the *p-root* relaxation is integral the separation problem for constraints (5–7) can be solved in a straightforward manner.

Proposition 2 *If there is a cycle in the integer solution of the p-root relaxation of the ref-submst formulation, then nodes $k^* \in W$ (where W is the set of root nodes) can never be a part of the cycle(s) present.*

Proof Observe that if a node $k^* \in W$ was a part of a cycle, it would imply that either one of the nodes in the cycle (other than node k^*) has 2 predecessors in the directed tree rooted at node k^*, or that node k^* has a predecessor in this tree. However, neither is possible by definition of constraints (5–7) for the set of variables w_{k^*ij}. □

Proposition 2 suggests that violated constraints (5–7) can always be identified as those corresponding to the directed trees rooted at nodes forming a cycle in the current integer solution of the *p-root* relaxation. We now show how these ideas can be applied to the PCGMST problem.

The *subel* formulation uses three types of variables to define the connections between clusters in the graph. The first group of variables, are *local edge variables* u_{ij} that define the use of specific edges between nodes in the graph. The other two groups of variables, y_{lr} and w_{klr}, on the other hand, define connections between clusters in the graph. The y_{lr} variables are *global edge variables* that indicate whether the solution includes an edge directly connecting clusters l and r. Observe that a GST forms a spanning tree on the graph obtained by shrinking each cluster to a single node. Thus, we would like the global edge variables to form a spanning tree on the clusters. Consequently, the w_{klr} variables play the same role as in the *ref-submst* formulation, but in relation to the global edge variables (i.e., they ensure the global edge variables do not form a cycle on the clusters). More precisely, each variable w_{klr} indicates whether cluster r is a predecessor of cluster l in the directed tree rooted at cluster k (there are K directed trees, one for each of the clusters in the graph). Finally, the formulation contains node variables z_i to indicate whether node i is in the solution.

Subtour elimination formulation (*subel*) for the PCGMST problem:

$$\text{Minimize} \quad \sum_{\forall (i,j)\in E} c_{ij}u_{ij} - \sum_{\forall i\in V} p_i z_i \tag{8}$$

subject to:

$$\sum_{i\in V_k} z_i = 1 \qquad \forall k = 1,\dots,K, \tag{9}$$

$$\sum_{i\in V_l, j\in V_r} u_{ij} = y_{lr} \qquad \forall l,r = 1,\dots,K,\ l\neq r, \tag{10}$$

$$\sum_{j\in V_r} u_{ij} \le z_i \qquad \forall r = 1,\dots,K,\ \forall i\in V\backslash V_r, \tag{11}$$

$$\sum_{(i,j)\in E} u_{ij} = K-1, \tag{12}$$

$$y_{ij} = w_{kij} + w_{kji} \qquad \forall k,i,j = 1,\dots,K,\ i\neq j, \tag{13}$$

$$\sum_{j} w_{kij} = 1 \qquad \forall k,\ i = 1,\dots,K,\ i\neq k, \tag{14}$$

$$w_{kkj} = 0 \qquad \forall k,j = 1,\dots,K, \tag{15}$$

$$w_{kij} \ge 0 \qquad \forall k,\ i,j = 1,\dots,K, \tag{16}$$

$$z_i \ge 0 \qquad \forall i\in V, \tag{17}$$

$$u_{ij} \ge 0 \qquad \forall (i,j)\in E, \tag{18}$$

$$y_{lr} \in B^1 \qquad \forall l,r = 1,\dots,\ K, l\neq r. \tag{19}$$

Formulation *subel* represents a straightforward extension of the *ref-submst* formulation, achieved through the introduction of local (i.e., u and z) and global (i.e., y and w) variables. Pop (2004) shows that the only variables that need to be defined as integer in the *subel* formulation are the global edge variables y_{lr}. Once the y_{lr} variables are integer, and form a spanning tree on the clusters, the other variables are automatically integer. Together, the variables define the GST in the following way. Constraint (9) ensures that exactly one node is selected from each cluster. Constraint (10) ensures that an edge connecting two nodes can be used only if the clusters corresponding to these nodes are connected by a global edge. Constraint (11) guarantees that node i belonging to cluster V_l can be connected to at most one node in any given cluster V_r $(l \neq r)$. Constraint (12) specifies the number of edges that are supposed to be selected for the GST. Constraint (13) ensures that clusters i and j can be adjacent in the directed tree rooted at cluster k only if clusters i and j are directly connected. Constraints (14) and (15) ensure that, with the exception of a root cluster k $(k \in K)$, every cluster in the directed tree rooted in the cluster k has exactly one predecessor cluster.

It is possible to define a *p-root* relaxation of the *subel* formulation using a similar relaxation strategy defined for the *ref-submst* formulation. In other words, the *p-root* relaxation of the *subel* formulation includes only a subset of the w_{kij} variables that define directed trees for the pre-specified set of the root clusters W $(1 \leq |W| \leq K - 1)$. (Note that the directed trees in the *p-root* relaxation for the PCGMST problem are defined over the clusters of the graph G, while the directed trees in the *p-root* relaxation for the MST problem are defined over the nodes of the graph G.) It is easy to establish that this relaxation has similar properties as the *p-root* relaxation for the MST problem.

Lemma 3 *The p-root relaxation of the* subel *formulation with the w_{kij} variables defined only for $k \in W \subset K$ $(1 \leq |W| \leq K - 1)$, provides an optimal solution to the PCGMST problem if the solution is integer feasible to the* subel *formulation.*

Proof First, observe that we can write the *subel* formulation using the subtour elimination constraints instead of the constraints (13–16). (We will refer to this new formulation as the *subel′* formulation for the PCGMST problem.) This directly implies that any relaxation of the *subel* formulation that does not include a set of constraints (13–16) is equivalent to the *subel′* formulation without a set of subtour elimination constraints. The rest of the proof is similar to the proof of Lemma 1. □

In an identical fashion to the MST problem, we can argue that if an integer solution of the *p-root* relaxation is infeasible to the *subel* formulation, then there must exist a cycle defined by the y_{lr} variables (we call these *global cycles* since y_{lr} represents global edge variables). Further, by definition of constraints (13–16), clusters that are used as roots of directed trees in the *p-root* relaxation of the *subel* formulation cannot belong to global cycles.

4 Exact procedures

In this section we discuss two exact procedures. First, we discuss a simple exact procedure called the rooting procedure (originally proposed by Pop (2004), for the GMST problem). We examine and test two variations of the rooting procedure to determine the best strategy while using the rooting procedure. Then, we develop a new (and simple) branch-and-cut algorithm for the PCGMST problem.

```
Step 0   Set $W = \emptyset$
Step 1   Randomly select cluster $r \notin W$
Step 2   Set $W = W \cup \{r\}$
         Add all global variables and constraints (13) - (15) for $k = r$, to p-root relaxation
         Solve p-root relaxation
Step 3   Check if the solution of the p-root relaxation is a GST
         If GST found
             Stop; optimal solution is found
         Else
             Go to Step 4
Step 4   Identify one cluster in the cycle from the solution of the p-root relaxation
         Set $r$ = identifed cluster from the previous step
         Go to Step 2.
```

Fig. 4 Steps of our implementation of the Rooting Procedure (RP$_{\text{cycle}}$)

4.1 Rooting procedure

This procedure is motivated by the fact that the *subel* formulation that includes all the w_{kij} variables and the corresponding constraints may be enormous and difficult to solve. The idea behind the rooting procedure is, therefore, to try to find the optimal solution by using only a small subset of clusters as root clusters.

The rooting procedure starts with the *p-root* relaxation of the *subel* formulation with a single randomly selected root cluster k_r. In the next step, the *p-root* relaxation that includes only variables and constraints defining a directed tree from root node k_r is solved. If the solution of this relaxation turns out to be a global tree, then, according to Lemma 3, the problem is solved; otherwise, variables and constraints for another root cluster are added (or used to replace existing root clusters) and the relaxation is resolved. This procedure is repeated until the optimal solution is found.

Pop (2004) does not elaborate on the strategy used to add or substitute root clusters in the rooting procedure. We suggest two simple ideas that come to mind when applying the rooting procedure. One is to randomly select clusters and add one root cluster at a time to the *p-root* relaxation. We refer to this variant of the rooting procedure as RP$_{\text{rand}}$. A more deliberate strategy for the addition of root clusters to the *p-root* relaxation that has a potential to provide better computational results is to examine the solution to a given *p-root* relaxation for the existence of global cycles (i.e., cycles defined on the global edge variables) and to only add a cluster (or clusters) that is a part of these global cycles. We refer to this procedure as RP$_{\text{cycle}}$. Our implementation of RP$_{\text{cycle}}$ is outlined in Fig. 4.

We computationally tested these two variants of the rooting procedure on a set of geographical TSPLIB instances. The results confirmed our expectations and showed a significant difference in the performance of the two versions of the rooting procedure. With a two hour CPU time limit, RP$_{\text{rand}}$ did not find the optimal solution in 11 out of 41 instances and, on average, required 2805.16 seconds of CPU time. Also, on average, RP$_{\text{rand}}$ required 7.34 root clusters to be added to the *p-root* relaxation in order to find the optimal solution. RP$_{\text{cycle}}$, on the other hand, did not find the optimal solution in 9 instances, and required, on average, 2126.34 seconds of CPU time. The average number of root clusters needed to be added to the *p-root* relaxation in this case was 4.09. Additionally, in instances solved to optimality, RP$_{\text{cycle}}$ never needed more than 8 root clusters to find the optimal solution, while RP_{rand} required 25 root clusters in the worst case. RP_{cycle} also provided better lower bounds than RP_{rand} in several instances. In particular, when compared over 11 instances that RP_{rand} did not solve to optimality, the lower bound provided by RP_{cycle} was on average 3.93% better than the one provided by RP_{rand}.

Pop et al. (2006) independently describe the variant that we call RP_{cycle}. Their computational results appear to be identical to Pop (2004) where no elaboration on the rooting procedure is provided (most of the results in Pop et al. (2006) appear to also be in Pop (2004)). In contrast, our experiments explicitly compare the performance of two different variants of the rooting procedure in the context of the PCGMST problem in order to get a better sense of the impact that the choice of the root clusters has on the efficiency of the rooting procedure for the PCGMST problem.

4.2 A simple branch-and-cut algorithm

In Sect. 3 we pointed out that an alternative approach for solving the *p-root* relaxation of the *subel* formulation is to progressively add violated subtour elimination constraints (SECs). In this section, we elaborate on this idea and present a new branch-and-cut algorithm for the PCGMST problem.

First, note that in a given solution of the *p-root* relaxation violated SECs can be defined over local edge variables or over global edge variables. Although it is not apparent which choice is better, it is clear that the number of SECs defined over global edge variables can be significantly lower than the number of SECs over local edge variables. Consequently, we use SECs defined over global edge variables (we call these global SECs) in our branch-and-cut algorithm. Another important decision that one needs to make is whether to add SECs at all nodes of the branch-and-bound tree or only at certain, designated, nodes. This issue is very important from an implementation perspective since the complexity of the separation problem may not be the same at different nodes of the branch-and-bound tree. In the *p-root* relaxation, the separation problem for the identification of violated SECs at any node of the branch-and-bound tree is a min-cut problem. At *incumbent (integer) nodes*, on the other hand, we can examine the set of edges selected in the solution and identify cycles using a simple depth-first search. These cycles immediately point to violated SECs. Consequently, once a cycle is identified, we can add a subtour elimination constraint defined over the clusters (i.e., a SEC defined over global edge variables) that are part of this cycle.

Besides the advantage of solving a simpler separation problem, the addition of cuts at incumbent (integer) nodes only may be more effective due to a fewer number of nodes at which the separation problem needs to be solved. We tested both ideas by defining two branch-and-cut algorithms. The first algorithm, BCA_{inc}, adds SECs only at incumbent (integer) nodes. The steps of this procedure are outlined in Fig. 5.

The second algorithm, BCA_{all}, solves the min-cut problem at every node of the branch-and-bound tree, and whenever violated global SECs are found, they are added to the entire branch-and-bound tree. In our implementation of the BCA_{all} we have used the min-cut algorithm of Stoer and Wagner (1997) implemented in LEDA 4.5 to solve the separation problem. The comparison of these two procedures is provided in Table 3. We can see that the proposed algorithm has comparable performance

```
Initialization
    Randomly select root cluster $r$
    Add all global variables and constraints (13) - (15) for $k = r$, to the p-root relaxation
    Start branch and bound algorithm
While optimal solution not found
    If current node in the branch-and-bound tree is integral
        If current solution is not a GST
            Add one violated subtour elimination constraint
        Else
            Continue optimization
    Else
        Continue Optimization
End While
```

Fig. 5 Steps of our branch-and-cut algorithm (BCA_{inc})

Table 3 Comparison of branch-and-cut algorithms BCA_{inc} and BCA_{all} for the PCGMST problem. (Note: The TSPLIB instances used for these tests were modified by adding randomly generated integer node weights in the range [0, 10])

Clustering type	Number of instances	BCA_{inc}	BCA_{all}	Avg relative difference BCA_{inc} vs BCA_{all}	
		Avg. time (sec)	Avg. time (sec)	UB	LB
Center	41	1606.23	1588.04	0.00%	−0.03%
Grid, $\mu = 3$	32	2362.58	2177.03	−0.07%	−0.31%
Grid, $\mu = 5$	32	2611.72	2293.01	−0.03%	−0.09%
Grid, $\mu = 7$	32	1534.83	1450.86	0.01%	−0.08%
Grid, $\mu = 10$	32	1189.16	1177.85	0.01%	−0.12%
Overall	169	1847.34	1729.41	−0.02%	−0.12%

with BCA_{inc} providing slightly better upper bounds, but also somewhat worse lower bounds. However, due to the extreme *simplicity* of the implementation of BCA_{inc}, we suggest the use of this algorithm.

The key advantage of BCA_{inc} is the fact that it does not require sophisticated routines and can be easily coded using commercial optimization software such as ILOG Concert Technology, and can even be easily implemented in a modeling language like AMPL or GAMS. Additionally, the improvement in the time needed to solve the PCGMST problem using these procedures compared to the time needed to solve some alternative mathematical formulations (without using branch-and-cut) is quite significant. For example, the mathematical formulation that we used for the GMST problem in Golden et al. (2005)[1] needed several days to solve several random instances (these instances are described in Sect. 5). Our BCA_{inc} procedure, on the other hand, solved most of these instances within two hours of CPU time.

[1]This formulation was solved using CPLEX 7.1 on a Sun Microsystems Enterprise 250 with 2×400 MHz processors and 2 GB RAM.

5 Additional computational results

All the procedures presented in this paper were coded using Microsoft Visual C++. The exact procedures were coded using ILOG Concert Technology 2.0 in CPLEX 9.0. All computations were performed on a workstation with 2.66 GHz Xeon processor and 2 GB RAM.

The test instances used in this paper include two sets of problems identical to those that we used in Golden et al. (2005), except that here we have used randomly generated integer node weights in the range [0, 10]. The first set represents the TSPLIB instances where the edge costs satisfy the triangle inequality. These instances were generated by Fischetti et al. (1997). This set contains five groups of problems that differ in the type of clustering in the network (the parameter μ in these instances can be interpreted as the average number of nodes in a cluster). The second set of test instances are random instances generated by Golden et al. (2005).

The two metaheuristic procedures, local search (LS) and genetic algorithm (GA), were applied with the following parameters that we determined using computational tests on a separate set of random instances. For the GA, we use a population size of 100 chromosomes, 15 generations, and we create 50% of offspring using crossover and mutation operators each. The fraction of the population discarded at the end of each generation is set to 0.5. The LS was performed using 500 independent randomly generated starting solutions. For the exact procedures, BCA_{inc} and RP_{cycle}, we have used a two hour time limit.

We first compared BCA_{inc} and RP_{cycle} over a set of geographical TSPLIB instances and obtained the following results. Out of 41 instances, BCA_{inc} did not find a solution in 8 instances, while RP_{cycle} did not find a solution in 9 instances. Overall, the average CPU time for the BCA_{inc} was 1606.23 seconds, while RP_{cycle} required 2126.34 seconds on average. In 9 instances where RP_{cycle} did not find the optimum, BCA_{inc} provided a better lower bound in 8 instances with an average improvement of 1.72%.

Due to the superiority of BCA_{inc} over RP_{cycle}, in our further computational tests, we have compared BCA_{inc} with the two metaheuristics—LS and GA. Table 4 summarizes results for 130 TSPLIB instances where BCA_{inc} found the optimum within two hours of CPU time, and Table 5 provides results for the remaining 39 unsolved

Table 4 Summary of computational results for BCA_{inc}, LS, and GA on TSPLIB instances where BCA_{inc} found the optimal solution within 2 hours of CPU times. Both LA and GA found the optimal solutions in all instances. Node weights are in the range [0, 10]

Clustering	Number of	BCA_{inc}		LS	GA
type	instances	Opt. sol.	Avg. time (sec)	Avg. time (sec)	Avg. time (sec)
Center	41	33	250.16	8.70	4.55
$\mu = 3$	32	22	163.72	38.84	10.74
$\mu = 5$	32	22	526.12	3.53	3.83
$\mu = 7$	32	26	227.47	2.51	3.34
$\mu = 10$	32	27	76.03	1.66	2.00
Overall	169	130	241.53	10.23	4.70

Table 5 Computational results for BCA_{inc}, LS, and GA on TSPLIB instances where the optimal solution is unknown. (The time limit for BCA_{inc} was 2 hours)

Problem	$\lvert E\rvert$	BCA_{inc}		LS		GA	
name		LB	UB	Soln	Time (sec)	Soln	Time (sec)
TSPLIB instances, center clustering							
28pr136	8879	30406.9	34003	34003	5.59	34003	5.70
30kroa150	10809	9330.25	9655	9655	7.33	9655	11.28
30krob150	10807	9568.43	9896	9896	6.78	9896	11.53
39rat195	18478	406.117	541	532	36.49	532	18.41
40kroa200	19409	10597.5	11442	11442	17.81	11442	18.28
40krob200	19430	10123.5	11080	11047	17.59	11047	18.72
40d198	18841	6662.5	6844	6844	17.19	6844	42.23
45ts225	24650	54554.8	62036	62080	196.02	62016	30.56
TSPLIB instances, grid clustering, $\mu = 3$							
50bier127	7729	70052.6	70965	70965	135.44	70965	14.86
60pr136	9064	49394	52569	52575	20.47	52562	22.13
57kroa150	11005	13318.8	13796	13796	206.31	13796	32.17
58u159	12321	23212.3	23916	23916	229.64	23916	23.47
81rat195	18745	635.438	646	649	67.33	646	61.05
72kroa200	19636	13613.6	14574	14533	246.41	14529	59.02
76krob200	19661	14028.1	14930	14930	328.14	14930	74.25
67d198	19101	7588.51	7956	7956	58.47	7956	44.02
75ts225	24900	73343.2	78639	78609	68.74	78609	136.94
84pr226	25118	58815.4	62156	62052	54.91	62052	74.66
TSPLIB instances, grid clustering, $\mu = 5$							
36kroa150	10868	9549.6	9929	9929	10.70	9929	31.05
36krob150	10870	9226.83	9580	9580	10.00	9580	29.55
33pr152	11083	37505.3	37985	37985	9.03	37985	16.19
32u159	12046	16498.4	16871	16871	8.03	16871	9.52
49rat195	18600	450.332	514	515	263.28	513	44.69
47kroa200	19464	11020.6	11383	11383	25.67	11383	23.22
48krob200	19511	10261.1	10866	10866	26.22	10866	24.27
40d198	18772	6527.09	6892	6892	15.88	6892	39.69
45ts225	24726	46446.4	60900	60496	210.20	60431	27.77
50pr226	24711	55338.1	56444	56444	265.52	56444	25.84
TSPLIB instances, grid clustering, $\mu = 7$							
25krob150	10725	6841.77	7174	7174	5.47	7174	5.00
36rat195	18454	349.141	409	408	18.27	408	15.25
35kroa200	19335	8016.5	9507	9476	14.61	9474	15.02
36krob200	19362	8802.5	9581	9580	14.97	9580	15.38
32d198	18372	5836.9	6329	6329	136.39	6329	17.83
35ts225	24544	42156.9	50753	50636	20.53	50635	15.73
TSPLIB instances, grid clustering, $\mu = 10$							
25rat195	18225	257.55	313	313	7.69	313	6.41
25kroa200	19100	6008.25	6754	6754	7.75	6754	22.11
25krob200	19082	6094.43	6801	6801	7.67	6801	21.06
25d198	18149	5600.93	6067	6053	6.66	6053	7.59
25ts225	24300	34771.1	40199	40187	8.64	40187	7.81

instances. Both metaheuristic procedures found optimal solutions in all 130 instances. The average CPU time required by LS in these instances was 10.23 seconds, while the average CPU time required by the GA was 4.7 seconds. Over all 169 TSPLIB instances, GA always provided solutions at least as good as the solutions provided by LS, and in eight instances GA was better than LS. In these instances, the upper bound provided by GA was, on average, 0.14% better than the upper bound provided by LS. The average time BCA_{inc} needed to solve 130 TSPLIB instances to optimality was 241.52 seconds. Of the remaining 39 instances where BCA_{inc} did not find the optimal solution, the upper bound provided by BCA_{inc} equals the best upper bound found by our metaheuristics in 24 instances. In 15 instances, the best upper bound found by our metaheuristics was, on average, 0.31% better than the one provided by BCA_{inc}. The average gap between the best upper bound and lower bound provided by BCA_{inc}, over the 39 instances that BCA_{inc} did not solve to optimality, was on average 8.94%.

In the case of random instances, our branch-and-cut algorithm found the optimal solution for 41 out of 42 instances within two hours of CPU time (see Table 6). In instances where the optimal solution is known, BCA_{inc} needed 32.93 seconds to find the optimal solution. Over the same set of instances, the average CPU time for LS was 6.2 seconds, and 5.98 seconds for GA. Both LS and GA did not find an optimal solution for one of the problems where the optimal solution is known.

In order to test the sensitivity of our procedures with respect to assigned node prizes, we have repeated our experiments over all 169 TSPLIB instances using a different node-prize function with 5 possible prize values (0, 1, 10, 100, 1000). These are summarized in Table 7. In these experiments, BCA_{inc} found the optimal solution in 166 out of 169 instances within two hours of CPU time, with an average CPU time of 113.95 seconds.[2] Both LS and GA found optimal solution in all 166 instances, with average CPU times of 14.37 and 9.56 seconds, respectively. In three instances where BCA_{inc} did not find the optimal solution, LS and GA provided the same solutions that were 12.65% from the best known lower bound, and 0.03% better than the upper bound provided by BCA_{inc}. These results indicate that higher node prizes (relative to edge costs) tend to make the problem easier to solve for BCA_{inc}. At the same time, the higher importance of the node prizes seems to make problems slightly more difficult to solve for the metaheuristics—LS and GA—in terms of computation time.

6 Conclusions

In this paper, we introduced the PCGMST problem, the prize collecting version of the GMST problem, that has potential applications in telecommunications network design.

We presented three polynomial-time upper bound heuristics for the PCGMST problem, and also adapted two heuristic search procedures—local search and a genetic algorithm—for the GMST problem to the PCGMST problem. We discussed a formulation and solution strategy called the rooting procedure introduced by Pop (2004) for the GMST problem. We experimented with two variants of the rooting

[2] In this case, we had to adjust the mip gap in CPLEX settings due to higher magnitude of cost functions.

Table 6 Computational results for BCA_{inc}, LS, and GA on random instances

Problem characteristics			BCA_{inc}		LS		GA	
K	$\lvert V\rvert$	$\lvert E\rvert$	Soln	Time (sec)	Soln	Time (sec)	Soln	Time (sec)
15	120	2,000	−54	7.63	−54	1.66	−54	1.66
		3,000	−69	2.77	−69	1.70	−69	2.00
		6,000	−83	3.16	−83	1.66	−83	1.59
	150	3,000	−82	0.33	−82	2.14	−82	1.45
		5,000	−80	0.33	−80	2.08	−80	1.91
		9,000	−98	19.24	−98	2.22	−98	1.94
	180	4,000	−81	8.30	−81	2.53	−81	2.30
		7,000	−96	6.60	−96	2.52	−96	2.39
		14,000	−94	11.36	−94	2.39	−94	2.31
20	120	1,500	−76	2.95	−76	2.70	−76	2.75
		3,000	−100	8.91	−100	2.86	−100	2.92
		6,000	−112	85.30	−112	2.66	−112	2.73
	160	3,000	−96	3.80	−96	4.09	−96	3.13
		5,000	−94	60.56	−94	3.78	−94	3.72
		10,000	−131	13.08	−131	3.52	−131	3.44
	200	5,000	−101	43.36	−101	4.94	−101	4.72
		10,000	−108	42.48	−108	4.58	−108	3.64
		15,000	−128	1.80	−128	4.50	−128	4.45
25	150	3,000	−89	88.73	−85	5.44	−89	5.67
		6,000	−111	19.47	−111	5.31	−111	4.14
		9,000	−130	3.81	−130	5.20	−130	4.78
	200	5,000	−111	12.00	−111	7.11	−111	6.99
		10,000	−134	60.33	−134	7.13	−134	6.59
		15,000	−148	12.39	−148	7.14	−147	6.56
30	120	2,000	−89	5.34	−89	5.78	−89	5.45
		3,000	−144	2.86	−144	6.17	−144	7.72
		6,000	−159	1.58	−159	6.30	−159	5.91
	150	3,000	−118	30.67	−118	7.22	−118	7.08
		6,000	−164	8.83	−164	7.14	−164	6.89
		9,000	−139	18.38	−139	7.09	−139	6.84
	180	4,000	−146	11.73	−146	9.30	−146	7.58
		7,000	−151	15.50	−151	9.17	−151	8.48
		14,000	−153	189.02	−153	8.38	−153	7.16
40	120	1,500	−112	2.58	−112	10.05	−112	9.33
		3,000	−157	22.00	−157	9.75	−157	9.45
		6,000	−180	12.83	−180	9.05	−180	9.58
	160	3,000	−142	9.25	−142	13.86	−142	12.77
		5,000	−170	20.14	−170	13.36	−170	12.70
		10,000	−205	6.97	−205	12.49	−205	11.36
	200	5,000	−177	150.20	−177	18.19	−177	17.00
		10,000	−195*	7,200.00	−194	16.94	−194	17.36
		15,000	−223	323.56	−223	15.33	−223	15.91

*BCA_{inc} did not complete search for this problem within 2 hours of CPU time. Upper bound at the time search was terminated was −194

Table 7 Summary of computational results of BCA$_{\text{inc}}$, LS, and GA on TSPLIB instances with node weights $[0, 1, 10, 100, 1000]$. Both LA and GA found the optimal solutions in the 166 instances that BCA$_{\text{inc}}$ solved to optimality within 2 hours of CPU time

Clustering	Number of	BCA$_{\text{inc}}$		LS	GA
type	instances	Opt. sol.	Avg. time (sec)	Avg. time (sec)	Avg. time (sec)
Center	41	41	55.89	8.54	7.17
$\mu = 3$	32	31	348.13	52.45	23.14
$\mu = 5$	32	31	56.82	7.46	10.18
$\mu = 7$	32	31	34.66	3.46	4.80
$\mu = 10$	32	32	93.63	2.24	3.50
Overall	169	166	113.95	14.37	9.56

procedure for the PCGMST problem. We then developed a very simple and effective exact solution procedure.

Our computational experiments show that our upper bound heuristics, which represent the straightforward adaptations of the MST heuristics, do not provide good bounds for the PCGMST problem. However, when applied within the framework of a tailored, heuristic-repetition paradigm called the pilot method we show that these procedures provide substantially better results (finding solutions that are on average less than 3% from optimality).

The exact procedure that we proposed in this paper is a novel branch-and-cut algorithm that works with incumbent (integer) nodes of the branch-and-bound tree. Results of our computational study indicate that this algorithm provides better bounds for the PCGMST problem than the two variants of the rooting procedure that we tested in this paper. Our computational experiments further suggest that our branch-and-cut algorithm is a good choice (i.e., can provide optimal solutions within relatively short CPU times) for networks with up to 200 nodes. For larger networks, the heuristic search procedures are a better alternative. These procedures (LS and GA) have demonstrated excellent performance for two different types of functions used for the node prizes. In particular, out of 296 TSPLIB instances for which the optimum is known, both LS and GA, very rapidly, found the optimum solution in all 296 instances. For the random instances, both algorithms found optimal solutions for 40 out of 41 instances for which the optimal solution is known.

References

Ahuja, R.K., Magnanti, T.L., Orlin, J.B.: Network Flows: Theory, Algorithms and Applications. Prentice-Hall, New Jersey (1993)

Cockayne, E.J., Melzak, Z.A.: Steiner's problem for set terminals. Q. Appl. Math. **26**(2), 213–218 (1968)

Dror, M., Haouari, M., Chaouachi, J.: Generalized spanning trees. Eur. J. Oper. Res. **120**(3), 583–592 (2000)

Duin, C.: Steiner problem in graphs: approximation, reduction, variation. Ph.D. thesis, University of Amsterdam (1993)

Duin, C., Voß, S.: The pilot method: a strategy for heuristic repetition with application to the Steiner problem in graphs. Networks **34**(3), 181–191 (1999)

Duin, C., Voß, S.: Solving group Steiner problems as Steiner problems. Eur. J. Oper. Res. **154**(1), 323–329 (2004)

Feremans, C.: Generalized spanning trees and extensions. Ph.D. thesis, Université Libre de Bruxelles (2001)

Feremans, C., Labbé, M., Laporte, G.: A comparative analysis of several formulations for the generalized minimum spanning tree problem. Networks **39**(1), 29–34 (2002)

Feremans, C., Labbé, M., Laporte, G.: The generalized minimum spanning tree problem: polyhedral analysis and branch-and-cut algorithm. Networks **43**(2), 71–86 (2004)

Fischetti, M., Salazar-Gonzalez, J.J., Toth, P.: Symmetric generalized traveling salesman problem. Oper. Res. **45**(3), 378–394 (1997)

Golden, B., Raghavan, S., Stanojević, D.: Heuristic search for the generalized minimum spanning tree problem. INFORMS J. Comput. **17**(3), 290–304 (2005)

Haouari, M., Chaouachi, J., Dror, M.: Solving the generalized minimum spanning tree problem by a branch-and-bound algorithm. J. Oper. Res. Soc. **56**(4), 382–389 (2005)

Magnanti, T.L., Wolsey, L.A.: Optimal trees. In: Ball, M., Magnanti, T.L., Monma, C.L., Nemhauser, G.L. (eds.) Network Models. Handbooks in Operations Research and Management Science, vol. 7, pp. 503–615. North-Holland, Amsterdam (1995)

Martin, R.: Using separation algorithms to generate mixed integer model reformulations. Oper. Res. Lett. **10**(3), 119–128 (1991)

Michalewicz, Z.: Genetic Algorithms + Data Structures = Evolution Programs. Springer, Heidelberg (1996)

Myung, Y.S., Lee, C.H., Tcha, D.W.: On the generalized minimum spanning tree problem. Networks **26**(4), 231–241 (1995)

Pop, P.C.: New models of the generalized minimum spanning tree problem. J. Math. Model. Algorithms **3**(2), 153–166 (2004)

Pop, P.C., Kern, W., Still, G.: A new relaxation method for the generalized minimum spanning tree problem. Eur. J. Oper. Res. **170**(3), 900–908 (2006)

Shyu, S.J., Yin, P.Y., Lin, B.M.T., Haouari, M.: Ant-tree: an ant colony optimization approach to the generalized minimum spanning tree problem. J. Exp. Theor. Artif. Intell. **15**(1), 103–112 (2003)

Stoer, M., Wagner, F.: A simple min-cut algorithm. J. ACM **44**(4), 585–591 (1997)