The p-Center Problem in Tree Networks Revisited

Aritra Banik[1], Binay Bhattacharya[2], Sandip Das[3], Tsunehiko Kameda[4], and Zhao Song[5]

1 **Advanced Computing and Microelectronics Unit, Indian Statistical Inst., Kolkata, India**
`aritrabanik@gmail.com`
2 **School of Computing Science, Simon Fraser University, Vancouver, Canada**
`binay@sfu.ca`
3 **Advanced Computing and Microelectronics Unit, Indian Statistical Inst., Kolkata, India**
`sandipdas@isical.ac.in`
4 **School of Computing Science, Simon Fraser University, Vancouver, Canada**
`tikokameda@gmail.com`
5 **Department of Computer Science, University of Texas, Austin, USA**
`zhaos@utexas.edu`

Abstract

We present two improved algorithms for weighted discrete p-center problem for tree networks with n vertices. One of our proposed algorithms runs in $O(n \log n + p \log^2 n \log(n/p))$ time. For all values of p, our algorithm thus runs as fast as or faster than the most efficient $O(n \log^2 n)$ time algorithm obtained by applying Cole's speed-up technique [10] to the algorithm due to Megiddo and Tamir [20], which has remained unchallenged for nearly 30 years.

Our other algorithm, which is more practical, runs in $O(n \log n + p^2 \log^2(n/p))$ time, and when $p = O(\sqrt{n})$ it is faster than Megiddo and Tamir's $O(n \log^2 n \log \log n)$ time algorithm [20].

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases Facility location, p-center, parametric search, tree network, sorting network

Digital Object Identifier 10.4230/LIPIcs.SWAT.2016.6

1 Introduction

Deciding where to locate facilities to minimize the communication or travel costs is known as the *facility location problem.* It has attracted much research interest since the publication of the seminal paper on this topic by Hakimi [14]. For a good review of this subject, the reader is referred to [15]. It can be applied to locate fire stations, distribution centers, etc.

In the *p-center problem*, p centers are to be located in a network $G(V, E)$, so that the maximum (weighted) distance from any demand point to its nearest center is minimized. The simplest version of the problem (V/V/p) allows centers to be located only on vertices (V), and restricts demand points to be vertices. Other variations allow points on edges to be demand points (V/E/p), or points on edges (E) to be centers (E/V/p), or both (E/E/p). The vertices of a network could be weighted, i.e., the vertex weights can be different, or unweighted. In this paper we refer to weighted E/V/p as the *weighted discrete* p-center problem (*WDpC*). The p-center problem in a general network is NP-hard [17]. In this paper, we focus on the tree networks, on which there has been very little progress (for arbitrary p) since the mid-1980s.

1.1 Previous work

Megiddo [19] solved E/V/1 for the tree networks in $O(n)$ time, where n is the number of vertices. Megiddo and Tamir also studied this problem [20]. Kariv and Hakimi [17] presented an $O(m^p n^{2p-1} \log n/(p-1)!)$ time algorithm for WDpC in a general network, where m is the number of edges. Tamir [25] improved the above bound to $O(m^p n^p \log n\alpha'(n))$, where $\alpha'(n)$ is the inverse of Ackerman's function. Recently, Bhattacharya and Shi [7] improved it to $O(m^p n^{p/2} 2^{\log^* n} \log n)$ for $p \geq 3$, where $\log^* n$ denotes the iterated logarithm of n. A recent result on Klee's measure due to Chan [8] implies that this bound can be further improved to $O(m^p n^{p/2} \log n)$.

Frederickson [11, 12] solved the unweighted V/V/p, E/V/p and V/E/p problems in $O(n)$ time, independently of p. For the weighted tree networks, linear time algorithms have been proposed in the case where p is a constant [3, 24]. For arbitrary p, Kariv and Hakimi [17] gave an exhaustive $O(n^2 \log n)$ time algorithm. Megiddo's linear time feasibility test [21] can be parameterized to solve the problem in $O(n^2)$ time, using the idea introduced in [21]. Megiddo and Tamir [20] then provided an $O(n \log^2 n \log\log n)$ time algorithm, which can be made to run in $O(n \log^2 n)$ time using the AKS or similar $n \times O(\log n)$ sorting networks [1, 13, 23], together with Cole's improvement [10]. The $O(pn \log n)$ time algorithm due to Jeger and Kariv [16] is faster than all others if $p = o(\log n)$.

The running time of the algorithm of Megiddo and Tamir [20] is dominated by the time for computing the distance queries in their binary-search based algorithm. Frederickson [11, 12] used parametric search to design optimal algorithms for the unweighted p-center problem in tree networks. In parametric search, one first designs an *α-feasibility* test to see if p centers can be placed in such a way that every vertex is within *cost* (=distance weighted by the weight of the vertex) α from some center. In general, a set of candidate values for α is explicitly or implicitly tested as the algorithm progresses. Eventually, the search will settle on the smallest α value, α^*. The ideas presented in [11, 12] are for the unweighted case only, and therefore cannot be extended easily to WDpC. The question of whether an algorithm which runs faster than $O(n \log^2 n)$ time is possible for the tree networks has been open for a long time since.

To present our basic approach clearly, we first solve WDpC for balanced binary tree networks. We then generalize it to general (unbalanced) tree networks based on *spine tree decomposition* [4, 5].

1.2 Our contributions

Our major contributions in this paper are (i) an $O(p \log(n/p))$ time algorithm for testing α-feasibility for an arbitrary α, with preprocessing that requires $O(n \log n)$ time, (ii) a practical $O(n \log n + p^2 \log^2(n/p))$ time WDpC algorithm, which outperforms the $O(n \log^2 n \log\log n)$ time algorithm proposed in [20] when $p = O(\sqrt{n})$, and iii) an $O(n \log n + p \log^2 n \log(n/p))$ time WDpC algorithm based on AKS-like sorting networks [1, 13, 23], which improves upon the currently best $O(n \log^2 n)$ time algorithm [10, 20].

The rest of the paper is organized as follows. In Section 2 we first define the terms that are used throughout the paper. We then give a rough sketch of our parametric search approach to solving WDpC on balanced tree networks. We also propose our location policy that guides the placement of the centers. Section 3 describes preprocessing that we perform, in particular, the computation of upper envelopes and a preparation for fractional cascading. We then present in Section 4 the details of the feasibility test part of parametric search for balanced tree networks. The optimization part of parametric search is discussed in detail in

Section 5 for balanced tree networks. At the end of the section, we present our results for the general (unbalanced) tree networks.

2 Preliminaries

2.1 Definitions

Let $T = (V, E)$ denote a tree network, where each vertex $v \in V$ has weight $w(v)$ (≥ 0) and each edge $e \in E$ has a non-negative length. We write $x \in T$, if point x lies anywhere in T, be it on an edge or at a vertex. For $a, b \in T$, let $\pi(a, b)$ denote the unique path from a to b, and $d(a, b)$ its length. If a or b is on an edge, its prorated length is used. If T is a binary rooted with root vertex r, for any vertex $v \in V$, the subtree rooted at v is denoted by $T(v)$, and the parent of v $(\neq r)$ is denoted by $p(v)$.

For a non-leaf vertex $v \in V$, let v_l (resp. v_r) denote its left (resp. right) child vertex, and define the left (resp. right) *branch* of v by $B(v_l) = T(v_l) \cup (v_l, v)$ (resp. $B(v_r) = T(v_r) \cup (v_r, v)$). We thus have $T(v) = B(v_l) \cup B(v_r)$, and the root of $B(v_l)$ (resp. $B(v_r)$) is v with degree 1 in $B(v_l)$ (resp. $B(v_r)$).

Let $V' \subseteq V$ and $x \in T$. We define the distance between a point x and V' by $d(x, V') \triangleq \min_{v \in V'}\{d(x, v)\}$. The *cost* of a vertex v at point x is given by $d(v, x)w(v)$. We say that point $x \in T$ *α-covers* V' $(\subseteq V)$ if $\max_{v \in V'}\{d(x, v)w(v)\} \leq \alpha$. If α is clear from the context, we may simply say that x *covers* V'. A problem instance is said to be *α-feasible* if there exists p centers such that every vertex is α-covered by at least one of the centers. Those p centers are said to form a *p-center* [17]. For a vertex $v \in V$ and points $x \in T \backslash T(v)$, we define the *upper envelope*

$$E_v(x) = \max_{u \in T(v)} \{d(x, u)w(u)\}. \tag{1}$$

If $E_v(x) = d(u, x)w(u) = \alpha$, then vertex u is said to be an *α-critical vertex* in $T(v)$ with respect to $x \in T \backslash T(v)$, and is denoted by $u = cv(x, T(v))$. If α is clear from the context, we may call it just a critical vertex

2.2 Spine tree decomposition and upper envelopes

We give a brief review of *spine tree decomposition* [4, 5]. The materials in this subsection is not needed until Sec. 5.2. We can assume that given T is a binary tree; otherwise we can introduce $O(n)$ vertices of 0 weight and $O(n)$ edges of 0 length to make it binary. Thus each vertex has degree at most 3. Let r be the root of T, which can be chosen arbitrarily. Traverse T, starting on an edge incident to r. At each vertex visited, move to the branch that contains the largest number of leaf vertices, breaking a tie arbitrarily. When a leaf vertex, u, is reached, the path $\pi(v, u)$ is generated, and it is called the *top spine*, denoted by σ_1. We then repeat a similar traversal from each vertex on the generated spine, to generate other spines, until every vertex of T belongs to some spine.

Let $STD(T)$ denote the tree constructed by the spine tree decomposition of tree T, together with the *search tree* τ_{σ_l} for each *spine* σ_l, whose root is denoted by ρ_l [4, 5]. Fig. 1 illustrates a typical structure of spine σ_l and its search tree τ_{σ_l}. The horizontal line represents spine σ_l, and we name the vertices on it $v_1, v_2, \ldots$ from left to right.

The triangles represent subtrees hanging from σ_l. If a hanging subtree t is connected to vertex $v_i \in \sigma_l$, then we call the subgraph consisting of t, v_i, and the edge connecting them a *branch* of σ_l and denote it by B_i. Since we assume that the vertices of T have degree at most 3, there is at most one branch hanging from any vertex on the spine.

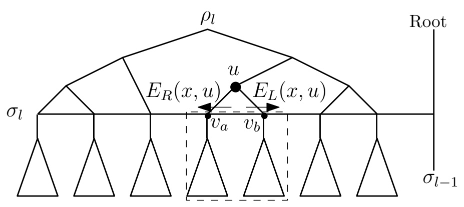

Figure 1 Search tree τ_{σ_l} for spine σ_l. $v_a = v_L(u)$ and $v_b = v_R(u)$.

For a node[1] u in τ_{σ_l}, let $v_L(u)$ (resp. $v_R(u)$) denotes the leftmost (resp. rightmost)[2] vertex on σ_l that belongs to the subtree $\tau_{\sigma_l}(u)$. We introduce upper envelope $E_L(x,u)$ (resp. $E_R(x,u)$) for the costs of the vertices in the branches of σ_l that belong to $\tau_{\sigma_l}(u)$, for point x that lies to the right (resp. left) of vertex $v_R(u)$ (resp. $v_L(u)$). See Fig. 1. Since $E_L(x,u)$ and $E_R(x,u)$ are upper envelopes of linear functions, they are piecewise linear. For each node u of $STD(T)$ we compute $E_L(x,u)$ and $E_R(x,u)$, and store them at u as sequences of bending points (their x and y coordinates). These upper envelopes can be computed in $O(n \log n)$ time by the following lemma.

▶ **Lemma 1** ([4, 5]). *The path from any leaf to the root of $STD(T)$ has $O(\log n)$ nodes on it.*

2.3 Our approach

Except in the last subsection of the paper, we assume that the given tree T is balanced with respect to its root r, so that its height is $O(\log n)$. If not, we can use spine tree decomposition that transforms T in linear time to a structure that has most of the properties of a balanced binary tree. Working on a balanced binary tree network also helps us to explain the essence of our approach, without getting bogged down in details. Our algorithms consist of a lower part and an upper part. In the lower part, we test α-feasibility for a given cost α, and in the upper part we carry out Megiddo's *parametric search* [18]. To perform a feasibility test, we first identify the *α-peripheral centers*, below which no center needs be placed. Once all the q $(< p)$ α-peripheral centers are identified, we place $p - q$ additional centers to α-cover the vertices that are not covered by the α-peripheral centers. If no more than p centers are used to α-cover the entire tree T, then the α-feasibility test is successful. Theorem 8 shows that, using fractional cascading, α-feasibility can be tested in $O(p \log(n/p))$ time after preprocessing, which takes $O(n \log n)$ time.

The second part of parametric search finds the smallest α value, α^*. We work on T bottom-up, doing essentially the same thing as in the first part. Whenever α is used in the first part, we need to invoke an α-feasibility test [18]. At each level of T, we need to invoke α-feasibility tests $O(l)$ times at level l. Therefore the total number of invocations is $O(\log^2 n)$, and the total time is $O(p \log^2 n \log(n/p))$ after preprocessing, yielding one of our main results stated in Theorem 10.

2.4 Center location policy

Suppose that we want to place a center c_i in a tree network T to α-cover a subset V_i of vertices that are connected. We propose the following location policy.

[1] A 'node' is more general than a vertex of T. A vertex is also a node, because it belongs to τ_{σ_l}, but not every node is a vertex.
[2] Right (resp. left) means towards (resp. away from) the parent spine σ_{l-1} of σ_l.

Root-centric policy: *Place c_i at the point that α-covers all the vertices in V_i and is closest to root r of T.*

It is easy to prove the following lemmas.

▶ **Lemma 2.** *If a set of p centers α-covers all the vertices in V, then there is a partition of vertex set $\{V_i \mid i = 1, \ldots, p\}$, where each V_i is the vertex set of a connected part of T, such that the root-centric location policy locates each center c_i that α-covers V_i.*

▶ **Lemma 3.** *Let $\{c_i \mid i = 1, \ldots, p\}$ be p centers obeying the root-centric policy that together α-cover V. For each center c_i, find a vertex $v \in V_i$ with maximum cost $d(v, c_i)w(v)$ that is the farthest from the root, and name it g_i. Then it satisfies $c_i \in \pi(g_i, r)$.*

Proof. If $c_i \notin \pi(g_i, r)$, then c_i could move closer to r, a contradiction. ◀

3 Preprocessing

3.1 Upper envelopes

According to our definition of upper envelope $E_v(x)$ for subtree $T(v)$ (see (1)), if v is a leaf vertex, we have

$$E_v(x) = d(x, v)w(v), \tag{2}$$

for any $x \in T$. Let v_l (resp. v_r) be the left (resp. right) child vertex of a non-leaf vertex $v \in V$. Then for any $x \in T \setminus T(v)$, we have

$$E_v(x) = \max\{E_{v_l}(x), E_{v_r}(x), d(x, v)w(v)\}. \tag{3}$$

Function $E_v(x)$ is piecewise linear in $x \in \pi(v, r)$ and can be represented by a sequence of bending points. In the sequence representing $E_v(x)$, in addition to the values of $E_v(x)$ at the bending points, we insert the values of $E_v(x)$ evaluated at all the $O(\log n)$ vertices on $\pi(v, r)$.[3]

▶ **Lemma 4.** *If T is balanced, then $\{E_v(x) \mid v \in V, x \in \pi(v, r)\}$ can be computed bottom-up in $O(n \log n)$ time and $O(n \log n)$ space.*

In the rest of this paper we assume that the given tree T is a balanced binary tree. If not we can use spine tree decomposition [4, 5, 6], which shares many useful properties of a balanced tree.

3.2 Fractional cascading

From now on we assume that we have the bending points of $\{E_v(x) \mid v \in V, x \in \pi(v, r)\}$ at our disposal. The second task of preprocessing is to merge the bending points of $\{E_v(x) \mid v \in V, x \in \pi(v, r)\}$ to prepare for fractional cascading [9]. Again we do this bottom up, merge-sorting the two sequences of bending points into one at each vertex. Since each vertex causes at most $O(\log n)$ bending points in $\{E_v(x) \mid v \in V, x \in \pi(v, r)\}$, the total number of bending points is $O(n \log n)$.

[3] We mix those values among the bending points, so that we know on which edges the bending points lie.

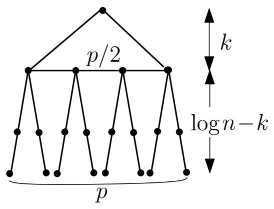

Figure 2 Illustration for the proof of Lemma 5.

4 α-Feasibility

4.1 Peripheral centers

As a result of preprocessing, we have the upper envelopes $\{E_v(x) \mid v \in V, x \in \pi(v,r)\}$. To find the peripheral centers, α-peripheral we carry out *truncated* pre-order DFS (depth-first-search), looking for the vertex-point pairs (v,x) satisfying $E_v(x) = \alpha$, which means v is an α-critical vertex in $T(v)$ with respect to x.

▶ **Procedure 1.** *`Find-Peripheral-Centers`* (α)

Perform pre-order DFS, modified as follows, where v is the vertex being visited.

1. *If $\exists x \in (v, p(v))$ such that $E_v(x) = \alpha$, return x as an α-peripheral center,*[4] *and backtrack.*
2. *If $p+1$ α-peripheral centers have been found, then return* `Infeasible` *and stop.* ◀

To carry out Step 1 efficiently, we perform binary search with key α in the merged sequence of bending points (of the upper envelopes) stored at the root r, and follow the relevant pointers based on fractional cascading.

▶ **Lemma 5.** *Procedure* `Find-Peripheral-Centers`(α) *visits $O(p\log(n/p))$ vertices.*

Proof. The number of vertices that Procedure `Find-Peripheral-Centers`(α) visits is the largest when the α-peripheral centers are as low as possible and they separate from each other as high as possible. This extreme case is illustrated in Fig. 2, where $p = 2^k - 1$ for some integer k.

The total number of edges that are traversed is given by

$$O(p(\log n - k) + p) = O(p(\log n - \log p) + p) = O(p\log(n/p)),$$

where the second term, p, is an upper bound on the number of vertices at depth k or shallower. ◀

▶ **Lemma 6.** *If $\{E_v(x) \mid v \in V\}$ are available, all the α-peripheral centers can be found in $O(p\log(n/p))$ time.*

Proof. If fractional cascading is used in Step 1 of Procedure 1, it runs in amortized constant time per vertex. The rest follows from Lemma 5. ◀

[4] We assume that the trivial case, where one center at root r α-covers the entire tree, is dealt with specially, which is straightforward.

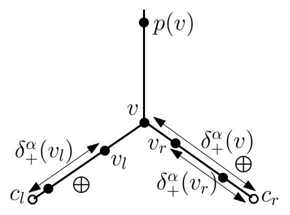

Figure 3 (Left) $c_l \in B(v_l)$ and $c_r \in B(v_r)$; (Middle) A center is needed within $\delta_-^\alpha(u)$ from u; (Right) $c_l \in B(v_l)$.

4.2 α-Feasibility test

Given an α value, suppose that we have found q $(< p)$ α-peripheral centers, following the root-centric location policy. We replace each α-peripheral center by a *dummy vertex*, and define the *trimmed tree* $T'_\alpha = (V'_\alpha, E'_\alpha)$. Its vertex set V'_α consists of two types of vertices: the *first type* is a vertex that lies on the path between a dummy vertex and root r, inclusive. If any such vertex has only one child vertex among them, then the other child vertex of T (called a vertex of the *second type*) is kept in T' to represent the α-critical vertex in the subtree of T rooted at that vertex. In what follows, we use T' instead of T'_α for simplicity, since the implied α will be clear from the context. It is easy to see that tree T' contains $O(q \log n)$ vertices. Without loss of generality, we consider each vertex of the second type as the right child of its parent.

Let u be a vertex of the second type. Then we must have visited u during the execution of `Find-Peripheral-Centers`(α), and no α-peripheral center was placed in subtree $T(u)$. At the time of this visit, we identified the α-critical vertex in $T(u)$, which implies that we can store this α-critical vertex at u as a by-product of `Find-Peripheral-Centers`(α) at no extra cost.

Later, we will be introducing more centers, in addition to α-peripheral centers, working on the trimmed tree T' bottom up. For each vertex in T', its subtrees can be one of the following types:

- $\ominus$-subtree: The centers in it, if any, do not α-cover all the vertices in the subtree.
- $\oplus$-subtree: The centers in it α-cover all the vertices in the subtree, and possibly outside it.

If $T'(v)$ is a $\oplus$-subtree, let $\delta_+^\alpha(v)$ denote the distance from v to the highest center in $T'(v)$ at or below v. See the leftmost figure of Fig. 3, where v_l (resp. v_r) is the left (resp. right) child vertex of v, and c_l (resp. c_r) is the highest center placed in $T'(v_l)$ (resp. $T'(v_r)$).

If $T'(v)$ is a $\ominus$-subtree, on the other hand, let $\delta_-^\alpha(v)$ denote the minimum distance from v to a point above $T'(v)$ within which a center must be placed to α-cover the uncovered vertices in $T'(v)$. See the middle figure in Fig. 3.

Let us discuss how to process the trimmed tree T', to introduce additional centers closer to the root in order to α-cover more vertices. We perform post-order DFS on T', always visiting the left child of a vertex first. Assume that we explored $T'(v_l)$ first and then $T'(v_r)$, and we are just back to v, and that $\delta_-^\alpha(v_l)$ or $\delta_+^\alpha(v_l)$ (resp. $\delta_-^\alpha(v_r)$ or $\delta_+^\alpha(v_r)$) are available at vertex v_l (resp. v_r). For each dummy leaf vertex v of T', we have $\delta_+^\alpha(v) = 0$. At each vertex v visited, we have one of the following three cases.

(a) [Both are $\oplus$-subtrees]. In the leftmost figure of Fig. 3, c_l (resp. c_r) is the highest center in $T'(v_l)$ (resp. $T'(v_r)$). We compute

$$\delta = \min\{\delta_+^\alpha(v_l) + d(v,v_l), \delta_+^\alpha(v_r) + d(v,v_r)\}, \tag{4}$$

which is the distance from v to the nearest center in $T'(v)$. If $\delta \cdot w(v) \leq \alpha$, then v is α-covered by c_l or c_r. Otherwise (i.e., even the center in $T'(v)$ that is nearer to v cannot α-cover v) v must be covered by a center placed above v, and $T'(v)$ $(= \{v\})$ now becomes a $\ominus$-subtree of $p(v)$.

(b) [Both are $\ominus$-subtrees]. See the middle figure of Fig. 3. If $\delta_-^\alpha(v_l) < d(v,v_l)$, for example, we need to place a center c_l on the edge (v,v_l), and $T'(v)$ now becomes a $\oplus$-subtree, provided v is α-covered by c_l. If both c_l and c_r are placed this way, we set $\delta_+^\alpha(v) = \min\{d(c_l,v), d(c_r,v)\}$, provided one of them α-covers v. If no center needs to be placed on (v,v_l) or (v,v_r), then we compute

$$\delta = \min\{\delta_-^\alpha(v_l) - d(v,v_l), \delta_-^\alpha(v_r) - d(v,v_r)\}. \tag{5}$$

We need a center within $\min\{\delta, \alpha/w(v)\}$ above v. These are some of the typical cases, which illustrate kinds of necessary operations. Procedure `Merge`$(v;\alpha,T)$, given below, deals with the other cases as well, not mentioned here, exhaustively.

(c) [One is a $\ominus$-subtree and the other is a $\oplus$-subtree]. We assume without loss of generality that the left (resp. right) subtree is a $\oplus$-subtree (resp. $\ominus$-subtree), as shown in the rightmost figure of Fig. 3, and c_l is the highest center in $T'(v_l)$. As in Case (b), we first test if $\delta_-^\alpha(v_r) < d(v,v_r)$, and if so place a center c_r on edge (v,v_r). Then we have case (a). Otherwise, we need to test if c_l α-covers the uncovered vertices in $T'(v_r)$ as well as v. If not, they must be covered by a new center above v.

We now present a formal procedure that deals with all possible cases. We will use it for $T = T'$.

▶ **Procedure 2.** *Merge*$(v;\alpha,T)$

Case (a): $[T(v_l)=\oplus, T(v_r)=\oplus]$ *Compute δ using (4). If $\delta \cdot w(v) \leq \alpha$, then set $\delta_+^\alpha(v) = \delta$. Otherwise, make $T(v)$ a $\ominus$-subtree of $p(v)$ with $\delta_-^\alpha(v) = \alpha/w(v)$.*

Case (b): $[T(v_l)=\ominus, T(v_r)=\ominus]$ *If $\delta_-^\alpha(v_l) < d(v,v_l)$ (resp. $\delta_-^\alpha(v_r) < d(v,v_r)$), place a center c_l (resp. c_r) on the edge (v,v_l), (resp. (v,v_r)) at distance $\delta_-^\alpha(v_l)$ from v_l (resp. $\delta_-^\alpha(v_r)$ from v_r). If c_l and/or c_r α-covers v, then make $T(v)$ a $\oplus$-subtree of $p(v)$ with $\delta_+^\alpha(v) = \min\{d(c_l,v), d(c_r,v)\}$, where $d(c_l,v) = 0$ (resp. $d(c_r,v) = 0$) if c_l (resp. c_r) is not introduced. If neither of them covers v, then make $T(v)$ a $\ominus$-subtree of $p(v)$ with $\delta_-^\alpha(v) = \alpha/w(v)$. If neither c_l nor c_r is introduced, then compute δ using (5) and make $T(v)$ a $\ominus$-subtree of $p(v)$ with $\delta_-^\alpha(v) = \min\{\delta, \alpha/w(v)\}$.*

Case (c): $[T(v_l)=\oplus, T(v_r)=\ominus]$[5] *If $\delta_-^\alpha(v_r) < d(v,v_r)$, then place a center c_r on edge (v,v_r) at distance $\delta_-^\alpha(v_r)$ from v_r, set $\delta_+^\alpha(c_r) = d(v,c_r) = d(v,v_r) - \delta_-^\alpha(v_r)$, and go to Case (a). Otherwise,*

[5] The case $[T(v_l)=\ominus, T(v_r)=\oplus]$ is symmetric.

(i) *If c_l covers v (i.e., $\{\delta_+^\alpha(v_l)+d(v_l,v))\}w(v) \le \alpha$), and c_l also covers $T(v_r)$ (i.e., $\delta_+^\alpha(v_l)+d(v_l,v_r) \le \delta_-^\alpha(v_r)$), then let $\delta_+^\alpha(v)=\delta_+^\alpha(v_l)+d(v_l,v)$.*
(ii) *In all the remaining cases, set $\delta_-^\alpha(v)=\min\{\delta_-^\alpha(v_r)-d(v,v_r),\alpha/w(v)\}$.* ◀

It is easy to show that

▶ **Lemma 7.** *After preprocessing,* `Merge-I`$(v;\alpha,T)$ *runs in constant time.*

We now formally state our algorithm for testing α-feasibility.

▶ **Algorithm 1.** `Feasibility-Test` (α, T)
1. *Call* `Find-Peripheral-Centers`(α).
2. *Construct the trimmed tree T', consisting of the vertices of the first type and those of the second type and the edges connecting them. For each vertex u of the second type, compute the α-critical vertex for $T'(u)$.*
3. *Perform a post-order depth-first traversal on T', invoking* `Merge`$(v;\alpha,T')$ *on each vertex v visited.*
4. *If a set of no more than p centers covering T has been found, then return* `Feasible` *and stop. If the p centers found so far do not totally cover T, then return* `Infeasible` *and stop.* ◀

▶ **Theorem 8.** *For a balanced tree network,* `Feasibility-Test`(α,T) *runs in $O(p\log(n/p))$ time, excluding the preprocessing time.*

Proof. Step 1 runs in $O(p\log(n/p))$ time by Lemma 6. Step 2 can be carried out at the same time as Step 1 in $O(p\log(n/p))$ time. Step 3 also runs in $O(p\log(n/p))$ time by Lemma 7. Lastly, Step 4 takes constant time. ◀

5 Optimization

We will employ Megiddo's parametric search [18], using the α-feasibility test we developed in Sec. 4.2. We maintain a lower bound $\underline{\alpha}$ and an upper bound $\overline{\alpha}$ on α^*, where $\underline{\alpha} < \alpha^* \le \overline{\alpha}$. Eventually we will end up with $\alpha^* = \overline{\alpha}$. If we succeed (resp. fail) in an α-feasibility test, then it means that $\alpha \ge \alpha^*$ (resp. $\alpha < \alpha^*$), so we update $\overline{\alpha}$ (resp, $\underline{\alpha}$) to α.

5.1 Balanced tree networks

Based on Theorem 8, the main theorem in [18] implies:

▶ **Theorem 9.** *WDpC for the balanced tree networks with n vertices can be solved in $O(n\log n + p^2\log^2(n/p))$ time.*

We propose another algorithm which performs better than the first algorithm referred to in the above theorem for some range of values of p. For this algorithm we will show later that we need to test feasibility $O(\log^2 n)$ times. This fact, together with Theorem 8, leads to the following theorem.

▶ **Theorem 10.** *WDpC for the balanced tree networks with n vertices can also be solved in $O(n\log n + p\log^2 n\log(n/p))$ time.*

In the rest of this subsection we prove Theorem 10. Let $l=1,2,\ldots,k$ be the *levels* of T from top to bottom, where the root r is at level 1 and the leaves are at level $k=\lceil\log n\rceil = O(\log n)$. At each vertex, we need to perform a few feasibility tests. Since there are 2^{l-1}

vertices at level l of T, using prune and search, we can know the results of the feasibility tests at all the vertices of level l after actually performing only $O(\log(2^{l-1})) = O(l)$ feasibility tests. The total for all levels is thus $O(\sum_{l=1}^{\log n} l) = O(\log^2 n)$, as claimed above.

It is easy to prove the following lemma.

▶ **Lemma 11.** *Let $v_a, v_b \in V$.*

(a) [17] *Vertices v_a and v_b have the equal cost*

$$\alpha(v_a, v_b) = \frac{d(v_a, v_b)w(v_a)w(v_b)}{w(v_a) + w(v_b)} \tag{6}$$

at a point $c(v_a, v_b) \in \pi(v_a, v_b)$.

(b) *Let $v_a, v_b \in T(v)$, and suppose that $w(v_a) \neq w(v_b)$, and let $w(v_a) < w(v_b)$ without loss of generality. If $d(v_a, v)w(v_a) \geq d(v_b, v)w(v_b)$ holds, then vertices v_a and v_b have the equal cost*

$$\alpha'(v_a, v_b) = \frac{\{d(v_a, v) - d(v_b, v)\}w(v_a)w(v_b)}{w(v_b) - w(v_a)}, \tag{7}$$

at a point $c'(v_a, v_b) \in \pi(v, r)$. If $d(v_a, v)w(v_a) < d(v_b, v)w(v_b)$, then vertex v_b has a higher cost than v_a at all points on $\pi(v, r)$.[6]

If we let $v_b = v$ in Case (b) in the above lemma, v_a and v have the equal cost

$$\alpha'(v_a, v) = \frac{d(v_a, v)w(v_a)w(v)}{w(v) - w(v_a)}, \tag{8}$$

at a point $c'(v_a, v_b) \in \pi(v, r)$.

We now need to modify the definition of the critical vertex given in Sec. 2.1. With respect to $x \in T \setminus T(v)$, we are interested in the vertex $u \in T(v)$, such that $\alpha(x, u)$ is maximum, We call such a u the *critical vertex* with respect to x and denote it by γ_v. The main difference of the optimization part from the feasibility test part is that we cannot find the exact locations of the centers until the very end. However, making use of critical vertices, it is possible to identify the component of T that is to be α^*-covered by each new center. So, we will isolate/detach them one by one from T, and repeat the process.

Let v_l and v_r be the two child vertices of a vertex v at level l. When we visit v, moving up T, we need to either isolate a subtree to be covered by a center that lies below v, or determine the critical vertex in $T(v)$ to be carried higher. Whenever the result of an α-feasibility test shows that $\alpha \geq \alpha^*$, we update $\overline{\alpha}$ and assume that $\overline{\alpha} > \alpha^*$ holds, and introduce a new center (without an exact location), as necessary. This assumption will be justified if $\overline{\alpha}$ is updated later. If $\overline{\alpha}$ is never updated thereafter,[7] it implies that $\overline{\alpha} = \alpha^*$. See Lemma 12.

Based on (6), if

$$\alpha(v, \gamma_{v_l}) \geq \alpha^* \text{ (resp. } \alpha(v, \gamma_{v_r}) \geq \alpha^*), \tag{9}$$

we assume that $\alpha(v, \gamma_{v_l}) > \alpha^*$ (resp. $\alpha(v, \gamma_{v_r}) > \alpha^*$), and cut the edge (v, v_l) (resp. (v, v_r)) to detach a new component below v to be covered by the new center placed in it.[8] We need

[6] In this case, the equal cost point lies on $\pi(v, v_b)$.
[7] $\underline{\alpha}$ may be updated.
[8] Note that if $\alpha(v, \gamma_{v_l}) = \alpha^*$, for example, we cannot isolate a component.

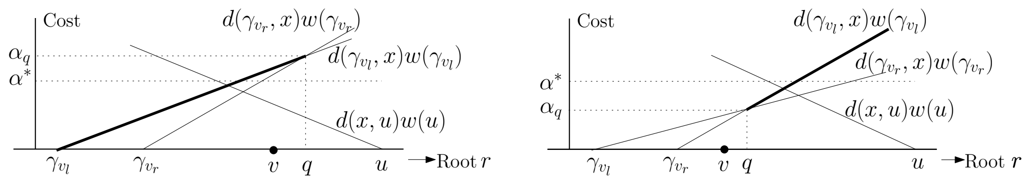

Figure 4 The cost lines of $\gamma_{v_l} \in B(v_l)$ and $\gamma_{v_r} \in B(v_r)$ intersect at q above v: (Left) Cost α_q at intersection q is higher than α^* ($\alpha_q > \alpha^*$); (Right) $\alpha_q < \alpha^*$.

not know the exact position of the new center. If two new centers are introduced this way, vertex v must be α^*-covered by a center placed above v, and v becomes a (tentative) critical vertex for $T(v)$ with respect to x above v. If only one of the inequalities in (9) holds and only (v, v_l) (resp. (v, v_r)) is cut, then either v or γ_{v_r} (resp. γ_{v_l}) becomes a critical vertex for $T(v)$, based on the outcome of $\alpha'(v_a, v)$-feasibility test. See (8).

Consider the remaining case, where neither inequality in (9) holds. We need to determine a critical vertex in $T(v)$ with respect to x above v.

To this end, we first find the intersection $q = c'(\gamma_{v_l}, \gamma_{v_r}) \in \pi[v, r]$ of the two cost lines $d(\gamma_{v_l}, x)w(\gamma_{v_l})$ and $d(\gamma_{v_r}, x)w(\gamma_{v_r})$, and its cost $\alpha_q = \alpha'(\gamma_{v_l}, \gamma_{v_r})$, assuming the condition for (7) is met. We then test α_q-feasibility. If $\alpha_q \geq \alpha^*$, as in the left figure of Fig. 4, then we set $\gamma'_v = \gamma_{v_l}$ (resp. $\gamma_v = \gamma_{v_r}$) if $w(\gamma_{v_l}) \leq w(\gamma_{v_r})$ (resp. $w(\gamma_{v_l}) > w(\gamma_{v_r})$). If $\alpha_q < \alpha^*$, on the other hand, as in the right figure of Fig. 4, then we set $\gamma'_v = \gamma_{v_r}$ (resp. $\gamma_v = \gamma_{v_l}$) if $w(\gamma_{v_l}) \leq w(\gamma_{v_r})$ (resp. $w(\gamma_{v_r}) < w(\gamma_{v_l})$). In order to find the true critical vertex γ_v in place of γ'_v, we need to take v into consideration as well. This time we use $\alpha'(v_a, v)$ of (8) instead of (7). In the future we will be testing vertices $u \notin T(v)$ to see if the cost of the intersection between $d(x, u)w(u)$ and $d(\gamma_v, x)w(\gamma_v)$ is lower than α^* or not. We must choose the critical vertex that gives the highest cost near α^*, which is indicated by a thick line segment in Fig. 4.

In any case, we need to perform a constant number of feasibility tests per vertex visited. Whenever an α-feasibility test in (9) succeeds (resp. fails), we update $\overline{\alpha}$ (resp. $\underline{\alpha}$) to α.

▶ **Lemma 12.** *The optimal cost α^* equals $\overline{\alpha}$ at the end of the above steps.*

Proof. It was shown by Kariv and Hakimi [17] that α^* has the value $d(u,v)/(1/w(u)+1/w(v))$ for some pair of vertices u and v. See Lemma 11(a). Assume that $\alpha^* < \overline{\alpha}$ and there is a pair of vertices u and v in the same partition $V_i \subset V$ (Lemma 2) such that $\alpha^* = d(u,v)/(1/w(u) + 1/w(v))$, but we haven't tested them, a contradiction. ◀

5.2 General tree networks

We use spine tree decomposition (STD), reviewed in Sec. 2.2, for general (unbalanced) tree networks. The counterparts to Theorems 8 and 9 hold with the same complexities.

▶ **Theorem 13.**
(a) *We can test α-feasibility in $O(p \log(n/p))$ time, excluding the preprocessing, which takes $O(n \log n)$ time.*
(b) *WDpC for general tree networks with n vertices can be solved in $O(n \log n + p^2 \log^2(n/p))$ time.*

Proof. Part (a) can be proved in essentially the same way as we proved Theorem 8 in Sec. 4.2. Instead of working directly on the given tree T, we first construct $STD(T)$ and compute upper envelopes at its nodes. The concepts of the $\ominus$-subtree and $\oplus$-subtree can be carried

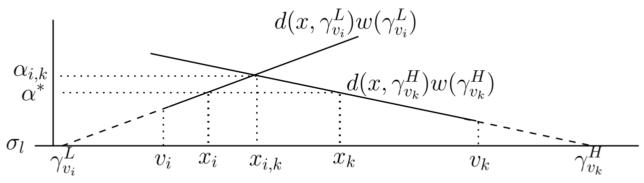

Figure 5 B_i and B_k are each a $\ominus$-branch.

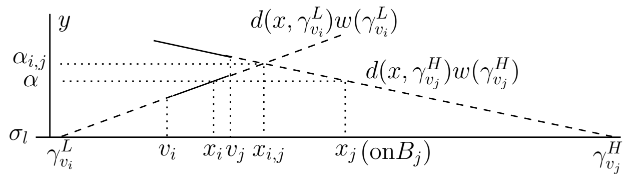

Figure 6 Point x_j is the mapped image onto σ_l of the highest center in B_j.

over to $STD(T)$. One complication is that we need to work on a group of $\ominus$-branches, instead of single $\ominus$-subtrees, but we can process them in the same order of time as in the balanced tree case. Part (b) is implied by part (a) by the main theorem in Megiddo [18]. ◀

As for the counterpart to Theorem 10, we need to use AKS-like sorting networks [1, 13, 22, 23], as in [10].

▶ **Theorem 14.** *WDpC for the general tree networks with n vertices can be solved in $O(n \log n + p \log^2 n \log(n/p))$ time.*

Proof (Informal). Let us first analyze how many times we need to perform feasibility tests when $STD(T)$ is used for a non-balanced tree network. Let n_l be the number of vertices in the spines at level l, so that we have $\sum_{l=1}^{\lambda} n_l = n$, where λ is the number of levels in $STD(T)$. We now consider one particular spine σ_l at level l. Let v_i and v_k be two vertices on σ_l, from which branches B_i and B_k hang. Assume first that both B_i and B_k are $\ominus$-branches, and let γ_{v_i} (resp. γ_{v_k}) be the α^*-critical vertices in B_i (resp. B_k). If γ_{v_i} is at distance d_i from v_i, then we map it onto σ_l at distance d_i from v_i.

There can be up to two such positions on σ_l (or its extension if it is not long enough), and we call the lower (resp. higher)[9] one $\gamma^L_{v_i}$ (resp. $\gamma^H_{v_i}$). Fig. 5 illustrates $\gamma^L_{v_i}$ and $\gamma^H_{v_k}$. In this figure each cost function $d(x,\gamma^L_{v_i})w(\gamma^L_{v_i})$ is represented by a solid and a dashed line, where the solid (resp. dashed) part shows its value on σ_l (in B_i). Similarly for the cost function $d(x,\gamma^H_{v_k})w(\gamma^H_{v_k})$. In this figure, they meet at $x_{i,k}$ on σ_l, and at this point the cost is $\alpha_{i,k} > \alpha^*$. This implies that $x_i \prec x_k$, where x_i (resp. x_k) is the point on σ_l where the cost of $\gamma^L_{v_i}$ (resp. $\gamma^H_{v_k}$) is α^*. This in turn means that a single center cannot α^*-cover both $\gamma^L_{v_i}$ and $\gamma^H_{v_k}$. If we had $\alpha_{i,k} \leq \alpha^*$, then a center would cover both of them.

Consider next the case where B_i is a $\ominus$-branch and B_j is a $\oplus$-branch, as shown in Fig. 6. In this case, the dashed part of the cost function $d(x,\gamma^H_{v_j})w(\gamma^H_{v_j})$ takes the value α^* at $x_j \in B_j$, which means that B_j is a $\oplus$-branch. The two cost functions $d(x,\gamma^L_{v_i})w(\gamma^L_{v_i})$ and $d(x,\gamma^H_{v_j})w(\gamma^H_{v_j})$ intersect at $x_{i,j}$ in their dashed parts, which implies that they meet in B_j. Since the corresponding cost $\alpha_{i,j}$ is larger than α^* in this figure, a center at $x_j \in B_j$ cannot α^*-cover γ_{v_i}.

[9] Lower (resp. higher) means farther (resp. nearer) from/to the root.

The above discussion implies that whether the cost at the intersection of two cost lines is higher or lower than α^*, which can be tested by a feasibility test, determines if an additional center needs to be introduced or not. Each feasibility test determines the relative order of x_i, x_j, x_k, etc., for all vertices on spine σ_l. This is tantamount to sorting x_i, x_j, x_k, etc., which we can do by a sorting network, such as the AKS sorter. By examining the sorted sequence, and scanning σ_l from its lower end, we can determine the number of centers needed on σ_l.

Finally, we need to find the α^*-critical vertex that represents the part of spine σ_l not covered by the centers introduced so far, or the center that could cover additional vertices in the next higher spine. Namely, spine σ_l may become a $\ominus$-branch or a $\oplus$-branch *vis-à-vis* the next higher spine. If it becomes a $\ominus$-branch, there may be several candidates for the α^*-critical vertex. The situation is somewhat to that depicted in the left figure in Fig. 4, where γ_{v_l} and γ_{v_r} are the two candidates. The α^*-critical vertex is whichever candidate whose cost line reaches α^* first, i.e., at the lowest position.

If σ_l becomes a $\oplus$-branch in the next higher spine, we want to find the α^*-critical vertex in σ_l that can cover the "farthest" vertex in the next higher spine. Therefore, among the candidate critical vertices we pick the one whose cost line reaches α^* last, i.e., at the highest position..

Following Megiddo [20], for each spine we employ an AKS sorting network. The number of inputs to the AKS sorting networks employed at level l is thus $2n_l$. Each such AKS sorting network has $O(\log n_l)$ layers of comparators, and their sorted outputs can be computed with $O(\log n_l)$ calls to a feasibility test with Cole's speed up [10]. The total number of calls at all levels $l = 1, 2, \ldots, \lambda$ with Cole's speed up is thus $O(\sum_{l=1}^{\lambda} \log n_l)$. Since $\sum_{l=1}^{\lambda} n_l = O(n)$, we have $\sum_{l=1}^{\lambda} \log n_l \leq \lambda \log(n/\lambda) = O(\log^2 n)$. Since each feasibility test takes $O(p \log(n/p))$ by Theorem 8 (extended to $STD(T)$), the total time spent by the feasibility tests is $O(p \log^2 \log(n/p))$. In addition, we need time to compute the median at each layer of the AKS networks, which is $O(n_l)$ per layer and $O(n_l \log n_l)$ at level l. Summing this for all levels, we get $O(\sum_{l=1}^{\lambda} n_l \log n_l) = O(n \log n)$. ◀

6 Conclusion and Discussion

We have presented an algorithm for the weighted discrete p-center problem for tree networks with n vertices, which runs in $O(n \log n + p \log^2 n \log(n/p))$ time. This improves upon the previously best $O(n \log^2 n)$ time algorithm [10]. The main contributors to this speed up are spine tree decomposition, which enabled us to limit the tree height to $O(\log n)$, and the root-centric location policy, which made locating centers simple. Fractional cascading helped to shave a factor of $O(\log n)$ off the time complexity in Theorem 8. The $O(n \log^2 n)$ time algorithm [10] and ours both make use of the AKS sorting network [1], which is impractically large. However, recently AKS-like sorting networks with orders of magnitude reduced sizes have been discovered [13, 23], and further size reduction in the not-so-distant future may make the above algorithms more practical. We also presented a practical $O(n \log n + p^2 \log^2(n/p))$ time WDpC algorithm, which improves upon the $O(n \log^2 n \log\log n)$ time algorithm [20] when $p = O(\sqrt{n})$.

In Lemma 4 we showed that it takes $O(n \log n)$ time and space to compute the set of bending point sequences for the upper envelopes at all the vertices. Suppose that the weight of a vertex is increased arbitrarily, which could influence the locations of some centers, if the vertex becomes critical for a center. We can test this situation without updating the upper envelopes, and thus without increasing the time requirement. Therefore, every p-center query with the weight of one vertex arbitrarily increased can be answered in $O(p \log(n/p) \log n)$ time. This result realizes a sub-quadratic algorithm for the minmax regret p-center problem in tree networks [2].

References

1 M. Ajtai, J. Komlós, and E. Szemerédi. An $O(n \log n)$ sorting network. In *Proc. 15th ACM Symp. on Theory of Comput. (STOC)*, pages 1–9, 1983.

2 I. Averbakh and O. Berman. Minimax regret p-center location on a network with demand uncertainty. *Location Science*, 5:247–254, 1997.

3 Boaz Ben-Moshe, Binay Bhattacharya, and Qiaosheng Shi. An optimal algorithm for the continuous/discrete weighted 2-center problem in trees. In *Proc. LATIN 2006*, volume LNCS 3887, pages 166–177, 2006.

4 R. Benkoczi. *Cardinality constrained facility location problems in trees.* PhD thesis, School of Computing Science, Simon Fraser University, Canada, 2004.

5 R. Benkoczi, B. Bhattacharya, M. Chrobak, L. Larmore, and W. Rytter. Faster algorithms for k-median problems in trees. *Mathematical Foundations of Computer Science, Springer-Verlag*, LNCS 2747:218–227, 2003.

6 Binay Bhattacharya, Tsunehiko Kameda, and Zhao Song. Minmax regret 1-center on a path/cycle/tree. In *Proc. 6th Int'l Conf. on Advanced Engineering Computing and Applications in Sciences (ADVCOMP)*, pages 108–113, 2012.

7 Binay Bhattacharya and Qiaosheng Shi. Improved algorithms to network p-center location problems. *Computational Geometry*, 47:307–315, 2014.

8 Timothy M. Chan. Klee's measure problem made easy. In *Proc. Symp. on Foundation of Computer Science (FOCS)*, pages 410–419, 2013.

9 Bernard Chazelle and Leonidas J. Guibas. Fractional cascading: I. A data structuring technique. *Algorithmica*, 1:133–162, 1986.

10 R. Cole. Slowing down sorting networks to obtain faster sorting algorithms. *J. ACM*, 34:200–208, 1987.

11 G.N. Frederickson. Optimal algorithms for partitioning trees and locating p centers in trees. Technical Report CSD-TR-1029, Purdue University, 1990.

12 G.N. Frederickson. Parametric search and locating supply centers in trees. In *Proc. Workshop on Algorithms and Data Structures (WADS), Springer-Verlag*, volume LNCS 519, pages 299–319, 1991.

13 Michael T. Goodrich. Zig-zag sort: A simple deterministic data-oblivious sorting algorithm running in $O(n \log n)$ time. arXiv:1403,2777v1 [cs.DS] 11 Mar2014, 2014.

14 S.L. Hakimi. Optimum locations of switching centers and the absolute centers and medians of a graph. *Operations Research*, 12:450–459, 1964.

15 Trevor S. Hale and Christopher R. Moberg. Location science research: A review. *Annals of Operations Research*, 123:21–35, 2003.

16 M. Jeger and O. Kariv. Algorithms for finding p-centers on a weighted tree (for relatively small p). *Networks*, 15:381–389, 1985.

17 O. Kariv and S.L. Hakimi. An algorithmic approach to network location problems, part 1: The p-centers. *SIAM J. Appl. Math.*, 37:513–538, 1979.

18 N. Megiddo. Combinatorial optimization with rational objective functions. *Math. Oper. Res.*, 4:414–424, 1979.

19 N. Megiddo. Applying parallel computation algorithms in the design of serial algorithms. *J. ACM*, 30:852–865, 1983.

20 N. Megiddo and A. Tamir. New results on the complexity of p-center problems. *SIAM J. Comput.*, 12:751–758, 1983.

21 N. Megiddo, A. Tamir, E. Zemel, and R. Chandrasekaran. An $O(n \log^2 n)$ algorithm for the kth longest path in a tree with applications to location problems. *SIAM J. Comput.*, 10:328–337, 1981.

22 M.S. Paterson. Improved sorting networks with $O(\log n)$ depth. *Algorithmica*, 5:75–92, 1990.

23 Joel Seiferas. Sorting networks of logarithmic depth, further simplified. *Algorithmica*, 53:374–384, 2009.

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

25 A. Tamir. Improved complexity bounds for center location problems on networks by using dynamic structures. *SIAM J. Discrete Mathematics*, 1:377–396, 1988.