The Weighted k-Center Problem in Trees for Fixed k

Binay Bhattacharya
Simon Fraser University, Burnaby, Canada
binay@cs.sfu.ca

Sandip Das
Indian Statistical Institute, Kolkata, India
sandipdas@isical.ac.in

Subhadeep Ranjan Dev
Indian Statistical Institute, Kolkata, India
srdev_r@isical.ac.in

Abstract

We present a linear time algorithm for the weighted k-center problem on trees for fixed k. This partially settles the long-standing question about the lower bound on the time complexity of the problem. The current time complexity of the best-known algorithm for the problem with k as part of the input is $O(n \log n)$ by Wang et al. [15]. Whether an $O(n)$ time algorithm exists for arbitrary k is still open.

2012 ACM Subject Classification Theory of computation → Facility location and clustering; Theory of computation → Network optimization

Keywords and phrases facility location, prune and search, parametric search, k-center problem, conditional k-center problem, trees

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2019.27

1 Introduction

In this paper, we study a popular facility location problem on graphs called the weighted k-center problem. The sites are the vertices of the graph and have positive weights associated with them. The edges of the graph have positive lengths and the facilities can be placed anywhere on them. The weighted distance between a facility and a site is the length of the shortest path between them times the weight of the site. Our objective is to place k facilities on the graph such that the maximum weighted distance of a site to its closest facility is minimized.

Kariv and Hakimi [9] in 1979 showed that the weighted k-center problem on general graphs is NP-hard. In fact, they proved a much stronger statement that finding the weighted k-center is NP-hard for planar graphs with maximum vertex degree 3. On the other hand in the same paper, they gave an $O(n^2 \log n)$ time algorithm for the problem if the underlying graph is a tree with n vertices. Later improvements in time complexity were done by Jeger and Kariv [8] to $O(kn \log n)$, and Megiddo and Tamir [12] to $O(n \log^2 n)$ using Cole's [6] optimization. Very recently in 2016 Banik et. al. [1] gave an $O(n \log n + k \log^2 n \log(n/k))$ time algorithm for the problem which was then improved to $O(n \log n)$ by Wang and Zhang [15] in 2018. Wang and Zang's solution is the current state of the art for the weighted k-center problem in trees for arbitrary value of k.

For $k = 1$ the problem had already been solved by Megiddo [11] in $O(n)$ time in 1983. In 2006 Ben-Moshe et al. [2] showed that the 2-center problem can also be computed in $O(n)$ time. In the same paper, they gave an $O(n \log n)$ time algorithm for the weighted k-center problem when k is 3 or 4. The problem has also been studied in the real line setting where the sites and facilities are constrained to be placed on a given line. In this setting Bhattacharya et al. [5] gave an $O(n)$ time solution for the weighted k-center problem for any fixed k. For the unweighted case, the k-center problem on trees has been solved optimally by Frederickson [7] in $O(n)$ time. Karmakar et al. [10] studied some constrained version of this problem in R^2 and gave algorithms which are near linear in n.

In this paper, we study the weighted k-center problem on trees for any constant k and present an $O(n)$ time solution for it. Our algorithm generalises the technique used in Bhattacharya et al. [4] where they give a linear time algorithm for finding the k-step fitting function of n points in $\mathbb{R}^2$. A different (unpublished) linear time solution to the weighted k-center problem was also suggested in 2008 by Shi [14]. The algorithm presented in this paper is simpler and is based on the fact that a generalization of Megiddo's [11] approach applies suitably to our problem. This enables us to prune vertices from a subgraph of the tree which in turn produces a linear time algorithm.

The rest of the paper is organized as follows. In Section 2 we define the weighted k-center problem and the conditional weighted k-center problem. We also briefly mention the r-feasibility test. In Section 3 we introduce the notion of a *big-component* of a tree and propose a linear time algorithm to find it. In Section 4 we show how to prune the vertices of the big-component in order to reduce the size of the original tree. In Section 5 we present our linear time algorithm for the conditional weighted k-center problem which is a generalisation of the weighted k-center problem. We then conclude in Section 6.

2 Preliminaries

2.1 Problem Definition

Let T be a tree with vertex set $V(T)$ and edge set $E(T)$. Also, let the number of vertices in T, denoted by $|T|$, be n. Each vertex $v \in V(T)$ is associated with a positive weight $w(v)$ and each edge $e \in E(T)$ is associated with a positive length $l(e)$. To define the notion of points on an edge e, we assume e to be a line segment with length $l(e)$. The distance between two points x and y in e is proportional to the portion of e in between x and y with respect to $l(e)$. $A(T)$ denotes the set of all points on all edges of T.

The distance between any two points $x, y \in A(T)$, denoted by $l(x, y)$, is the sum of the lengths of the edges and the partial edges in the unique path between x and y in T. The weighted distance between a vertex $v \in V(T)$ and a point $x \in A(T)$ is defined as $d(x, v) = w(v) \cdot l(x, v)$. We extend this definition to include the weighted distance between a set of points $X \subseteq A(T)$ and a set of vertices $V' \subseteq V(T)$ which is given by the expression

$$d(X, V') = \max_{v \in V'}\{\min_{x \in X}\{d(x, v)\}\}$$

The objective of the weighted k-center problem on T is to find a set of points $X \subset A(T)$ with $|X| = k$ such that $d(X, V(T))$ is minimum. The points in X are called *centers*.

We solve the weighted k-center problem by solving a more general problem on trees which is called the *conditional* weighted (p, S)-center problem or the (p, S)-center problem in short. The problem was first introduced in Minieka [13]. Here, $S \subset A(T)$ is the set of centers already placed in T. We call S the set of *old centers*. The objective of the (p, S)-center problem is to find a set of points $X \subset A(T)$ with $|X| = p$ such that $d(X \cup S, V(T))$ is minimized. We call X the set of *new centers*. Observe that the solution to the (p, S)-center problem is also the solution to the weighted k-center problem, when $S = \varnothing$ and $p = k$.

2.2 The r-feasibility test

Let X be an optimal solution to the (p, S)-center problem in T. The optimal radius r^* is defined as the maximum weighted distance of a vertex to its closest center in $X \cup S$ i.e. $r^* = d(X \cup S, V(T))$. A point $x \in A(T)$ is said to *cover* a vertex $v \in V(T)$ with radius r if $d(x, v) \leq r$. Similarly, we say that a set of centers X' covers a set of vertices V' with radius r if $d(X', V') \leq r$. If no radius is mentioned, we assume r to be r^*.

The r-feasibility test for the (p, S)-center problem on T takes as input a radius r and returns (a) *feasible* if $r \geq r^*$ and, (b) *infeasible* if $r < r^*$.

The algorithm was first presented by Kariv and Hakimi [9] in 1979 for the weighted k-center problem. The feasibility test for the (p, S)-center problem follows the same principle and is described in Shi [14]. The feasibility test takes $O(n)$ time.

2.3 Our Approach

The solution to the (p, S)-center problem presented here is recursive in nature. We first introduce the notion of a *big-component* of T. For a suitable constant c, a big-component is a subtree T' of T with at least $\frac{n}{c}$ vertices. Note that $T \setminus T'$ can be disconnected. It has the additional property that for some optimal solution X to the (p, S)-center problem the vertices of the big-component are "covered" by at most one new center from X. This property enables us to use the prune and search technique introduced by Megiddo [11], and generalized by Shi [14], to prune a constant fraction of the vertices of the big-component. The pruning step leads to a new tree T' with a fraction of the number of vertices in T. We recursively perform the last two steps of finding a big-component and then pruning its vertices on these new tree T' and the trees that follow until the size of the tree falls below a certain threshold. We then use any brute force technique to calculate the (p, S)-center in this final tree. The pruning step guarantees that the (p, S)-center solution of this reduced tree is also the (p, S)-center solution to the original tree T.

3 Big-Component

In this section we define a big-component and provide linear time algorithm to find it. Let $V \subseteq V(T)$ be such that the subgraph of T induced by V is connected. We call this induced subgraph an *induced subtree* of T. Let $V_u(v) = \{w \in V(T) \mid \text{the path from } v \text{ to } w \text{ passes through } u\}$; then $T_u(v)$ denotes the subtree induced by $V_u(v)$ and rooted at u (see Figure 1). Let $N(v)$ denote the neighbouring vertices of v in T, and let $N_{T'}(v)$ denotes the neighbouring vertices of v in the subtree T' of T. For $m \in \mathbb{Z}^+$, $[m]$ denotes the sequence $\{1, 2, \ldots, m\}$.

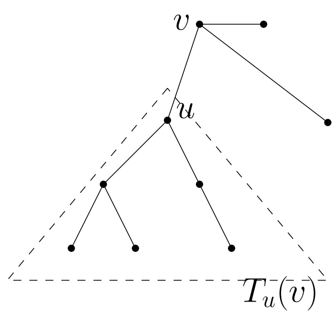


Figure 1 $T_u(v)$ is a subtree of T rooted at vertex u and not containing v.

We now introduce the notion of a big-component of T with respect to the (p, S)-center problem. Here, $k = p + |S|$.

▶ **Definition 1.** *Let B be a subtree of T with at least $\frac{n}{2k}$ vertices and let $S_B \subseteq S$ be the old centers contained in it. B is a big-component of T with respect to the (p, S)-center problem if for some solution X to the problem all vertices of B are covered by either (a) S_B or (b) a single new center $x \in X$.*

If B satisfies (a) we call B a type-a big-component and if it satisfies (b) we call it a type-b big-component.

In order to find a big-component, we first find a sequence of "candidate subtrees" of T. On of these candidate subtrees is a big-component of T. We search the subtrees in the candidate sequence sequentially until one of them identifies as a big-component.

3.1 Candidate Sequence

▶ **Definition 2.** *A subtree C of T is a candidate subtree of T if*

1. $\frac{n}{2k} \leq |C| < \frac{n}{k}$,
2. *C is connected to the rest of T i.e. $T \setminus C$ by a single vertex*

The vertex through which C is connected to the rest of T is called the exit vertex of C and is denoted by $v(C)$.

Note that removing all vertices of C except $v(C)$ from T still keeps the rest of T i.e. $T \setminus C \cup \{v(C)\}$ connected.

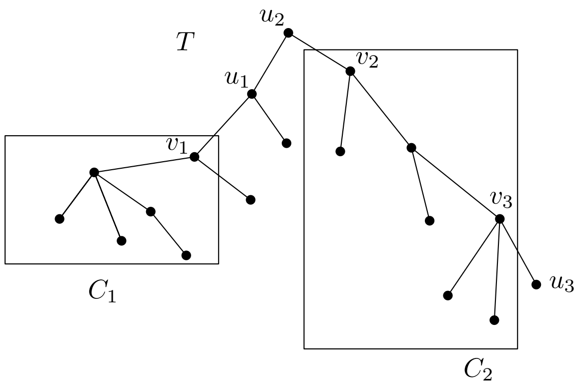


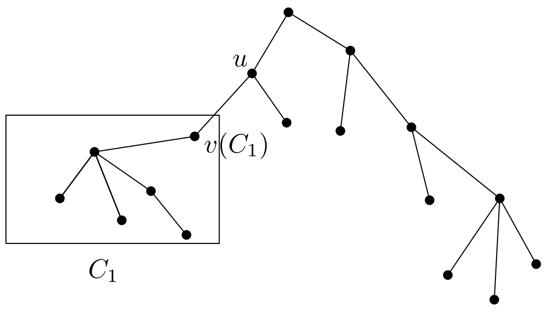
Figure 2 C_1 with exit vertex v_1 is a candidate subtree of T but C_2 is not. Here $n = 18$ and $k = 2$.

Let $T_1 = T$. For $i = 2, 3, \ldots$ we define T_i to be the tree generated by removing all vertices of C_{i-1} except $v(C_{i-1})$ from T_{i-1}. Here, C_i is a candidate subtree of T_i.

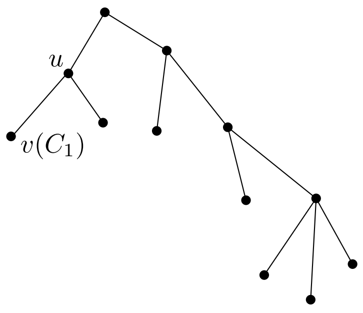
▶ **Definition 3.** *Let $\langle C_i \rangle_{i=1}^{m} = \langle C_1, C_2, \ldots, C_m \rangle$ be a sequence of m subtrees of T such that C_i is a candidate subtree of T_i. We call this sequence a* candidate sequence *of T.*

We now describe an algorithm to find a candidate sequence of T with respect to the (p, S)-center problem. Here we assume $k << n$.

▶ **Algorithm 1.**
Input: Tree T.
Output: A candidate sequence of T.



(a) T_1.



(b) T_2.

Figure 3 T_2 is formed by deleting all vertices of C_1 except $v(C_1)$.

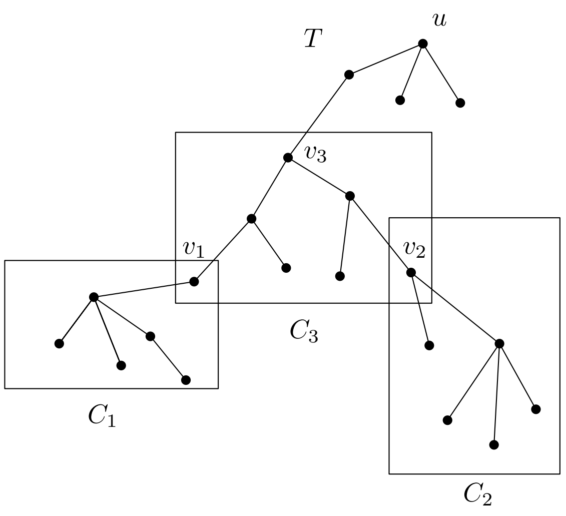


Figure 4 First C_1 is reported as a candidate subtree, then C_2 and then C_3.

Step 1: We consider $T' = T$ and assume it to be rooted at an arbitrary vertex u. We traverse the vertices of T' by first visiting the leaf vertices of T' and then visiting any vertex whose all children have already been visited. For each vertex visited we perform Step 2 until $|T'| < \frac{n}{2k}$.

Step 2: **a.** Let v be the current vertex in our traversal. All vertices in $T'_v(u)$ have already been visited. If $|T'_v(u)| < \frac{n}{2k}$ we mark v as visited and continue to the next vertex in our traversal. Otherwise, if $|T'_v(u)| \geq \frac{n}{2k}$ we do the next step.

b. Let $v_1, v_2, \dots, v_p$ be the children of v and let q be the smallest integer such that the subtree $C = \bigcup_{i=1}^{q} T'_{v_i}(v)$ has at least $\frac{n}{2k}$ vertices. Since for each child v_i, $|T'_{v_i}(v)| < \frac{n}{2k}$, we have that $|C| \leq \frac{n}{k}$. We report C as a candidate subtree with $v(C) = v$ and update T' by deleting all vertices of C except $v(C)$. We repeat Step 2b on v until $|T'_v(u)| < \frac{n}{2k}$.

The sequence of candidate subtrees found by Algorithm 1 is a candidate sequence of T. For an example see Figure 4.

▶ **Time Complexity.** Since Algorithm 1 performs a single traversal of T, it takes $O(n)$ time.

▶ **Observation 1.** *The length of the candidate sequence generated by Algorithm 1 is at least k and at most $2k$.*

Proof. The maximum size of a candidate subtree is $\lfloor \frac{n}{k} \rfloor$. Our algorithm stops only when the size of T_i falls below $\frac{n}{2k}$. Therefore, the length of the candidate sequence is at least k. Similarly, the minimum size of a candidate subtree is $\lceil \frac{n}{2k} \rceil$ and hence the length of the candidate sequence is at most $2k$. ◀

The next lemma justifies the need of finding a candidate sequence in order to find a big-component of T.

▶ **Lemma 4.** *Let $\langle C_i \rangle_{i=1}^{m}$ be a candidate sequence generated by Algorithm 1. Then there exists a $C_j \in \langle C_i \rangle_{i=1}^{m}$ which is a big-component of T.*

Proof. Let X be any optimal solution to the (p, S)-center problem. The covering of $X \cup S$ partitions T into k subtrees. These subtrees exclude exactly $k-1$ edges of T from it. Since, $m \geq k$, there exists at least one candidate subtree C_j which contains none of these $k-1$ edges. Then, by definition, this C_j is a big-component. ◀

3.2 The Big-Component Algorithm

We now present an algorithm to find a big-component B in T. The algorithm sequentially examines whether a subtree in the candidate sequence of T is a big-component. The algorithm is as follows.

▶ **Algorithm 2.**

Input: Tree T, integer p and old centers S.

Output: A big-component B of T with respect to the (p, S)-center problem.

Step 1: Compute a candidate sequence $\langle C_i \rangle_{i=1}^{m}$ of T using Algorithm 1. For each candidate subtree $C_i,\ i = 1, 2, \ldots, k-1 (k \leq m)$ we do Step 2 and Step 3.

Step 2: Let C_j be the current candidate subtree. Let $\langle C_{i_p} \rangle_{p=1}^{\hat{j}}$ be the longest subsequence of $\langle C_i \rangle_{i=1}^{j-1}$ such that the subtree $\hat{C}_j = C_j \cup (\bigcup_{p=1}^{\hat{j}} C_{i_p})$ is connected. Note that $\hat{C}_j$ can be computed using a standard tree traversal. See Figure 5 for an example. Also let $\hat{S}_j$ be the old centers in $\hat{C}_j$.

Step 3: We define p_j to be the number of candidate subtrees in $\hat{C}_j$ which do not contain any old centers. We do the following:

a. If $p_j < p$, we compute the $(p_j, \hat{S}_j)$-center on $\hat{C}_j$ and set r_j to be its radius. We do an r_j-feasibility test on T with respect to the (p, S)-center problem. We declare C_j to be a big-component and stop if the feasibility test returns infeasible.

b. If $p_j \geq p$, we compute the $(0, S \setminus \hat{S}_j)$-center in $(T \setminus \hat{C}_j) \cup \{v(C_j)\}$ and set r_j to be its radius. Do an r_j-feasibility test on T. We declare C_j to be a big-component and stop if the feasibility test returns feasible.

Step 4: If no subtree of $\langle C_i \rangle_{i=1}^{k-1}$ has been returned as a big-component in the above steps then we return C_k as a big-component of T. ◀

▶ **Lemma 5.** *The candidate subtree returned by Algorithm 2 is a big component of T.*

Proof. Consider a candidate subtree C_j and suppose it is not returned as a big-component. Then for the case $p_j < p$ the r_j-feasibility test returned feasible. Here, we assume $r_j > r^*$ since otherwise we have an optimal solution. This implies that in some optimal solution to the (p, S)-center problem on T, p_j new centers along with the old centers $\hat{S}_j$ are not enough to cover $\hat{C}_j$. For the case $p_j \geq p$, the r_j-feasibility test returned infeasible. Which implies that in the optimal case the old centers $S \setminus \hat{S}_j$ have to cover more than the vertices

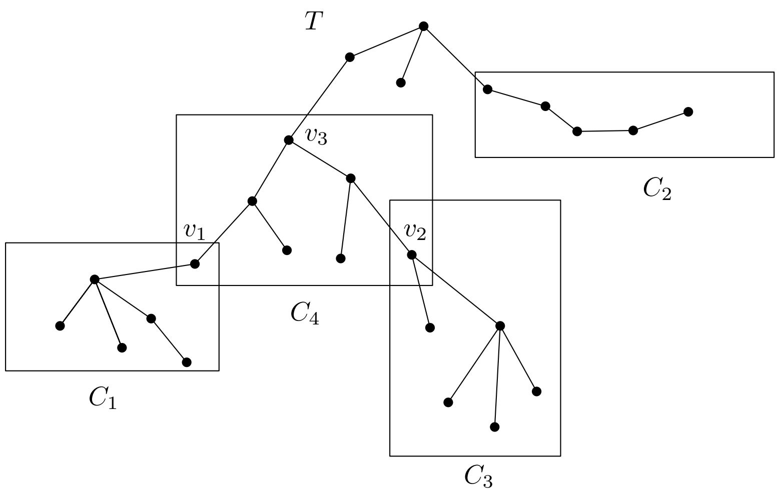


Figure 5 $\langle C_i \rangle_{i=1}^{4}$ is a candidate sequence of T. $\hat{C}_4 = C_4 \cup (C_1 \cup C_3)$. Note that we have already tested C_1, C_2 and C_3 for big-component.

in $T \setminus \hat{C}_j \cup \{v(C_j)\}$. This in turn implies that p new centers along with $\hat{S}_j$ are not sufficient to cover $\hat{C}_j$.

By a similar argument we can show that if C_j is returned as a big-component then at most $\min\{p, p_j\}$ centers are enough to cover the vertices in $\hat{C}_j$. And since all previous candidate subtrees $C_i, i \in [j-1]$ were not returned as big-components, it implies that C_j is either covered by the old centers in C_j or exactly one new center (if C_j contains no old centers). Therefore C_j, by definition is a big-component.

Now, if C_k was returned as a big-component then, to cover all vertices in $\bigcup_{i=1}^{k-1} C_i$ at least $k-1$ centers (old and new) are necessary. This in turn implies that C_k is completely covered by just 1 center, which again by definition is a big-component. ◀

▶ **Time Complexity.** Step 1 takes $O(n)$ time as shown in Section 3.1. Step 2 and step 3 iterates at most $k-1$ times. On the j^{th} iteration, the time taken to execute theses two steps is $T_{total}(p-1, j\frac{n}{k}) + O(n)$. Here, $T_{total}(p,n)$ is the time required to compute the (p,S)-center problem on a tree with n vertices. Therefore the time taken by Algorithm 2 is given by the following recurrence.

$$T_{big}(p,n) \leq \begin{cases} \sum_{j=1}^{k-1} T_{total}(p-1, j\frac{n}{k}) + O(kn) & , p \geq 1 \\ O(n) & , p = 0 \end{cases}$$

In this section we proposed an algorithm to find a big-component of T. In the rest of the paper, we present an algorithm to prune a constant fraction of vertices from the big-component without affecting solution of the (p,S)-center in T.

4 Vertex Pruning from a Big-Component

Pruning a vertex is analogous to deleting a vertex but with certain differences. While pruning a vertex v we take actions based on the degree of v.

If v is a degree 1 vertex and its incident edge does not contain an old center from S, we simply delete the vertex and its edge from T. Otherwise, v is not deleted. If v is a degree 2 vertex with neighbours v_1 and v_2 joined by edges e_1 and e_2 then, we delete v, e_1 and e_2 and join v_1 and v_2 with a new edge e with $l(e) = l(e_1) + l(e_2)$. If e_1 and e_2 had old centers, we place them on e at appropriate locations. If v is a vertex of degree 3 or more then we do not immediately delete it; instead we flag it for later deletion. The flagged vertices do not contribute to the size of the candidate subtree or the big-component. But, they do contribute when considering the degree of other vertices to be pruned. When we delete a vertex v, we also recursively delete any adjacent flagged vertex whose current degree has fallen below 3.

Let T' be the tree generated after pruning some vertices from T. T' can contain vertices of T which have been pruned out but have not been deleted. We have the following lemma.

▶ **Observation 2.** *The number of vertices in T' is proportional to the number of unpruned degree 1 and degree 2 vertices of T in T'.*

Proof. Let n_1 be the number of degree 1 vertices, n_2 the number of degree 2 vertices and n_3 the number of degree 3 or more vertices of T'. If n is the total number of vertices in T' then $n = n_1 + n_2 + n_3$. Since, the number of leaf vertices in a tree is always greater than the number of degree 3 or more vertices, $n_1 \geq n_3$. Therefore $n = n_1 + n_2 + n_3 \leq 2n_1 + n_2$. Here, n_2 does not count any pruned vertex of T which are still present in T'. The number of degree 1 vertices of T which have been pruned and yet have not been deleted from T' and which are counted in n_1 is at most k, a constant. Therefore, the claim holds. ◀

We are now ready to present an algorithm which prunes a constant fraction of the vertices of a big-component B without affecting the optimal solution to the (p, S)-center problem.

▶ **Algorithm 3.**
Input: Tree T, integer p, old centers S and a big-component B.
Output: Prune a constant fraction of vertices of B.

Step 1: If B has old centers present in it then B is a type-a big-component. Let $v_{far} \in V(B)$ be a vertex such that $d(S_B, v_{far}) = d(S_B, V(B))$. We prune all vertices in $V(B)$ except v_{far} and stop.
Else B is a type-b big-component and we proceed to the next step.

Step 2: We find a centroid v_c (a centroid is a vertex of tree whose removal splits the tree into forest such that all trees in the forest have size at most half the original tree. Such a vertex can be found out in time linear to the number of vertices) of B and compute the $(p-1, S \cup \{v_c\})$-center in T. Let the set of new centers be X and let $r_{far} = d(X \cup S \cup \{v_c\}, V(T))$. Let V_{far} be the set of vertices of T at a weighted distance r_{far} from their closest center i.e. $V_{far} = \{v_i \in V(T) \mid d\big(X \cup S \cup \{v_c\}, v_i\big) = r_{far}\}$.

Step 3: If the vertices in V_{far} lie in more than one subtree $T_{v_i}(v_c),\ v_i \in N(v_c)$ then moving the center placed at v_c away from the vertex in any direction will only increase the covering radius. Therefore, we can conclude that $r_{far} = r^*$ and $X \cup \{v_c\}$ is an optimal (p, S)-center solution of T. Let $v_{far} \in V_{far}$ be any arbitrary vertex. We prune all vertices in $V_{far} \setminus \{v_{far}\}$ and stop.
Else V_{far} is a subset of the vertices in a single subtree (say) $T_{v_0}(v_c)$ for some $v_0 \in N(v_c)$, we go to the next step.

Step 4: Since V_{far} is a subset of the vertices of $T_{v_0}(v_c)$, shifting the center from v_c in the direction of v_0 will reduce the covering radius. Therefore, the center which will cover the vertices of B lies in the direction of v_0 from v_c. Let V_{rest} be the set of vertices in the union of the subtrees $B_{v_i}(v_c), \forall v_i \in N_B(v_c), v_i \neq v_0$. All the vertices in V_{rest} are covered by a facility in $T_{v_0}(v_c)$, and therefore, are candidates for the pruning step.

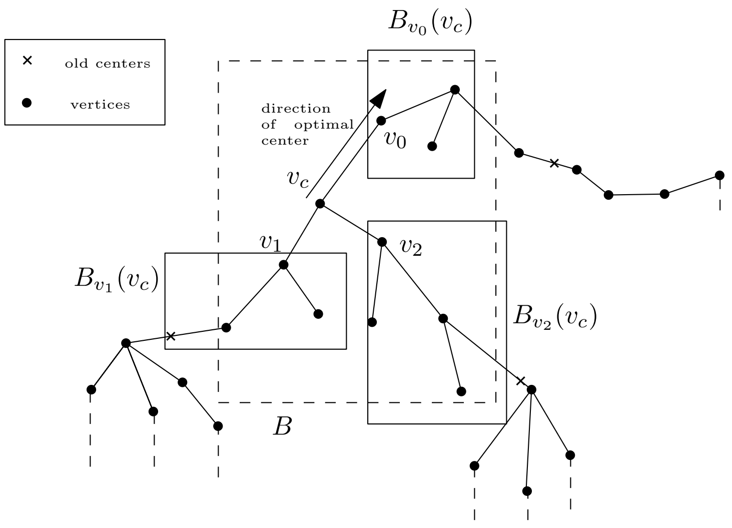


Figure 6 All vertices in V_{far} are inside $T_{v_0}(v_c)$. V_{rest} are the vertices in $B_{v_1}(v_c)$ and $B_{v_2}(v_c)$ together.

Step 5: Arbitrarily pair the vertices in V_{rest}. Leave out the last vertex if it cannot be paired. For each pair (a_i, b_i) with $w(a_i) \geq w(b_i)$, we compute the value

$$t_i = \frac{w(b_i) \times l(b_i, v_c) - w(a_i) \times l(a_i, v_c)}{w(a_i) - w(b_i)}$$

If t_i is negative then the weighted distance from the center (that covers both a_i and b_i) from a_i is always greater than that from b_i. Therefore we simply prune out b_i, as it cannot affect the value r^*. For the rest of the vertex pairs (a_i, b_i) we define $r_i = w(a_i) \times (l(a_i, v_c) + t_i)$. Let r_m be the median value of all these r_i's. We do an r-feasibility test on T with $r = r_m$.

Step 6: Consider the case when the feasibility test returns feasible. Then for each pair (a_i, b_i) with $r_i \geq r_m$, we prune out the vertex a_i, since in any optimal solution, the distance of the center to a_i always dominates that to b_i. Similarly, we can prune out vertex b_j from each pair (a_j, b_j) with $r_j < r_m$ when the feasibility test returns infeasible. ◀

▶ **Time Complexity.** Finding centroid v_c takes $O(n)$ time, where n is the size of T. Finding the $(p-1, S \cup \{v_c\})$-center takes $T_{total}(p-1, n)$ time. Rest of the steps takes $O(n)$ time. Therefore total time takes by the algorithm is

$$T_{prune}(p, n) = \begin{cases} T_{total}(p-1, n) + O(n) & , p \geq 1 \\ O(n) & , p = 0 \end{cases}$$

▶ **Lemma 6.** *Algorithm 3 does not change the solution to the (p, S)-center in T.*

Proof. If B is a type-a big-component then in some optimal solution all vertices in $V(B)$ are covered by S_B. Therefore removing all vertices in $V(B)$ except v_{far} does not affect the (p, S)-center radius.

Again, if B is a type-b big-component then the vertices in V_{rest} are covered by a one new center say x. In Step 6 we prune from each pair only those vertices which can never be the farthest vertex from x. Therefore, deleting these vertices will not affect the position of x in anyway. ◀

▶ **Lemma 7.** *Algorithm 3 prunes at least $\frac{n}{16k}$ vertices from B.*

Proof. Similar to the analysis in Megiddo [11] Algorithm 3 prunes $\frac{1}{8}$-fraction of the vertices in B. Since B is a big-component, it has al least $\frac{n}{2k}$ vertices. Therefore, number of vertices pruned is at least $\frac{1}{8}\frac{n}{2k} = \frac{n}{16k}$. ◀

5 The (p, S)-center Algorithm

In this section, we present a recursive $O(n)$ time algorithm for the (p, S)-center problem in T where $k = p + |S|$ is a constant.

▶ **Algorithm 4.**
Input: Tree T, integer p and the set of centers S.
Output: An optimal solution X to the (p, S)-center problem in T.

Step 1: Let n_0 be some suitable constant. We set $T' = T$ and if $|T'| \geq n_0$ we do Step 2, otherwise we jump to Step 3.
Step 2: Find a big-component B in T' using Algorithm 2. Next we prune the vertices of B using Algorithm 3. The pruning step leads to a new tree T'' with lesser number of vertices than T'. We set $T' = T''$ and repeat this step if $|T'| \geq n_0$.
Step 3: We compute the (p, S)-center solution X in T' using standard brute force technique. From Lemma 6, X is also the solution to the (p, S)-center in the original tree T. ◀

▶ **Time Complexity.** The time taken by the Algorithm 4 is given by the following recursion

$$T_{total}(p, n) = \begin{cases} T_{big}(p, n) + T_{prune}(p, n) + T_{total}\left(p, n - \frac{n}{16k}\right) + O(n) & , n \geq n_0 \\ O(1) & , \text{otherwise} \end{cases}$$

Now,

$$\begin{aligned} T_{total}(p, n) &= T_{total}\left(p, n - \frac{n}{16k}\right) + T_{big}(p, n) + T_{prune}(p, n) + O(n) \\ &\leq T_{total}\left(p, \frac{16k-1}{16k}n\right) + T_{total}(p-1, n) + \sum_{j=1}^{k-1} T_{total}(p-1, j\frac{n}{k}) + O(kn) \\ &\leq T_{total}\left(p, \frac{16k-1}{16k}n\right) + T_{total}(p-1, n) + (k-1)T_{total}(p-1, n) + O(kn) \\ &\leq T_{total}\left(p, \frac{16k-1}{16k}n\right) + k \cdot T_{total}(p-1, n) + O(kn) \end{aligned}$$

To show that $T_{total}(p, n) \leq c2^{pk}n$ (c is a constant such that $O(kn) \leq ckn$) it suffices to show that

$$\begin{aligned} & T_{total}\left(p, \frac{16k-1}{16k}n\right) + k \cdot T_{total}(p-1, n) + O(kn) && \leq c2^{pk}n \\ \iff & c2^{pk} \cdot \frac{16k-1}{16k}n + ck2^{(p-1)k}n + ckn && \leq c2^{pk}n \\ \iff & -\frac{1}{16k} + k2^{-k} + k2^{-pk} && \leq 0 \end{aligned}$$

By numerical computation we can show this to be true for $k \geq 13$ and $1 \leq p \leq k$. This implies that the (p, S)-center problem can be solved in $O(2^{pk}n)$ time. By similar computation we can show that for $1 \leq k \leq 12$, $T_{total}(p, n) \leq O(n)$. Therefore the weighted k-center problem can be solved in $O(2^{k^2}n)$ time. For fixed k this time is linear in n.

6 Conclusion

The most efficient algorithm for the weighted k-center problem in trees for arbitrary k takes $O(n \log n)$ time and is given by Wang et al. [15]. For $k = 1$ and 2, Megiddo [11] and Ben-Moshe et al. [2] give $O(n)$ time solutions, respectively. It is not known whether a linear time algorithm exists for this problem. We settle this problem partially by presenting a linear time algorithm for any constant k. However, the question of whether an optimal linear time algorithm exists for an arbitrary k still remains open.

For the weighted k-center problem in cactus Ben-Moshe [3] presented an $O(n^2)$ time algorithm. In the same paper, they also showed that for $k = 1$ and 2, the problem can be solved in $O(n \log n)$ and $O(n \log^3 n)$ time respectively. No subquadratic time algorithm is known for the weighted k-center problem for cactus when $k > 2$.

References

1 Aritra Banik, Binay Bhattacharya, Sandip Das, Tsunehiko Kameda, and Zhao Song. The p-Center Problem in Tree Networks Revisited. In *15th Scandinavian Symposium and Workshops on Algorithm Theory*, 2016.

2 Boaz Ben-Moshe, Binay Bhattacharya, and Qiaosheng Shi. An Optimal Algorithm for the Continuous/Discrete Weighted 2-Center Problem in Trees. In José R. Correa, Alejandro Hevia, and Marcos Kiwi, editors, *LATIN 2006: Theoretical Informatics*, pages 166–177, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

3 Boaz Ben-Moshe, Binay Bhattacharya, Qiaosheng Shi, and Arie Tamir. Efficient algorithms for center problems in cactus networks. *Theoretical Computer Science*, 378(3):237–252, 2007. Algorithms and Computation.

4 Binay Bhattacharya, Sandip Das, and Tsunehiko Kameda. Linear-time fitting of a k-step function. *Discrete Applied Mathematics*, 2017. `doi:10.1016/j.dam.2017.11.005`.

5 Binay Bhattacharya and Qiaosheng Shi. Optimal Algorithms for the Weighted p-Center Problems on the Real Line for Small p. In Frank Dehne, Jörg-Rüdiger Sack, and Norbert Zeh, editors, *Algorithms and Data Structures*, pages 529–540, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

6 Richard Cole. Slowing Down Sorting Networks To Obtain Faster Sorting Algorithm. In *Foundations of Computer Science, 1984. 25th Annual Symposium on*, pages 255–260. IEEE, 1984.

7 Greg N. Frederickson. Parametric search and locating supply centers in trees. In Frank Dehne, Jörg-Rüdiger Sack, and Nicola Santoro, editors, *Algorithms and Data Structures*, pages 299–319, Berlin, Heidelberg, 1991. Springer Berlin Heidelberg.

8 M Jeger and Oded Kariv. Algorithms for finding P-centers on a weighted tree (for relatively small P). *Networks*, 15(3):381–389, 1985.

9 Oded Kariv and S Louis Hakimi. An algorithmic approach to network location problems. I: The p-centers. *SIAM Journal on Applied Mathematics*, 37(3):513–538, 1979.

10 Arindam Karmakar, Sandip Das, Subhas C Nandy, and Binay K Bhattacharya. Some variations on constrained minimum enclosing circle problem. *Journal of Combinatorial Optimization*, 25(2):176–190, 2013.

11 Nimrod Megiddo. Linear-time algorithms for linear programming in $\mathbb{R}^3$ and related problems. *SIAM journal on computing*, 12(4):759–776, 1983.

12 Nimrod Megiddo and Arie Tamir. New results on the complexity of p-centre problems. *SIAM Journal on Computing*, 12(4):751–758, 1983.

13 Edward Minieka. Conditional centers and medians of a graph. *Networks*, 10(3):265–272, 1980.

14 Qiaosheng Shi. *Efficient algorithms for network center/covering location optimization problems.* PhD thesis, School of Computing Science-Simon Fraser University, 2008.

15 Haitao Wang and Jingru Zhang. An O(n log n)-Time Algorithm for the k-Center Problem in Trees. In Bettina Speckmann and Csaba D. Tóth, editors, *34th International Symposium on Computational Geometry (SoCG 2018)*, volume 99 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 72:1–72:15, Dagstuhl, Germany, 2018.