2-Median Problems in Tree Networks

by

Rashmisnata Acharyya

B.Tech., Tezpur University, 2013

Project Submitted in Partial Fulfillment of the
Requirements for the Degree of
Master of Science

in the
School of Computing Science.
Faculty of Applied Sciences

SIMON FRASER UNIVERSITY
Spring 2017

Approval

Name:	**Rashmisnata Acharyya**
Degree:	**Master of Science (Computing Science)**
Title:	***2-Median Problems in Tree Networks***
Examining Committee:	**Chair:** Faraz Hach Research Associate

Binay Bhattacharya
Senior Supervisor
Professor

Ramesh Krishnamurti
Supervisor
Professor

Abraham P. Punnen
External Examiner
Professor
Department of Mathematics
Simon Fraser University

Date Defended: 29 March 2017

Abstract

Facility Location Problems have a great significance for allocating resources efficiently in a network. The interaction mainly involves a price which depends on the distances between the objects and the order of significance of the objects(clients). The applications of such problems are immense in many application areas such as medical and transportation.
In this project, we consider the p-median facility location problem in tree-networks. This p-median problem in general tree-networks is NP-hard. In this project, we have looked at efficiently solving the 2-median problem in tree networks. Using simple techniques of computational geometry, we give a $O(n \log s)$ time solution to the 2-median of a tree with s number of leaves. Our technique is then applied to solve other variants of the 2-median problem with the same complexity.

Keywords: Facility Location; 2-median; Link-deletion; Path Partition

Acknowledgements

I would like to thank Professor Binay Bhattacharyya for helping me in the best possible way throughout this project. It would not have been possible without his help and supervision. I also thank him for helping me with my report and providing me the resources to understand the concepts in a better way. I would also like to thank Dr. Ante Custic for helping me understand the concepts very well and teaching me how to write a standard project report. His patience and constant support served as the best source of encouragement for completion of this project.

Contents

List of Figures

Chapter 1

Introduction

Efficient usage of resources is a very significant concern today. A tremendous amount of study has been done in the field of resource allocation. Facility location problem is one of them. A facility location problem is an optimization problem where we find optimal sites for placing facilities so that they can serve a large set of clients. These clients which are provided services by the facilities around them are called demand points. For example, when a city decides to set up its new hospitals, their locations are of utmost significance. If the hospital is at a walking distance or within a smaller distance from a locality, then many emergency cases can be taken care of. Also setting up a business enterprise at a optimized location ensures a good amount of profit. Depending on the type of facility that has to be located, we decide on the constraint to be optimized. For some cases it may be the sum of distances whereas for some it may be the maximum distance. The p-median, p-center and p-cover problems are two of the very popular classes of facility location problems that model both public and private location decision problems. The p-median problem is a model that decides the best location for p new hospitals such that the sum of the distances from all the demand points (total service cost) to these hospitals is minimized. The p-center decides the best location of the p facilities such that the maximal distance from all demand points to its closest facility (service cost to the most expensive client) can be minimum. The p-cover problem decides the best location of the p facilities such that most demand points are covered. All the facilities are assigned a fixed coverage distance (discussed later).
The applications of p-median problems are immense. Nooradelena [11] has used the concept of p-median for solving emergency medical problems. Also, Fagueye [19] resolves the problem of location of schools for high school students using this popular concept. Hakimi [15] explains the problem of placing switching centers of a telephone network by using the concept of p-median problems. Location of a business franchise is another example where p-median and p-center problems are used extensively.

1.0.1 Classical Facility Location Problems

A network $G = (V, E)$ comprises a set V of vertices where $V=\{v_1, v_2, v_3, \ldots, v_n\}$ and E is the set of edges which are 2-element subsets of V. Let $l_{ij} > 0$ be the length of the edge $(v_i, v_j) \in E$. Let $w(v_i) \geq 0$ be the weight assigned to each vertex v_i. We define the distance from v_i to v_j denoted by $d(v_i, v_j)$ to be the shortest path length from vertex v_i to v_j using the edge lengths. We also define another important term called the weighted distance. The weighted distance from v_i to v_j is $w(v_i).d(v_i, v_j)$. It is important to note that the weighted distance from v_i to v_j is not equal to the weighted distance from v_j to v_i in general.

p-median (min-sum) problem

We define the p-median problem in network graphs as the problem of locating p points (on vertices or edges) in the graph G such that the sum of the weighted distances from all vertices of the graph to the nearest of these p points is minimized. Mathematically, the p-median problem is the problem of finding the set of points S, such that $|S| = p$ which minimizes the objective function

$$cost(S) := \sum_{v \in V} w(v) \cdot d(v, S). \tag{1.1}$$

The term $d(v, S)$ denotes $min\{d(v, w)\}, \forall w \in (S - v)$.

For convenience, a term *p-medians* will be used to denote an optimal solution of a p-median problem, i.e., the set S of p points minimizing (1.1). Thus, a 1-median of a graph is the point of the graph such that the sum of the weighted distances from all the vertices of the graph to this point is minimized. Similarly, 2-medians of a graph is a pair of points in a graph such that the weighted distances from all the vertices of the graph to the nearest of these two points is minimized.

A useful property of p-median problems on graphs is *vertex optimality*. It means that there exists an optimal solution where all the facility points of the optimal solution are located at a vertices. The proof has been discussed later in the report.

Theorem 1.0.1. *The problem of finding a p-median is NP-hard even in the case when the network is a planar graph of maximum vertex degree 3, all whose edges are of length 1 and all whose vertices have weight 1. (Kariv and Hakimi [17]).*

The proof uses a simple transformation from a well known dominating set problem which is already known to be NP-complete. Let us assume that all the edges of our graph are of unit length. A dominating set for the graph with cardinality p exists if and only if the cost of optimal p-median is $n - p$.

Since we saw that p-median problem is NP-hard for a general graph with arbitrary p. Therefore, in this report, we focus on special and simpler graphs (tree networks) in order to devise fast polynomial time algorithms.

The very first algorithms which were introduced for p-median problems in tree networks used dynamic programming approach. Kariv and Hakimi, in their paper on facility location [17] gave an $O(p^2n^2)$ algorithm for p-median problem using dynamic programming and later Tamir [20] introduced another algorithm using dynamic programming with an improved complexity of $O(pn^2)$. Chrobak, Larmore and Rytter [10] solved the p-median problem on rooted directed trees in a bound of O($npolylog(n)$) time whenever p is constant. The first subquadratic time complexity was given by Benkoczi [4] with complexity $O(n \log^{(p+2)} n)$ for tree networks and for general p.

p-center (min-max) problems

Another important facility location problem is the p-center problem. We define p-center problems as the problem of locating p facilities in a graph such that the maximum distance from the facilities to the clients can be minimized. Mathematically the p-center problem is the problem of finding the set of points S, such that $|S| = p$ which minimizes the objective function

$$cost(S) := \max\{d(v, S)\}. \tag{1.2}$$

There are two variations of the p-center problem namely veryex center and absolute center. Vertex center problems are when the facilities are located at the vertices and absolute center problems are when they are located anywhere in the graph. Thus vertex optimality is not satisfied in p-center problems.

Theorem 1.0.2. *The discrete p-center problem is NP-hard even in the case of planar graphs with maximum degree 3 with unit length edges and vertex weight (Kariv and Hakimi [17]).*

The proof of this theorem is identical to proof of theorem 1.0.1.

p-cover problems

p-cover problem is the problem of locating p points (on vertices or edges) in the network graph G such that the maximum number of clients can be served by these facilities. Each facility can serve a client within a given fixed coverage weighted distance. There is another variation of the cover problem called the covering location problem. In this problem, each client has to be within a fixed coverage weighted distance of at least one facility. The objective, thus is to minimize the number of facilities needed to cover all clients. Like the p-center problem, facilities here can also be located anywhere on the graph. This problem is also NP-hard for general graphs.(Kariv and Hakimi [17])

1.0.2 1-median problem in tree networks

1-median of a graph is the point (facility) in a graph such that the weighted distances from all the vertices (clients) of the graph to this point is minimized. The classical linear time

algorithm for finding the 1-median vertex of a tree was introduced by Goldman[13] known as the Goldman's Majority algorithm. In this algorithm it was observed that the location of the 1-median depends only on the weight of the vertices and is irrespective of the edge length. This observation was due to a theorem which is as follows.

Theorem 1.0.3. *If $v \in V$ is a vertex of a rooted tree T with children $v_1, v_2, \ldots, v_x$ then v is the 1-median of T if and only if the weights of T_{v_i} (denoted by $w(T_{v_i})$) is less than or equal to $w(T)/2$ and $w(T - T_v) \leq w(T)/2$.*

Here T_v is the subtree of T rooted at v. Given a graph, the eccentricity of a vertex v is defined as the greatest distance from v to any other vertex [22]. A centroid of a graph is a vertex with minimal eccentricity [22]. The vertex v is also called the centroid of T if the vertices of the tree are of equal weights. Also, vertex v is said to satisfy Goldman's condition. Following is the algorithm.

Algorithm 1 1-median (T)

```
1: Let v be the child of root(T) for which w(T_v) is maximum
2: if w(T_v) ≤ w(T)/2 then
3:     return root(T)
4: else
5:     w(v) = w(v) + w(T − T_v)
6:     return 1-median(T_v)
```

1.0.3 2-median problem in tree networks

As discussed earlier, 2-medians of a graph is a pair of points (facilities) in a graph such that sum of the weighted distances from all the vertices (clients) of the graph to the nearest of these two points is minimized. The 2-median problem thus involves finding this pair of points in the graph. The 2-median problem has been a popular problem. Following is a brief overview of the work done on this problem so far.

For the 2-median problem, Gavish and Sridhar [12] introduced an algorithm with a running time of $O(n \log n)$ using the link deletion method (discussed later in section 3.0.1). Another $O(n \log n)$ method for solving the 2-median problem was introduced by Breton [6]. A data structure called the *spine tree decomposition* was used which allow the time complexity of computing the 1-median tree to logarithmic time. Auletta et al. [1] proposed a linear time algorithm for the 2-median problem in a tree however their analysis was found to have errors [6]. So far there is no such algorithm which solves the 2-median problem in linear time.

Breton [6] also introduced linear time algorithms for 2-median of trees under special properties. The two special constraints were firstly the edges were considered to be of integral length bounded by a fixed constant and secondly the weights of the vertices could be sorted in linear time. Similar results are also reported in [18].

Ku et al. [18] introduced a $O(n \log n)$ time algorithm for generalizations of the 2-median problem on trees, which is done by imposing constraints on the two facility vertices considered for 2-medians. The first constraint being limiting the distance between two facilities and the other is to limit their eccentricity. They also solved the generalized cases in linear time (under some additional conditions).

1.0.4 Our Contribution

In this report, we introduce an algorithm to find 2-medians of a tree through the link deletion method. The time complexity of the algorithm is $O(n \log s)$, where s is the number of leaves of the tree. This is an improvement on all the existing polynomial time algorithms on 2-medians. We use linear time implementable preprocessing ideas from the thesis of Breton [6]. Furthermore, we explore the generalizations of the 2-median problem on trees stated by Ku et al. [18], and show that those can also be solved in $O(n \log s)$ time.

1.0.5 Report Outline

In Chapter 2, we provide the basic notations used throughout the report and also discuss the important property of vertex optimality that applies to p-median problem in arbitrary graph. In the same chapter we discuss the 1-median algorithm and its preprocessing steps.

In Chapter 3, we discuss the concept of link deletion and some important lemmas related to the 2-median algorithm. After that, we present the 2-median algorithm of $O(n \log s)$ complexity in details explaining their time complexity and proof of correctness.

In Chapter 4, the generalized algorithms for 2-median problem is presented with eccentricity and distance constraints. Both the algorithms are of complexity $O(n \log s)$. The important concept of path partition is also discussed in the chapter.

In Chapter 5, we conclude the report and discuss some future works.

Chapter 2

Background

In this chapter, we start by stating the notations which are used throughout the paper. We also explain the vertex optimality constraint related to the p-medians along with its proof. After that, we discuss the preprocessing algorithm (Algorithm 2) of linear time complexity.

2.0.1 Preliminaries

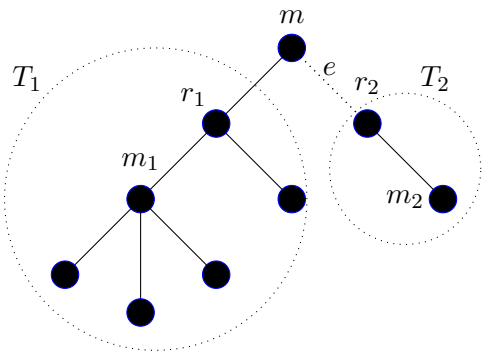

Figure 2.1: Tree T representing different notations

Notations

- T: The input tree with n nodes rooted at its 1-median.
- m: The 1-median node of the input tree T.
- T_1: The largest weighted subtree among subtrees rooted at children of 1-median (node m) of the tree. The root of T_1 is r_1. In Fig 2.1, T_1 has been encircled.

- T_2: The second heaviest tree among subtrees rooted at children of 1-median (m) of the tree. The root of T_2 is r_2. In Fig 2.1, T_2 has been encircled.
- m_1: The 1-median node of T_1.
- m_2: The 1-median node of T_2.
- π_1: The path from node m to node m_1 .
- π_2: The path from node m to node m_2 .
- s: The number of leaves in T.
- W: Total weight of the vertices of the input tree T.
- T_v : Subtree of T rooted at v. In Fig 2.1, T_1 can also be represented as T_{r_1}
- $m(T_v)$: 1-median node of T_v where T_v is the subtree of T. In Fig 2.1, $m(T_{r_1}) = m_1$.
- $p(u)$: Parent of node u in T. In Fig 2.1, r_1 is the $p(m_1)$.
- $T'(e)$: On the removal of e in T (in Fig 2.1) it decomposes into two subtrees. The subtree that contains m is $T'(e)$ (as depicted in Fig 2.2).
- $T''(e)$: T-$T'(e)$ (The subtree that does not contain m)

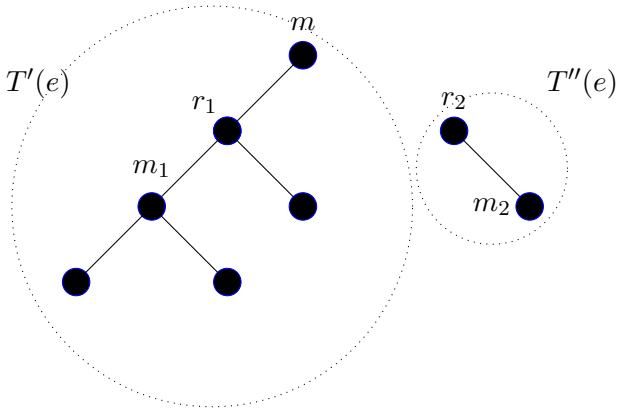

Figure 2.2: Decomposition of T from Figure 1

2.0.2 Concepts

Following is an important observation of the p-median problem.

Lemma 2.0.1. *The p-median problem satisfies vertex optimality property.*

Proof. Let us assume that one of the selected facilities x^* is located on an edge $e = (v, u)$. Let $C_{x^*}(u)$ be the set of clients served by x^* such that their shortest path to x^* goes through u. We define $C_{x^*}(v)$ likewise. We can claim that if x* is located in the interior of e, then $w(C_{x^*}(u))$ and $w(C_{x^*}(v))$ must be equal. If not then let us assume $w(C_{x^*}(u)) \geq w(C_{x^*}(v))$. If we move the facility x^* to u, then there is a decrease in the solution. The reason is that although there is an increase by $w(C_{x^*}(v)) \cdot d(u, x^*)$ but, at the same time there is a decrease in the value by $w(C_{x^*}(u)) \cdot d(x^*, u)$. Since $w(C_{x^*}(u)) \geq w(C_{x^*}(v))$ so moving x^* to u will only make the solution better. □

As stated earlier, chapter 3 of this report introduces a simple algorithm to find the 2-median of an arbitrary tree in $O(n \log s)$. The algorithm uses the popular link deletion method (which is discussed in section 3.0.1). Also, some preprocessing is done so that some commonly used computations can be retrieved in constant time thus improving the runtime of the algorithm. The preprocessing step has been discussed below.

2.0.3 Preprocessing

The following information are precomputed for every vertex v in T rooted at the 1-median m.

(i) The location of $m(T_v)$

(ii) $cost_{T_v}(m(T_v))$

(iii) $d(v, m(T_v))$.

(iv) $d(v, m)$

(v) $cost_{T_v}(v)$

The preprocessing of the tree is done so that the important information can be retrieved in constant time. Algorithm 2 (described below) computes 1-median of T_v and its costs for all v in linear time [6] .

Computation of (i) and (ii)

The algorithm is based on the property that the 1-median of a tree changes as we add weights to a tree node. If we do not do so then the Goldman's condition (as stated in the introduction) will be violated. Thus the new 1-median lies in the path

Algorithm 2 preprocessing (v)

1: $Cost = 0$
2: **if** v is a leaf **then**
3: $m(T_v) = v$;
4: $cost_{T_v}(v) = 0$
5: **else**
6: **for** all children v_c of v **do**
7: preprocessing(v_c)
8: **for** the heaviest subtree T_h of v **do** (See Figure 2.3)
9: $u = m(T_h)$
10: $Cost = cost_{T_u}(u)$
11: **while** $w(T_u) < w(T_v)/2$ **do**
12: $Cost+ = w(T_u) \cdot d(u, p(u)) - w(T_v - T_u) \cdot d(u, p(u))$
13: $u = p(u)$
14: **if** $(u \neq v)$ **then**
15: $m(T_v) = u$
16: $return$
17: $m(T_v) = v$
18: $cost_{T_v}(m(T_v)) = Cost$
19: return

connecting the old 1-median and the newly added weight. So to compute the 1-median of the subtrees rooted at every vertex of the tree (as described in Algorithm 2) each edge of the tree is traversed at most once (Step 11). Similar is the case of the cost computation of 1-median of T_v. Hence (i) and (ii) can be retrieved in linear time for all v.

Also from that algorithm, we can find the cost of 1-median of the tree.

Lemma 2.0.2. *The location of $m(T_v)$ and $cost_{T_v}(m(T_v))$ for all v can be computed in linear time*

Computation of (iii) and (iv)

For computing (iii) and (iv), a depth first search is done in T starting from the 1-median of the tree i.e. m. We can directly obtain $d(v, m)$ by doing so. Also

$$d(v, m(T_v)) = d(m, m(T_v)) - d(m, v). \tag{2.1}$$

Lemma 2.0.3. *$d(v, m(T_v))$ and $d(v, m)$ for all v can be computed in linear time*

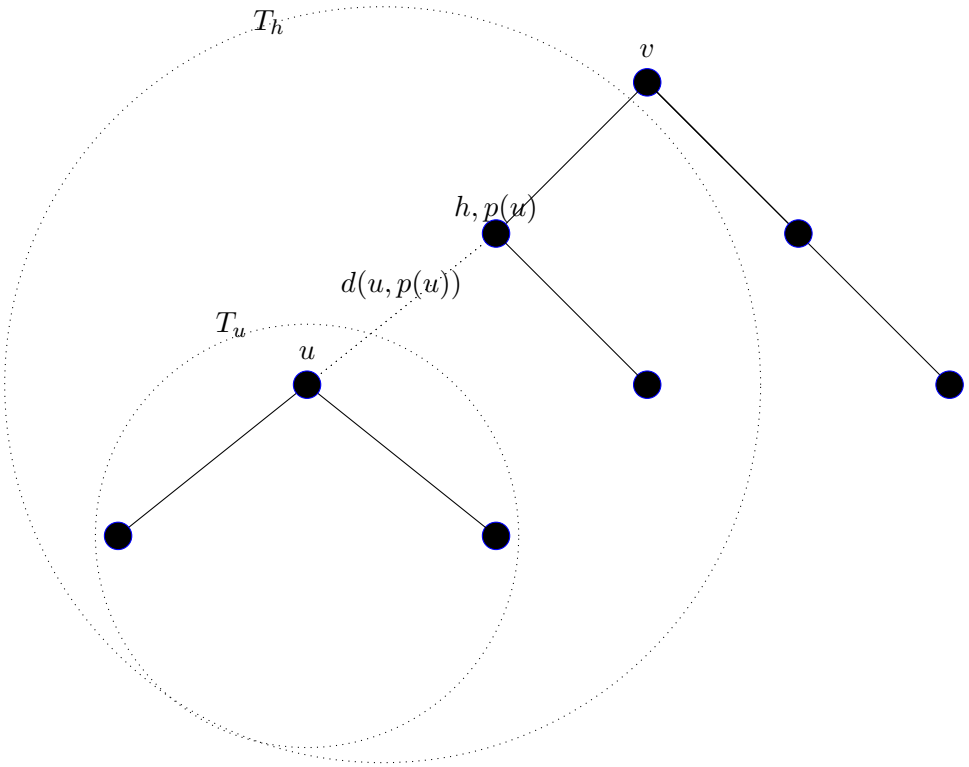

Figure 2.3: Computation of $cost_{T_h}(p(u))$ from $cost_{T_h}(u)$

Computation of (v)

We compute $cost_{T_v}(v)$ for every v by the bottom up approach as follows.

$$cost_{T_v}(v) = \sum_{u \in child(v)} (cost_{T_u}(u) + w(T_u) \cdot d(v, u)). \tag{2.2}$$

Lemma 2.0.4. *$cost_{T_v}(v)$ for all v can be computed in linear time.*

Theorem 2.0.5. *All the preprocessing steps can be computed by invoking the function preprocessing(m) and the total cost is linear.*

Chapter 3

2-median Problem

In this chapter, we start by explaining the concept of link deletion which has been used to find the 2-medians in tree networks. The cost computation for link deletion is also briefly described. A few important theorems and lemmas related to this concept are discussed. Finally, a detailed description of the 2-median algorithm (Algorithm 3) is given whose computation time is $\mathrm{O}(n \log s)$ where s is the number of leaf nodes in the tree. Also, we discuss the proof of correctness and time complexity evaluation.

3.0.1 The 2-median algorithm

Link deletion method and the concept of split edge

The link deletion method is a strategy used extensively in finding p-median of trees. Gavish and Sridhar [12] used this popular method to devise the $O(n \log n)$ algorithm to find 2-medians of tree networks. The strategy involves in removing one edge of a tree at a time. Removal of an edge decomposes the tree into two parts. Then 1-median of both these trees and their corresponding costs are computed. The edge for which the sum of the cost of 1-medians is the lowest is the optimal link or split edge, and the two 1-median vertices are the resulting 2-medians of the tree.

If in a tree T, two vertices are fixed as facility locations (say m_1 and m_2), then we can find exactly one edge that is not traversed by any vertex on its way to its closest facility. This edge is called split-edge. The deletion of this edge e partitions the tree T into two subtrees. The cost of this m_1 and m_2 is given by

$$cost_T(m_1, m_2) = \sum_{v \in T'(e)} w(v) \cdot d(v, m_1) + \sum_{v \in T''(e)} w(v) \cdot d(v, m_2) \tag{3.1}$$

It is possible to efficiently compute 2-medians of a tree by removing one edge e at a time from the tree and computing the 1-medians of the $T'(e)$ and $T''(e)$ and

checking the lowest cost as shown in (3.1). The edge e', whose removal generates the minimum cost is called the optimal split edge. The 2-medians for the tree T are the two 1-medians of $T'(e')$ and $T''(e')$. This approach is called the *link deletion method*.

3.0.2 Theorems and lemmas

For the ease of understanding we restate the following notations.

- T_1: The largest weighted subtree among subtrees rooted at children of 1-median (node m) of the tree. The root of T_1 is r_1.
- T_2: The second heaviest tree among subtrees rooted at children of 1-median (m) of the tree. The root of T_2 is r_2.
- m_1: The 1-median node of T_1.
- m_2: The 1-median node of T_2.
- π_1: The path from node m to node m_1.
- π_2: The path from node m to node m_2.
- s: The number of leaves in T.
- W: Total weight of the vertices of the input tree T.
- T_v : Subtree of T rooted at v.
- $m(T_v)$: 1-median node of T_v where T_v is the subtree of T.
- $p(u)$: Parent of node u in T.
- $T'(e)$: On the removal of e in T, it decomposes into two subtrees. The subtree that contains m is $T'(e)$.
- $T''(e)$: T-$T'(e)$

Following are two important results that support our algorithm.

Lemma 3.0.1. *Let e be an edge of T that splits the tree into two parts namely $T^{'}(e)$ and $T^{''}(e)$. If $m \in T^{'}(e)$, then $w(T^{'}(e)) \geq w(T^{''}(e))$.*

Theorem 3.0.2. *If for the tree T, an edge e in the subtree (T - T_1) is removed, then one of the two 1-medians of the trees formed by the removal of e will lie on π_1. Similarly, if an edge is removed from the heaviest subtree (T_1), then one of the two 1-medians lies in π_2.*

Proof. In this theorem, we only discuss the case when the edge is removed from $(T - T_1)$ in details. Let us assume that the edge removed in $(T - T_1)$ is incident to node m. So we have two trees $T^{'}(e)$ and $T^{''}(e)$ separated by an edge such that $T^{'}(e)$ contains T_1. Since T_1 is the heaviest subtree in $T^{'}(e)$, so by Goldman's condition, the 1-median of $T^{'}(e)$ will be in T_1. Now we argue that the 1-median would always lie on π_1 of T_1. For proving this, let us first consider only the subtree T_1 with median m_1. If we keep adding the weights to the root of T_1, then we claim that the new 1-median of the tree will be along π_1. Let the child nodes of m_1 be a and b. Since the 1-median of T_1 is m_1, so m_1 satisfies Goldman's condition. Now when an additional weights (w_1 and w_2) are introduced at the root of the tree, then depending on the weights, the position of new 1-median will change. It is to be noted that the position of the new 1-median will only move towards the root of T_1 and not towards any of the children of m_1 (Figure 3.1). The reason is that the weight is added along π_1 w.r.t to m_1. So to satisfy Goldman's condition, the 1-median has to move towards its ancestor along π_1. Since $\forall v \notin \pi_1\ w(T_1 - T_v) > w(T_1)/2$, therefore $w(T'(e) - T_v) > w(T'(e))/2$. The second part can also be explained in a similar way. □

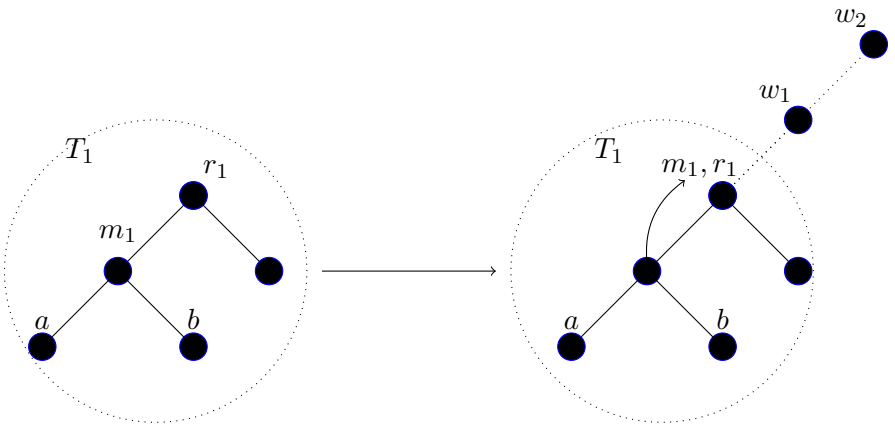

Figure 3.1: On addition of extra weights to T_1, the 1-median m_1 moves towards the root

3.0.3 The Algorithm

Following is the algorithm for finding the 2-medians of a tree. We follow the link deletion approach here where given a tree T, we remove one edge at a time. Removal of each edge e results in two trees $T'(e)$ and $T''(e)$. We then compute $cost_{T^{'}(e)}(m(T^{'}(e)))$ and $cost_{T''(e)}(m(T''(e)))$. The lowest sum of these costs gives us our 2-medians. From Theorem 3.0.2, we know that one of the two optimal 1-medians will lie on paths π_1 and π_2. Also if e_1 and e_2 lie on π_1 or π_2 such that $w(T'(e_1)) > w(T'(e_2))$, then $m(T'(e_1))$ is an ancestor of $m(T'(e_2))$.

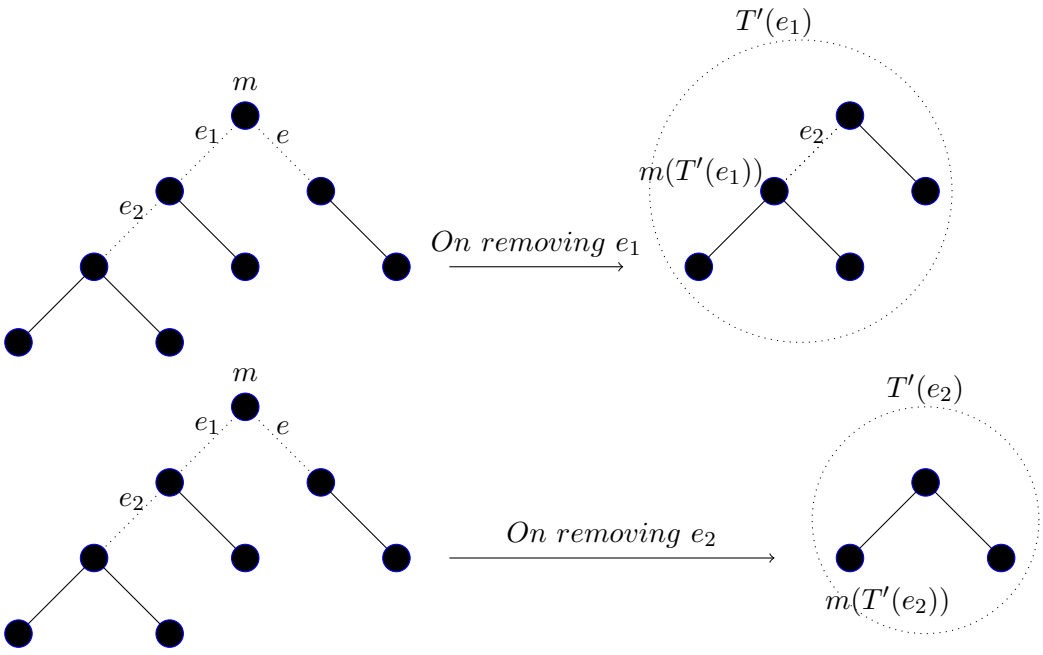

Figure 3.2: The 1-median of $T'(e_1)$ and $T'(e_2)$ after removal of e_1 and e_2 respectively

In Algorithm 3 we search for the optimal split edge in $(T - T_1)$. We process the edges in increasing order of $w(T'(e))$ using a min-heap H. The elements in the min-heap are edge and weight pairs $(e, w(T'(e)))$ such that the keys of the heap are the weights. The same algorithm is repeated to find the split edge in $T_1 \cup e'$ where $e' = (r_1, m)$. The two 1-medians that result in minimum cost is our solution.

Algorithm 3 Link deletion on $T - T_1$

1: $m_1' \leftarrow m_1$
2: $minC \leftarrow \infty$
3: $H \leftarrow empty$
4: $E' \leftarrow$ the set of edges in $(T - T_1)$ incident to m
5: PUSH$(e, w(T'(e)))$ to H, $\forall e \in E'$
6: **while** (H is not empty) **do**
7: $(e, w) \leftarrow$ POP(H)
8: **while** $(w - w(T_{m_1'}) > w/2)$ **do**
9: $m_1' \leftarrow p(m_1')$
10: Calculate $cost_{T_1}(m_1^{'})$ and $cost_{T'(e)}(m_1')$
11: $C_1 = cost_{T'(e)}(m_1')$
12: $C_2 = cost_{T''(e)}(m(T''(e)))$
13: $Cost = C_1 + C_2$
14: **if** $(Cost < minC)$ **then**
15: $minC = Cost$
16: $minM_1 = m_1'$
17: $minM_2 = m(T''(e))$
18: PUSH$(e', w(T'(e')))$ to H, $\forall e'$ such that $p(e') = e$
19: **return** $(minM_1, minM_2, minC)$

As seen in Algorithm 3, we need to compute the 1-median and the cost of 1-median for $T'(e)$ and $T''(e)$ when we remove the edge e from the tree T. The location of the 1-median of the subtree rooted at T_v, and the cost of the 1-medians are known from preprocessing steps. So C_2 (in line 12 of Algorithm 3) can be retrieved in constant time. From Theorem 3.0.2, we can observe that the 1-median for $T''(e)$ where $T_1 \in T''(e)$ lies on π_1. As the weight of $T'(e)$ increases in every iteration, so the location of the 1-median in π_1 moves towards the root of T_1. This means that if ce is a child edge of the edge e in π_1, then the 1-median for $T''(ce)$ will be the ancestor of the 1-median of $T''(e)$.

C_1 (line 11 of Algorithm 3) can be retrieved in constant time from the preprocessing steps discussed in section 2.0.3. For computation of C_1, we do the following. We know $cost_{T_1}(m_1)$ from preprocessing. From this cost, we can compute $cost_{T_1}(parent(m_1))$ and likewise the $cost_{T_1}(v)$, $\forall v \in \pi_1$ can be computed in linear time. This can be done as follows. If the cost of child of m_1' (denoted by cm_1') is known, then we can easily compute $cost_{T_1}(m_1^{'})$ i.e if we move the 1-median from $cm_1^{'}$ to $m_1^{'}$ in the tree T_1, then the cost increases by $w(T_{cm_1'}) \cdot d(cm_1', m_1^{'})$ and decreases by

$w(T_1 - T_{cm_1'}) \cdot d(cm_1', m_1')$ (explained in Figure 3.3):

$$cost_{T_1}(m_1') = cost_{T_1}(cm_1') + w(T_{cm_1'}) \cdot d(cm_1', m_1') - w(T_1 - T_{cm_1'}) \cdot d(cm_1', m_1') \tag{3.2}$$

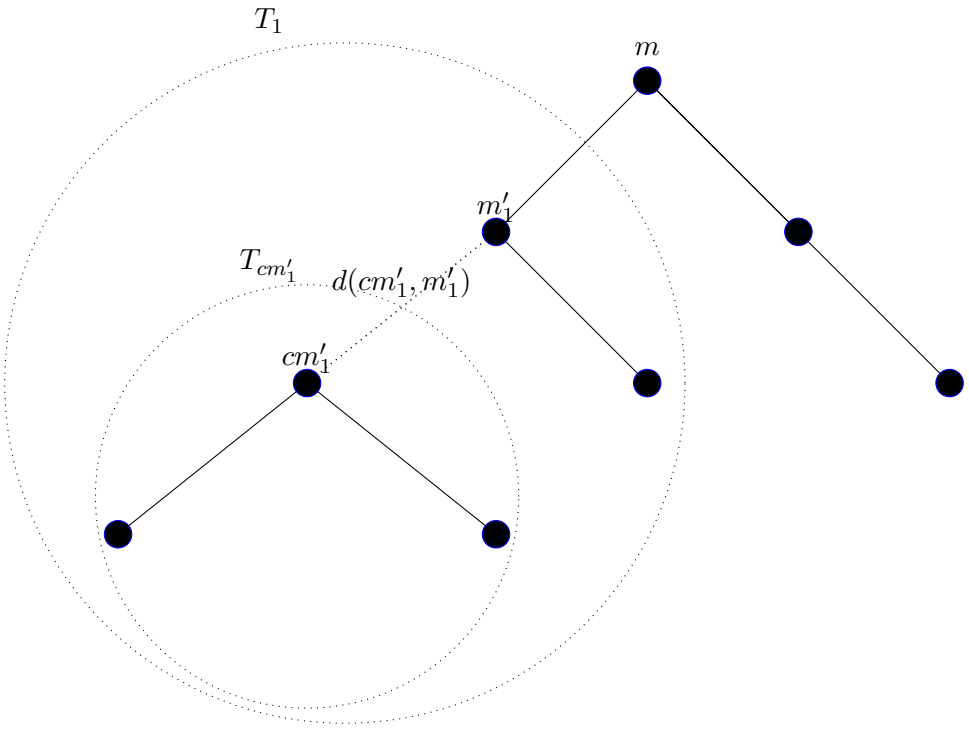

Figure 3.3: Computation of $cost_{T_1}(m_1')$ from $cost_{T_1}(cm_1')$

If v' is the root of $T''(e)$, then in order to find the cost of 1-median of $T'(e)$ we use (6) and derive the following formula (refer to Figure 3.4):

$$cost_{T'(e)}(m_1') = cost_{T_1}(m_1') + cost_T(m) - cost_{T_1}(r_1) - w(T_1) \cdot d(r_1, m) - cost_{T''(e)}(v') - w(T''(e)) \cdot d(v', m) + w(T - T''(e) - T_1) \cdot d(m, m_1') \tag{3.3}$$

3.0.4 Proof of Correctness

The algorithm considers computing the cost for 1-medians of the two trees formed by removing an edge from the tree. The edges of T_1 are removed first followed by the edges of T-T_1. As the edges from T_1 popped from H are in the increasing order of the weights of their corresponding trees (T'(e)), so the 1-medians will keep monotonically

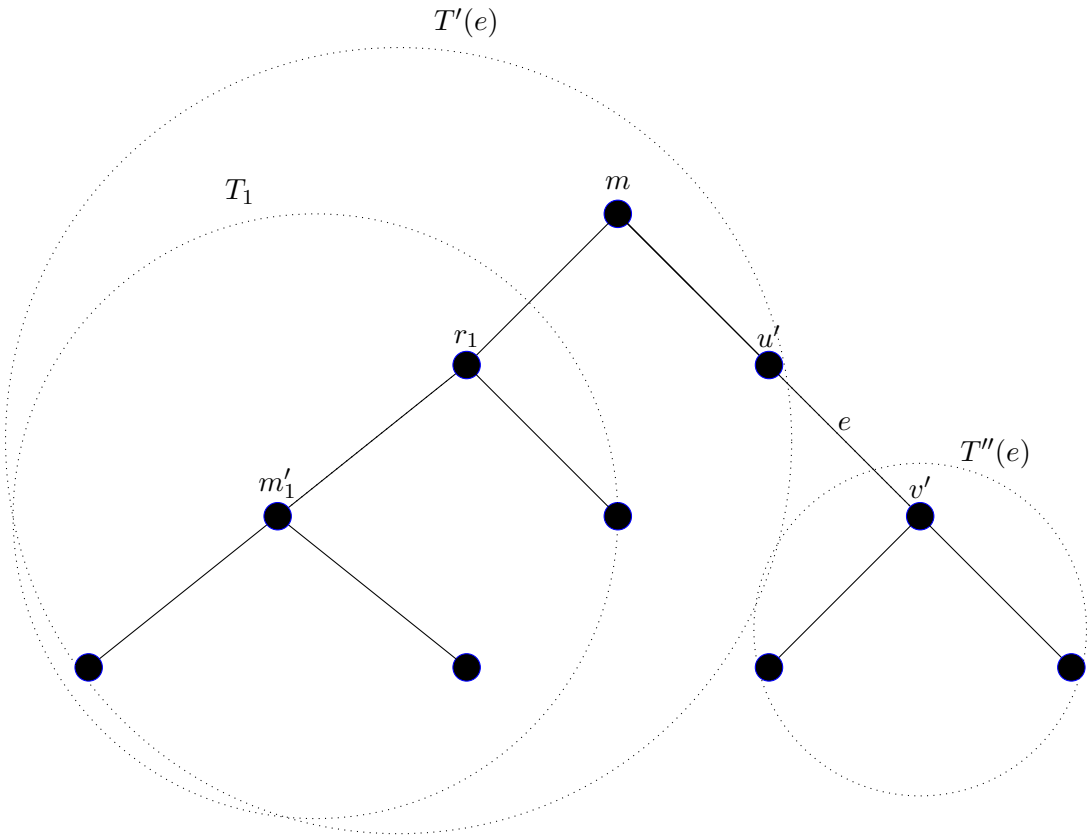

Figure 3.4: Computation of $cost_{T_1}(m'_1)$ from $cost_{T_1}(cm'_1)$

moving towards the root m along π_2 till the Goldman's condition is satisfied (step 8 and 9). We know that optimal 1-median lies on the path π_1 as stated by Theorem 3.0.2. Since we start the algorithm with edges incident on m in $(T - T_1)$, so the search for the 1-median starts at the end-point of π_1 (which is m'_1) and with the popping of edges from H, the 1-medians move towards the root m (which is the other end-point of π_1). Thus we can say that all the possible candidates for optimal 1-median are being considered and the best of these candidates are found out to give us the two optimal 1-medians.

3.0.5 Time Complexity

Now we analyze the time complexity of the algorithm. As described above, the preprocessing takes $O(n)$ time. In the while loop of step 6 of the algorithm, we push and pop from a heap H until all the edges have been processed. As the number of leaves in the tree is s, so at any point of time in the algorithm, the heap would not have more than s elements in it. The reason for this is that we push the child edges into the heap only after the parent edge has been popped out from the heap. So if π

is a path in the tree, then the heap would only contain one edge of π at any point in time. The pushing or popping thus takes $O(\log s)$ time. Each link or edge is pushed and popped from the heap exactly once, and hence the whole process of popping and pushing takes $O(n \log s)$ time. Also for removal of each link, we need to recompute the medians in the trees. Every line in the inner while loop in step 8 has constant time complexity. The while loop might iterate for n times for an edge but overall the number of iterations of this while loop will be no more than $O(n)$. The reason for this is that the 1-median in each consecutive iteration will either remain the same or will move to its ancestor. It will never move towards its child. The other parts of the algorithm are either $O(1)$ time or involve preprocessing steps which result in $O(1)$ time. So the overall time complexity of the algorithm is $O(n \log s)$.

3.0.6 Conclusion

In this chapter, we used the link deletion method to design a 2-median algorithm for tree networks in $(n \log s)$ time where s is the number of leaf nodes of the tree. If s is known to be constant, then our algorithm will be linear. Finding a linear time solution for arbitrary s is still an open problem. Link deletion method has $\Omega(n \log n)$ lower bound [18]. So a different approach is needed to solve this problem in $o(n \log n)$ time.

Chapter 4

Generalized 2-median problems on trees

In Ku et al.[18] authors consider two generalizations of the 2-median problem on trees, where they show that the generalizations can also be solved in $O(n \log n)$ time. In this chapter, we show that these generalizations can be solved in $O(n \log s)$ time, where s is the number of leaves of the input tree.

4.0.1 The 2-median problem with eccentricity constraint

Consider a 2-median problem on trees with non-negative vertex weights where we are given an additional *eccentricity constraint* L_e. The goal is to find two vertices r_1 and r_2 of T that minimize

$$\sum_{v \in V} w(v) \cdot d(v, \{r_1, r_2\})$$

under the eccentricity constraint

$$\max_{v \in V} d(v, \{r_1, r_2\}) \leq L_e.$$

In other words, we want to find two points of the tree that minimize the 2-median objective function under the condition that their 2-median objective function value is no worse than L_e. Note that this is indeed a generalization of the 2-median problem since by choosing L_e to be large, we get the 2-median problem.

Authors in [18] present an $O(n \log n)$ algorithm for this problem. Moreover, they obtain the following result.

Theorem 4.0.1 ([18])**.** *The generalized 2-median problem with eccentricity constraint on trees can be solved in $O(n + t_x)$ time, where t_x is the time needed for finding all $m(T'(e))$, $e \in E$.*

The proof of this theorem follows the idea that the two nodes r_1 and r_2 satisfying the eccentricity constraint will lie on the path$(m(T'(e))$,$c(T'(e)))$ and $(m(T''(e))$,$c(T''(e)))$ where $c(T'(e))$ is the centroid of $T'(e)$. Here e is the split edge (or the edge that is removed). To find the centroid of a tree, it takes linear time [18]. Hence the theorem.

As a byproduct, our Algorithm 3 for the 2-median problem finds all $m(T'(e))$'s. Hence we have the following theorem.

Theorem 4.0.2. *The generalized 2-median problem with eccentricity constraint on trees with non-negative weights can be solved in $O(n \log s)$ time.*

4.0.2 The 2-median problem with distance constraint

In this section, we consider a generalization of the 2-median problem on trees with non-negative weights where we are given an additional *distance constraint* L_d. Now the goal is to find two vertices z_1 and z_2 of T that minimize

$$\sum_{v \in V} w(v) \cdot d(v, \{z_1, z_2\}) \tag{4.1}$$

under the condition

$$d(z_1, z_2) \leq L_d.$$

In other words, we want to find two points of the tree that minimize the 2-median objective function, such that their distance is no greater than L_d. We will show that the vertex variant of the problem (i.e. where only vertices of the graph are considered for 2-medians) can be solved in $O(n \log s)$ time.

Consider the link-deletion approach. For an edge e, let $z'(e)$ and $z''(e)$ denote the respective vertices of $T'(e)$ and $T''(e)$ that minimize (4.1) under the constraint $d(z'(e), z''(e)) \leq L_d$. Instead of considering $z'(e)$ and $z''(e)$ for all edges e, it turns out we can restrict our search space on the following two sets. Given an edge $e = (v, p(v))$, let y_e denote a vertex on $path(v, m(T''(e)))$ which is farthest from $m(T'(e))$ but still satisfies the distance condition $d(m(T'(e)), y_e) \leq L_d$. Then we define the set

$$S_1 := \{(m(T'(e)), y_e) \colon e \in E\}.$$

Next, for every vertex $y \neq m$, we define vertex x_y as follows. If $y \in T_1$, then x_y is the vertex farthest away from y but within distance L_d, which is on $path(y, m) \cup path(m, m(T_2))$. Otherwise, if $y \notin T_1$, then x_y is the vertex farthest away from y but within distance L_d, which is on $path(y, m) \cup path(m, m(T_1))$. Then we define the set

$$S_2 := \{(y, x_y) \colon y \in V\}.$$

The size of both S_1 and S_2 is $O(n)$
Note: In this paper $BS(x_1, x_2)$ will be used to denote the bisector of points x_1 and x_2 in T

Lemma 4.0.3 ([18]). *For every instance of the vertex version of the 2-median problem with distance constraint on trees, there exist an optimal solution which is an element of $S_1 \cup S_2$.*

In both the sets S_1 and S_2, there is a common property. This property is that y_e in S_1 is the furthest vertex from $m(T'(e))$ and x_y is the furthest vertex from any $y \in T_1$ and within the distance L_d. This gives us the idea that not only does this satisfy the distance constraint but also these are the nearest possible vertices to $m(T''(e))$ (in S_1) and $m(T_2)$ (in S_2). So we can conclude that these vertices will contribute to minimum cost in comparison to all those set of vertices which lie within a range of L_d. Thus the theorem.

Lemma 4.0.4 ([21]). *After an $O(n)$ time preprocessing, the 2-median objective function $\sum_{v \in V} w(v) \cdot d(v, \{x_1, x_2\})$ for any $x_1, x_2 \in V$ can be computed in $O(1)$ time if the bisector $BS(x_1, x_2)$ is given.*

The proof follows from the idea of the tree-marker problem [18].

Now we will show how all pairs from $S_1 \cup S_2$ and their bisectors can be found in $O(n \log s)$ time. Together with Lemma 4.0.3 and 4.0.4, this implies that the vertex version of 2-median problem with distance constraint on trees with non-negative weights can be solved in $O(n \log s)$ time.

We start by describing how elements of S_1 and their bisectors can be found. To do so, we create a tree T^*, which is obtained from T as follows. We remove each sequence of (non-root) vertices with degree two and replace it with one edge of length which is a sum of all the edges removed. See Figure 4.1 for an illustration. Such newly created edges of T^* are denoted by $e_1^*, \ldots, e_k^*$. And every such edge e_i^* corresponds to a path in T denoted by π_i. Note that the number of vertices in T^* is less than $2s$.

We start by explaining how $(m(T'(e)), y_e) \in S_1$, for $e \in E$, and their bisectors can be found. First, we find $(m(T'(e)), m(T''(e))$ for all $e \in E$. That can be done in $O(n \log s)$ time using Algorithm 3. Now we do the pre-order depth first traversal of T and T^*, in parallel. During this traversal, we maintain the path π from the root m of T^* to the current position in T^*, in an array. Whenever we reach a vertex of T which is $m(T''(e))$ for some e, we do a binary search on the path connecting $m(T'(e))$ and m to find a point on π which is $L - d(m(T'(e)), m)$ away from m. (We discard cases when $d(m(T'(e)), m) \geq L$.) The binary search takes $O(\log s)$ time. If such a

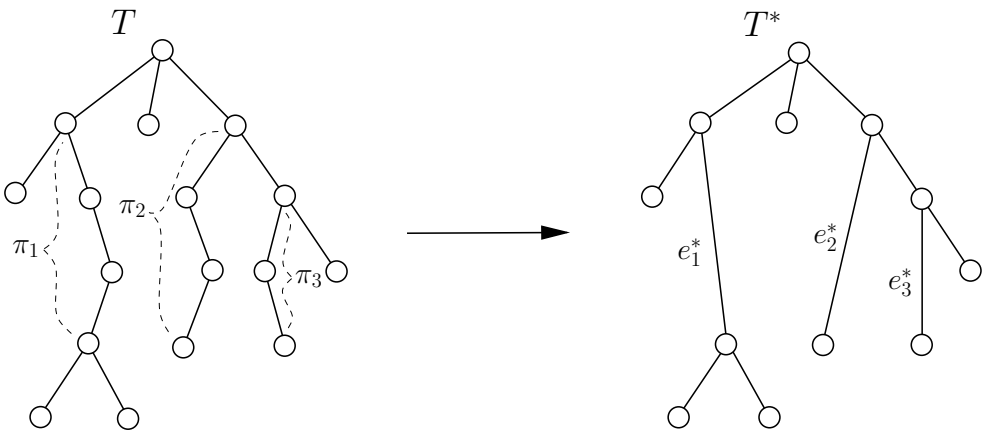

Figure 4.1: Transformation of T to T^*.

point falls on an edge $(v, p(v))$ which is not e_i^* for some i, then $p(v)$ is the y_e we are looking for. Otherwise, when $(v, p(v)) = e_i^*$, we need to find a vertex on π_i which is furthest away from $p(v)$ but still within $\ell = L - d(m(T'(e)), m) - d(m, p(v))$. In that case, we create and save the triple (e, π_i, ℓ) for finding y_e. We will do the search in π_i after we finish the depth-first traversal to do the search more efficiently. So, for each $e \in E$, it takes $O(\log s)$ time to find either y_e or (e, π_{i_e}, ℓ_e) Hence it takes $O(n \log s)$ time to finish the depth-first traversal.

In order to show how y_e's can be efficiently obtained from (e, π_{i_e}, ℓ_e)'s, we need to describe in more details the way we store (e, π_{i_e}, ℓ_e)'s during our depth-first traversal. To do so, we define the *path partition* of T. A maximal sequence of vertices in our pre-order depth-first traversal which contains only one leaf, which is on the end of the sequence, we call a *path* of T. See Figure 4.2 for an example of the path partition of a tree.

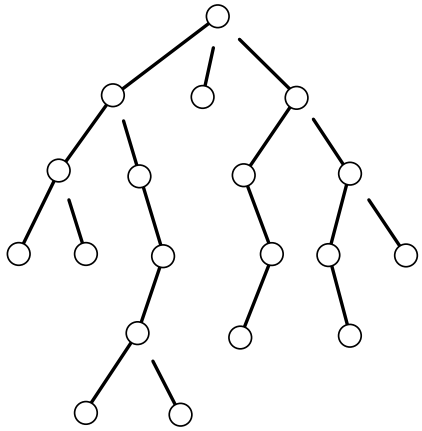

Figure 4.2: A tree partitioned into paths

For every usual path t (from root to leaf) in path partition and every edge e_i^*, we create a list L_{ti} in which we save triples (e, π_i, ℓ_e) in the order we generate them so that a new triple is added to the end of the corresponding list. Note that for every edge e_i^*, there are exactly s paths in every path partition; some of which may be empty. Hence, for every e_i^*, there will be s lists $L_{t_j i}$, $j = 1, 2, \ldots, s$. The key observation is the fact that in every list $L_{t_j i}$, the triples (e, π_i, ℓ_e) will be ordered so that the values ℓ_e are in increasing order. This is true since in every path, if $m(T''(e_1))$ is visited before $m(T''(e_2))$, then $m(T''(e_1))$ is an ancestor of $m(T''(e_2))$. And in that case $m(T'(e_2))$ is an ancestor of $m(T'(e_1))$, which implies that $L - d(m(T'(e_1)), m) \leq L - d(m(T'(e_2)), m)$.

For every fixed e_i^* we have lists of triples $L_{t_1 i}, \ldots, L_{t_s i}$. If the collection of all triples from these lists are sorted on the third components ℓ_e, we can process them and find corresponding y_e's by scanning π_i once from top to bottom. And we can sort the triples by iteratively popping the first elements of the lists $L_{t_1 i}, \ldots, L_{t_s i}$ and putting them on a heap. Each heap operations take $O(\log s)$ time. We repeat this process for all edges e_i^* and corresponding paths π_i. Scanning all π_i's takes $O(n)$ time, there is $O(n)$ triples (e, π_i, ℓ_e) in total, hence processing all of them takes $O(n \log s)$ time. Therefore, S_1 can be computed in $O(n \log s)$ time.

Now we can find bisectors $BS(x_1, x_2)$, $(x_1, x_2) \in S_1$, using the same approach. Namely, we do depth-first traversal to scan all y_e's and do the binary search same as before, but now L is replaced with $d(m(T'(e)), y_e)/2$. We assume that $BS(m(T'(e)), y_e)$ is on $path(m, y_e)$, the case when it is on the $path(m, m(T'(e)))$ can be dealt using the approach analogous to Algorithm 3. Under such assumption, if y_{e_1} is an ancestor of y_{e_2}, then $m(T'(e_2))$ is an ancestor of $m(T'(e_1))$. And then it must be that $BS(m(T'(e_1)), y_{e_1})$ is an ancestor edge of $BS(m(T'(e_2)), y_{e_2})$, see Figure 4.3. There-

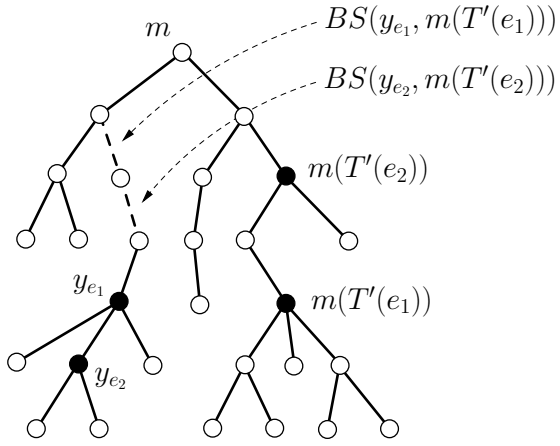

Figure 4.3: Example of bisector positions along a path

fore, the key property that the triples in lists L_{ti} are sorted on the third component, will be preserved. So S_1 and its bisectors can be found in $O(n \log s)$ time.

Elements of set S_2 and its bisectors can be found using the same approach in $O(n \log s)$ time. We leave details to the reader.

Theorem 4.0.5. *The vertex version of 2-median problem with distance constraint on trees with non-negative weights can be solved in $O(n \log s)$ time.*

4.0.3 Conclusion

We have designed $O(n \log s)$ solutions to both the problems thereby improving the $O(n \log n)$ solutions in [18]. The proposed algorithms applied the ideas developed in Chapter 3. We have also shown that an arbitrary network of size n with s leaves could be transformed into an almost equivalent tree network of size at most $2s$. The algorithms are still link-deletion based. Finding linear time solutions to these problems are left as open problems.

Chapter 5

Conclusion

Facility location problem in general graph is NP-hard. So polynomial time algorithms for p-median, p-cover and p-center problems could not be found. However in simpler graphs like trees, polynomial time algorithms are possible.

We have described an $O(n \log s)$ algorithm for the 2-median problem in tree networks of size n where s represents the number of leaf nodes. This problem has $\Omega(n \log n)$ lower bound in the worst case for any method based on link-deletion. There are many optimal $O(n \log n)$ algorithms known for this problem. Therefore, when $s \in o(\log n)$, our algorithm is better. In a general tree, our algorithm is still $O(n \log n)$. To obtain an $o(n \log n)$ algorithm for the 2-median problem in general tree is still open.

Bibliography

[1] V. Auletta, D. Parente and G. Persiano, "Dynamic and static algorithms for optimal placement of resources in a tree," *Theoretical Computer Science* vol. 165, pp. 441–461, 1996.

[2] V. Auletta and D. Parente, "Placing resources on growing line," *Journal of Algorithms,* vol. 26, pp. 87–100, 1998.

[3] R. Benkoczi, "Cardinality Constrained Facility location problem in trees," PhD Thesis, Simon Fraser University, Canada 2004.

[4] R. Benkoczi and B.K. Bhattacharya, "A new template for solving p-median problems for trees in sub-quadratic time," *European Symposium on Algorithms,* vol. 3669, pp. 271—282, 2005.

[5] R. Benkoczi, B. Bhattacharya, and D. Breton, "Efficient computation of 2-medians in a tree network with positive/negative weights," *Electronic Notes in Discrete Mathematics,* vol. 15, pp. 39–42, 2003.

[6] D. Breton, "Facility location problems on trees," M.Sc.Thesis, Simon Fraser University, 2002.

[7] R.E. Burkard, E. Cela and H. Dollani, "2-medians in trees with pos/neg-weights," *Discrete Applied Mathematics,* vol. 105, pp. 51—71, 2001.

[8] R.E. Burkard and J. Krarup, "A Linear Algorithm for the Pos/Neg-Weighted 1-Median Problem on a Cactus," *Computing,* vol. 60, pp. 193–215, 1998.

[9] R. Chandrasekaran and A. Tamir, "An $O((n\log(p))^2)$ algorithm for continuous p-center on a tree," *SIAM Journal of Algorithms in Discrete Mathematics,* vol. 1, pp. 370–375, 1980.

[10] M. Chrobak, L.L. Larmore and W. Rytter, "The k-median problem for Directed Trees," in *Proc. 26th International Symposium of Mathematical Foundations of Computer Science (MFCS'01)*, no. 136 in lecture notes of Computer Science, pp. 260–271, 2001.

[11] C.B. Djiba, M. Balde, B.M. Ndiaye, R.M. Faye and D. Seck, " Breakdown Mechanic Location Problem," *Applied Mathematics,* vol.3, no.5, 2012.

[12] B. Gavish and S. Sridhar, "Computing the 2-median on tree networks in $O(n\log n)$ time," *Networks* vol. 26, pp. 305, 1995.

[13] A.J. Goldman, "Optimal center location in simple networks," *Transactions on Computational Science,* vol. 5, pp. 212–221, 1971.

[14] S.L Hakimi, "Locating new facilities in competitive environment," *European Journal of Operational Reasearch,* vol. 12, pp. 29–35, 1983.

[15] S.L. Hakimi, "Optimal locations of switching centers and the absolute centers and medians of a graph," *Operations Research* vol. 12, pp. 450–459, 1964.

[16] O. Kariv and S.L Hakimi, "An algorithmic approach to network location problems. I: The p-medians," *SIAM Journal of Applied Mathematics,* vol. 37, pp. 513—538, 1979.

[17] O Kariv and S.L Hakimi "An algorithmic approach to network location problems. II: The p-medians," *SIAM Journal of Applied Mathematics,* vol. 37, pp. 539—560, 1979.

[18] C.Y. Chan, S.C. Ku, C.J. Lu and B.F. Wang, "Efficient algorithms for two generalized 2-median problems and the group median problem on trees," in *Theoretical Computer Science,* vol. 410, pp. 867–876, 2009.

[19] F. Ndiaye, B.M. Ndiaye, and I. Ly, "Application of the p-median problem in school allocation," *American Journal of Operations Research,* vol. 2, pp. 253—259, 2012.

[20] A. Tamir, "An $O(pn^2)$ algorithm for the p-median and related problems on tree graphs," *Operations Research Letters,* vol. 19, pp. 59–64, 1996.

[21] B.F. Wang, S.C. Ku and K.H. Shi, "Cost optimal parallel algorithms for the tree bisector problem and applications," *IEEE Transactions on Parallel and Distributed Systems* vol. 12, pp. 888–898, 2001.

[22] Wikipedia contributors. "Graph center." Wikipedia, The Free Encyclopedia. Wikipedia, The Free Encyclopedia, 31 Jan. 2016. Web. 15 Mar. 2017.