Lecture Notes in Management Science (2014) Vol. **6**: 207–216
6th International Conference on Applied Operational Research, Proceedings

ISSN 2008-0050 (Print), ISSN 1927-0097 (Online)

A fast heuristic for the prize-collecting Steiner tree problem

Murodzhon Akhmedov, Ivo Kwee and Roberto Montemanni

Dalle Molle Institute for Artificial Intelligence IDSIA-USI/SUPSI
Galleria 2, CH-6928 Manno, Switzerland
{murodzhon, roberto}@idsia.ch; ivo.kwee@ior.iosi.ch

Abstract. The Prize-Collecting Steiner Tree Problem (PCSTP) is a generalized version of the Steiner Tree Problem. PCSTP is well known and well studied problem in Combinatorial Optimization. Since PCSTP is NP-hard, it is computationally costly to achieve solutions for large instances. However, many real life network problems come with a wide range of variables and large instance sizes. Therefore, there is a need for efficient and fast heuristic algorithms to discover the hidden knowledge behind vast networks. There exists a fast heuristic algorithm for the Steiner Tree Problem in the literature, which is based on Minimum Spanning Trees. In this paper, we propose to extend the existing heuristic algorithm to solve PCSTP. The performance of the extended heuristic (MST-PCST) is evaluated on available benchmark instances from the literature. We also test MST-PCST on randomly generated huge graph instances with up to 40000 nodes and 120000 edges. We report the average gap percentage between the solutions of MST-PCST and existing solution approaches in the literature. Results show that overall performance of MST-PCST is promising with tolerable gap percentage and reasonable running time on larger instances. It has a significantly faster running time when graphs scale up which can shed light on large real world network instances.

Keywords: prize-collecting steiner tree problem; minimum spanning tree; combinatorial optimization

Introduction

The Steiner Tree Problem (STP) is a popular and well-studied problem in Graph Theory as well as in Combinatorial Optimization. So-called Prize-Collecting Steiner Tree Problem (PCSTP) is a generalized version of STP. Given an undirected network $G = (V, E)$ where nodes and arcs of network are respectively associated with node prizes $p_j \geq 0$ and arc costs $c_e > 0$. For simplicity, the set of nodes with $p_j > 0$ are called *terminal nodes* and the rest of the nodes as *Steiner nodes* (Tuncbag *et al*, 2012). PCSTP finds a sub-graph $G' = (V', E')$ of G where it minimizes the total arc costs in the sub-graph and prizes of

nodes that are not in the sub-graph. This problem corresponds to minimization of the following equation and it is easy to see that every optimal sub-graph will have a tree structure.

$$GW(G') = \sum_{e \in E'} c_e + \sum_{v \notin V'} p_v \qquad (1)$$

This is also known as the *Geomans − Williamson Minimization* problem (Johnson *et al*, 2000) in the literature. PCST has a variety of applications in utility networks such as making fiber optic, gas or district heating connections for households economically feasible. Recently a connection between PCSTP and biological networks has been identified (Becheta *et al*, 2010). Briefly, in a large gene-gene or protein-protein interaction, network PCSTP attempts to find a neighborhood or sub-network where genetic aberrations are most concerned. For example using gene expression data, one can find a connected neighborhood where many genes are differently expressed.

Since PCSTP is NP-hard it is time consuming or in some cases it is impossible to obtain a solution for large instances in a reasonable time. However, real life problems from biology, for instance, come with a huge number of variables and instance sizes. We aim to perform functional analyses of genes by using gene expression profile networks and gene interaction networks, and usually the size of these networks can be very large with up to 20,000 nodes and more than 100,000 arcs. In order to attain our goal, these giant networks should be analyzed in a reasonable time. The existing exact solution approaches in the literature suffer from solving the problem when the input graph scales up. Thus, we need efficient and fast heuristic algorithms to interpret the hidden knowledge behind the giant biological networks. There exists a fast heuristic algorithm for STP in the literature, which is based on Minimum Spanning Tree (Kou *et al*, 1981). For STP, it is proven that the heuristic has an approximation ratio of $2(1-1/l)$ where l is the number of leaf nodes in the optimal tree. In this paper, we propose to extend the existing heuristic algorithm to solve PCSTP and enrich the heuristic with an effective leaf node pruning strategy (MST-PCST). However, it is not guaranteed yet that the heuristic still preserves the approximation ratio or not. The remainder of the paper is organized as follows: Related work in the literature is discussed in next section. Then, the extended MST-PCST heuristic approach is described in section 3. The fourth section consists of computational results followed by conclusions and future work.

Related Work

There have been developed exact solution approaches as well as heuristics to solve PCSTP in the literature. The PCSTP was introduced by Bienstock *et al* (1993). Segev (1987) proposed the node weighted Steiner tree problem where a predetermined set of nodes should be included into the final tree solution. Exact methods were developed in Ljubić *et al* (2005) and (2006) where PCSTP was formulated as a mixed integer linear programming, and a branch-and-cut algorithm was proposed as a solution methodology. The quota version of PCSTP was studied by Haouari *et al* (2010) in which the budget or quota limit was considered within the constraint set. Canuto *et al* (2001) proposed a heuristic

local search with perturbations. Johnson *et al* (2000) investigated a different variant of PCSTP and proposed a strong pruning rule for primal-dual 2-approximation algorithm. Klau *et al* (2004) studied the combination of a memetic algorithm with integer programming. Recently a robust optimization approach was devised for PCSTP by Miranda *et al* (2013). Mixed integer linear programming models for similar problems, the minimum power broadcasting and multicasting problems, have been proposed by Barta *et al* (2010), and Montemanni *et al* (2008, 2011), and Montemanni and Leggieri (2010).

Methodology

In this section we demonstrate our MST-PCST heuristic algorithm to solve PSCTP for the Goemans-Williamson minimization function. Given an undirected network $G = (V, E)$, the set of terminal nodes can be identified, which consists of nodes with $p_j \geq 0$. One is asked to identify a sub-graph of G according to the minimization function in equation (1). By analyzing the objective function it is easy to notice that not all terminal nodes necessarily should be included into the output tree structure.

Algorithm 1 : MST-PCST Algorithm

Require: undirected graph G = (V, E)
  Initialize: $S \leftarrow$ all terminal nodes in G, $C \leftarrow \infty$,
    $C' \leftarrow \sum_{e \in E} c_e$;
  Phase I
  while $C' \leq C$ **do**
    $C \leftarrow C'$;
    Construct a complete graph $G' = (V', E')$ where $V' = S$ and each arc in E' corresponds to the shortest path between nodes in G;
    Solve MST on G' and obtain tree T, $C' \leftarrow \sum_{e \in T} c_e$;
    Convert T into original graph G and obtain a subgraph T', $S \leftarrow$ all nodes in T';
  end while

  Phase II
  while all leaf nodes v of T' not pruned **do**
    if $p_v <$ connection cost **then**
      eliminate v from T';
    end if
  end while

PCSTP is a generalized version of the Minimum Spanning Tree (MST) problem. In contrast to PCSTP, all nodes are considered as terminal nodes in MST. There exist fast and efficient algorithms to solve MST. The pseudocode of a proposed MST-PCST heuristic is presented in Algorithm 1. MST-PCST targets to obtain a solution for PCSTP with large network instances in a reasonable time. The intuition behind the algorithm is likening the PCSTP to the MST problem and employing existent efficient algorithms to solve huge instances. Simply, MST-PCST reduces the vast network to a smaller network

with particular nodes and arc costs, solves MST on a smaller artificial network, and projects the solution obtained in the smaller network back to the large network. The heuristic consists of two phases.

Each step of the algorithm is illustrated below along with its explanation. After the initialization of variables, we focus on the first phase of the heuristic to analyze. As a first iteration within the 'while' loop, the algorithm constructs a complete graph $G' = (V', E')$ from G where V' consist of all terminal nodes. Please see Figure 1 for further clarification. Each arc length between terminal nodes in E' corresponds to a shortest path length between those terminal nodes in the original network G (Fig. 1 B). The positions of the terminal nodes are reorganized in the figure for clarity. The shortest path problem is solved by Dijkstra's algorithm (Dijkstra, 1959). All nodes on the shortest path between terminal nodes in graph G are recorded for sub-graph construction in later stages. Afterward the heuristic solves MST on G' by employing Prim's algorithm (Prim, 1957) (Fig. 1 C). The total cost of arcs in the resulting MST tree is recorded as C. Then the tree solution in G' is converted into the original graph G (Fig. 1 D). It is possible that some arcs can appear more than once in G after this conversion. In this case they are reduced to a single arc. It is guaranteed that we again obtain a tree in G after this conversion since each arc in G' corresponds to a path.

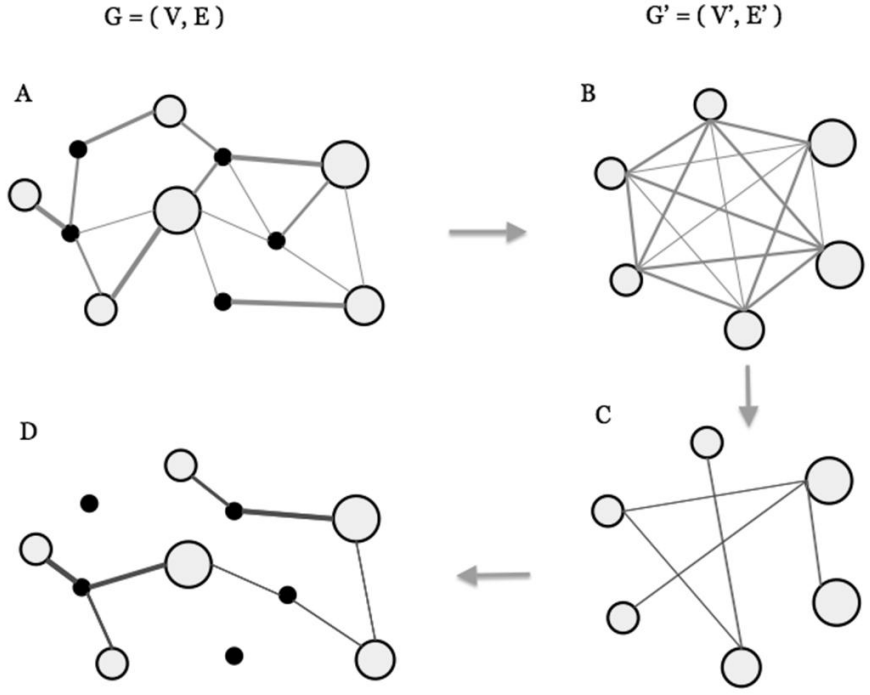

Fig. 1. MST-PCST Algorithm: Iteration 1. Yellow nodes correspond to terminal nodes, black nodes represent Steiner nodes, and arc widths represent the arc costs

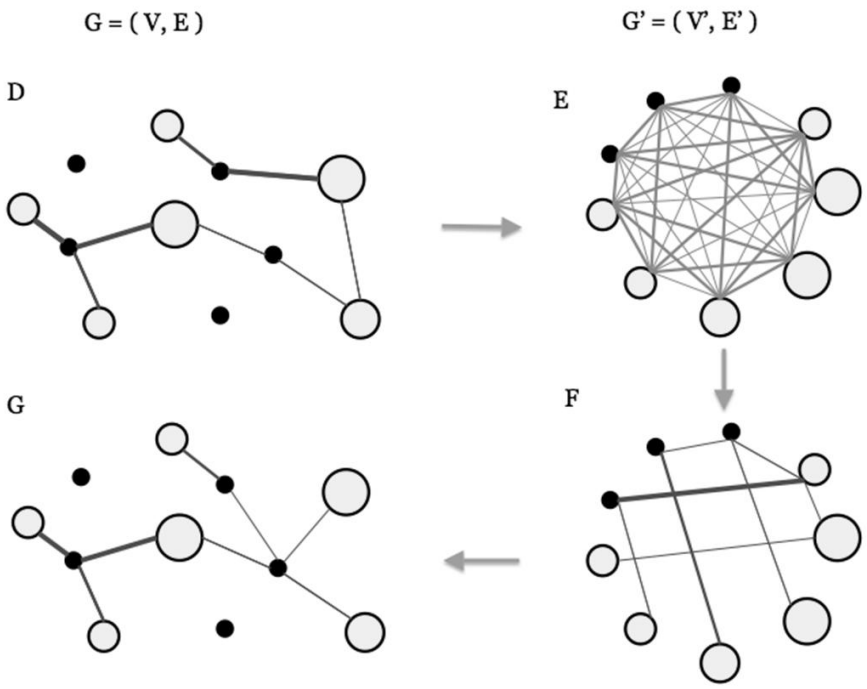

Fig. 2. MST-PCST Algorithm: Iteration 2

The second iteration of the 'while' loop is demonstrated in Figure 2. Now there already exists a sub-graph consisting of all terminal nodes and possibly some Steiner nodes in *G*, which is obtained in the previous iteration (Fig. 2 D). In this iteration, the algorithm again constructs a complete graph *G'* in the same way as explained before where *G'* = (*V'*, *E'*) and *V'* includes all nodes of the sub-graph in *G* (Fig. 2 E). Then MST is solved on *G'* and the total cost of arcs in the resulting MST tree is recorded as *C'* (Fig. 2 F). If *C'* < *C*, then the resulting tree is converted into the original network (Fig. 2 G). This process, generating *G'* from *G*, solving MST on *G'* and converting the resulting tree back into *G* is continued until the algorithm converges, meaning it is not possible to obtain a tree with cost of *C'* < *C*. No other Steiner node could be added to *G'* anymore and the decrease in *C'* is monotonic, because at each iteration the algorithm is looking for a new spanning tree with lesser cost that consists of nodes in a recent tree and probably some other nodes. The logic behind this process is generating a tree with as low as possible total arc cost while keeping the total node prize high by maintaining all terminal nodes in a tree. Then, this phase is terminated and the algorithm continues with the pruning phase.

In the second phase of the heuristic each leaf node of sub-graph in *G* is tested for pruning. If the prize of the leaf node is smaller than the connection cost, that node is simply eliminated from the sub-graph. The connection cost could be a single arc cost or cost of the path in the case where a terminal node is connected to a sub-graph via several Steiner nodes. The final solution of the MST-PCST algorithm is presented in Figure 3 that is a pruned version of the graph in Fig. 2 *G*.

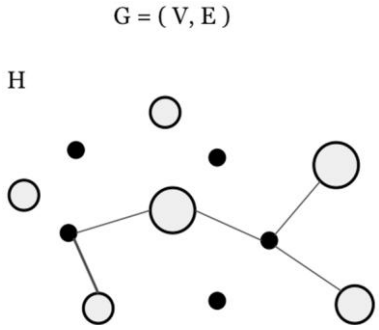

Fig. 3. Final sub-graph: A Tree

Computational Results

This section consists of computational studies on two sets of problem instances. The performance of MST-PCST is evaluated on benchmark instances in the literature and randomly generated large instances. We compare the running time of the MST-PCST heuristic with the running times of existing approaches in the literature. Detailed information about instances and running time data of approaches are reported later in this section.

The first problem set is the benchmark instances that are available in the literature. The *C* and *D* instance series from Canuto *et al* (2001), and Cologne1 and Cologne2 real world instances from Ljubić *et al* (2005) are used for computational studies. The C and *D* instances are generated from the Steiner problem of the OR-Library (Beasley, 1990) and detailed descriptions of generation can be found in Canuto *et al* (2001). The Cologne1 and Cologne2 real world instances have been used for designing the fiber optic networks of a German city. These instances are generated based on real network infrastructure and detailed descriptions of generation can be found in Ljubić *et al* (2005). The *C* set contains 40 instances with 500 nodes and 625-12500 edges, and *D* set contains 40 graphs with 1000 nodes and 1000-25000 edges. Cologne1 and Cologne2 consist of total 35 instances with 768-1819 nodes and 69077-213973 edges. Canuto *et al* (2001) is a multi-start local search algorithm for PCSTP where the initial solution is generated by a primal-dual algorithm and a variable neighborhood search is utilized in a post-optimization procedure. Ljubić *et al* (2005) is an exact solution approach for PCSTP where the problem was formulated as a MILP and the branch-and-cut algorithm construction was proposed to solve the MILP. In their implementation, they associated the connectivity inequalities with the cuts in a directed graph, and found the violating inequalities by using a maximum flow algorithm.

The second set of problem instances consists of 30 randomly generated huge graphs with n nodes (ranging from 5625 to 40000) and m edges (ranging from 16875 to 120000). Since our goal is devising the heuristic algorithms to analyze the giant real world instances, we would like to see the performance of MST-PCS when the graph scales up. The generated graphs that consist of n nodes have m edges with corresponding values of $3n$ and $6n$, respectively. The instances are generated in the following manner.

Each integer point in a Cartesian plain in the range of [1, k] in the *x-axis* and in the range of [1, k] in the *y-axis* represents a single node where k is an integer value. Basically, k^2 gives the number of nodes n in a graph. At the beginning, we build a random spanning tree that contains all nodes to maintain the connectivity of the graph. Then we randomly select two nodes and add an edge, if there is no edge between them. The edge between node i and j with the corresponding coordinates of (a_1, b_1) and (a_2, b_2), respectively, is added to the graph with the cost of c_{ij} and cost is determined as follows:

$$c_{ij} = \sqrt{(a_1 - a_2)^2 + (b_1 - b_2)^2}$$

Associating the prize with the terminal nodes is nontrivial. Especially, prizing the node is the issue in real world network examples from the biology. The prize of terminal nodes should be proportional to the cost of edges in the graph in order to obtain a meaningful output tree. The maximum possible edge cost in the generated graph could be $C_{max} = \sqrt{(k-1)^2 + (k-1)^2}$. In our implementation, the terminal nodes are given prizes from the discrete uniform distribution in $U[\theta_1 * C_{max}, \theta_2 * C_{max}]$. We used three different (θ_1, θ_2) couples in graph generation and those are: (0.1, 0.5), (0.6, 0.8) and (0.8, 1.0). The terminal nodes are randomly selected among the nodes and each randomly generated graph is analyzed with 150 terminal nodes. The new instances are available upon request to the authors.

Since the computational studies of approaches are performed on different machines, we use the scaling factor to bring the running time data of approaches into fairly equivalent base in order to compare the performance. For comparing MST-PCST running time with the results of Ljubić *et al* (2005) (obtained on a Pentium IV with 2.8 GHz, 2 GB RAM, SPECint2000 = 1204) and Canuto *et al* (2001) (achieved by Pentium II with 400 MHz, 64 MB RAM) the scaling factor is obtained from Dongarra (2013). The computational studies for MST-PCST were performed on 2.9 GHz MacBook with 8 GB of memory. However, Dongarra (2013) does not provide a direct scaling factor for these three machines. We can establish a conservative estimate by taking the performance of similar machines into account (Intel Pentium IV 2.8 GHz with 1317 Mflops/s) and (Intel Pentium II 333 MHz with 69 Mflops/s). Dividing the running time of Ljubić *et al* (2005) by factor of 12 and Canuto *et al* (2001) by factor of 50 gives a reasonable basis of comparison to our running time data. The MST-PCST algorithm is implemented in C++ environment and Boost Graph Library (Siek *et al*, 2000) is employed.

Figure 4 demonstrates the scatter plot of the number of nodes in a graph versus the running time of the various approaches. Ljubić *et al* (2005) is an exact solution methodology that is formulated as a MILP and provides the optimal solutions. For this approach, we do not have figures for when the best solutions (not proven to optimality) are retrieved. So, we report the time needed to obtain optimal solutions in the figure. Canuto *et al* (2001) is a multi-start local search algorithm for PCSTP. From the figure, it is clear that the relation between the instance size and the running time of Ljubić *et al* (2005) and Canuto *et al* (2001) is exponential. Since the Ljubić *et al* (2005) solution was formulated based on a MILP, the number of variables become huge when the graph scales up and it is computationally costly to use this approach in large instances. However, the running time of MST-PCST is more preferable and this supports that the extended heuristic could be useful to interpret the instances with a large number of nodes and edges.

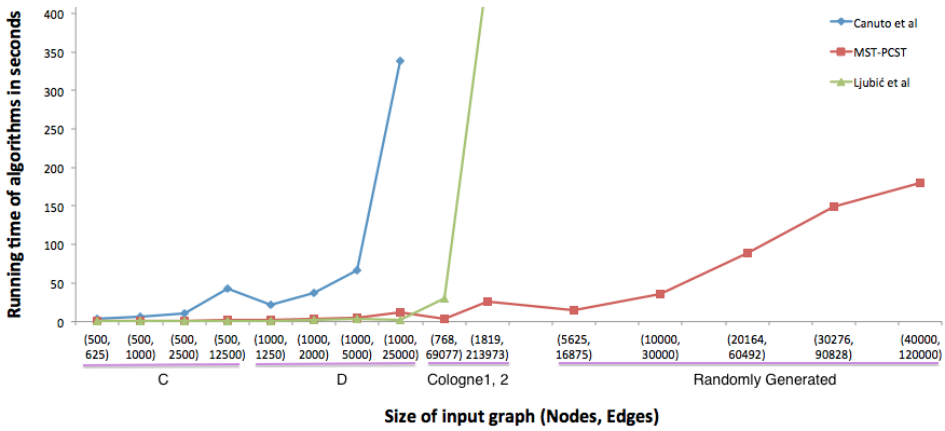

Fig. 4. The number of nodes in input graph versus MST-PCST algorithm running time plot

The Table 1 summarizes the average gap percentage of results obtained by MST-PCSTP with respect to the optimal solutions. The first and third rows of the table provide the information about the size of test instances. The last two instance sets in the third row belong to Cologne 1 and 2 test sets, and each corresponding gap value demonstrates the average optimality gap of 15 instances. The rest of the instances refer to C and D sets, and each corresponding gap value indicates the average optimality gap of 10 instances. Although MST-PCST is a simple heuristic, it demonstrated a good performance on these instances with the tolerable gap percentages. From the table it is easy to notice that MST-PCSTP achieved solutions with smaller optimality gap on a larger graph instances. Furthermore, the heuristic's faster running time on the larger instances makes it advantageous to analyze the huge networks from real-world applications.

Table 1. The average optimality gap of MST-PCSTP on test instances C, D, and Cologne 1, 2

(V, E)	(500, 625)	(500, 1000)	(500, 2500)	(500, 12500)	(1000, 1250)
Gap (%)	1.53	3.37	3.51	8.47	1.80
(V, E)	(1000, 2000)	(1000, 5000)	(1000, 25000)	(768, 69077)	(1819, 213973)
Gap (%)	3.32	2.97	9.49	0.81	0.69

Conclusion

We extended a simple heuristic to solve PCSTP in this study. The developed approach is tested on benchmark instances of Ljubić *et al* (2005) and Canuto *et al* (2001) that are available in the literature, and randomly generated instances. The results obtained by MST-PCST demonstrated that the overall performance of the heuristic is good with a slight deviation from optimum. MST-PCST has a fast running time on real world instances and randomly generated large instances, therefore it has a potential to investigate the giant

real world instances. As future work, we plan to enrich the MST-PCST algorithm with local search with some perturbations in order to improve the objective value. Furthermore, it also can be integrated with Mixed Integer Linear Programming to achieve better results.

Acknowledgments— M. A. is supported by Swiss National Science Foundation through project 205321- 147138/1: "Steiner Trees for Functional Analysis in Cancer System Biology".

References

Barta J, Leggieri V, Montemanni R, Nobili P and Triki C (2010). Some valid inequalities for the Probabilistic Minimum Power Multicasting Problem. Electronic Notes in Discrete Mathematics, 36:463-470

Beasley JE (1990). OR-Library: distributing test problems by electronic mail. Journal of the Operational Research Society, 1069–1072

Becheta MB, Borgsb C, Braunsteinc A, Chayesb J, Dagkessamanskaiad A, Françoisd JM and Zecchina R (2010). Finding undetected protein associations in cell signaling by belief propagation. PNAS, 108:882–887

Bienstock D, Goemans MX, Simchi-Levi D and Williamson D (1993). A note on the prize collecting traveling salesman problem. Mathematical Progamming, 59:413–420

Canuto SA, Resende MGC and Ribeiro CC (2001). Local search with perturbation for the prize-collecting Steiner tree problem in graphs. Networks, 38:50–58

Chapovska O and Punnen AP (2006). Variations of the prize-collecting Steiner tree problem. Networks, 47:199-205

Dijkstra EW (1959). A note on two problems in connexion with graphs. Numerische Mathematik, 1:269–271

Dongarra JJ (2013). Performance of various computers using standard linear equations software. ACM SIGARCH Computer Architecture News, Technical report CS - 89 - 85, University of Tennessee

Feofiloff P, Fernandes CG, Ferreira CE (2007). Primal-dual approximation algorithms for the Prize-Collecting Steiner Tree Problem. Information Processing Letters, 103:195-202

Haouari M, Layeb SB and Sherali HD (2010). Strength of three MIP formulations for the prize collecting Steiner tree problem with a quota constraint. Electronic Notes in Discrete Mathematics, 36:495–5021

Johnson DS, Minkoff M and Phillips S (2000). The prize collecting Steiner tree problem: theory and practice. Proc. 11th ACM-SIAM Symp. on Discrete Algorithms,

Klau GW, Ljubić I, Moser A, Mutzel P, Neuner P, Pferschy U, Raidl G and Weiskircher R (2004). Combining a memetic algorithm with integer programming to solve the prize-collecting Steiner tree problem. Genetic and Evolutionary Computation, 3102:1304-1315

Kou L, Markowsky G and Berman L (1981). A fast algorithm for Steiner Trees. Acta Informatica, 141-145

Ljubić I (2004). Exact and Memetic Algorithms for Two Network Design Problems. PhD thesis, Faculty of Computer Science, Vienna University of Technology

Ljubić I, Weiskircher R, Pferschy U, Klau G, Mutzel P and Fischetti M (2005). Solving the prize-collecting Steiner tree problem to optimality. Proceedings of ALENEX, Seventh Workshop on Algorithm Engineering and Experiments

Ljubić I, Weiskircher R, Pferschy U, Klau GW, Mutzel P and Fischetti M (2006). An algorithmic framework for the exact solution of the prize-collecting Steiner tree problem. Mathematical Progamming

Miranda EA, Ljubić I and Toth P (2013). Exact approaches for solving robust prize-collecting Steiner tree problems. European Journal of Operational Research, 229:599–612

Montemanni R and Leggieri V (2010). An exact algorithm for the minimum power multicasting problem in wireless sensor networks. Electronic Notes in Discrete Mathematics, 36:215-222

Montemanni R and Leggieri V (2011). A Branch and Price Algorithm for the Minimum Power Multicasting Problem in Wireless Sensor Networks. Mathematical Methods of Operations Research, 74(3):327-342

Montemanni R, Leggieri V and Triki C (2008). Mixed integer formulations for the probabilistic minimum energy broadcast problem in wireless networks. European Journal of Operational Research, 190(2):578-585

Prim RC (1957). Shortest connection networks and some generalizations. Bell System Technical Journal,1389–1401

Segev A (1987). The node-weighted Steiner tree problem. Networks, 17:1–17

Siek J, Lee LQ and Lumsdaine A (2000). Boost Graph Library, http://www.boost.org/libs/graph/

Tuncbag N, McCallum S, Huang SC and Fraenkel E (2012). SteinerNet: a web server for integrating 'omic' data to discover hidden components of response pathways. Nucleic Acids Research, 1-5