EFFICIENT ALGORITHMS FOR NETWORK CENTER/COVERING LOCATION OPTIMIZATION PROBLEMS

by

Qiaosheng Shi

B.S., Nanjing University of Science and Technology, 1999

M.S., Institute of Computing Technology, Chinese Academy of Sciences, 2002

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY
in the School
of
Computing Science

SIMON FRASER UNIVERSITY
Spring 2008

APPROVAL

Name:	Qiaosheng Shi
Degree:	Doctor of Philosophy
Title of thesis:	Efficient Algorithms for Network Center/Covering Location Optimization Problems

Examining Committee: Dr. Joseph G. Peters,
School of Computing Science, SFU
Chair

Dr. Binay Bhattacharya, Senior Supervisor
School of Computing Science, SFU

Dr. Pavol Hell, Supervisor
School of Computing Science, SFU

Dr. Thomas C. Shermer, SFU Examiner
School of Computing Science, SFU

Dr. Pat Morin, External Examiner,
School of Computer Science,
Carleton University

Date Approved: April 10, 2008

Abstract

We consider the algorithmic issues for the center and covering location optimization problems in network metric space. The demand set consists of all points of the network that require services and the supply set consists of all candidate locations of facilities in the underlying network. The center location problems aim to establish an optimal placement of facilities in the supply set in order to minimize the maximum distance from a demand point to its closest facility. The covering location problems seek to establish the minimum number of facilities such that the maximum distance from a demand point to its closest facility is no more than a predefined non-negative value. There is a tight relationship between the two problems. Generally, a solution for the covering location problem with a given value can be used to test the feasibility of the value in the corresponding center location problem.

Four cases of the center problem and the corresponding covering problem, where the demand set and the supply set are either subsets of the vertex set or subsets of the point set of the underlying network, are considered. Moreover, when the demand set is a subset of the vertex set, its weighted version of the problem is considered where each demand vertex is associated with a non-negative weight.

We study the center/covering location problems in general networks as well as specialized networks, such as tree networks, cactus networks, and partial k-tree networks for fixed k. We also look at some variations of the network center/covering location problem in an edge-weighted tree network, including conditional extensive facility location problems, continuous p-edge-partition problems, and constrained covering problems.

Keywords: facility location optimization, p-center problems, covering location problems, extensive facility, p-edge-partitions, parametric pruning.

To my wife Bei

"The journey of a thousand miles begins with a single step."

— Lao Tzu

Acknowledgments

I owe deep debt of gratitude to my senior supervisor, Dr. Binay Bhattacharya, for his continuous support, skillful guidance, enthusiasm and patience over the past five years. Binay gave me not only invaluable remarks on my research, but also insightful advices for my career development. Without his help this work would have not been possible.

I would also like to acknowledge the members of my committee, Dr. Pavol Hell, Dr. Thomas C. Shermer, and Dr. Pat Morin, for their time and effort.

I would like to express my sincere thanks to Dr. Arie Tamir at Tel Aviv University. His valuable feedback contributed greatly to this thesis. I enjoy the fruitful discussions with Arie and all the help from him.

I thank as well my dear friends and colleagues in and out of Simon Fraser University. Especially, I would like to thank Robert Benkoczi, Zhengbing Bian, David Breton, Yuzhuang Hu, Zengjian Hu, Wei Luo, Feng Ni, Zhongmin Shi, Yong Wang, and Xiang Zhang, for memorable years at Simon Fraser University.

Thanks full of love go to my wife, Bei, for her understanding and constant support. Bei, I just want you to know how happy I am to have you in my life.

Finally, I thank all those who I have not mentioned, but have encouraged me in so many ways.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Almost every public and private sector enterprise that we can think of has been faced with the problem of locating facilities. Government agencies need to determine locations of offices and other public services such as schools, hospitals, fire stations, ambulance bases, and so on. Industrial firms must determine locations for fabrication and assembly plants as well as warehouses. In these cases, the success or failure of facilities depends in part on the locations chosen for those facilities. Such problems are known as *facility location problems* [28].

In other words, facility location problems investigate where to physically locate a set of facilities (i.e., resources, servers) to satisfy some set of demands (i.e., customers, clients). The goal is to place these facilities such that the quality of service provided is optimized. In general, the quality of service is measured by some objective function, subject to a set of constraints. There are many different objective functions of possible interests, among which the minimization of maximum distance is one of the most studied. The corresponding problem is referred to as the *center problem* in the literature [34]. In this thesis, we focus on the center problem and a related problem, called the *covering location problem.* The covering location problem is to locate the minimum number of new facilities such that the maximum distance is no more than a predefined non-negative value.

One of the key components of facility location models is the metric space in which the customers are located and the facilities are to be located. Location problems are generally solved in one of the three basic spaces, i.e., continuous space, network space, and discrete space. Continuous space deals with the location problems in which customers and facilities are located in some subset of the d-dimensional Euclidean space $\mathbf{R}^d$. In discrete location models, customers are positioned at a (often finite) number of points and facilities are selected from a given (often finite) set of candidate points [28]. In a network space, the customers and facilities are confined to the edges and vertices of an underlying network. In this thesis, we discuss location problems in a network, which are referred to as *network location problems.*

1.1 Network center/covering location problems

Almost all location models are concerned with distances between points in the underlying space. Also, the cost of a facility to serve a client is a function of the distance traveled between the facility and the client. Depending on the applications, many different forms of distance measures are considered in the literature.

For location problems in a network $G = (V(G), E(G))$, where $V(G)$ is the vertex set of G and $E(G)$ is the edge set of G, each edge $e \in E(G)$ is associated with a positive length $l(e)$, and each vertex $v \in V(G)$ is associated with a non-negative weight $w(v)$. Let $A(G)$ denote the continuum set of points on the edges of G. We refer to interior points on an edge by their distances, along the edge, from the two endpoints of the edge. Generally, we use the length of the shortest path $\pi(x, y)$ to measure distance between a pair of points x and y in G, denoted by $d(x, y)$. Let $d(y, X) = \min_{x \in X} d(y, x)$ be the distance between a point y and a set $X \subseteq A(G)$.

To unify the formulations of different network location models, let $\mathcal{D}(G)$ be the set of demand points (or the *demand set*) and let $\mathcal{X}(G)$ be the set of candidate facility locations (or the *supply set*) in G. Note that it is possible that one or both of these two sets could be infinite and $\mathcal{D}(G)$ and $\mathcal{X}(G)$ need not be disjoint.

p-center problems In a p-center problem, a set of p centers (i.e., $X = \{\alpha_1, \ldots, \alpha_p\}$) is to be located in $\mathcal{X}(G)$ so that the maximum (weighted) distance from a demand point in $\mathcal{D}(G)$ to its nearest center in X is minimized, i.e.,

$$\min_{X \subseteq \mathcal{X}(G), |X| = p} \{F(X, \mathcal{D}(G)) = \max_{y \in \mathcal{D}(G)} \{w(y) \cdot d(y, X)\}\}. \tag{1-1}$$

Here $F(X, \mathcal{D}(G))$ denotes the cost of serving the demand set $\mathcal{D}(G)$ using facilities in X, and $w(y)$ is the non-negative weight associated with a demand point $y \in \mathcal{D}(G)$. As the solution to this problem may be not unique (there might be an infinite number of optimal solutions), we usually settle for computing just one optimal solution. If $w(y)$ is the same for all y in $\mathcal{D}(G)$, then the problem is called the *unweighted p-center problem.* Otherwise, the problem is called the *weighted p-center problem.* When the set $\mathcal{X}(G)$ of candidate center locations is finite, the problem is known as the *discrete p-center problem.* Accordingly, the problem is called the *continuous p-center problem* when $\mathcal{X}(G)$ is infinite. When both $\mathcal{X}(G)$ and $\mathcal{D}(G)$ are infinite, the problem is called *general p-center problem.* The p-center problem and its variants on general networks and in the plane have been shown to be NP-hard [47, 55].

Handler and Mirchandani [39] proposed a classification scheme for network center location models. It is a 4-position scheme, i.e., *Pos1/Pos2/Pos3/Pos4.* In Pos1 information about where the new facilities will be located is given. Pos2 contains information about where the demand points are located. In Pos3 the number of new facilities is given and in Pos4 the network type is described. In this thesis, we will use a 3-position scheme by simply removing Pos4 in the classification scheme of Handler and Mirchandani [39], that is, $\mathcal{X}(G)/\mathcal{D}(G)/p$ (supply set/demand set/number of facilities). Since the objective function and underlying space are clear in each problem, a 3-position scheme is sufficient.

Covering location problems The objective of a covering location problem is to locate the minimum number of new facilities needed to cover all of the demand points. Each demand point y is associated with a non-negative weight $w(y)$. Here we say a demand point y is covered by $x \in \mathcal{X}(G)$ if $w(y) \cdot d(y, x)$ is smaller than a predefined coverage radius r_c. This model can be formulated as follows:

$$m(r_c) = \min_{X \subseteq \mathcal{X}(G)} \{|X| : w(y) \cdot d(y, X) \leq r_c, \ \forall y \in \mathcal{D}(G)\}. \tag{1-2}$$

The objective function can be generalized by considering open-facility costs, that is, the open-facility costs of the candidate locations in $\mathcal{X}(G)$ are not uniform. In this case, the objective will be to minimize the total open-facility cost rather than the number of facilities opened. Both versions of the covering location problems are NP-hard [31]. However, in some special cases, such as the case when all the demand points are embedded on a tree network, the covering location problems can be efficiently solved [20, 56].

For a p-center problem (i.e., $\mathcal{X}(G)/\mathcal{D}(G)/p$), a value $r > 0$ is *feasible* if there exists a set of at most p points, say $\alpha_1, \ldots, \alpha_p$, in $\mathcal{X}(G)$ such that the distance between any demand point in $\mathcal{D}(G)$ and its nearest α_i $(1 \leq i \leq p)$ is not greater than r. An approach to test whether a given positive value is feasible is called a *feasibility test.* Clearly, if $m(r) \leq p$ then

r is feasible in the corresponding p-center problem, and otherwise r is infeasible. Therefore, an efficient solution for the covering location problem is useful for designing an efficient algorithm for the corresponding p-center problem.

In this thesis we discuss the center/covering location problems in general networks as well as specialized networks, such as tree networks, cactus networks, and partial k-tree networks. A detailed literature survey on the center/covering location problems in general networks, as well as various restrictive classes of networks, is given in Section 1.2. In Section 1.3, three generalizations of network center/covering location problems are presented. The explicit mention of the scope of this thesis appears in Section 1.4.

1.2 Review of related works

The network center/covering location problem has been extensively studied in the literature. In this section we review previous results on center location problems in an undirected network $G = (V(G), E(G))$. That is, p centers (i.e., $X = \{\alpha_1, \dots, \alpha_p\}$) are to be located in the supply set $\mathcal{X}(G)$ such that the maximum weighted distance $F(X, \mathcal{D}(G))$ from a point in the demand set $\mathcal{D}(G)$ to its nearest center in X is minimized. We will not separately discuss the covering location models. Instead, their results are mentioned in the section for the corresponding center location problem.

There are four special cases where the sets $\mathcal{X}(G)$ and $\mathcal{D}(G)$ are either subsets of $V(G)$ or subsets of $A(G)$. When $\mathcal{X}(G) = A(G)$, an optimal p-center is called *an absolute p-center.* In the case where $\mathcal{X}(G) = V(G)$ (that is, facilities are restricted to vertices), an optimal p-center is called *a vertex p-center.* The service cost of an absolute (resp. a vertex) p-center is called the absolute (resp. vertex) *p-radius.* When the centers serve all the points of the underlying network not merely the vertices, i.e., $\mathcal{D}(G) = A(G)$, centers are called *general centers.* Note that the weighted version of a p-center problem is considered only when $\mathcal{D}(G) = V(G)$ since the weighted version of the continuous demand set is not well-defined.

In [47, 55], p-center problems in general networks have been shown to be NP-hard (even when the network is a planar network of maximum vertex degree 4). The proofs use a simple transformation from the dominating set problem, which is known to be NP-complete [31]. The dominating set problem is stated below.

Definition 1.2.1 (Dominating set problem) *Given a network $G = (V(G), E(G))$ and* *a positive integer p, does there exist a dominating set $V' \subseteq V(G)$ with $|V'| \leq p$ such that each vertex of G is either in V' or adjacent to a vertex in V'?*

However, center problems are no longer NP-hard when either p is constant [47, 62, 69], or the underlying network is restricted to be a specialized network, such as a tree [29, 46, 47, 54, 55, 56], a cactus [8, 30, 48], or a partial k-tree (fixed k) [32]. In the following, we first survey unweighted/weighted $A(G)/V(G)/p$ problems when G is a general network. It is followed by the survey on tree and tree-like networks. Later in Section 1.3, we discuss three generalizations of center/covering location problems in trees, such as conditional extensive facility location problems, continuous edge-partition problems, and constrained covering problems.

1.2.1 General networks

The best known algorithms [47, 62, 69] to solve the $A(G)/V(G)/p$ problem are based on the following two simple observations (Observations 1.2.2 and 1.2.3).

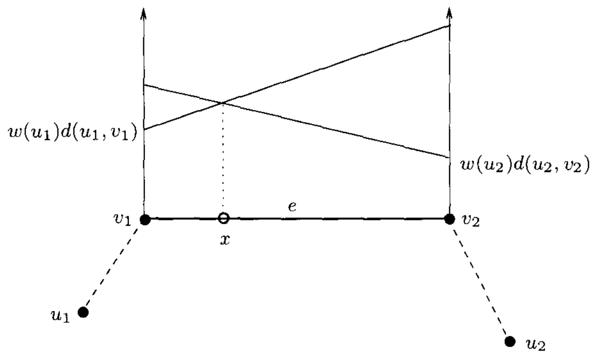


Figure 1.1: Observation 1.2.2.

Observation 1.2.2 *[47] There exists an absolute p-center such that all the centers are intersection points of service functions of pairs of vertices on edges and therefore, the optimal objective value is of the form $(w(u) \cdot w(v) \cdot L_e(u,v))/(w(u)+w(v))$, where $L_e(u,v)$ is the length of some simple path connecting u and v through edge e for some pair of vertices $u, v \in V(G)$.*

Refer to Figure 1.1. The intersection point x of service functions of vertices u_1 and u_2 on edge e is a candidate location of facilities. Also, $w(u_1)d(u_1,x) = w(u_2)d(u_2,x)$ is a candidate value for the optimal objective value. Therefore, there are at most $O(n^2)$ candidate points on each edge e where centers may be located in an optimal solution. Also, for each candidate point x on e which is determined by a pair of vertices u and v, the weighted distance from u (or v) to x is a candidate value for the absolute p-radius. It is not hard to see that the set of $O(mn^2)$ candidate points and their candidate values (denoted by $\mathcal{R}$) can be computed in $O(mn^2)$ time. Based on Observation 1.2.2 and an $O(mn \log n)$-time algorithm for the $A(G)/V(G)/1$ problem, Kariv and Hakimi [47] proposed an $O(m^p n^{2p-1} \log n/(p-1)!)$-time algorithm for finding an absolute p-center of a vertex-weighted network. For the vertex-unweighted network, the running time of the algorithm [47] is $(O(m^p n^{2p-1}/(p-1)!)$.

The second observation is for feasibility tests of the continuous p-center problem.

Observation 1.2.3 *[62] If a given non-negative value r is feasible for the $A(G)/V(G)/p$ problem, then there is a p-center solution in which each center is located at a (weighted) distance of exactly r from some demand vertex and all demand vertices are covered with service cost $\leq r$.*

The advantage of Observation 1.2.3 over Observation 1.2.2 is that only $O(mn)$ candidate points are needed to be considered for a feasibility test. Based on this property, Moreno [62] proposed an $O(m^p n^{p+1})$-time algorithm for the feasibility test of a given value r.

Since the absolute p-radius, denoted by r_p, of the $A(G)/V(G)/p$ problem is an element of a set $\mathcal{R}$ of cardinality $O(mn^2)$ (Observation 1.2.2), r_p can be found by performing $O(\log(mn^2)) = O(\log n)$ feasibility tests. This implies that the weighted continuous p-center problem in general networks can be solved in $O(m^p n^{p+1} \log n)$ time [62].

Tamir [69] improved Moreno's result [62] by efficiently solving a feasibility test for the 2-center problem in $O(m^2 n^2 \alpha(n))$ time (for a vertex-unweighted network, it is $O(m^2 n \log n\alpha(n))$ time). Here $\alpha(n)$ is the inverse Ackermann function [26]. The improvements [69] for $p \geq 3$ are achieved as follows. To determine the feasibility of r for the p-center problem, a set of $(p-2)$ points is selected from the $O(mn)$ candidate points, and then the 2-center feasibility test is applied to the vertices that are not covered by the selected $(p-2)$ centers within (weighted) distance r. Therefore, the $O(m^p n^p \log n\alpha(n))$ and $O(m^p n^{p-1} \log^2 n\alpha(n))$ bounds are achieved for the weighted and unweighted $A(G)/V(G)/p$ problems, respectively.

In this thesis we propose an improved algorithm for the unweighted/weighted $A(G)/V(G)/p$ problem, which is based on the geometric properties of a transformed version of the problem. The improved running time of this algorithm is $O(m^p n^{p/2} \log^2 n)$ time.

The weighted discrete p-center problem, $p \geq 2$, in a general network is trivially solvable in $O(pn^{p+1}/(p-1)!)$ time by testing all possible solutions. Unfortunately, the above ideas cannot be applied to improve the exhaustive algorithm for finding a vertex p-center in a general network.

1.2.2 Tree networks

In this section, we assume that the underlying structure of the network is a tree, denoted by $T = (V(T), E(T))$. A *centroid* of T, which can be found in linear time [47], is a vertex $o \in V(T)$ such that each subtree with the removal of o has the size at most $|V(T)|/2$. Note that a tree might have either one centroid or two. In the latter case, the two centroid vertices are connected by an edge [40].

Kariv and Hakimi [47] pointed out that the service cost function $F(x, V(T)), x \in A(T)$, is convex on every simple path in T. Based on this convex property, Kariv and Hakimi designed an $O(n \log n)$-time algorithm for 1-center problems. Later, Megiddo [54] showed that the unweighted/weighted $A(T)/V(T)/1$ and $V(T)/V(T)/1$ problems can be solved in linear time with a clever "trimming" technique, which is described in Section 3.2. In fact, for the unweighted 1-center problems in T, a simpler and more efficient algorithm was proposed by Handler [38]. It is based on one simple observation, that is, the absolute 1-center in a tree is the midpoint of a longest path. His algorithm is described as follows. Choose any point x in $A(T)$ and find the farthest point away from x in $A(T)$, say v_1. Then find the farthest point away from v_1, say v_2. The absolute 1-center of T is the midpoint of the path $\pi(v_1, v_2)$. Vertex 1-center is located at the closest vertex to the absolute 1-center.

When p is a part of the input, Kariv and Hakimi [47] proposed the first polynomially bounded algorithm, which runs in $O(n^2 \log n)$ time for the weighted $A(T)/V(T)/p$ and $V(T)/V(T)/p$ problems. The algorithms follow a common principle that is often adopted when designing algorithms for center location problems in a tree. First, a finite set $\mathcal{R}$ of real numbers, which is known to contain the optimal objective function value, is identified. Next, $\mathcal{R}$ is searched for the minimum feasible value. The set $\mathcal{R}$ for various location problems in T is listed in Table 1.1.

In a tree network, efficient algorithms to test the feasibility of a given r are known, and hence the corresponding location problem can be solved by a binary search of $\mathcal{R}$ using such a feasibility test. For the six problems listed in Table 1.1, the feasibility test on tree networks runs in $O(n)$ time [20, 47]. This implies that all of the six problems can be

Table 1.1: The sets $\mathcal{R}$ of different models in T [20, 21, 47]

	Unweighted	Weighted
$V(T)/V(T)/p$	$\{d(v_i, v_j)\}_{v_i,v_j \in V(T)}$	$\{w(v_i) \cdot d(v_i, v_j)\}_{v_i,v_j \in V(T)}$
$A(T)/V(T)/p$	$\{\frac{1}{2}d(v_i, v_j)\}_{v_i,v_j \in V(T)}$	$\{\frac{w(v_i) \cdot w(v_j)}{w(v_i)+w(v_j)} d(v_i, v_j)\}_{v_i,v_j \in V(T)}$
$V(T)/A(T)/p$	$\{d(v_i, v_j), \frac{1}{2}d(v_i, v_j)\}_{v_i,v_j \in V(T)}$	–
$A(T)/A(T)/p$	$\{\frac{1}{2l}d(v_i, v_j)\}_{v_i,v_j \in V(T), l=1,\ldots,p}$	–

solved in $O(|R| + n \log |R|)$ time. Therefore, the unweighted/weighted $A(T)/V(T)/p$ and $V(T)/V(T)/p$ problems and the unweighted $V(T)/A(T)/p$ problem can be solved in $O(n^2)$ time, and the unweighted $A(T)/A(T)/p$ problem can be solved in $O(pn^2)$ time.

In order to get a better bound (i.e., sub-quadratic time complexity), one cannot afford to compute the set $\mathcal{R}$ explicitly. This is essentially where the succinct representation of $\mathcal{R}$ is applied. Since the sets $\mathcal{R}$ for the various versions of the p-center problem have a close resemblance to the set $\mathcal{M}$ of inter-vertex distances in a tree network, we can use a succinct representation of $\mathcal{M}$ instead. Based on this observation, efficient algorithms (i.e., the ones in [56] by Megiddo et al. and [30] by Frederickson and Johnson) are designed for finding the k-th largest element in $\mathcal{M}$ to support a binary search over $\mathcal{R}$. In this way, Megiddo et al. [56] showed that, the unweighted/weighted $V(T)/V(T)/p$, the unweighted $A(T)/V(T)/p$ and $V(T)/A(T)/p$ problems are solved in $O(n \log^2 n)$ time. For the $A(T)/A(T)/p$ problem, their algorithm [56] runs in $O(n \min\{p \log^2 n, n \log n\})$ time. In [30], Frederickson and Johnson proposed an $O(n \log n)$ algorithm for the unweighted $V(T)/V(T)/p$ and $A(T)/V(T)/p$ problems as well as an $O(n \min(n, p) \log(\max(p, n)))$ algorithm for the $A(T)/A(T)/p$ problem.

Another type of method for center problems on a tree network is based on the parametric-searching technique. Since the early 1980s, the parametric-searching technique, proposed by Megiddo [52, 53], has become a powerful and ingenious tool for efficiently solving a variety of optimization problems. In this thesis we use this technique to design efficient algorithms for center problems in cactus networks (in Chapter 4) and for continuous edge-partition problems in tree networks (in Chapter 6).

The main principle of the parametric-searching technique is described as follows [2, 52]. Suppose we have a decision problem $\mathcal{P}(\lambda)$ that depends on a real parameter λ, and is monotone in λ, which means that if $\mathcal{P}(\lambda')$ is true for some λ', then $\mathcal{P}(\lambda)$ is true for all

$\lambda \leq \lambda'$. Our goal is to find the maximum λ for which $\mathcal{P}(\lambda)$ is true. Suppose further that $\mathcal{P}(\lambda)$ can be solved by a sequential algorithm A_s whose input is a set of data objects (independent of λ) and λ, and whose control flow is governed by comparisons, each of which amounts to testing the sign of some low-degree polynomial in λ, i.e., the number of roots of this polynomial is constant. Megiddo's technique then runs A_s "generically" at the unknown maximum λ^*. Whenever A_s reaches a branching point that depends on some comparison with associated low-degree polynomial $f(\lambda)$, it computes all its roots and runs A_s with the value of λ equal to each of these roots. This yields an interval between two adjacent roots, known to contain λ^*, and thus enables A_s to determine the sign of $f(\lambda^*)$, resolving the comparison and allowing the generic execution to proceed. As the algorithm proceeds, the interval known to contain λ^* keeps shrinking as a result of resolving further comparisons, and at the end either the interval becomes a singleton, which is thus the desired λ^*, or else λ^* can be shown to be equal to its upper endpoint of the final interval, since the final interval contains λ^* and $\mathcal{P}(\lambda)$ is true for any value in it.

Megiddo and Tamir [55] proposed an $O(n \log^2 n \log\log n)$-time algorithm for the weighted $A(T)/V(T)/p$ problem and an $O(n \log^3 n)$ algorithm for the $A(T)/A(T)/p$ problem, which are based on Megiddo's parallel version [53] of the above parametric-searching technique. Both of the algorithms can be improved to $O(n \log^2 n)$ by applying the result by Cole [25].

Megiddo's parallel version [53] of the parametric-searching technique and the result by Cole [25] are briefly described as follows.

A parallel algorithm A_p for the decision problem $\mathcal{P}(\lambda)$ is needed, which uses P processors and runs in $O(T_p)$ parallel steps. In each parallel step, we need to find out the results of P comparisons. As we know, the result of one comparison with the unknown value λ^* is determined by solving a constant number of decision problems. In Megiddo's parallel version [53], since $\mathcal{P}(\lambda)$ is monotone in λ, the results of all the P comparisons can be computed by solving $O(\log P)$ decision problems. Clearly, the time complexity to compute the results of all the PT_p comparisons is $O(T_s T_p \log P)$ where T_s is the running time of algorithm A_s.

Cole [25] observed that in certain applications of the parametric-searching technique, the running time can be improved to $O((P+T_s)T_p)$ by mixing the parallel steps of the algorithm A_p. Instead of invoking the decision procedure $O(\log P)$ times at each parallel step to resolve all the P comparisons, we invoke it only a constant number of times, say, three times. This will determine the outcome of 7/8 of the comparisons and will leave 1/8 of them unresolved. *Assume that each of the unresolved comparisons can influence only a constant number, say,* *two, of comparisons executed at the next parallel step.* The assumption is very important when Cole's idea is adopted to speed up Megiddo's parametric search technique. Then 3/4 of the comparisons in the next parallel step can still be simulated generically. Cole showed that if carefully implemented (by assigning an appropriate time-dependent weight to each unresolved comparison and choosing the weighted median each time), the number of parallel steps of the algorithm increases only by an additive logarithmic term, as desired.

In this thesis, we study weighted center problems in a tree network, including the weighted $V(T)/V(T)/p$ and $A(T)/V(T)/p$ problems, and propose optimal algorithms for the problems when p is an arbitrary fixed constant.

1.2.3 Tree-like networks

Although most of the reported works on center problems are for trees or for general networks, more and more attention has been paid to the classes of networks that are between these two extremes [33]. The location problems in cactus networks [30, 48], and in partial k-trees [41, 32] are worth mentioning.

A *tree decomposition* of a network $G = (V(G), E(G))$ is a pair $(\{b_i : i \in I\}, \mathcal{T} = (I, Y))$, where $\{B_i, i \in I\}$ is a family of subsets of $V(G)$ (called *bags*), I is its index set, and $\mathcal{T}$ is a tree with the following properties [66]:

(i) $\bigcup_{i\in I} B_i = V(G)$;

(ii) For every edge $e = (v, w) \in E(G)$, there is an $i \in I$ with $v, w \in B_i$;

(iii) For all $i, j, k \in I$, if j lies on the path between i and k in $\mathcal{T}$, then $B_i \cap B_k \subseteq B_j$.

The *treewidth* of a tree decomposition $(\{b_i : i \in I\}, \mathcal{T})$ is $\max_{i\in I} |B_i| - 1$. The treewidth of G is the minimum treewidth over all possible tree decompositions of G. A *partial k-tree* is a graph whose treewidth is k.

Partial k-trees One of the most important properties of trees, which is useful in designing efficient algorithms, is the existence of a 1-separator between any two disjoint subtrees. Partial k-trees are a more general class of graphs for which similar property is available. Let $\mathcal{TD}(G)$ denote a tree decomposition with treewidth k of a partial k-tree G. Clearly, there exists an i-separator ($i \leq k$) between two subnetworks represented by two disjoint subtrees $\mathcal{T}_1, \mathcal{T}_2$ of $\mathcal{TD}(G)$.

It is known that $\mathcal{TD}(G)$ can be found in linear time for fixed k [17]. Given a tree decomposition $\mathcal{TD}(G)$ with treewidth k of a partial k-tree G, an $O(p^2n^{k+2})$-time algorithm [32] was proposed by Granot and Skorin-Kapov [32] to solve the unweighted/weighted $V(G)/V(G)/p$ problems. The algorithm is based on a dynamic programming technique, which is described in Section 4.1.1.

Cactus networks A *cactus* network is a connected graph where any two simple cycles in the graph have at most one vertex in common. Cactus networks are partial 2-trees.

Frederickson and Johnson [30] showed that a feasibility test of the unweighted $V(G)/V(G)/p$ problem in a cactus network G can be solved in linear time (Lemma 13 in [30]). Observe that the optimal objective value lies in the set $\mathcal{R} = \{d(u,v)|u,v \in V(G)\}$, which is the set of inter-vertex distances. Actually, for any network G', the set $\mathcal{R}'$ that contains the optimal objective value for the $V(G')/V(G')/p$ problem has a close resemblance to the set $\mathcal{M}'$ of inter-vertex distances in G'.

Similar to the case when the underlying network is a tree, the set $\mathcal{M}$ of all inter-vertex distances in a cactus network G, has a special structure which enables searching in $\mathcal{M}$ without generating the entire set in advance. Note that cactus network G is a partial 2-tree. Indeed, it can be represented by a set of $O(n)$ sorted lists, which is computed using a centroid decomposition [56] of $\mathcal{TD}(G)$ of treewidth 2. This representation is very similar to the succinct representation of all inter-vertex distances in a tree, which is proposed by Megiddo et al. [56]. Here, instead of a centroid vertex decomposing a tree or a subtree into smaller parts, there is a centroid 2-separator decomposing G or a subnetwork of G into parts. Each list is associated with distances from a given vertex to some subset of vertices.

Using the linear-time feasibility test and the succinct representation of set $\mathcal{R}$, the unweighted $V(G)/V(G)/p$ problem is solvable in $O(n \log n)$ time [30].

1.3 Generalizations of network center/covering location problems

Three generalizations of center/covering location problems in an edge-weighted network G are introduced in this section. These are conditional extensive facility location problems, continuous p-edge-partition problems, and constrained covering problems.

More notations are needed. For any two different points x, y on an edge $e \in E(G)$, if x

and y are not the two endpoints of e, then we call the simple path from x to y a *partial edge* of e. We define the *length of a subgraph* G' of G, denoted by $l(G')$, to be the total length of its edges and partial edges. The *diameter of a subgraph* G' is the length of a longest simple path of G'.

1.3.1 Conditional extensive facility location problems

Usually a facility is represented by a point in a metric space [6, 14, 29, 30, 34, 46, 54, 56]. However, in recent years there has been a growing interest in studying the location of connected structures (referred to as *extensive* facilities), i.e. those that cannot be represented by isolated points but as some connected structures, such as subtrees [12, 15, 67, 70, 71, 73], paths [12, 15, 35, 42, 57, 59], straight lines [1], line segments [3], or polygonal chains [5, 4, 27]. These studies were motivated by concrete decision problems related to routing and network design [70]. Moreover, in many practical situations a number of facilities are already located in the network and provide service to the customers. These service providers are fixed and cannot be changed. However, due to some necessity, there might be a need to place more facilities to provide improved services to the clients. This kind of facility location problem is known as the *conditional location problem* [11, 58].

In this thesis we study conditional path-/subtree-center location problems in a tree network. In specific terms the conditional 1-extensive center location problem is to locate a path-/subtree-shaped facility, whose length is no more than a predefined non-negative value, such that the maximum weighted distance from demand points to the union of this facility and existing facilities is minimized.

It is not hard to see that the subtree-shaped facility location problem in a general network is NP-hard. The proof uses a simple transformation from the connected dominating set problem which is known to be NP-complete [31]. The connected dominating set problem is defined below.

Definition 1.3.1 (Connected dominating set problem) *Given a network $G = (V(G), E(G))$ and a positive integer p, does there exist a dominating set $V' \subset V(G)$ with $|V'| \leq p$ such that V' is a dominating set of G and the subgraph induced by V' is connected?*

In the case when there are no existing facilities in a tree network, Hedetmiemi et al. [42] proposed linear-time algorithms for locating a path-shaped facility without length constraint. When the existing facilities are taken into consideration, Mesa [57] provided an

$O(n \log n)$-time algorithm for the conditional path-shaped center problem in the pure topological case of a tree, where vertices and edges in the tree are unweighted.

In [70, 71], Tamir et al. presented $O(n \log n)$-time algorithms to solve the conditional problems in trees where the vertices and the edges are weighted. The basic technique used in their algorithms is the parametric-searching technique. In this thesis we propose improved algorithms by combining parametric-searching and pruning techniques (i.e., parametric-pruning technique [15]).

1.3.2 Continuous p-edge-partition problems

Another way to look at various p-center problems ($p > 1$) in a network $G = (V(G), E(G))$ is through the network partitioning problems. A *discrete partition* of G into p connected components is a collection of p connected subnetworks such that their vertex sets are pairwise disjoint and the union of their vertex sets is the vertex set of G. It can be done by deleting a subset of edges from G. For instance, when G is a tree network, a discrete partition of G into p connected components (i.e., subtrees here) is achieved by removing $p - 1$ edges. A *continuous partition* of G into p connected components is a collection of p connected subnetworks such that no pair of them intersect at more than m points and their union is the point set of G (i.e., $A(G)$). It can be achieved by splitting at a subset of points in G. For instance, when G is a tree network, a continuous partition of G into p subtrees is done by splitting at $p - 1$ points.

When the demand set is the vertex set $V(G)$, a p-center problem (i.e., $V(G)/V(G)/p$ and $A(G)/V(G)/p$) is actually to find a discrete partition of G into p connected subnetworks, and the objective is to minimize the maximum service cost of 1-centers of these subnetworks. Similarly, we need to find a continuous partition of G into p connected subnetworks with minimum service cost when the demand set is the point set $A(G)$ (i.e., $V(G)/A(G)/p$ and $A(G)/A(G)/p$ models).

The continuous partition problem in a weighted tree network space $T = (V(T), E(T), l)$ is to partition a tree network space into p subtrees, minimizing (resp. maximizing) the maximum (resp. minimum) "size" of the subtrees. When the size refers to the diameter of the component, the min-max problem coincides with the general p-center problem (i.e., $A(T)/A(T)/p$). Polynomial-time algorithms solving this problem appear in [20, 55]. When the size is the length of the component, the continuous partition problem is NP-hard, since the Partition problem is a special case of the problem. The Partition problem is stated as follows.

Definition 1.3.2 (The Partition problem) *Given a multiset B of integers, is there a way to partition B into two subsets B_1 and B_2 such that the sums of the numbers in each subset are equal? The subsets B_1 and B_2 are disjoint and they cover B.*

Without loss of any generality, we assume that all integers in the multiset B are positive and that any integer in B is less than one half of the sum of the numbers in B. We construct a tree network consisting of $|B|$ leaves and a central vertex. Each leaf is connected by an edge to the central vertex and for each integer k in B, there is an edge in the tree network whose length is equal to k. Then, it is not hard to see that the answer to the Partition problem on B is "Yes" if and only if there is a continuous $p(=2)$-partition of the corresponding tree network such that the two subtrees induced by the continuous 2-partition have the same length [37].

Due to the above NP-hardness result, in this thesis, we consider a class of continuous partitions in the tree, where the $p-1$ cut points, splitting the tree network into p components, are restricted to be on the edges, called *continuous edge-partitions.* Its difference from continuous partition is as follows.

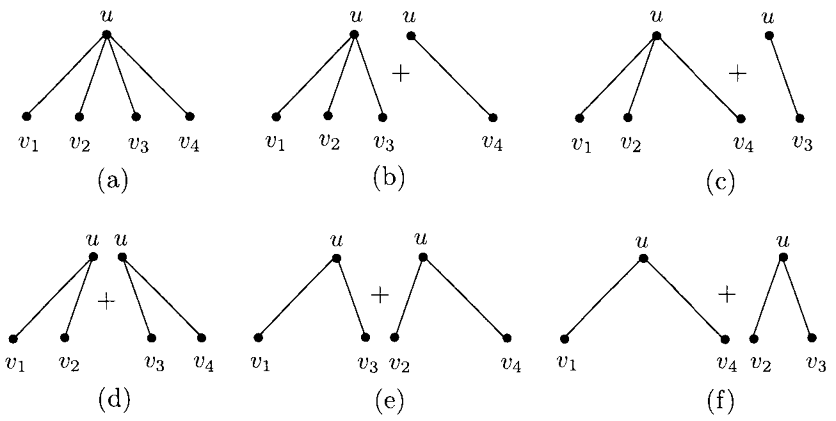


Figure 1.2: Main difference between continuous edge-partition and continuous partition. (a) a tree network; (b) the vertex cut on vertex u and edge $\overline{uv_4}$ in a continuous edge-partition; (c) the vertex cut on vertex u and edge $\overline{uv_3}$ in a continuous edge-partition; (d) (e) (f) possible cuts (partitions) at vertex u in a continuous partition.

Let x be a point on an edge $e : \overline{uv} \in E(G)$. A *cut* at x is a splitting of e into two closed partial edges: one is from vertex u to x and the other is from x to vertex v. A cut at an interior point of an edge is called an *interior cut*. A cut at an endpoint of an edge is called a *vertex cut*. The main difference between continuous partitions and continuous edge-partitions is on vertex cuts. As illustrated in Figure 1.2(b)(c), a vertex cut in a continuous edge-partition is uniquely defined by a vertex and an edge incident to this vertex. However, a cut at a vertex in a continuous partition is a partition of branches attached to that vertex (see Figure 1.2(d)(e)(f)).

Basically, a continuous p-edge-partition of T is a set of p subtrees induced by $p-1$ cuts. The *max-min continuous p-edge-partition problem* (*max-min CEP*, for short) is to find a continuous p-edge-partition of T that maximizes the smallest length of a subtree; and the *min-max continuous p-edge-partition problem* (*min-max CEP*, for short) is to find a continuous p-edge-partition of T that minimizes the largest length of a subtree.

Halman and Tamir [37] presented $O(n^2 \log(\min\{p, n\}))$-time algorithms for the max-min and min-max CEP problems. Recently, Lin et. al. [49] proposed more efficient algorithms for the two problems. The proposed algorithms of Lin et. al. [49], which run in time $O(n^2)$, are based on efficiently solving a problem, called the *ratio search problem* (defined in Chapter 6). In this thesis we study the max-min and min-max CEP problems and propose the first sub-quadratic algorithm for the max-min problem, which runs in $O(n \log^2 n)$ time, and an $O(nh_T \log n)$ (or $O(n \sum_{v \in V(T)} \delta_T(v))$) algorithm for the min-max problem, where h_T is the height of the underlying tree network and $\delta_T(v)$ is the degree of vertex v. When $h_T = o(n/\log n)$, our result for the min-max problem is better.

1.3.3 Constrained covering problems

The constrained covering problem is a generalization of the covering location problem, also known as the *conditional covering problem* [44, 45] (first introduced by Moon and Chaudhry in [60]). To distinguish from conditional location problem described above, we use the name of constrained covering problem (for short, CCP). This problem is defined on a network $G = (V(G), E(G))$. The vertex set $V(G)$ represents the set of demand points that must be covered by a facility, as well as the set of potential facility locations. A facility located at vertex $u \in V(G)$ incurs a non-negative *open-facility cost* $c(u)$, and provides a non-negative *coverage radius* of $r(u)$. A demand point $v \in V(G)$ is *covered* by a facility u if and only if $u \neq v$ and $d(u, v) \leq r(u)$. That is, a demand point is covered by a facility if it lies within the coverage radius of the facility and an established facility must be covered by another established facility. The CCP seeks to minimize the sum of open-facility costs required to cover all vertices in $V(G)$.

The CCP has applications in security and military services. An example is that a set of service centers needs to be established, but due to the threat of terrorist attack, each service center might be destroyed and in that case it must rely on other centers to provide service to its customers. Another scenario is for inter facility support, for example, a set of warehouses needs to be located and each warehouse must resort to other warehouses in case of shortage of its inventory [60].

CCP in a general network is strongly NP-hard [44], because the total dominating set problem, which is shown to be strongly NP-hard even on bipartite graphs by Pfaff et al. [64], is a special case of CCP.

Definition 1.3.3 (Total dominating set problem) *Given a network $G = (V(G), E(G))$ and a positive integer p, does there exist a dominating set $V' \subset V(G)$ with $|V'| \leq p$ such that each vertex of G is adjacent to a vertex in V'?*

The total dominating set problem is a version of CCP where all the open-facility costs, edge lengths, and coverage radii are 1.

However, CCP is polynomially solvable on some special underlying networks. Lunday et al. [50] considered CCPs with uniform coverage radius in path networks. They gave a linear time dynamic programming algorithm for CCP with uniform open-facility cost and an $O(n^2)$-time dynamic programming algorithm for CCP with non-uniform open-facility costs [50]. Horne and Smith worked on CCP with non-uniform coverage radii and proposed an $O(n^2)$ time algorithm for CCP on paths and extended stars [45], and an $O(n^4)$ time algorithm for CCP on tree networks [44]. Their algorithms are also based on the dynamic programming technique. It is worth mentioning that Moon and Papayanopoulos [61] solved a variation of CCP optimally on tree networks. In their problem, the facilities with a uniform open cost can be located at any place in the underlying network (not necessarily at the vertices of the network) and each demand point (instead of facility) has a location specific radius within which a facility must be located.

In this thesis, we study this problem in a path, extended-star, or tree network. An *extended star* is a network in which three or more path networks are connected by a single root vertex.

1.4 Scope of this thesis

In this section, we summarize the list of problems considered in this thesis, and the results obtained on those problems.

In Chapter 2, we study continuous p-center problems in a general network. Here demand points are located at the vertices of the network, and centers can be located anywhere in the network. In the weighted case, each demand point is also associated with a non-negative weight. The objective is to compute a set of p centers such that the maximum unweighted/weighted distance from demand points to their closest centers is minimized. We provide an efficient algorithm for both the unweighted and weighted problems in a general network. The running time of the improved algorithm is $O(m^p n^{p/2} \log^2 n)$.

In Chapter 3, we consider the restriction of p-center problems to tree networks in which demand points are located at vertices and each demand point is associated with a non-negative weight, including the weighted discrete and continuous p-center problems where p is an arbitrary fixed constant. Megiddo [54] used a "trimming" technique to solve the weighted 1-center problems (i.e., the weighted discrete and continuous 1-center problems) in linear time. The problem of generalizing the trimming approach to solve the weighted p-center problem for $p > 1$ has been open for over twenty years. Our results in this chapter partially resolve this long standing open problem. Moreover, we present a simple parametric-pruning approach for the weighted 1-center problem, and the running time of this approach is $O(n)$. The proposed approach for the weighted 1-center problem can be adapted to solve the weighted p-center problem on the real line in linear time for any fixed value p. The result for the real line has been published [14].

In Chapter 4, we discuss various p-center problems in tree-like networks, i.e., partial k-trees, cactus networks. When the underlying network is a partial k-tree, we study the weighted discrete p-center problem in which centers are restricted to the vertices of the network and present an efficient algorithm for relatively small p. The running of our algorithm is $O(pn^p \log^{k-1} n)$, that is, when $p < k + 2$, our algorithm is better than the one of Granot and Skorin-Kapov [32]. Following this result, we discuss the weighted continuous p-center problem in which centers can be located at any place in the network, and devise an $O(p^2 k^{k+1} n^{2k+3} \log n)$-time algorithm for it. When the underlying network is a cactus, the following center problems are considered. We first provide an $O(n \log n)$-time algorithm to solve the weighted discrete and continuous 1-center problems. We then show that the weighted continuous 2-center problem can be solved in $O(n\log^3 n)$ time. We also look at various center problems in a cactus network where p is a part of the input. Our algorithms for the p-center problems in cactus networks are based on the parametric-searching technique. We propose an $O(n\log^2 n)$ algorithm for the weighted discrete p-center problem, $O(n^2)$ algorithms for the weighted continuous p-center problem and the unweighted discrete p-center problem with demand set of infinite size, and an $O(n^2\log^2 n)$ algorithm for the general p-center problem in a cactus network. The results for the center problems in cactus networks have been published [8].

In Chapters 5, 6, and 7, we look at generalizations of the center/covering location problem in an edge-weighted tree network.

In Chapter 5, the conditional extensive facility location problem in a tree network is discussed, in which a set of existing facilities is given. The objective is to locate a path-/subtree-shaped facility with minimum service cost whose length is no more than a predefined non-negative value. We propose optimal algorithms for them using the parametric-pruning technique. The result in this chapter has been published [12].

In Chapter 6, we study the continuous p-edge-partition problem in a tree that is to divide the underlying tree into p subtrees by selecting $p-1$ cut points along the edges of the underlying tree such that the maximum (minimum) length of the subtrees is minimized (maximized). For the max-min continuous p-edge-partition problem in a tree, we propose the first sub-quadratic algorithm, which runs in time $O(n\log^2 n)$. For the min-max problem, an $O(nh_T\log n)$ (or $O(n\sum_{v\in V(T)}\delta_T(v))$) algorithm is proposed, where h_T is the height of the underlying tree network and $\delta_T(v)$ is the degree of vertex v.

In Chapter 7, we look at one generalization of the covering location problem in a tree, that is, the constrained covering problem. The constrained covering problem in a tree seeks to minimize the sum of open-facility costs required to cover all vertices with the constraints that a vertex is covered by a facility if it lies within the coverage radius of the facility and an established facility must be covered by another established facility. We propose efficient algorithms for constrained covering problems on path, extended star, and tree networks. In particular, we provide an $O(n\log n)$-time algorithm for path networks, an $O(n^{1.5}\log n)$-time algorithm for extended-star networks and an $O(n^3\log n)$-time algorithm for tree networks.

Finally, the concluding remarks on our studies in this thesis appear in Chapter 8. Here, once again, we discuss our results on different problems along with their possible extensions for future work.

Chapter 2

Continuous center problems in general networks

In this chapter, we consider the unweighted/weighted continuous p-center problems in a general network $G = (V(G), E(G))$ ($|V(G)| = n$ and $|E(G)| = m$), i.e., the unweighted/weighted $A(G)/V(G)/p$ problems. Here demand points are located at vertices of the network G, and centers can be located anywhere in G. Each demand point $v \in V(G)$ is also associated with a non-negative weight $w(v)$. In the unweighted case, $w(v) = 1$ for all $v \in V(G)$. The objective is to compute a set $X(\subseteq A(G))$ of p centers such that the maximum unweighted/weighted distance from demand points in $V(G)$ to their closest centers in X, i.e., $F(X, V(G))$, is minimized. We provide an improved algorithm for both the unweighted and weighted problems, which runs in time $O(m^p n^{p/2} \log^2 n)$. The best previous result for the weighted (resp. unweighted) $A(G)/V(G)/p$ problem, proposed by Tamir [69], is $O(m^p n^p \log n\alpha(n))$ (resp. $O(m^p n^{p-1} \log^2 n\alpha(n))$) where $\alpha(n)$ is the inverse Ackermann function [26].

Organization of the chapter The main idea and overall approach of our algorithm is presented in Section 2.1, in which we show that our problem is related to a general geometrical problem called p-dimensional *Klee's measure problem.* In Section 2.2, we provide the process of transforming our problem to a collection of p-dimensional Klee's measure problems. Section 2.3 gives a brief summary.

2.1 Main idea and overall approach

As we mentioned in Section 1.2.1, the best known algorithms [47, 69] to solve the $A(G)/V(G)/p$ problem are based on two simple observations, that is, Observations 1.2.2 and 1.2.3. Observation 1.2.2 states that there is an absolute p-center such that all the centers are in a set of $O(mn^2)$ points and the absolute p-radius is in a set $\mathcal{R} = \{[w(u) \cdot w(v) \cdot (d(u, u') + l(e) + d(v, v')]/(w(u) + w(v))\}_{u,v \in V(G); e \in E(G)}$. Obviously, $|\mathcal{R}| = O(mn^2)$. On the other hand, from Observation 1.2.3, we know that only $O(mn)$ candidate points need to be considered for a feasibility test of the $A(G)/V(G)/p$ problem.

To achieve a better upper bound for the $A(G)/V(G)/p$ problem, we continue to decrease the size of the set that contains an absolute p-center. The following observation (Observation 2.1.1) shows that instead of $O(mn)$ candidate points, only m candidate continuous regions (i.e., edges) and n candidate points (i.e., vertices) are considered for a feasibility test.

Observation 2.1.1 *If a given non-negative value r is feasible for the $A(G)/V(G)/p$ problem, then there is a p-center solution in which every edge (not including its two endpoints) in G contains at most one center and all demand vertices are covered with service cost $\leq r$.*

The reason for the above observation is that if an edge $e : \overline{uv}$ contains more than one center in X (where $|X| \leq p$, $X \subseteq A(G)$, and $F(X, V(G)) \leq r$), then a new set of centers X', constructed by replacing the centers in $X \cap A(e)$ with u and v, has a service cost of no more than r, i.e., $F(X', V(G)) \leq r$, and is of cardinality no more than p, i.e., $|X'| \leq p$.

A *local feasibility test* of the $A(G)/V(G)/p$ problem is to determine if there exists a set of p centers on a given set $E_{p'}$ of p' $(0 \leq p' \leq p)$ edges $\{e_1, \ldots, e_{p'}\}$ (note that each edge contains one center and does not include its two endpoints) and a given set of $p - p'$ vertices such that all demand points in $V(G)$ can be served within a given non-negative value r. It is easy to see that the feasibility test of a given value r can be completed by solving $O((m+n)^p) = O(m^p)$ local feasibility tests of r on all possible subsets of p' edges and $p - p'$ vertices, $0 \leq p' \leq p$.

Our algorithm for the $A(G)/V(G)/p$ problem is described as follows.

Step 1: Compute the set $\mathcal{R}$ that contains the absolute p-radius.

Step 2: Perform a binary search over $\mathcal{R}$. At each iteration, test the feasibility of a non-negative value r as follows. For each set $E_{p'} \subseteq E(G)$ of p' edges and each set of $p - p'$ vertices, $0 \leq p' \leq p$,

Step 2.1: remove all demand vertices that can be covered by the $p - p'$ vertices with service cost $\leq r$; and

Step 2.2: for the remaining demand vertices, execute the local feasibility test of r on the set $E_{p'}$ as described in the remaining part of this chapter.

It is sufficient to show our approach for a local feasibility test of r on a set E_p of p edges. The main idea of our decision approach is to transform the local feasibility test of r on E_p to a general geometrical problem called p-dimensional *Klee's measure problem* (for short, KMP) [63].

Definition 2.1.2 (Klee's Measure Problem) *Given a set of intervals (of the real line), find the length of their union.*

The natural extension of KMP to d-dimensional space is to ask for the d-dimensional measure of a set of d-boxes, where d is a positive integer. A *d-box* is the cartesian product of d intervals in d-dimensional space. It is known that, given a set of n d-boxes, a $d(\geq 2)$-dimensional KMP can be solved in time $O(n^{d/2} \log n)$ using $O(n)$ storage [63]. Thus, a feasibility test of the unweighted/weighted $A(G)/V(G)/p$ model can be solved in $O(m^p n^{p/2} \log n)$ time if we are able to transform a local feasibility test into a KMP. The following theorem is then implied.

Theorem 2.1.3 *The unweighted/weighted $A(G)/V(G)/p$ center problems, for $p \geq 2$, can be solved in $O(m^p n^{p/2} \log^2 n)$ time, where n is the number of vertices and m is the number of edges.*

In the remaining part of this chapter, we show the process of transforming a local feasibility test of the weighted $A(G)/V(G)/p$ problem to a p-dimensional KMP.

2.2 Transformation of a local feasibility test to a KMP

Let us consider the case where $p = 2$. The transformation for the case where $p > 2$ can be developed in a similar way. Let $e_1 : \overline{u_1 v_1}$ and $e_2 : \overline{u_2 v_2}$ be the two edges to test the local feasibility of a given non-negative value r. A local 2-center solution is composed of two points in which one point lies on e_1 and the other one lies on e_2.

We consider a 2-dimensional space in which x_i-axis represents edge $e_i, i = 1, 2$. Let u_1 and u_2 be the origin, as shown in Figure 2.1(b). In this coordinate system, the x_i-coordinate

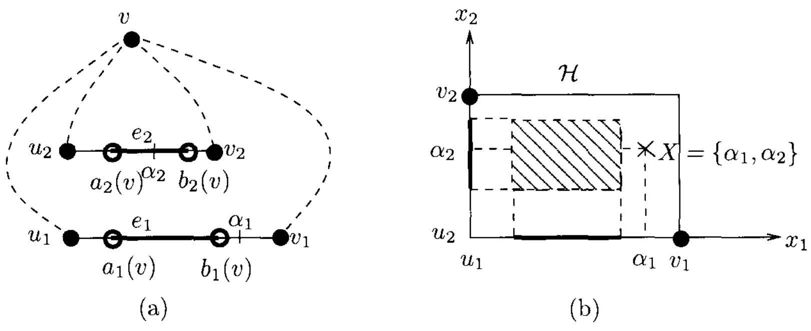


Figure 2.1: Mapping a 2-center local feasibility test to a 2-dimensional KMP.

of a point represents a location on edge e_i with respect to $u_i, i = 1, 2$. Therefore, a point in this 2-dimensional space can be considered as a possible 2-center solution on edges e_1, e_2. We denote a point y by $(x_1(y), x_2(y))$. Clearly, only points within the bounded rectangular area $\mathcal{H} : \{y | 0 \leq x_1(y) \leq l(e_1), 0 \leq x_2(y) \leq l(e_2)\}$ are candidate 2-center solutions on e_1, e_2, see Figure 2.1(b). In other words, $\mathcal{H}$ consists of all possible 2-center solutions on e_1, e_2.

For a demand vertex v, there is at most one continuous region on each edge $e_i, i = 1, 2$, denoted by $R_i(v)$, which contains all points on e_i with (weighted) distance to v larger than r. It is possible that $R_i(v)$ is empty for some i $(\in \{1, 2\})$, in which case v can be served by any 2-center solution on e_1, e_2 with service cost $\leq r$. In Figure 2.1(a), the bold (partial) edge of e_1 (resp. e_2) is $R_1(v)$ (resp. $R_2(v)$). Let $a_i(v)$ (resp. $b_i(v)$) be the left (resp. right) endpoint of $R_i(v), i = 1, 2$. Note that $R_i(v), i = 1, 2$ might be a closed, half-closed, or open region, see Figure 2.2. Furthermore, we observe that if $a_i(v) \in R_i(v)$ (resp. $b_i(v) \in R_i(v)$) then $a_i(v) = u_i$ (resp. $b_i(v) = v_i$), $i = 1, 2$. In other words, if an endpoint of $R_i(v)$, say $a_i(v)$, is an interior point of an edge e_i, that is, not u_i or v_i, then $a_i(v) \notin R_i(v)$.

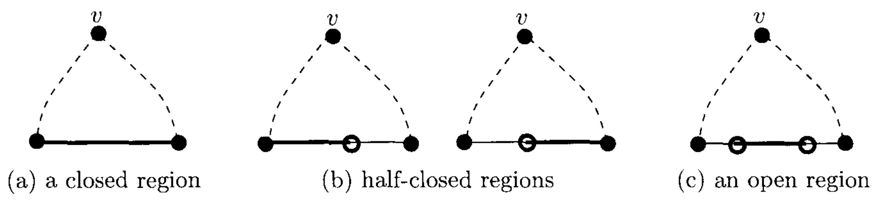


Figure 2.2: Close, half-close, and open regions.

A rectangular area in the 2-dimensional space (the shadow part in Figure 2.1(b)) is obtained for every demand vertex v, denoted by $\mathcal{H}(v)$, which is constructed from the two continuous regions $R_1(v), R_2(v)$. That is, $\mathcal{H}(v) = \{y | x_1(y) \in R_1(v), x_2(y) \in R_2(v)\}$. It is easy to see that any 2-center solution (point) in $\mathcal{H}(v)$ cannot cover v with a service cost $\leq r$ and any 2-center solution in $\mathcal{H} \setminus \mathcal{H}(v)$ can cover v with a service cost of no more than r. In Figure 2.1, the 2-center solution $X = \{\alpha_1, \alpha_2\}$ can cover v with a service cost no of more than r, but any solution in the shadow area cannot cover v with a service cost $\leq r$. We call $\mathcal{H}(v)$ the *forbidden area* of v. Note that, in Figure 2.1(a), $R_1(v)$ and $R_2(v)$ are open regions, and then the boundary of $\mathcal{H}(v)$ is not included in $\mathcal{H}(v)$.

We compute such forbidden areas for all demand vertices in $V(G)$. Thus, the local feasibility test on edges e_1, e_2 is transformed into the following question: *does the union* $\bigcup_{v \in V(G)} \mathcal{H}(v)$ *of forbidden areas cover* $\mathcal{H}$*?* If the answer is 'yes' then r is infeasible on edges e_1, e_2, otherwise r is feasible on edges e_1, e_2. This question can be answered by solving a 2-dimensional KMP on a new set of rectangles, which are constructed from these forbidden areas.

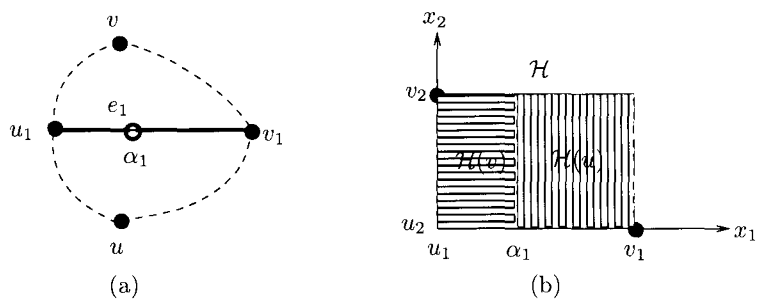


Figure 2.3: The reason for constructing a new set of rectangles.

The reason for constructing a new set of rectangles, instead of directly using the forbidden areas, is that the boundary or some part of the boundary of a forbidden area might not be included in the forbidden area. For example, let u and v be a pair of demand points such that $R_2(v) = R_2(u) = [u_2, v_2]$ and such that $R_1(v) = [u_1, \alpha_1), R_1(u) = (\alpha_1, v_1]$ (i.e., $w(v) \cdot d(v, \alpha_1) = w(u) \cdot d(u, \alpha_1) = r$). See Figure 2.3 for reference. Clearly, the union of the two forbidden areas $\mathcal{H}(v), \mathcal{H}(u)$ does not cover $\mathcal{H}$ (i.e., any point in $\mathcal{H}$ with its x_1-coordinate equal to $d(u_1, \alpha_1)$ is not covered.). However, the measure of the union of the two corresponding rectangles is equal to the measure of $\mathcal{H}$. To fix this problem, we construct a new set of rectangles from these forbidden areas. Lemma 2.2.1 shows that we can construct a set of rectangles in a way such that the above question can be answered by solving a KMP on this new set of rectangles. Let

$$\epsilon_1 = \min_{u,v \in V(G), i=1,2} \{|a_i(u) - a_i(v)|, l(e_i) : a_i(u) \neq a_i(v)\},$$

$$\epsilon_2 = \min_{u,v \in V(G), i=1,2} \{|a_i(u) - b_i(v)|, l(e_i) : a_i(u) \neq b_i(v)\},$$

and

$$\epsilon_3 = \min_{u,v \in V(G), i=1,2} \{|b_i(u) - b_i(v)|, l(e_i) : b_i(u) \neq b_i(v)\}.$$

Let ϵ be $(\min\{\epsilon_1, \epsilon_2, \epsilon_3\})/2$. Clearly, $\epsilon > 0$ and ϵ can be computed in $O(n \log n)$ time.

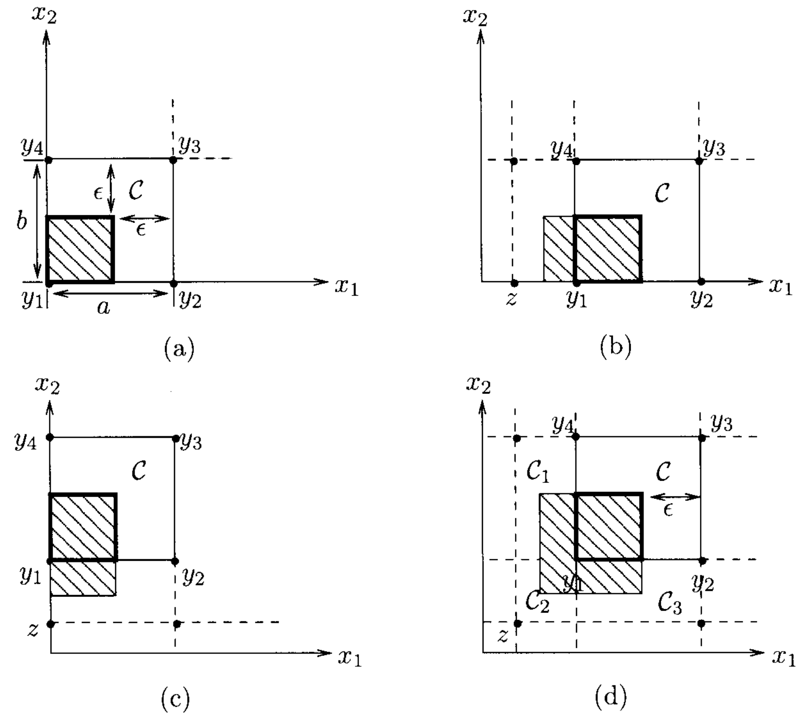


Figure 2.4: Lemma 2.2.1 (a) Case 1: y_1 is located at the origin; (b) Case 2: $x_2(y_1) = 0$; (c) Case 3: $x_1(y_1) = 0$; (d) Case 4: $x_1(y_1) \neq 0$ and $x_2(y_1) \neq 0$.

Lemma 2.2.1 *For each $v \in V(G)$, let $a'_i(v) = a_i(v) + \epsilon$ if $a_i(v) \notin R_i(v)$, and otherwise let $a'_i(v) = a_i(v)$; let $b'_i(v) = b_i(v) - \epsilon$ if $b_i(v) \notin R_i(v)$, and otherwise let $b'_i(v) = b_i(v), i = 1, 2$. Let $\mathcal{H}'(v) = \{y | a_1(v) \leq x_1(y) \leq b_1(v), a_2(v) \leq x_2(y) \leq b_2(v)\}$.*

Therefore, the measure of $\sum_{v \in V(G)} \mathcal{H}'(v)$ is equal to the measure of $\mathcal{H}$ if and only if the union $\bigcup_{v \in V(G)} \mathcal{H}(v)$ covers $\mathcal{H}$.

Proof It is trivial that $\mathcal{H}'(v) \subseteq \mathcal{H}(v), v \in V(G)$. Hence, if the measure of $\sum_{v\in V(G)} \mathcal{H}'(v)$ is equal to the measure of $\mathcal{H}$ then the union $\bigcup_{v\in V(G)} \mathcal{H}(v)$ covers $\mathcal{H}$. We prove in the following that if $\sum_{v\in V(G)} \mathcal{H}(v)$ covers $\mathcal{H}$, then the measure of $\sum_{v\in V(G)} \mathcal{H}'(v)$ is equal to the measure of $\mathcal{H}$.

We draw lines along boundaries of all forbidden areas $\mathcal{H}(v), v \in V(G)$. These lines partition $\mathcal{H}$ into rectangular cells. From the computation of ϵ, it is evident that the vertical (resp. horizontal) length of each cell is at least 2ϵ.

Refer to Figure 2.4. Let y_1, y_2, y_3, y_4 be the four corner points of a cell $\mathcal{C}$. Let a be the horizontal length of $\mathcal{C}$ and b be its vertical length. Note that $a \geq 2\epsilon$ and $b \geq 2\epsilon$. We want to show that there exists a new rectangle $\mathcal{H}'(v)$ such that y_1 is contained by $\mathcal{H}'(v)$, and such that all the points z in cell $\mathcal{C}$ with $x_1(z) \leq x_1(y_1) + a - \epsilon$ and $x_2(z) \leq x_2(y_1) + b - \epsilon$ lie within $\mathcal{H}'(v)$. That is, the shadow part in $\mathcal{C}$ is contained by a new rectangle $\mathcal{H}'(v)$.

We have four possible cases according to locations of y_1, as shown in Figure 2.4. We only consider Case 4, the other cases can handled analogously. In Case 4, y_1 is not on the boundary of $\mathcal{H}$. As we know, for a forbidden area $\mathcal{H}(v)$, any point that is on the boundary of $\mathcal{H}(v)$ but not on the boundary of $\mathcal{H}$, is not contained in $\mathcal{H}(v)$. Thus, a forbidden area that contains y_1 must cover the interior part of cell $\mathcal{C}$ and the interior parts of cells $\mathcal{C}_1, \mathcal{C}_2, \mathcal{C}_3$ where $\mathcal{C}_1, \mathcal{C}_2, \mathcal{C}_3$ are cells adjacent to $\mathcal{C}$, as shown in Figure 2.4(d). Note that the vertical (resp. horizontal) length of each cell is at least 2ϵ. Therefore, the new rectangle constructed from this forbidden area must cover the shadow parts in cells $\mathcal{C}, \mathcal{C}_1, \mathcal{C}_2, \mathcal{C}_3$, as desired.

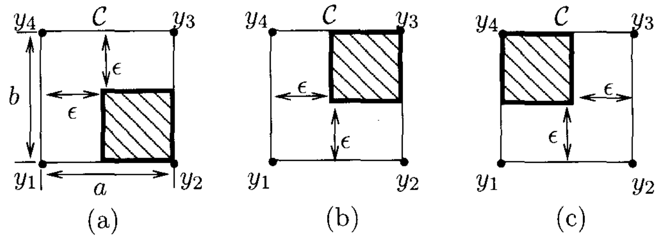


Figure 2.5: Lemma 2.2.1: (a) y_2; (b) y_3; (c) y_4.

Similarly, the above statement about y_1 is also correct for y_2, y_3, and y_4 (see Figure 2.5). Observe that the union of these four shadow rectangles cover the cell $\mathcal{C}$ (including the boundary of $\mathcal{C}$). Therefore, the measure of $\sum_{v\in V(G)} \mathcal{H}'(v)$ is equal to the measure of $\mathcal{H}$, which completes the proof of this lemma. $\square$

It takes $O(n \log n)$ time to compute the new set of rectangles $\mathcal{H}'(v)$, described in Lemma 2.2.1, from forbidden areas $\mathcal{H}(v), v \in V(G)$. Also, it has been shown [63] that, given a set of n axis-parallel rectangles, a 2-dimensional KMP can be solved in $O(n \log n)$ time. Therefore, a local feasibility test of weighted $A(G)/V(G)/2$ model on edges e_1, e_2 can be solved in time $O(n \log n)$. Thus, we have the following theorem.

Theorem 2.2.2 *The unweighted/weighted $A(G)/V(G)/2$ center problems can be solved in $O(m^2 n \log^2 n)$ time.*

The extension of the above approach to the case where $p > 2$ is straightforward. Now a local p-center solution is represented as a point in a bounded p-dimensional box (p-box) $\mathcal{H}'$ and for each demand vertex v, we obtain a p-box in $\mathcal{H}'$ containing all p-center solutions that serve v with a service cost $> r$. Thus, the local feasibility test on edges $e_1, \ldots, e_p$, is transformed into the following p-dimensional Klee's measure problem: *does the union of $O(n)$ axis-parallel p-boxes cover $\mathcal{H}'$?* It is known [63] that the measure of the union of n axis-parallel p-boxes ($p \geq 2$) can be computed in $O(n^{p/2} \log n)$ time. Therefore, we have the following lemma.

Lemma 2.2.3 *A local feasibility test of the weighted $A(G)/V(G)/p$ problem on p edges $e_1, \cdots, e_p$ can be solved in $O(n^{p/2} \log n)$ time, for $p > 1$.*

This establishes Theorem 2.1.3.

2.3 Summary

An efficient algorithm for the unweighted/weighted continuous p-center problem in a general network, where p is an arbitrary fixed constant, is presented in this chapter. The worst-case time complexity of our algorithm is $O(m^p n^{p/2} \log^2 n)$. This is an improvement over the existing result on this problem by a factor of almost $O(n^{p/2})$ [69]. Further reduction in the time complexity of the problem is an interesting open problem.

For the general p-center problem in which the demand set contains all points of the underlying network, a candidate set containing the optimal solution value is characterized in Tamir's paper [68]. In spite of the nice structure, the size of this set is not polynomial even for simple structures such as cactus networks. Until now, no efficient algorithm is known for the problem in a general network. It is a challenge to design an efficient algorithm to solve the problem even for a relatively small p.

Chapter 3

Weighted p-center problems in tree networks

In this chapter, we are concerned with the restriction of p-center problems to tree networks where the demand points are located at vertices and each demand point is associated with a non-negative weight, including the weighted discrete and continuous p-center problems. Here we only consider the case where p is a fixed constant.

For the case when $p = 1$, Megiddo [54] used a prune-and-search technique to solve the weighted 1-center problems in linear time. The problem of generalizing the trimming approach to solve the p-center problem for $p > 1$ was open for over twenty years. We propose an optimal algorithm for the weighted p-center problems when p is a fixed constant. It is a nontrivial generalization of Megiddo's prune-and-search approach [54]. This result partially resolves the long standing open problem. Moreover, we introduce a simple parametric-pruning approach for the weighted 1-center problem, which can be adapted to solve the weighted p-center problem on the real line in linear time for any fixed value p.

Organization of the chapter Notations and definitions are provided in Section 3.1. In Section 3.2 Megiddo's prune-and-search approach for the weighted 1-center problem is reviewed. We also give a parametric-pruning approach for the 1-center problem in Section 3.2. Sections 3.3 and 3.4 provide the main results of this chapter - linear-time algorithms to solve the weighted p-center problems in a tree network and on the real line for a fixed value p. Finally, Section 3.5 gives a brief conclusion.

3.1 Notation and definitions

For any real x, $\lceil x \rceil$ denotes the smallest integer that is at least x and $\lfloor x \rfloor$ denotes the largest integer bounded above by x.

We denote the underlying tree network by $T = (V(T), E(T), w, l)$, or simply by T. Recall that the demand set is denoted by $\mathcal{D}(T)$ and the supply set is denoted by $\mathcal{X}(T)$. In this chapter, we only consider the problems where $\mathcal{D}(T) = V(T)$.

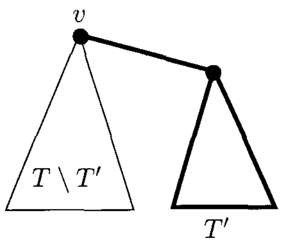
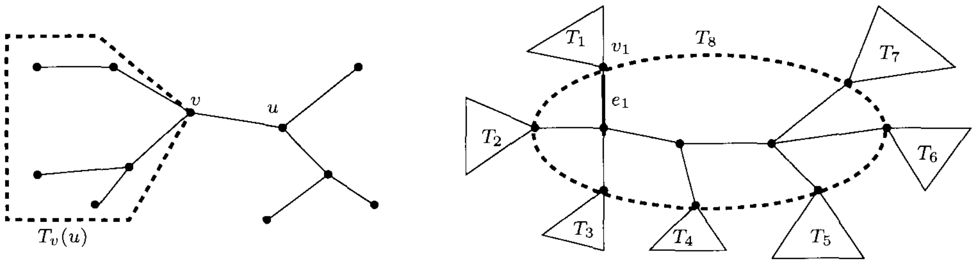


Figure 3.1: Subtree T' is anchored to vertex v.

Let $T(V')$ be the induced subtree with vertex set $V' \subseteq V$. For a subtree T' of T, let $\delta_{T'}(v)$ be the degree of vertex v in T'. By a *leaf* of T' we mean a vertex v with $\delta_{T'}(v) = 1$. A subtree T' is *anchored* to a vertex v with respect to T if v is a leaf of T' and $\delta_{T'}(u) = \delta_T(u)$, for any $u \in V(T') \setminus \{v\}$. In Figure 3.1, the bold part is T' that is anchored to vertex v with respect to T. The vertex v is called the *anchor vertex* of T'.



(a) $T_v(u)$ (b) Real subtree and core subtree

Figure 3.2: Examples for $T_v(u)$, real subtree, and core subtree.

Let $V_v(u)$ ($v \neq u$) denote the set of vertices v' such that the vertex v lies on the simple path from the vertex u to v', i.e., $v \in \pi(u, v')$. Let $T_v(u)$ denote the induced subtree rooted at v with the vertex set $V_v(u)$ as demonstrated in Figure 3.2(a). A subtree T' is called a *real subtree* of T if the component $T \setminus T'$ is connected. The vertex of a real subtree T' which is closest to $T \setminus T'$ is called *the root* of T', and the edge linking T' and $T \setminus T'$ is called *the root-edge* of T'. For example, $T_1, \ldots, T_7$ in Figure 3.2(b) are real subtrees; v_1 is the root of T_1, and e_1 is the root-edge of T_1. A subtree T' is called a *core subtree* of T if for $v \in V(T')$, either $\delta_{T'}(v) = 1$ or $\delta_{T'} = \delta_T(v)$. In Figure 3.2(b), T_8 is a core subtree.

For a $\mathcal{X}(T)/V(T)/p$ problem, a vertex u ($\in V(T)$) is called a *dominating vertex* of a set $X(\subseteq \mathcal{X}(T))$ of p centers if $F(X, \{u\}) = F(X, V(T))$.

Definition of split-edges Let $X = \{\alpha_1, \ldots, \alpha_p\} \subset A(T)$ be a set of p centers in T. Let $V_i \subseteq V(T)$ be the set of vertices closest to a particular center $\alpha_i \in X$ (ties are broken in such a way that $T(V_i)$ remains connected where $T(V_i)$ is a subtree induced by V_i). The edges whose endpoints belong to different subtrees $T(V_i)$ are called *split-edges*. Thus, locating p centers in a tree is equivalent to finding a set of split-edges whose removal defines p connected components such that the maximum service cost of the 1-centers of these components is equal to the optimal p-center cost of the entire tree.

It is trivial that the number of split edges is $p-1$ for the $V(T)/V(T)/p$ or $A(T)/V(T)/p$ problems.

3.2 Weighted 1-center problems

The weighted 1-center problem is to locate a center in $\mathcal{X}(T)$ (two cases are considered, that is, either $\mathcal{X}(T) = A(T)$ or $\mathcal{X}(T) = V(T)$) such that the maximum weighted distance from the vertices of T to the center is minimized. Since our approach for weighted p-center problems is a generalization of Megiddo's method [54], we begin with a brief introduction of Megiddo's linear-time method for the weighted $A(T)/V(T)/1$ problem, followed by our parametric pruning approach, which also runs in linear time. The extensions of these techniques to solve the $V(T)/V(T)/1$ problem in linear time are not hard to obtain.

3.2.1 Megiddo's approach for the $A(T)/V(T)/1$ model

Megiddo [54] proposed a prune-and-search algorithm for the weighted $A(T)/V(T)/1$ problem, which is carried out in two phases. The first phase is to locate a subtree network T', containing an optimal 1-center, that is anchored to a centroid vertex o of T (refer to Figure 3.3(a)). It is easy to see that the optimal 1-center provides services to all the clients in

T outside T' (i.e., $T \setminus T'$) through the vertex o. Therefore, the topology of the subtree network $T \setminus T'$ is not important. For each vertex in $T \setminus T'$, we only need to keep its distance information to o. We call a subtree of T a *big component* if it contains a constant fraction of the vertices of T. The subtree $T \setminus T'$ is a big component since $|V(T \setminus T')| \geq n/2$.

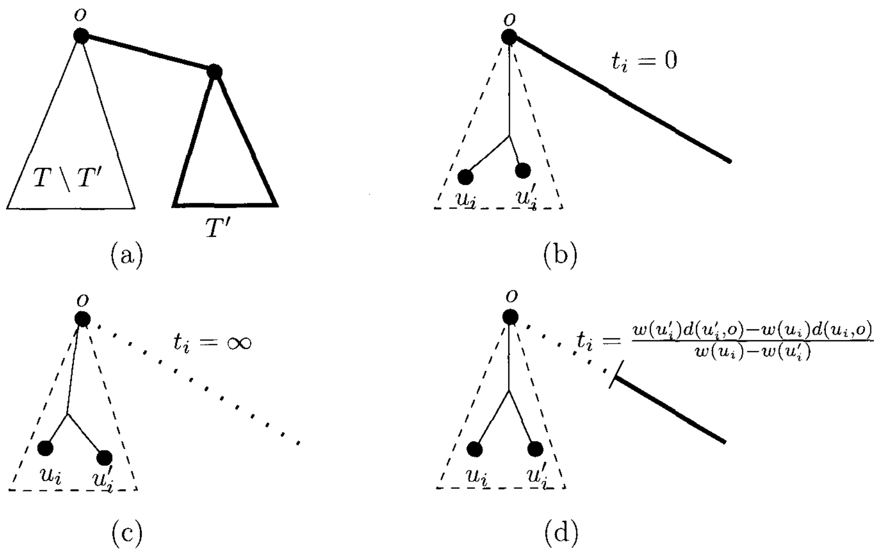


Figure 3.3: Megiddo's method: (a) Subtree T' contains an optimal 1-center and $|V(T')| \leq n/2$; (b) $w(u_i) \geq w(u_i')$ and $w(u_i)d(u_i, o) \geq w(u_i')d(u_i', o)$; (c) $w(u_i) = w(u_i')$ and $d(u_i, o) < d(u_i', o)$; (d) $w(u_i) > w(u_i')$ and $w(u_i)d(u_i, o) < w(u_i')d(u_i', o)$.

In the second phase the following *key question* is answered: determine whether there is an optimal 1-center in T' within distance t to o. We call t a *critical distance*. An appropriate value of t is determined in the following way. We arbitrarily pair the vertices in $T \setminus T'$. Let $(u_1, u_1'), (u_2, u_2'), \ldots, (u_g, u_g')$ be the pairs where $w(u_i) \geq w(u_i')$ for any $i, 1 \leq i \leq g$. For every such pair $(u_i, u_i'), 1 \leq i \leq g$, if $w(u_i)d(u_i, o) \geq w(u_i')d(u_i', o)$ then let $t_i = 0$; or else if $w(u_i) = w(u_i')$ and $d(u_i, o) < d(u_i', o)$, then let $t_i = \infty$; otherwise (i.e., $w(u_i) > w(u_i')$ and $w(u_i)d(u_i, o) < w(u_i')d(u_i', o)$), let $t_i = [w(u_i')d(u_i', o) - w(u_i)d(u_i, o)]/[w(u_i) - w(u_i')]$. The significance of t_i is explained as follows. For any point y lying in T', if its distance to o is less than t_i (i.e., $d(y, o) < t_i$) then $w(u_i)d(u_i, y) < w(u_i')d(u_i', y)$; otherwise (i.e., $d(y, o) \geq t_i$), $w(u_i)d(u_i, y) \geq w(u_i')d(u_i', y)$. In Figures 3.3(b), 3.3(c), and 3.3(d), the dotted part represents the set of points whose distance to o is less than t_i and the bold part represents the set of points whose distance to o is at least t_i. We let t be a median of these

t_i values $(1 \le i \le g)$.

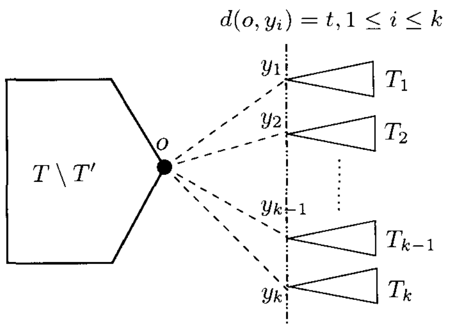


Figure 3.4: Answer the key question: does there exist an optimal 1-center in T' within distance t to o.

If there is an optimal 1-center in T' within distance t to o then, for any pair (u_i, u_i') with $t_i \ge t$, u_i is not a dominating vertex in an optimal solution $(1 \le i \le g)$; otherwise (there is an optimal 1-center in T' with distance $> t$ to o), for any pair (u_i, u_i') with $t_i \le t$, u_i' is not a dominating vertex in an optimal solution $(1 \le i \le g)$. Hence, once the answer to the key question is known, approximately $1/4$ of the vertices in $T \setminus T'$ cannot be dominating vertices in an optimal solution, and therefore can be discarded.

Refer to Figure 3.4. The approach to answer the key question is described as follows. Let $y_1, \ldots, y_k$ be the points in T' such that $d(y_i, o) = t, 1 \le i \le k$. For each $i, 1 \le i \le k$, we denote by T_i the subtree rooted at y_i. Without loss of any generality, assume that $F(y_1, V(T_1)) \ge F(y_i, V(T_i))$ for any $i, 2 \le i \le k$. It is straightforward that if there is an optimal 1-center in T' with distance $> t$ to o, then T_1 contains an optimal 1-center. Therefore, the answer to the key question can be achieved by checking if T_1 contains an optimal 1-center. The checking can be done in linear time.

The algorithm performs $O(\log n)$ above iterations. Each iteration takes linear time, linear in the size of the current tree. Therefore, the continuous 1-center problem can be solved in linear time.

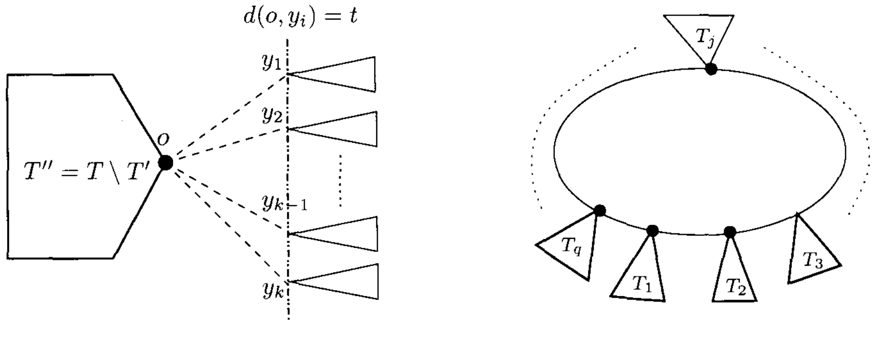
3.2.2 A parametric-pruning approach for the $A(T)/V(T)/1$ model

In this section, we present a *parametric-pruning* approach for the weighted continuous 1-center problem in T. Our approach is also carried out in two phases. The first phase is same as the one of Megiddo's approach[54]. The main difference between our approach and

Megiddo's approach is in the second phase. In the following, we present the second phase of our parametric-pruning approach.

In the second phase, we still arbitrarily pair the vertices in $T \backslash T'$. Let $(u_1, u'_1), (u_2, u'_2), \ldots, (u_g, u'_g)$ be the pairs where $w(u_i) \geq w(u'_i)$. For every such pair $(u_i, u'_i), 1 \leq i \leq g$, if $w(u_i)d(u_i, o) \geq w(u'_i)d(u'_i, o)$ (see Figure 3.3(b)) then u'_i cannot be a dominating vertex in an optimal solution; else if $w(u_i) = w(u'_i)$ and $d(u_i, o) < d(u'_i, o)$ (see Figure 3.3(c)), then u_i cannot be a dominating vertex in an optimal solution. Therefore, we can immediately discard one non-dominating vertex from such a pair that belongs to one of these two cases.

All the other pairs (u_i, u'_i) satisfy that $w(u_i) > w(u'_i)$ and $w(u_i)d(u_i, o) < w(u'_i)d(u'_i, o)$. For every remaining pair (u_i, u'_i), we compute $t_i = [w(u'_i)d(u'_i, o) - w(u_i)d(u_i, o)] / [w(u_i) - w(u'_i)]$. It is easy to see that $w(u_i) \cdot (d(u_i, o) + t_i) = w(u'_i) \cdot (d(u'_i, o) + t_i)$. Let $c_i = w(u_i) \cdot (d(u_i, o) + t_i)$, which is called the *switch service cost* of (u_i, u'_i). Let c^*_1 denote the optimal service cost of the $A(T)/V(T)/1$ problem. If c^*_1 is larger (resp. no more than) than c_i then u'_i (resp. u_i) cannot be a dominating vertex. Let c be a median of these switch service costs c_i, called *critical service cost*. We can find either $c^*_1 > c$ or $c^*_1 \leq c$ after solving the following feasibility test problem: does there exist a point $y \in A(T)$ such that $F(y, V(T)) \leq c$? We know that one feasibility test in a tree network can be solved in linear time [56], linear in the size of the current underlying tree. Therefore, the pruning of the vertices in $T \setminus T'$ can be performed using this parametric-pruning method.



(a) Megiddo's method (b) parametric-pruning method

Figure 3.5: Comparison between Megiddo's method and parametric-pruning method

The advantage of our parametric-pruning approach over Megiddo's approach is shown as follows. In Megiddo's method, one big component needs to be located, i.e. T'' in Figure

3.5(a), which is served by same center lying outside of T'' so that at least a constant fraction of the vertices in T'' can be identified for pruning. Here the center facility provides services to the demand points in T'' through the vertex o only. However, the parametric-pruning method still works even if q disjoint components, $q = O(n)$, (see Figure 3.5(b)) are being served by same center from the outside. We will see this point in its application to solve conditional extensive facility location problems in trees (in Chapter 5).

3.3 Weighted p-center problems (fixed p)

The problem of generalizing Megiddo's prune-and-search approach [54] to solve the p-center problem for $p > 1$ was open for a long time. In [7], we used the interactions between two centers and a split-edge to guide us in trimming the underlying tree, and proposed an optimal algorithm for solving the weighted 2-center problems (i.e., $A(T)/V(T)/2$ and $V(T)/V(T)/2$). In this section, we generalize the approaches proposed in [7, 54] to solve the weighted p-center problem when p is a fixed constant.

A new type of problems, called *constrained (s,t)-center problems*, is described as follows. In a constrained (s,t)-center problem, locations of t facilities are already known, and the objective is to compute the locations of s new facilities such that the maximum weighted distance from demand points to the t existing facilities and s new facilities is minimized. There are $s+t-1$ split-edges in a solution of a constrained (s,t)-center problem. We have the following constraints in a constrained (s,t)-center problem. A collection of $q(1 \leq q < s+t)$ pairwise disjoint subtrees, called *split-edge subtrees*, is given such that all $s+t-1$ split-edges are distributed among them, and we are also given another collection of $q'(0 \leq q' \leq s+t)$ pairwise disjoint subtrees whose center areas are predetermined. The formal definition of constrained (s,t)-center problems is presented in Section 3.3.1.

It is easy to see that a weighted p-center problem in a tree T can be reformulated as a constrained $(p,0)$-center problem, in which $p-1$ split-edges lie in T and no subtree has its center area predetermined. We show in the remaining part of this section that a constrained (s,t)-center problem in T can be solved in linear time in an incremental way when both s and t are fixed constants (Theorem 3.3.19). Therefore, we have the following theorem.

Theorem 3.3.1 *A weighted p-center problem in a tree can be solved in linear time, for any fixed p.*

3.3.1 Formal definition of constrained (s,t)-center problems

A constrained (s,t)-center problem is actually a conditional center problem with extra restrictions. In the following, we focus on the continuous version of the problem. Its discrete version can be defined similarly. Also, an optimal solution for the discrete version can be obtained in a very similar way and therefore is omitted here. A constrained (s,t)-center problem R is expressed as:

$$R : \langle T; s; \{\beta_{s+1}, \dots, \beta_{s+t}\}; \{T_1^e, \dots, T_q^e\}; \{(\tilde{T}_1, T_1^c), \dots, (\tilde{T}_{q'}, T_{q'}^c)\}\rangle.$$

The parameters in a constrained (s,t)-center problem include:

- T: the underlying tree network;
- s: the number of new facilities to be opened;
- $\beta_{s+1}, \dots, \beta_{s+t}$: t existing facilities in $A(T)$. An edge in T contains at most two existing facilities, and if there are two existing facilities on an edge then the two existing facilities are located at the endpoints of the edge;
- $T_i^e, 1 \le i \le q$: subtrees of T, called *split-edge* subtrees. A subtree is called a split-edge subtree if it contains at least one split-edge. All $s+t-1$ split-edges lie in these split-edge subtrees. The following invariants are always maintained. For each $i, 1 \le i \le q$, T_i^e is either a path or a core subtree. For T_i^e and T_j^e with $i \ne j$, $E(T_i^e) \cap E(T_j^e) = \emptyset$. It is not hard to see that $q \le s+t-1$. Let $N^s(T_i^e)$ denote the number of split-edges lying in $T_i^e, 1 \le i \le q$. A subtree T' is *clear* if $E(T') \cap E(T_i^e) = \emptyset, i = 1, \dots, q$, that is, $V(T')$ is served by the same 1-center;
- $(\tilde{T}_i, T_i^c), 1 \le i \le q'$: For each $i, 1 \le i \le q'$, $\tilde{T}_i$ is a clear subtree. The center serving $V(\tilde{T}_i)$ lies in subtree T_i^c, and $A(\tilde{T}_i) \cap A(T_i^c) \ne \emptyset$. We call T_i^c the *center subtree* of $\tilde{T}_i$ and $\tilde{T}_i$ the *Center-Area-Predefined* (for short, *CAP*) subtree. Moreover, the following conditions are also satisfied:
  - For any existing facility β_j $(s+1 \le j \le s+t)$, there is a CAP subtree which it serves, that is, $\forall j \in [s+1, s+t]\ \exists i\ (1 \le i \le q') \wedge (T_i^c = \beta_j)$;
  - $A(\tilde{T}_i) \cap A(\tilde{T}_j) = \emptyset$, $1 \le i < j \le q'$;

- There is at least one split-edge subtree that is a path between any two different CAP subtrees. In other words, different CAP subtrees are served by different centers. Hence, $t \le q' \le (s+t)$ is always true.

Therefore, a weighted $A(T)/V(T)/p$ problem can be restated as the following constrained $(p, 0)$-center problem:

$$\langle T; p; \emptyset; \{T\}; \emptyset \rangle,$$

where the number of split-edges contained in T is $p-1$, i.e., $N^s(T) = p-1$. It is easy to see that, for fixed s and t, the total size of a constrained center problem is $O(|V(T)|)$.

3.3.2 The main idea and overall approach

The proposed linear-time algorithm uses the solutions of lower-order constrained center problems. That is, we assume that a linear-time algorithm is available for any constrained (x, y)-center problem, where $x \le s$ and $x + y \le s + t$ ($x < s$ if $x + y = s + t$).

Our linear-time algorithm follows the lines of Megiddo's method for weighted 1-center problems in a tree [54] (see Section 3.2.1). We call any change performed on a constrained center problem a *transfer*. In Megiddo's method [54], only one type of transfer is performed, that is, pruning vertices from the current underlying tree network. However, we also perform some other types of transfers in our algorithm, such as splitting a split-edge subtree, shrinking center subtree of a CAP subtree, and so on. One common theme behind all these different types of transfers is that, the optimal service cost remains unchanged.

Definition 3.3.2 (Safe transfer) *A transfer from a constrained center problem R_1 to a new constrained center problem R_2 is called a* safe transfer *if the optimal service cost of R_2 is equal to the optimal service cost of R_1.*

Definition 3.3.3 (Safe operation for a center) *Discarding one vertex v is called a* safe operation for a center α *if it is already known that v is served by α, and v is not the farthest weighted vertex to α.*

Similarly, we have the following definition.

Definition 3.3.4 (Safe operation for a subtree) *Discarding one vertex is a* safe operation for a subtree T' *if it is a safe operation for any center located in $A(T')$.*

For a clear subtree T' in a constrained center problem, if the vertices in T' are served by some center in T'' in an optimal solution, and if discarding $v \in V(T')$ is a safe operation for T'', then pruning v is a safe transfer. In our approach, we construct CAP components and shrink their center subtrees for this purpose.

Let $R : \langle T; s; \{\beta_{s+1}, \dots, \beta_{s+t}\}; \{T_1^e, \dots, T_q^e\}; \{(\tilde{T}_1, T_1^c), \dots, (\tilde{T}_{q'}, T_{q'}^c)\}\rangle$ be a given constrained (s,t)-center problem. The stepwise description of our approach to solve R is given below.

Step 1: Transfer R into a new constrained (s,t)-center problem R' that contains at least one big clear subtree $\tilde{T}_i$ (here a subtree T' is big if $|T'| \geq |T|/2(s+t)$). This is achieved by applying split-edge-splitting transfers (Lemma 3.3.14) a constant number of times.

Step 2: In this step, we focus on shrinking center subtree T_i^c of $\tilde{T}_i$ that is computed in Step 1. Step 2 consists of three sub-steps.

Step 2.1: Let o_i be a centroid vertex of $\tilde{T}_i$. We find the branch T_i' anchored to o_i such that it contains the center serving vertices in $\tilde{T}_i$ in an optimal solution, that is, we safely transfer the current constrained center problem to a new one in which the center subtree T_i^c of $\tilde{T}_i$ is a subtree of T_i'. Hence, at least half of the vertices in $\tilde{T}_i$ are served by some facility in T_i^c through o_i.

Step 2.2: Compute critical distance d_i as follows. We arbitrarily pair the vertices in $\tilde{T}_i \setminus T_i'$. Let $(u_1, u_1'), (u_2, u_2'), \dots, (u_g, u_g')$ be the pairs where $w(u_j) \geq w(u_j'), j = 1, \dots, g$. For every such pair $(u_j, u_j'), 1 \leq j \leq g$,

- if $w(u_j)d(u_j, o_i) \geq w(u_j')d(u_j', o_i)$ then let $x_j = 0$;
- or else if $w(u_j) = w(u_j')$ and $d(u_j, o_i) < d(u_j', o_i)$ then let $x_j = \infty$;
- otherwise (i.e., $w(u_j) > w(u_j')$ and $w(u_j)d(u_j, o_i) < w(u_j')d(u_j', o_i)$), let $x_j = [w(u_j')d(u_j', o_i) - w(u_j)d(u_j, o_i)]/[w(u_j) - w(u_j')]$.

Basically, x_j indicates the intersection of service functions of pairs of vertices $(u_j, u_j'), 1 \leq j \leq g$. We let d_i be the median of these x_j values ($1 \leq j \leq g$).

Step 2.3: We safely transfer the current constrained center problem to a new one in which either $d(o_i, A(T_i^c)) > d_i$ or $\max_{x \in A(T_i^c)} d(o_i, x) \leq d_i$ (recall that the center serving $V(\tilde{T}_i)$ lies in subtree T_i^c).

(Steps 2.1 and 2.3 are done by solving constrained $(s-1, t+1)$-center problems as described in Section 3.3.5.)

Step 3: Similar to Megiddo's method [54], discard non-dominating vertices in $\tilde{T}_i$ thus computed.

Step 4: Repeat the above process until the size of the underlying tree is no more than a given constant.

It is not hard to see that after each iteration of the first three steps, at least a constant fraction of vertices is discarded. Therefore the process terminates within $O(\log n)$ iterations. If each iteration of executing the three steps can be done in linear time, then the total cost is $O(n)$ time.

The remaining part of Section 3.3 is organized as follows. Section 3.3.3 solves the base step of our incremental algorithm, that is, a linear-time algorithm is proposed for a constrained $(0, h)$-center problem where h is a fixed constant. Important properties used for splitting split-edge subtrees and shrinking center subtrees are explored in Section 3.3.4. Finally, Section 3.3.5 reviews the whole algorithm and establishes our result for constrained center problems.

3.3.3 Constrained $(0, h)$-center problems

From the definition of a constrained (s, t)-center problem, we know that the number q' of CAP components in it is between t and $s+t$. Therefore, there are exactly h CAP components in a constrained $(0, h)$-center problem R, as shown in the following.

$$R : \langle T; 0; \{\beta_1, \cdots, \beta_h\}; \{T_1^e, \ldots, T_q^e\}; \{(\tilde{T}_1, \beta_1), \ldots, (\tilde{T}_h, \beta_h)\}\rangle,$$

where $\beta_1, \cdots, \beta_h$ are h existing centers. In this section, we present an algorithm for problem R which runs in linear time and space.

We observe that T consists of three types of pairwise edge-set disjoint subtrees: CAP subtrees $\tilde{T}_i, 1 \le i \le h$ (type-1), split-edge subtrees $T_j^e, 1 \le j \le q$ (type-2), and other clear subtrees whose center areas are not yet predefined (type-3). Note that type-1 and type-3 subtrees are clear subtrees. All these subtrees are glued together with *hinge vertices*, where a hinge vertex is a common vertex contained in at least two different types of subtrees. Since a clear subtree cannot be connected to another clear subtree, type-1 and type-3 subtrees are separated by type-2 subtrees.

For any clear subtree, such as a type-1 or type-2 subtree, its vertices are served by same center in a solution. Furthermore, the center serving a type-1 subtree is predefined, i.e., vertices in $\tilde{T}_i, 1 \leq i \leq h$, will choose β_i to obtain service. We divide T into a collection of subtrees, denoted by $\mathcal{S}(T)$, by considering type-1 subtrees as cuts. Figure 3.6 demonstrates an example of T that is divided into two subtrees T_1 and T_2 by type-1 subtrees. Type-1 subtrees are represented by bold circles with a point inside, type-2 subtrees are represented by dashed ellipses (each type-2 subtree contains a number denoting the number of split-edges), and type-3 subtrees are represented by solid circles.

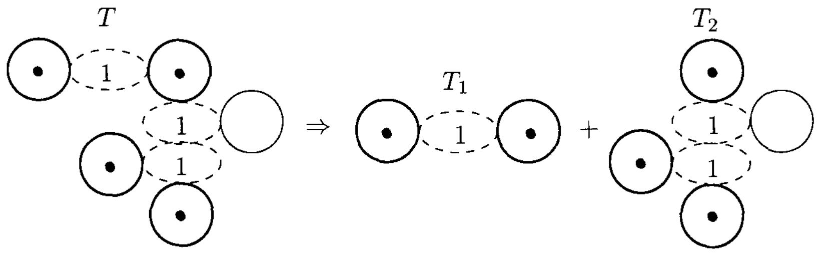


Figure 3.6: T is divided into T_1 and T_2 by type-1 subtrees, i.e., $\mathcal{S}(T) = \{T_1, T_2\}$.

Observation 3.3.5 *Each subtree in $\mathcal{S}(T)$ contains at least one type-2 subtree.*

Observation 3.3.6 *For each subtree T' in $\mathcal{S}(T)$, a type-1 subtree is attached to at most one type-2 subtree in T'. Let Φ_1 be the set of type-1 subtrees in T' and Φ_2 be the set of type-2 subtrees in T'. Then,*

$$|\Phi_1| = \sum_{T_i^e \in \Phi_2} N^s(T_i^e) + 1.$$

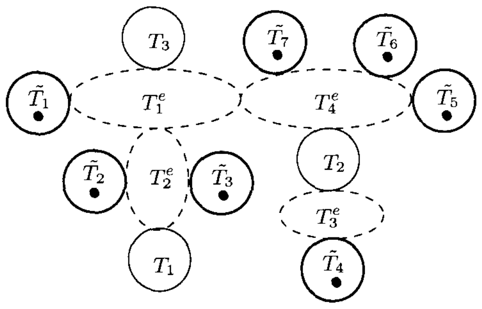
Observation 3.3.7 *The combination of optimal solutions of subtrees in $\mathcal{S}(T)$ is an optimal solution of T since any two subtrees in $\mathcal{S}(T)$ are separated by a type-1 subtree.*

According to Observation 3.3.7, it is sufficient to show an algorithm to find an optimal solution of a subtree in $\mathcal{S}(T)$. In Figure 3.7, an example of a subtree T' in $\mathcal{S}(T)$ is demonstrated. Note that type-1 subtree $\tilde{T}_i$ is served by $\beta_i, i = 1, \cdots, 7$.

Next, we present a simple method to find an optimal solution of a subtree $T' \in \mathcal{S}(T)$.

For each type-2 subtree T_i^e in T', let $N^f(T_i^e)$ be the number of type-1 subtrees attached to T_i^e. For example, $N^f(T_1^e) = 1, N^f(T_2^e) = 2, N^f(T_3^e) = 1$, and $N^f(T_4^e) = 3$ in Figure 3.7.

Lemma 3.3.8 *For each type-2 subtree $T_i^e (1 \leq i \leq q)$ in T', $N^f(T_i^e) \leq N^s(T_i^e) + 1$.*



$N^s(T_1^e) = 1$
$N^s(T_2^e) = 1$
$N^s(T_3^e) = 1$
$N^s(T_4^e) = 3$

Figure 3.7: An example of a subtree T' in $\mathcal{S}(T)$.

Proof If a type-1 subtree $\tilde{T}_j(1 \leq j \leq h)$ is attached to T_i^e, then at least one vertex in T_i^e is served by β_j, i.e., the hinge vertex between $\tilde{T}_j$ and T_i^e. In other words, vertices in T_i^e are served by at least $N^f(T_i^e)$ existing facilities. Therefore, T_i^e contains at least $N^f(T_i^e) - 1$ split-edges. □

Lemma 3.3.9 *For a type-2 subtree $T_i^e(1 \leq i \leq q)$ in T' where $N^f(T_i^e) = N^s(T_i^e) + 1$, let Δ_i be the set of existing facilities contained in type-1 subtrees attached to T_i^e, i.e., $\Delta_i = \{\beta_j | \tilde{T}_j$ is attached to $T_i^e\}$. Then, all vertices in $V(T_i^e)$ are served by facilities in Δ_i.*

Proof The proof of this lemma is very similar to the one of Lemma 3.3.8. If a connected subtree is served by k facilities in a solution, then it contains exactly $k - 1$ split-edges in that solution. Since $N^f(T_i^e) = N^s(T_i^e) + 1$, all vertices in $V(T_i^e)$ are served by facilities in Δ_i. □

Lemma 3.3.10 *There exists at least one type-2 subtree $T_i^e(1 \leq i \leq q)$ in T' such that $N^f(T_i^e) = N^s(T_i^e) + 1$.*

Proof Suppose that there is no type-2 subtree T_i^e where $N^f(T_i^e) = N^s(T_i^e) + 1$. Then, according to Lemma 3.3.8, for each type-2 subtree $T_j^e(1 \leq j \leq q)$ in T', $N^f(T_j^e) < N^s(T_j^e)+1$, which contradicts with Observation 3.3.6.

Therefore, there exists at least one type-2 subtree T_i^e in T' where $N^f(T_i^e) = N^s(T_i^e) + 1$ (For example, in Figure 3.7, $N^f(T_2^e) = N^s(T_2^e) + 1$). □

Our approach is therefore as follows. At each step, we find a type-2 subtree T_i^e in T' where $N^f(T_i^e) = N^s(T_i^e) + 1$, $1 \leq i \leq q$. Then we locate $N^s(T_i^e)$ split-edges in T_i^e, according to the locations of existing facilities in type-1 subtrees attached to T_i^e. Once split-edges are located, the facility serving each vertex in T_i^e is determined. Therefore, T_i^e is divided into pieces and each piece is merged into a type-1 subtree. The approach is repeated until no type-2 subtree is left in T'.

Since q and h are constants, we have the following result.

Lemma 3.3.11 *The optimal service cost of a constrained* $(0, h)$*-center problem in a tree can be computed in linear time, if* h *is a fixed constant.*

3.3.4 Important properties

We assume here that while solving a constrained (s,t)-center problem the solution to any constrained (x, y)-center problem with $x + y \leq s + t$ and $x \leq s$ ($x < s$ if $y = t$) is known.

In this section we explore several important properties that are crucial in our algorithm. Based on them, we can split split-edge subtrees to locate a big clear subtree, and shrink center subtrees to prune non-dominating vertices in the big subtree.

In a constrained (s,t)-center problem $R :< T; s; \beta_{s+1}, \ldots, \beta_{s+t}; \{T_1^e, \ldots, T_q^e\}; \{(\tilde{T}_1, T_1^c), \ldots, (\tilde{T}_{q'}, T_{q'}^c)\} >$, we pick up a vertex v from a split-edge subtree $T_i^e (1 \leq i \leq q)$. Let e ($\overline{uv}$) be an edge in $E(T_i^e)$ that is incident to v. We divide R into two subproblems, i.e., R_1 and R_2, by considering e as a split-edge. As illustrated in Figure 3.8(a), the underlying tree of R_1 is T_1 and the underlying tree of R_2 is T_2. We denote by T_k' the intersection subtree of T_i^e and $T_k, k = 1, 2$. Other $N^s(T_i^e) - 1$ split-edges in T_i^e are distributed among T_1' and T_2'. Assume that T_1' contains j split-edges with $|E(T_1')| \geq j$, $0 \leq j \leq N^s(T_i^e) - 1$. Then T_2' contains $N^s(T_i^e) - j - 1$ split-edges (note that $|E(T_2')| \geq N^s(T_i^e) - j - 1$). Let t_k be the number of existing facilities in T_k and let s_k be the total number of split-edges of split-edge subtrees (except T_i^e) in $T_k, k = 1, 2$. Therefore, we need to locate $s_1 + j$ split-edges in T_1 and $s_2 + N^s(T_i^e) - j - 1$ split-edges in T_2. We have the following lemma.

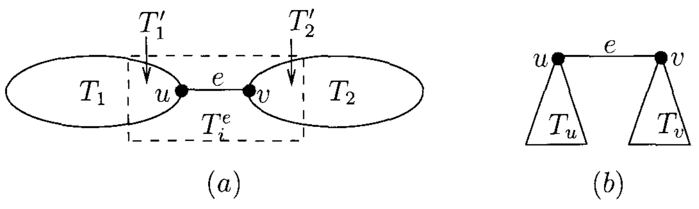


Figure 3.8: Lemma 3.3.12 and Lemma 3.3.13.

Lemma 3.3.12 *If the optimal service cost of the constrained $(s_1 + j - t_1 + 1, t_1)$-center problem R_1 is larger than or equal to the optimal service cost of the constrained $(s_2 + N^s(T_i^e) - j - t_2, t_2)$-center problem R_2, then $E(T_1') \cup \{e\}$ contains at least $j + 1$ split-edges in an optimal solution of R. Otherwise, $E(T_2') \cup \{e\}$ contains at least $N^s(T_i^e) - j$ split-edges in an optimal solution of R, in other words, $E(T_1') \cup \{e\}$ contains at most $j + 1$ split-edges.*

Lemma 3.3.13 *If it is known that $E(T_1') \cup \{e\}$ contains $j+1$ split-edges and if the optimal service cost of the constrained $(s_1 + j - t_1 + 1, t_1)$-center problem R_1 is smaller than the optimal service cost of the constrained $(s_2 + N^s(T_i^e) - j - t_2, t_2)$-center problem R_2, then edge e is an optimal split-edge in an optimal solution of R.*

We show Lemma 3.3.12 and Lemma 3.3.13 by a simpler example. In Figure 3.8(b), an edge $e : \overline{uv}$ separates the underlying tree into two subtrees T_u and T_v. If the service cost of an optimal 1-center solution of T_u is larger (resp. smaller) than or equal to the service cost of an optimal 1-center solution of T_v, then, in an optimal 2-center solution, the optimal split-edge lies in $E(T_u) \cup \{e\}$ (resp. $E(T_v) \cup \{e\}$) since it is impossible to have a solution with a split-edge in T_v (resp. T_u) that has a smaller service cost. For Lemma 3.3.13, if it is known that $E(T_u) \cup \{e\}$ contains an optimal split-edge and the service cost of an optimal 1-center solution of T_u is smaller than the service cost of an optimal 1-center solution of T_v, then e is an optimal split-edge.

Lemma 3.3.14 (Split Lemma) *Given a constrained (s,t)-center problem $R :< T; s; \beta_{s+1}, \ldots, \beta_{s+t}; \{T_1^e, \ldots, T_q^e\}; \{(\tilde{T}_1, T_1^c), \ldots, (\tilde{T}_{q'}, T_{q'}^c)\} >$ and a vertex v in T, either we solve R or we can safely transfer R into a new constrained (s,t)-center problem in which v is not an internal vertex in any split-edge subtree, and the process takes linear time.*

Proof Since split-edge subtrees are pairwise edge-set disjoint, either v is not an internal vertex of any split-edge subtree or v is an internal vertex of some split-edge subtree. In the former case, R itself is the desired constrained center problem. Assume that v is an internal vertex of split-edge subtree $T_k^e (1 \le k \le q)$ where $N^s(T_k^e) = h, 1 \le h \le s + t - 1$.

If T_k^e is a core subtree, then v does not lie in any $\tilde{T}_i, 1 \le i \le q'$, since $\tilde{T}_i$ is a clear subtree. Let $T_1, \ldots, T_b$ be the subtrees anchored to v such that each of them does not contain $\tilde{T}_i (1 \le i \le q')$ or $T_j^e (1 \le j \ne k \le q)$. Subtrees $T_1, \ldots, T_b$ do not contain any of the existing facilities. The number of remaining subtrees anchored to v is no more than

$s + t - 1 - h + q'$, since there are at most $s + t - h$ split-edge subtrees (including T_k^e) in R, i.e., $q \leq s + t - h$.

We first show that at most $th + h + 1$ subtrees among $T_1, \ldots, T_b$ need to be considered that may contain split-edges in an optimal solution of R. Assume that $b > th + h + 1$. We consider the following two cases.

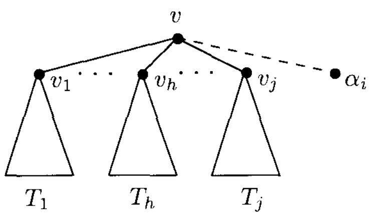


Figure 3.9: Case 1 in proof of Lemma 3.3.14: v is served by an existing facility β_i.

- **Case 1: v is served by an existing facility.** Suppose that β_i serves v, $1 \leq i \leq t$, and $F(\beta_i, V(T_1)) \geq F(\beta_i, V(T_2)) \geq \ldots \geq F(\beta_i, V(T_h)) \geq F(\beta_i, V(T_j)), j > h$. Then, in an optimal solution, $T_j, j > h$, does not contain split-edges. The reason is as follows. Assume that a subtree T_j $(j > h)$ contains γ $(\leq h)$ split-edges in a solution. Let Ω be the set of subtrees among $\{T_1, \ldots, T_h\}$ that do not contain any split-edges in this solution. Obviously, $|\Omega| \geq \gamma$ since $N^s(T_k^e) = h$. Therefore, the service cost is at least $F(\beta_i, V(T_h))$. We can see that the service cost is not increased if we replace γ split-edges in T_j by γ edges $\overline{vv_a}$ where $T_a \in \Omega$ (see Figure 3.9). Hence, we can assume that a subtree $T_j (j > h)$ does not contain split-edges in an optimal solution if v is served by β_i.

  Therefore, in the case when v is served by some existing facility, at most th subtrees might contain split-edges in an optimal solution.

- **Case 2: v is served by a newly opened facility.** Suppose that $F(v, V(T_1)) \geq F(v, V(T_2)) \geq \ldots \geq F(v, V(T_{h+1})) \geq F(v, V(T_j)), j > h + 1$. In this case, subtree $T_j, j > h + 1$ does not contain split-edges in an optimal solution. The reason is briefly described as follows. We observe that the service cost of a solution Θ, in which a subtree $T_j (j > h + 1)$ contains split-edges, is at least $F(v, V(T_{h+1}))$. Similar to the idea in Case 1, we can find another solution with the same or smaller service cost in which there is no split-edge in T_j.

Since T_j contains split-edges in Θ, there is at least one facility in T_j. We create a new solution by removing from Θ split-edges and facilities that are lying in T_j, and inserting a facility v. It is not hard to see that the service cost of the new solution is no larger than the service cost of Θ.

Thus, there exists an optimal solution in which no split-edge lies in $T_j, j = h+2, \dots, b$, if v is served by a newly opened facility $\beta_i, i \leq s$.

Therefore at most $th + h + 1$ subtrees among $T_1, \dots, T_b$ need to be considered that might contain split-edges in an optimal solution. Let Γ be the set of subtrees anchored to v that might contain split-edges. We can see that $|\Gamma| \leq th + s + t + q'$ (note that there are no more than $s + t - 1 - h + q'$ subtrees anchored to v other than $T_1, \dots, T_b$). The set Γ can be found in linear time since s, t, h, q' are fixed constants.

The second step is to compute the number of split-edges of each subtree in Γ. For each such subtree $T_i \in \Gamma$, $1 \leq i \leq b$, we consider vv_i as a split-edge. We denote by T_i' the subtree $T_i \setminus v$. Let γ_1 be the number of existing facilities in T_i' and γ_2 be the total number of split-edges of split-edge subtrees (except T_k^e) in T_i'. If $E(T_i') \cup E(T_k^e)$ contains j split-edges, then T_i' contains $\gamma_2 + j$ split-edges and $T \setminus T_i'$ contains $s + t - 2 - \gamma_2 - j$ split-edges. We compute the smallest value of $j \in [0, h-1]$ such that the optimal service cost of the constrained $(\gamma_2 + j + 1 - \gamma_1, \gamma_1)$-center problem on T_i' is smaller than or equal to the optimal service cost of the constrained $(s + \gamma_1 - \gamma_2 - j - 1, t - \gamma_1)$-center problem on $T \setminus T_i'$. This value can be computed by considering all possible values of j (if it exists). For each value of j, two subproblems need to be solved, which can be done in linear time by assumption. We have the following two cases.

- *Let j_i be the smallest value of j in $[0, h-1]$ such that the optimal service cost of the constrained $(\gamma_2 + j_i + 1 - \gamma_1, \gamma_1)$-center problem on T_i' is smaller than or equal to the optimal service cost of the constrained $(s + \gamma_1 - \gamma_2 - j_i - 1, t - \gamma_1)$-center problem on* $T \setminus T_i'$. Then, in an optimal solution of R, at least j_i split-edges of T_k^e lie in T_i and at most $j_i + 1$ split-edges of T_k^e lie in T_i.

- *For any value of j in $[0, h-1]$, the optimal service cost of the constrained $(\gamma_2 + j + 1 - \gamma_1, \gamma_1)$-center problem on T_i' is larger than the optimal service cost of the constrained $(s + \gamma_1 - \gamma_2 - j - 1, t - \gamma_1)$-center problem on $T \setminus T_i'$.* Then, in an optimal solution of R, all h split-edges of split-edge subtree T_k^e lie in T_i. We let j_i be h.

For each subtree T_i in Γ, we compute its corresponding value of j_i as described above. Since only a constant number of subtrees anchored to v might contain split-edges, i.e., $|\Gamma| \leq th + s + t + q'$, the above process of computing all j_i's can be done in linear time.

It is easy to see that $\sum_{T_i \in \Gamma} j_i \leq h$. If $\sum_{T_i \in \Gamma} j_i = h$, then the distribution of h split-edges of T_k^e among subtrees in Γ in an optimal solution is known. Thus, we update the information in the constrained center problem R accordingly, i.e. delete split-edge subtree T_k^e, insert new split-edge subtrees, and merge clear areas if they are now served by same center.

Otherwise, $\sum_{T_i \in \Gamma} j_i < h$. We already know that a subtree T_i in Γ contains at most $j_i + 1$ split-edges in an optimal solution. By using Lemma 3.3.13, if a subtree T_i in Γ contains $j_i + 1$ split-edges in an optimal solution, then $\overline{vv_i}$ is an optimal split-edge. We conclude that there are $h - \sum_{T_i \in \Gamma} j_i$ split-edges among edges $\overline{vv_i}, T_i \in \Gamma$. In this case, we are able to solve R in linear time after locating $h - \sum_{T_i \in \Gamma} j_i$ split-edges among edges $\overline{vv_i}, T_i \in \Gamma$.

In the case when T_k^e is a path, we know that h split-edges of T_k^e are distributed in two subtrees anchored to v, say T_1 and T_2. We perform the second step described above to compute the values of j_1 and j_2, that is, in an optimal solution of R, at least j_i split-edges of T_k^e lie in T_i and at most $j_i + 1$ split-edges of T_k^e lie in T_i, $i = 1, 2$. Clearly, $j_1 + j_2 \leq h$. If $j_1 + j_2 = h$, then the distribution of h split-edges of T_k^e among T_1 and T_2 in an optimal solution is known. Thus, we update the information in the constrained center problem R accordingly. Otherwise, $j_1 + j_2 < h$. In this case, there are $h - j_1 - j_2$ split-edges among edges $\overline{vv_1}$ and $\overline{vv_2}$. We solve R in linear time after locating $h - j_1 - j_2$ split-edges among edges $\overline{vv_1}$ and $\overline{vv_2}$. □

Split Lemma shows that, in linear time, we are able to compute the number of split-edges contained in each subtree anchored to v. In the following, we show that, by using Split Lemma at most $\lceil \log(s + t) \rceil$ times, we can obtain a new constrained center problem that contains a big clear subtree (it contains more than $|V(T)|/2(s + t)$ vertices).

Locate one big clear subtree

Let o be a centroid vertex of T. By using Split Lemma on o, we get the number of split-edges in each branch anchored to o. Refer to Figure 3.10(a). Let $T_1, \cdots, T_b$ be subtrees anchored to o such that each of them contains at least one split-edge. We denote by T_{b+1} the subtree that consists of all subtrees anchored to o that do not contain any split-edge.

For each subtree $T_i, 1 \leq i \leq b$, we define its *average size* as $|V(T_i)|/s_i$ if T_i contains s_i

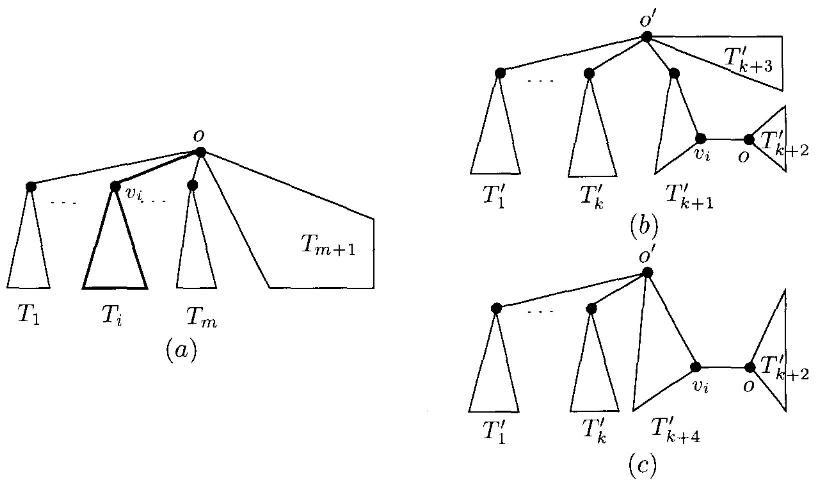


Figure 3.10: Locate one big clear subtree.

split-edges. If T_{b+1} contains at least $|V(T)|/(s+t)$ vertices then T_{b+1} is a big clear subtree. We assume that $|V(T_{b+1})| < |V(T)|/(s+t)$. Among $T_1, \cdots, T_b$, there is at least one subtree T_i with its average size $\geq |V(T)|/(s+t)$, since the total number of split-edges is $s+t-1$. Therefore, $|V(T_i)| \geq s_i \times |V(T)|/(s+t)$. Let $x_1 = s_i$. We can see that $x_1 \leq (s+t)/2$ since o is a centroid of T.

Let o' be a centroid vertex of T_i. The above process is repeated on o' and T_i. We will see a minor difference in the following description. We first apply the Split Lemma on vertex o'. Each of subtrees $T'_1, \cdots, T'_k$ contains at least one split-edge. In Figure 3.10(b)(c), the subtree T'_{k+2} is union of $T_1, \cdots, T_{i-1}, T_{i+1}, \cdots, T_{b+1}$ in Figure 3.10(a). Let T'' be the subtree anchored to o' that contains T'_{k+2}, i.e., $T'' = T'_{k+1} \cup T'_{k+2}$ in Figure 3.10(b). We denote by T'_{k+3} the subtree that consists of all subtrees anchored to o' that do not contain any split-edge. We have the following two cases.

- *Case 1:* the number of split-edges lying in T'' is larger than the number of split-edges lying in T'_{k+2}, i.e., there are split-edges in T'_{k+1} (see Figure 3.10(b)).
- *Case 2:* all split-edges of T'' lie in T'_{k+2} (see Figure 3.10(c)). In this case, T'_{k+1} is a clear subtree. Therefore, $T'_{k+4} = T'_{k+1} \cup T'_{k+3}$ is a clear subtree.

If $|V(T'_{k+3})| \geq |V(T_i)|/(x_1+1)$ in Case 1 or $|V(T'_{k+4})| \geq |V(T_i)|/(x_1+1)$ in Case 2, then a clear subtree with size $\geq |V(T)|/2(s+t)$ is found, since $|V(T_i)| \geq x_1 \times |V(T)|/(s+t)$ and

x_1 is a positive integer. Otherwise, for each subtree T'_j ($1 \leq j \leq k+1$ in Case 1, $1 \leq j \leq k$ in Case 2), we compute its average size. There is at least one subtree T'_j with its average size $\geq |V(T_i)|/(x_1+1)$. We denote by x_2 the number of split-edges contained in T'_j. With the same reason for $x_1 \leq (s+t)/2$, $x_2 \leq (x_1+1)/2$.

We continue the process on T'_j and its centroid vertex. Obviously, this process terminates in $\lceil \log(s+t) \rceil$ steps and we get a clear subtree whose size is at least $|V(T)| \cdot [x_1/(s+t)] \cdot [x_2/(x_1+1)] \cdot \ldots \cdot 1/2$, where $x_i \leq [(x_{i-1}+1)/2]$ with $x_0 = s+t-1$. This value is at least $|V(T)|/2(s+t)$ (it can be proved by induction). Therefore, we have the following lemma.

Lemma 3.3.15 *Given a constrained (s,t)-center problem, a clear subtree whose size is at least $|V(T)|/2(s+t)$ can be found in linear time.*

The following lemma is an extension of the Split Lemma.

Lemma 3.3.16 *Given a set Φ of disjoint real subtrees in a constrained (s,t)-center problem, we can safely transfer R into a new constrained (s,t)-center problem, in which the root vertices of real subtrees in Φ are not internal vertices of any split-edge subtree, and the process takes linear time.*

Proof Let Φ' be the set of real subtrees in Φ whose roots are internal vertices of split-edge subtrees. Similar to the first step in the proof of the Split Lemma, we show that only a constant number of real subtrees in Φ' are candidates to contain split-edges in an optimal solution and that they can be found in linear time.

If a real subtree in Φ' contains some existing facility or some split-edge subtree, then apply the Split Lemma to its root. Since there are only a constant number of such real subtrees, it can be done in linear time. Let Φ'' be the set of real subtrees in Φ' that do not contain any existing facility and any split-edge subtree. We assume that $|\Phi''| \geq s+t-1$. For each subtree T' in Φ'', we compute the service cost of an optimal 1-center solution of T'.

We observe that the first $s+t-1$ subtrees with larger optimal 1-center service cost are candidates to contain split-edges in an optimal solution, by the same reason mentioned in the proof of the Split Lemma. Therefore, we need to apply the Split Lemma only on the root vertices of the $s+t-1$ real subtrees. Note that it takes linear time to compute their 1-center service costs since they are pairwise disjoint.

As desired, after the above processing, the root of each real subtree in Φ is not an internal vertex of any split-edge subtree. The entire process takes linear time. □

Lemma 3.3.17 *Given two vertices u, v in a constrained (s,t)-center problem R, we can in linear time safely transfer it into a new constrained (s,t)-center problem, in which either u and v are served by same center or they are served by different centers.*

Proof The idea is to, in linear time, safely transfer R into a new constrained (s,t)-center problem, in which either

- $\pi(u,v)$ is clear. In this case, u and v are served by same center; or
- a split-edge path intersects $\pi(u,v)$. In this case, u and v are served by different centers.

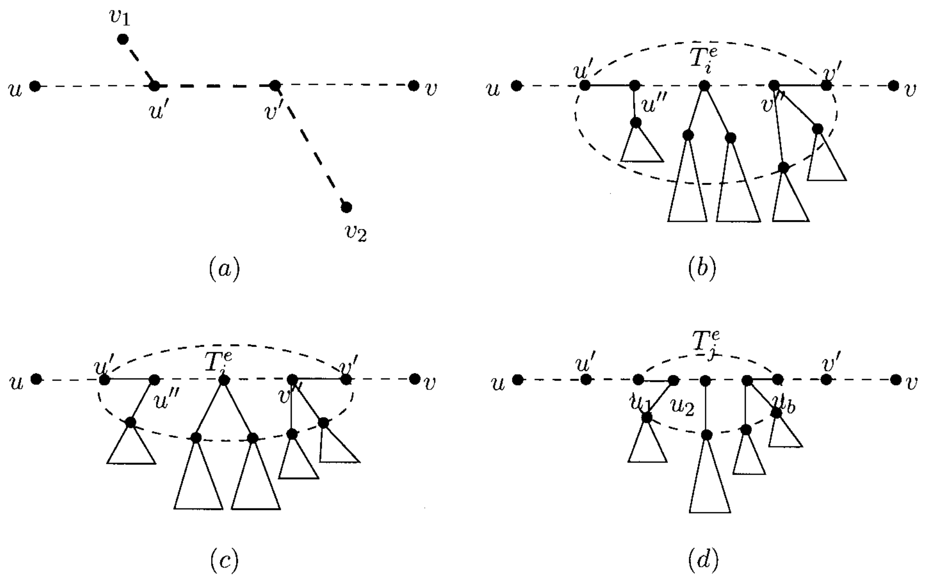


Figure 3.11: Proof of Lemma 3.3.17.

We achieve the above goal by the following steps.

Step 1: apply the Split Lemma on u and v. If $\pi(u,v)$ is clear, then terminate the process.

Step 2: we do the following for each split-edge subtree that is a path $\pi(v_1,v_2)$ and that intersects with $\pi(u,v)$, i.e., $E(\pi(v_1,v_2)) \cap E(\pi(u,v))) \neq \emptyset$. Let u' (resp. v') be the vertex in $\pi(v_1, v_2)$ closest to u (resp. v) (see Figure 3.11(a)). By applying the Split Lemma on u', v' respectively, either $\pi(u', v')$ is clear or $\pi(u', v')$ contains at least one split-edge. In the latter case, u and v are served by different centers, and we therefore terminate the process.

Step 3: we do the following for each split-edge subtree that is a core subtree T_i^e and that intersects with $\pi(u, v)$, i.e., $E(T_i^e) \cap E(\pi(u, v))) \neq \emptyset$. Let u' (resp. v') be the vertex in T_i^e closest to u (resp. v). We can see that u' and v' are leaves of T_i^e. Let u'' (resp. v'') be the vertex in $\pi(u', v')$ adjacent to u' (resp. v'). See Figure 3.11(b) for reference. Note that it is possible that $u'' = v''$.

By using Lemma 3.3.16 on the set of real subtrees hanging from $\pi(u'', v'')$, their root vertices are not internal vertices of any split-edge subtree in a new constrained center problem. In this new constrained center problem, T_i^e is updated to be a core subtree that is composed of vertices in $\pi(u'', v'')$ and vertices adjacent to $\pi(u'', v'')$ (see Figure 3.11(c)).

To each vertex in $\pi(u'', v'')$ adjacent to a real subtree that hangs from $\pi(u'', v'')$ and contains an existing facility or a split-edge subtree, we apply the Split Lemma. We note that there are only a constant number of such vertices, bounded by $s + t$. Now, T_i^e is split into pieces.

For each piece T_j^e that intersects with $\pi(u, v)$, we do the following. We assume that T_j^e is not a path. Let $u_1, \cdots, u_b$ be the vertices in T_j^e that lie on $\pi(u, v)$ (see Figure 3.11(d)). We can see that u_1 and u_b are leaves of T_j^e. Let $N^s(T_j^e) = k$.

We observe that all real subtrees hanging from $\pi(u_2, u_{b-1})$ are clear (otherwise, the Split Lemma will be applied in the above). For each real subtree T' hanging from vertex $u_h, 2 \leq h \leq b - 1$, we compute service cost $F(u_h, V(T'))$. Let $T'_1, \cdots, T'_{k+1}$ be subtrees that have the first $k + 1$ largest service costs. There is an optimal solution in which the root-edge of a subtree T' hanging from $\pi(u_2, u_{b-1})$ with $T' \neq T'_h, 1 \leq h \leq k + 1$, is not a split-edge. Note that, in the proof of the Split Lemma, we use the similar reason to show that there are only a constant number of subtrees containing split-edges.

To each vertex in $\pi(u_2, u_{b-1})$ that is adjacent to at least one of the subtrees $T'_1, \cdots, T'_{k+1}$, we apply the Split Lemma. Then, either the root-edge of a real subtree T'_h $(1 \leq h \leq$

$k+1$) is a split-edge, or all k split-edges of T_j^e lie in $\pi(u_1, u_b)$.

It is not hard to see that all of these steps can be accomplished in linear time. This completes the proof. □

Lemma 3.3.18 (Shrink Lemma) *Given a constrained (s,t)-center problem $R :< T; s; \beta_{s+1}, \ldots, \beta_{s+t}; \{T_1^e, \ldots, T_q^e\}; \{(\tilde{T}_1, T_1^c), \ldots, (\tilde{T}_{q'}, T_{q'}^c)\} >$ and an integer $i, 1 \le i \le q'$, let v be a non-leaf point in T_i^c. We can in linear time safely transfer R into a new constrained (s,t)-center problem R' in which v is a leaf point of T_i^c.*

Proof If $v \notin \tilde{T}_i$ then we first identify that either $\tilde{T}_i$ and v are served by same center or they are served by different centers. It can be done using Lemma 3.3.17 on v and any vertex in $\tilde{T}_i$.

We have two cases. If $\tilde{T}_i$ and v are served by different centers, then we are able to shrink the center subtree T_i^c of $\tilde{T}_i$ such that v is a leaf vertex of T_i^c.

Otherwise, $\tilde{T}_i$ and v are served by same center. In this case, we obtain a desired constrained center problem R' by solving a constrained $(s-1, t+1)$-center problem $R'' :< T; s-1, \beta_s = v, \ldots, \beta_{s+t};\ \{T_i^e, \ldots, T_q^e\}; \{(\tilde{T}_1, T_1^c),\ \ldots, (\tilde{T}_i, \beta_s), \ldots, (\tilde{T}_{q'}, T_{q'}^c)\} >$. By assumption, R'' can be solved in linear time. Let T' be the subtree, anchored at v, that contains the dominating vertex which realizes the optimal service cost of R'. Then, we replace T_i^c with $T_i^c \cap T'$. It is easy to see that this transfer is safe. □

3.3.5 Review of the overall approach

In this section, we review the steps of our linear-time algorithm for a constrained (s,t)-center problem R, which are described in Section 3.3.2.

In step 1, we locate a big clear subtree $\tilde{T}_i$ whose size is at least $|V(T)|/2(s+t)$. Our method for this step is presented in Section 3.3.4. The objective of step 2 is to shrink the center subtree of T' for pruning. However, we first need to make certain that either the center serving $\tilde{T}_i$ also serves another CAP subtree or the center serving $\tilde{T}_i$ does not serve any other CAP subtree, which can be done by Lemma 3.3.17.

Among the three sub-steps of step 2, steps 2.1 and 2.3 can be achieved by applying the Shrink Lemma. However, in step 2.3, there might be many points which have the distance d_i (defined in Section 3.3.2) to o_i (a centroid vertex of $\tilde{T}_i$). We cannot afford to check the points one by one using the Shrink Lemma. Instead we use the following procedure.

Let $y_1, \cdots, y_b$ be the points with the distance d_i to o_i. Let v_j be the vertex closest to y_j such that y_j lies in $\pi(v_j, o_i)$, $1 \leq j \leq b$, and let $\Phi = \{T_{v_j}(o_i), j = 1, \cdots, b\}$. By using Lemma 3.3.16, in linear time, we can obtain a new constrained center problem in which the roots of subtrees in Φ are not internal vertices of any split-edge subtree. Therefore, the number of split-edges in each subtree of Φ is known.

Let $\Phi' : \{T'_1, \ldots, T'_k\}$ be the set of real subtrees in Φ that do not contain any split-edge or CAP subtree. Here T'_j is rooted at $v_j, j = 1, \ldots, k$ (consider o_i as the root of T). It is not hard to see that there are a constant number of elements in $\Phi \setminus \Phi'$, i.e., $|\Phi \setminus \Phi'| \leq s + t$.

Without loss of generality, we assume that $F(y_1, V(T'_1)) \geq \cdots \geq F(y_{s+t}, V(T'_{s+t}))$ and $F(y_{s+t}, V(T'_{s+t})) \geq F(y_j, V(T'_j)), s + t < j \leq k$. Obviously, at least two subtrees in $\{T'_1, \ldots, T'_{s+t}\}$ are served by same center in a solution, which implies that the optimal service cost is at least $F(y_{s+t}, V(T'_{s+t}))$. Therefore, there exists an optimal solution in which the center serving $\tilde{T}_i$ does not lie in $\overline{y_j v_j} + T'_j, s + t + 1 \leq j \leq k$.

Therefore, there are at most $2(s+t)$ points in the set of $\{y_1, \ldots, y_b\}$ (including $y_1, \cdots, y_{s+t}$ and points adjacent to real subtrees in $\Phi \setminus \Phi'$) which need to be checked by using the Shrink Lemma.

In summary, it takes linear time to locate a big clear subtree, and the step of pruning a constant fraction (around 1/8) of vertices in it also takes linear time. We can find an optimal solution within $O(\log n)$ iterations. Therefore, we have the following theorem.

Theorem 3.3.19 *A constrained (s, t)-center problem in a tree can be solved in linear time, where s and t are fixed constants.*

3.4 Weighted p-center problems on the real line

In this section, we study the weighted p-center problem for points on the real line $\mathcal{L}$ when p is an arbitrary fixed constant. The input is a set $V(\mathcal{L})$ of n points lying on $\mathcal{L}$ where each point $v \in V(\mathcal{L})$ is associated with a non-negative weight $w(v)$. Let $q_L(u)$ denote the *coordinate* of a point u on $\mathcal{L}$. Clearly, for a pair of points u and v, $d(u, v) = |q_L(u) - q_L(v)|$. The continuous p-center problem on $\mathcal{L}$ is to determine a set X of p points on $\mathcal{L}$ such that $F(X, V(\mathcal{L}))$ is minimized. The discrete p-center problem on $\mathcal{L}$ is to determine a set X of p points on $V(\mathcal{L})$ such that $F(X, V(\mathcal{L}))$ is minimized. The algorithm for the discrete problem is very similar to the one for the continuous problem and therefore is omitted here.

The main difference between the problem of computing the weighted p-center for points on the real line and the weighted p-center problem in a path network is that the path topology provides the ordering of these n points, whereas no ordering information of the demand points on $\mathcal{L}$ is available.

The point in $V(\mathcal{L})$ with the smallest (resp. largest) coordinate is labeled v_1 (resp. v_n). It is easy to see that there exists an optimal solution $X^* = \{\alpha_1, \ldots, \alpha_p\}$ such that $q_L(v_1) \leq q_L(\alpha_i) \leq q_L(v_n), 1 \leq i \leq p$. A link connecting two consecutive points u, v in $V(\mathcal{L})$ is called *an edge* $e : \overline{uv}$. Let $E(\mathcal{L})$ denote the set of edges constructed from points in $V(\mathcal{L})$. We note that $E(\mathcal{L})$ is not available without the ordering information of points on $V(\mathcal{L})$. Clearly, $|E(\mathcal{L})| = n - 1$.

Given two real values x_1, x_2 $(x_1 \leq x_2)$, $[x_1, x_2]$ denotes the interval on $\mathcal{L}$ from x_1 to x_2 (inclusive); x_1 (resp. x_2) is called the *left* (resp. *right*) *endpoint* of this interval. Two subsets, say V_1 and V_2, of $V(\mathcal{L})$ are *disjoint* if the coordinate of any point in one subset is smaller than the coordinates of all the points of the other subset.

The remaining part of this section is organized as follows. Section 3.4.1 discusses the main idea of our linear-time algorithm. Then, in Section 3.4.2, we present a linear-time algorithm for the conditional 1-center problem on $\mathcal{L}$. Our algorithm for the weighted p-center problem of $V(\mathcal{L})$ on $\mathcal{L}$ is described in Section 3.4.3.

3.4.1 Main idea

Our linear-time algorithm to solve the weighted continues p-center problem on $\mathcal{L}$ is an incremental one. We assume that linear-time algorithms for any k-center problem and any conditional k-center problem (note that we only consider conditional problems in which there are a constant number of existing facilities), where $k < p$, are available.

Let $X = \{\alpha_1, \cdots, \alpha_p\}$ be a set of p centers on $\mathcal{L}$. Recall that a demand point v is called a dominating demand point of X if $F(X, v) = F(X, V(\mathcal{L}))$. Observe that, in an optimal solution, its center set always has an equal or smaller service cost to the dominating demand points of X. Let $V_i \subseteq V(\mathcal{L})$ denote the set of demand points closest to a particular center $\alpha_i \in X, i = 1, \cdots, p$. Clearly, these subsets are mutually disjoint. Hence, there is at least one subset $V_i (1 \leq i \leq p)$ that contains at least n/p demand points in $V(\mathcal{L})$. Let v_i' denote the leftmost point and v_i'' denote the rightmost point in V_i, that is, $q_L(v_i') \leq q_L(v) \leq q_L(v_i''), v \in V_i$. The edges of $E(\mathcal{L})$ whose endpoints belong to different subsets of $V_i, 1 \leq i \leq p$, are split-edges. Thus, locating an absolute p-center on $\mathcal{L}$ is equivalent to finding a set of $p - 1$

split-edges which define p regions such that the maximum service cost of absolute 1-centers of these regions is equal to the absolute p-radius of $V(\mathcal{L})$.

Discarding one demand point v is called a *safe operation for a center* α, if v is served by α, and v is not the farthest weighted demand point to α. Similarly, discarding one demand point is called a *safe operation for an interval* $[x, y]$, if it is a safe operation for any center located in $[x, y]$. Our main idea here is to locate one subset of demand points that are served by the same center in some optimal solution. Safely pruning a fraction of demand points in this subset will result in a smaller-size similar problem whose optimal solution is the same as that of the original problem. The pruning step at each iteration is done using the parametric-pruning technique introduced in Section 3.2.2.

Let S represent the set of existing facilities ($|S| = s$ is a constant number). In a center problem without any existing facility, $s = 0$. A value $r \geq 0$ is feasible for a conditional p-center problem on $\mathcal{L}$ if there exists a set of at most p points $X = \{\alpha_1, \ldots, \alpha_p\}$ on $\mathcal{L}$ such that $F(X \cup S, V(\mathcal{L})) \leq r$. Clearly, the optimal cost is the minimum value of r that is feasible. In the following, we show that a feasibility test for a conditional p-center problem on $\mathcal{L}$ can be done in linear time.

An algorithm to test the feasibility of r Each demand point v in $V(\mathcal{L})$ is bundled with a *center region* $[q_L(v) - r/w(v), q_L(v) + r/w(v)]$ which contains all the points of $V(\mathcal{L})$ with a weighted distance to v of no more than r. A given value r is feasible if there exists a solution (a set of p points on $\mathcal{L}$) such that at least one center in this solution lies in the center region of each demand point.

Initially, $X = \emptyset$. Among all these center regions, we find a region whose right endpoint has the smallest coordinate. Let it be a center (i.e., insert it into X) and remove all demand points whose center regions contain it. Repeat this process on the remaining demand points. Finally, if $|X| \leq p$, then r is feasible, and otherwise r is infeasible. This process takes $O(pn)$ time.

Lemma 3.4.1 *A feasibility test of a given non-negative value r for a conditional p-center problem can be done in $O(pn)$ time.*

3.4.2 The conditional 1-center problem

In this section, we describe a linear-time algorithm for the conditional 1-center problem for points on $\mathcal{L}$ with a fixed number of existing facilities (i.e., $S = \{\beta_1, \ldots, \beta_s\}$). We assume that $q_L(\beta_i) < q_L(\beta_{i+1}), 1 \leq i \leq s-1$.

The set S divides $\mathcal{L}$ into $s+1$ continuous regions, i.e., $(-\infty, q_L(\beta_1)), (q_L(\beta_1), q_L(\beta_2)), \ldots, (q_L(\beta_s), \infty)$. Let $Q_L(\subseteq V(\mathcal{L}))$ be the set of dominating demand points of S. Since s is a constant, Q can be computed in linear time. If the points in Q are distributed in more than one continuous region listed above, then for any point x in $\mathcal{L}$, $F(\{x\} \cup S, V(\mathcal{L})) = F(S, V(\mathcal{L}))$, and therefore, any point in $\mathcal{L}$ is an optimal 1-center solution.

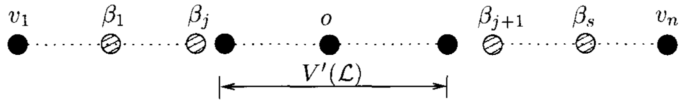


Figure 3.12: The conditional 1-center problem with a set S of existing facilities.

Otherwise, without the loss of any generality, we assume that all demand points in Q lie in region $(q_L(\beta_j), q_L(\beta_{j+1})), 0 \leq j \leq s$ (let $q_L(\beta_0) = -\infty$ and $q_L(\beta_{s+1}) = \infty$). Refer to Figure 3.12. Then, in an optimal solution, the center lies in $(q_L(\beta_j), q_L(\beta_{j+1}))$ and therefore, demand points lying in other regions are served by existing facilities. For each demand point v not lying in $(q_L(\beta_j), q_L(\beta_{j+1}))$, find its closest existing facility $\beta_i, 1 \leq i \leq s$, and compute the service cost $w(v)d(\beta_i, v)$. Let $\mu = \max_{v \in V(\mathcal{L}) \text{ and } v \notin (q_L(\beta_j), q_L(\beta_{j+1}))} w(v)d(S, v)$. Let $V'(\mathcal{L})$ be the set of demand points in $(q_L(\beta_j), q_L(\beta_{j+1}))$. We solve the conditional 1-center problem of $V'(\mathcal{L})$ with existing facilities β_j and β_{j+1}, using the parametric-pruning technique as follows.

Let $S' = \{\beta_j, \beta_{j+1}\}$ and o be a median point of $V'(\mathcal{L})$. Based on the location of the dominating demand points of center set $\{o\} \cup S'$, we can determine the relative location of new facility with respect to o in an optimal solution. Without loss of generality, assume that all dominating demand points of $\{o\} \cup S'$ lie in $(q_L(o), q_L(\beta_{j+1}))$. Then, arbitrarily pair the demand points in $(q_L(\beta_j), q_L(o))$. For each such pair (u_i, v_i), the new facility α serves them through o (see Figure 3.13).

Below we show that at most one intersection exists between the two service cost functions $F(\{\alpha\} \cup S', u_i)$ and $F(\{\alpha\} \cup S', v_i)$. If $d(o, v) \geq d(S', v)$ for a demand point v in $(q_L(\beta_j), q_L(o))$, then v will always be served by some existing facility in S'. Without loss of generality, assume that $d(o, u_i) < d(S', u_i), d(o, v_i) < d(S', v_i)$, and $d(u_i, o) \geq d(v_i, o)$.

Lemma 3.4.2 *For any two points u and v in $(q_L(\beta_j), q_L(o))$, if $d(o, u) \geq d(o, v)$ then $d(S', u) - d(o, u) \leq d(S', v) - d(o, v)$.*

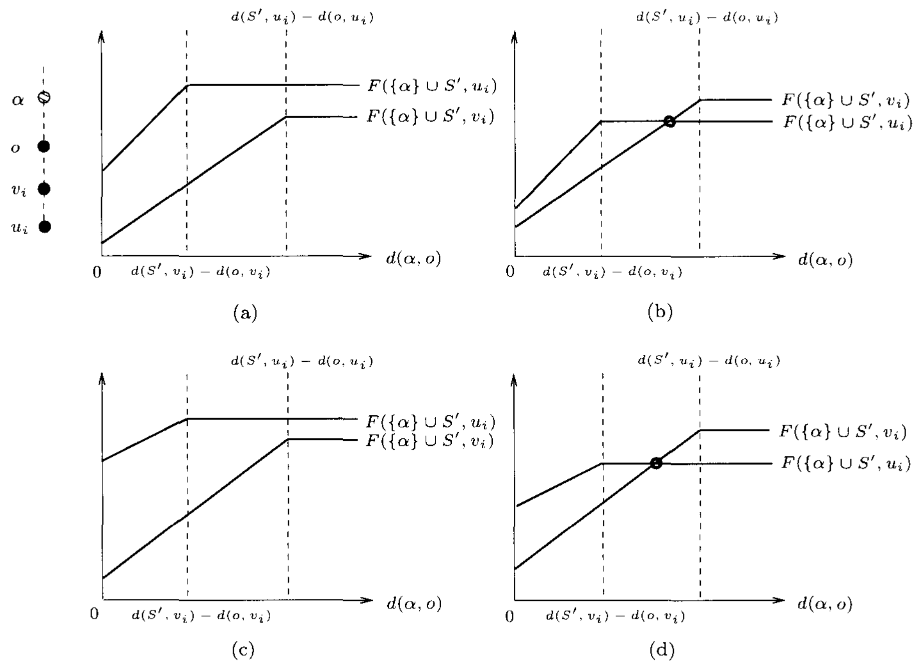


Figure 3.13: The number of intersections between the two service cost functions $F(\{\alpha\} \cup S', u_i)$ and $F(\{\alpha\} \cup S', v_i)$. (a)(b) $w(u_i) \geq w(v_i)$; (c)(d) $w(u_i) < w(v_i)$.

Proof There are three cases for $d(S', u)$ and $d(S', v)$, that is, either $d(S', u) = d(\beta_j, u)$ and $d(S', v) = d(\beta_j, v)$, $d(S', u) = d(\beta_j, u)$ and $d(S', v) = d(\beta_{j+1}, v)$, or $d(S', u) = d(\beta_{j+1}, u)$ and $d(S', v) = d(\beta_{j+1}, v)$. In any one of them, it is true that $d(S', u) - d(o, u) \leq d(S', v) - d(o, v)$. $\square$

As illustrated in Figure 3.13, at most one intersection exists between the two service cost functions $F(\{\alpha\} \cup S', u_i)$ and $F(\{\alpha\} \cup S', v_i)$. We can prune one by solving at most one feasibility test. Thus, similar to our parametric-pruning approach for the weighted continuous 1-center problem in a tree, we are able to prune at least $\lfloor |V'((L))|/8 \rfloor$ non-dominating demand points in $(q_L(\beta_j), q_L(o))$ from further consideration by solving one feasibility test. Therefore, the conditional 1-center problem of $V'(\mathcal{L})$ with existing facilities β_j and β_{j+1} can be solved in linear time. Let μ' be its optimal service cost. The optimal service cost of the conditional 1-center problem of $V(\mathcal{L})$ with existing facility set S is $\max\{\mu, \mu'\}$.

In summary, we have the following theorem.

Theorem 3.4.3 *The weighted conditional* 1*-center problem of* $V(\mathcal{L})$ *with a fixed number of existing facilities on the real line* $\mathcal{L}$ *can be solved in linear time.*

3.4.3 The weighted p-center problem (fixed p)

Suppose that a weighted k-center problem and a conditional k-center problem (with a fixed number of existing facilities) of $V(\mathcal{L})$ on $\mathcal{L}$ can be solved in linear time, for all k less than p. We show that for fixed p the weighted p-center problem of $V(\mathcal{L})$ can also be solved in linear time.

We first determine one region which contains at least $\lfloor n/p \rfloor$ demand points of $V(\mathcal{L})$ served by same center in some optimal solution. Such a region is called a *big region.*

Since the ith largest element of a set can be found in linear time [16], it costs $O(n \log p)$ time to divide $V(\mathcal{L})$ into p mutually disjoint subsets: $\{V_1, \ldots, V_p\}$ where $\lfloor n/p \rfloor \leq |V_i| \leq \lceil n/p \rceil, 1 \leq i \leq p$, and the coordinate $q_L(v_i'')$ of the rightmost point v_i'' of V_i is less than the coordinate $q_L(v_{i+1}')$ of the leftmost point v_{i+1}' of $V_{i+1}, i = 1, \ldots, p-1$. Note that $v_1' = v_1$ and $v_p'' = v_n$.

Consider the split-edge $\overline{v_i'' v_{i+1}'}$, $1 \leq i < p$. Refer to Figure 3.14. It is easy to prove the following lemma.

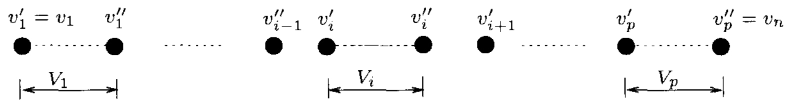


Figure 3.14: Locate p centers of $V(\mathcal{L})$ on the real line $\mathcal{L}$.

Lemma 3.4.4 *Let* c *be the optimal cost of the weighted* i*-center problem of demand set* $\bigcup_{j=1}^{i} V_j$. *If* c *is feasible for* $V(\mathcal{L})$*, then* $\exists$ *an optimal solution such that the region served by the first* i *centers contains all demand points in* $[q_L(v_1'), q_L(v_i'')]$*, and the interval* $[q_L(v_1'), q_L(v_i'')]$ *contains a big region. Otherwise, in some optimal solution, the region served by the first* i *centers is contained in* $[q_L(v_1'), q_L(v_i'')]$.

As a consequence of Lemma 3.4.4, one of the following cases must be true for the p-center problem on $\mathcal{L}$.

Case 1 The optimal cost of the 1-center problem with demand set V_1 is feasible. In this case, the subset V_1 is served by the same center in some optimal solution.

Case 2 The optimal cost of the 1-center problem with demand set V_p is feasible. Similar to Case 1, the subset V_p is served by the same center in some optimal solution.

Case 3 There exists an i, $2 \leq i \leq p-1$, such that the optimal cost of the i-center problem with demand set $\bigcup_{j=1}^{j=i} V_j$ is feasible, and the optimal cost of the $(i-1)$-center problem with demand set $\bigcup_{j=1}^{j=i-1} V_j$ is not feasible. In this case, there exists some optimal solution where V_i is served by the same center. The reason is as follows.

In some optimal solution, the first i centers serve all demand points lying within $[q_L(v'_1), q_L(v''_i)]$, since the cost of an optimal i-center solution of $\bigcup_{j=1}^{j=i} V_j$ is feasible (Lemma 3.4.4). Similarly, all demand points served by the first $i-1$ centers lie within $[v'_1, v''_{i-1}]$, since the cost of an optimal $(i-1)$-center solution of $\bigcup_{j=1}^{j=i-1} V_j$ is not feasible (Lemma 3.4.4).

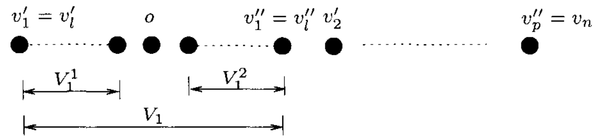
Thus we can find one big region after considering each edge $\overline{v''_i v'_{i+1}}$, $i = 1, \ldots, p-1$, as a split-edge. Note that we need to solve one center problem and one feasibility test for each split-edge $\overline{v''_i v'_{i+1}}$: one i-center problem on a demand set $\bigcup_{j=1}^{j=i} V_j$ and one feasibility test of its optimal cost. The total time is linear when p is fixed. Therefore, we have the following result.

Lemma 3.4.5 *It takes a linear time to locate one big region served by the same center, in some optimal solution to the weighted p-center problem of $V(\mathcal{L})$ on $\mathcal{L}$ when p is a fixed number.*

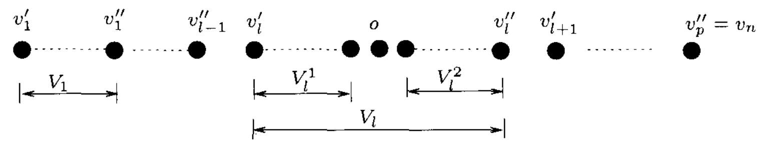
Actually, one big region can be located using a binary search instead of a linear search on the edges $v''_i v'_{i+1}, 1 \leq i \leq p-1$. Let V_h be the big region thus computed.

Next, we show a method to identify approximately 1/8 of demand points of V_h that are not dominating, and hence can be discarded. This method is very similar to the method described in Section 3.4.2. Let o be a median point of V_h which can be found in $O(n/p)$ time. Consider a facility (center) located at o to serve the points of V_h. Let V_h^1 (resp. V_h^2) be the subset of demand points of V_h lying to the left (resp. right) of o.

We now consider Case 1 described above, i.e., when $h = 1$. The arguments for Case 2 (i.e., when $h = p$) are similar. Refer to Figure 3.15(a). For Case 1, we compute $c_1 = F(o, V_1^1)$ and check the feasibility of c_1. If c_1 is feasible, then in some optimal solution, the center serving V_1 lies to the right of o. Otherwise, in some optimal solution, the center serving V_1 lies to the left of o.



(a) Case 1



(b) Case 3

Figure 3.15: Case 1 and Case 3

In Case 3 (see Figure 3.15(b)), where $2 \leq h \leq p-1$, we solve the $(h-1)$-center problem of $V_h^1 \cup \bigcup_{j=1}^{j=h-1} V_j$ with one existing center located at o. Let c_2 be its optimal cost. Like Case 1, after checking the feasibility of c_2, either we find that in some optimal solution the center serving V_h lies to the left of o or we find that the center serving V_h lies to the right of o.

Now, our parametric-pruning approach for conditional 1-center problems (in Section 3.4.2) can be applied to prune approximately 1/8 of demand points of V_h, i.e., $\lfloor n/8p \rfloor$ demand points of $V(\mathcal{L})$. The process is repeated with the reduced set of demand points. Thus, for fixed p, the algorithm performs $O(\log n)$ such iterations, and each iteration takes linear time, linear in the size of the current demand set.

Therefore, we have the following theorem.

Theorem 3.4.6 *For any fixed p, the weighted p-center problem of $V(\mathcal{L})$ on the real line $\mathcal{L}$ can be solved in linear time.*

3.4.4 The conditional p-center problem (fixed p)

In this section, we briefly describe a linear-time algorithm for the conditional p-center problem on $\mathcal{L}$ with a fixed number existing facilities. The method for this problem is an extension of the steps described in Section 3.4.2 and Section 3.4.3.

We divide $V(\mathcal{L})$ into p mutually disjoint subsets of almost equal size. One big region of size no less than $\lfloor n/p \rfloor$ can be located in linear time, using steps similar to the ones described in Section 3.4.3. The only difference is that the subproblems are conditional center problems here. Still, our parametric-pruning approach is applicable to prune approximately 1/8 of the demand points lying in the big region.

Theorem 3.4.7 *For any fixed p, the weighted conditional p-center problem of $V(\mathcal{L})$ with a fixed number of existing facilities located on the real line $\mathcal{L}$ can be solved in linear time.*

3.5 Summary

The weighted version of discrete/continuous p-center problems in a tree network and the real line are studied for a fixed constant p. The time complexity of our proposed algorithms for them are $O(n)$. The results partially resolve the long standing open problem, that is, generalizing Megiddo's trimming approach [54] to solve the p-center problem in a tree network or the real line for $p > 1$. It needs to be mentioned that the running time of our algorithms is exponential in p. One challenging task is to design an $O(f(p) \cdot n)$-time algorithm for the weighted p-center problem in a tree network or a real line where $f(p)$ is a low-degree polynomial of p.

Chapter 4

Various p-center problems in tree-like networks

In this chapter we focus on various p-center location problems in tree-like networks, such as cactus networks and partial k-trees. When the underlying network is a partial k-tree, Granot and Skorin-Kapov [32] gave an $O(p^2n^{k+2})$-time algorithm for the weighted discrete p-center problem in which centers are restricted to the vertices of the network and p is a part of the input. We study this problem and present an efficient algorithm for relatively small p. The running time of our algorithm is $O(pn^p \log^k n)$. Then, when $p < k+2$, our algorithm is better. We also discuss the weighted continuous p-center problem in which centers can be located at any place in the network, and we devise the first polynomially bounded algorithm for fixed k, which runs in $O(p^2k^{k+1}n^{2k+3} \log n)$ time.

For a cactus network, we first provide an $O(n \log n)$-time algorithm to solve the weighted discrete and continuous 1-center problems. We then show that the weighted continuous 2-center problem can be solved in $O(n \log^3 n)$ time. We also, for the first time, look at various p-center problems in a cactus network where p is a part of the input, including the weighted discrete and continuous p-center problems, the unweighted discrete p-center problem with the demand set of infinite size, and the unweighted general p-center problem. Our algorithms for these p-center problems are based on the parametric-searching technique.

Organization of the chapter In Sections 4.1 and 4.2, we discuss various center problems in partial k-trees and cactus networks respectively. A brief summary is given in Section 4.3.

4.1 Weighted p-center problems in a partial k-tree

It is known that a tree decomposition (of treewidth k) of a partial k-tree G, denoted by $\mathcal{TD}(G)$, can be found in linear time for fixed k [17]. A tree decomposition $\mathcal{TD}(G) : (\{B_i : i \in I\}, \mathcal{T} = (I, Y))$ of treewidth k is called *smooth* if for all $i \in I : |B_i| = k + 1$, and for all $(i, j) \in Y : |B_i \cap B_j| = k$. A tree decomposition of a graph G can be transformed to a smooth tree decomposition of G with the same treewidth in linear time [17].

For a partial k-tree G, there exists an i-separator ($i \leq k$) between two subnetworks represented by two disjoint subtrees $\mathcal{T}_1, \mathcal{T}_2$ of $\mathcal{TD}(G)$. Let B_1 be the closest bag in $\mathcal{T}_1$ to $\mathcal{T}_2$ and B_2 be the closest bag in $\mathcal{T}_2$ to $\mathcal{T}_1$. It is easy to see that $B_1 \cap B_2$ is a separator between the two subnetworks represented by $\mathcal{T}_1 \& \mathcal{T}_2$ and that $|B_1 \cap B_2| \leq k$.

A leaf bag is arbitrarily chosen to be the root of $\mathcal{TD}(G)$. Let $\mathcal{T}_B$ denote the subtree rooted at a bag B, and let $V(\mathcal{T}_B)$ denote the union of bags of vertices in the subtree $\mathcal{T}_B$. Note that there is a k-separator between the subgraphs induced by $V(\mathcal{T}_B)$ and $V(\mathcal{T} \setminus \mathcal{T}_B)$. Let the k-separator be denoted by the set $\{v_1^B, \dots, v_k^B\}$.

Observation 4.1.1 *Given a bag B in $\mathcal{TD}(G)$, let $B_1, \cdots, B_i$ be the bags adjacent to B. There is a k-separator between B and $B_j, 1 \leq j \leq i$. The union of these k-separators is a subset of B and therefore contains at most $k + 1$ vertices in $V(G)$.*

A smooth tree decomposition can be transformed into a binary smooth tree decomposition in linear time by replacing every bag containing more than two children with a path. It is easy to see that the size of the binary smooth tree decomposition obtained is $O(n)$. Without loss of generality, we assume that $\mathcal{TD}(G)$ of treewidth k is smooth and binary.

4.1.1 The weighted discrete p-center problem

Given a tree decomposition $\mathcal{TD}(G)$ of treewidth k, an $O(p^2 n^{k+2})$ algorithm [32] was proposed to solve the unweighted/weighted $V(G)/V(G)/p$ problems. The result is true for any value of p. The algorithm of Granot and Skorin-Kapov [32] is based on the dynamic programming technique, which is described as follows.

For each bag B in $\mathcal{TD}(G)$, we solve the following q-center subproblems on the subtree $\mathcal{T}_B$ where q is a nonnegative integer no more than p. Recall that $\{v_1^B, \dots, v_k^B\}$ is the k-separator between the subgraphs induced by $V(\mathcal{T}_B)$ and $V(\mathcal{T} \setminus \mathcal{T}_B)$. Given a set of vertices $u_1, \dots, u_k$

(it is possible that $u_i = u_j$ for $1 \le i \ne j \le k$), the subproblem on $\mathcal{T}_B$ is to compute the minimum value of $F(X, V(\mathcal{T}_B))$ such that

- $u_i \in X, i = 1, \ldots, k$,
- $X \subseteq V(\mathcal{T}_B) \cup \{u_1, \ldots, u_k\}$,
- $|X \cap V(\mathcal{T}_B)| = q$, and
- v_i^B is served by a center located at $u_i, i = 1, \ldots, k$.

Obviously, $|V(\mathcal{T}_B) \cap \{u_1, \ldots, u_k\}| > q$. We can see that there are $O(pn^k)$ such subproblems on $\mathcal{T}_B$. Let B_1 and B_2 be the two children of B, and B' be the parent of B (see Figure 4.1). We assume that the optimal solutions of all subproblems on $\mathcal{T}_{B_j}$ are available, $j = 1, 2$.

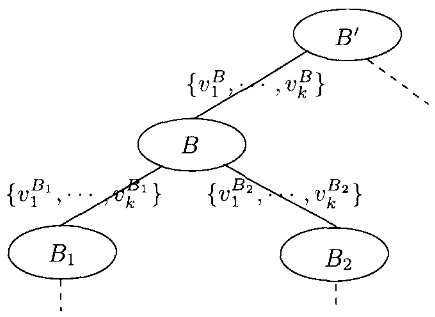

B'
$\{v_1^B, \cdots, v_k^B\}$
B
$\{v_1^{B_1}, \cdots, v_k^{B_1}\}$
$\{v_1^{B_2}, \cdots, v_k^{B_2}\}$
B_1
B_2

Figure 4.1: A bag B and its neighbors in a binary smooth tree decomposition of treewidth k.

A q-center subproblem on $\mathcal{T}_B$ ($0 \le q \le p$) can be solved by considering all possible subproblems on $\mathcal{T}_{B_1}$ and $\mathcal{T}_{B_2}$, i.e., q'-center subproblems on $\mathcal{T}_{B_1}$ ($0 \le q' \le q$) and $(q - q')$-center subproblems on $\mathcal{T}_{B_2}$ (where v_i^B is served by a center located at $u_i, i = 1, \ldots, k$). According to Observation 4.1.1, $\{v_1^B, \cdots, v_k^B, v_1^{B_1}, \cdots, v_k^{B_1}, v_1^{B_2}, \cdots, v_k^{B_2}\} \subseteq B$. Therefore, there is at most one vertex in $\{v_1^{B_1}, \cdots, v_k^{B_1}, v_1^{B_2}, \cdots, v_k^{B_2}\} \setminus \{v_1^B, \cdots, v_k^B\}$, say v.

Lemma 4.1.2 *v is served by some center in $V(\mathcal{T}_B) \cup \{u_1, \ldots, u_k\} \setminus \{v_1^B, \ldots, v_k^B\}$ in the $V(G)/V(G)/p$ model.*

Proof The reason is that $\{v_1^B, \ldots, v_k^B\}$ is the k-separator between the subgraphs induced by $V(\mathcal{T}_B)$ and $V(\mathcal{T} \setminus \mathcal{T}_B)$ and v_i^B is served by $u_i, i = 1, \ldots, k$. □

Hence, $O(q \cdot |V(\mathcal{T}_B)|)$ subproblems on $\mathcal{T}_{B_1}, \mathcal{T}_{B_2}$ need to be solved to obtain an optimal solution for one q-center subproblem on $\mathcal{T}_B$, since in the $V(G)/V(G)/p$ model there are $O(n)$ candidate centers.

Using a dynamic programming approach, each subproblem can be solved in time $O(pn)$ and therefore the optimization problem on the whole tree decomposition $\mathcal{TD}(G)$ can be solved in time $O(n) \cdot O(pn^k) \cdot O(pn) = O(p^2 n^{k+2})$.

Theorem 4.1.3 *[32] Given a tree decomposition with treewidth k of a partial k-tree G, the weighted $V(G)/V(G)/p$ problem can be solved in $O(p^2 n^{k+2})$ time.*

Next, we present an $O(pn^p \log^k n)$-time algorithm for the $V(G)/V(G)/p$ problem when p is small, i.e., $p \le k+1$.

An algorithm for the $V(G)/V(G)/p$ problem when $p \le k+1$

A distance query of a pair of points x, y in a network is to obtain the distance between x, y. Considering the tight relationship between the service cost and distance queries, an efficient approach is to preprocess the network so that distance queries can be efficiently answered. This approach is particularly promising when the network is sparse [22]. Chaudhuri and Zaroliagis [22] gave algorithms for distance queries that depend on the treewidth of the input network. Their algorithms can answer each distance query in $O(1)$ time for constant treewidth networks after $O(n \log n)$ preprocessing. Based on this result, we introduced a two-level tree decomposition structure on a partial k-tree network [6], which can be built on any partial k-tree G in $O(n \log^2 n)$ time requiring $O(n \log^2 n)$ storage space for $k = 2$ and in $O(nk \log^{k-1} n)$ time requiring $O(nk \log^{k-1} n)$ storage space for $k > 2$. Given such a two-level tree decomposition structure, the service cost of any set X of p centers to the demand set $V(G)$, i.e., $F(X, V(G))$, can be answered in time $O(p \log^2 n)$ for $k = 2$ and in time $O(pk \log^{k-1} n)$ for $k > 2$. The main idea behind this two-level tree decomposition data structure is described as follows.

We consider the case when G is a partial 2-tree. Refer to Figure 4.2. Given a subgraph G' represented by a subtree of $\mathcal{TD}(G)$ and a point x outside G', there is a 2-separator in G' between G' and x. Let $\{u_1, u_2\}$ be the 2-separator. The service cost of x to cover $v \in V(G')$ is $w(v) \cdot \min\{d(v, u_1) + d(u_1, x), d(v, u_2) + d(u_2, x)\}$. Suppose $a = d(x, u_1) - d(x, u_2)$ and $a' = d(v, u_1) - d(v, u_2)$. Clearly the shortest path $\pi(v, x)$ from v to x will go through u_1 if $a + a'$ is negative, otherwise $\pi(v, x)$ will go through u_2.

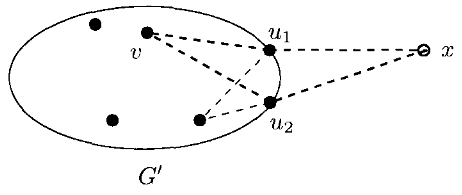


Figure 4.2: The 2-separator $\{u_1, u_2\}$ between G' and any point x outside G'.

Based on the above observation, we create two lists of the vertices in G', $\mathcal{J}_1$ and $\mathcal{J}_2$. The vertices of $\mathcal{J}_1$ are sorted in the increasing order of $\chi_1(\cdot)$ where $\chi_1(v)$ is the distance difference from a vertex $v(\in V(G'))$ to the 2-separator $\{u_1, u_2\}$, i.e., $\chi_1(v) = d(v, u_1) - d(v, u_2)$. The vertices of $\mathcal{J}_2$ are sorted in the increasing order of $\chi_2(\cdot)$ where $\chi_2(v) = d(v, u_2) - d(v, u_1)$ for all $v \in V(G')$. These two lists $\mathcal{J}_1$ and $\mathcal{J}_2$ are associated with u_1 and u_2 respectively.

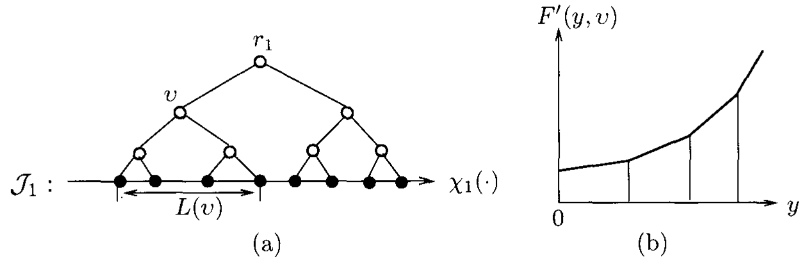


Figure 4.3: A balanced binary search tree S_{χ_1} over the sorted list χ_1.

A balanced binary search tree over $\mathcal{J}_1$ (resp. $\mathcal{J}_2$) is built (see Figure 4.3(a)), denoted by T_1 (resp. T_2). Let r_1 and r_2 be the root nodes of T_1 and T_2 respectively. The set of vertices in G' whose shortest paths to a point x outside G' go through u_1 (resp. u_2) is represented by $O(\log |V(G')|)$ sublists in $\mathcal{J}_1$ (resp. $\mathcal{J}_2$). It is not hard to see that, given a point x outside G' and its distances to u_1 and u_2, these sublists can be identified in $O(\log |V(G')|)$ time. Note that $d(x, u_1)$ and $d(x, u_2)$ can be computed in constant time after $O(n \log n)$-time preprocessing [22].

Compute $F(x, V(G'))$ for any point x outside G' In the following we show that $F(x, V(G'))$ can be computed in $O(\log |V(G')|)$ time by applying the fractional cascading technique [23]. The fractional cascading technique is a technique to reduce the query time in the case of many 1-dimensional searches with the same range, using the result of one search to speed up other searches. We briefly describe the application of the fractional cascading technique on T_1 as follows.

For every node v in T_1, let $L(v)$ be the set of leaves descending from v in T_1 (see Figure 4.3(a)), and let $F'(y, v)$ denote the service cost function of a point, lying outside G' with distance y to u_1, to cover the vertices in $L(v)$ via u_1. For every vertex $v \in L(v)$ (v is a node in T_1), there is at most one continuous region of y in which v determines the service cost $F'(y, v)$ (i.e., v is the weighted farthest vertex in the sublist $L(v)$ to a point lying outside G' with distance y to u_1), called the *dominating region* of v in the sublist $L(v)$. Moreover, these dominating regions are sorted in increasing order of weights of vertices in $L(v)$ (see Figure 4.3(b)).

Therefore, in a bottom-up way, we can compute the service cost function $F'(y, v)$ for every v in T_1 in time $O(|V(G')| \log |V(G')|)$. By applying the fractional cascading technique [23], we are able to locate the weighted farthest vertex in $L(v)$ for every v in T_1 in constant time after an $O(\log |V(G')|)$-time computation of the weighted farthest vertex in $L(r_1)$. A similar result is obtained after applying the fractional cascading technique on T_2. As we already know, the set of vertices in G' whose shortest paths to x go through u_1 (resp. u_2) is represented by $O(\log |V(G')|)$ sublists in $\mathcal{J}_1$ (resp. $\mathcal{J}_2$). Therefore, the total cost of computing the maximum service cost $F(x, V(G'))$ of x to the vertices in G' is $O(\log |V(G')|)$ after $O(|V(G')| \log |V(G')|)$ preprocessing time.

A two-level tree decomposition of G Since the tree decomposition $\mathcal{TD}(G)$ of G might not be balanced, we add another balanced tree structure over $\mathcal{TD}(G)$, such that the height of the new tree $\overline{\mathcal{TD}}(G)$ is logarithmic. We call such a balanced tree structure $\overline{\mathcal{TD}}(G)$ *a two-level tree decomposition of* G. There are several methods to achieve this, such as centroid tree decomposition [55], spine tree decomposition [9] etc.. After completing the two-level tree decomposition of G, for each bag in $\mathcal{TD}(G)$, there are $O(\log n)$ subtrees of $\overline{\mathcal{TD}}(G)$ containing all the other bags in $TD(G)$. Moreover, for a set X of p vertices (resp. points) in $V(G)$ (resp. $A(G)$) there are $O(p \log n)$ subgraphs, represented by $O(p \log n)$ subtrees of $\overline{\mathcal{TD}}(G)$, that contain the rest of the vertices in G. A 2-separator exists between a vertex (resp. point) in X and each such subgraph. We get the following lemma.

Lemma 4.1.4 *A two-level tree decomposition data structure of a partial 2-tree can be computed in $O(n \log^2 n)$ time requiring $O(n \log n)$ storage space, such that the service cost of a set of p points in the partial 2-tree can be answered in $O(p \log^2 n)$.*

Thus, the unweighted/weighted $V(G)/V(G)/2$ problem in a partial 2-tree can be solved in $O(pn^p \log^2 n)$ time, using a brute-force-like approach.

Theorem 4.1.5 *Given a tree decomposition (of treewidth 2) of a partial 2-tree G, the weighted $V(G)/V(G)/2$ problem can be solved in $O(pn^p \log^2 n)$ time.*

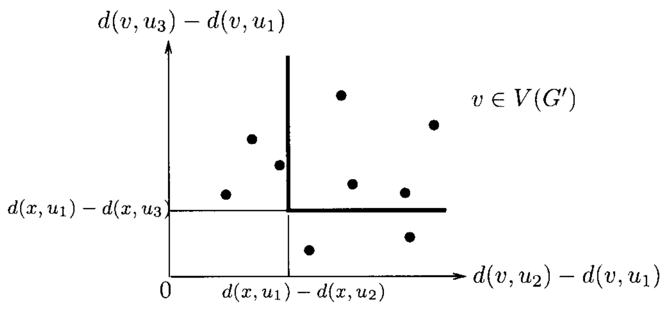


Figure 4.4: The set of vertices in $V(G')$ to which the shortest path from x goes through u_1.

This result can be extended to a partial k-tree, $k > 2$. In the case when G is a partial k-tree, given a subgraph G' represented by a subtree of $\mathcal{TD}(G)$ and a point x outside G', there is a k-separator in G' between G' and x. Let $\{u_1, \cdots, u_k\}$ be the k-separator. The main difference from the case when the underlying network is a partial 2-tree, is the representation of the set of vertices to which the shortest path from x goes through $u_i, i = 1, \cdots, k$. Refer to Figure 4.4 in which G is a partial 3-tree. All vertices in $V(G')$ are embedded in a 2-dimensional space (note that it is a $(k-1)$-dimensional space when G is a partial k-tree). Given a point x with its distances to the 3-separator $\{u_1, u_2, u_3\}$, all the vertices lying above and right of the bold line in Figure 4.4 are the vertices in $V(G')$ to which the shortest path from x goes through u_1. The service cost of x to these vertices can be computed in $O(\log |V(G')|)$ time by the combining priority search tree [51] (Algorithm 2 in Chapter 6 shows the steps to construct a priority search tree) and the fractional cascading technique [23] after $O(|V(G')| \log |V(G')|)$ preprocessing time. We build such a data structure for every vertex in the 3-separator (or a k-separator for a partial k-tree).

Similarly, when G is a partial k-tree, we can compute $F(x, V(G'))$ in $O(k \log^{k-2} |V(G')|)$ time after $O(k|V(G')| \log^{k-2} |V(G')|)$ preprocessing time.

To compute the service cost of a set of p vertices (or points) to a partial k-tree G, we build a two-level tree decomposition data structure over G. We have the following lemma.

Lemma 4.1.6 *For $k > 2$, a two-level tree decomposition data structure of a partial k-tree can be computed in $O(nk \log^{k-1} n)$ time requiring $O(nk \log^{k-1} n)$ storage space such that the service cost of a set of p points in the partial k-tree can be answered in $O(pk \log^{k-1} n)$.*

Thus, the unweighted/weighted $V(G)/V(G)/p$ problem in a partial k-tree ($k > 2$ is fixed) can be solved in $O(pkn^p \log^{k-1} n)$ time, using a brute-force-like approach.

Theorem 4.1.7 *Given a tree decomposition (of treewidth $k > 2$) of a partial k-tree G, the weighted $V(G)/V(G)/p$ problem can be solved in $O(pkn^p \log^{k-1} n)$ time.*

4.1.2 The weighted continuous p-center problem

In this section, we discuss the weighted $A(G)/V(G)/p$ problem on a partial k-tree G and devise two simple algorithms for it. The algorithms described below for the weighted $A(G)/V(G)/p$ problem are based on dynamic programming technique.

The 1st algorithm According to Observation 1.2.2, for the weighted $A(G)/V(G)/p$ problem, there is a set of $O(kn^3)$ candidate points where centers may be located in an optimal solution and the optimal cost is in a candidate set of cardinality $O(kn^3)$, since the number of edges m of a partial k-tree is no more than kn. The two sets can be computed in $O(kn^3)$ time. Combining the results from Observation 1.2.2 and the method of Granot and Skorin-Kapov [32], we can design an algorithm for the $A(G)/V(G)/p$ problems, which runs in $O(p^2 \cdot (kn^3)^{k+1} \cdot n) = O(p^2 k^{k+1} n^{3k+4})$.

Unlike the algorithm of Granot and Skorin-Kapov [32], the next algorithm does not solve the optimization problems directly. Instead, we focus on designing an efficient algorithm for solving the corresponding feasibility tests.

The 2nd algorithm Recall that if a non-negative value c is feasible, then there is a p-center solution in which each center is located at a (weighted) distance of exactly c from some demand vertex, and all demand vertices are covered with service cost $\leq c$ (Observation 1.2.3). Therefore, only $O(kn^2)$ candidate points are needed to be considered for the feasibility test of c. Let $\mathcal{W}$ denote the set of these $O(kn^2)$ candidate points.

Similar to the method of Granot and Skorin-Kapov [32], for each bag B in $\mathcal{TD}(G)$, we test the feasibility of c on the subtree $\mathcal{T}_B$ with q centers lying in $\mathcal{W} \cap A(G(V(\mathcal{T}_B)))$ ($0 \leq q \leq p$). Each vertex v_i^B in the k-separator between the subgraphs $G(V(\mathcal{T}_B))$ and

$G(V(\mathcal{T} \setminus \mathcal{T}_B))$ is associated with a point $u_i \in \mathcal{W}(i = 1, \dots, k)$. We assume that the results of possible feasibility tests of c on $\mathcal{T}_{B_j} 1 \leq j \leq 2$ are available (B_1 and B_2 are the two children of B in $\mathcal{TD}(G)$).

If $d(u_i, v_i^B) > c$ for some $i, 1 \leq i \leq k$, then c is infeasible. Otherwise, the feasibility of of c can be evaluated by testing the feasibility of c on $\mathcal{T}_{B_1}$ with q' centers lying in $\mathcal{W} \cap A(G(V(\mathcal{T}_{B_1})))$ and on $\mathcal{T}_{B_2}$ with $q - q'$ centers lying in $\mathcal{W} \cap A(G(V(\mathcal{T}_{B_2})))$, $q' = 0, \dots, q$. Hence, the feasibility test of c on $\mathcal{T}_B$ with q centers lying in $\mathcal{W} \cap A(G(V(\mathcal{T}_B)))$ can be done in $O(qkn^2)$ time. Combining this result and the method of Granot and Skorin-Kapov [32], an algorithm can be obtained for a feasibility test of the $A(G)/V(G)/p$ problems, which runs in $O(n \cdot p \cdot (kn^2)^k \cdot pkn^2) = O(p^2 k^{k+1} n^{2k+3})$ time, since we consider $O(p \cdot (kn^2)^k)$ subproblems on each rooted subtree of $\mathcal{TD}(G)$, which implies that the $A(G)/V(G)/p$ problem can be solved in $O(p^2 k^{k+1} n^{2k+3} \log n)$ time by a binary search of the set of size $O(kn^3)$ containing the optimal cost.

Theorem 4.1.8 *Given a tree decomposition (of treewidth k) of a partial k-tree G, the weighted $A(G)/V(G)/p$ problem can be solved in $O(p^2 k^{k+1} n^{2k+3} \log n)$ time.*

4.2 Various center problems in a cactus network

In this section, various types of center location problems in a cactus network G are discussed. Section 4.2.1 provides an $O(n \log n)$-time algorithm to solve the weighted $A(G)/V(G)/1$ and $V(G)/V(G)/1$ problems. We then show that the linear-time solutions to unweighted $A(G)/A(G)/1$ and $V(G)/A(G)/1$ problems follow quite easily. Our algorithms for the weighted $V(G)/V(G)/2$ and $A(G)/V(G)/2$ problems are presented in Section 4.2.2. Section 4.2.3 describes some results on the weighted/unweighted p-center problems in cactus networks.

In order to facilitate the overview of the proposed algorithms, we start with the well-known tree structure of a cactus network [19]. The vertex set $V(G)$ is partitioned into three different subsets. A *C-vertex* is a vertex of degree 2 which is included in exactly one simple cycle. A *T-vertex* is a vertex not included in any simple cycle. The remaining vertices, if any, will be referred to as *H-vertices* or *hinges* (See Fig.4.5(a)). We use the dotted ellipses to emphasize blocks.

It is not difficult to see that a cactus consists of blocks where each block is either a cycle or a subtree, and these blocks are glued together with H-vertices. Therefore, we can

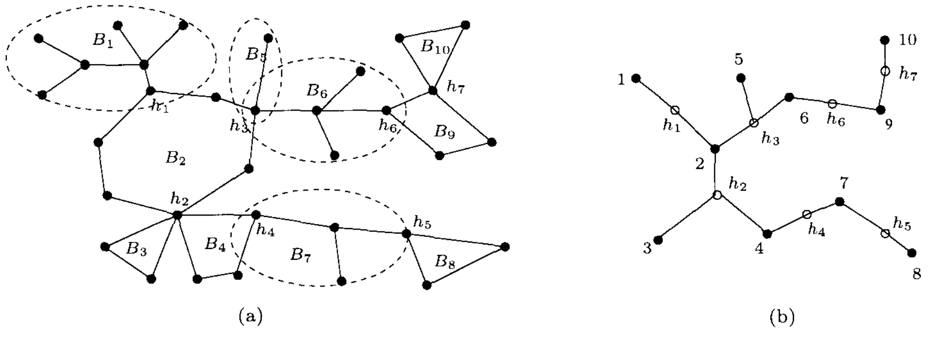


Figure 4.5: A cactus network G and its corresponding tree structure T_G.

use a tree $T_G = (V_G, E_G)$ to represent the important structure of G, where each node in V_G represents a block or a hinge vertex in G (see Figure 4.5(b)). Let B_b denote the block represented by a block node $b \in V_G$. There is an edge between a block node b and a hinge node h if $h \in V(B_b)$. In this case we say that B_b is attached to h.

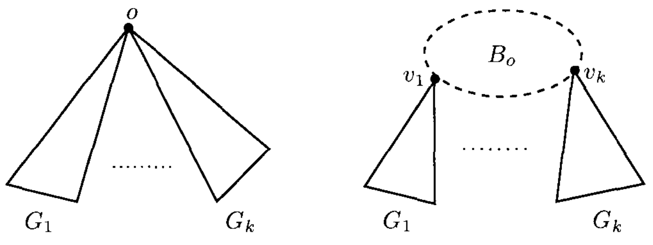
4.2.1 Weighted discrete and continuous 1-center problems

Let $G_1, \ldots, G_k$ denote the connected components attached to a hinge vertex h. If the maximum value of $F(h, \mathcal{D}(G_1)), \cdots, F(h, \mathcal{D}(G_k))$ is attained at more than one component then clearly h itself is an optimal 1-center. On the other hand, if the maximum value is attained in a unique G_i ($1 \leq i \leq k$), then G_i must contain an optimal 1-center. This allows one to find in linear time whether a given hinge vertex h is an optimal 1-center, and in the case when h is not an optimal 1-center, determines which component attached to h contains an optimal 1-center. The proposed algorithm has two steps. The first step is to locate the block containing an optimal 1-center α_G^*, denoted by B^*. The second step is to determine α_G^* in B^*. Similar steps were discussed in [24] for general networks. Our version here for the cactus networks is slightly modified.

Step 1: locating the block B^*

Let o be a centroid node of T_G, the tree structure of G described earlier. The node o could be a hinge vertex or a block (either a cycle or a subtree) in G. These cases are separately considered below.

Case 1: o is a hinge vertex (Figure 4.6(a)). If there exist subnetworks G_i and G_j attached



(a) o is a hinge vertex (b) o is a block node

Figure 4.6: Locate the sub-cactus where the center lies.

to o, $1 \leq i \neq j \leq k$, such that $F(o, \mathcal{D}(G_i)) = F(o, \mathcal{D}(G_j)) \geq F(o, \mathcal{D}(G_s))$ for every G_s, $s \neq i, j$ $(1 \leq s \leq k)$, attached to o, then o itself is an optimal 1-center α_G^*. Suppose that the subnetwork G_i with the largest $F(o, \mathcal{D}(G_i))$ is unique. In this case an optimal center lies in G_i. Clearly, $F(o, \mathcal{D}(G_j))$, for all $j, 1 \leq j \leq k$ can be evaluated in linear time.

Case 2: o is a block node (Figure 4.6(b)). If there are two subnetworks G_i and G_j attached to B_o, such that $F(v_i, \mathcal{D}(G_i)) = F(v_j, \mathcal{D}(G_j)) \geq F(v_s, \mathcal{D}(G_s))$ for every G_s attached to B_o, $s \neq i, j$, then an optimal center lies in the block B_o. In this case B_o is B^*. Suppose that G_i with the largest value of $F(v_i, \mathcal{D}(G_i))$ is unique. Therefore an optimal 1-center either lies in block B_o or in sub-cactus G_i. We compare $F(v_i, \mathcal{D}(G_i))$ with $F(v_i, \mathcal{D}(G \setminus G_i))$. If $F(v_i, \mathcal{D}(G_i)) < F(v_i, \mathcal{D}(G \setminus G_i))$, then an optimal center certainly lies in block B_o. Similarly, if $F(v_i, \mathcal{D}(G_i)) > F(v_i, \mathcal{D}(G \setminus G_i))$, then an optimal center lies in G_i. The remaining case is when $F(v_i, \mathcal{D}(G_i)) = F(v_i, \mathcal{D}(G \setminus G_i))$. In this case, the hinge vertex v_i itself is an optimal center α_G^*. Clearly all of these steps require linear time to compute.

Thus we have the following situations: either α_G^* is found and in this case the algorithm terminates, B^* is found, or a sub-cactus G_i containing α_G^* is found and in this case the process is repeated on G_i. The above situation can be tested in linear time. Therefore, it takes $O(n \log n)$ time to locate B^*.

Lemma 4.2.1 *It takes $O(n \log n)$ time to locate B^*.*

Observation 4.2.2 *The process of identifying B^* can be performed in linear time if the points in $\mathcal{D}(G)$ are unweighted. This is due to the fact that, unlike the weighted case, the complement of G_i, i.e. $G \setminus G_i$, can be replaced by just one demand point in $\mathcal{D}(G \setminus G_i)$.*

Step 2: determining α_G^* in B^*

We now consider the problem of locating α_G^* in B^*. If B^* is a subtree, $F(x, \mathcal{D}(G))$ is convex on every simple path of B^* [47]. Note that the structure of the cactus network G, except for the part of B^*, can be transformed to an equivalent tree structure. Thus, the $O(n \log n)$-time algorithm in [47] can be used to determine local center α_G^* in B^*. Also the linear-time algorithm for the weighted $V(G)/V(G)/1$ and $A(G)/V(G)/1$ problems in trees [54] can be applied here.

Suppose B^* is a cycle block. Observe that locating α_G^* in the cycle block B^* is very similar to locating 1-center in a cycle. In [65], Rayco et al. mentioned that the weighted continuous 1-center problem (i.e. weighted $A(G)/V(G)/1$ problem) in a cycle is solvable in $O(n \log n)$ time. Here, for completeness, we describe an algorithm to solve the weighted 1-center problem in a cycle block.

Weighted continuous 1-center problem in a cycle block Let $v_1, v_2, \ldots, v_t$ be the vertices of a cycle O. Let α^* denote an optimal 1-center of O we are interested in computing. We notice that there is exactly one edge in O not used by α^* to cover the vertices of O. We call this edge as the *optimal cut-edge* of α^*. Thus the 1-center on the path constructed by removing the optimal cut-edge from O is also an optimal 1-center of O. Thus our idea is to consider each edge as a cut-edge and compute the 1-center on the resulting path. The data structure described below is dynamic, which allows efficient updating of the structure as the cut-edge changes.

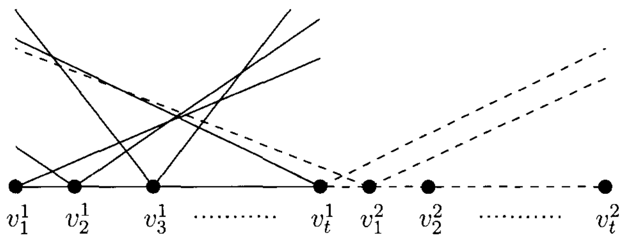


Figure 4.7: Locate α^* in a cycle block O.

The algorithm is described as follows. Consider Figure 4.7 for reference. The vertices on the cycle are indexed as $v_1, v_2, \ldots, v_t$ in counterclockwise order and the edge connecting the vertices v_{i-1} and v_i is indexed as $e_{i-1}, 1 < i \leq t$ ($e_t = \overline{v_t v_1}$). We put the $2t - 1$ vertices $\{v_1^1 = v_1, v_2^1 = v_2, \ldots, v_t^1 = v_t, v_1^2 = v_1, \ldots, v_{t-1}^2 = v_{t-1}\}$ on the real line. Let $Pos(z)$ denote the position of z on the real line. The positions of the $2t - 1$ vertices are determined in the following way. $Pos(v_1^1) = 0$, $Pos(v_i^1) = Pos(v_{i-1}^1) + l(e_{i-1})$, $1 < i \leq t$; and $Pos(v_1^2) = Pos(v_t^1) + l(e_t)$, $Pos(v_i^2) = Pos(v_{i-1}^2) + l(e_{i-1})$, $1 < i \leq t - 1$.

The path constructed by removing edge e_i from O is called the i-th path, which is the path from v_{i+1}^1 to v_i^2. Let V^i be the vertex set of the i-th path. The service cost function $f^i(x)$ on the i-th path is defined as

$$f^i(x) = \max_{v \in V^i} w(v)|d(x, v_1^1) - Pos(v)|,$$

where x is a point on the i-th path at a distance $d(x, v_1^1)$ from v_1^1. Let x^i denote an optimal 1-center of the i-th path. It is easy to compute $f^i(x)$ and determine x^i in linear time [18]. However, it is not efficient to separately compute functions $f^i(x), 1 \leq i \leq t$.

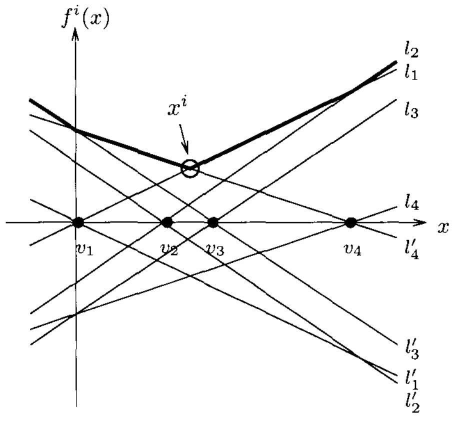


Figure 4.8: $f^i(x)$.

We can represent the function $w(v)|d(x, v_1^1) - Pos(v)|$ for all x by two straight lines through the point $(Pos(v), 0)$ with slopes $w(v)$ and $-w(v)$ (Figure 4.8). Then $f^i(x)$ is the upper envelope of $2t$ linear functions where t linear functions have positive slopes and t linear functions have negative slopes. Since $f^i(x)$ is convex, the optimal solution can be easily computed. Observe that the upper envelope of the lines generated by the $(i + 1)$-th path is constructed from the upper envelope of the lines generated by the i-path by simply removing the lines generated by v^1_{i+1} and inserting the lines generated by v^2_{i+1}. The upper envelope can be maintained by the algorithm proposed by Hershberger and Suri [43]. Each updating step can be performed in amortized logarithmic time, since the sequence of insertions and deletions of lines is already known [43]. We observe that the above approach also works if some of the vertices in O are hinge vertices and are attached to other components. If v is a vertex in a component attached to a hinge vertex, say v_i, the corresponding two lines generated by v will have slopes $w(v)$ and $-w(v)$ and they will go through the point $(Pos(v_i), b_v)$, where b_v is the weighted distance of v to v_i. Thus

Lemma 4.2.3 *Optimal solutions corresponding to all the cut-edges in O can be computed in total $O(n \log n)$ time. The storage space requirement is linear.*

In summary, we have the following theorem.

Theorem 4.2.4 *The weighted $A(G)/V(G)/1$ and $V(G)/V(G)/1$ problems in cactus networks can be solved in $O(n \log n)$ time using linear space.*

We also have the following result.

Theorem 4.2.5 *The four unweighted models ($V(G)/V(G)/1$, $V(G)/A(G)/1$, $A(G)/V(G)/1$, $A(G)/A(G)/1$) in cactus networks can be solved in $O(n)$ time using linear space.*

Proof The result for the models where $\mathcal{D}(G) = V(G)$ is in [18, 48]. From Observation 4.2.2, we note that in $O(n)$ time we can restrict the problems $A(G)/A(G)/1$ and $V(G)/A(G)/1$ to a cactus having at most one cycle. But then in this case the $A(G)/A(G)/1$ problem is equivalent to $A(G)/V(G)/1$ and $V(G)/A(G)/1$ is equivalent to $V(G)/V(G)/1$. □

4.2.2 The weighted continuous 2-center problem

In this section, an efficient algorithm for the weighted $A(G)/V(G)/2$ problem in cactus networks is proposed. As we know, locating an optimal 2-center in G is equivalent to finding a set of split-edges whose removal defines two connected components and optimal 1-centers of the resulting two components constitute an optimal 2-center solution of G. The split-edges in an optimal solution are called optimal split-edges.

In a tree network, the number of optimal split-edges is just one. However, for a cactus network the number of optimal split-edges is at most two. As a matter of fact, it can be shown that

Lemma 4.2.6 *Optimal split-edges in a cactus network G lie in one block B_i. If B_i is a subtree, there is one optimal split-edge, otherwise (B_i is a cycle) the number of optimal split-edges is two.*

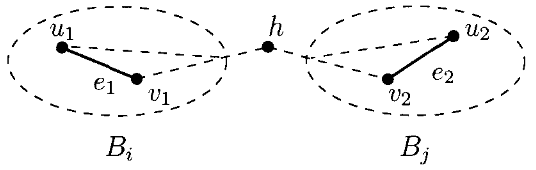


Figure 4.9: Lemma 4.2.6.

Proof Suppose that optimal split-edges lie in more than one block. Let α_1 and α_2 be the centers of the subnetworks obtained after the removal of the optimal split-edges from the cactus network. Let B_i and B_j be the blocks containing the split-edges $e_1 = \overline{u_1 v_1}$ and $e_2 = \overline{u_2 v_2}$ respectively. (Figure 4.9). Assume that u_1 and u_2 are served by α_1, and v_1 and v_2 are served by α_2. Let h be a hinge vertex lying between B_i and B_j, that is, the shortest path between any vertex in B_i and any vertex in B_j passes through h. Such a hinge vertex h always exists, since B_i and B_j are two different blocks. Since the subnetwork served by each 1-center is connected, and since h lies in the shortest paths $\pi(u_1, u_2)$ and $\pi(v_1, v_2)$, h is served by both α_1 and α_2, which is impossible. Hence, optimal split-edges must lie in one block of G. Therefore, if the block containing an optimal split-edge set is a subtree, then there is only one split-edge in the set, and if the block containing optimal split-edge set is a cycle, then there are two split-edges in the set. □

Let Δ denote a set of split-edges of G where, if $|\Delta| = 2$, both the split-edges come from one cycle block. Let G^1_Δ, G^2_Δ denote the two subnetworks obtained after the split-edges in Δ are removed from G. Let $\gamma_{G^i_\Delta}$ be the optimal service cost of the $A(G^i_\Delta)/V(G^i_\Delta)/1$ problem, $i = 1, 2$. Let $\phi(\Delta) = \max\{\gamma_{G^1_\Delta}, \gamma_{G^2_\Delta}\}$. A split-edge set Δ^* is called *an optimal split-edge set* of G if $\phi(\Delta^*) = \min_{\Delta \subseteq B_i, i=1,\ldots,t'} \phi(\Delta)$. Here $B_1, B_2, \ldots, B_{t'}$ are the blocks in G.

Step 1: locating the optimal split-block B^*

We now focus on exploring the properties of the optimal split-edge set in cactus networks.

Lemma 4.2.7 *(Figure 4.6(a)) Let $G_1, \ldots, G_k$ be the subnetworks of G attached to a hinge vertex o. In $O(n \log n)$ time we can either identify an optimal split-edge set or determine the subnetwork attached to o that contains an optimal split-edge set.*

Proof Suppose that $F(o, V(G_1)) \geq F(o, V(G_2)) \geq F(o, V(G_j)), j = 3, \ldots, k$. Let Δ_i denote the set of edges of G_i incident to o, $i = 1, \ldots, k$. Obviously, $|\Delta_i| \leq 2$ for all i. The service cost $\phi(\Delta_j)$ with a split-edge set Δ_j, $j \neq 1, 2$, is greater than $\max\{\gamma_{G_1}, F(o, V(G_2))\}$, since G_1 and G_2 are served by the same 1-center. But, the service cost $\phi(\Delta_1)$ is no more than $\max\{F(o, V(G_2)), \gamma_{G_1}\}$. Therefore, there exists an optimal split-edge set in $G_1 \cup G_2$.

In the following, we determine whether G_1 or G_2 contains an optimal split-edge set. We consider three cases based on the service costs $\phi(\Delta_1)$ and $\phi(\Delta_2)$.

- $\phi(\Delta_1)$ is determined by the service cost of the center in subnetwork G_1. It implies that G_1 contains an optimal split-edge set.
- $\phi(\Delta_2)$ is determined by the service cost of the center in subnetwork G_2. It implies that G_2 contains an optimal split-edge set.
- $\phi(\Delta_1)$ is determined by the service cost of the center in subnetwork $G \setminus G_1$ and $\phi(\Delta_2)$ is determined by the service cost of the center in subnetwork $G \setminus G_2$. In this case, if $\phi(\Delta_1) \leq \phi(\Delta_2)$, Δ_1 is an optimal split-edge set, otherwise (i.e., $\phi(\Delta_2) \leq \phi(\Delta_1)$) Δ_2 is an optimal split-edge set.

In Theorem 4.2.4 we have shown that the weighted $A(G)/V(G)/1$ problem in cactus networks can be solved in $O(n \log n)$ time. Therefore, it takes $O(n \log n)$ time to either identify an optimal split-edge set or determine the subnetwork that contains an optimal split-edge set. □

Lemma 4.2.8 *(Figure 4.6(b)) Let $G_1, \ldots, G_k$ be the subnetworks of G attached to a cycle block B_o. In $O(n \log n)$ time, we can either identify an optimal split-edge set, locate the block B^* containing an optimal split-edge set, or determine the subnetwork attached to B_o that contains an optimal split-edge set.*

The proof of Lemma 4.2.8 is very similar to that of Lemma 4.2.7, and, therefore, is omitted here.

We can recursively search either G_1 or G_2 that contains an optimal split-edge set. Thus, in $O(n \log n \cdot \log |V_G|)$ time, we either identify an optimal split-edge set or determine the block B^* that contains an optimal split-edge set Δ^*.

Step 2: computing Δ^* in B^*

If B^* is a subtree, Δ^* contains exactly one edge. We can locate Δ^* in B^* recursively using a centroid tree decomposition [55] of B^*. Each recursive step takes $O(n \log n)$ time (note that Lemma 4.2.7 also works for T-vertices). Therefore, Δ^* in a subtree B^* can be computed in $O(n \log n \cdot \log |B^*|)$ time.

When B^* is a cycle, an optimal split-edge set Δ^* contains two edges. Let $v_1, v_2, \dots, v_t$ be the vertices of B^* in counterclockwise order, and let $e_1 = \overline{v_1 v_2}, \dots, e_t = \overline{v_t v_1}$ be the edges of B^* in counterclockwise order. Let $[e_i, e_j]$ denote the chain of B^* in counterclockwise order from e_i to e_j in B^*. Similarly, let $[v_i, v_j]$ denote the sequence of vertices in counterclockwise order from v_i to v_j in B^*.

If one of the two optimal split-edges in B^* is known, we can locate the other one in $O(n \log n \cdot \log |B^*|)$ time, since it is equivalent to locating one optimal split-edge in a subtree. Here the subtree is path-like. An edge $e'_i \in E(B^*)$ is called a *match-edge* of e_i if $\phi(\{e_i, e'_i\}) = \min_{e_k \in E(B^*) \text{ and } e_k \neq e_i} \phi(\{e_i, e_k\})$. The match-edge of an edge e_i may not be unique, but all the match-edges must be consecutive along the path $\pi :< v_{i+1}, v_{i+2}, \dots, v_1, \dots, v_{i-1} >$. This is due to the unimodality property of $\phi(\{e_i, e\})$ as e moves away from e_i along the path π. For uniqueness, the last match-edge is paired with e_i. We cannot afford to separately find the match-edge of each edge in B^*. However, the following simple observation is helpful. Assume that e'_i is the match-edge of $e_i, i = 1, \dots, t$, and $e_j \in (e_i, e'_i)$. The match-edge e'_j of e_j must lie in $[e'_i, e_i]$ (the unimodality property).

The algorithm to locate Δ^* in the cycle block B^* proceeds as follows. The process starts from e_1. After the match-edge e'_i of e_i is found, the first edge $e_j \in [e'_i, e_i]$ which satisfies $\phi(\{e_{i+1}, e_j\}) < \phi(\{e_{i+1}, e_{j+1}\})$, is taken to be the match-edge e'_{i+1} of e_{i+1} (the unimodality property). Thus, the running time to compute Δ^* in B^* is $O(|B^*|)$ times the time complexity of computing the maximum service cost $\phi(\Delta)$ for a given split-edge set $\Delta = \{e_i, e_j\}$.

Computing $\phi(\Delta = \{e_i, e_j\})$: Let G'_k denote the subnetwork of B^*, attached to v_k $(1 \leq k \leq t)$. Let i_1 and i_2 be two different integers in $[1, t]$ such that $F(v_{i_1}, V(G'_{i_1})) \geq F(v_{i_2}, V(G'_{i_2})) \geq F(v_k, V(G'_k))$, for any k, $1 \leq k \leq t$ and $k \neq i_1, i_2$. The following lemma is crucial to the algorithm of computing $\phi(\Delta = \{e_i, e_j\})$.

Lemma 4.2.9 *Two centers corresponding to a given split-edge set $\Delta \in B^*$ lie in either block B^*, subnetwork G'_{i_1}, or subnetwork G'_{i_2}.*

Proof Suppose that the vertices in $[v_{i+1}, v_j]$ are contained in $V(G^1_\Delta)$. It is clear that an optimal 1-center of G^1_Δ (resp. G^2_Δ) lies either in B^* or in some subnetwork G'_k where $k \in [i+1, j]$ and $F(v_k, V(G'_k)) \geq F(v_f, V(G'_f))$, for all $f \in [i+1, j]$ (resp. $k \in [j+1, i]$ and $F(v_k, V(G'_k)) \geq F(v_f, V(G'_f))$, for all $f \in [j+1, i]$).

If possible, suppose that one of the two centers, say α_1, lies in G'_k where $k \neq i_1, i_2$. In this case, the service cost $\gamma_{G^1_\Delta}$ must be less than or equal to $F(v_k, V(G'_k))$. The vertices v_{i_1} and v_{i_2} are not served by α_1, since if either v_{i_1} or v_{i_2} is served by α_1 then $\gamma_{G^1_\Delta}$ is at least $F(v_{i_2}, V(G'_{i_2}))$ ($\geq F(v_k, V(G'_k))$). Therefore, the service cost $\gamma_{G^2_\Delta}$ must be at least $F(v_{i_2}, V(G'_{i_2}))$, since both v_{i_1} and v_{i_2} are served by α_2. We can see $\gamma_{G^1_\Delta} \leq F(v_k, V(G'_k)) \leq F(v_{i_2}, V(G'_{i_2})) \leq \gamma_{G^2_\Delta}$. Therefore, $F(\{v_k, \alpha_2\}, V(G)) \leq \max\{F(v_k, V(G^1_\Delta)), F(\alpha_2, V(G^2_\Delta))\} = F(\alpha_2, V(G^2_\Delta)) = \phi(\Delta)$. We can use the vertex v_k as the center instead of α_1 without increasing the service cost $\phi(\Delta)$. Hence G'_k where $k \neq i_1, i_2$ can be eliminated from searching two centers for any given split-edge set $\Delta \in B^*$. □

Suppose that $v_{i_1} \in V(G^1_\Delta)$. We can determine $\phi(\Delta)$ by computing

- an optimal center α'_1 of G^1_Δ constrained to lie on B^*,
- an optimal center α''_1 of G^1_Δ constrained to lie on G'_{i_1},
- an optimal center α'_2 of G^2_Δ constrained to lie on B^*, and
- an optimal center α''_2 of G^2_Δ constrained to lie on G'_{i_2} if $v_{i_2} \in V(G^2_\Delta)$, otherwise, α''_2 is undefined.

All α'_1 and all α'_2 are restricted to be on the cycle block, B^*. Hence, they can be found in $O(n \log n)$ time by the algorithm for the weighted continuous 1-center problem in a cycle block described in Section 4.2.1.

Since computing α''_2 is similar to computing α''_1, we concentrate on computing α''_1 only. Let $G' = G^1_\Delta \setminus G'_{i_1}$. We can see that G' changes as Δ changes (refer to Figure 4.10(a)). Let x_{i_1} be an optimal 1-center of G'_{i_1}, and B denote the block in G'_{i_1} where x_{i_1} lies.

Lemma 4.2.10 *(Figure 4.10(b)) α''_1 lies in one of the blocks that the shortest path $\pi(v_{i_1}, x_{i_1})$ goes through.*

Proof Suppose that α''_1 lies in some block B' that $\pi(v_{i_1}, x_{i_1})$ does not go through. Let h be the closest vertex to α''_1 in the blocks that $\pi(v_{i_1}, x_{i_1})$ goes through. Clearly, h is a hinge

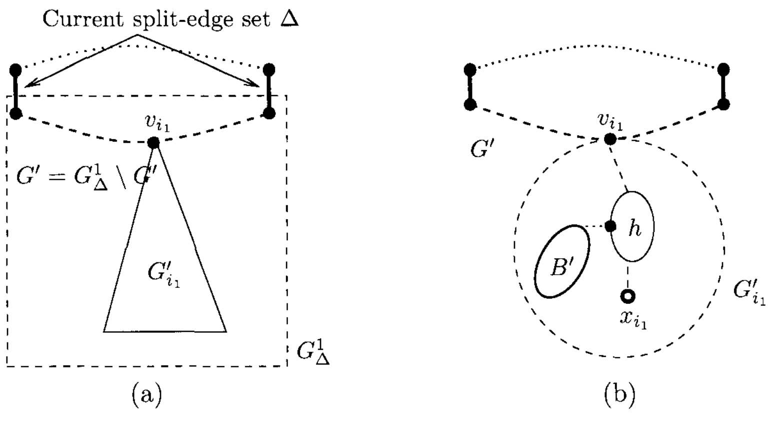


Figure 4.10: An example with a split-edge set Δ.

vertex. It is not difficult to see that the service cost $F(h, V(G^1_\Delta))$ is less than the service cost $F(\alpha''_1, V(G^1_\Delta))$. Therefore, α''_1 cannot lie in B'. □

Forcing the convexity of $F(x, V(G'_{i_1}))$ on an edge Unlike in tree structures, the service cost function $F(x, V(G))$ in a cactus network may not be convex as x moves from one endpoint of the edge to the other [47]. Fortunately, for a cactus network, it is possible to force the service cost function to be convex on each edge of the *block path*, denoted by $P(v_{i_1}, x_{i_1})$, which is a list of blocks that the path $\pi(v_{i_1}, x_{i_1})$ goes through. This is achieved by adding extra vertices as follows. If a block on the block path is a subtree, then clearly the service cost function is convex on each edge of this block. When a block on the block path is a cycle, for every vertex v in this block, we find its *match-point* v' in the same block such that $d^c_{v,v'} = d^{cc}_{v,v'}$ where $d^c_{v,v'}$ and $d^{cc}_{v,v'}$ are the respective clockwise and counterclockwise distances from v to v' in the block. Then, v' is added as a vertex to the network by breaking the edge containing v'. We assign weight zero to these added vertices. In this way, the service cost function $F(x, v)$, for every v is monotone as x ranges over an edge in the updated network. Due to the insertion of match-points, the service cost function on each edge is therefore convex. The total number of match-points added to force the convexity is no more than $2n$. These match-points can easily be determined in $O(n)$ time.

An algorithm to locate α''_1 in G'_{i_1} In the following, we assume that G'_{i_1} contains the *match-point* vertices in the cycle blocks of the block path. Also, G^1_Δ, G' and G'_{i_1} are preprocessed to construct two-level tree decomposition data structures, described in Section 4.1.1.

Note that a cactus network is a partial 2-tree. Thus, a two-level tree decomposition structure $\overline{\mathcal{TD}}(G'_{i_1})$ of G'_{i_1} is built in $O(|G'_{i_1}|\log^2|G'_{i_1}|)$ time (Lemma 4.1.4). The storage space requirement is $O(|G'_{i_1}|\log|G'_{i_1}|)$. Using this data structure, the service cost $F(x, V(G'_{i_1}))$ of any point x in G'_{i_1} can be answered in $O(\log^2|G'_{i_1}|))$ time.

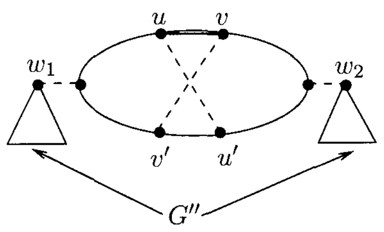


Figure 4.11: $w_1 \in V_u$ and $w_2 \in V_v$.

We first show that the optimal local 1-center of G'_{i_1} on an edge $e = \overline{uv}$ of G'_{i_1} can be computed in $O(\log^3 n)$ time. Let u be the counterclockwise neighbor of v (Figure 4.11). Let V_u be the set of vertices in G'_{i_1} which are closer to u than v, and let $V_v = V(G'_{i_1}) \setminus V_u$. The vertices in V_v are closer to v than u. There are $O(\log n)$ subtrees of $\overline{\mathcal{TD}}(G'_{i_1})$, say H, spanning all the vertices of G'_{i_1} and there is a 2-separator (or 1-separator, but it is safe to only consider 2-separator) between any point in the edge $\overline{uv}$ and each of the subtrees in H. We start from a node of $\overline{\mathcal{TD}}(G'_{i_1})$ that contains both the vertices u and v. Let w_1 and w_2 be the 2-separator between u (resp. v) and a subtree G'' (an element of H). We can compute $d(u, w_i)$ and $d(v, w_i)$ in constant time using the results in [22]. Clearly, if $w_1, w_2 \in V_u$ (or $w_1, w_2 \in V_v$), then all the vertices in G'' belong to V_u (or V_v); if $w_1 \in V_u, w_2 \in V_v$ (resp. $w_1 \in V_v, w_2 \in V_u$), then all the vertices in G'', whose shortest path to u goes through w_1, belong to V_u (resp. V_v) and the remaining vertices in G'' belong to V_v (resp. V_u). The latter case can be observed in Figure 4.11. Let u' and v' be the match-points of u and v, respectively, on the cycle block that contains $\overline{uv}$. All the vertices on the counterclockwise path from u to v' together with the vertices in the components attached to the path are closer to u than v. The shortest paths from u to these vertices do not use the edge $\overline{v'u'}$. These vertices determine V_u. Similarly, all the vertices on the clockwise path from v to u', together with the vertices in the components attached to the path, are closer to v than u. The shortest paths from v to all these vertices also do not use the edge $\overline{v'u'}$. These vertices determine V_v. From $\overline{\mathcal{TD}}(G'_{i_1})$, the vertices in G'' that belong to V_u can be reported in a sorted list of distances from w_1 in $O(\log n)$ time. Similarly, all distances from w_2 to the vertices in G'' that belong to V_v can be reported in a sorted list in $O(\log n)$ time. Therefore, V_u and V_v can be represented by $O(\log n)$ sorted lists and the maximum service cost function of each such list is monotone on $\overline{uv}$. More precisely, the maximum service cost function of each list in V_u monotonically increases on $\overline{uv}$ from u to v and the maximum service cost function of each list in V_v monotonically decreases on $\overline{uv}$ from u to v. Since the maximum service cost to G'_{i_1} of a point on $\overline{uv}$ can be computed in $O(\log^2 n)$ time (Lemma 4.1.4), we have the following lemma.

Lemma 4.2.11 *The optimal local center of G'_{i_1} constrained to lie on a given edge can be computed in $O(\log^3 n)$ time. The preprocessing step takes $O(n\log^2 n)$ time and $O(n\log n)$ space.*

Thus, in the following, it is assumed that the optimal local center of G'_{i_1} on every edge of G'_{i_1} is already known. The remaining step to compute α''_1 has two parts. We first determine the block that contains α''_1 and then determine α''_1 within this block. Suppose that $u_1 = v_{i_1}, u_2, \ldots, u_k$ are the hinge vertices lying on the path $\pi(v_{i_1}, x_{i_1})$.

Locating the block $B_{u_i^*}$ containing α''_1: Observe that the farthest (weighted) vertex v'_j in G'_{i_1} to u_j must lie below u_j (further away from v_{i_1} compared to u_j), otherwise, x_{i_1} cannot be a weighted 1-center of G'_{i_1}. Therefore, we can have the following lemma.

Lemma 4.2.12 *For any j, $1 \le j \le k$, if the farthest (weighted) vertex to u_j in G^1_Δ comes from $G' = G^1_\Delta \setminus G'_{i_1}$, then α''_1 can not lie on the block path $P(u_j, x_{i_1})$; otherwise (the farthest (weighted) vertex to u_j lies in G'_{i_1}), α''_1 can not lie on the block path $P(v_{i_1}, u_j)$.*

Using Lemma 4.2.12, it is easy to locate the block that contains α''_1 in $O(\log n)$ time. In each step, we need to compute $F(u_i, V(G^1_\Delta))$, $1 \le i \le k$. We already know that, for any u_i, $F(u_i, V(G'_{i_1}))$ can be computed in $O(\log^2 n)$ time using the structure $\overline{\mathcal{TD}}(G'_{i_1})$. Since $F(u_i, V(G^1_\Delta)) = \max\{F(u_i, V(G')), F(u_i, V(G'_{i_1}))\}$, we need to compute $F(u_i, V(G'))$ effciently. Observe that the upper envelope $\overline{f^i}(x)$ of the lines generated by the vertices of G' can be dynamically maintained. There are $O(n)$ insertions and deletions in all, and each operation costs $O(\log n)$ amortized time [43]. Once $\overline{f^i}(x)$ is known, $F(u_i, V(G'))$ can be computed in $O(\log n)$ time for any u_i. Therefore,

Lemma 4.2.13 *After a preprocessing step requiring $O(n \log^2 n)$ time, the block of G'_{i_1} containing α''_1, say $B_{u_i^*}$, can be found in $O(\log^2 n)$ time. The storage space requirement is $O(n \log n)$.*

Locating α''_1 in the block $B_{u_i^*}$: We now prune away most of the edges of $B_{u_i^*}$ using the information of optimal local centers on edges. Note that u_{i^*} is the closest hinge vertex of $B_{u_i^*}$ to v_{i_1}. We partition the edges of $B_{u_i^*}$ into two groups if $B_{u_i^*}$ is a cycle block: $\pi_{ccw} = [u_{i^*}, \ldots, u'_{i^*}]$ and $\pi_{cw} = [u_{i^*}, \ldots, u'_{i^*}]$, where u'_{i^*} is the match-point of u_{i^*} and the edges of π_{ccw} and π_{cw} are traversed in counterclockwise and clockwise orientations respectively. The edges of π_{ccw} and π_{cw} are, therefore, ordered in increasing order from u_{i^*}. Note that, when $B_{u_i^*}$ is a subtree, only the edges on $\pi(u_i^*, u_{i+1}^*)$ need to be considered and the procedure for this case is similar to the one described in the following. Thus, we assume that $B_{u_i^*}$ is a cycle block in the following.

We consider the π_{ccw} chain only. The process is similar for the other chain. The following lemma eliminates some edges of $B_{u_i^*}$.

Lemma 4.2.14 *If the optimal local center q of G'_{i_1} on an edge $e \in \pi_{ccw}$ has a larger service cost to G'_{i_1} than a point y on another edge of π_{ccw} closer to v_{i_1}, then α''_1 cannot lie on e.*

Proof It is clear that $F(x, V(G^1_\Delta)) = \max\{F(x, V(G')), F(x, V(G'_{i_1}))\}$ for any x in G'_{i_1}. Since the local minimum service cost to G'_{i_1} on e is greater than $F(y, V(G'_{i_1}))$ and y is closer to v_{i_1} than any point in e, the service cost $F(y, G^1_\Delta)$ is always less than the service cost of any point on e. □

As a consequence of the above lemma we can order the edges of π_{ccw} in increasing distances from u_i^* and with decreasing optimal local center service costs. Similar ordering of the edges is performed on π_{cw}. These orderings are possible without the knowledge of G', and, therefore, are done once. The rest of the edges of B_i^* are labeled and will not be considered further. We only need to consider the unlabeled edges of $B_{u_i^*}$ to find α''_1. The following lemma allows us to prune the unlabeled edges of $B_{u_i^*}$ further.

Lemma 4.2.15 *Consider any unlabeled edge $e = \overline{uv}$ in π_{ccw} (u is closer to u_{i^*} than v) and its optimal local center q to G'_{i_1}. If the service cost of G^1_Δ from q is determined by some vertex in G', then all the unlabeled edges in $\pi_{ccw}[v, \ldots, u'_{i^*}]$ cannot contain α''_1. Otherwise (i.e., $F(q, V(G')) < F(q, V(G'_{i_1}))$), all the unlabeled edges of $\pi_{ccw}[u_{i^*}, \ldots, u]$ cannot contain α''_1.*

The reason why we can not directly use Lemma 4.2.15 on all the edges in π_{ccw}, is that, if $e = \overline{uv}$ is a labeled edge, then it is possible to have the case where $F(q, V(G')) < F(q, V(G'_{i_1}))$ (q is the local center of e to G'_{i_1}) and α''_1 lies in $\pi_{ccw}[u_{i^*}, \dots, u]$.

Therefore, we can apply the binary-search technique to the unlabeled edges in π_{ccw} until one unlabeled edge is left, say e_{ccw}. We can similarly determine e_{cw} by performing a binary search on π_{cw}. Since the service cost of any point in G^1_Δ to G^1_Δ can be computed in $O(\log^2 n)$ time (Lemma 4.1.4),

Lemma 4.2.16 *The number of candidate edges on which α''_1 could lie can be narrowed down to at most two in $O(\log^3 n)$ time.*

The remaining step of locating α''_1 on e_{ccw} and e_{cw} is very similar to computing the optimal local center of G'_{i_1} on a given edge.

The above results can now be summarized as follows. After an $O(n \log^2 n)$-time processing, either we already have an optimal split-edge set or know the block B^* that contains an optimal split-edge set Δ^*. If B^* is a subtree, then it takes an additional $O(n \log^2 n)$ time to compute an optimal split-edge and the optimal service cost. Otherwise, B^* is a cycle block. It is easy to see that finding G'_{i_1} and G'_{i_2} and adding match-points can be done in linear time. Due to the unimodality property of split-edges on a simple path, we only need to compute the service costs for $O(|B^*|)$ pairs of split-edges. After an $O(n \log^3 n)$-time preprocessing (including building two-level tree decomposition data structures and computing local centers), the service cost for each pair of split-edges can be computed in $O(\log^3 n)$ time. Therefore, we can claim Theorem 4.2.17.

Theorem 4.2.17 *The weighted $A(G)/V(G)/2$ problem in a cactus network can be solved in $O(n \log^3 n)$ time complexity. The storage space complexity is $O(n \log n)$.*

4.2.3 Various p-center problems

Frederickson and Johnson [30] designed an $O(n \log n)$-time algorithm for the unweighted $V(G)/V(G)/p$ problem in a cactus network. They showed that a feasibility test in a cactus network where vertices are unweighted, can be performed in linear time (Lemma 13 in [30]). Using this linear-time feasibility test and a succinct representation of the set of all the inter-vertex distances, the unweighted $V(G)/V(G)/p$ problem in a cactus network is solvable in $O(n \log n)$ time [30].

The weighted $V(G)/V(G)/p$ problem

Actually, the algorithm for feasibility tests described in [30] can also be applied to the case when the demand points in $V(G)$ are weighted. In this case, for a given service cost r, the demand points may now have different covering radii. We present below a simple transformation that transforms a feasibility test in the weighted model to a feasibility test in the unweighted model.

In the weighted model, we have a cactus where each demand vertex v_i is associated with a nonnegative covering radius $r_i = r/w_i$. The problem is to find a subset of vertices, X, of minimum cardinality, such that for each vertex $v_i \in V(G)$, $F(X, v_i) \leq r_i$. Lemma 13 in [30] provides an $O(n)$ algorithm for the case where $r_i = r_c$, for each $v_i \in V(G)$. We can convert the above weighted model to an equivalent unweighted model as follows. Each vertex v_i is augmented by a new edge, say $\overline{v_i v_i'}$ of length $r_c - r_i$, where $r_c = \max\{r_j : v_j \in V(G)\}$. Let G' be the augmented graph with $2n$ vertices. Clearly, G' is a cactus network. We now associate a radius r_c with each vertex v_i and v_i'. The feasibility test on G is equivalent to a feasibility test on G', and, therefore, can be done in linear time. Thus,

Lemma 4.2.18 *The feasibility test in a weighted model of a cactus network can be performed in $O(n)$ time.*

Frederickson and Johnson [30] gave a succinct representation of the inter-vertex distances of the vertices of a cactus. The representation allows one to implement an efficient binary search on the distances. Similarly, the set of all inter-vertex distances in a partial 2-tree [32] has a special structure that enables searching the set without explicitly generating the entire set in advance. Indeed, the set of inter-vertex distances can be implicitly represented by a set of $O(n \log n)$ sorted lists. Each sorted list is associated with weighted distances from a given weighted vertex u to some subset V_u of the vertices of G whose shortest path distances to u pass through a separator vertex. These distances to the separator vertex are kept in sorted order. There are $O(\log n)$ such sorted lists for every vertex u. In this way the inter-vertex distances of any partial 2-tree can be represented by a set of $O(n \log n)$ sorted lists. This representation is very similar to the succinct representation of all inter-vertex distances in a tree proposed by Megiddo et al. [56]. Therefore, using the method proposed by Megiddo et al. [56], one can solve the discrete p-center problem in a weighted cactus network for any p, in $O(n \log^2 n)$ time.

Theorem 4.2.19 *The weighted $V(G)/V(G)/p$ problem in a cactus network can be solved in $O(n \log^2 n)$ time. The storage space requirement is $O(n \log n)$.*

Weighted $A(G)/V(G)/p$ and unweighted $V(G)/A(G)/p$ problems

From the fact that Lemma 4.2.18 is also applicable to the test corresponding to the weighted $A(G)/V(G)/p$ and the unweighed $V(G)/A(G)/p$ models, we can obtain an $O(n^2)$ algorithm for these problems. Since the numeric operations of the feasibility test are additions/subtractions and comparisons, we can directly apply Megiddo's generic parametric-serching technique [52] and get the $O(n^2)$ time algorithm. Therefore, we have the following theorem.

Theorem 4.2.20 *The weighted $A(G)/V(G)/p$ and unweighted $V(G)/A(G)/p$ problems in a cactus network can be solved in $O(n^2)$ time.*

The unweighted $A(G)/A(G)/p$ problem

A candidate set containing the optimal solution value for the $A(G)/A(G)/p$ model for a general graph is characterized in Tamir's paper [68]. In spite of the nice structure, this set is not of polynomial cardinality, even for cactus networks. Nevertheless, in the discussion below, we show that the $A(G)/A(G)/p$ problem in a cactus is efficiently solvable.

The idea is again to use the feasibility test parametrically [52]. First, we note that, for this model, p can be significantly larger than n. Nevertheless, the allocation of the p centers to the edges can be properly bounded. Let $t(e)$ denote the number of centers established at optimality on an edge e of length $l(e)$. Therefore, $\lceil l(e)/2c^* \rceil - 1 \leq t(e) \leq \lceil l(e)/2c^* \rceil + 1$ where c^* is the optimal service cost. It is shown in [68] that $p - m \leq l(G)/2c^* \leq p + m$ where m and $l(G)$ are the number of edges and the total length of the edges in G respectively. Therefore,

$$\max\{0, l(e)(p-m)/l(G) - 1\} \leq t(e) \leq \min\{p, l(e)(p+m)/l(G) + 1\}.$$

Hence, $t(e)$ can a priori bounded in a range of length $O(n)$ for cactus networks. In particular, when applying the test parametrically, we will need $O(\log n)$ tests per edge to find the exact value of $t(e)$. An $O(n \log n)$ test for a more general class is mentioned in [33, 68]. It is not hard to obtain the following theorem by using Megiddo's generic parametric-searching technique [52].

Theorem 4.2.21 *The $A(G)/A(G)/p$ problem in a cactus network can be solved in $O(n^2 \log^2 n)$ time.*

We remark that when the data of the above p-center problems are integers or even rational, and "relatively small", (e.g. sub-exponential in n), better complexity bounds can be achieved by applying an efficient search for rational techniques [75].

For the weighted $A(G)/V(G)/p$ model, the optimal objective value is of the form

$$w(u)w(v)L(u,v)/(w(u)+w(v)),$$

where $L(u,v)$ is the length of some simple path connecting u and v for some pair of vertices $u, v \in V(G)$ [47]. For the $A(G)/A(G)/p$ model, the optimal solution value is of the form M/q, where M is the sum of the edge lengths of an Eulerian tour of some subgraph of G, and q is an integer, $1 \leq q \leq 4p$. The respective value for the $V(G)/A(G)/p$ model is of the form M/q, where M is the sum of the edge lengths of an Eulerian tour of some subgraph of G, and q is integer, $1 \leq q \leq 4$ [68].

Assuming integer data, denote $w_{max} = \max_{v \in V(G)} \{w(v)\}$. Then, observing that $M \leq 2l(G)$ and using the results in Zemel [75], we conclude that the weighted $A(G)/V(G)/p$, the $V(G)/A(G)/p$ and the $A(G)/A(G)/p$ problems can be solved in $O(n \log(n + l(G) + w_{max}))$, $O(n \log(n + l(G)))$ and $O(n \log n \log(n + l(G) + p))$ times, respectively.

4.3 Summary

In this chapter we have studied the center problems in tree-like networks, i.e., partial k-trees, cactus networks, and proposed non-trivial algorithms to solve a variety of problems. When the underlying network is a partial k-tree, we study the weighted discrete p-center problem and present an efficient algorithm for relatively small p. The running time of our algorithm is $O(pn^p \log^k n)$, which is better than the $O(p^2 n^{k+2})$ result of Granot and Skorin-Kapov [32] when $p < k + 2$. We also discuss the weighted continuous p-center problem and devise an $O(p^2 k^{k+1} n^{2k+3} \log n)$-time algorithm for it.

For a cactus network, we have proposed, for the first time, an $O(n \log n)$-time algorithm to solve the weighted continuous 1-center problem, an $O(n \log^3 n)$-time algorithm for the weighted continuous 2-center problem, and efficient algorithms for various p-center problems where p is a part of the input. Our algorithms for the p-center problems are based on the parametric-searching technique. In particular, we propose an $O(n\log^2 n)$-time algorithm for the weighted discrete p-center problem, $O(n^2)$ algorithms for the weighted continuous p-center problem, and the unweighted discrete p-center problem with a demand set of infinite size, and an $O(n^2\log^2 n)$ algorithm for the general p-center problem.

Many issues in a cactus network are still unresolved. For instance, it would be interesting to find out whether there exists an optimal linear-time algorithm for the weighted 1-center problem. We conjecture that the weighted $A(G)/V(G)/p$ problem and the unweighted $V(G)/A(G)/p$ and $A(G)/A(G)/p$ problems can be solved in subquadratic time by designing a polylog parallel algorithm for the feasibility test, and using the results in Megiddo [53]. For example, we suspect that the $O(\log^3 n)$ parallel time algorithm of Wang [74] for the test on trees, can be extended to cactus networks. If indeed, there is an $O(\log^q n)$ parallel algorithm for cactus networks (with $O(n)$ processors), Megiddo [53] implies an $O(n\log^{q+1} n)$ serial agorithm for the weighted $A(G)/V(G)/p$ problem in cactus networks. To obtain the result in Theorem 4.2.21 we have used an existing $O(n\log n)$ feasibility test. We suspect that an $O(n)$ test for $A(G)/A(G)/p$ in a cactus can be derived by properly modifying the test for $V(G)/V(G)/p$ in [30]. This will lead to the improved bound $O(n^2)$ for $A(G)/A(G)/p$ in a cactus network.

The most challenging problem is to find more efficient algorithms to solve the p-center problems in edge-weighted partial k-trees of bounded treewidth.

Chapter 5

Conditional extensive facility location in trees

In this chapter, we consider the problems of locating a path-shaped or tree-shaped facility in a tree under the condition that existing facilities are already located and propose optimal algorithms for them, which improve the recent results of $O(n \log n)$ by Tamir et al. [70]. Our algorithms are based on the parametric-pruning technique introduced in Section 3.2.2. The formal definitions of the problems are descried as follows.

Let $T = (V(T), E(T), w, l)$ be the underlying tree network. A subtree network is called *discrete* if all its leaf points are vertices of T, and otherwise is called *continuous*. Let S represent the set of existing facilities, which by itself can be a subtree network or even a forest network [70]. We assume that the size of S is at most $O(n)$. Let Γ_1 (resp. Γ_2) be the set of all the continuous path networks (resp. subtree networks) in T whose lengths are at most a predefined nonnegative value L_c, and let Φ_1 (resp. Φ_2) be the set of all the discrete path networks (resp. subtree networks) in T whose lengths are at most L_c. The goal is to establish one facility K^*, either a path network in Γ_1/Φ_1 or a subtree network in Γ_2/Φ_2, such that

$$F(K^* \cup S, V(T)) = \min_{K \in \Gamma_1(\text{ or } \Phi_1, \Gamma_2, \Phi_2)} F(K \cup S, V(T)).$$

When the facility K is selected from Γ_1 or Γ_2, we call the model *continuous*, and if K is chosen from Φ_1 or Φ_2, we call the model *discrete*. An extensive facility is *valid* if it is in Γ_1 (or $\Phi_1/\Gamma_2/\Phi_2$) for the continuous path-shaped (or discrete path-shaped/continuous tree-shaped/discrete tree-shaped) center problem.

Since a problem in the unconditional model is a special case of the conditional one, it suffices to develop algorithms for problems in the conditional model only.

Organization of the chapter In Section 5.1, we present the main ideas of our algorithms. The linear-time algorithms for the conditional path-shaped and tree-shaped center problems in tree networks are provided in Sections 5.2 and 5.3 respectively. Section 5.4 gives a brief summary and shows that our approach can be extended to optimally solve the problem for more general service cost functions.

5.1 Main idea of our algorithms

Given an extensive facility K and a set S of existing facilities, a vertex $v \in V(T)$ is called a *dominating vertex* of $K \cup S$ if $w(v) \cdot d(K \cup S, v) = F(K \cup S, V(T))$. Observe that, in the center problem, an optimal facility $K^* \cup S$ always has an equal or smaller service cost to the dominating vertices of a valid facility $K \cup S$. In other words, given such dominating information of a valid facility K, we are able to determine the relative location of an optimal facility K^* with respect to the location of this valid facility. In the Path Lemma and Tree Lemma discussed later, we will see how this idea works in solving our problems.

The main idea of the proposed algorithms is to 'prune' the vertices that do not determine the optimal service cost (i.e., vertices which are not dominating), and to 'shrink' the facility if some path or subtree network is known to be a part of an optimal facility. In the following, we present an algorithm to locate non-dominating vertices in an optimal solution.

5.1.1 Locating non-dominating vertices in an optimal solution

For the path/tree-shaped center problems, more idea is needed in order to make the parametric-pruning work. In the conditional model, we cannot afford to keep the information of the existing facilities in S at each pruning iteration, as the size of S could be $O(n)$. However, it is not difficult to design a linear-time computation step to find the distance $d(S, v)$ for each vertex $v \in V(T)$ [70]. Arbitrarily choose one vertex as the root of T. Let T_v denote the subtree rooted at v. In the first round, visit the vertices in a bottom-up manner, and, for the current visiting vertex v, compute its distance to the closest existing facility within T_v, i.e., $d(S \cap V(T_v), v)$. In the second round, visit the vertices in a top-down manner and for the visiting vertex v, compute its distance to the closest existing facility outside T_v, i.e.,

$d(S \setminus V(T_v), v)$. We note that $d(S,v) = \min\{d(S \cap V(T_v), v), d(S \setminus V(T_v), v)\}$. After this preprocessing step, it is safe to discard the vertices of S in the subsequent steps.

The following lemma is established in [70]. We provide a different proof here for completeness.

Lemma 5.1.1 *[70] Given a point y in $A(T)$ and a nonnegative value c, the tree-shaped facility K of shortest length with $y \in A(K)$ and $F(K \cup S, V(T)) \leq c$ can be computed in linear time.*

Proof Consider y as the root of T. For each $v \in V$, define $d_v = c/w(v)$. If $d(S,v) \leq d_v$, reset $d_v = \infty$. We define the maximal service distance d_{T_v} of T_v as follows. If $F(\{v\} \cup S, V(T_v)) > c$ then $d_{T_v} = 0$, since, in this case, the facility K to be computed must contain v. If $F(\{v\} \cup S, V(T_v)) \leq c$, $d_{T_v} = \min_{u \in V(T_v)} (d_u - d(v,u))$.

Starting from the leaves of T, and proceeding recursively towards the root y, we do the following for each vertex v. Let e_v be the edge linking v with its parent vertex. If v is a leaf vertex, then $d_{T_v} = d_v$. Suppose v is an internal vertex. Let $u_1, \ldots, u_k$ be its children. If T_v contains some marked point (described below), then $d_{T_v} = 0$. Otherwise, $d_{T_v} = \min\{d_v, d_{T_{u_1}} - l(e_{u_1}), \ldots, d_{T_{u_k}} - l(e_{u_k})\}$, and, if $l(e_v) > d_{T_v}$, then label the point on e_v with the distance d_{T_v} from v as marked. Note that, in discrete problems, v is marked instead. All the marked points must lie on the facility K, if it exists.

The maximal service distance of any rooted subtree is computable in linear time. The spanning subtree of y and the marked points is the new facility K, with shortest length such that $F(K \cup S, V(T)) \leq c$. The whole process takes linear time. This completes the proof of Lemma 5.1.1. $\square$

An algorithm for feasibility decision problems

To solve the path/tree-shaped center problems with the parametric-pruning technique, we need an algorithm to solve the following feasibility decision problem: given a nonnegative real number c, does there exist a path/tree facility K of length not exceeding L_c in $A(T)$ such that $F(K \cup S, V(T)) \leq c$? The number c is *feasible* if $F(K^* \cup S, V(T)) \leq c$ (i.e., $c \geq c^*$), and is *infeasible* otherwise (i.e., $c < c^*$). Here c^* is the service cost of an optimal solution for the path-shaped (resp. tree-shaped) center problem.

A linear time algorithm is already presented in [70]. For completeness, we briefly restate such an algorithm based on the use of Lemma 5.1.1 above. We first select a leaf vertex as the root of T. For each $v \in V(T)$, define $d_v = c/w(v)$. If $d(S,v) \le d_v$, reset $d_v = \infty$. A procedure similar to the one in Lemma 5.1.1 is used, but it is terminated when a point y (or a vertex in discrete problems) is marked. We then compute the facility K with shortest length such that $y \in A(K)$ and $F(K \cup S, V(T)) \le c$ (Lemma 5.1.1). If the facility K is valid, i.e., K is a path/tree with length at most L_c, then the service cost c is feasible, otherwise, c is infeasible.

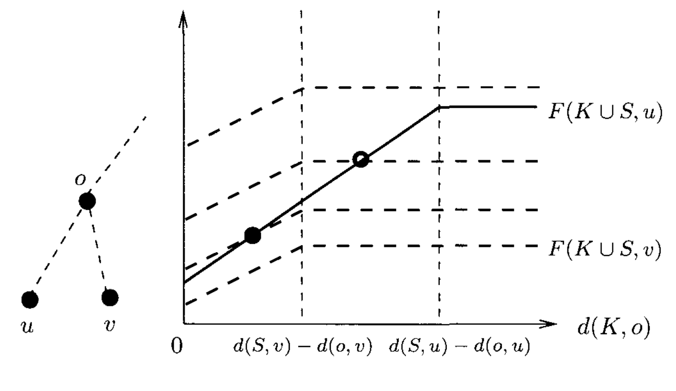
The correctness of this process is shown as follows. The possible error happens when c is feasible, and when for any valid facility K containing y, $F(K \cup S, V(T)) > c$. Let T_y be the subtree rooted at y. It is easy to see that $F(\{y\} \cup S, V(T_y)) = c$ (or $F(\{y\} \cup S, V(T_y)) \le c$ in discrete problems). So the facility, which can serve all the clients within service cost c, must lie in T_y entirely (which is also true for the discrete case). Let K' be such a facility. We can see that, $F(K' \cup S, V(T) \backslash V(T_y)) \le c \Rightarrow F(\{y\} \cup S, V(T) \backslash V(T_y)) \le c \Rightarrow F(K \cup S, V(T)) \le c$, which contradicts that $F(K \cup S, V(T)) > c$.

Number of switch service costs for a pair of vertices

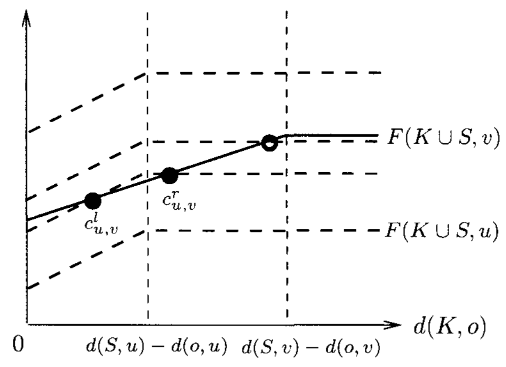
Suppose that a facility K serves a pair of vertices (u,v) through o (see Figure 5.1). In the unconditional model, it is trivial to see that there exists at most one switch service cost of the pair (u,v). However, in the conditional model, each service cost function $F(K \cup S, u)$ ($F(K \cup S, v)$) is a continuous, concave piecewise linear function of distance $d(K,o)$ with at most one break point, therefore, there might be more than one switch service cost. In the following, we show that, in the conditional model, at most two switch service costs exist for a given pair of vertices (u,v). If $d(o,v) \ge d(S,v)$, then v will always be served by some existing facility. Without loss of generality, assume that $d(o,u) < d(S,u), d(o,v) < d(S,v)$, and $w(u) \ge w(v)$. Figure 5.1 shows the service cost functions $F(K \cup S, u)$ and $F(K \cup S, v)$ with the change of $d(K,o)$.

- $d(S,u) - d(o,u) \ge d(S,v) - d(o,v)$: see Figure 5.1(a). We fix the function $F(K \cup S, u)$ and use the dashed lines to represent all possibilities of the function $F(K \cup S, v)$. There is at most one switch service cost between the service cost functions $F(K \cup S, u), F(K \cup S, v)$.

- $d(S,u) - d(o,u) < d(S,v) - d(o,v)$: see Figure 5.1(b). We fix the function $F(K \cup S, v)$ and use the dashed lines to represent all possibilities of the function $F(K \cup S, u)$. In this case, it is possible to have two switch service costs, but at most two.



(a) $d(S,u) - d(o,u) \geq d(S,v) - d(o,v)$



(b) $d(S,u) - d(o,u) < d(S,v) - d(o,v)$

Figure 5.1: The number of switch service costs for (u, v) in the conditional model

When there are two switch service costs for (u, v), we call the switch service cost with the smaller value the *left switch service cost* and call the other one the *right switch service cost.*

In fact, given any pair of real continuous piecewise linear functions defined on a common domain on the real line, if the number of pieces is constant, the pair has a constant number of isolated intersection points. In particular, in our case, considering the service cost functions, each of a pair of vertices has a constant number of dominating regions. Zemel [76] provided a prune-and-search method for the case when there are a constant number of dominating regions for each vertex. Here we describe an algorithm to locate non-dominating vertices in our simpler case.

Compute non-dominating vertices

For those pairs having only one switch service cost, we can locate the non-dominating vertices easily by checking the feasibility of the median switch service cost. In this way, half of such pairs in which one vertex in a pair is identified as a non-dominating vertex. Those non-dominated vertices can now be disregarded. Let $(u_1, v_1), (u_2, v_2), \ldots, (u_k, v_k)$ be the pairs of vertices with two switch service costs, where $w(u_i) \geq w(v_i), 1 \leq i \leq k$. Let $c^l_{u_i,v_i}$ (resp. $c^r_{u_i,v_i}$) be the left (resp. right) switch service costs of $(u_i, v_i), i = 1, \ldots, k$. Select one value c^l (resp. c^r) such that one third of left (resp. right) switch service costs $c^l_{u_i,v_i} > c^l$ (resp. $c^r_{u_i,v_i} \leq c^r$), and the remaining ones are no more than (resp. larger than) it. We call c^l (resp. c^r) the *left switch value* (resp. *right switch value*). After solving the feasibility decision problems with parameters c^l and c^r, we can determine at least $\lfloor \frac{k}{3} \rfloor$ non-dominating vertices for an optimal facility, as shown in the following cases:

- $c^* \leq c^l$. For any pair (u_i, v_i) with $c^l_{u_i,v_i} \geq c^l$, u_i is a non-dominating vertex (dominated by v_i) of an optimal facility.
- $c^* > c^r$. For any pair (u_i, v_i) with $c^r_{u_i,v_i} \leq c^r$, u_i is a non-dominating vertex (dominated by v_i) of an optimal facility.
- $c^l < c^* \leq c^r$ (if possible). There are at least one third of pairs (u_i, v_i) such that $c^l_{u_i,v_i} \leq c^l < c^* \leq c^r \leq c^r_{u_i,v_i}$, since at most one third of such pairs (u_i, v_i) satisfy $c^l_{u_i,v_i} > c^l$ and at most one third of such pairs (u_i, v_i) satisfy $c^r_{u_i,v_i} < c^r$. In each of such pairs (u_i, v_i), v_i is a non-dominating vertex of an optimal facility.

5.2 The weighted path-shaped center problem

In this section we apply the ideas introduced in Section 5.1 to design a linear-time algorithm that solves the path-shaped center location problem in a tree.

Lemma 5.2.1 (Path Lemma) *Given a point q in T, we can find in linear time either the optimal service cost c^*, one subtree network anchored to q containing an optimal path facility, or two subtree networks anchored to q containing an optimal path facility that contains q.*

Proof Let $T_1, \ldots, T_m$ be the subtree networks anchored to q such that $F(\{q\} \cup S, V(T_1)) \geq F(\{q\} \cup S, V(T_2))$ and $F(\{q\} \cup S, V(T_2)) \geq F(\{q\} \cup S, V(T_i)), i = 3, \ldots, m$. Let Z be the set of dominating vertices for the facility set $\{q\} \cup S$.

- If some dominating vertices are in $T \setminus \{T_1 \cup T_2\}$, then $c^* = F(\{q\} \cup S, V(T_1))$ and q is an optimal path facility. In this case, both T_1 and T_2 contain dominating vertices.

- All the dominating vertices are distributed in T_1 and T_2 only. That is, $F(\{q\} \cup S, V(T_1)) = F(\{q\} \cup S, V(T_2)) > F(\{q\} \cup S, V(T_i)), i = 3, \ldots, m$. In this case, an optimal facility lies in $T_1 \cup T_2$, and q lies on it.

- T_1 contains all the dominating vertices, i.e., $Z \subseteq V(T_1)$. Let $c = F(\{q\} \cup S, V(T_2))$. If c is infeasible, then T_1 contains an optimal path facility; otherwise, $c^* \leq c$. Note that q must be on an optimal path facility if $c^* < c$. By Lemma 5.1.1, we can find whether or not there is a valid path facility containing q with a service cost of no more than c. If there exists such a facility, then an optimal path facility lies in $T_1 \cup T_2$ and q lies on it. If not, $c^* = c$.

$\square$

5.2.1 Pruning the tree

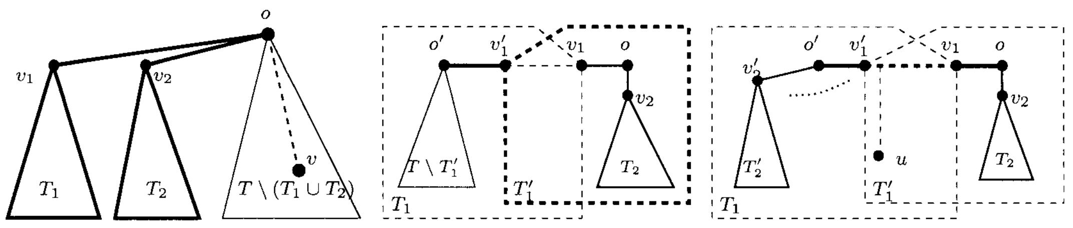
One of the following cases occurs when the Path Lemma is applied to a centroid vertex o of T. Let $T_1, T_2, \ldots, T_m$ be the subtree networks anchored to o, as described in the proof of the Path Lemma.

- *Case 1*: an optimal path facility lies in a subtree, i.e., T_1, anchored to o.

- *Case 2*: an optimal path facility lies in two subtrees anchored to o, i.e., T_1 and T_2, which contains o.

In Case 1, if a vertex $v \in V(T) \setminus V(T_1)$ is served by the new facility, then v is served by the new facility through o. Since o is a centroid vertex, $|V(T \setminus T_1)| \geq n/2$. Our goal now is to prune a fraction of the vertices in $T \setminus T_1$. Randomly pair the vertices in $T \setminus T_1$, and compute the switch service costs for each pair. At least $\lfloor n/4 \times 1/3 \rfloor$ non-dominating vertices in $T \setminus T_1$ can be found and then discarded in linear time, using the method described in Section 5.1.

In Case 2, o is the closest point in an optimal path facility to any vertex in $T \setminus (T_1 \cup T_2)$. We discard all the vertices in $T \setminus (T_1 \cup T_2)$ except one vertex v with $w(v) \times d(\{o\} \cup S, v) =$



(a) Case 2.1 (b) Case 2.2 (c) Case 2.3

Figure 5.2: Case 2 – an optimal path facility lies in T_1 and T_2, which contains o.

$\max_{u \in V(T \setminus (T_1 \cup T_2))} w(u) \cdot d(\{o\} \cup S, u)$. If $|V(T) \setminus V(T_1 \cup T_2)| \geq n/3$ (*Case 2.1*, see Figure 5.2(a)), then at least $n/3 - 1$ vertices in T have been removed. Therefore, assume that $|V(T_1) \cup V(T_2)| > 2n/3$ and $|V(T_1)| > n/3$. Let o' be a centroid vertex of T_1. We get the following cases after applying the Path Lemma to o':

- *Case 2.2*: the path facility lies in one subtree network T_1', anchored to o'. It is easy to see that T_1' contains o, since we are considering the case when o lies on an optimal facility. Therefore, the path facility serves the vertices in $T \setminus T_1'$ through o'. Since $|V(T_1 \setminus T_1')| > n/6$, we can discard at least $\lfloor n/12 \times 1/3 \rfloor$ vertices in $T_1 \setminus T_1'$, using the same idea described for Case 1.

- *Case 2.3*: the path facility lies in two subtrees, T_1' and T_2', anchored to o'. Assume that T_1' contains o. We observe that an optimal facility contains $\pi(o, o')$, the path from o to o'. It implies that, in an optimal solution, the closest point of the new facility to any vertex in $V(T_1) \setminus V(T_2')$ is determined. We are also able to contract $\pi(o, o')$ to o and update L_c correspondingly. Therefore, all the vertices in $T_1 \setminus T_2'$ can be discarded, except a vertex u with the maximum service cost. Since $|V(T_1 \setminus T_2')| \geq n/6$, at least $n/6 - 1$ vertices are removed from T in this case.

Denote by T' the new tree thus computed. The size of T' is at most $\lceil 35n/36 \rceil$ and the process is repeated until the size of the new tree is small. The process terminates within $O(\log n)$ iterations. Since each iteration takes linear time, linear in the size of the current underlying tree, the total cost is linear in n.

It is not hard to see that the algorithm works for the discrete case as well. Summing up, we have the following theorem.

Theorem 5.2.2 *The conditional discrete/continuous path-shaped center problem with length constraint in tree networks can be solved in linear time.*

5.3 The weighted tree-shaped center problem

In this section we present a linear-time algorithm to locate a tree-shaped center in a weighted tree. The following lemma shows a property for the tree-shaped facility location problem, which is similar in spirit to the Path Lemma. Its proof is also very similar.

Lemma 5.3.1 (Tree Lemma) *Given a vertex u in T, we can find, in linear time, either the optimal service cost c^*, that one subtree anchored to u contains an optimal facility, or that u lies in an optimal facility.*

Proof Let $T_1, \cdots, T_m$, be the subtree networks anchored to u such that $F(\{u\} \cup S, V(T_1)) \geq F(\{u\} \cup S, V(T_i)), i = 2, \ldots, m$. Let Z be the set of dominating vertices for the facility set $\{u\} \cup S$.

- If some dominating vertices are in $T \setminus T_1$, then u lies in an optimal facility.
- Otherwise, T_1 contains all the dominating vertices, i.e., $Z \subseteq V(T_1)$. Let $c = F(\{u\} \cup S, V(T_2))$. If c is infeasible, then T_1 contains an optimal tree-shaped facility; otherwise, $c^* \leq c$. Note that u must be on an optimal facility if $c^* < c$. By Lemma 5.1.1, we can find whether or not there is a valid tree-shaped facility containing u with a service cost of no more than c. If there exists such a facility, then u lies in an optimal facility. If not, $c^* = c$.

□

5.3.1 Pruning the tree

Two cases arise after the application of the Tree Lemma to a centroid vertex o of T.

Case 1: here we consider the situation when one subtree network anchored to o contains an optimal facility. In this case we can discard $\lfloor n/12 \rfloor$ vertices in linear time by a procedure similar to the one described for Case 1 in the path-shaped center problem.

Case 2: here we consider the situation when o is contained in an optimal facility. Let o be the root of T. For a leaf vertex u of T, let $p(u)$ denote the unique vertex adjacent to u in T, and let e_u be the edge connecting u with $p(u)$.

In Case 2, we focus on pruning the vertices in T that are of no more than two degrees. The main reason is that it is easy to remove non-dominating vertices of no more than two degrees from T. To prune a leaf vertex u, we simply remove it and the edge e_u incident on u; and, to prune a vertex u of degree two, we remove it and merge the two edges e_1, e_2 incident to u into one new edge whose length is equal to $l(e_1)+l(e_2)$. Also, we note that the number of vertices whose degrees are at most two is at least $n/2$. To verify this result, let n_1, n_2 and n_3^+, denote the number of vertices whose degrees are equal to one, equal to two, and greater than two, respectively. Then, counting the degrees, we have $n_1+n_2+n_3^+ = n$, and $n_1+2n_2+3n_3^+ \leq 2(n-1)$. Therefore, $n_3^+ \leq n_1 - 2$ and $n_1+n_2 \geq n/2+1$. We will next show how to prune a fixed proportion of the vertices whose degrees are bounded by two.

Prune the tree in Case 2

In Section 5.1, we describe a linear-time algorithm to test the feasibility of a given nonnegative real number c. If c is feasible, then $c \geq c^*$, and otherwise $c < c^*$. For the continuous tree-shaped center problem, we can find out whether $c > c^*$, or $c = c^*$, or $c < c^*$ in linear time in Case 2.

Lemma 5.3.2 *Given a point $x \in A(T)$ and a nonnegative value c, if there exists an optimal continuous tree-shaped center containing x, then we can find out whether $c > c^*$, or $c = c^*$, or $c < c^*$ in linear time.*

Proof We can identify whether $c \geq c^*$ or $c < c^*$, after a linear-time feasibility test of c. Suppose that $c \geq c^*$.

For any vertex $v \in V(T)$ with $w(v) \cdot d(S,v) = c$, we update $d(S,v) = \infty$. By applying Lemma 5.1.1 with the point x and the nonnegative value c, the facility K of shortest length with $x \in A(K)$, and $F(K \cup S, V(T)) \leq c$ can be computed in linear time. If the length of K thus computed is $\geq L_c$, then $c = c^*$, and otherwise $c > c^*$. The reason is that now all demand vertices v with $w(v) \cdot d(S,v) \geq c$ are served by the new facility K. Therefore, the optimal service cost c^* must be smaller than c if the length of K is shorter than the length constraint L_c, and $c^* \geq c$ otherwise. □

For each leaf vertex u of T, we compute $c_u = w(u) \cdot d(\{p(u)\} \cup S, u)$. We observe that, for any leaf vertex u,

- if $c^* < c_u$, then u is a dominating leaf vertex. A part of the edge e_u must lie in an optimal facility. In this case, $p(u)$ is contained in the new facility and u is served by the new facility. Therefore, we can shrink the path $\pi(o, p(u))$ to one point, o, and update L_c correspondingly. Lemma 5.3.3 provides a linear-time process to prune such dominating leaf vertices; and
- if $c^* > c_u$, the new facility lies outside the edge e_u in an optimal solution. In this case, if $p(u)$ is contained in the new facility in an optimal solution, then u is a non-dominating vertex and therefore can be discarded.

Given two dominating leaf vertices u and v, let T' be the tree after deleting u, e_u, v, e_v from T, adding a new leaf vertex v' with $w(v') = w(u) \cdot w(v)/(w(u) + w(v))$, and linking v' to o with $l(e_{v'}) = l(e_u) + l(e_v)$. We have the following lemma, which is used to prune dominating leaf vertices. But first we need to shorten an edge e_u if u is a dominating leaf vertex and $d(S, u) < l(e_u)$, i.e., we let $L_c = L_c - l(e_u) + d(S, u)$ and $l(e_u) = d(S, u)$. Now, for any dominating leaf vertex u, the optimal service cost is less than $w(u) \cdot l(e_u)$ and $p(u) = o$.

Lemma 5.3.3 *The optimal service cost in T' is equal to the optimal service cost c^* in T.*

Proof First, in T, $p(u) = p(v) = o$ and u, v are served by the optimal facility K^* (not served by some existing facility), since $c^* < \min\{w(u) \cdot l(e_u), w(v) \cdot l(e_v)\}$, $d(S, u) \geq l(e_u)$, and $d(S, v) \geq l(e_v)$. Let d_1, d_2 be the distance from u, v to the optimal facility K^* respectively. Clearly, $w(u) \cdot d_1 = w(v) \cdot d_2 = c^*$. The total length of the parts of K^* on e_u and e_v in T is equal to $l(e_u) + l(e_v) - d_1 - d_2$.

In T', it is not hard to see that v' is a dominating leaf vertex. Since $(d_1 + d_2) \cdot w(u) \cdot w(v)/(w(u) + w(v)) = w(u) \cdot d_1 = w(v) \cdot d_2$, the length of the part of the new facility on the edge $e_{v'}$ is $l(e_u) + l(e_v) - d_1 - d_2$ with the same service cost c^*, which is equal to the total length of the parts of K^* on e_u and e_v in T. Therefore, the optimal service cost in T' is equal to c^*. □

The above discussion shows that we are able to prune dominating leaf vertices u (i.e., $c^* < c_u$) and non-dominating leaf vertices v (i.e., $c^* > c_v$, and $p(v)$ is contained in the new facility).

Next, we introduce two types of pairs of vertices with degree one or two, and show how to prune non-dominating vertices among them.

A pair of vertices (v_1, v_2) is called a *Type-I pair* if the new facility lies outside subtree T_u in an optimal solution where u is the least common ancestor of v_1 and v_2. We know that, if the new facility lies outside subtree T_u in an optimal solution, then the topology information of T_u is not important and, therefore, for each vertex in T_u, we only need to keep its distance information to u (like a leaf vertex). In Section 5.1, we show that there are at most two switch service costs between v_1 and v_2, and among k such Type-I pairs, at least $\lfloor k/3 \rfloor$ non-dominating vertices can be identified after solving at most two feasibility decision problems.

A pair of vertices (v_1, v_2) is called a *Type-II pair* if the degrees of v_1 and v_2 are no more than two, and v_1, v_2 have an 'ancestor-descendant' relation, i.e., v_1 is on the path $\pi(v_2, o)$ or v_2 is on $\pi(v_1, o)$. We claim that for each such pair (v_1, v_2), there are at most two switch service costs between v_1 and v_2, regardless of the location of the optimal facility. Suppose that v_2 lies on the path $\pi(v_1, o)$. Note that the new facility contains o. If the new facility in an optimal solution contains v_2, then v_2 is a non-dominating vertex. Otherwise, the new facility lies outside the path $\pi(v_1, v_2)$, in which case we already know there exist at most two switch service costs. This implies that among k such Type-II pairs, at least $\lfloor k/3 \rfloor$ non-dominating vertices can be identified after solving at most two feasibility decision problems.

The algorithm for the weighted continuous tree-shaped center problem in T, is descried in Algorithm 1.

In the following, we first describe a simple algorithm to determine disjoint Type-II pairs of vertices, and show that the output size depends on the number of leaf vertices in the tree. We then discuss Lines $10 - 15$ in Algorithm 1 to prune the dominating leaf vertices or non-dominating vertices for the case when there are at least $n/5$ leaf vertices in T.

Generate Type-II pairs of vertices: We first remove all the vertices of T with degrees greater than two, along with the edges incident on them. Each component of the remaining forest is a sub-path of some path connecting a leaf vertex to the root o. We arbitrarily group the vertices of each component into disjoint pairs (leaving one vertex if the number of vertices is odd).

Lemma 5.3.4 *Let n_1 be the number of leaf vertices in T. Then the number of vertices not in any selected pair cannot be more than $3n_1$.*

Algorithm 1 ContinuousTree-Center($T = (V(T), E(T)), S, L_c$)

Input: an undirected tree network T with vertex set $V(T)$ ($|V(T)| = n$) and edge set $E(T)$, a set S of existing facilities, and a nonnegative real number L_c.

Output: the optimal service cost c^*.

```
begin
1:  repeat
2:    Compute the centroid $o$ of $T$. Apply the Tree Lemma on $o$ and consider two cases:
3:    if a subtree $T'$ anchored to $o$ contains an optimal facility then
4:      prune non-dominating vertices in the subtree $T \setminus T'$ and update $T$ accordingly.
5:    else
6:      (In this case, $o$ lies in an optimal facility.)
7:      if the number of leaf vertices is no more than $n/5$ then
8:        we obtain at least $n/5$ Type-II pairs of vertices (Lemma 5.3.4). Prune non-
          dominating vertices among them and update $T$ accordingly.
9:      else
10:       Compute $c_u$ for each leaf vertex $u$. Let $c$ be the value such that at least one third
          of these values are $\geq c$, and such that at least two thirds of these values are $\leq c$.
11:       if $c^* < c$ then
12:         prune dominating leaf vertices (Lemma 5.3.3).
13:       else
14:         Algorithm 2.
15:       end if
16:     end if
17:   end if
18: until the size of $T$ is no more than 270.
19: Solve the problem on $T$ with a constant size.
end
```

Proof There are at most n_1 vertices of degree more than two in T. Then there are at most $2n_1$ disjoint paths after the removal of all the vertices of degree more than two. Note that there is at most one vertex in each path that is not in any selected pair. Therefore, the number of vertices not in any selected pair cannot be more than $3n_1$. □

In the case when the number of leaf vertices is no more than $n/5$, we can obtain at least $n/5$ (Type-II) pairs of vertices (Lemma 5.3.4). As we know, for each Type-II pair, there are at most two switch service costs. Then the parametric-pruning method proposed in Section 5.1 is applied to prune non-dominating vertices in these Type-II pairs. In this way, at least $\lfloor 1/3 \times n/5 \rfloor$ vertices can be discarded after solving the corresponding feasibility decision problems (at most two).

We now consider the case when the number of leaf vertices in T is more than $n/5$. Let c be the value described in Line 10 in Algorithm 1. According to Lemma 5.3.2, we have three cases.

In the case when $c = c^*$, output c and then terminate the procedure.

In the case when $c^* < c$, for each leaf vertex u with $c_u \geq c$, u is a dominating vertex served by the new facility, and $p(u)$ lies in the new facility. Hence, there are at least $1/3 \times n/5$ such dominating vertices. Lemma 5.3.3 provides the process to prune these dominating leaf vertices.

In the case when $c^* > c$, see Algorithm 2 described as follows.

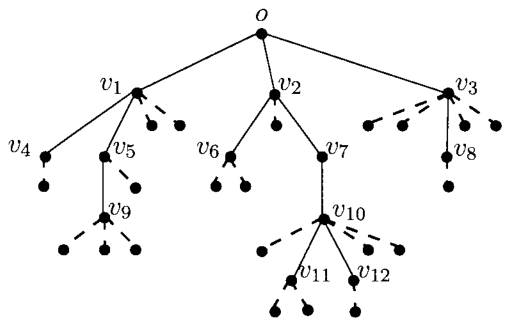
Algorithm 2: pruning when n_1 is greater than $n/5$ and $c^* > c$. For each leaf vertex u with $c_u \leq c$, the new facility in an optimal solution lies outside e_u. Let X be the set of such leaf vertices and let X' be the set of vertices that are parents of leaf vertices in X. Clearly, $|X| \geq 2/3 \times n/5$ (the definition of c). For each $v \in V(T)$, we denote by $n_l(v)$ the number of leaf vertices in X that are children of v, and by $N_l(v)$ the number of leaf vertices in X that are descendants of v. Obviously, $N_l(o) = |X|$. Let T' be the smallest subtree of T that contains all the vertices in $X' \cup \{o\}$.

We next show how to find a set of vertices V' of T' satisfying the following three conditions.

- Condition 1: there is no pair of vertices in V' with 'ancestor-descendant' relationship.
- Condition 2: $\sum_{v \in V'} N_l(v) \geq 1/2 \times N_l(o)$ (Lemma 5.3.5).

- Condition 3: for any subset V'' of V', let U'' be the set of vertices that are proper ancestors of vertices in V''. Then $\sum_{v \in U'' \cup V''} n_l(v) \geq 1/2 \times \sum_{v \in V''} N_l(v)$ (Lemma 5.3.6).

To generate the above sets, we use a depth-first search on T'. We denote by u the current vertex and define a term $g(v)$ for each vertex in T'. Initially, $g(o) = 0, u = o$, and $V' = \emptyset$. Let $v_1, \cdots, v_k$ be the children of u. If $g(u) + n_l(u) \geq N_l(u) - n_l(u)$, then insert the vertex u into V' and update u to be the next vertex after visiting all vertices in T_u in a depth-first order. Otherwise, for each $i, 1 \leq i \leq k$, we compute $g(v_i) = N_l(v_i) \cdot (g(u) + n_l(u)) / \sum_j N_l(v_j)$ (Note that $\sum_j N_l(v_j) = N_l(u) - n_l(u)$). Update u to be v_1. The process is repeated on a new vertex u.



$n_l(o) = 0, N_l(o) = 21;$
$n_l(v_1) = 2, N_l(v_1) = 7;$
$n_l(v_2) = 1, N_l(v_2) = 9;$
$n_l(v_3) = 4, N_l(v_3) = 5;$
$n_l(v_4) = 1, N_l(v_4) = 1;$
$n_l(v_5) = 1, N_l(v_5) = 4;$
$n_l(v_6) = 2, N_l(v_6) = 2;$
$n_l(v_7) = 0, N_l(v_7) = 6;$
$n_l(v_9) = 3, N_l(v_9) = 3;$
$n_l(v_{10}) = 3, N_l(v_{10}) = 6.$

Figure 5.3: An example for Algorithm 2.

For example, in Figure 5.3, the leaf vertices incident to dashed edges are in the set X, and T' is the tree without dashed edges (i.e., the vertex set of T' is $\{o, v_1, v_2, \cdots, v_{12}\}$). The $n_l(\cdot)$ and $N_l(\cdot)$ values of the non-leaf vertices are listed at the right side of Figure 5.3. Initially, $g(o) = 0$. Since $g(o) + n_l(o) < N_l(o) - n_l(o)$, we distribute $g(o) + n_l(o) = 0$ among the children of o, based on their $N_l(\cdot)$ values, i.e., $g(v_1) = g(v_2) = g(v_3) = 0$.

For the vertex v_1, since $g(v_1) + n_l(v_1) = 2 < 5 = N_l(v_1) - n_l(v_1)$, we distribute $g(v_1) + n_l(v_1) = 2$ among the children of v_1, i.e., $g(v_4) = 2/5$ and $g(v_5) = 8/5$. Then the algorithm visits v_4. Since $g(v_4) + n_l(v_4) = 7/5 \geq 0 = N_l(v_4) - n_l(v_4)$, the algorithm outputs v_4 and jumps to vertex v_5.

The final output list of the algorithm on this example will be $V' = \{v_4, v_9, v_6, v_{10}, v_3\}$. The corresponding $g(\cdot)$ values are $g(v_4) = 2/5$, $g(v_9) = 13/5$, $g(v_6) = 1/4$, $g(v_{10}) = 3/4$, $g(v_3) = 0$.

It is not hard to see that this procedure takes linear time. Let U' be the set of vertices in T' that are proper ancestors of vertices in V'. We can see that the sets V' and U' satisfy the three conditions described above.

Lemma 5.3.5 *Let V' and U' be the two sets defined above. Then $\sum_{v \in V'} N_l(v) \geq 1/2 \times N_l(o)$.*

Proof First, $\sum_{v \in U'} n_l(v) = \sum_{v \in V'} g(v)$. In the above example, $V' = \{v_4, v_9, v_6, v_{10}, v_3\}$, $U' = \{o, v_1, v_2, v_5, v_7\}$, $\sum_{v \in V'} g(v) = 4$, and $\sum_{v \in U'} n_l(v) = 4$. Second, for each vertex $v \in V'$, $g(v) < N_l(v)$. Since all the vertices in T' are either in U' or descendants of vertices in V', $N_l(o) = \sum_{v \in V'} N_l(v) + \sum_{v \in U'} n_l(v)$. Therefore, $\sum_{v \in V'} N_l(v) \geq 1/2 \times N_l(o)$. □

Lemma 5.3.6 *Let V' be the set defined above. For any subset V'' of V', let U'' be the set of vertices in T' that are ancestors of vertices in V''. Then $\sum_{v \in U'' \cup V''} n_l(v) \geq 1/2 \times \sum_{v \in V''} N_l(v)$.*

Proof Let $V_{U''}$ be the set of vertices that are children of vertices in U'' but not in U''. It is not hard to see that $V'' \subseteq V_{U''}$ (Condition 1). From the computation of $g(v)$, we know that $g(v)$ is contributed from the $n_l(\cdot)$ values of the ancestors of v, and the $n_l(\cdot)$ value of a vertex is distributed among its children. Thus, $\sum_{v \in U''} n_l(v) = \sum_{v \in V_{U''}} g(v)$ (with the same reasoning, we obtain $\sum_{v \in U'} n_l(v) = \sum_{v \in V'} g(v)$ in Lemma 5.3.5), which implies that $\sum_{v \in U''} n_l(v) \geq \sum_{v \in V''} g(v)$.

Note that when a vertex v is inserted in V', $g(v) + n_l(v) \geq N_l(v) - n_l(v)$. Therefore,

$$\begin{aligned} \sum_{v \in U'' \cup V''} n_l(v) &\geq \sum_{v \in V''} (g(v) + n_l(v)) \\ &\geq \sum_{v \in V''} \max\{n_l(v), N_l(v) - n_l(v)\} \\ &\geq 1/2 \times \sum_{v \in V''} N_l(v). \end{aligned}$$

□

In the remaining part of this section we will discuss the procedure of the algorithm after we obtain the set V'.

For each $v \in V'$, we compute $F(\{v\} \cup S, V(T_v))$, which can be done in linear time (Condition 1). Let c' be the weighted median of these values, where the weight of each value $F(\{v\} \cup S, V(T_v))$ is $N_l(v)$. According to Lemma 5.3.2, we have three cases. It is trivial to deal with the case when $c' = c^*$.

When $c^* < c'$, for any $v \in V'$ with $F(\{v\} \cup S, V(T_v)) \geq c'$, v is contained in the new facility in an optimal solution, and therefore, we can shrink the path $\pi(v, o)$ and discard all leaf vertices in X that are adjacent to vertices on $\pi(v, o)$. These leaf vertices u in X can be discarded, since $p(u)$ is contained in the new facility in an optimal solution (i.e., u is a non-dominating leaf vertex). The number of leaf vertices thus discarded is at least

$$\begin{aligned} & 1/2 \sum_{v \in V' : F(\{v\} \cup S, T_v) \geq c'} N_l(v) \text{ (Condition 3)} \\ \geq \ & 1/4 \times \sum_{v \in V'} N_l(v) \text{ (the definition of } c') \\ \geq \ & 1/12 \times n/5 \text{ (Condition 2).} \end{aligned}$$

In the case when $c^* > c'$, let V''' be the set of vertices v in V' with $F(\{v\} \cup S, V(T_v)) \leq c'$. For any $v \in V'''$, v is not contained in the new facility in an optimal solution. For each vertex $v \in V'''$, we arbitrarily pair the vertices in $V(T_v) \setminus \{v\}$ (leaving one vertex if there is an odd number of vertices in $V(T_v) \setminus \{v\}$), and if $|T_v| = 2$ then the two vertices in T_v have the Type-II relation. Note that $\sum_{v \in V'''} N_l(v) \geq 1/4 \times |X|$. For each $v \in V'''$, at most one leaf vertex in T_v is not in any pair, and if there is one leaf vertex in T_v that is not in any pair, then $|V(T_v) \setminus \{v\}| \geq 3$. Therefore, there are at least $1/12 \times |X| \geq n/90$ disjoint Type-I pairs. Observe that each Type-I pair has at most two switch service costs. Therefore, the algorithm described in Section 5.1 can be used to prune at least $\lfloor n/270 \rfloor$ vertices in $|X|$.

From the above discussion, it takes linear time to prune at least a constant fraction of vertices of the current tree at each iteration, linear in the size of the current tree. Therefore, we establish Theorem 5.3.7.

Theorem 5.3.7 *The conditional weighted continuous tree-shaped center problem with length constraint in trees can be solved in linear time.*

Adapting the algorithm for the discrete case is not difficult. For example, we can get Lemma 5.3.8, which is similar to Lemma 5.3.2, in the discrete case.

Lemma 5.3.8 *Given a vertex $u \in V(T)$ and a nonnegative value c, if there exists an optimal discrete tree-shaped center containing u, then we can find out whether $c > c^*$, or $c = c^*$, or $c < c^*$ in linear time.*

Proof The proof of this lemma is very similar to the proof of Lemma 5.1.1.

It is sufficient to show a procedure to construct a discrete tree-shaped facility K of shortest length that contains u and $F(K \cup S, V(T)) < c$. Consider u as the root of T. For each $v \in V$, define $d_v = c/w(v)$. If $d(S,v) \leq d_v$, reset $d_v = \infty$. For each rooted subtree T_v, we define a term d_{T_v}, whose initial value is ∞.

Starting from the leaves of T, and proceeding recursively towards the root, we do the following for each vertex v. If T_v contains some marked vertex (described below) then $d_{T_v} = 0$. Otherwise, let $v_1, \ldots, v_k$ be the children of v (if v is a leaf vertex, then $k = 0$). $d_{T_v} = \min\{d_v, d_{T_{u_1}} - l(e_{u_1}), \ldots, d_{T_{u_k}} - l(e_{u_k})\}$, and if $l(e_v) \geq d_{T_v}$ then label the vertex v as marked. All the marked vertices must lie in the facility K, if $F(K \cup S, V(T)) < c$.

The spanning tree of u and the marked vertices is the new facility K with shortest length such that $F(K \cup S, V(T)) < c$. The whole process takes linear time. □

For a leaf vertex v with $c^* < c_v$ in Case 2, v is contained in an optimal facility in the discrete case. It is easy to adapt all the other components of the algorithm for the discrete case. Therefore, we have the following theorem.

Theorem 5.3.9 *The conditional weighted discrete tree-shaped center problem with length constraint in trees can be solved in linear time.*

5.4 Summary

In this chapter, we propose optimal algorithms for the extensive facility location problems in tree networks, where the new facility (center) is either path-shaped or tree-shaped. The main technique is to 'prune' the non-dominating vertices and to 'shrink' the facility if some path or subtree is known to be a part of an optimal facility. These results improve the recent $O(n \log n)$ results of Tamir et al. [70].

For the case where the service cost $f_i(x)$ of a client v_i is a nondecreasing piecewise linear function of the service distance x to the facility with a fixed number of breakpoints (in the 'conditional' case $f_i(x)$ has only one breakpoint), all the ideas presented in this chapter can be extended to achieve an optimal algorithm for locating the facility in the new case. Actually, our method works even when the piecewise linearity assumption is relaxed to piecewise polynomiality (e.g. quadratic, or cubic) of fixed degree.

Chapter 6

Continuous tree p-edge-partition problems

In this chapter we consider continuous tree edge-partition problems (CEPs, for short) on an edge-weighted tree network $T = (V(T), E(T), l)$. A continuous p-edge-partition of T is to divide it into p subtrees by selecting $p-1$ cut points along the edges of T. The objective is to minimize (maximize) the maximum (minimum) length of the subtrees.

Recently, Lin et. al. [49] proposed $O(n^2)$-time algorithms for the two problems, which improves the $O(n^2 \log(\min\{p, n\}))$ result of Halman and Tamir [37]. The proposed algorithms of Lin et. al. [49] are based on efficiently solving a problem called the *ratio search problem.* In this chapter we consider a more general version of the ratio search problem that is defined as follows.

Definition 6.0.1 (Ratio search problem) *Given a positive integer q, a real number t, a non-increasing function $\mathcal{F} : \mathbf{R} \to \mathbf{R}$, a set of k non-negative real numbers $\Delta = \{b_i, i = 1, \ldots, k\}$, and each number b_i in Δ is associated with a nonnegative integer number $g_i, 1 \leq i \leq k$, compute the largest real number z in $\{b_i/a_i, i = 1, \cdots, k \mid a_i \in [g_i + 1, g_i + q], b_i \in \Delta\}$ such that $\mathcal{F}(z) \geq t$.*

Lin et. al. [49] proposed an algorithm for the ratio search problem with uniform values of $g_i = 0, i = 1, \cdots, k$. We present an approach that solves the ratio search problem with the same time complexity, for non-uniform values of $g_i, i = 1, \cdots, k$. Using our efficient algorithm for the ratio search problem, we are able to solve the max-min CEP problem in

$O(n\log^2 n)$ time, which is a substantial improvement of previous results. For the min-max CEP problem, the proposed algorithm runs in time $O(nh_T \log n)$, where h_T is the height of the underlying tree. When $h_T = o(n/\log n)$, our result for the min-max problem is better.

Organization of the chapter Related works are reviewed in Section 6.1. Sections 6.2 and 6.3 provide the main results of this chapter - algorithms to solve max-min and min-max CEP problems on a weighted tree network. An algorithm for the ratio search problem is presented in Section 6.4. Finally, Section 6.5 gives a brief conclusion.

6.1 Related works

In this section, related works are discussed, including spine tree decomposition [9] (Section 6.1.1) and linear-time feasibility tests for the max-min and min-max CEP problems [37] (Sections 6.1.2 and 6.1.3).

6.1.1 Spine decomposition of T

We consider a rooted tree T whose root vertex r_T is of degree one. We denote by $p(v)$ the parent of v in T. Let T_v be the subtree of T that is rooted at a vertex v and let $C(v)$ denote the set of all immediate children of v. In the following, we introduce a *spine decomposition* of T [9] to control the depth of the recursion in our algorithm for the max-min problem. Let $N_l(v)$ be the number of leaves that are descendants of v in T.

A path $\pi(r_T, v')$ from the root r_T to a leaf v' of T is first identified such that for any two consecutive vertices v_i and v_{i+1} on $\pi(r_T, v')$ ($v_0 = r_T$, $p(v_{i+1}) = v_i$, and $v_m = v'$), the following condition is satisfied: for any child u of v_i other than v_{i+1}, $N_l(u) \leq N_l(v_{i+1})$. That is, the path follows vertices from the root to a leaf such that the next vertex chosen is always the child of the current vertex with the most number of leaf descendants. The path $\pi(r_T, v')$ is called a *spine* and r_T is the root of this spine. We label the spine as the 1^{st}-level spine.

The procedure is then applied recursively on each T'_u where u is a child of vertex v_i and T'_u is composed of T_u and the edge $\overline{uv_i}$, $i = 1, \cdots, m-1$. Note that v_i is the root of T'_u. We label a spine that hangs from another j^{th}-level spine, a $(j+1)^{th}$-level spine. See Figure 6.1 for reference.

We have the following property about this spine decomposition of T.

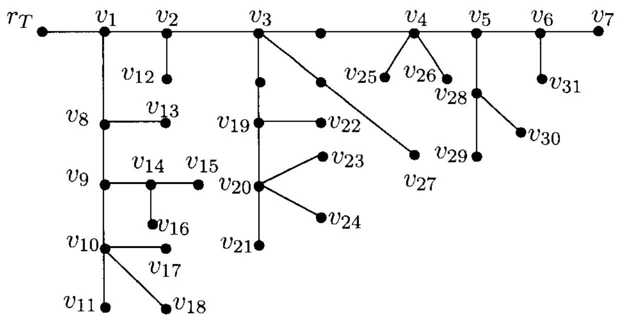


Figure 6.1: Spine tree decomposition. 1st-level spine: $\pi(r_T, v_7)$; 2nd-level spines: $\pi(v_1, v_{11})$, $\pi(v_2, v_{12})$, $\pi(v_3, v_{21})$, $\pi(v_3, v_{27})$, $\pi(v_4, v_{25})$, $\pi(v_4, v_{26})$, $\pi(v_5, v_{29})$, $\pi(v_6, v_{31})$; 3rd-level spines: $\pi(v_8, v_{13})$, $\pi(v_9, v_{15})$, $\pi(v_{10}, v_{17})$, $\pi(v_{10}, v_{18})$, $\pi(v_{19}, v_{22})$, $\pi(v_{20}, v_{23})$, $\pi(v_{20}, v_{24})$, $\pi(v_{28}, v_{30})$; 4th-level spine: $\pi(v_{14}, v_{16})$.

Lemma 6.1.1 *[9] For any vertex $v \in V(T)$, the simple path $\pi(r_T, v)$ goes through at most $O(\log n)$ spines.*

In other words, the maximum level of spines in a spine decomposition of T is $O(\log n)$, denoted by τ.

6.1.2 A linear-time feasibility test for the max-min CEP problem

Let $l^*_{p(1)}$ be the length of the smallest subtree in an optimal solution to the max-min CEP problem. For any positive real number $l \leq l(T)$, define $Z_1(l)$ to be the largest number such that there exists an $Z_1(l)$-edge-partition in which the length of each subtree is at least l. A length l is *feasible* in the max-min model if $Z_1(l) \geq p$, and *infeasible*, otherwise.

Lemma 6.1.2 *In the max-min model, $l^*_{p(1)} \leq l(T)/p$.*

In [37], Halman and Tamir presented an $O(n)$-time algorithm for determining the feasibility of a given length l, as mentioned in Lemma 6.1.3.

Lemma 6.1.3 *[37] Whether a given positive length l is feasible in the max-min CEP model can be determined in $O(n)$ time.*

Algorithm: computing $Z_1(l)$ [37]

We use a bottom-up approach, starting with the leaves of T rooted at r_T. Recall that for each vertex v, $C(v)$ denotes the set of all immediate children of v and T_v denotes the subtree of T rooted at v. A vertex v is called a *cluster vertex* of a rooted tree, if all of its children are leaves of this tree.

The algorithm to compute $Z_1(l)$ [37] is described as follows.

We start with the rooted tree T. Initially we set $Z_1(l) = 0$. In a generic iteration of the algorithm we select a cluster vertex v_i of the (current) tree. Let $\{v_{i(1)}, \cdots, v_{i(t)}\}$ be the set of children of v_i.

Step 1: Trimming a cluster. For each $k = 1, \cdots, t$, define $n_k = \lfloor l(\overline{v_i v_{i(k)}})/l \rfloor$, and add n_k to $Z_1(l)$. Reduce the length of the edge $\overline{v_i v_{i(k)}}$ from $l(\overline{v_i v_{i(k)}})$ to $a_k = l(\overline{v_i v_{i(k)}}) - n_k \times l$. (This accounts for adding n_k cut points on the edge, where the distance between adjacent cut points is exactly l.) If $l(\overline{v_i v_{i(k)}}) = n_k \times l$ then delete the edge $\overline{v_i v_{i(k)}}$ from the current tree. Define $\mu = \sum_{k=1}^{t} a_k$. If $v_i = r_T$ delete all edges $\overline{v_i v_{i(k)}}$ from the current tree, and go to Step 3. If all edges are deleted (i.e., $\mu = 0$) and $v_i \neq r_T$, repeat the process with a cluster of the updated tree.

Step 2: Deleting a cluster. Delete all remaining edges $\overline{v_i v_{i(k)}}$, $k = 1, \cdots, t$, from the current tree. If $v_i = r_T$ go to Step 3. If $v_i \neq r_T$ and $\mu \geq l$, add 1 to $Z_1(l)$ (corresponding to the vertex-cut on v_i and $\overline{v_i p(v_i)}$). Repeat the process with a cluster of the updated tree. If $v_i \neq r_T$ and $\mu < l$, increase the length of the edge $\overline{v_i p(v_i)}$ from $l(\overline{v_i p(v_i)})$ to $l(\overline{v_i p(v_i)}) + \mu$, and repeat the process with a cluster of the updated tree.

Step 3: Termination at the root r_T. If $\mu \geq l$ add 1 to $Z_1(l)$. Stop and return the current value of $Z_1(l)$. If $\mu < l$ remove the cut point which is the closest to r_T amongst all $Z_1(l)$ cut points that have been established. Stop and return the current value of $Z_1(l)$.

Clearly, the algorithm runs in linear time.

6.1.3 A linear-time feasibility test for the min-max CEP problem

Similarly, we define $l^*_{p(2)}$ and $Z_2(l)$ for the min-max model as follows. Let $l^*_{p(2)}$ be the largest length of the subtrees in an optimal solution to the min-max CEP problem. For any positive real number $l \leq l(T)$, define $Z_2(l)$ to be the smallest number such that there exists an $Z_2(l)$-edge-partition in which the length of each subtree is at most l. A length l is *feasible* in the min-max CEP model if $Z_2(l) \geq p$, and *infeasible* otherwise.

Lemma 6.1.4 *In the min-max model, $l^*_{p(2)} \geq l(T)/p$.*

The linear-time algorithm of Halman and Tamir [37] for determining the feasibility of a given length l in the min-max model is presented as follows.

Lemma 6.1.5 *[37] Whether a given positive length l is feasible in the min-max CEP model can be determined in $O(n)$ time.*

Algorithm: computing $Z_2(l)$ [37]

The computation of $Z_2(l)$ is very similar to the computation of $Z_1(l)$. As before, a bottom-up approach is used to compute $Z_2(l)$ [37], starting with the leaves of T, which is described as follows.

Initially we set $Z_2(l) = 0$. We select a cluster vertex v_i of the (current) tree and let $\{v_{i(1)}, \cdots, v_{i(t)}\}$ be the set of children of v_i.

Step 1: Trimming a cluster. For each $k = 1, \cdots, t$, define $n_k = \lfloor l(\overline{v_i v_{i(k)}})/l \rfloor$, and add n_k to $Z_2(l)$. Reduce the length of the edge $\overline{v_i v_{i(k)}}$ from $l(\overline{v_i v_{i(k)}})$ to $a_k = l(\overline{v_i v_{i(k)}}) - n_k \times l$. If $l(\overline{v_i v_{i(k)}}) = n_k \times l$ delete the edge $\overline{v_i v_{i(k)}}$, from the current tree. Define $\mu = \sum_{k=1}^{t} a_k$. If all child edges are deleted and $v_i = r_T$, go to Step 3. If all child edges are deleted and $v_i \neq r_T$, repeat the process with a cluster of the updated tree.

Step 2: Deleting a cluster. If $\mu = l$, delete all remaining edges $\overline{v_i v_{i(k)}}$, $k = 1, \cdots, t$, from the current tree. Add 1 to $Z_2(l)$ (corresponding to the vertex-cut on v_i and $\overline{v_i p(v_i)}$) and repeat the process with a cluster of the updated tree.

If $\mu < l$, delete all remaining edges $\overline{v_i v_{i(k)}}$, $k = 1, \cdots, t$, from the current tree. Increase the length of the edge $\overline{v_i p(v_i)}$ from $l(\overline{v_i p(v_i)})$ to $l(\overline{v_i p(v_i)}) + \mu$, and repeat starting with a cluster of the updated tree.

If $\mu > l$, find q, and the subset $C'(v_i)$ of children of v_i, corresponding to the q smallest non-zero elements in the multiset $\{a_k | k = 1, \ldots, t\}$, such that the sum of these q smallest elements, denoted by μ', is $\leq l$, and the sum of the smallest nonzero $q+1$ elements is $> l$. For each vertex $v_{i(k)} \notin C'(v_i)$, such that $a_k > 0$, delete the edge $\overline{v_i v_{i(k)}}$, and add 1 to $Z_2(l)$. For each vertex $v_{i(k)} \in C'(v_i)$, delete the edge $\overline{v_i v_{i(k)}}$. Increase the length of the edge $\overline{v_i p(v_i)}$ from $l(\overline{v_i p(v_i)})$ to $l(\overline{v_i p(v_i)}) + \mu'$, and repeat the process with a cluster of the updated tree.

Step 3: Termination at the root r_T. If $\mu = 0$, stop and return the current value of $Z_2(l)$.

If $0 < \mu \leq l$, add 1 to $Z_2(l)$. Stop and return the current value of $Z_2(l)$.

If $\mu > l$, find q, and the subset $C'(r_T)$ of children of r_T, corresponding to the q smallest non-zero elements in the multiset $\{a_k | k = 1, \dots, t\}$, such that the sum of these q smallest elements, denoted by μ'', is at most l, and the sum of the smallest nonzero $q+1$ elements is greater than l. Let $t' = |\{k | a_k > 0; k = 1, \dots, t\}|$. Add $t' - q + 1$ to $Z_2(l)$. Stop and return the current value of $Z_2(l)$.

The running time of the above algorithm is linear, since the time to process a vertex is proportional to its number of children. (We can apply the linear-time median-finding algorithm [16] successively to find the term $C'(v_i)$ defined above.)

6.2 The max-min continuous edge-partition problem

In this section, an $O(n \log^2 n)$-time algorithm for the max-min CEP problem is presented.

For each $v \in V(T) \setminus \{r_T\}$, let $q(v)$ be the smallest positive integer *s.t.* $l(\overline{vp(v)})/q(v)$ is feasible. In other words, $q(v) = \lceil l(\overline{vp(v)})/l^*_{p(1)} \rceil$. We first show that all $q(v), v \in V(T) \setminus \{r_T\}$ can be computed in $O(n \log n)$ time.

6.2.1 Computation of $q(v), v \in V(T) \setminus \{r_T\}$

Lemma 6.2.1 $p \leq \sum_{v \in V(T) \setminus \{r_T\}} q(v) \leq (p + n - 1)$.

Proof In an optimal solution, there are at most $q(v) + 1$ cuts and at least $q(v) - 1$ cuts on an edge $\overline{vp(v)}$ [49], which implies that $(p - n - 1) \leq \sum_{v \in V(T) \setminus \{r_T\}} q(v) \leq (p + n - 1)$ (since there are $p - 1$ cuts in an optimal solution). Moreover, since $q(v) \geq l(\overline{vp(v)})/l^*_p$, $\sum_{v \in V(T) \setminus \{r_T\}} q(v) \geq l(T)/l^*_p \Rightarrow \sum_{v \in V(T) \setminus \{r_T\}} q(v) \geq p$ (Lemma 6.1.2). □

Let z_1^* be the largest feasible real number in $Z_1 = \{l(\overline{vp(v)})/a | a \in [1, p], v \in V(T) \setminus \{r_T\}\}$. Lin et. al. [49] showed that $q(v) = \lceil l(\overline{vp(v)})/z_1^* \rceil$.

Lemma 6.2.2 *[49] For each $v \in V(T) \setminus \{r_T\}$, $q(v) = \lceil l(\overline{vp(v)})/z_1^* \rceil$.*

Therefore, it is sufficient to present an $O(n \log n)$-time algorithm to compute z_1^*.

Compute z_1^* It is not hard to obtain the following inequations.

$$q(v) = \lceil \frac{l(\overline{vp(v)})}{l^*_{p(1)}} \rceil \geq \lceil \frac{p \times l(\overline{vp(v)})}{l(T)} \rceil, v \in V \setminus \{r_T\} \text{ (Lemma 6.1.2)}, \tag{6-1}$$

$$p \leq \sum_{v \in V \setminus \{r_T\}} \lceil \frac{p \times l(\overline{vp(v)})}{l(T)} \rceil < p + n - 1. \tag{6-2}$$

By combining Equation 6-2 and Lemma 6.2.1, we have the following result.

$$\sum_{v \in V \setminus \{r_T\}} (q(v) - \lceil \frac{p \times l(\overline{vp(v)})}{l(T)} \rceil) \leq n - 1,$$

which implies that

$$0 \leq q(v) - \lceil \frac{p \times l(\overline{vp(v)})}{l(T)} \rceil \leq n - 1, v \in V \setminus \{r_T\} \text{ (Equation 6-1)}.$$

In other words, z_1^* is the largest feasible real number in

$$Z_1' = \{l(\overline{vp(v)})/a | a \in [\lceil \frac{p \times l(\overline{vp(v)})}{l(T)} \rceil, \lceil \frac{p \times l(\overline{vp(v)})}{l(T)} \rceil + n - 1], v \in V \setminus \{r_T\}\}.$$

The algorithm for the ratio search problem presented in Section 6.4 can be used to compute z_1^*. By using the linear time feasibility test (Lemma 6.1.3) and the result in Theorem 6.4.2, z_1^* can be computed in $O(n \log n)$. Therefore, we have the following lemma.

Lemma 6.2.3 *All $q(v), v \in V(T) \setminus \{r_T\}$, can be computed in $O(n \log n)$ time.*

6.2.2 The main idea and overall approach

Our approach for the solution of the max-min tree edge-partition model is to apply the algorithm described in Section 6.1.2 parametrically, using l as the single parameter, to compute $Z_1(l^*_{p(1)})$ without specifying the value of $l^*_{p(1)}$ a priori. Note that for a fixed value of the parameter, the algorithm is executed in $O(n)$ steps. At each step we possibly trim the lengths of some edges of the cluster by an integer multiple of the parameter l, and perform some additions and comparisons with the updated lengths of the edges. Imagine that we start the algorithm without specifying a value of the parameter l. The parameter is restricted to some interval which is known to contain the optimal value $l^*_{p(1)}$. (Initially, we may start with the interval $[0, l(T)]$.) As we go along, at each step of the algorithm we update and shrink the size of the interval, ensuring that it includes the optimal value $l^*_{p(1)}$.

The approaches in [37, 49] are also based on Megiddo's general parametric approach [52]. The main difference between our approach and the two previous approaches [37, 49] is described as follows. In the approaches in [37, 49], one feasibility test is needed at each vertex. But we plan to find the edge-partitions at all j^{th}-level spines in an optimal solution by solving $O(\log m_j)$ feasibility tests where m_j is the number of vertices in j^{th}-level spines, $j = 1, \cdots, \tau$. The details of this solution are presented in Section 6.2.3. Basically, we show that the edge-partitions at j^{th}-level spines in an optimal solution can be computed in time $O(n \log n + m_j \log^2 m_j)$. Therefore, based upon Lemma 6.1.1, i.e., $\tau = O(\log n)$, the max-min CEP can be solved in $O(n \log^2 n)$ time since $\sum_{j=1}^{\tau} m_j = n$.

Theorem 6.2.4 *The max-min CEP problem can be solved in $O(n \log^2 n)$ time.*

6.2.3 Computing edge-partitions at all j^{th}-level spines, $1 \le j \le \tau$

We assume that, for any $k \in (j, \tau]$, the number of cuts at edges in k^{th}-level spines are known, and that the remainder of each $(j+1)^{th}$-level spine is known. In Figure 6.2, an example is demonstrated. The bold paths are 2^{nd}-level spines and the dashed parts represent the remainders contributed from 3^{rd}-level spines. We note that the remainder from each 3^{rd}-level spine is a 1-degree polynomial of $l^*_{p(1)}$ in the form of $\eta - \kappa \times l^*_{p(1)}$ where η is a positive real number and κ is a nonnegative integer number, and that each remainder is less than $l^*_{p(1)}$.

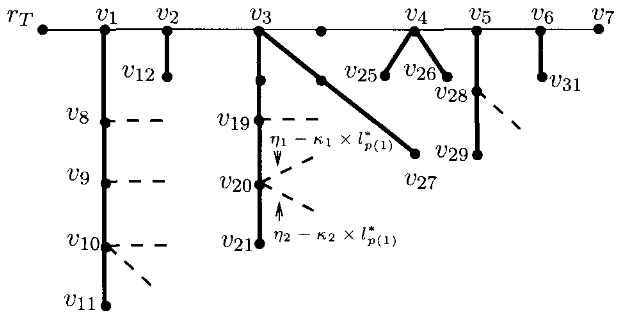


Figure 6.2: 2^{nd}-level spines and remainders of 3^{rd}-level spines.

We first merge the remainders that are attached to the same vertex. For example, in Figure 6.2, two remainders $\eta_1 - \kappa_1 \times l^*_{p(1)}$ and $\eta_2 - \kappa_2 \times l^*_{p(1)}$ from 3^{rd}-level spines are attached to vertex v_{20}, then we remove the two remainders and attach v_{20} with a new remainder

$\sum_{s=1}^{2} \eta_s - \sum_{s=1}^{2} \kappa_s \times l^*_{p(1)}$. We note that now it is possible to have new remainders that are $\geq l^*_{p(1)}$.

Let λ be the number of j^{th}-level spines and m_j be the total number of vertices on these j^{th}-level spines. For each j^{th}-level spine, i.e., $\Phi : \{v_0, \cdots, v_t\}$ (v_t is the root of Φ) in Figure 6.3, we create a balanced binary tree structure over it. Before we present the balanced binary tree structure over each j^{th}-level spine Φ, more notations and definitions are introduced as follows.

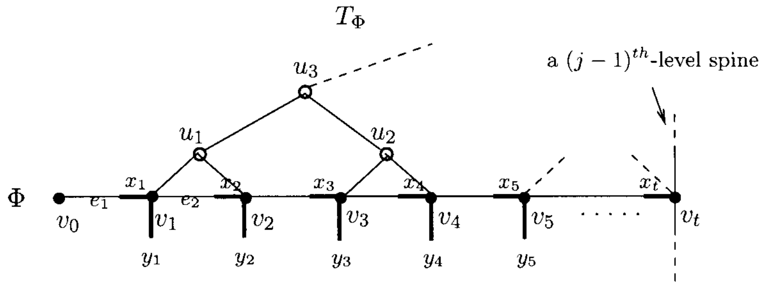


Figure 6.3: A balanced binary tree structure over a j^{th}-level spine Φ.

The edge connecting v_{i-1} and v_i is denoted by e_i, $i = 1, \cdots, t$. We define the remainder x_i of each edge e_i on Φ to be $l(e_i) - (q(v_{i-1}) - 1) \times l^*_{p(1)}$, $i = 1, \cdots, t$. Note that v_i is the parent of v_{i-1} in T and $q(v_{i-1}) = \lceil l(e_i)/l^*_{p(1)} \rceil$. Therefore, $0 < x_i \leq l^*_{p(1)}$. We continue with the hypothesis that $0 < x_i < l^*_{p(1)}$. For each $i, 1 \leq i < t$, we denote by y_i the remainder attached to vertex v_i (after merging remainders of $(j+1)^{th}$-level spines). All x_i's and y_i's are in the form of $\eta - \kappa \times l^*_{p(1)}$ where η is a positive real number and κ is a nonnegative integer number. Let $x_i = \eta_i - \kappa_i \times l^*_{p(1)}$ and $y_k = \eta'_k - \kappa'_k \times l^*_{p(1)}$ where $i = 1, \cdots, t$ and $k = 1, \cdots, t-1$ (y_t is undefined).

Figure 6.3 demonstrates a balanced binary tree structure over a spine Φ, denoted by T_Φ, where the vertices on Φ are leaves of T_Φ. We denote by T_u the subtree of T_Φ rooted at a leaf or an internal node u of T_Φ. Let $V(T_u)$ be the set of leaf vertices in T_u. For example, in Figure 6.3, $V(T_{v_2}) = \{v_2\}$ and $V(T_{u_2}) = \{v_3, v_4\}$. The term I_u is defined to be the smallest index of vertices contained in the subtree T_u of T_Φ, i.e., $I_{u_1} = I_{u_3} = 1$ and $I_{u_2} = 3$ in Figure 6.3.

Preprocessing step

In this step, we compute three sets of 1-degree polynomials with unknown $l^*_{p(1)}$, i.e., Z_r, Z'_r, Z''_r, and sort elements in these sets respectively. The three sets are described below.

For each vertex v_i in $V(T_u)$, let $z(u,i) = \sum_{k=I_u}^{i} x_k + \sum_{k=I_u}^{i-1} y_k$ and $z'(u,i) = \sum_{k=I_u}^{i} x_k + \sum_{k=I_u}^{i} y_k$. For example, in Figure 6.3, $z(u_2, 4) = x_3 + y_3 + x_4, z'(u_2, 4) = x_3 + y_3 + x_4 + y_4, z(v_4, 4) = x_4$, and $z'(v_4, 4) = x_4 + y_4$. Clearly, there are $O(m_j \log m_j)$ such values in j^{th}-level spines since each vertex belongs to $O(\log m_j)$ rooted subtrees. All these $z(\cdot,\cdot)$ and $z'(\cdot,\cdot)$ values are in the form of $\eta - \kappa \times l^*_{p(1)}$ where η is a positive real number and κ is a nonnegative integer number. Also, we have the following property about these values.

Lemma 6.2.5 *For each vertex v_i in $V(T_u)$, $z(u,i) < n \times l^*_{p(1)}$ and $z'(u,i) < n \times l^*_{p(1)}$.*

Proof The reason is that $z(u,i)$ (resp. $z'(u,i)$) is the sum of at most n remainders and each remainder is less than $l^*_{p(1)}$ by hypothesis. □

Let A be the set of constant parts of all these $z(\cdot,\cdot)$ 1-degree polynomials. As we know, each η_i in A is associated with a nonnegative integer κ_i. According to Lemma 6.2.5, $\kappa_i \times l^*_{p(1)} < \eta_i < (\kappa_i + n) \times l^*_{p(1)}$ for each $\eta_i \in A$. For each $\eta \in A$, let $q(\eta)$ be the smallest positive integer *s.t.* $\eta/q(\eta)$ is feasible. In other words, $q(\eta) = \lceil \eta / l^*_{p(1)} \rceil$.

Let z_2^* be the largest feasible real number in $Z_2 = \{\eta_i/f_i | \eta_i \in A, f_i \in [\kappa_i + 1, \kappa_i + n]\}$. We obtain that $q(\eta) = \lceil \eta / z_2^* \rceil$, for each $\eta \in A$ [49]. According to Theorem 6.4.2 and Lemma 6.1.3, z_2^* can be computed in $O(n \log n)$ since $|A| = O(m_j \log m_j)$. Therefore, all $q(\eta)$ ($\eta \in A$) can be computed in $O(n \log n)$ time.

Definition of Z_r and Z'_r: For each vertex v_i in $V(T_u)$, we let $z_r(u,i)$ be the remainder of $z(u,i)$ and let $z'_r(u,i)$ be $y_i + z_r(u,i)$, i.e., if $z(u,i) = \eta - \kappa \times l^*_{p(1)}$ then $z_r(u,i) = z(u,i) - (q(\eta) - \kappa - 1) \times l^*_{p(1)}$. Clearly, $0 < z_r(u,i) \le l^*_{p(1)}$. We continue with the hypothesis that $0 < z_r(u,i) < l^*_{p(1)}$. We define Z_r (resp. Z'_r) to be the set of these $z_r(\cdot,\cdot)$ (resp. $z'_r(\cdot,\cdot)$) 1-degree polynomials with unknown $l^*_{p(1)}$.

Definitions of Z''_r: Let v_i be a vertex lying in a j^{th}-level spine $\Phi : \{v_0, \cdots, v_t\}$. In the balanced binary tree structure T_Φ, there are $O(\log m_j)$ subtrees containing all the vertices in path $\pi(v_{i+1}, v_{t-1})$, say $T_{u_1}, \cdots, T_{u_h}$ (where $I_{u_1} = i + 1 < I_{u_2} < \cdots < I_{u_h} < t$). For each subtree $T_{u_s}, 1 \le s \le h$, let $z''(u_s, i) = \sum_{k=i+1}^{I_{u_s}-1} (x_k + y_k)$ (see Figure 6.4). Clearly, these

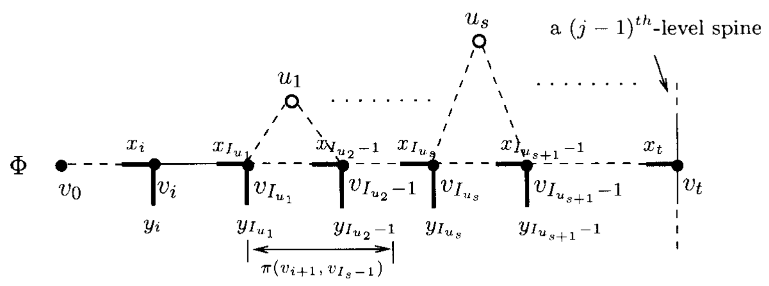


Figure 6.4: $z''(u_s, k), 1 \leq s \leq h$.

$z''(\cdot,\cdot)$ values are 1-degree polynomials with unknown l_p^*. Let $z_r''(u_s, i)$ be the remainder of $z''(u_s, i)$. Similar to the computation of $z_r(\cdot,\cdot)$, all these $z_r''(\cdot,\cdot)$ 1-degree polynomials can be computed in $O(n \log n)$ time. We define Z_r'' to be the set of these $z_r''(\cdot,\cdot)$ 1-degree polynomials.

Sort elements in the sets Z_r, Z_r', Z_r'': Obviously, each of Z_r, Z_r', and Z_r'' is of size $O(m_j \log m_j)$. A comparison between two 1-degree polynomials with unknown $l_{p(1)}^*$ can be resolved by solving one feasibility test. Under Valiant's comparison model [72], all the $z_r(\cdot,\cdot)$ (resp. $z_r'(\cdot,\cdot)$, $z_r''(\cdot,\cdot)$) can be sorted in $O(n \log^2 m_j)$ time by applying Megiddo's parametric-searching technique [53].

Actually, this sorting step can be speeded up by applying the result of Cole [25]. In this way, the sorting can be done in time $O(n \log m_j + m_j \log^2 m_j)$.

Finally, we have the following lemma.

Lemma 6.2.6 *The computation and sorting of elements of Z_r, Z_r', Z_r'', that is, the pre-processing step for computing edge-partitions at all j^{th}-level spines, can be done in time $O(n \log n + m_j \log^2 m_j)$, $1 \leq j \leq \tau$.*

Algorithm

Our algorithm to compute the edge-partitions at j^{th}-level spines in an optimal solution consists of two steps.

The first step is to locate vertex-cuts on all j^{th}-level spines in an optimal solution. The second step is to compute the remainder of each j^{th}-level spine.

First step: computing vertex-cuts We explore a property (see Lemma 6.2.7) between two consecutive vertex-cuts in an optimal solution. Based upon this property, given a vertex-cut on a j^{th}-level spine, we are able to locate the next vertex-cut efficiently, if it exists.

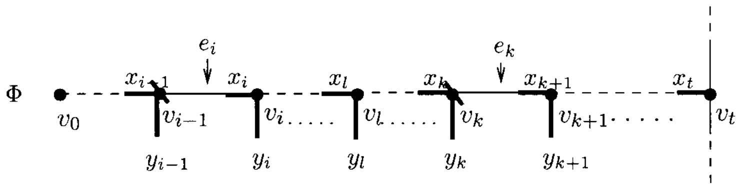


Figure 6.5: Lemma 6.2.7 shows a property of the next vertex-cut on vertex v_k and edge e_{k+1} after a vertex-cut on vertex v_{i-1} and edge e_i.

Lemma 6.2.7 *Refer to Figure 6.5. Assume that there is a vertex-cut on vertex v_{i-1} and edge e_i in an optimal solution. If the next vertex-cut is on vertex v_k and edge e_{k+1} ($i \le k \le t-1$), then*

$$\lceil \frac{\sum_{s=i}^{l}(x_s+y_s)}{l^*_{p(1)}} \rceil = \lceil \frac{\sum_{s=i}^{l-1}(x_s+y_s)+x_h}{l^*_{p(1)}} \rceil, \text{ where } i \le l < k,$$

and

$$\lceil \frac{\sum_{s=i}^{k}(x_s+y_s)}{l^*_{p(1)}} \rceil > \lceil \frac{\sum_{s=i}^{k-1}(x_s+y_s)+x_k}{l^*_{p(1)}} \rceil.$$

Proof In the algorithm to compute $Z_1(l)$ described in Section 6.1.2, we know that a vertex-cut happens when the sum of remainders of child edges is $\ge l$. Obviously, it is true for $l = l^*_{p(1)}$.

Therefore, the next vertex-cut after the vertex-cut on v_{i-1} and e_i will be on vertex v_k and edge v_{k+1} ($i \le k \le t-1$) if the condition

$$\lceil \frac{\sum_{s=i}^{k}(l(e_s)+y_s)}{l^*_{p(1)}} \rceil > \lceil \frac{\sum_{s=i}^{k-1}(l(e_s)+y_s)+l(e_k)}{l^*_{p(1)}} \rceil$$

is satisfied the first time, which is equivalent to the two conditions described in this lemma. □

Lemma 6.2.7 says that we can find all vertex-cuts on a j^{th}-level spine Φ one by one, starting from v_0 (we can assume that there is a vertex-cut on v_0 and e_1 since v_0 is a leaf vertex in T). However, this straightforward approach is inefficient and its running time is $O(m_j n)$ (in the worst case, we need to solve $O(m_j)$ feasibility tests).

In order to improve the running time of computing vertex-cuts on all j^{th}-level spines, we propose a parallel approach that is described as follows. We first describe an algorithm to compute the next vertex-cut for a vertex v_i in a j^{th}-level spine $\Phi : \{v_0, \cdots, v_t\}$ $(0 \leq i < t-1)$ with the assumption that there is a vertex-cut on v_i and e_{i+1}, and then present our parallel approach to compute the next vertex-cuts for all vertices in j^{th}-level spines.

Computing the next vertex-cut for a vertex v_i: refer to Figure 6.4. There are $O(\log m_j)$ subtrees of T_Φ containing all the vertices in path $\pi(v_{i+1}, v_{t-1})$, i.e., $T_{u_1}, \cdots, T_{u_h}$ $(I_{u_1} = i+1 < I_{u_2} < \cdots < I_{u_h} < t)$. For each subtree T_{u_s}, $1 \leq s \leq h$, we locate the first vertex $v_k \in V(T_{u_s})$ *s.t.* the following condition is satisfied.

$$C1:\ \lceil \frac{\sum_{a=i+1}^{k} (x_a + y_a)}{l^*_{p(1)}} \rceil > \lceil \frac{\sum_{a=i+1}^{k-1} (x_a + y_a) + x_k}{l^*_{p(1)}} \rceil.$$

Note that we might not be able to find such a vertex in some subtrees. If such a vertex does not exist for any subtree T_{u_s}, $1 \leq s \leq h$, then the next vertex-cut does not exist after v_i on spine Φ. Otherwise, let $T_{u_{s1}}$ be the first subtree in which such a type of vertex, say v_{i1}, does exist. It is not difficult to see that the next vertex-cut after v_i will be on vertex v_{i1} and edge e_{i1+1}.

We next show an approach to locate the first vertex $v_k \in V(T_{u_s})$ $(1 \leq s \leq h)$ *s.t.* *C1* is satisfied.

We depict important information of vertices in $V(T_{u_s}), 1 \leq s \leq h$, using a two-dimensional diagram (see Figure 6.6). For each $v_k \in V(T_{u_s})$, the horizontal coordinate of v_k in the two-dimensional diagram corresponds to $z_r(u_s, k)$ and the vertical coordinate of v_k corresponds to $z'_r(u_s, k)$. Note that these $z_r(u_s, \cdot)$ and $z'_r(u_s, \cdot)$ values are 1-degree polynomials and are already sorted in the preprocessing step. We denote by $\mathcal{I}$ the region in the two-dimensional diagram that contains all points whose horizontal coordinates are in $[0, l^*_{p(1)} - z''_r(u_s, i)]$ and whose vertical coordinates are in $(l^*_{p(1)} - z''_r(u_s, i), \infty)$. We denote by $\mathcal{II}$ the region in the two-dimensional diagram that contains all points whose horizontal coordinates are in $(l^*_{p(1)} - z''_r(u_s, i), \infty)$ and whose vertical coordinates are in $(2l^*_{p(1)} - z''_r(u_s, i), \infty)$.

Lemma 6.2.8 *Suppose that there is no vertex-cut in $\pi(v_{i+1}, v_{I_s - 1})$. Then, there is a vertex-cut in $\pi(v_{I_s}, v_{I_{s+1}-1})$ if and only if $\mathcal{I} \cup \mathcal{II}$ contains at least one point. Also, if points exist in*

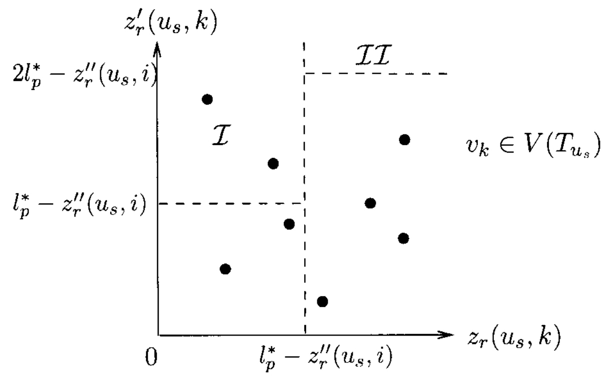


Figure 6.6: Check if there is a vertex-cut in T_{u_s} and locate it if exists.

$\mathcal{I} \cup \mathcal{II}$, let $i1$ be the smallest index among them, then the first vertex-cut in $\pi(v_{I_s}, v_{I_{s+1}-1})$ is on vertex v_{i1} and edge e_{i1+1}.

Proof We know that a vertex-cut happens after v_i if *C1* is satisfied for some vertex $v_k, i < k < t$. By assumption, there is no vertex-cut in $\pi(v_{i+1}, v_{I_s-1})$. That is,

$$\lceil\frac{\sum_{a=i+1}^{b}(x_a+y_a)}{l^*_{p(1)}}\rceil = \lceil\frac{\sum_{a=i+1}^{b-1}(x_a+y_a)+x_b}{l^*_{p(1)}}\rceil, \text{ where } i+1 \le b < I_s.$$

Therefore, there is a vertex-cut in $\pi(v_{I_s}, v_{I_{s+1}-1})$ if and only if *C1* is satisfied for some vertex $v_k \in V(T_{u_s})$.

Note that $z_r(u_s,k)$ is the remainder of $z(u_s,k) = \sum_{a=I_s}^{i} x_a + \sum_{a=I_s}^{i-1} y_a$ and $z''_r(u_s,i)$ is the remainder of $z''(u_s,i) = \sum_{a=i+1}^{I_s-1}(x_a+y_a)$. Assume that $z(u_s,k) = z_r(u_s,k) + \kappa \times l^*_{p(1)}$ and $z''(u_s,i) = z''_r(u_s,i) + \kappa'' \times l^*_{p(1)}$ where κ and κ'' are non-negative integers.

Since $\sum_{a=i+1}^{k-1}(x_a+y_a)+x_k = z''(u_s,i)+z(u_s,k) = z''_r(u_s,i)+z_r(u_s,k)+(\kappa+\kappa'')\times l^*_{p(1)}$,

$$\lceil\frac{\sum_{a=i+1}^{k-1}(x_a+y_a)+x_k}{l^*_{p(1)}}\rceil = \lceil\frac{z''_r(u_s,i)+z_r(u_s,k)}{l^*_{p(1)}}\rceil + \kappa + \kappa'',$$

and

$$\begin{aligned}\lceil\frac{\sum_{a=i+1}^{k}(x_a+y_a)}{l^*_{p(1)}}\rceil &= \lceil\frac{z''_r(u_s,i)+z_r(u_s,k)+y_i}{l^*_{p(1)}}\rceil + \kappa + \kappa'' \\ &= \lceil\frac{z''_r(u_s,i)+z'_r(u_s,k)}{l^*_{p(1)}}\rceil + \kappa + \kappa''.\end{aligned}$$

It is known that $0 < z_r(u_s, k) < l^*_{p(1)}$ and $0 < z''_r(u_s, i) < l^*_{p(1)}$. If $z_r(u_s, k) \leq l^*_{p(1)} - z''_r(u_s, i)$ then $\lceil (z''_r(u_s, i) + z_r(u_s, k))/l^*_{p(1)} \rceil = 1$. To satisfy *C1*, we need $\lceil (z''_r(u_s, i) + z'_r(u_s, k))/l^*_{p(1)} \rceil > 1$, i.e., $z'_r(u_s, k) > l^*_{p(1)} - z''_r(u_s, i)$. Hence, all points in the region $\mathcal{I}$ satisfy *C1*. Similarly, all points in the region $\mathcal{II}$ also satisfy *C1*. □

Using a priority search tree structure [51] to maintain the two-dimensional diagram for subtree T_{u_s}, we can check if there exists a vertex-cut and locate it if exists after $O(|V(T_{u_s})|) \subseteq O(\log m_j)$ comparisons between 1-degree polynomials with unknown $l^*_{p(1)}$. Note that one feasibility test needs to be solved for each comparison.

A priority search tree over $\Pi = V(T_{u_s})$ is created as follows (Algorithm 2).

```
**Algorithm 2** Create-PrioritySearchTree(Π)
Input: a set Π of points that are depicted in the two-dimensional diagram described above.
Output: the root of a priority search tree data structure over Π.
begin
  1: If Π is empty, terminate the process and return "NULL".
  2: The point $u$ in Π with the largest vertical coordinate becomes the root.
  3: Let $\Pi = \Pi \setminus \{u\}$ and if Π is empty, then terminate the process and return the pointer
     to $u$.
  4: Let $x_m(\Pi)$ be a value such that half of points in Π have horizontal coordinates lower
     that $x_m(\Pi)$, and half have higher.
  5: Recursively create a priority search tree on the lower half of Π and let its root be the
     left child of $u$.
  6: Recursively create a priority search tree on the upper half of Π and let its root be the
     left child of $u$.
end
```

Note that the vertical and horizontal coordinates of points in Π are 1-degree polynomials and are already sorted in the preprocessing step. Hence, we are able to create the priority search tree in $O(|\Pi| \log |\Pi|)$ time [51].

Computing the next vertex-cuts for vertices in j^{th}-level spines Since it is inefficient to locate vertex-cuts one by one, in order to speed up the computation, we can compute them in a parallel way. For each vertex v in a j^{th}-level spine, we need to locate candidates for its next vertex-cut among $O(\log m_j)$ subtrees. As shown above, a candidate in a subtree (for v) can be computed in $O(\log m_j)$ steps where each step is a comparison between two 1-degree polynomials with unknown $l^*_{p(1)}$.

Therefore, the computation of all possible vertex-cuts can be done in $O(\log m_j)$ parallel steps by using $O(m_j \log m_j)$ processors (there are $O(\log m_j)$ processors associated with each vertex), where each step is a comparison between two 1-degree polynomials with unknown $l^*_{p(1)}$. By applying Cole's idea [25], the computation can be done in time $O(n \log m_j + m_j \log^2 m_j)$.

Second step: computing remainders of spines After locating vertex-cuts on all j^{th}-level spines in an optimal solution (the first step), we are able to compute the remainder of each j^{th}-level spine efficiently. For a j^{th}-level spine $\Phi : \{v_0, \cdots, v_t\}$, assume that the last vertex-cut is on vertex v_k and edge e_{k+1}, $0 \leq k \leq t-1$ ($k = 0$ means that there is no vertex-cut on Φ). Then the remainder of spine Φ is the remainder of $\sum_{i=k+1}^{t} x_i + \sum_{i=k+1}^{t-1} y_i$. It is trivial that the 1-degree polynomial $\sum_{i=k+1}^{t} x_i + \sum_{i=k+1}^{t-1} y_i$ is less than $n \times l^*_{p(1)}$. We need to compute the remainders of λ 1-degree polynomials (λ is the number of j^{th}-level spines). Similar to the computation of $z_r(\cdot, \cdot)$ (in the preprocessing step), all these remainders can be computed in $O(n \log n)$ time.

From the above discussion, the total effort (including the effort for the preprocessing step) to compute the edge-partitions in j^{th}-level spines in an optimal solution is $O(n \log n + m_j \log^2 m_j)$. It completes the proof of Theorem 6.2.4.

6.3 The min-max continuous edge-partition problem

Unfortunately, we cannot use the same approach to solve the min-max CEP problem as the one for the max-min CEP problem (described in the previous section). The main difficulty is caused by vertex-cuts in the min-max model.

Recall that in Step 2 of the feasibility test of a given positive value l for the max-min model, a vertex-cut on v_i and $\overline{v_i p(v_i)}$ is added if $\mu \geq l$ where μ is the sum of remainders of edges $\overline{v_i v_{i(k)}}, 1 \leq k \leq t$ ($v_{i(k)}$ is a child of v_i, see Figure 6.7(a)(b)). However, in Step 2 of the feasibility test of l for the min-max model, if $\mu > l$, we need to find the largest $q (1 \leq q \leq t)$ such that the sum of the q smallest remainders of edges $\overline{v_i v_{i(k)}}, 1 \leq k \leq t$, is $\leq l$. Assume that the remainder of edge $\overline{v_i v_{i(k)}}$ is no more than the remainder of edge $\overline{v_i v_{i(k+1)}}, 1 \leq k \leq t-1$. A vertex-cut on v_i and $\overline{v_i v_{i(k)}}$ is added, $q+1 \leq k \leq t$, and the edge $\overline{v_i p(v_i)}$ is extended by the sum of remainders of edges $\overline{v_i v_{i(k)}}, 1 \leq k \leq q$ (see Figure 6.7(c)).

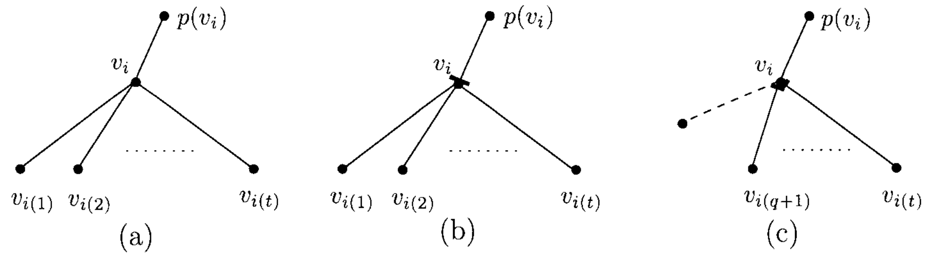


Figure 6.7: Vertex-cuts in the max-min and min-max models. (a) current vertex v_i and its children $v_{i(1)}, \dots, v_{i(t)}$ (assume that the sum of remainders of edges $\overline{v_i v_{i(k)}}, 1 \leq k \leq t$, is $\geq l$); (b) a vertex-cut on vertex v_i and edge $\overline{v_i p(v_i)}$ in the max-min model; (c) vertex-cuts at vertex v_i and edge $\overline{v_i v_{i(k)}}, q+1 \leq k \leq t$, in the max-min model, and the dashed part is the extension of edge $\overline{v_i p(v_i)}$ that is equal to the sum of remainders of edges $\overline{v_i v_{i(k)}}, 1 \leq k \leq q$.

Therefore, in the max-min model, we are able to locate next vertex-cuts for all vertices on a spine in a parallel way, but, we cannot do it in a parallel way for the min-max model since the length of edge $\overline{vp(v)}$ is changed even if we have vertex-cuts on vertex v and edges incident to it.

Main idea Our approach for the min-max problem is also to apply the algorithm described in Section 6.1.3 parametrically, using l as the single parameter, to compute $Z_2(l^*_{p(2)})$ without specifying the value of $l^*_{p(2)}$ a priori.

From the feasibility test for the min-max problem in Section 6.1.3, we can see that all current cluster vertices can be handled in a parallel way. In Section 6.3.1, we present an algorithm to compute the edge-partitions at all current cluster vertices in an optimal solution by solving logarithmic feasibility tests.

More notations are introduced as follows. For each $v \in V(T) \setminus \{r_T\}$, let $q'(v)$ be the smallest positive integer *s.t.* $l(\overline{vp(v)})/q'(v)$ is feasible in the min-max model. In other words, $q'(v) = \lceil l(\overline{vp(v)})/l^*_{p(2)} \rceil$. Similar to the computation of $q(v), v \in V(T) \setminus \{r_T\}$, we can compute all $q'(v), v \in V(T) \setminus \{r_T\}$, in $O(n \log n)$ time.

6.3.1 Computing edge-partitions at all current cluster vertices

Let $u_1, \dots, u_k$ be current cluster vertices. Let $v_{i(1)}, \dots, v_{i(n_i)}$ be the children of $u_i, i = 1 \dots, k$. Assume that the remainder of each edge $\overline{u_i v_{i(j)}}, 1 \leq i \leq k, 1 \leq j \leq n_i$, is known, denoted by $x_{i(j)}$, which is a 1-degree polynomial of unknown $l^*_{p(2)}$.

The algorithm consists of four steps as follows.

The first step is to compare $\sum_{j=1}^{n_i} x_{i(j)}, i = 1, \ldots, k$, with $l^*_{p(2)}$, which can be done by solving $O(\log k)$ feasibility tests. Assume that $\sum_{j=1}^{n_i} x_{i(j)} \geq l^*_{p(2)}, 1 \leq i \leq k'(\leq k)$. Let $y_i = \sum_{j=1}^{n_i} x_{i(j)}, k' < i \leq k$.

The second step is to sort the elements in the set $\{x_{i(j)} : 1 \leq i \leq k', 1 \leq j \leq n_i\}$. Similar to the sorting of elements in the sets Z_r, Z'_r, Z''_r (defined in Section 6.2.3), this step can be done in time $O(n \log \sum_{i=1}^{k'} n_i)$. Without loss of any generality, we assume that $x_{i(1)} \leq x_{i(2)} \leq \ldots \leq x_{i(n_i)}, 1 \leq i \leq k'$.

The third step is to compare $\sum_{j=1}^{q_i} x_{i(j)}, i = 1, \ldots, k', 1 \leq q_i \leq n_i$, with $l^*_{p(2)}$, which can be done by solving $O(\log \sum_{i=1}^{k'} n_i)$ feasibility tests. Then, for each $i, 1 \leq i \leq k'$, in $O(n_i)$ time, we can find the largest $q_i (1 \leq q_i \leq n_i)$ such that the sum of the q_i smallest remainders of edges $\overline{u_i v_{i(j)}}, 1 \leq j \leq n_i$, is $\leq l^*_{p(2)}$. In other words, there are vertex-cuts on vertex u_i and edges $\overline{u_i v_{i(j)}}, q_i + 1 \leq j \leq n_i, i = 1, \ldots, k'$ in an optimal solution. Let $y_i = \sum_{j=1}^{q_i} x_{i(j)}, 1 \leq i \leq k'$.

The last step is to compute the remainder of new edge $\overline{u_i p(u_i)}$ since $\overline{u_i p(u_i)}$ is extended by $y_i, 1 \leq i \leq k$. In this step, we only need to compare $y_i + l(\overline{u_i p(u_i)}) - (q'(u_i) - 1) \times l^*_{p(2)}$ with $l^*_{p(2)}, i = 1, \ldots, k$. Obviously, it can be done by solving $O(\log k)$ feasibility tests. For each $i, 1 \leq i \leq k$, if $y_i + l(\overline{u_i p(u_i)}) - (q'(u_i) - 1) \times l^*_{p(2)} > l^*_{p(2)}$ then the remainder of new edge $\overline{u_i p(u_i)}$ is $y_i + l(\overline{u_i p(u_i)}) - q'(u_i) \times l^*_{p(2)}$, and $y_i + l(\overline{u_i p(u_i)}) - (q'(u_i) - 1) \times l^*_{p(2)}$, otherwise.

Time analysis Clearly, the time complexity to compute the edge-partitions at all current cluster vertices in an optimal solution is $O(n \log \sum_{i=1}^{k} n_i) = O(n \log \sum_{i=1}^{k} \delta_T(u_i)) \subseteq O(n \log n)$ (recall that $\delta_T(u_i)$ is the degree of vertex u_i in T).

We denote by h_T the height of the underlying tree T. Then, the algorithm described above runs h_T times. The total time cost is therefore $O(n h_T \log n)$ (or $O(n \sum_{v \in V(T)} \delta_T(v))$).

Theorem 6.3.1 *The continuous min-max tree edge-partitioning problem can be solved in* $O(n h_T \log n)$ *(or* $O(n \sum_{v \in V(T)} \delta_T(v))$*) time, where* h_T *is the height of the underlying tree network and* $\delta_T(v)$ *is the degree of vertex* v.

6.4 An algorithm for the ratio search problem

In the ratio search problem, we are given a positive integer q, a set of k non-negative real numbers $\Delta = \{b_i, i = 1, \ldots, k\}$, a non-increasing function $\mathcal{F} : \mathbf{R} \rightarrow \mathbf{R}$, and a real number

t. Each number b_i in Δ is associated with a nonnegative integer number $g_i, 1 \leq i \leq k$. The problem is to find the largest real number, denoted b z^*, in $\{b_i/a_i, i = 1, \cdots, k \mid a_i \in [g_i + 1, g_i + q], b_i \in \Delta\}$ such that $\mathcal{F}(z^*) \geq t$. Let a^*, b^* be numbers *s.t.* $b^* \in \Delta$, a^* is an integer number in $[g^* + 1, g^* + q]$ (g^* is the value associated with b^*), and $z^* = b^*/a^*$. Without loss of any generality, we assume that $\mathcal{F}(0) \geq t$.

Lin et. al. [49] proposed an algorithm for the ratio search problem with the uniform value of $g_i = 0, i = 1, \cdots, k$, which runs in time $O(k + t_{\mathcal{F}} \times (\log k + \log q))$, where $t_{\mathcal{F}}$ is the time required to evaluate the function value $\mathcal{F}(z)$ for any real number z. In this section, we present an approach that solves the ratio search problem with the same time complexity, for non-uniform values of $g_i, i = 1, \cdots, k$.

Notation Δ': Let Δ' be the subset of Δ *s.t.* $\mathcal{F}(b_i/(g_i + q)) \geq t$ for any $b_i \in \Delta'$.

Clearly, $b^* \in \Delta'$ since $\mathcal{F}(\cdot)$ is a non-increasing function. If Δ' is empty then z^* does not exist, and if Δ' contains only one element then b^* is determined. In the latter case, we can compute z^* by a binary search of the corresponding a^* in $[g^* + 1, g^* + q]$, which takes time $O(t_{\mathcal{F}} \times \log q)$. We assume that Δ' contains at least two elements in the following.

For each $b_i \in \Delta'$, we denote by $a(b_i)$ the smallest integer number in $[g_i + 1, g_i + q]$ with $\mathcal{F}(b/a(b_i)) \geq t$.

Notations Δ'_1, Δ'_2: Let $\Delta'_1 = \{b_i | a(b_i) > g_i + 1, b_i \in \Delta'\}$ and let $\Delta'_2 = \Delta' \setminus \Delta'_1$.

Obviously, $\mathcal{F}(b_i/(g_i + 1)) \geq t$ for any $b_i \in \Delta'_2$, but, $\mathcal{F}(b_i/(g_i + 1)) < t$ for any $b_i \in \Delta'_1$. We can identify the elements in Δ'_1 and Δ'_2 by sorting the elements in Δ' and evaluating values of $\mathcal{F}(\cdot)$ for $O(\log |\Delta'|)$ elements in Δ'.

Notation z^*_2: Let $z^*_2 = \max_{b_i \in \Delta'_2} b_i/(g_i + 1)$.

It is trivial to see that $z^* \geq z^*_2$ since $\mathcal{F}(z^*_2) \geq t$. In the following, we assume that $z^* > z^*_2$.

The case when $z^* > z^*_2$ In this case, b^* must be in Δ'_1. We define $k_b(z) = \lfloor b/z \rfloor$ for each $b \in \Delta'_1$ where z is a parameter in $(0, \infty)$. Let $f(z) = \sum_{b \in \Delta'_1} k_b(z)$. We observe that $f(z)$ is a step function with jumps, including the jump point at z^*.

Notations a', z': Let a' be the smallest integer number *s.t.* $\mathcal{F}((\sum_{b \in \Delta'_1} b)/a') \geq t$. Let $z' = (\sum_{b \in \Delta'_1} b)/a'$.

Lemma 6.4.1 $\sum_{b_i \in \Delta'_1} g_i + |\Delta'_1| < a' \leq \sum_{b_i \in \Delta'_1} g_i + q \times |\Delta'_1|$.

Proof Since $\mathcal{F}(b_j/(g_j+1)) < t$ for any $b_j \in \Delta_1'$ and $\mathcal{F}((\sum_{b_i\in\Delta_1'} b_i)/a') \geq t$, $(\sum_{b_i\in\Delta_1'} b_i)/a' < b_j/(g_j+1)$ for any $b_j \in \Delta_1'$ (because $\mathcal{F}(\cdot)$ is a non-increasing function). For any $b_j \in \Delta_1'$,

$$(\sum_{b_i\in\Delta_1'} b_i) \times (g_j+1) < b_j \times a'.$$

Then,

$$(\sum_{b_i\in\Delta_1'} b_i) \times (\sum_{b_i\in\Delta_1'} g_i + |\Delta_1'|) < (\sum_{b_j\in\Delta_1'} b_j) \times a',$$

which implies that $a' > \sum_{b_i\in\Delta_1'} g_i + |\Delta_1'|$.

Let $b' = \max_{b_i\in\Delta_1'} b_i/(g_i+q)$. Therefore,

$$b' \times (\sum_{b_i\in\Delta_1'} g_i + q \times |\Delta_1'|) \geq \sum_{b_i\in\Delta_1'} b_i.$$

Since $\mathcal{F}(b') \geq t$ and $\mathcal{F}(\cdot)$ is a non-increasing function, $\mathcal{F}((\sum_{b\in\Delta_1'} b)/(\sum_{b_i\in\Delta_1'} g_i + q \times |\Delta_1'|)) \geq t$. Hence, $a' \leq \sum_{b_i\in\Delta_1'} g_i + q \times |\Delta_1'|$ because a' is the smallest integer number *s.t.* $\mathcal{F}((\sum_{b\in\Delta_1'} b)/a') \geq t$. □

Since there are at most $(q-1) \times |\Delta_1'|$ candidate integer values for a' (Lemma 6.4.1), we are able to compute a' in time $O(k + t_{\mathcal{F}} \times (\log q + \log k))$ (note that $|\Delta_1'| \leq k$).

For any $b \in \Delta_1'$, we can see that $b/a(b) < (\sum_{b_i\in\Delta_1'} b_i)/(a'-1)$ since $\mathcal{F}((\sum_{b_i\in\Delta_1'} b_i)/(a'-1)) < t$ and $\mathcal{F}(b/a(b)) \geq t$. (Recall that $a(b)$ is the smallest integer number in $[g+1, g+q]$ with $\mathcal{F}(b/a(b)) \geq t$ where g is the value associated with b.)

Similarly, for any $b \in \Delta_1'$, $z' < b/(a(b)-1)$ since $\mathcal{F}(z') \geq t$ and $\mathcal{F}(b/(a(b)-1)) < t$ (note that $a(b) > 1$).

For any $b \in \Delta_1'$,

$$\begin{aligned} & b/a(b) < (\sum_{b_i\in\Delta_1'} b_i)/(a'-1) \\ \Rightarrow\ & a(b) > b(a'-1)/\sum_{b_i\in\Delta_1'} b_i \\ \Rightarrow\ & a(b) > (b/z') - 1 \text{ (since } z' = (\sum_{b\in\Delta_1'} b)/a'); \end{aligned}$$

and

$$\begin{aligned} & z' < b/(a(b)-1) \\ \Rightarrow\ & a(b) < (b/z') + 1. \end{aligned}$$

Therefore, $(b/z') - 1 < a(b) < (b/z') + 1$ for any $b \in \Delta'_1$. We note that $a(b)$ is an integer and that there are at most two integers between $(b/z') - 1$ and $(b/z') + 1$ (not including $(b/z') - 1, (b/z') + 1$), denoted by a^1_b and a^2_b. It is not difficult to see that z^* is in the set $\{b/a^1_b, b/a^2_b | b \in \Delta'_1\}$ if $z^* > z^*_2$.

The finding can be made by using the prune-and-search technique. First, we compute the median x of the numbers in $\{b/a^1_b, b/a^2_b | b \in \Delta'_1\}$ by the linear-time select algorithm in [16]. If $\mathcal{F}(x) \geq t$ we prune away all numbers smaller than x in this set; otherwise, we prune away all numbers larger than x. The process is repeated on the remaining numbers. Note that the size of this set is no more than $2|\Delta'_1| \leq 2k$. Therefore, the above finding of the largest real number z^*_3 in $\{b/a^1_b, b/a^2_b | b \in \Delta'_1\}$ such that $\mathcal{F}(z^*_3) \geq t$ requires $O(k + t_{\mathcal{F}} \times \log k)$ time.

Based upon the above discussion, an algorithm for the ratio search problem is presented as follows (Algorithm 3).

Algorithm 3 Ratio-Search$(q, \mathcal{F}, t, \Delta)$

Input: an integer $q > 0$, a non-increasing function $\mathcal{F}$, a real number t, a set Δ of k non-negative real numbers, and each real number b_i in Δ is associated with a nonnegative integer $g_i, i = 1, \cdots, k$.

Output: the largest real number z in $\{b_i/a_i | a_i \in [g_i + 1, g_i + q], b_i \in \Delta\}$ such that $\mathcal{F}(z) \geq t$.

```
begin
1: $b''$ ← the largest number in $\{b_i/(g_i + q) | b_i \in \Delta\}$ s.t. $\mathcal{F}(b'') \geq t$.
2: $\Delta'$ ← the subset of $\Delta$ that contains all the numbers $b_i \in \Delta$ with $b_i/(g_i + q) \leq b''$.
3: $z^*_2$ ← the largest real number in $\{b_i/(g_i + 1) | b_i \in \Delta'\} \cup \{0\}$ s.t. $\mathcal{F}(z^*_2) \geq t$.
4: $\Delta'_1$ ← the subset of $\Delta'$ that contains all the numbers $b_i \in \Delta'$ with $b_i/(g_i + 1) > z^*_2$.
5: $a'$ ← the smallest integer number s.t. $\mathcal{F}((\sum_{b \in \Delta'_1} b)/a') \geq t$.
6: $z'$ ← $(\sum_{b \in \Delta'_1} b)/a'$.
7: $a^1_b, a^2_b$ ← the two integers between $(b/z') - 1$ and $(b/z') + 1$, for each $b \in \Delta'_1$.
8: $z^*_3$ ← the largest real number in $\{b/a^1_b, b/a^2_b | b \in \Delta'_1\}$ such that $\mathcal{F}(z^*_3) \geq t$.
9: return $\max\{z^*_2, z^*_3\}$.
end
```

The time complexity of Algorithm 3 is analyzed as follows. Both Line 1 and Line 3 can be done in time $O(k + t_{\mathcal{F}} \times \log k)$ by the prune-and-search technique described above for completing Line 8. It is easy to see that the steps in Line 2, Line 4, Line 6, and Line 7 can be done in $O(k)$ time. It is known that Line 5 can be completed in time $O(k + t_{\mathcal{F}} \times (\log q + \log k))$. Therefore, the ratio search problem can be solved in time $O(k + t_{\mathcal{F}} \times (\log q + \log k))$.

Theorem 6.4.2 *The ratio search problem can be solved in time* $O(k + t_{\mathcal{F}} \times (\log q + \log k))$.

6.5 Summary

In this chapter we study continuous tree p-edge-partition problems on a tree network. Basically, a continuous p-edge-partition of a tree T is to divide T into p subtrees by selecting $p-1$ cut points along the edges of the underlying tree. The objective is to minimize (maximize) the maximum (minimum) length of the subtrees. We propose an $O(n \log^2 n)$-time algorithm for the max-min problem and an $O(nh_T \log n)$-time algorithm for the min-max problem where h_T is the height of the underlying tree network.

Similar to the approaches developed in [37, 49], our approaches are also based on the general parametric approach of Megiddo [52]. The main difference between our approach and the two previous ones [37, 49] is that: In their approaches, one feasibility test is needed at each vertex; However, in our approach for the max-min problem, we build a spine tree decomposition structure [9] over the underlying tree and locate edge-partitions at all spines at the same level by solving logarithmic feasibility tests, and in our approach for the min-max problem, we locate the edge-partitions at all current cluster vertices by solving logarithmic feasibility tests. In this way, we are able to solve the max-min CEP in sub-quadratic time since the highest level is $O(\log n)$ in a spine tree decomposition structure, and we are able to solve the min-max CEP in time $O(nh_T \log n)$.

In [37], Halman and Tamir mentioned that their algorithms for the CEP problems can be extended to yield polynomial algorithms of the same complexity for the CEP problems in cactus networks, that is, $O(n^2 \log(\min\{p, n\}))$. We conjecture that our algorithms for the CEP problems in tree networks can be extended to cactus networks.

Chapter 7

Constrained covering problems in tree networks

In this chapter we consider a variant of the covering location problem called a constrained covering problem (for short, *CCP*). This problem is defined on a network G with vertex set $V(G)$ and edge set $E(G)$. The vertex set $V(G)$ represents the set of demand points that must be covered by a facility, as well as the set of potential facility locations. A facility located at vertex $u \in V(G)$ incurs a non-negative open-facility cost $c(u)$, and provides a non-negative coverage radius of $r(u)$. A demand point v is covered by a facility u if and only if $u \neq v$ and $d(u, v) \leq r(u)$. In other words, a demand point is covered by a facility if it lies within the coverage radius of the facility and an established facility must be covered by another established facility. The CCP seeks to minimize the sum of open-facility costs required to cover all vertices in $V(G)$.

We propose efficient algorithms for CCPs on path, extended star, and tree networks in this chapter. Our results improve the previous results in [45, 44]. In particular, we provide an $O(n \log n)$-time algorithm for path networks (the algorithm in [45] for path networks runs in $O(n^2)$ time), an $O(n^{1.5} \log n)$-time algorithm for extended-star networks (the algorithm in [45] for extended-star networks runs in $O(n^2)$ time), and an $O(n^3 \log n)$-time algorithm for tree networks (the algorithm in [44] for tree networks runs in $O(n^4)$ time).

Organization of the chapter Section 7.1 and Section 7.2 present our sub-quadratic algorithms for the CCP on path networks and extended-star networks, respectively. Section 7.3 develops an $O(n^3 \log n)$-time algorithm for the CCP on tree networks. A brief conclusion is given in Section 7.4.

7.1 Path networks

In this section, we present a sub-quadratic algorithm for the CCP on a path network G. The path network G has vertex set $V(G) = \{v_1, \dots, v_n\}$ and edge set $E(G) = \{\overline{v_i v_{i+1}}, i = 1, \dots, n-1\}$ with vertex v_1 designating the beginning of the path and vertex v_n designating the end of the path.

7.1.1 Recursive functions for computing an optimal solution

Lunday et al. [50] considered the CCP problem on a path network with uniform coverage radius, that is, $r(v_1) = r(v_i), i = 2, \dots, n$, and proposed a dynamic programming algorithm for it. The dynamic programming algorithm iteratively calculates two costs for each vertex $v_i \in V(G)$: the *protected cost* $p_c(v_i)$, and the *unprotected cost* $u_c(v_i)$. The protected cost $p_c(v_i)$ of v_i is the minimum cost to locate a facility at vertex v_i and cover vertices v_1 through v_i, with no facilities placed at vertices $v_{i+1}, \dots, v_n$. The unprotected cost $u_c(v_i)$ of v_i, is the minimum cost to locate a facility at vertex v_i and cover vertices v_1 through v_{i-1}, with no facilities placed at vertices $v_{i+1}, \dots, v_n$.

Later, Horne and Smith [45] generalized the algorithm of Lunday et al. [50] to solve the CCP on path networks with non-uniform coverage radii. We will present the recursive functions introduced in [45], which are used to compute the protected costs and unprotected costs. Before that, more notations are needed.

The *upper reach* $g(v_i)$ and the *lower reach* $h(v_i)$ of vertex $v_i (1 \leq i \leq n)$ are defined as follows.

$$g(v_i) = \max_{i \leq j \leq n} \{j : d(v_i, v_j) \leq r(v_i)\};$$

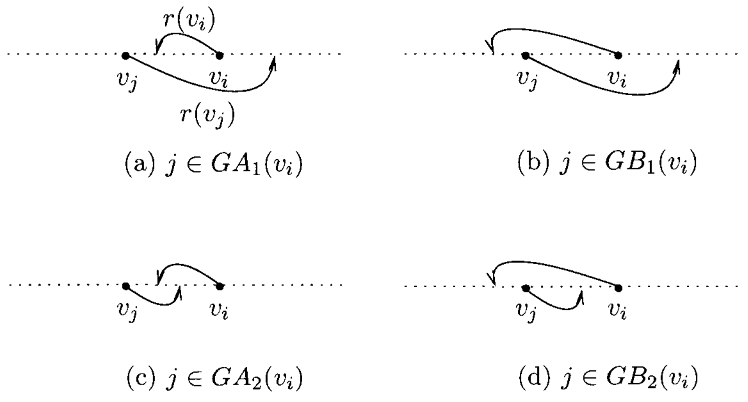
$$h(v_i) = \min_{1 \leq j \leq i} \{j : d(v_i, v_j) \leq r(v_i)\}.$$

In other words, the upper reach $g(v_i)$ (resp. lower reach $h(v_i)$) indicates the largest (resp. smallest) index of the vertex that is within the coverage radius of vertex v_i. The reach functions $g(v_i)$ and $h(v_i), i = 1, \dots, n$, are easily computable in $O(n \log n)$ time.

To compute the protected cost $p_c(v_i), 1 \leq i \leq n$, all vertices between v_1 and v_{i-1} that can cover v_i need to be identified. The indices of these vertices are identified in two separate sets $GA_1(v_i), GA_2(v_i)$ (see Figure 7.1(a)(b) for reference), defined as follows. Additionally, a dummy vertex v_0 is defined with $r(v_0) = g(v_0) = p_c(v_0) = 0$ to initialize the algorithm.

$$GA_1(v_i) = \{j : 1 \leq j < h(v_i) \leq i \leq g(v_j)\}$$

$$GB_1(v_i) = \{j : h(v_i) \leq j < i \leq g(v_j)\}$$



(a) $j \in GA_1(v_i)$ (b) $j \in GB_1(v_i)$

(c) $j \in GA_2(v_i)$ (d) $j \in GB_2(v_i)$

Figure 7.1: $GA_1(v_i), GB_1(v_i), GA_2(v_i)$, and $GB_2(v_i)$.

The difference between $GA_1(v_i)$ and $GB_1(v_i)$ is that for a vertex $v_j, 1 \leq j < i$, that can cover v_i. If v_i can cover v_j, then $j \in GB_1(v_i)$ and $j \in GA_1(v_i)$, otherwise. The subsets $GA_2(v_i), GB_2(v_i)$, defined as follows, are identified for each vertex $v_i, 1 \leq i \leq n$, in order to calculate the unprotected costs.

$$GA_2(v_i) = \{j : 0 \leq j < h(v_i) \leq g(v_j) + 1 \leq i\}$$

$$GB_2(v_i) = \{j : h(v_i) \leq j \text{ and } g(v_j) < i\}$$

Note that the union of $GA_2(v_i)$ and $GB_2(v_i)$ contain the indices of all vertices $v_j, j < i$, that do not cover v_i, but that can cover demand vertices v_{j+1} through v_{i-1} in conjunction with v_i. For a vertex $v_j, j < i \leq n$, with $g(v_j) < i$ and with $h(v_i) \leq g(v_j) + 1$, if v_i cannot cover v_j, then $j \in GA_2(v_i)$ (see Figure 7.1(c)) and $j \in GB_2(v_i)$, otherwise (see Figure 7.1(d)).

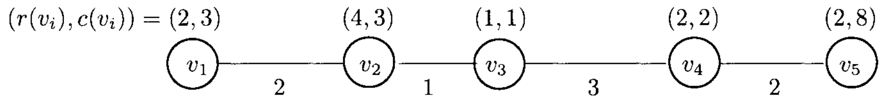
The recursive functions to compute the protected cost $p_c(v_i)$ and unprotected cost $u_c(v_i)$ for vertex $v_i, i = 1 \ldots, n$, are presented as follows [45].

$$p_c(v_i) = \min\{\min_{j \in GA_1(v_j)} p_c(v_i), \min_{j \in GB_1(v_i)} u_c(v_j)\} + c(v_i); \tag{7.1-1}$$

$$u_c(v_i) = \min\{p_c(v_i), \min_{j \in GA_2(v_i)} \{p_c(v_j)\} + c(v_i), \min_{j \in GB_2(v_i)} \{u_c(v_j)\} + c(v_i)\}. \qquad (7.1\text{-}2)$$

The optimal solution is determined by the minimum protected cost for all vertices having an upper reach equal to n.

However, the above recursive functions [45] cannot always produce an optimal solution. A counter-example is shown in Figure 7.2. Numbers below edges indicate the length of



Vertex	v_1	v_2	v_3	v_4	v_5
$GA_1(\cdot)$	$\emptyset$	$\emptyset$	$\emptyset$	$\{2\}$	$\emptyset$
$GB_2(\cdot)$	$\emptyset$	$\{1\}$	$\{2\}$	$\emptyset$	$\{4\}$
$GA_2(\cdot)$	$\{0\}$	$\{0\}$	$\{1\}$	$\emptyset$	$\{2\}$
$GB_2(\cdot)$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
$p_c(\cdot)$	∞	6	4	8	16
$u_c(\cdot)$	3	3	4	8	14

Figure 7.2: A counter-example for the recursive functions in [45]

edges. According to Equations 7.1-1 and 7.1-2, $\{v_1, v_2, v_4\}$ is the computed solution, and the cost is 8. But the optimal solution is $\{v_2, v_3, v_4\}$, and the optimal cost is 6. In this counter-example, the protected cost $p_c(v_4)$ is not computed correctly. It is easy to see that $GA_1(v_4) = \{2\}$ and $GB_1(v_4) = \emptyset$. According to Equation 7.1-1, the protected cost $p_c(v_2)$ is used to compute $p_c(v_4)$. But, the correct computation of $p_c(v_4)$ uses the unprotected cost $u_c(v_2)$ since v_2 can be covered by v_3, and the open-facility cost $c(v_3)$ of v_3 is small.

From the above counter-example, we can see that to compute the protected cost $p_c(v_i), 1 \leq i \leq n$, it is possible to use the unprotected cost $u_c(v_j), j \in GA_1(v_i)$, if v_j is covered by v_k, located between v_j and v_i, and has a small open-facility cost. We redefine the four subsets and two recursion functions in the following.

Definitions of $GA'_1(v_i), GB'_1(v_i), GA'_2(v_i)$, and $GB'_2(v_i), 1 \leq i \leq n$: In a solution Θ : $\{v_{i_1}, \ldots, v_{i_k}\}$ $(i_1 < \cdots < i_k)$ of the CCP on path G, a facility $v_{i_j}, 1 \leq j \leq k$ is called a *critical facility* in Θ if $h(v_{i_j}) < h(v_{i_{j'}}), j' = j+1, \ldots, k$, and otherwise called a *non-critical* *facility.* Clearly, the purpose to open a non-critical facility v_{i_j} is to cover some critical facility $v_{i_{j'}}$ in Θ. All the vertices (except $v_{i_{j'}}$) that can be covered by the non-critical facility v_{i_j} are covered by $v_{i_{j'}}$.

For each $i = 1, \ldots, n$, let

$$w_1'(v_i) = \min_{1 \le j < i} \{c(v_j) : i \le g(v_j)\}$$

and

$$w_2'(v_i) = \min_{i < j \le n} \{c(v_j) : h(v_j) \le i\}.$$

That is, $w_1'(v_i)$ (resp. $w_2'(v_i)$) is the smallest open-facility cost of vertices which can cover v_i and whose indices are smaller (resp. larger) than i. It is possible that one of $w_1'(v_i)$ and $w_2'(v_i)$ does not exist, in which case we assume its value to be ∞. Let

$$w'(v_i) = \min\{w_1'(v_i), w_2'(v_i)\},$$

that is, $w'(v_i)$ is the smallest open-facility cost of vertices that can cover v_i.

Observation 7.1.1 *In an optimal solution, the open-facility cost of a non-critical facility used to cover a critical facility v_i is equal to $w'(v_i)$.*

In our algorithm, we will pre-compute the value of $w'(v_i)$ for all $i = 1, \ldots, n$, and hence the focus is to locate critical facilities in an optimal solution.

The four subsets, $GA_1'(v_i), GB_1'(v_i), GA_2'(v_i), GB_2'(v_i)$ for each vertex $v_i, 1 \le i \le n$, are defined as follows.

$$GA_1'(v_i) = GA_1(v_i)$$

$$GB_1'(v_i) = \{j : h(v_j) < h(v_i) \le j < i \le g(v_j)\} \text{ (Figure 7.3(a))}$$

$$GA_2'(v_i) = \{j : 0 \le j < h(v_i) \le g(v_j) + 1\} \text{ (Figure 7.3(b)(c))}$$

$$GB_2'(v_i) = \{j : h(v_j) < h(v_i) \le j < i\} \text{ (Figure 7.3(d))}$$

Similar to $GA_1(v_i), GB_1(v_i), GA_2(v_i)$, and $GB_2(v_i)$, we identify subsets $GA_1'(v_i), GB_1'(v_i)$ to compute the protected cost of v_i, and subsets $GA_2'(v_i), GB_2'(v_i)$ to compute the unprotected cost of v_i.

From the definitions of $GA_1'(v_i), GB_1'(v_i), GA_2'(v_i)$, and $GB_2'(v_i), 1 \le i \le n$, we can see that for any index j in $GA_1'(v_i) \cup GB_1'(v_i) \cup GA_2'(v_i) \cup GB_2'(v_i)$, $h(v_j) < h(v_i)$. Therefore, if v_i is a critical facility in an optimal solution, then $v_j, j \in GA_1'(v_i) \cup GB_1'(v_i) \cup GA_2'(v_i) \cup GB_2'(v_i)$, is a candidate for a critical facility lying left of v_i in that optimal solution.

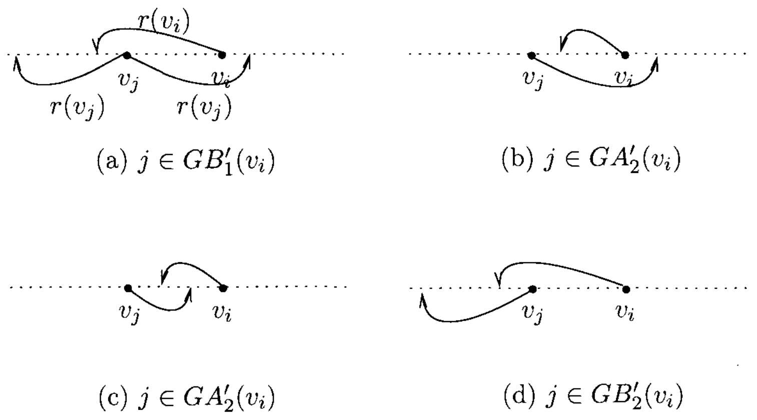


Figure 7.3: $GB'_1(v_i), GA'_2(v_i)$ and $GB'_2(v_i)$.

Recursive functions $p'_c(\cdot)$ and $u'_c(\cdot)$: Equations 7.1-3 and 7.1-4 are recursive functions used to compute the new protected costs $p'_c(v_i)$ and unprotected costs $u'_c(v_i), 1 \le i \le n$.

$$p'_c(v_i) = \min \left\{ \begin{array}{l} \min_{j \in GA'_1(v_i)} \{p'_c(v_j), u'_c(v_j) + w'_2(v_j)\} + c(v_i), \\ \min_{j \in GB'_1(v_i)} \{u'_c(v_j)\} + c(v_i), \\ u'_c(v_i) + w'_1(v_i) \end{array} \right\}; \qquad (7.1\text{-}3)$$

$$u'_c(v_i) = \min \left\{ \begin{array}{l} \min_{j \in GA'_2(v_i)} \{p'_c(v_j), u'_c(v_j) + w'_2(v_j)\}, \\ \min_{j \in GB'_2(v_i)} \{u'_c(v_j)\} \end{array} \right\} + c(v_i). \qquad (7.1\text{-}4)$$

In the computation of the protected cost $p'_c(v_i), 1 \le i \le n$, current critical facility v_i is covered either by a non-critical facility, in which case the protected cost $p'_c(v_i) = u'_c(v_i) + w'_1(v_i)$, or by a critical facility in $GA'_1(v_i) \cup GB'_1(v_i)$. The vertex in $GA'_1(v_i)$ must be covered either by a smaller indexed facility or by a non-critical facility lying between v_{j+1} and v_{i-1}. Because the vertices in $GB'_1(v_i)$ are covered by v_i, the unprotected costs for these vertices are examined when computing $p'_c(v_i)$.

In the computation of the unprotected cost $u'_c(v_i), 1 \le i \le n$, since a vertex v_j in $GA'_2(i)$ is not covered by v_i, its protected cost is used if v_j is covered by a smaller indexed facility or its unprotected cost is used if v_j is covered by a non-critical facility lying between v_{j+1} and v_{i-1}. On the other hand, vertices in $GB'_2(v_i)$ are covered by v_i, and hence we use their unprotected costs when computing $u'_c(v_i)$.

The optimal cost is still determined by the minimum protected cost for all vertices having an upper reach equal to n, that is, $\min_{1 \le i \le n : g(v_i) = n} p'_c(v_i)$.

We next present another pair of recursions $p''_c(\cdot)$ and $u''_c(\cdot)$, which can be used to compute an optimal solution and are simpler than Equations 7.1-3 and 7.1-4.

Recursive functions $p''_c(\cdot)$ and $u''_c(\cdot)$: We relax the constraint mentioned in the definition of protected costs, that is, to compute the protected cost $p''_c(v_i)$ of v_i, it is allowed that v_i may be covered by $v_j (j > i)$ with open-facility cost $w'(v_i)$. Then, we redefine the four subsets and two recursion functions as follows.

$$GA''_1(v_i) = GA_1(v_i) = \{j : 1 \le j < h(v_i) \le i \le g(v_j)\}$$

$$GB''_1(v_i) = GB_1(v_i) = \{j : h(v_i) \le j < i \le g(v_j)\}$$

$$GA''_2(v_i) = GA'_2(v_i) = \{j : 0 \le j < h(v_i) \le g(v_j) + 1\}$$

$$GB''_2(v_i) = \{j : h(v_i) \le j < i\}$$

$$p''_c(v_i) = \min \left\{ \begin{array}{l} \min_{j \in GA''_1(v_i)} \{p''_c(v_j)\} + c(v_i), \\ \min_{j \in GB''_1(v_i)} \{u''_c(v_j)\} + c(v_i), \\ u''_c(v_i) + w'(v_i) \end{array} \right\}; \tag{7.1-5}$$

$$u''_c(v_i) = \min \left\{ \begin{array}{l} \min_{j \in GA''_2(v_i)} \{p''_c(v_j)\}, \\ \min_{j \in GB''_2(v_i)} \{u''_c(v_j)\} \end{array} \right\} + c(v_i). \tag{7.1-6}$$

It is not hard to verify that the pair of recursions will generate a solution that can cover all vertices if the optimal cost is not infinite. We next show that $p''_c(v_i) \le p'_c(v_i)$ and $u''_c(v_i) \le u'_c(v_i), 1 \le i \le n$ (Lemma 7.1.2). Therefore, $\min_{1 \le i \le n; g(v_i)=n} p''_c(v_i)$ is the optimal cost.

Lemma 7.1.2 *$p''_c(v_i) \le p'_c(v_i)$ and $u''_c(v_i) \le u'_c(v_i), 1 \le i \le n$.*

Proof We prove the lemma by induction. When $i = 1$, the lemma is trivially true since $p'_c(v_i) = \infty, u'_c(v_i) = c(v_1)$ and $p''_c(v_i) = c(v_1) + w'(v_1), u''_c(v_i) = c(v_1)$. Assume that the lemma is true for any $i, 1 \le i \le j < n$. Next, we prove that the lemma is true for $i = j + 1$.

For any k $(k \le j)$, $p''_c(v_k) \le u''_c(v_k) + w'(v_k)$, $u''_c(v_k) \le u'_c(v_k)$, and $p''_c(v_k) \le p'_c(v_k)$. Then, $p''_c(v_k) \le \min\{p'_c(v_k), u'_c(v_k) + w'(v_k)\}$. Since $GA''_2(v_{j+1}) = GA'_2(v_{j+1})$, $\min_{k \in GA''_2(v_{j+1})} p''_c(v_k)$ $\le \min_{k \in GA'_2(v_{j+1})} \{p'_c(v_k), u'_c(v_k) + w'(v_k)\} \le \min_{k \in GA'_2(v_{j+1})} \{p'_c(v_k), u'_c(v_k) + w'_2(v_k)\}$. Note that $GB'_2(v_{j+1}) \subseteq GB''_2(v_{j+1})$ and $u''_c(v_k) \le u'_c(v_k), k \le j$. Hence, $\min_{k \in GB''_2(v_{j+1})} u''_c(v_k) \le$ $\min_{k \in GB'_2(v_{j+1})} u'_c(v_k)$. Therefore, $u''_c(v_{j+1}) \le u'_c(v_{j+1})$.

Similarly, we can prove that $p''_c(v_{j+1}) \leq p'_c(v_{j+1})$, which completes the proof of this lemma. $\square$

7.1.2 An algorithm to compute $p''_c(v_i), u''_c(v_i), i = 1, \ldots, n$

In this section, an $O(n \log n)$-time algorithm is presented to compute $p''_c(v_i)$ and $u''_c(v_i), i = 1, \ldots, n$.

Computing $w'(v_i), i = 1, \ldots, n$

We show that the values of $w'_1(v_i)$, $w'_2(v_i)$, and $w'(v_i), i = 1, \ldots, n$, can be computed in $O(n \log n)$ time, which is sufficient to present our $O(n \log n)$-time solution to compute all $w'_1(\cdot)$'s. A balanced binary search tree (BST, for short) is constructed and dynamically maintained during a sweep approach from v_1 to v_n. The data structure is a list of visited vertices sorted in their remaining coverage radii with respect to the current vertex (the remaining coverage radius of v_j with respect to v_i ($j < i$) is $r(v_j) - d(v_i, v_j)$). At each step of the forward approach, the vertices in the data structure that cannot cover current vertex v_i are removed. Each node in the BST stores the minimum open-facility cost of vertices that are descendant of the node. We have the following properties of the balanced BST.

- An update operation, such as deleting a leaf vertex and inserting a leaf vertex, can be performed in $O(\log n)$ time.
- The $w'_1(\cdot)$ value of the current vertex is the minimum open-facility cost of vertices in the current data structure, which is stored in the root of BST.

The algorithm to compute $w'_1(v_i), i = 1, \ldots, n$, is described as follows. We denote by Π_b the balanced BST.

1. Initially, Π only contains v_1.
2. For each $i, i = 2, \ldots, n$,
    (a) remove all vertices that cannot cover v_i from Π_b;
    (b) let $w'_1(v_i)$ be the minimum open-facility cost of vertices in Π_b; and
    (c) insert v_i into Π_b.

Clearly, each vertex in $V(G)$ is inserted into and deleted from the BST at most once. Therefore, the values of $w'_1(v_i), i = 1, \ldots, n$, can be computed in a total of $O(n \log n)$ time.

Computing $p''_c(v_i)$ and $u''_c(v_i), i = 1, \ldots, n$

In order to compute $p''_c(v_i)$ and $u''_c(v_i)$, we create a dynamic priority search tree data structure [51] to answer range queries such as $GA''_1(v_i), GB''_1(v_i), GA''_2(v_i)$, and $GB''_2(v_i), i = 1, \ldots, n$. Actually, a balanced binary search tree structure is enough to dynamically maintain $GB''_2(v_i)$, since there is only one piece of information (i.e., the relative positions on the path G) in the definition of $GB''_2(v_i)$. However, there are two pieces of information, including the relative positions on the path and the corresponding $g(\cdot)$-values, in the definitions of $GA''_1(v_i), GB''_1(v_i)$, and $GA''_2(v_i)$. To simplify the description of our algorithm, we use the dynamic priority search tree to answer range queries $GB''_2(v_i), i = 1, \ldots, n$.

Note that $p''_c(v_1) = c(v_1) + w'(v_1)$ and $u''_c(v_1) = c(v_1)$.

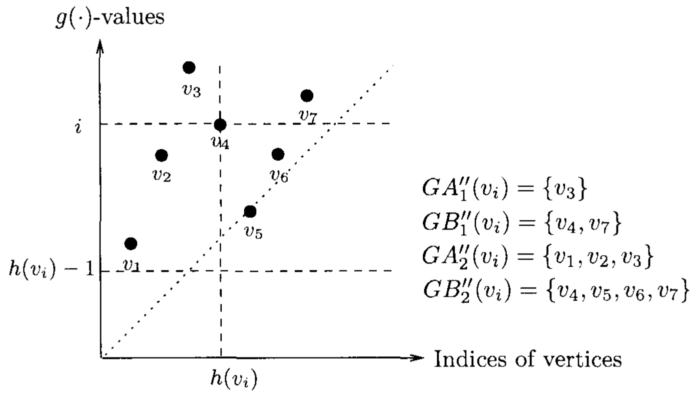


Figure 7.4: Two-dimensional diagram, all points lie above or on the dotted line since $g(v_j) \geq j, 1 \leq j \leq n$.

We use a two-dimensional diagram to depict important information about vertices in $V(G)$ (see Figure 7.4). For each $v_i, 1 \leq i \leq n$, the horizontal coordinate of v_i in the two-dimensional diagram corresponds to indices of vertices, and the vertical coordinate of v_k corresponds to the $g(v_i)$-value. A priority search tree (PST, for short) data structure [51] is constructed and dynamically maintained during a sweep approach from v_1 to v_n. Each node v in the PST maintains the following two values: $\mu_p(v) = \min_{u \in V'} p''_c(u)$ and $\mu_u(v) = \min_{u \in V'} u''_c(u)$ where V' is the set of vertices contained in the subtree rooted at node v in the PST. We have the following properties of the PST.

- Inserting a vertex u with the values of $p''_c(u)$ and $u''_c(u)$ can be performed in logarithmic time.

- For any two-sided range query, all vertices within it are contained in logarithmic subtrees.

The algorithm to compute $p''_c(v_i)$ and $u''_c(v_i), i = 2, \ldots, n$, is described as follows. We denote by Π_p the dynamic PST.

1. Initially, Π_p only contains v_1.

2. For each $i, i = 2, \ldots, n$,

   (a) find the $\log |\Pi_p|$ subtrees that contain all vertices in $GA''_1(v_i), GB''_1(v_i), GA''_2(v_i)$, and $GB''_2(v_i)$ (they are two-sided range queries in the two-dimensional diagram), respectively;

   (b) let $u''_{c(1)}(v_i)$ be the minimum value of $\mu_p(\cdot)$ stored at the roots of $O(\log |\Pi_p|)$ subtrees for $GA''_2(v_i)$ and $u''_{c(2)}(v_i)$ be the minimum value of $\mu_u(\cdot)$ stored at the roots of $O(\log |\Pi_p|)$ subtrees for $GB''_2(v_i)$. Then, $u''_c(v_i) = \min\{u''_{c(1)}(v_i), u''_{c(2)}(v_i)\} + c(v_i)$;

   (c) compute $p''_c(v_i)$ from $u''_c(v_i)$ and the values stored at the roots of $O(\log |\Pi_p|)$ subtrees for $GA''_1(v_i)$ and $GB''_1(v_i)$; and

   (d) insert v_i with the values of $p''_c(v_i)$ and $u''_c(v_i)$ into Π_p.

Clearly, each minimal query such as $u''_c(v_i)$ or $p''_c(v_i)$, can be answered in $O(\log n)$ time. Therefore, $O(n \log n)$ time is required to compute the values of $u''_c(v_i), p''_c(v_i), i = 1, \ldots, n$.

Computing an optimal solution

We already know that the optimal cost is equal to $\min_{1 \leq i \leq n : g(v_i) = n} p''_c(v_i)$. Thus, the optimal cost is computable in $O(n \log n)$ time. To locate facilities opened in an optimal solution, we only need to remember the best previous critical facility and non-critical facility (if needed) for the current critical facility v_i during the step to compute $u''_c(v_i), p''_c(v_i), i = 2, \ldots, n$. Obviously, it takes extra $O(n)$ time and $O(n)$ space to obtain such information for each $v_i, 2 \leq i \leq n$.

Therefore, the constrained covering problem on a path network can be solved in $O(n \log n)$ time.

Theorem 7.1.3 *The constrained covering problem on a path network can be solved in $O(n \log n)$ time.*

7.2 Extended-star networks

An extended-star is a network in which three or more path networks are connected by a single root vertex, which we label v_n. In this section, two algorithms for the CCP on an extended star network G are presented. One runs in $O(n\tau \log n)$ time (τ is the number branches in G) and the other runs in $O(n^{1.5} \log n)$ time.

Let $\chi_1, \dots \chi_\tau$ be the branches of the root vertex v_n where each branch χ_i contains the set of vertices on the path from a leaf vertex $v_{1(i)}$ to vertex $v_{n_i(i)}$ (n_i is the number of vertices on the branch χ_i). Clearly, $v_{n_i(i)}$ is adjacent to v_n, $i = 1, \dots, \tau$.

For each vertex $v \in V(G)$, define the *external coverage* of v as $ext(v) = r(v) - d(v, v_n)$. This measure represents the amount of coverage that vertex v can provide to branches not containing v.

The following property, which is described in [45], is easy to obtain.

Observation 7.2.1 *In any optimal solution, there exists a vertex $v \in V(G)$ such that $d(v, v_n) \le r(v)$ and no facility is located at vertex $v' \in V(G)$ with $ext(v') > ext(v)$.*

A vertex that satisfies these conditions is referred as the *external covering vertex.* We have the following cases regarding the external covering vertex.

- *Case 1*: v_n is the external covering vertex. In this case, vertices in a branch χ_i are covered either by v_n or by facilities in $\chi_i, 1 \le i \le \tau$.

- *Case 2*: $v \ne v_n$ is the external covering vertex (clearly, $ext(v) \ge 0$). Assume that v is on a branch $\chi_i, 1 \le i \le \tau$. In this case, vertices in χ_i (possibly except v) are covered by facilities in χ_i, and vertices in branch $\chi_j, 1 \le j \ne i \le \tau$ are covered either by v or by facilities in χ_j.

Our strategy is to consider each vertex $v \in V(G)$ as an external covering vertex and then compute the optimal cost under this constraint, denoted by $\eta^*(v)$.

Note that some value of $\eta^*(v)$ ($v \in V(G)$) is undefined if $ext(v) < 0$, in which case, we let the value $\eta^*(v)$ be ∞. It is trivial that the optimal cost is $\min_{v \in V(G)} \eta^*(v)$.

7.2.1 Computing $\eta^*(v)$, $v \in V(G)$

In this section, we first introduce a pseudo vertex for each branch and show that the protected and unprotected costs of the pseudo vertices can be computed efficiently. Then, we present two methods to efficiently compute costs $\eta^*(v), v \in V(G)$.

Pseudo vertices We observe that the cost of each branch is only affected by the external covering vertex v if the branch does not contain v, and hence a pseudo vertex $v_{n_i^+(i)}$ is introduced for each branch χ_i to simulate an external covering vertex located outside $\chi_i, 1 \leq i \leq \tau$. The pseudo vertex $v_{n_i^+(i)}$ is appended after vertex $v_{n_i(i)}$ (Recall that $v_{n_i(i)}$ is the vertex of the branch χ_i that is adjacent to v_n).

For each $i, i = 1, \ldots, \tau$, the open-facility cost $c(v_{n_i^+(i)})$ of $v_{n_i^+(i)}$ is set to be zero, and the coverage radius of $v_{n_i^+(i)}$ and the distance $d(v_{n_i(i)}, v_{n_i^+(i)})$ are not fixed. In the following, we construct a new PST for branch χ_i such that, using the PST, the values of $u''_c(v_{n_i^+(i)})$ and $p''_c(v_{n_i^+(i)})$ can be computed in $O(\log n_i)$ time for any given values of the coverage radius $r(v_{n_i^+(i)})$ of $v_{n_i^+(i)}$ and $d(v_{n_i(i)}, v_{n_i^+(i)})$.

Computing $u''_c(v_{n_i^+(i)})$ and $p''_c(v_{n_i^+(i)}), 1 \leq i \leq \tau$ For all $j, 1 \leq j \leq n_i$, we can compute the $p''_c(\cdot)$ and $u''_c(\cdot)$ values of vertices $v_{1(i)}, \ldots, v_{n_i(i)}$ in $O(n_i \log n_i)$ time, using the dynamic PST described in Section 7.1. However, we cannot directly use the dynamic PST to compute $u''_c(v_{n_i^+(i)})$ and $p''_c(v_{n_i^+(i)})$ since the upper reach values of vertices $(v_{1(i)}, \ldots, v_{n_i(i)})$ on χ_i might be incorrect without knowing the position of $v_{n_i^+(i)}$. In the following, we show a simple method to fix the problem.

For a vertex $v_{j(i)}(1 \leq j \leq n_i^+ = n_i + 1)$, we define its *upper cover* $g'_i(v_{j(i)})$ and *lower cover* $h'_i(v_{j(i)})$ as follows.

$$g'_i(v_{j(i)}) = d(v_{1(i)}, v_{j(i)}) + r(v_{j(i)});$$

$$h'_i(v_{j(i)}) = \max\{0, d(v_{1(i)}, v_{j(i)}) - r(v_{j(i)})\}.$$

Note the tight relation between the upper (resp. lower) cover and the upper (resp. lower) reach of a vertex. Also, the $g'_i(\cdot)$ and $h'_i(\cdot)$ values of a vertex are not affected by the position of $v_{n_i^+(i)}$. We redefine $GA''_1(v_{j(i)}), GB''_1(v_{j(i)}), GA''_2(v_{j(i)})$, and $GB''_2(v_{j(i)})$ by upper cover and lower cover as follows, $1 \leq j \leq n_i^+$. Let $b_j = h(v_{j(i)}) - 1$ where $h(v_{j(i)})$ is the lower reach of $v_{j(i)}$ on the path χ_i.

$$GA''_1(v_{j(i)}) = \{k : d(v_{1(i)}, v_{k(i)}) < h'_i(v_{j(i)}) \leq d(v_{1(i)}, v_{j(i)}) \leq g'_i(v_{k(i)})\}$$

$$GB''_1(v_{j(i)}) = \{k : h'_i(v_{j(i)}) \leq d(v_{1(i)}, v_{k(i)}) < d(v_{1(i)}, v_{j(i)}) \leq g'_i(v_{k(i)})\}$$

$$GA''_2(v_{j(i)}) = \{k : d(v_{1(i)}, v_{k(i)}) \leq d(v_{1(i)}, v_{b_j(i)}) \leq g'_i(v_{k(i)})\}$$

$$GB''_2(v_{j(i)}) = \{k : h'_i(v_{j(i)}) \leq d(v_{1(i)}, v_{k(i)}) < d(v_{1(i)}, v_{j(i)})\}$$

That is, we can determine whether a vertex $v_{k(i)}$ is in $GA''_1(v_{j(i)}), GB''_1(v_{j(i)}), GA''_2(v_{j(i)}),$

or $GB_2''(v_{j(i)})$ $(k < j)$, if we know information about $v_{k(i)}$, such as

1. the position of $v_{k(i)}$ (i.e., $d(v_{1(i)}, v_{k(i)})$) and
2. the upper cover $g_i'(v_{k(i)})$ of $v_{k(i)}$,

and information about $v_{j(i)}$, such as

1. the position of $v_{j(i)}$ (i.e., $d(v_{1(i)}, v_{j(i)})$),
2. the lower cover $h_i'(v_{j(i)})$ of $v_{j(i)}$, and
3. the lower reach $h(v_{j(i)})$ of $v_{j(i)}$.

Recall that in Section 7.1, to compute the $p_c''(\cdot)$ and $u_c''(\cdot)$ values of vertices on a path, a priority search tree structure is constructed and dynamically maintained during the sweep approach from one end of the path to the other. Each vertex is embedded in a two-dimensional diagram where the horizontal coordinates correspond to indices of vertices and the vertical coordinates correspond to the $g(\cdot)$-values.

Here we put the vertices, $v_{1(i)}, \ldots, v_{n_i(i)}$, in a two-dimensional diagram where the horizontal coordinates correspond to locations of vertices instead of indices and the vertical coordinates correspond to the $g_i'(\cdot)$-values instead of the $g(\cdot)$-values. A new PST data structure [51] is then constructed over the set of points in the two-dimensional diagram. Similar to the dynamic PST in Section 7.1, each node w in the new PST maintains two values: $\mu_p(w) = \min_{u \in V'} p_c''(u)$ and $\mu_u(w) = \min_{u \in V'} u_c''(u)$ where V' is the set of vertices contained in the subtree of the new PST rooted at node w.

It is easy to see that for any two-sided range query in the two-dimensional diagram, all vertices within it are contained in logarithmic subtrees of the new PST.

For the vertex $v_{n_i^+(i)}$, we can obtain its position, lower cover, and lower reach from given values of $d(v_{n_i(i)}, v_{n_i^+(i)})$ and $r(v_{n_i^+(i)})$. Its position and lower cover can be computed in $O(1)$ time and its lower reach can be computed in $O(\log n_i)$ time [45]. Therefore, using the new PST structure, we can compute $u_c''(v_{n_i^+(i)})$ and $p_c''(v_{n_i^+(i)})$ in time $O(\log n_i)$ for any given values of $d(v_{n_i(i)}, v_{n_i^+(i)})$ and $r(v_{n_i^+(i)})$.

Lemma 7.2.2 *For a branch $\chi_i, 1 \leq i \leq \tau$, after $O(n_i \log n_i)$ preprocessing time, we can compute $u_c''(v_{n_i^+(i)})$ and $p_c''(v_{n_i^+(i)})$ in time $O(\log n_i)$ for any given values of $d(v_{n_i(i)}, v_{n_i^+(i)})$ and $r(v_{n_i^+(i)})$.*

An $O(n\tau \log n)$ approach

To meet the constraint of the CCP, the external covering vertex itself must be covered by some other facility.

We consider the two cases described above.

Case 1 In this case, v_n is the external covering vertex. Only one branch, say $\chi_i (1 \leq i \leq \tau)$, is needed to contribute one facility to cover v_n. The cost of branch χ_i is then equal to $p''_c(v_{n_i^+(i)})$ with $r(v_{n_i^+(i)}) = r(v_n)$ and with $d(v_{n_i(i)}, v_{n_i^+(i)}) = d(v_{n_i(i)}, v_n)$. The cost of any other branch $\chi_j (1 \leq j \neq i \leq \tau)$ is equal to $u''_c(v_{n_j^+(j)})$ with $r(v_{n_j^+(j)}) = r(v_n)$ and with $d(v_{n_j(j)}, v_{n_j^+(j)}) = d(v_{n_j(j)}, v_n)$. The total cost will be

$$\begin{aligned} & c(v_n) + p''_c(v_{n_i^+(i)}) + \sum_{j=1,\ldots,\tau; j \neq i} u''_c(v_{n_j^+(j)}) \\ = \;& c(v_n) + p''_c(v_{n_i^+(i)}) - u''_c(v_{n_i^+(i)}) + \sum_{j=1,\ldots,\tau} u''_c(v_{n_j^+(j)}). \end{aligned}$$

Therefore, the distinguished branch χ_i will be the branch that has the minimum value of $p''_c(v_{n_i^+(i)}) - u''_c(v_{n_i^+(i)})$. Since each cost such as $p''_c(v_{n_j^+(j)})$ and $u''_c(v_{n_j^+(j)}) (1 \leq j \leq \tau)$ can be computed in $O(\log n_j)$ (Lemma 7.2.2), we can compute $\eta^*(v_n)$ in $O(\tau \log n)$ time.

Case 2 Assume that the external covering vertex, say $v_{j(i)}$, is located in branch $\chi_i, 1 \leq i \leq \tau, 1 \leq j \leq n_i$. Note that if $ext(v_{j(i)}) < 0$ then $\eta^*(v_{j(i)}) = \infty$. We assume that $ext(v_{j(i)}) \geq 0$ and consider the following three cases.

- *Case 2.1*: $v_{j(i)}$ is covered by v_n (if $r(v_n) < d(v_{j(i)}, v_n)$ then Case 2.1 does not exist);
- *Case 2.2*: $v_{j(i)}$ is covered by some facility in χ_i;
- *Case 2.3*: $v_{j(i)}$ is covered by some facility in $\chi_k (1 \leq k \neq i \leq \tau)$.

Case 2.1: In this case, the cost of χ_i is $u''_c(v_{j(i)})$ and the cost of $\chi_t, 1 \leq t \neq i \leq \tau$, is $u''_c(v_{n_t^+(t)})$ with $r(v_{n_t^+(t)}) = r(v_{j(i)})$ and with $d(v_{n_t(t)}, v_{n_t^+(t)}) = d(v_{n_t(t)}, v_{j(i)})$. Thus, the total cost in Case 2.1 is

$$c(v_n) + u''_c(v_{j(i)}) + \sum_{t=1,\ldots,\tau; t \neq i} u''_c(v_{n_t^+(t)}).$$

Case 2.2: In this case, the cost of χ_i is $p''_c(v_{j(i)})$ and the cost of $\chi_t, 1 \leq t \neq i \leq \tau$, is $u''_c(v_{n_t^+(t)})$ with $r(v_{n_t^+(t)}) = r(v_{j(i)})$ and with $d(v_{n_t(t)}, v_{n_t^+(t)}) = d(v_{n_t(t)}, v_{j(i)})$. The total cost in Case 2.2 is

$$p''_c(v_{j(i)}) + \sum_{t=1,\ldots,\tau;t\neq i} u''_c(v_{n_t^+(t)}).$$

Case 2.3: In this case, the cost of χ_i is $u''_c(v_{j(i)})$. Suppose that branch $\chi_k(1 \leq k \neq i \leq \tau)$ contributes a facility to cover $v_{j(i)}$. Then, the cost of χ_k is $p''_c(v_{n_k^+(k)})$ with $r(v_{n_k^+(k)}) = r(v_{j(i)})$ and with $d(v_{n_k(k)}, v_{n_k^+(k)}) = d(v_{n_k(k)}, v_{j(i)})$. For each branch $\chi_t, 1 \leq t \leq \tau$ and $t \neq i, k$, its cost is $u''_c(v_{n_t^+(t)})$ with $r(v_{n_t^+(t)}) = r(v_{j(i)})$ and with $d(v_{n_t(t)}, v_{n_t^+(t)}) = d(v_{n_t(t)}, v_{j(i)})$. The total cost in Case 2.3 is

$$\begin{aligned} & u''_c(v_{j(i)}) + p''_c(v_{n_k^+(k)}) + \sum_{t=1,\ldots,\tau;t\neq i,k} u''_c(v_{n_t^+(t)}) \\ = \; & u''_c(v_{j(i)}) + p''_c(v_{n_k^+(k)}) - u''_c(v_{n_k^+(k)}) + \sum_{t=1,\ldots,\tau;t\neq i} u''_c(v_{n_t^+(t)}). \end{aligned}$$

Therefore, the distinguished branch χ_k will be the branch (among $\chi_1, \ldots, \chi_{i-1}, \chi_{i+1}, \ldots, \chi_\tau$) that has the minimum value of $p''_c(v_{n_k^+(k)}) - u''_c(v_{n_k^+(k)})$.

From the above discussion, the cost of $\eta^*(v_{j(i)})$ can be computed in $O(\tau \log n)$ since each protected/unprotected cost can be computed in $O(\log n_j)$ (Lemma 7.2.2). Therefore, in $O(n\tau \log n)$ time, we can compute the values of $\eta^*(v), v \in V(G)$.

Theorem 7.2.3 *The constrained covering problem on an extended-star network can be solved in $O(n\tau \log n)$ time, where τ is the number of branches in the extended-star network.*

An $O(n^{1.5} \log n)$ approach

We call the above $O(n\tau \log n)$ approach the *first approach* for CCP on an extended-star network. Note that, in the worst case, τ might be $O(n)$. Next, we present another approach which runs in $O(n^{1.5} \log n)$ time.

We separate the set of branches in G into two classes according to their sizes. If one branch contains at least $\sqrt{n}$ vertices, we call it a *big branch*, and the branch is called a *small branch*, otherwise. Obviously, the number of big branches in G is no more than $\sqrt{n}$. Using the PST constructed for each branch in the first approach, we can compute the (protected and unprotected) costs of these big branches in $O(\sqrt{n} \log n)$ time for a given external covering vertex.

In the following, we create a data structure to merge the information contained in all small branches, which can answer a total cost (of these small branches) query in amortized $O(\sqrt{n}\log n)$ time, for any given external covering vertex.

We only consider Case 2 in which the external covering vertex $v_{j(i)}(1 \leq j \leq n_i)$ is located in branch $\chi_i(1 \leq i \leq \tau)$.

For each branch $\chi_k(1 \leq k \neq i \leq \tau)$, we need to compute the values of $p''_c(v_{n_k^+(k)}), u''_c(v_{n_k^+(k)})$ with $r(v_{n_k^+(k)}) = r(v_{j(i)})$ and with $d(v_{n_k(k)}, v_{n_k^+(k)}) = d(v_{n_k(k)}, v_{j(i)})$. In the first approach, we embed all vertices $(v_{1(k)}, \dots, v_{n_k(k)})$ of a branch $\chi_k(1 \leq k \neq i \leq \tau)$ in a two-dimensional diagram (denoted by $\mathcal{H}$, where the horizontal coordinates correspond to locations of vertices and the vertical coordinates correspond to $g'_k(\cdot)$-values of vertices) and then construct a PST to compute $p''_c(v_{n_k^+(k)}), u''_c(v_{n_k^+(k)})$ in $O(\log n_k)$ time for any given values of $r(v_{n_k^+(k)})$ and $d(v_{n_k(k)}, v_{n_k^+(k)})$.

Here we first show that there are at most $n_k(n_k+1)$ possible different pairs of values of $p''_c(v_{n_k^+(k)})$ and $u''_c(v_{n_k^+(k)})$ as follows.

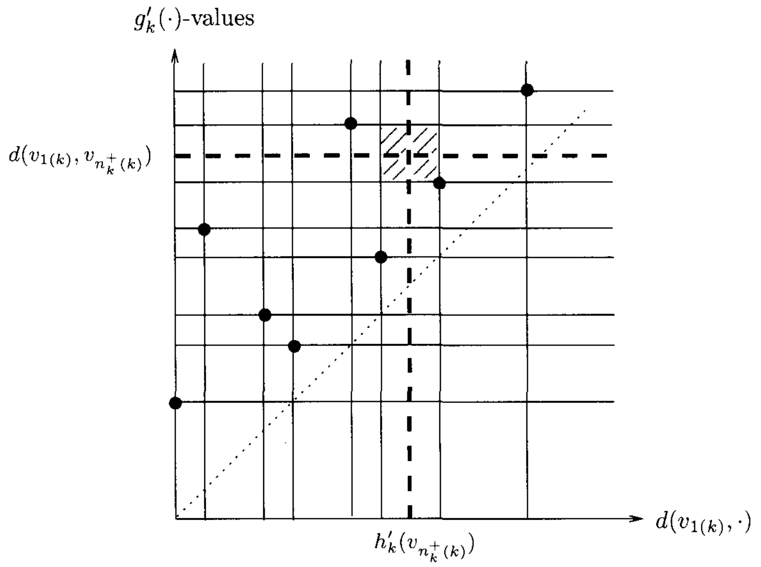


Figure 7.5: The two-dimensional diagram for branch χ_k, all points lie above or on the dotted line since $g'_i(v_{t(k)}) \geq d(v_{1(k)}, v_{t(k)}), 1 \leq t \leq n_k$.

As demonstrated in Figure 7.5, we draw horizontal and vertical lines through n_k points in the two-dimensional diagram $\mathcal{H}$. These lines partition $\mathcal{H}$ into at most $n_k(n_k+1)$ rectangular cells and we assume that each cell is only associated with its upper boundary and right boundary.

We know that, for given values of $r(v_{n_k^+(k)})$ and $d(v_{n_k(k)}, v_{n_k^+(k)})$,

- $GA_1''(v_{n_k^+(k)})$ contains all vertices whose horizontal coordinate is less than $h_k'(v_{n_k^+(k)}) = \max\{0, d(v_{1(k)}, v_{n_k^+(k)}) - r(v_{n_k^+(k)})\}$ and whose vertical coordinate is $\geq d(v_{1(k)}, v_{n_k^+(k)})$,
- $GB_1''(v_{n_k^+(k)})$ contains all vertices whose horizontal coordinate is at least $h_k'(v_{n_k^+(k)})$ and whose vertical coordinate is at least $d(v_{1(k)}, v_{n_k^+(k)})$,
- $GA_2''(v_{n_k^+(k)})$ contains all vertices whose horizontal coordinate is less than $h_k'(v_{n_k^+(k)})$ and whose vertical coordinate is at least $d(v_{1(k)}, v_{b(k)})$ (where $v_{b+1(k)}$ is covered by $v_{n_k^+(k)}$ and $v_{b(k)}$ cannot be covered by $v_{n_k^+(k)}$), and
- $GB_2''(v_{n_k^+(k)})$ contains all vertices whose horizontal coordinate is at least $h_k'(v_{n_k^+(k)})$.

We can see that $GA_1''(v_{n_k^+(k)}), GB_1''(v_{n_k^+(k)}), GA_2''(v_{n_k^+(k)})$, and $GB_2''(v_{n_k^+(k)})$ stay unchanged when $(h_k'(v_{n_k^+(k)}), d(v_{1(k)}, v_{n_k^+(k)}))$ lies in a cell since b is fixed for any point $(h_k'(v_{n_k^+(k)}), d(v_{1(k)}, v_{n_k^+(k)}))$ in a cell. Therefore, for any given values of $r(v_{n_k^+(k)})$ and $d(v_{n_k(k)}, v_{n_k^+(k)})$ such that $(h_k'(v_{n_k^+(k)}), d(v_{1(k)}, v_{n_k^+(k)}))$ lies in a cell, their $p_c''(v_{n_k^+(k)})$-values (resp. $u_c''(v_{n_k^+(k)})$-values) are equal, which implies that there are at most $n_k(n_k+1)$ possible different pairs of values of $p_c''(v_{n_k^+(k)}), u_c''(v_{n_k^+(k)})$.

In our approach, we pre-compute all these possible values of $p_c''(v_{n_k^+(k)}), u_c''(v_{n_k^+(k)})$ for each small branch $\chi_k, 1 \leq k \leq \tau$. Each pair of $p_c''(v_{n_k^+(k)}), u_c''(v_{n_k^+(k)})$ values represents a range of values of $h_k'(v_{n_k^+(k)})$ and a range of values of $d(v_{n_k(k)}, v_{n_k^+(k)})$.

Let v be the external covering vertex. Recall that the pseudo vertex $v_{n_k^+(k)}$ is introduced for branch χ_k to simulate an external covering vertex outside χ_k. To merge the information contained in all small branches, we use distance $d(v_{n_k(k)}, v_{n_k^+(k)}) - d(v_{n_k(k)}, v_n)$ instead of $d(v_{n_k(k)}, v_{n_k^+(k)})$ for each small branch $\chi_k, 1 \leq k \leq \tau$ since when v is given, $d(v_{n_k(k)}, v_{n_k^+(k)}) - d(v_{n_k(k)}, v_n) = d(v, v_n)$ for any $k, 1 \leq k \leq \tau$ and χ_k does not contain v. Furthermore, if χ_k does not contain v then $h_k'(v_{n_k^+(k)}) = \max\{0, d(v_{1(k)}, v_{n_k(k)}) - ext(v)\}$. Note that $d(v_{1(k)}, v_{n_k(k)})$ is known for each branch χ_k.

Hence, we are able to say that for each small branch $\chi_k, 1 \leq k \leq \tau$ (v is not on χ_k), each pair of $p_c''(v), u_c''(v)$ values represents a range of values of $ext(v)$ and a range of values of $d(v, v_n)$, and each range of values of $ext(v)$ corresponds to at most $n_k + 1$ pairs of $p_c''(v), u_c''(v)$ values. It is easy to see that it costs $O(n_k^2 \log n_k)$ time to compute all these

$n_k(n_k+1)$ possible different pairs of $p''_c(v)$ & $u''_c(v)$ values and their corresponding ranges of $ext(v)$ & $d(v, v_n)$, $1 \le k \le \tau$.

The following lemma (Lemma 7.2.4) shows that we are able to compute such values for all small branches in $O(n^{1.5}\log n)$ time.

Lemma 7.2.4 *Given a set of positive numbers $\Delta = \{a_1, \ldots, a_q\}$ where each number is no more than $\sqrt{n}$ and $\sum_{i=1}^{q} a_i = n$, let $\mathcal{W}(\Delta) = \sum_{i=1}^{q} a_i^2$. Then $\mathcal{W}(\Delta) \le n^{1.5}$.*

Proof Obviously, $q \ge \sqrt{n}$. We prove the lemma by induction. When $q = \sqrt{n}$, the lemma is trivially true. Assume that the lemma is true for any q, $\sqrt{n} \le q \le k$. Next, we prove that the lemma is true for $q = k+1$.

If there are two numbers in Δ, say a_i and a_j $(i < j)$, such that $a_i + a_j \le \sqrt{n}$ then the new set of positive numbers $\Delta' = \{a_1, \ldots, a_{i-1}, a_{i+1}, \ldots, a_{j-1}, a_{j+1}, \ldots a_q, a_i + a_j\}$ has a larger value of $\mathcal{W}(\Delta')(> \mathcal{W}(\Delta))$. Without loss of any generality, we assume that for any two numbers a_i and a_j in Δ, their sum is larger than $\sqrt{n}$.

If there is a number in Δ, say a_i, equal to $\sqrt{n}$, then $\mathcal{W}(\Delta) = \sum_{i=1}^{q} a_i^2 \le (n-\sqrt{n})^{1.5} + a_i^2 < n^{1.5}$. Hence, we assume that any number in Δ is smaller than $\sqrt{n}$.

Without loss of any generality, assume that a_1 is no less than any other number in Δ. Then,

$$\begin{aligned}
\mathcal{W}(\Delta) &= \sum_{i=1}^{q} a_i^2 \\
&= a_1^2 + [(\sqrt{n}-a_1) + (a_1+a_2-\sqrt{n})]^2 + \sum_{i=3}^{q} a_i^2 \\
&= a_1^2 + (\sqrt{n}-a_1)^2 + 2(\sqrt{n}-a_1)(a_1+a_2-\sqrt{n}) + (a_1+a_2-\sqrt{n})^2 + \sum_{i=3}^{q} a_i^2 \\
&\le a_1^2 + 2(\sqrt{n}-a_1)(a_1+a_2-\sqrt{n}) + (\sqrt{n}-a_1)^2 + (n-\sqrt{n})^{1.5} \text{ (by assumption)} \\
&\le (a_1+\sqrt{n}-a_1)^2 + (n-\sqrt{n})^{1.5} \text{ (since } a_2 \le a_1 < \sqrt{n}) \\
&\le n^{1.5}.
\end{aligned}$$

It completes the proof of Lemma 7.2.4. □

We denote by Γ_1 the set of indices of big branches and by Γ_2 the set of indices of small branches.

In our algorithm, the optimal costs $\eta^*(v)$ of external covering vertex candidates v ($v \neq v_n$ and $ext(v) \geq 0$) are queried in a non-decreasing order of their external coverage $ext(v)$. Let v be the current external covering vertex and χ_i be the branch containing it. The computation of $\eta^*(v)$ is described as follows.

Step 1: Get the values of $\mu_1 = p_c''(v)$ and $\mu_2 = u_c''(v)$ on branch χ_i (both of them are already computed), and compute the value of $\mu_3 = u_c''(v_{n_i^+(i)})$ with $ext(v_{n_i^+(i)}) = ext(v)$ and with $d(v_{n_i^+(i)}, v_n) = d(v, v_n)$ (it can be computed in $\log n_i$ time).

Step 2: For all big branches $\chi_t, t \in \Gamma_1 \setminus \{i\}$, compute the values of $u_c''(v_{n_t^+(t)})$ and $p_c''(v_{n_t^+(t)})$, using the PST described in the first approach. Let $\mu_4 = \sum_{t\in\Gamma_1\setminus\{i\}} u_c''(v_{n_t^+(t)})$. Find the branch among them with minimum value μ_5 of $p_c''(v_{n_t^+(t)}) - u_c''(v_{n_t^+(t)})$.

Step 3: Do the following for small branches:

- If v is the first external covering vertex visited, a segment tree structure is constructed to maintain the possible values of $u_c''(\cdot)$ and $p_c''(\cdot)$ from all small branches $\chi_k, 1 \leq k \leq \tau$ with $ext(v)$, which is described later. Search the structure with the value of $d(v, v_n)$ to obtain the value of $\mu_6 = \sum_{k\in\Gamma_2} u_c''(v_{n_k^+(k)})$ and the branch with minimum value μ_7 of $p_c''(v_{n_k^+(k)}) - u_c''(v_{n_k^+(k)})$.
- Otherwise, update the segment tree structure such that for each pair of $u_c''(\cdot)$ and $p_c''(\cdot)$ values, the range of external coverage associated with the pair contains $ext(v)$. Search the structure with the value of $d(v, v_n)$ to obtain the value of $\mu_6 = \sum_{k\in\Gamma_2} u_c''(v_{n_k^+(k)})$ and the branch with minimum value μ_7 of $p_c''(v_{n_k^+(k)}) - u_c''(v_{n_k^+(k)})$.

Step 4: $\eta^*(v)$ is the minimum value of the following three values:

- The value of $\eta^*(v)$ in *Case 2.1* is $c(v_n) + \mu_2 + \mu_4 + \mu_6 - \mu_3$ if χ_i is a small branch, and $c(v_n) + \mu_2 + \mu_4 + \mu_6$, otherwise.
- The value of $\eta^*(v)$ in *Case 2.2* is $\mu_1 + \mu_4 + \mu_6 - \mu_3$ if χ_i is a small branch, and $\mu_1 + \mu_4 + \mu_6$, otherwise.
- The value of $\eta^*(v)$ in *Case 2.3* is $\mu_2 + \min\{\mu_5, \mu_7\} + \mu_4 + \mu_6 - \mu_3$ if χ_i is a small branch, and $\mu_2 + \min\{\mu_5, \mu_7\} + \mu_4 + \mu_6$, otherwise.

A segment tree structure for small branches For a given value of $ext(v)$, there are at most $n_k + 1$ possible pairs of values of $u''_c(\cdot)$ and $p''_c(\cdot)$ from a small branch $\chi_k, 1 \leq k \leq \tau$, and therefore at most n pairs in total. Each pair of $u''_c(\cdot)$ and $p''_c(\cdot)$ values is associated with an interval (a range of $d(\cdot, v_n)$) and two pairs from the same branch are associated with disjoint intervals (note that each pair is associated with a range of external coverage and distance to v_n of an external covering vertex). We create a segment tree [10] to maintain the set of such pairs for current v.

We define a set of coordinates by the endpoints of the intervals. Any two adjacent coordinates build an elementary interval. Every leaf corresponds to an elementary interval. Inner nodes correspond to the union of the subtree intervals of the node. Each node u contains a list that contain all pairs of $u''_c(\cdot)$ and $p''_c(\cdot)$ values, such that the interval associated with each pair contains the interval of node u but not the interval of the parent node of u. At each node, we maintain the sum of $u''_c(\cdot)$ values and the minimum value of $p''_c(\cdot) - u''_c(\cdot)$. Using this segment tree, we can answer a query in step 3 in $O(\log n)$ time, and insert or delete a pair of $u''_c(\cdot)$ and $p''_c(\cdot)$ values in $O(\log n)$ time.

When the value of $ext(v)$ is moving forward, the set of pairs of $u''_c(\cdot)$ and $p''_c(\cdot)$ values from some branch (say χ_k) might change, in which case, we need to delete all old $n_k + 1$ pairs and insert new $n_k + 1$ pairs from χ_k into the structure. Since each pair is inserted and deleted at most once, the total number of inserting and deleting operations is $O(n^{1.5})$ (Lemma 7.2.4).

Therefore, it costs $O(n^{1.5} \log n)$ time to complete step 3 for all possible external covering vertices.

Since step 2 can be done in time $O(\sqrt{n} \log n)$ for each external covering vertex, we have the following theorem.

Theorem 7.2.5 *The constrained covering problem on an extended-star network can be solved in $O(n^{1.5} \log n)$ time.*

7.3 Tree networks

Our algorithm for the CCP on a tree network $T = (V(T), E(T))$ is based on the dynamic programming technique, which is similar to the method of Horne and Smith [44]. The main difference is in the definition of a coverage matrix (refer to [44] for their definition of a coverage matrix).

A vertex is arbitrarily chosen to be the root vertex of the tree, denoted by r_T. Let T_v denote the subtree rooted at v and $V(T_v)$ be the vertex set of T_v. For a vertex u in $V(T_v)$, we define its *external coverage with respect to* v to be $r(u) - d(u, v)$, denoted by $ext_v(u)$. For a vertex u' lying outside T_v ($u' \in V(T) \setminus V(T_v)$), we define its *internal coverage with respect to* v to be $r(u') - d(u', v)$, denoted by $int_v(u')$.

In a solution to the CCP on T, we call a facility $u \in V(T_v)$ an *external covering vertex with respect to* v if no facility is located at a vertex $u' \in V(T_v)$ with $ext_v(u') > ext_v(u)$. Similarly, a vertex u, which satisfies that $u \in V(T) \setminus V(T_v)$ and that no facility is located at a vertex $u' \in V(T) \setminus V(T_v)$ with $int_v(u') > int_v(u)$, is called an *internal covering vertex with respect to* v.

Coverage matrix A *coverage matrix* $M(v)$ is constructed for a rooted subtree $T_v, v \in V(T)$.

The rows of $M(v)$ represent external coverage provided by possible external covering vertices in T_v. Only one distinguished row is used to represent the case where the external coverage is negative, referred to as *negative row* and labeled as '$-$'. Let $EC(v)$ be the set of external coverage in $M(v)$, i.e., $EC(v) = \{ext_v(u) : u \in V(T_v), ext_v(u) \geq 0\}$. Note that each external coverage is associated with an external covering vertex with respect to v.

The columns represent internal coverage provided by internal covering vertices in $V(T) \setminus V(T_v)$. Also, one column is used to represent the case where internal coverage are negative, referred to as *negative column* and labeled as '$-$'. Let $IC(v)$ be the set of internal coverage in $M(v)$, i.e., $IC(v) = \{int_v(u) : u \in V(T) \setminus V(T_v), int_v(u) \geq 0\}$. Similar to $EC(v)$, each element in $IC(v)$ is associated with an internal covering vertex with respect to v.

Rows (resp. columns) of a coverage matrix are sorted in a non-decreasing order of external (resp. internal) coverages. Element $M(v)[x, y]$ is the optimal cost to cover vertices in T_v, using only facilities in T_v, given that the external coverage of the external covering vertex with respect to v is x and the internal coverage of the internal covering vertex with respect to v is y. $M(v)[-, -]$ is undefined, since every vertex should be covered by some facility in a solution.

Observation 7.3.1 *[44] The optimal solution for the CCP on T is determined by the minimum cost of the negative column in the coverage matrix $M(r_T)$.*

Additionally, we add a balanced binary tree structure over each row (resp. each column) to find the minimum value of a sub-row or a sub-column efficiently.

7.3.1 An algorithm to compute coverage matrices $M(v), v \in V(T)$

In this section, a dynamic programming algorithm is presented to compute the coverage matrices in a depth-first order.

First, it is easy to see that the sets $EC(v)$ and $IC(v)$, $v \in V(T)$, can be computed and sorted in $O(n^2 \log n)$ time.

Let v be the current vertex and $v_1, \ldots, v_k$ be the immediate children of v. We assume that $M(v_1), \ldots$, and $M(v_k)$ are known.

Computing $M(v)[-,\cdot]$ We first examine the case where the external coverage provided by an external covering vertex is negative, that is, the computation of elements in the negative row of $M(v)$. The following facts are trivial in this case:

- v cannot be a facility (otherwise, the external coverage of T_v cannot be negative),
- v is covered by some facility located at a vertex in $V(T) \setminus V(T_v)$ since v can not cover itself, and
- vertices in T_{v_i} are not be covered by facilities located in T_{v_j} for any i, j with $1 \leq i, j \leq k$ and $i \neq j$.

Therefore, for any $y \in IC(v)$,

$$M(v)[-,y] = \sum_{i=1}^{k} \min_{x \in EC(v_i); x < d(v,v_i)} M(v_i)[x, y - d(v, v_i)].$$

Note that $M(v_i)[x, y - d(v, v_i)] = M(v_i)[x, -]$ if $y < d(v, v_i)$, and otherwise $y - d(v, v_i) \in IC(v_i), 1 \leq i \leq k$. Using the binary search trees built for each column of coverage matrices, it costs $O(k \log n)$ time to compute the value $M(v)[-, y]$ for any $y \in IC(v)$. Thus, the values of elements in the negative row of $M(v)$ can be computed in $O(nk \log n)$ time.

Computing $M(v)[r(v),\cdot]$ In the case when the external covering vertex of T_v is v itself, facilities in a subtree T_{v_i} do not need to provide any covering service to vertices in $T_{v_j}, j \neq i$. Furthermore, if the internal coverage is negative (i.e., $M(v)[r(v), -]$), we need one subtree T_{v_i} to contribute one facility to cover v.

$M(v)[r(v), -]$: We compute two values for each subtree $T_{v_i}, i = 1, \ldots, k$. One value for T_{v_i}, denoted by $cost_1(T_{v_i})$, is the optimal cost to cover vertices in T_{v_i}, using only facilities in T_{v_i}, given that the external coverage is no more than $r(v) + d(v, v_i)$ (to make sure that v is the external covering vertex of T_v) and the internal coverage is $r(v) - d(v, v_i)$. The other value, denoted by $cost_2(T_{v_i})$, is the optimal cost to cover vertices in T_{v_i}, using only facilities in T_{v_i}, given that the external coverage is in $[d(v, v_i), r(v) + d(v, v_i)]$ (to cover v) and the internal coverage is $r(v) - d(v, v_i)$. Obviously, $cost_2(T_{v_i}) \geq cost_1(T_{v_i}), i = 1, \ldots, k$.

If there is no need for a subtree T_{v_i} to provide one facility to cover v, then the cost of T_{v_i} is $cost_1(T_{v_i})$. Otherwise, its cost is $cost_2(T_{v_i})$. Since only one facility is needed to cover v, the subtree with the smallest value of $cost_2(T_{v_i}) - cost_1(T_{v_i})$ will contribute one facility to cover v. Let T_{v_1} be such a subtree, without loss of any generality. Then,

$$M(v)[r(v), -] = c(v) + cost_2(T_{v_1}) + \sum_{2 \leq j \leq k} cost_1(T_{v_j}).$$

$M(v)[r(v), y], y \in IC(v)$: When the internal coverage y is non-negative, let $v(y)$ be the corresponding internal covering vertex. Now, v is already covered by $v(y)$. We only need to compute one value, denoted by $cost'_1(T_{v_i})$, for each subtree $T_{v_i}, i = 1, \ldots, k$, which is similar to $cost_1(T_{v_i})$. The only difference is that the internal coverage is $\max\{r(v), y\} - d(v, v_i)$ for T_{v_i}, instead of $r(v) - d(v, v_i)$. $M(v)[r(v), y]$ $(y \in IC(v))$ is computed as follows.

$$M(v)[r(v), y] = c(v) + \sum_{i=1}^{k} cost'_1(T_{v_i}).$$

Since $cost_1(T_{v_i}), cost_2(T_{v_i})$, and $cost'_1(T_{v_i})$ can be computed in $O(\log n)$ time for each $i, 1 \leq i \leq k$, each element in the row $r(v)$ of matrix $M(v)$ is computable in $O(k \log n)$ time.

Computing $M(v)[x, \cdot], x \in EC(v)$ and v is not the corresponding external covering vertex Let $v(x)$ be the corresponding external covering vertex of external coverage x. Without loss of any generality, assume that $v(x)$ lies in subtree T_{v_1}.

To compute $M(v)[x, -]$, we consider three subcases:

- $v(x)$ is covered by v (i.e., $r(v) \geq d(v, v(x))$ and $r(v) \leq x$). In this case,

$$\begin{aligned} M(v)[x, -] = \quad & c(v) + M(v_1)[x + d(v, v_1), r(v) - d(v, v_1)] \\ & + \sum_{i=2}^{k} \{ \min_{x' \in EC(v_i); x' \leq x + d(v, v_i)} M(v_i)[x', x - d(v, v_i)] \}; \end{aligned}$$

- $v(x)$ is covered by a facility in T_{v_1}. In this case, all vertices in T_{v_1} are covered by facilities in T_{v_1}. Therefore,

$$\begin{aligned} M(v)[x,-] = \quad & M(v_1)[x+d(v,v_1),-] \\ + & \sum_{i=2}^{k}\{\min_{x'\in EC(v_i);x'\le x+d(v,v_i)} M(v_i)[x',x-d(v,v_i)]\}; \end{aligned}$$

and

- $v(x)$ is covered by a facility in $T_{v_j}, j \neq 1$. In this case, all vertices in $T_{v_1} \setminus \{v(x)\}$ are covered by facilities in T_{v_1}.

$$\begin{aligned} M(v)[x,-] = \quad & M(v_1)[x+d(v,v_1), d(v(x),v_1)] \\ + & \min_{x'\in EC(v_j);d(v(x),v_j)\le x'\le x+d(v,v_j)} M(v_j)[x',x-d(v,v_j)] \\ - & \min_{x'\in EC(v_j);x'\le x+d(v,v_j)} M(v_j)[x',x-d(v,v_j)] \\ + & \sum_{i=2}^{k}\{\min_{x'\in EC(v_i);x'\le x+d(v,v_i)} M(v_i)[x',x-d(v,v_i)]\}. \end{aligned}$$

The distinguished subtree T_{v_j} is the subtree that has the minimum value of

$$\begin{aligned} & \min_{x'\in EC(v_j);d(v(x),v_j)\le x'\le x+d(v,v_j)} M(v_j)[x',x-d(v,v_j)] \\ - & \min_{x'\in EC(v_j);x'\le x+d(v,v_j)} M(v_j)[x',x-d(v,v_j)]. \end{aligned}$$

It is not hard to see that $M(v)[x,-]$ can be computed in $O(k\log n)$ time.

The method to compute values of $M(v)[x,y], y \in IC(v)$ is very similar to the computation of $M(v)[x,-]$. Its details are omitted here.

In summary, the value of each element in the coverage matrix $M(v)$ can be computed in $O(k\log n)$ time, where k is the number of children of v. Thus, the total cost to compute an optimal solution is

$$\sum_{v\in V(T)} O(n^2 k_v \log n) = O(n^3\log n),$$

where k_v is the number of children of v in the rooted tree T.

Theorem 7.3.2 *The constrained covering problem on a tree network can be solved in $O(n^3)$ space and $O(n^3\log n)$ time.*

7.4 Summary

In this chapter we have studied the constrained covering problems (CCPs) in a path network, an extended-star network, and a tree network. For the first time, sub-quadratic algorithms are proposed to solve the CCP on a path network and an extended star network. For the CCP on a tree network, an $O(n^3 \log n)$ algorithm is presented which improves the previous result of $O(n^4)$ by Horne and Smith [44]. The main data structures used in our algorithms are binary search tree structures, segment search tree structures [10], and priority search tree structures [51].

Possible future studies on this problem include the design of more efficient algorithms for the CCP on a tree network, and the examination of the CCP on various other graph topologies such as cactus networks, partial k-trees, etc. Another direction might be the design of polynomial-time approximation algorithms for the CCP on general networks, since the CCP on a general network is strongly NP-hard [45].

Chapter 8

Conclusion

In this thesis, we consider the algorithmic issues for the center and covering location optimization problems where the setting is a network. The demand set consists of all points of the network that require services and the supply set consists of all candidate locations of facilities in the underlying network. The center location problems aim to establish an optimal placement of facilities in the supply set in order to minimize the maximum (weighted) distance from a demand point to its closest facility. The covering location problems seek to establish the minimum number of facilities such that the maximum (weighted) distance from a demand point to its closest facility is no more than a predefined non-negative value. There is a tight relationship between the two problems. Generally, a solution for the covering location problem with a given value can be used to test the feasibility of the value in the corresponding center location problem. Therefore, using the binary search technique or the parametric-searching technique, one can easily obtain an efficient solution for the center problem from an efficient solution for the corresponding covering problem.

Four cases of the center problem and the corresponding covering problem, where the demand set and the supply set are either subsets of the vertex set or subsets of the point set of the underlying network, are considered. Moreover, when the demand set is a subset of the vertex set, its weighted version of the problem is also considered where each demand vertex is associated with a non-negative weight.

We first studied center/covering location problems in general networks as well as specialized networks, such as tree networks, cactus networks, and partial k-tree networks (fixed k). We then studied some variations of the network center/covering location problem, including conditional extensive facility location problems, continuous p-edge-partition problems, and constrained covering problems. Here, we have only considered three variations in an edge-weighted tree network.

The specific problems considered in this thesis are

The continuous p-center problem in general networks: Here demand points are located at vertices of a general network G with n vertices and m edges, and centers can be located anywhere in G. The objective is to locate a set of p centers such that the maximum distance from demand points to their closest centers is minimized. Two cases of this problem are considered: (i) unweighted, and (ii) weighted. We provide an $O(m^p n^{p/2} \log^2 n)$-time algorithm for both cases. An $O(m^p n^p \log n\alpha(n))$-time algorithm for the weighted case and an $O(m^p n^{p-1} \log^2 n\alpha(n))$-time algorithm for the unweighted case were presented in [69], where $\alpha(n)$ is the inverse Ackermann function [26]. Thus our algorithm is an improvement over the existing result on the problem by a factor of almost $O(n^{p/2})$.

For the general p-center problem in which the demand set contains all points of the underlying network, a candidate set containing the optimal solution value is characterized in Tamir's paper [68]. In spite of the nice structure, the size of this set is not polynomial even for simple structures such as cactus networks. Until now, no efficient algorithm is known for the problem in a general network. It is a challenge to design an efficient algorithm to solve the problem even for a relatively small p.

Weighted p-center problems in trees: Here the underlying network is restricted to be a tree and each demand point is associated with a non-negative weight. We only consider the case when p is a fixed constant, and have considered the following two cases: (i) the supply set is a subset of the vertex set, and (ii) the supply set is a subset of the point set. An optimal algorithm is provided for both cases, which is a nontrivial generalization of Megiddo's 'trimming' technique [54]. When $p = 1$, Megiddo [54] used the trimming technique to solve the weighted 1-center problems in linear time. The problem of generalizing the trimming approach to solve the p-center problem for $p > 1$ has been open open for over twenty years. This result partially resolves the long standing open problem.

Moreover, we introduce a simple parametric-pruning approach for the weighted 1-center problem, which is adapted to solve the weighted p-center problem on the real line in linear time for any fixed value p.

Note that the running time of our algorithms for the p-center problems in a tree network or the real line is exponential in p. One challenging task will be to design an $O(f(p) \cdot n)$-time algorithm for the problems where $f(p)$ is a low-degree polynomial of p.

Various p-center problems in tree-like networks: Here the center problems are studied in a tree-like network. We consider two cases: (i) a partial k-trees, and (ii) a cactus network.

When the underlying network is a partial k-tree, we consider the weighted case only, and have considered both discrete and continuous versions of the problem. For the discrete version, we proposed an $O(pn^p \log^k n)$-time algorithm. This result is better than the $O(p^2 n^{k+2})$ result of Granot and Skorin-Kapov [32] when $p < k + 2$. For the continuous version, we devised the first polynomially bounded algorithm for fixed k, which runs in $O(p^2 k^{k+1} n^{2k+3} \log n)$ time.

When the underlying network is a cactus network, we have considered several variations of the problem for the first time. When $p = 1$, an $O(n \log n)$-time algorithm is proposed to solve the weighted continuous 1-center problem. When $p = 2$, an $O(n \log^3 n)$-time algorithm is proposed for the weighted continuous 2-center problem. When p is a part of the input, we have devised efficient algorithms for various p-center problems, using the parametric-searching technique. In particular, we propose an $O(n \log^2 n)$-time algorithm for the weighted discrete p-center problem algorithm, $O(n^2)$ algorithms for the weighted continuous p-center problem and the unweighted discrete p-center problem with a demand set of infinite size, and an $O(n^2 \log^2 n)$ algorithm for the general p-center problem.

Many issues in a cactus network are still unresolved. For instance, it would be interesting to find out whether there exists an optimal linear-time algorithm for the weighted 1-center problem. We conjecture that all the p-center problems can be solved in subquadratic time by designing polylog parallel algorithms for the corresponding feasibility tests, and by using Megiddo's results [53]. Furthermore, we suspect that an $O(n)$ test for the general p-center problem can be derived by properly modifying the test for $V(G)/V(G)/p$ in [30]. This will lead to the improved bound $O(n^2)$ for $A(G)/A(G)/p$ in a cactus network.

The most challenging problem is to find more efficient algorithms to solve the p-center problems in an edge-weighted partial k-tree of bounded treewidth.

Conditional extensive facility location problems in tree networks: Here the new facility is not represented by a point, and a set of existing facilities are already located in the underlying tree. The objective is to minimize the maximum weighted distance from the demand points to the union of this new facility and the set of existing facilities. Two cases of the problem have been considered: (i) a path-shaped facility, and (ii) a tree-shaped facility. We propose optimal linear-time algorithms for both cases, using the parametric-pruning technique. These results improve the recent $O(n \log n)$ results of Tamir et al. [70].

For the case when the service cost $f_i(x)$ of a demand point v_i is a nondecreasing piecewise linear function of the service distance x to the facility with a fixed number of breakpoints (in our 'conditional' problems $f_i(x)$ has only one breakpoint), all the ideas presented to solve the conditional extensive facility location problem in a tree network can be extended to achieve an optimal algorithm for the new case in a tree network. Actually, our method works even when the piecewise linearity assumption is relaxed to piecewise polynomiality (e.g. quadratic, or cubic) of fixed degree.

Continuous tree p-edge-partition problems: Here the vertices in the underlying tree network are unweighted. A continuous p-edge-partition of a tree is to divide it into p subtrees by selecting $p - 1$ cut points along the edges. We have considered two objective functions: (i) maximize the minimum length of the p subtrees, and (ii) minimize the maximum length of the p subtrees. An $O(n^2)$-time algorithm for both objectives was presented in [49]. We propose an $O(n \log^2 n)$-time algorithm for the max-min objective, which is a substantial improvement of the previous result. For the min-max objective, an $O(nh_T \log n)$-time algorithm is proposed where h_T is the height of the underlying tree network. When $h_T = o(n/\log n)$, our result for the min-max problem is better. We conjecture that our algorithms for the tree network can be extended to cactus networks.

Constrained covering problems in tree networks: Here open-facility costs and coverage radii of vertices in the underlying tree networks are considered. A facility located at a vertex u incurs a non-negative open-facility cost $c(u)$, and provides a non-negative coverage radius of $r(u)$. A demand point is covered by a facility if the demand point lies within the coverage radius of the facility, and an established facility must be covered by another established facility. The objective is to minimize the sum of open-facility costs required to cover all demand vertices.

We have studied the constrained covering problems in a path network, an extended star network, and a tree network. In [45], $O(n^2)$-time algorithms for the problem in a path network and an extended star network were presented, and in [44], an $O(n^4)$ result for a tree network was presented. We proposed the first sub-quadratic algorithms to solve the problem in a path network and an extended star network. In particular, our algorithm for a path network runs in time $O(n \log n)$ and our algorithm for an extended-star network runs in time $O(n^{1.5} \log n)$. For the problem in a tree network, we proposed an $O(n^3 \log n)$ algorithm. The main data structures used in our algorithms are the binary search tree structure, the segment search tree structure [10], and priority search tree structures [51].

Possible future studies on this problem include the design of more efficient algorithms for the problem on a tree network, and the examination of the problem on various other graph topologies such as cactus networks, partial k-trees, etc. Another direction might be the design of polynomial-time approximation algorithms for the problem on general networks, since the problem on a general network is strongly NP-hard [45].

Bibliography

[1] P.K. Agarwal and M. Sharir. Planar geometric location problems. *Algorithmica*, 11(2):185–195, 1994.

[2] P.K. Agarwal and M. Sharir. Efficient algorithms for geometric optimization. *ACM Comput. Surv.*, 30(4):412–458, 1998.

[3] J.A. Barcia, J.M. Díaz-Báñez, A.J. Lozano, and I. Ventura. Computing an obnoxious anchored segment. *Oper. Res. Lett.*, 31(3):293–300, 2003.

[4] B. Ben-Moshe, B.K. Bhattacharya, S. Das, D.R. Gaur, and Q. Shi. Computing a planar widest empty alpha-siphon in $o(n^3)$ time. In *CCCG*, pages 33–36, 2007.

[5] B. Ben-Moshe, B.K. Bhattacharya, and Q. Shi. Computing the widest empty boomerang. In *CCCG*, pages 80–83, 2005.

[6] B. Ben-Moshe, B.K. Bhattacharya, and Q. Shi. Efficient algorithms for the weighted 2-center problem in a cactus graph. In *ISAAC*, pages 693–703, 2005.

[7] B. Ben-Moshe, B.K. Bhattacharya, and Q. Shi. An optimal algorithm for the continuous/discrete weighted 2-center problem in trees. In *LATIN*, pages 166–177, 2006.

[8] B. Ben-Moshe, B.K. Bhattacharya, Q. Shi, and A. Tamir. Efficient algorithms for center problems in cactus networks. *Theor. Comput. Sci.*, 378(3):237–252, 2007.

[9] R. Benkoczi. *Cardinality constrainted facility location problems in trees.* PhD thesis, School of Computing Science, SFU, Canada, 2004.

[10] M. Berg, M. Kreveld, M. Overmars, and O. Schwarzkopf. *Computational geometry: algorithms and applications.* Springer-Verlag, second edition, 2000.

[11] O. Berman and D. Simchi-Levi. Conditional location problems on networks. *Transportation Science*, 24:77–78, 1990.

[12] B.K. Bhattacharya, Y. Hu, Q. Shi, and A. Tamir. Optimal algorithms for the path/tree-shaped facility location problems in trees. In *ISAAC*, pages 379–388, 2006.

[13] B.K. Bhattacharya and Q. Shi. *Optimal algorithms for weighted p-center problem in trees, any fixed p.* SFU, 2006.

[14] B.K. Bhattacharya and Q. Shi. Optimal algorithms for the weighted p-center problems on the real line for small p. In *WADS*, pages 529–540, 2007.

[15] B.K. Bhattacharya, Q. Shi, and A. Tamir. Optimal algorithms for the path/tree-shaped facility location problems in trees. *Algorithmica*, 2008.

[16] M. Blum, R.W. Floyd, V.R. Pratt, R.L. Rivest, and R.E. Tarjan. Time bounds for selection. *J. Comput. Syst. Sci.*, 7(4):448–461, 1973.

[17] H.L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM J. Comput.*, 25(6):1305–1317, 1996.

[18] R.E. Burkard and H. Dollani. Center problems with pos/neg weights on trees. *Eur. J. Oper. Res.*, 145(3):483–495, 2003.

[19] R.E. Burkhard and J. Krarup. A linear algorithm for the pos/neg-weighted 1-median problem on a cactus. *Computing*, 60(3):193–215, 1998.

[20] R. Chandrasekaran and A. Tamir. An $o((n \log p)^2)$ algorithm for the continuous p-center problem on a tree. *SIAM J. Alg. Disc. Meth.*, 1:370–375, 1980.

[21] R. Chandrasekaran and A. Tamir. Polynomially bounded algorithms for locating p-centers on a tree. *Math. Prog.*, 22(1):304–315, 1982.

[22] S. Chaudhuri and C.D. Zaroliagis. Shortest paths in digraphs of small treewidth. part i: sequential algorithms. *Algorithmica*, 27(3):212–226, 2000.

[23] B. Chazelle and L.J. Guibas. Fractional cascading: I. a data structuring technique. *Algorithmica*, 1(2):133–162, 1986.

[24] M.-L. Chen, R.L. Francis, and T.J. Lowe. The 1-center problem: exploiting block structure. *Transport. Sci.*, 22(4):259–269, 1988.

[25] R. Cole. Slowing down sorting networks to obtain faster sorting algorithms. *J. ACM*, 34(1):200–208, 1987.

[26] T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein. *Introduction to Algorithms, Second Edition.* The MIT Press and McGraw-Hill Book Company, 2001.

[27] J.M. Díaz-Báẽz, M.A. López, and J.A. Sellarès. On finding a widest empty 1-corner corridor. *Inf. Process. Lett.*, 98(5):199–205, 2006.

[28] Z. Drezner and H.W. Hamacher. *Facility location: application and theory.* Springer-Verlag, 2002.

[29] G.N. Frederickson. Parametric search and locating supply centers in trees. In *WADS*, pages 299–319, 1991.

[30] G.N. Frederickson and D.B. Johnson. Finding k-th paths and p-centers by generating and searching good data structures. *J. Algorithms*, 4(1):61–80, 1983.

[31] M.R. Garey and D.S. Johnson. *Computers and intractability: a guide to the theory of NP-completeness.* W. H. Freeman & Co., New York, NY, USA, 1990.

[32] D. Granot and D. Skorin-Kapov. On some optimization problems on k-trees and partial k-trees. *Disc. App. Math.*, 48(2):129–145, 1994.

[33] Y. Gurevich, L. Stockmeyer, and U. Vishkin. Solving np-hard problems on graphs that are almost trees and an application to facility location problems. *J. ACM*, 31(3):459–473, 1984.

[34] S.L. Hakimi. Optimum location of switching centers and the absolute centers and medians of a graph. *Oper. Res.*, 12:450–459, 1964.

[35] S.L. Hakimi, E.F. Schmeichel, and M. Labbe. On locating path or tree shaped facilities on networks. *Networks*, 23:543–555, 1993.

[36] N. Halman. *Discrete and lexicographic helly theorems and their relations to LP-type problems.* PhD thesis, Tel Aviv Univ., 2004.

[37] N. Halman and A. Tamir. Continuous bottleneck tree partitioning problems. *Discrete Appl. Math.*, 140(1-3):185–206, 2004.

[38] G.Y. Handler. Minimax location of a facility in an undirected tree graph. *Transport. Sci.*, 7:287–293, 1973.

[39] G.Y. Handler and P.B. Mirchandani. *Location on networks theory and algorithms.* MIT Press, Cambridge, 1979.

[40] F. Harary. *Graph theory.* Addison-Wesley, 1969.

[41] R. Hassin and A. Tamir. Efficient algorithms for optimization and selection on series-parallel graphs. *SIAM J. Algebraic Discrete Methods*, 7(3):379–389, 1986.

[42] S.M. Hedetniemi, E.J. Cockaine, and S.T. Hedetniemi. Linear algorithms for finding the jordan center and path center of a tree. *Transport. Sci.*, 15:98–114, 1981.

[43] J. Herschberger and S. Suri. Offline maintenance of planar configurations. In *SODA '91: Proceedings of the second annual ACM-SIAM symposium on Discrete algorithms*, pages 32–41, Philadelphia, PA, USA, 1991. Society for Industrial and Applied Mathematics.

[44] J.A. Horne and J.C. Smith. A dynamic programming algorithm for the conditional covering problem on tree graphs. *Netw.*, 46(4):186–197, 2005.

[45] J.A. Horne and J.C. Smith. Dynamic programming algorithms for the conditional covering problem on path and extended star graphs. *Netw.*, 46(4):177–185, 2005.

[46] M. Jeger and O. Kariv. Algorithms for finding p-centers on a weighted tree (for relatively small p). *Networks*, 15(3):381–389, 1985.

[47] O. Kariv and S.L. Hakimi. An algorithmic approach to network location problems. i: the p-centers. *SIAM Journal on Applied Mathematics*, 37(3):513–538, 1979.

[48] Y.-F. Lan, Y.-L. Wang, and H. Suzuki. A linear-time algorithm for solving the center problem on weighted cactus graphs. *Inf. Process. Lett.*, 71(5-6):205–212, 1999.

[49] J.-J. Lin, C.-Y. Chan, and B.-F. Wang. Improved algorithms for the continuous tree edge-partition problems. submitted to Disc. App. Math., 2007.

[50] B.J. Lunday, J.C. Smith, and J.B. Goldberg. Algorithms for solving the conditional covering problem on paths. *Naval Res Logistics*, 52(4):293–301, 2005.

[51] E.M. McCreight. Priority search trees. *SIAM J. Comput.*, 14(2):257–276, 1985.

[52] N. Megiddo. Combinatorial optimization with rational objective functions. *Math. of Oper. Res.*, 4(4):414–424, 1979.

[53] N. Megiddo. Applying parallel computation algorithms in the design of serial algorithms. *J. ACM*, 30(4):852–865, 1983.

[54] N. Megiddo. Linear-time algorithms for linear programming in r^3 and related problems. *SIAM Journal on Computing*, 12(4):759–776, 1983.

[55] N. Megiddo and A. Tamir. New results on the complexity of p-center problems. *SIAM J. Comput.*, 12(4):751–758, 1983.

[56] N. Megiddo, A. Tamir, E. Zemel, and R. Chandrasekaran. An $o(n \log n)$ algorithm for the k-th longest path in a tree with applications to location problems. *SIAM J. Comput.*, 10(2):328–337, 1981.

[57] J.A. Mesa. The conditional path center problem in tree graphs. presented to EWGLA8 held in Lambrecht, Germany, 1995.

[58] E. Minieka. Conditional centers and medians on a graph. *Networks*, 10:265–272, 1980.

[59] E. Minieka. The optimal location of a path or tree in a tree network. *Networks*, 15:309–321, 1985.

[60] I. Douglas Moon and S.S. Chaudhry. An analysis of network location problems with distance constraints. *Manag. Sci.*, 30(3):290–307, 1984.

[61] I.D. Moon and L. Papayanopoulos. Facility location one a tree with maximum distance constraints. *Compu. Oper. Res.*, 22(9):905–914, 1995.

[62] J.A. Moreno. A new result on the complexity of the p-center problem. Technical report, Universidad Complutense, Madrid, Spain, 1986.

[63] M.H. Overmars and C.-K. Yap. New upper bounds in klee's measure problem. *SIAM J. Comput.*, 20(6):1034–1045, 1991.

[64] J. Pfaff, R. Laskar, and S.T. Hedetniemi. Np-completeness of total and connected domination, and irredundance for bipartite graphs. Technical report, Dept. Mathematical Sciences, Clemson University, Clemson, South Carolina, 1983.

[65] M.B. Rayco, R.L. Francis, and A. Tamir. A p-center grid-positioning aggregation procedure. *Computers & Oper. Res.*, 26(10-11):1113–1124, 1999.

[66] N. Robertson and P.D. Seymour. Graph minors. ii. algorithmic aspects of tree-width. *J. Algorithms*, 7(3):309–322, 1986.

[67] A. Shioura and M. Shigeno. The tree center problems and the relationship with the bottleneck knapsack problems. *Networks*, 29(2):107–110, 1997.

[68] A. Tamir. On the solution value of the continuous p-center location problem on a graph. *Math. Oper. Res.*, 12(2):340–349, 1987.

[69] A. Tamir. Improved complexity bounds for center location problems on networks by using dynamic data structures. *SIAM J. Discret. Math.*, 1(3):377–396, 1988.

[70] A. Tamir, J. Puerto, J.A. Mesa, and A.M. Rodríguez-Chía. Conditional location of path and tree shaped facilities on trees. *J. Algorithms*, 56(1):50–75, 2005.

[71] A. Tamir, J. Puerto, and D. Pérez-Brito. The centdian subtree on tree networks. *Discrete Appl. Math.*, 118(3):263–278, 2002.

[72] L. G. Valiant. Parallelism in comparison problems. *SIAM Journal on Computing*, 4(3):348–355, 1975.

[73] B.-F. Wang. Efficient parallel algorithms for optimally locating a path and a tree of a specified length in a weighted tree network. *J. Algorithms*, 34(1):90–108, 2000.

[74] B.-F. Wang. Finding r-dominating sets and p-centers of trees in parallel. *IEEE Trans. Parallel Distrib. Syst.*, 15(8):687–698, 2004.

[75] E. Zemel. On search over rationals. *Oper. Res. Lett.*, 1:34–38, 1981.

[76] E. Zemel. An $o(n)$ algorithm for the linear multiple choice knapsack problem and related problems. *Inf. Process. Lett.*, 18(3):123–128, 1984.