A Fast Vertex-Swap Operator for the Prize-Collecting Steiner Tree Problem

Yi-Fei Ming, Si-Bo Chen, Yong-Quan Chen, Zhang-Hua Fu*(corresponding)

Robotics Laboratory for Logistics Service, Institute of Robotics and Intelligent Manufacturing, The Chinese University of Hong Kong, Shenzhen, 518172, China.
115010203@link.cuhk.edu.cn(Y.F.Ming),115010121@link.cuhk.edu.cn(S.B.Chen)
yqchen@cuhk.edu.cn(Y.Q.Chen), Fzh.cuhksz@gmail.com(Z.H.Fu)

Abstract. The prize-collecting Steiner tree problem (PCSTP) is one of the important topics in computational science and operations research. The vertex-swap operation, which involves removal and addition of a pair of vertices based on a given minimum spanning tree (MST), has been proven very effective for some particular PCSTP instances with uniform edge costs. This paper extends the vertex-swap operator to make it applicable for solving more general PCSTP instances with varied edge costs. Furthermore, we adopt multiple dynamic data structures, which guarantee that the total time complexity for evaluating all the $O(n^2)$ possible vertex-swap moves is bounded by $O(n) \cdot O(m \cdot \log n)$, where n and m denote the number of vertices and edges respectively (if we run Kruskal's algorithm with a Fibonacci heap from scratch after swapping any pair of vertices, the total time complexity would reach $O(n^2) \cdot O(m + n \cdot \log n)$). We also prove that after applying the vertex-swap operation, the resulting solutions are necessarily MSTs (unless infeasible).

Keywords: Computational complexity, network design, prize-collecting Steiner tree, vertex-swap operator, dynamic data structures.

1 Introduction

The prize-collecting Steiner tree problem (PCSTP) has a wide range of applications, e.g., design of utility network, telecommunication network, signal processing. As a variant of the classic Steiner tree problem in graphs, the PCSTP is *NP-hard*, thus being important in the field of computational science.

Given an undirected graph $G = (V, E)$ with a set V ($|V| = n$) of vertices and a set E ($|E| = m$) of edges, where each edge $e \in E$ is associated with a non-negative edge cost c_e, and each vertex $v \in V$ is associated with a non-negative prize p_v (vertex v is a customer vertex if $p_v > 0$ and a non-customer vertex otherwise), the goal of the PCSTP is to find a subtree $T = (V_T, E_T)$ of G in which the total cost of edges in the tree plus the total prize of vertices not in the tree is minimized, i.e., [1]:

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

ICCS Camera Ready Version 2018
To cite this paper please use the final published version:
DOI: 10.1007/978-3-319-93701-4_43

Many algorithms have been proposed to solve the PCSTP, including several heuristics, such as multi-start local-search algorithm combined with perturbation [2], trans-genetic hybrid algorithm [3], divide-and-conquer meta-heuristic method [4], knowledge-guided tabu search [5], etc. Among various heuristics for solving the PCSTP, local search enjoys popularity in the literature, which commonly relies on two basic move operators, i.e., vertex addition and vertex deletion. Typically, the vertex addition (deletion) operator tries to add (delete) a vertex $v \notin V_T (v' \in V_T)$ to (from) an original minimum spanning tree (MST) and then tries to reconstruct a new MST, leading to a neighboring solution. Though these two basic move operators are generally effective, improvements could be achieved by introducing a new vertex-swap operator, which substitutes one vertex in the original MST with another one out of the original MST, and then reconstructs a new MST as the neighboring solution.

Unfortunately, although the basic idea of the vertex-swap operator is natural, it has not been widely employed in the existing PCSTP heuristics, possibly due to its unaffordable complexity: if we choose to reconstruct an MST using Kruskal's algorithm (with the aid of a Fibonacci heap) from scratch after swapping any pair of vertices, the overall time complexity for evaluating all the $O(n^2)$ possible vertex-swap moves would reach $O(n^2) \cdot O(m + n \cdot \log n)$, being unaffordable for large-sized (even mid-sized) instances.

During the 11^{th} DIMACS Implementation Challenge, Zhang-Hua Fu (corresponding author of this paper) and Jin-Kao Hao implemented a dynamic vertex-swap operator [6], based on which they proposed a local-search heuristic [5], which won three out of the eight PCSTP competing sub-categories of the DIMACS challenge. Actually, the vertex-swap operator contributed significantly to the outstanding performance of the proposed algorithm. However, its application was limited to a number of particular PCSTP instances with uniform edge costs. In this paper, we extend the previous work in order to develop an efficient vertex-swap operator which is suitable for more general PCSTP instances, not only limited to the ones with uniform edge costs. With the aid of dynamic data structures, the time complexity for evaluating all the $O(n^2)$ possible vertex-swap moves could be reduced from $O(n^2) \cdot O(m + n \cdot \log n)$ to $O(n) \cdot O(m \cdot \log\ n)$. The details as well as proof of complexity and correctness are given below.

2 Method and complexity

Given a solution $T = (V_T, E_T)$ of the PCSTP, two basic move operators (vertex-addition and vertex-deletion) are commonly used, which adds a vertex $v' \notin V_T$ to (respectively, removes a vertex $v \in V_T$ from) V_T, and then tries to reconstruct an MST denoted by $\mathrm{MST}(V_T \cup \{v'\})$ (respectively, $\mathrm{MST}(V_T \backslash \{v\})$). Corresponding to these two move operators, two sub-neighborhoods are defined as follows:

$$\begin{aligned} N_1(T) &= MST(V_T \cup \{v'\}), \forall v' \notin V_T, \\ N_2(T) &= MST(V_T \backslash \{v\}), \forall v \in V_T. \end{aligned} \tag{2}$$

Based on the above two basic operators, the vertex-swap operator consists of the following two phases (outlined in Algorithm 1). The solutions are represented

ICCS Camera Ready Version 2018
To cite this paper please use the final published version:
DOI: 10.1007/978-3-319-93701-4_43

as dynamic data structures such as ST-trees [7, 8], which takes $O(\log n)$ time to perform basic operations, i.e., searching, removing and inserting an edge.

Algorithm 1 Procedure of evaluating all the $O(n^2)$ possible vertex-swap moves.

Input: An MST $T = (V_T, E_T)$
Output: Cost difference $\Delta(v, v')$ after swapping any vertices $v \in V_T$ and $v' \notin V_T$
$T^* \leftarrow T$ //T^* always denotes the incumbent solution
for each vertex $v \in V_T$ (processed in post order) **do**
  $T^* \leftarrow Deletion(T^*, v)$ //apply the deletion phase to T^* relative to v
  $T^{Del} \leftarrow T^*$
  for each vertex $v' \notin V_T$ **do**
    $T^* \leftarrow Addition(T^*, v')$ //apply the addition phase to T^* relative to v'
    if T^* is a tree **then**
      $\Delta(v, v') \leftarrow f(T^*) - f(T)$
    else
      $\Delta(v, v') \leftarrow Null$
    end if
    $T^* \leftarrow T^{Del}$ //restore the solution before addition (only restore the changes)
  end for
  $T^* \leftarrow T$ //restore the original solution (only restore the changes)
end for

Vertex deletion phase: Given an original MST $T = (V_T, E_T)$, for a chosen vertex $v \in V_T$, we first remove it from T, together with the edges incident to v. This operation leads to an minimum spanning forest (MSF) consisting of a number of sub-trees (consider an MST as a special case of MSF with only one sub-tree, so as follows), where each sub-tree is an MST. After that, we try to reconnect the remaining sub-trees as far as possible. To do this, it suffices to compact each sub-tree into a super-vertex, and then run Kruskal's algorithm on the subgraph consisting of all the super-vertices along with edges between different super-vertices (if there are multiple edges between two super-vertices, just retain the one with the lowest cost). After this process, we get an MSF consisting of k ($k \geq 1$) sub-trees: $T_1, T_2, \cdots, T_k$, where each sub-tree is an MST and there is no edge between any two different sub-trees.

Complexity: As illustrated in Algorithm 1, given an original MST $T = (V_T, E_T)$, each vertex $v \in V_T$ should be deleted only once. Using the dynamic data structures slightly adapted from the vertex-elimination operator detailed in [9], which process the vertices of V_T in post order and classify the edges of E_T into horizontal edges (stored in lists) and vertical edges (stored in logarithmic-time heaps and updated dynamically), the total time complexity of this phase is bounded by $O(m \cdot \log n)$ (proven in [9]).

Vertex addition phase: For a chosen vertex $v' \notin V_T$, add it to each sub-tree T_i ($1 \leq i \leq k$) of the above MSF, to form a new MST. To do this, Spira and Pan [10] showed that for one sub-tree $T_i = (V_{T_i}, E_{T_i})$, it is enough to determine

ICCS Camera Ready Version 2018
To cite this paper please use the final published version:
DOI: 10.1007/978-3-319-93701-4_43

the MST on sub-graph $G' = (V_{T_i} \cup \{v'\}, E_{T_i} \cup E_N(T_i, v'))$, where $E_N(T_i, v')$ denotes the collection of edges connecting v' to T_i. For each edge e incident to v', if $e \in E_N(T_i, v')$, insert e into T_i at first and then check if a cycle is formed. If so, remove the edge with the highest cost on the cycle [9]. After repeating this process for every edge e, a new MST is reconstructed (unless infeasible).

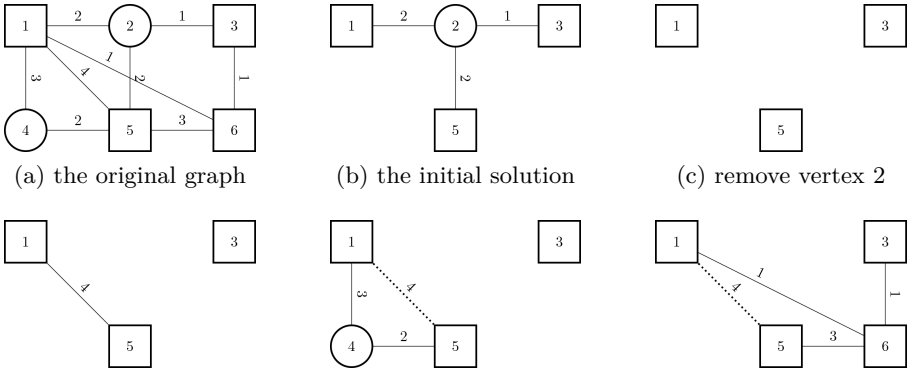
Complexity: After performing the vertex deletion phase for each vertex $v \in V_T$, we try to add every vertex out of V_T (added one by one) into the resulting MSF and then eliminate cycles. During this process, at most m edges would be inserted or removed in total. With the help of ST tree, it takes $O(\log n)$ to insert/remove one edge to/from a sub-tree [7, 8]. Therefore, after deleting each vertex $v \in V_T$, the complexity of adding all the vertices is $O(m) \cdot O(\log\ n)$. Since at most $O(|V_T|) \leq O(n)$ vertices should be deleted, the total complexity of the vertex addition phase is bounded by $O(n) \cdot O(m \cdot \log\ n)$.

In addition to above two phases, we further analyze the complexity of storage and restoration. As illustrated in Algorithm 1, we only store and restore the changed vertices and edges whenever needed, instead of the whole tree. During the whole procedure, every edge belonging to E_T is deleted twice by the **vertex deletion phase**, and at most $2|E_T|$ edges are added to connect the sub-trees. Furthermore, during the **vertex addition phase**, each edge (in total m edges) is added at most n times (at most once after deleting each vertex of V_T), and at most $m \cdot n$ edges are deleted (totally no more than added edges) to eliminate cycles. It means at most $O(m \cdot n)$ changes in total should be stored and restored. Since the complexity for storing or restoring a change is $O(1)$ and $O(\log\ n)$ respectively, the total complexity of these steps is $O(n) \cdot O(m \cdot \log\ n)$.

Summary: Given an original MST $T = (V_T, E_T)$, the total complexity for evaluating all the $O(n^2)$ vertex-swap based neighboring solutions (Algorithm 1) is bounded by $O(n) \cdot O(m \cdot \log\ n)$.

Fig.1 gives an example, where sub-figure (a) is the original graph consisting of 4 customer vertices (drawn in boxes, each with a prize of 1) and 2 non-customer vertices (drawn in circles). Sub-figure (b) is an initial solution (MST) with an objective value of 6. Now we show how to swap vertex 2 with vertices 4 and 6 (similar for others). At first, we remove vertex 2 and its incident edges, leading to a MSF shown in sub-figure (c). Then we run Kruskal's algorithm to reconnect these sub-trees (regarding each sub-tree as a super-vertex), leading to the MSF shown in sub-figure (d), where vertex 1 is reconnected to vertex 5. Furthermore, to add vertex 4, we add the edge between vertex 1 and vertex 4 first, and add the edge between vertex 4 and vertex 5, which leads to a cycle. To eliminate the cycle, we remove the edge between vertex 1 and vertex 5, leading to the solution shown in sub-figure (e), which is infeasible. Similarly, for vertex 6, we at first restore the solution before addition of vertex 4, and insert in sequence three edges (between vertex 6 and vertices 1, 3, 5 respectively), then we remove the edge between vertex 1 and vertex 5 to eliminate cycle, resulting a MST with an objective value of 5 ($\Delta(2, 6) = -1$), as shown in sub-figure (f).

ICCS Camera Ready Version 2018
To cite this paper please use the final published version:
DOI: 10.1007/978-3-319-93701-4_43



(a) the original graph (b) the initial solution (c) remove vertex 2

(d) reconnect the forest (e) add vertex 4 (infeasible) (f) add vertex 6 (feasible)

Fig. 1: Example showing how to apply the swap-vertex move operator

3 Proof of correctness

Now we prove that using above dynamic techniques, the final solution after swapping any pair of vertices is necessarily an MST (unless being a forest).

Lemma 1. Given an MST $T = (V_T, E_T)$, performing the vertex deletion phase with respect to vertex $v \in V_T$ would lead to an minimal spanning forest (MSF), consisting of $k \geq 1$ sub-trees (denoted by $T_1, T_2, \cdots T_k$ respectively, and each is an MST).

Proof: Proven in [11]. ■

Lemma 2. For any vertex $v' \notin V_T$, if v' can be connected to sub-tree $T_i (1 \leq i \leq k)$, after performing the vertex addition phase, T_i would become a new MST denoted by T_i' ($V_{T_i'} = V_{T_i} \cup \{v'\}$).

Proof: Proven in [9]. ■

For lemma 3 to lemma 5, we consider two trees (unnecessarily MSTs) $T_i' = (V_{T_i'}, E_{T_i'})$ and $T_j' = (V_{T_j'}, E_{T_j'})$, which satisfy the following two conditions:

(1) v' is the only common vertex between $V_{T_i'}$ and $V_{T_j'}$, i.e., $V_{T_i'} \cap V_{T_j'} = \{v'\}$.

(2) there is no direct edge between $V_{T_i'} \backslash \{v'\}$ and $V_{T_j'} \backslash \{v'\}$.

Lemma 3. By merging T_i' and T_j', the resulting graph $G' = (V_{G'}, E_{G'}) = (V_{T_i'} \cup V_{T_j'}, E_{T_i'} \cup E_{T_j'})$ is a tree.

Proof: (1) T_i' and T_j' are both trees, thus any vertex $h \in V_{T_i'} \backslash \{v'\}$ ($g \in V_{T_j'} \backslash \{v'\}$) is connected to v', implying that any two vertices of $V_{T_i'} \cup V_{T_j'}$ are connected. (2) T_i' and T_j' are both trees, and v' is the only common vertex, so:

ICCS Camera Ready Version 2018
To cite this paper please use the final published version:
DOI: 10.1007/978-3-319-93701-4_43

$$
\begin{aligned}
|V_{G'}| &= |V_{T_i'}| + |V_{T_j'}| - 1, \\
|E_{G'}| &= |E_{T_i'}| + |E_{T_j'}| \\
&= |V_{T_i'}| - 1 + |V_{T_j'}| - 1 \\
&= |V_{G'}| - 1
\end{aligned}
$$

Above information indicates that G' is a tree. ■

Lemma 4. Any tree T_{any} based on vertex set $V_{T_i'} \cup V_{T_j'}$ can be exactly partitioned into two sub-trees based on vertex set $V_{T_i'}$ and $V_{T_j'}$ respectively.

Proof: (1) T_{any} is a tree, thus no cycle exists among $V_{T_i'} \cup V_{T_j'}$, so no cycle exists among $V_{T_i'}$ and $V_{T_j'}$. (2) Now we prove that any two vertices $h, g \in V_{T_i'}$ can be connected only via vertices of $V_{T_i'}$. Since T_{any} is a tree, there must be one and only one path connecting h and g. Assume another vertex $l \in V_{T_j'} \backslash \{v'\}$ appears on this path, since there is no edge between $V_{T_i'} \backslash \{v'\}$ and $V_{T_j'} \backslash \{v'\}$, v' must appear on the path from h to l, so does on the path from l to g, leading to a cycle (v' appears twice), contradicting to the statement that T_{any} is a tree, indicating $V_{T_i'}$ is internally connected. Similarly, $V_{T_j'}$ is internally connected. ■

Lemma 5. If T_i' and T_j' are both MSTs with cost $C_{T_i'} = \sum_{e \in E_{T_i'}} c_e = C_{T_i'}^{min}$ and $C_{T_j'} = \sum_{e \in E_{T_j'}} c_e = C_{T_j'}^{min}$ respectively, the graph G' formed by merging T_i' and T_j' is also an MST with cost $C_{G'} = \sum_{e \in E_{G'}} c_e = C_{T_i'}^{min} + C_{T_j'}^{min}$.

Proof: (1) According to lemma 3, G' is a tree with cost $C_{G'} = C_{T_i'}^{min} + C_{T_j'}^{min}$. (2) According to lemma 4, any solution T_{any} based on vertex set $V_{T_i'} \cup V_{T_j'}$ can be exactly partitioned into two sub-trees based on vertex set $V_{T_i'}$ and $V_{T_j'}$, so its cost $C_{any} \geq C_{T_i'}^{min} + C_{T_j'}^{min} = C_{G'}$, implying that the cost of G' is minimized. ■

Theorem 1. Given an initial MST $T = (V_T, E_T)$, after performing the procedure illustrated in Algorithm 1, the final solution after swapping a pair of vertices $v \in V_T$ and $v' \notin V_T$ is necessarily an MST (unless infeasible).

Proof: (1) According to lemma 1, applying the vertex deletion phase respect to vertex $v \in V_T$ leads to a MSF consisting of $k \geq 1$ sub-trees $T_1, T_2, \cdots, T_k$ (each is an MST). (2) Assume $v' \notin V_T$ can be connected to every sub-tree obtained above (otherwise, the solution after swapping v with v' is a forest, being infeasible), according to lemma 2, after applying the vertex addition phase with respect to vertex v', each sub-tree $T_i (1 \leq i \leq k)$ becomes a new MST T_i'. (3) Note that any two sub-trees T_i' and T_j' $(1 \leq i \neq j \leq k)$ satisfy the two conditions mentioned before lemma 3. According to lemma 5, the graph formed by combining T_i' and T_j' is an MST. By induction, the whole graph formed by combining $T_1', T_2', \cdots, T_k'$ is an MST (unless infeasible). ■

ICCS Camera Ready Version 2018
To cite this paper please use the final published version:
DOI: 10.1007/978-3-319-93701-4_43

4 Conclusion

This paper develops an efficient vertex-swap operator for the prize-collecting Steiner tree problem (PCSTP), which is applicable to general PCSTP instances with varied edge costs, not only limited to instances with uniform edge costs. A series of dynamic data structures are integrated to guarantee that the total time complexity for evaluating all the $O(n^2)$ possible vertex-swap moves is bounded by $O(n) \cdot (m \cdot \log n)$, instead of the complexity $O(n^2) \cdot O(m + n \cdot \log n)$ by running Kruskal's algorithm from scratch after swapping any pair of vertices (with the aid of a Fibonacci heap). We also prove that using the developed techniques, the resulting solutions are necessarily minimum spanning trees (unless infeasible).

5 Acknowledgements

This paper is partially supported by the National Natural Science Foundation of China (grant No: U1613216), the State Joint Engineering Lab on Robotics and Intelligent Manufacturing, and Shenzhen Engineering Lab on Robotics and Intelligent Manufacturing, from Shenzhen Gov, China.

References

1. Johnson, D.S., Minkoff, M., and Phillips, S., The prize collecting steiner tree problem: theory and practice. *Proceeding of the Eleventh Annual Acm-SIAM Symposium on Discrete Algorithms*, Philadelphia, USA, 760-769, 2000.
2. Canuto, S.A., Resende, M.G.C., and Ribeiro, C.C., Local search with perturbations for the prize collecting Steiner tree problem in graphs. *Networks* 38, 50-58, 2001.
3. Goldbarg, E.F.G, Goldbarg, M.C, and Schmidt, C.C., A Hybrid Transgenetic Algorithm for the Prize Collecting Steiner Tree Problem. *Jounal of Universal Computer Science*, 14, 2491-2511, 2008.
4. Akhmedov, M., Kwee, I., and Montemanni, R., A divide and conquer matheuristic algorithm for the prize-collecting Steiner tree problem. *Computers & Operations Research*, 70, 18-25, 2016.
5. Fu, Z.H., and Hao, J.K., Knowledge-guided local search for the prize-collecting Steiner tree problem in graphs. *Knowledge-Based Systems*, 128, 78-92, 2017.
6. Fu, Z.H., and Hao, J.K., Swap-vertex based neighborhood for Steiner tree problems. *Mathematical Programming Computation*, 9, 297-320, 2017.
7. Sleator, D.D., Tarjan, R.E., A data structure for dynamic trees. *Journal of Computer and System Sciences*, 26, 362-391, 1983.
8. Sleator, D.D., Tarjan, R.E., Self-adjusting binary search trees. *Journal of the ACM*, 32, 652-686, 1985.
9. Uchoa, E., and Werneck, R.F., Fast local search for steiner trees in graphs. *2010 Proceedings of the Twelfth Workshop on Algorithm Engineering and Experiments (ALENEX). Society for Industrial and Applied Mathematics*, 1-10, 2010.
10. Spira, P.M., and Pan, A., On finding and updating spanning trees and shortest paths. *SIAM Journal on Computing*, 4, 375-380, 1975.
11. Das, B., and Michael, C.L., Reconstructing a minimum spanning tree after deletion of any node. *Algorithmica*, 31, 530-547, 2001.

ICCS Camera Ready Version 2018
To cite this paper please use the final published version:
DOI: 10.1007/978-3-319-93701-4_43