8-2017

Geometric Facility Location Problems on Uncertain Data

Jingru Zhang
Utah State University

GEOMETRIC FACILITY LOCATION PROBLEMS ON UNCERTAIN DATA

by

Jingru Zhang

A dissertation submitted in partial fulfillment
of the requirements for the degree

of

DOCTOR OF PHILOSOPHY

in

Computer Science

Approved:

Haitao Wang, Ph.D. Major Professor	Minghui Jiang, Ph.D. Committee Member
Curtis Dyreson, Ph.D. Committee Member	Kyumin Lee, Ph.D. Committee Member
Rose Qingyang Hu, Ph.D. Committee Member	Mark R. McLellan, Ph.D. Vice President for Research and Dean of the School of Graduate Studies

UTAH STATE UNIVERSITY
Logan, Utah

2017

ABSTRACT

Geometric Facility Location Problems on Uncertain Data

by

Jingru Zhang, Doctor of Philosophy

Utah State University, 2017

Major Professor: Haitao Wang, Ph.D.
Department: Computer Science

Facility location, as an important topic in computer science and operations research, is concerned with placing facilities for "serving" demand points (each representing a customer) to minimize the (service) cost. In the real world, data is often associated with uncertainty because of measurement inaccuracy, sampling discrepancy, outdated data sources, resource limitation, etc. Hence, problems on uncertain data have attracted much attention.

In this dissertation, we mainly study a classical facility location problem: the k-center problem and several of its variations, on uncertain points each of which has multiple locations that follow a probability density function (pdf). We develop efficient algorithms for solving these problems. Since these problems more or less have certain geometric flavor, computational geometry techniques are utilized to help develop the algorithms. In particular, we first study the k-center problem on uncertain points on a line, which is aimed to find k centers on the line to minimize the maximum expected distance from all uncertain points to their expected closest centers. We develop efficient algorithms for both the continuous case where the location of every uncertain point follows a continuous piecewise-uniform pdf and the discrete case where each uncertain point has multiple discrete locations each associated with a probability. The time complexities of our algorithms are nearly linear and match those for the same problem on deterministic points. Then, we consider the one-center problem (i.e., $k = 1$) on a tree, where each uncertain point has multiple locations in the tree and we want to compute a center in the tree to minimize the maximum expected distance from it to all uncertain

points. We solve the problem in linear time by proposing a new algorithmic scheme, called the refined prune-and-search. Next, we consider the one-dimensional one-center problem of uncertain points with continuous pdfs, and the one-center problem in the plane under the rectilinear metric for uncertain points with discrete locations. We solve both problems in linear time, again by using the refined prune-and-search technique. In addition, we study the k-center problem on uncertain points in a tree. We present an efficient algorithm for the problem by proposing a new tree decomposition and developing several data structures. The tree decomposition and these data structures may be interesting in their own right. Finally, we consider the line-constrained k-center problem on deterministic points in the plane where the centers are required to be located on a given line. Several distance metrics including L_1, L_2, and L_∞ are considered. We also study the line-constrained k-median and k-means problems in the plane. These problems have been studied before. Based on geometric observations, we design new algorithms that improve the previous work. The algorithms and techniques we developed in this dissertation may find other applications as well, in particular, on solving other related problems on uncertain data.

(177 pages)

PUBLIC ABSTRACT

Geometric Facility Location Problems on Uncertain Data

Jingru Zhang

In this dissertation, we study several facility location problems on uncertain data. We mainly consider the k-center problem and many of its variations. These are classical problems in computer science and operations research. These problems on deterministic data have been studied extensively in the literature. We consider them on uncertain data because data in the real world is often associated with uncertainty due to measurement inaccuracy, sampling discrepancy, outdated data sources, resource limitation, etc. Although we focus on the theoretical study, the algorithms developed in this dissertation may find applications in other areas such as data clustering, wireless sensor locations, object classification, etc.

In our problems, the input consists of uncertain points each of which has multiple locations following a probability density function (pdf). Specifically, we first study the k-center problem on uncertain points on a line to compute k centers to minimize the maximum expected distance from all uncertain points to their expected closest centers. We consider two cases of the uncertainty: the continuous case and the discrete case. In the continuous case, the location of every uncertain point follows a continuous piecewise-uniform pdf, whereas in the discrete case, each uncertain point has multiple discrete locations each associated with a probability. We then consider the one-center problem (i.e., $k = 1$) on a tree, where each uncertain point has multiple discrete locations in the tree and we want to compute a center in the tree to minimize the maximum expected distance from it to all uncertain points. Next, we consider the one-dimensional one-center problem of uncertain points with continuous pdfs, and the one-center problem in the plane under the rectilinear metric for uncertain points with discrete locations. In addition, we study the more general k-center problem on uncertain points in a tree. Finally, we consider the line-constrained k-center problem on deterministic points in the plane with the constraint that the centers are required to be located on a given line.

Several distance metrics including L_1, L_2, and L_∞ are considered. We also study the line-constrained k-median and k-means problems in the plane.

Based on interesting problem modeling and observations, we develop efficient algorithms for solving these problems with the help of computational geometry techniques. Some of our algorithms have time complexities either linear or nearly linear. Others almost match those for the same problems on deterministic data or improve the previous work. The algorithms, data structures, and techniques developed in this dissertation may be used to solve other related problems as well.

ACKNOWLEDGMENTS

Over the past four and half years I have received support and encouragement from a number of individuals.

Firstly, I would like to express my sincere gratitude to my advisor Dr. Haitao Wang for the continuous support of my Ph.D study and related research. His guidance helped me a lot in my courses, research and writing this thesis. Thank you for the advice, supports, and willingness that allowed me to pursue research on topics. Thank you for all of the meetings and discussion over the years that check in on me so that I stayed on the right path. I could not have imagined having a better advisor and mentor for my Ph.D study, and this dissertation would truly have never been completed without your support. Thank you.

Besides my advisor, I would like to thank the rest of my thesis committee: Dr. Minghui Jiang, Dr. Curtis Dyreson, Dr. Kyumin Lee, Dr. Rose Qinghua Hu and Dr. Tam Chantem for their insightful comments and constructive suggestions, but also for questions which encouraged me to widen my research from various perspectives. I also thank Mr. Shimin Li for inspiring technical discussions. In addition, my sincere thanks go to the staff in Department of Computer Science for their administrative support. I am grateful to them.

I wish to express my full thanks to my families, my father, my mother and my husband. Without their love, support and understanding, I could not have gone through the doctoral program overseas. Also, thank you for the great comfort during that period I was suffering from insomnia. You are forever the source of my joy and happiness, and I have been appreciating that greatly.

Last but not least, thanks to all of my friends. I shall never forget all delightful moments in my life together with you. Also, I am really grateful to your support as I have needed a listening ear and met troubles.

This work was sponsored in part by the National Science Foundation through Grant CCF-1317143.

Jingru Zhang

CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

In this dissertation, we propose and study several facility location problems on uncertain data. In particular, we consider the k-center and k-median problems and many of their variations. These are classical problems in many areas, such as combinatorial optimization, operations research, computational geometry, etc. Most of our problems have geometric flavor and thus computational geometry techniques have been extensively used to tackle these problems. On the other hand, besides working on input data that are deterministic or certain, many of our problems involve data that may be uncertain.

In this chapter, we first briefly discuss the topics of computational geometry and uncertain data because they are closely related to the problems studied in this proposal as well as the techniques we used to tackle these problems. Then, we give an overview on the problems we studied in this proposal. Finally, we outline the proposal.

1.1 Computational Geometry

Computational geometry is a branch of computer science concerned with the design, analysis, and implementation of efficient algorithms for solving problems by exploiting their geometric structures. Geometric structures are often described in terms of elementary geometric objects such as points, lines, curves, polygons, polyhedra and surfaces. Computational geometry techniques and algorithms play a significant role in practical applications since many geometric problems originate from important applied areas, e.g., computer-aided design, computer graphics, computer vision, geographic information systems, image processing, intelligent transportation systems, medicine, military operations, pattern recognition, plant and facility layout, robotics, statistics, and very-large-scale integration design. Computational geometry also has strong connections with other areas (e.g., graph algorithms, combinatorial optimization, operations research, etc.), since it not only draws diverse ideas and techniques from these areas, but also

offers high-level formulations, general frameworks and paradigms to enrich such areas. Refer to [1–4] for several great books on computational geometry.

1.2 Uncertain Data

In the real world, data is often associated with uncertainty because of measurement inaccuracy, sampling discrepancy, outdated data sources, resource limitation, etc. Recently, due to the observation that many real-world measurements are inherently accompanied with uncertainty, problems on uncertain data have attracted dramatically increasing amount of attention. This is especially true due to the wide deployment of sensor monitoring infrastructure and increasing prevalence of technologies, such as data integration and cleaning, e.g., [5, 6]. Hence, problems with uncertain data have been studied extensively, e.g., in the areas of computational geometry and database [5–16].

Two models are commonly used for data uncertainty: the *existential* model (or tuple model) [11, 12, 16] and the *locational* model (or attribute model) [7, 8, 15]. In the existential model, each uncertain point has a specific location but its existence is uncertain, following a given probability density function (pdf). In the locational model, each uncertain point always exists but its location is uncertain and follows a probability density function (pdf). Several problems studied in this proposal that involve undertain data belong to the locational model. In fact, from the algorithm design point of view, the existential model is a special case of the locational model.

1.3 Problem Overview

In this dissertation, we mainly consider several facility location problems, where the input usually contains points on a line, a tree, or in the plane. The input of these problems may contain uncertain data seen as demand points each subjected to the locational model. The goal is to develop efficient algorithms and data structures to compute the optimal placement of facilities for "serving" these demand points to minimize the total costs. We briefly introduce these problems below. Note that these facility location problems on deterministic or certain data can be considered as special cases of our problems on uncertain data. The techniques we developed on solving these problems may have many other applications as well.

1.3.1 The One-Dimensional k-Center Problem on Uncertain Data

The one-dimensional k-center problem is a classical geometrical problem that has been extensively studied on certain (or deterministic) points. In this dissertation, we study this problem on uncertain points on a real line where each uncertain point is specified by its probability density function (pdf) which is a piecewise-uniform function (i.e., a histogram). The goal is to find a set Q of k points on the line to minimize the maximum expected distance from all uncertain points to their expected closest points in Q. We present an efficient algorithm for this k-center problem on uncertain points with the continuous pdf (a piecewise-uniform function). In addition, we give a better algorithm for the discrete case of this problem where each uncertain point has the discrete pdf. The running times of our algorithms almost match the current best algorithms for the same k-center problems on deterministic data. Refer to Chapter 2 for the details on the problem definitions and our results on this topic.

1.3.2 The One-Center Problem of Uncertain Points on Tree Networks

The second problem we study is to compute the center of uncertain points on tree networks. In this problem, we are given a tree T and n uncertain points each of which has m possible locations on T associated with probabilities. The goal is to find a point x^*, i.e, the center, on T such that the maximum expected distance from x^* to all uncertain points is minimized. To the best of our knowledge, this problem has not been studied before, although the same problem on deterministic data has been solved in linear time. To solve this problem, we develop a refined prune-and-search technique that solves the problem in linear time, and the result is clearly optimal. Refer to Chapter 3 for the details on the problem definitions and our results on this topic.

1.3.3 The One-Center of Uncertain Data on the Real Line

The third problem we consider is to compute the center of uncertain points on a line where each follows a continuous piecewise-uniform pdf. This is actually the special case with $k = 1$ of the first problem—the one-dimensional k-center problem on uncertain data, and can be solved in the super-linear time by the algorithm presented in Chapter 2. Based on the refined prune-and-search technique, we design an improved algorithm to

solve this special case in the linear time, which is optimal. Refer to Chapter 4 for the details on the problem definitions and our results on this topic.

1.3.4 The Rectilinear Center of Uncertain Points in the Plane

We consider the one-center problem on uncertainty data in the Euclidean space under the rectilinear or L_1 distance metric, where every uncertain point has m possible locations in the plane associated with probabilities. The goal is to find a point q^* in the plane which minimizes the maximum expected rectilinear distance from it to all uncertain points. Such a point q^* is called a rectilinear center. We present a linear-time algorithm that solves the problem. Refer to Chapter 5 for the details on the problem definitions and our results on this topic.

1.3.5 The k-Center Problem of Uncertain Points on Tree Networks

We study a more general k-center problem of uncertain points in a tree T. To solve this problem, we first present an algorithm for a coverage problem of uncertain points on a tree: Given a *covering range* λ, the problem is to find a minimum number of points (centers) on T to build facilities for serving (or covering) these uncertain points in the sense that the expected distance from each uncertain point to at least one center is no more than λ. This is actually the decision version (or the dual version) of the k-center problem. We develop an $O(|T| + M \log^2 M)$ time algorithm for this coverage problem, where M is the total number of all locations of all uncertain points in $\mathcal{P}$ (the uncertain points may have different numbers of locations) and $|T|$ is the total number of all vertices of the tree T. Using this algorithm as the decision algorithm, we further solve the k-center problem for the uncertain points on T. Refer to Chapter 6 for the details on the problem definitions and our results on this topic.

1.3.6 The Line-Constrained k-Median, k-Means, and k-Center Problems in the Plane

The k-median, k-means, and k-center problems on certain points in the plane are known to be NP-hard [17–19]. We study the line-constrained versions of these problems where the sought k facilities are required to be on a given line, and the input data in these problems are all certain. Specifically, we study the line-constrained k-median

under L_1, L_∞, and L_2^2 distance measures where the L_2^2 distance refers to the square of the L_2 distance. In the constrained k-median problem, the input includes a line L and n certain points in the plane. The objective is to find a set of k points (medians) on the given line L to minimize the total sum of the distances from all points to their closest points in the sought set. In fact, the k-median problem under the L_2^2 distance is the k-means problem. We present efficient algorithms for these problems. The time complexities of our algorithms for these "1.5-dimensional" problems almost match those of the current best algorithms for the corresponding one-dimensional problems. We also study the line-constrained k-center problem which asks for a set Q of k points (centers) on the line L such that the maximum distance from the input points to their closest points in Q is minimized. We solve this problem in $O(n \log n)$ time under three distance metrics: L_1, L_2, and L_∞. Then, we consider its unweighted version under L_1 and L_∞ metrics. We propose a faster way to solve the problem in both metrics in $O(n)$ time. All these results are optimal. Refer to Chapter 7 for the details on the problem definitions and our results on this topic.

1.4 Outline

The rest of the dissertation is organized as follows. In Chapter 2, we present our algorithms for the one-dimensional k-center problem on uncertain data. In Chapter 3, we discuss our result for computing the center of uncertain points on tree networks. In Chapter 4, we describe our improvements for solving the one-dimensional one-center on uncertain data. In Chapter 5, we introduce our approach to compute the rectilinear center of uncertain points in the plane. In Chapter 6, we solve the coverage problem for uncertain points on tree networks and then apply it to solve the k-center problem. In Chapter 7, we present our solutions for the line-constrained k-median, k-means, and k-center problems in the plane. Finally in Chapter 8, we give a summary of our work and discuss the possible future research directions.

CHAPTER 2

THE ONE-DIMENSIONAL K-CENTER PROBLEM ON UNCERTAIN DATA

In this chapter, we consider the k-center problem on one-dimensional uncertain data under the locational model. The results in this chapter have been published in [20, 21].

2.1 Introduction

Let $\mathcal{P} = \{P_1, P_2, \ldots, P_n\}$ be a set of n uncertain points on the x-axis, where each uncertain point P_i is specified by its pdf f_i: $\mathbb{R} \to \mathbb{R}^+ \cup \{0\}$, which is a piecewise-uniform function (i.e., a histogram), consisting of at most $m+1$ pieces (e.g., see Fig. 2.1). More specifically, for each uncertain point P_i, there are m x-coordinates $x_{i1} < x_{i2} < \ldots < x_{im}$ and $m-1$ nonnegative values $y_{i1}, y_{i2}, \ldots, y_{i,m-1}$ such that $f_i(x) = y_{ij}$ ($y_{ij} = 0$ is possible) for $x_{ij} \leq x < x_{i,j+1}$ with $0 \leq j \leq m$, and we set $x_{i0} = -\infty$, $x_{i,m+1} = +\infty$, and $y_{i0} = y_{im} = 0$. In addition, the uncertain points of $\mathcal{P}$ are independent. As discussed in [7], in practice such a histogram can be used to approximate any pdf with arbitrary precision.

Note that in some applications each uncertain point has a *discrete* pdf, that is, it could appear at one of a few locations, each with a probability. This discrete case can also be represented by the above histogram model using infinitesimal pieces around these locations, and thus the histogram model also incorporate the discrete case. In other words, the discrete case is a special case of our model.

Let L denote the x-axis. For any certain point $p \in L$, we let x_p denote its x-coordinate. The *expected distance* between p and any uncertain point P_i is defined as

$$\mathsf{Ed}(p, P_i) = \int_{-\infty}^{+\infty} f_i(x)|x - x_p|dx.$$

Let Q be a set of (certain) points on L, called *facilities*. For any uncertain point P_i, we use $\mathsf{Ed}(Q, P_i)$ to denote the smallest expected distance from P_i to all points of Q, i.e., $\mathsf{Ed}(Q, P_i) = \min_{q \in Q} \mathsf{Ed}(q, P_i)$. The facility q with $\mathsf{Ed}(q, P_i) = \mathsf{Ed}(Q, P_i)$ is called

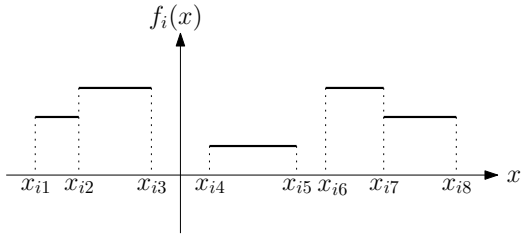

Figure 2.1. Illustrating the pdf f_i of an uncertain point P_i with $m = 8$.

the *expected closest* facility of P_i in Q, and we also say P_i is "served" by the facility q or P_i is "assigned" to q. The *k-center* problem is to find a set Q of k points on L to minimize the maximum expected distance from the uncertain points of $\mathcal{P}$ to their expected closest facilities in Q, i.e., the value $\max_{P_i \in \mathcal{P}} \mathsf{Ed}(Q, P_i)$.

In a *realization*, each uncertain point will appear at a deterministic location abiding by its pdf. We should point out that our problem definitions imply that we always assign each uncertain point P_i to its expected closest facility and we never change the assignment in any realization even through the actual location of P_i in a realization may be closer to a different facility.

For differentiation, we refer to the traditional k-center problem where each point is given in an exact position as the *deterministic version*.

In this chapter, we present an algorithm for the uncertain k-center problem and the running time is $O(mn \log mn + n \log k \log n \log mn)$. Further, for the *discrete* case where the pdf of each uncertain point of $\mathcal{P}$ is discrete, i.e., each uncertain point P_i has m possible locations, each with a probability, we have a more efficient algorithm with running time $O(mn \log mn + n \log k \log n)$. In addition, for the discrete case with $k = 1$, if the m locations of each uncertain point are given sorted, then we can solve the one-center problem in $O(mn)$ time. Considering that $\Theta(mn)$ is the input size, as will be seen later, these results almost match those for the corresponding deterministic k-center problem.

Note that our algorithms can also solve the *weighted* case where each uncertain point P_i has a nonnegative weight w_i and we consider the *weighted expected distance*, i.e., $w_i \cdot \mathsf{Ed}(q, P_i)$, from P_i to each facility q in Q. To solve the weighted problems (for both the general and the discrete cases), we only need to change each value y_{ij} to

$w_i \cdot y_{ij}$ for $1 \leq j \leq m-1$, and then simply apply our algorithms for the corresponding unweighted problems. The time complexities do not change asymptotically.

2.1.1 Related Work

The deterministic k-center (and k-median) problems are classical problems that have been extensively studied. It is well-known that the k-center problem is NP-hard even in the plane [19] and approximation algorithms have been proposed (e.g., see [22–24]). Efficient algorithm were also given for some special cases, e.g., the smallest enclosing circle and its weighed version and discrete version [25–27], the Fermat-Weber problem [28], k-center on trees [29–31]. Refer to [32] for other variations of facility location problems. The deterministic k-center in one-dimensional space is solvable in $O(n \log n)$ time [33].

The k-center problems on uncertain data in high-dimensional space have been considered. For example, approximation algorithms were given in [34] for different problem models, e.g., the assigned model that is similar to our problem model and the unassigned model which was relatively easy because it can be reduced to the corresponding deterministic problem, as shown in [34]. Other problems on clustering uncertain data were also studied and heuristic algorithms were proposed [35,36]. Other facility location problems on uncertain data under various models, e.g., the minmax regret [37–39], have also been studied (see [40] for a survey).

To the best of our knowledge, the uncertain k-center problem proposed in this chapter has not been particularly studied before.

2.1.2 Our Approach

For the deterministic one-dimensional k-center problem, there exists an optimal facility set Q such that the input points served by each facility are consecutive if we order them from left to right on L; this observation is crucial for designing the algorithms [30, 33, 41–43]. In our uncertain problem, however, since the input points of $\mathcal{P}$ are uncertain, it is not clear how to "sort" them; consequently, the algorithmic techniques used before for solving the deterministic problems are not applicable here. Instead, we use a new approach, as follows.

We first solve the *decision* problem which is to determine whether the minimized value $\max_{P_i \in \mathcal{P}} \mathsf{Ed}(Q, P_i)$ in the optimal solution is less than or equal to a given value ϵ, and if yes, ϵ is called a *feasible value*. We solve the decision problem with the following result: with $O(mn)$ time preprocessing, for any given ϵ, we can determine whether ϵ is a feasible value in $O(\log m + n \log k)$ time.

By using the above result for the decision problem, we solve the k-center problem by using parametric search [43, 44]; however, there are some issues that do not allow us to use the parametric search in [43, 44] directly and we have to make certain modifications. A useful observation discovered by us is that the expected distance $\mathsf{Ed}(p, P_i)$ is a unimodal function (i.e., first monotonically decreasing and then increasing) as p moves from left to right on L. These efforts lead to an $O(mn \log mn + n \log k \log n \log mn)$ time algorithm for the k-center problem. Further, by replacing the parametric search scheme with the randomized quicksort as in [45], we can obtain an expected $O(mn \log mn + n \log k \log n \log mn)$ time randomized algorithm that is relatively practical.

For the discrete case, we reduce the problem to finding a particular vertex in a line arrangement. By using the arrangement searching technique in [46], we can solve the discrete case in a faster way, in $O(mn \log mn + n \log k \log n)$ time. If $k = 1$ and the m locations of each uncertain point are given sorted, then we can reduce the problem to a linear programming problem and thus solve the problem in $O(mn)$ time by applying the linear time algorithm [27].

The rest of the chapter is organized as follows. In Section 2.2, we give some observations. In Section 2.3, we present our algorithm for the decision problem. In Section 2.4, we solve the k-center problem, which is referred to as the *optimization* problem. Section 2.5 presents our algorithm for the discrete case.

2.2 Observations

Consider any uncertain point P_i of $\mathcal{P}$. For any point p, its expected distance to P_i is $\mathsf{Ed}(p, P_i) = \int_{-\infty}^{+\infty} f_i(x)|x - x_p|dx$. With a little abuse of notation, we also use $\mathsf{Ed}(x_p, P_i)$ to denote $\mathsf{Ed}(p, P_i)$, but we normally consider $\mathsf{Ed}(x_p, P_i)$ as a function of x_p for $x_p \in \mathbb{R} = (-\infty, +\infty)$ as p moves on L.

A function $g : \mathbb{R} \to \mathbb{R}$ is a *unimodal function* if there exists a value x' such that

$g(x)$ is monotonically decreasing on $x \in (-\infty, x']$ and monotonically increasing on $x \in [x', +\infty)$, i.e., for any $x_1 < x_2$, $g(x_1) \geq g(x_2)$ holds if $x_2 \leq x'$ and $g(x_1) \leq g(x_2)$ holds if $x' \leq x_1$.

We assume the m coordinates $x_{i1}, \ldots, x_{im}$ of P_i are given sorted. We have the following lemma, which is crucial to our algorithm.

Lemma 2.2.1. *The function* $\mathsf{Ed}(x_p, P_i)$ *for* $x_p \in \mathbb{R}$ *is a unimodal function and can be explicitly computed in* $O(m)$ *time. More specifically,* $\mathsf{Ed}(x_p, P_i)$ *is a parabola (of constant complexity) on the interval* $[x_{ik}, x_{i,k+1})$ *for each* $0 \leq k \leq m$.

Proof. Recall that $f_i(x) = y_{ij}$ if $x_{ij} \leq x < x_{i,j+1}$ for any $0 \leq j \leq m$. To simplify the notation, for each j with $0 \leq j \leq m+1$, we use x_j and y_j to refer to x_{ij} and y_{ij}, respectively. Then we have

$$\mathsf{Ed}(x_p, P_i) = \int_{-\infty}^{+\infty} f_i(x)|x - x_p|dx = \sum_{j=0}^{m} \int_{x_j}^{x_{j+1}} y_j|x - x_p|dx.$$

Consider any fixed point p. Without loss of generality, we assume $x_k \leq x_p < x_{k+1}$ for some k with $0 \leq k \leq m$. Then we can obtain

$$\mathsf{Ed}(x_p, P_i) = \int_{-\infty}^{x_p} f_i(x)(x_p - x)dx + \int_{x_p}^{+\infty} f_i(x)(x - x_p)dx.$$

Further, we have

$$\begin{aligned}
&\int_{-\infty}^{x_p} f_i(x)(x_p - x)dx = \sum_{j=0}^{k-1} \int_{x_j}^{x_{j+1}} y_j(x_p - x)dx + \int_{x_k}^{x_p} y_k(x_p - x)dx \\
&= \sum_{j=0}^{k-1} y_j \Big[\int_{x_j}^{x_{j+1}} x_p dx - \int_{x_j}^{x_{j+1}} x dx \Big] + y_k \Big[\int_{x_k}^{x_p} x_p dx - \int_{x_k}^{x_p} x dx \Big] \\
&= \sum_{j=0}^{k-1} y_j \Big[x_p(x_{j+1} - x_j) - (x_{j+1}^2 - x_j^2)/2 \Big] + y_k \Big[x_p^2 - x_p x_k - (x_p^2 - x_k^2)/2 \Big] \\
&= \frac{1}{2} y_k \cdot x_p^2 + \Big[\sum_{j=0}^{k-1} y_j(x_{j+1} - x_j) - y_k x_k \Big] \cdot x_p + \frac{1}{2} \Big[y_k x_k^2 - \sum_{j=0}^{k-1} y_j(x_{j+1}^2 - x_j^2) \Big].
\end{aligned}$$

Similarly, we can obtain that $\int_{x_p}^{+\infty} f_i(x)(x - x_p)dx$ is equal to

$$\frac{1}{2}y_k \cdot x_p^2 - \Big[y_k x_{k+1} + \sum_{j=k+1}^{m} y_j(x_{j+1} - x_j)\Big] \cdot x_p + \frac{1}{2}\Big[y_k x_{k+1}^2 + \sum_{j=k+1}^{m} y_j(x_{j+1}^2 - x_j^2)\Big].$$

We define $a(k) = y_k$,

$$b(k) = \sum_{j=0}^{k-1} y_j(x_{j+1} - x_j) - y_k(x_k + x_{k+1}) - \sum_{j=k+1}^{m} y_j(x_{j+1} - x_j),$$

and

$$c(k) = \frac{1}{2} \cdot [\sum_{j=k+1}^{m} y_j(x_{j+1}^2 - x_j^2) + y_k(x_k^2 + x_{k+1}^2) - \sum_{j=0}^{k-1} y_j(x_{j+1}^2 - x_j^2)].$$

The discussion above leads to $\mathsf{Ed}(x_p, P_i) = a(k) \cdot x_p^2 + b(k) \cdot x_p + c(k)$.

An easy observation is that as long as $x_p \in [x_k, x_{k+1})$, the three values $a(k)$, $b(k)$, and $c(k)$ are constants, and thus the function $\mathsf{Ed}(x_p, P_i)$ is a parabola that opens up due to $a(k) = y_k \geq 0$ (if $y_k = 0$, then $\mathsf{Ed}(x_p, P_i)$ is a line, which is considered as a special parabola). Further, when $x_p \leq -\frac{b(k)}{2a(k)}$, the function $\mathsf{Ed}(x_p, P_i)$ is monotonically decreasing; when $x_p \geq -\frac{b(k)}{2a(k)}$, $\mathsf{Ed}(x_p, P_i)$ is monotonically increasing. Therefore, we have the following three cases.

- If $x_k \geq -\frac{b(k)}{2a(k)}$, then $\mathsf{Ed}(x_p, P_i)$ is monotonically increasing on $x_p \in [x_k, x_{k+1})$.

  Now suppose p is located in any place on L with $x_p \geq x_{k+1}$. Let $x_{k'} \leq x_p < x_{k'+1}$ for some k' with $k < k' \leq m$. Below, we prove that $x_{k'} \geq -\frac{b(k')}{2a(k')}$ must hold, which implies that $\mathsf{Ed}(x_p, P_i)$ is also monotonically increasing on $x_p \in [x_{k'}, x_{k'+1})$.

  To simplify the notation, for any $0 \leq i_1 \leq i_2 \leq m$, we define $\beta(i_1, i_2) = \sum_{j=i_1}^{i_2} y_j (x_{j+1} - x_j)$. We have

$$\begin{aligned} -\frac{b(k')}{2a(k')} &= \frac{\beta(k'+1, m) - \beta(0, k'-1) + y_{k'}(x_{k'} + x_{k'+1})}{2y_{k'}} \\ &= \frac{\beta(k'+1, m) - \beta(0, k'-1)}{2y_{k'}} + \frac{x_{k'} + x_{k'+1}}{2}. \end{aligned}$$

  Hence, to prove $x_{k'} \geq -\frac{b(k')}{2a(k')}$, it is sufficient to show that $\beta(0, k'-1) \geq (x_{k'+1} - x_{k'})y_{k'} + \beta(k'+1, m)$. Observe that $(x_{k'+1} - x_{k'})y_{k'} + \beta(k'+1, m) = \beta(k', m)$. Therefore, our goal is to show that $\beta(0, k'-1) \geq \beta(k', m)$.

Recall that $x_k \geq -\frac{b(k)}{2a(k)}$. By the similar analysis as above, we can show that $\beta(0, k-1) \geq \beta(k, m)$. Since $k < k'$, it is easy to see that $\beta(0, k'-1) \geq \beta(0, k-1)$ and $\beta(k', m) \leq \beta(k, m)$, and thus, we can obtain $\beta(0, k'-1) \geq \beta(k', m)$. Hence, $x_{k'} \geq -\frac{b(k')}{2a(k')}$ is proved.

- If $x_{k+1} \leq -\frac{b(k)}{2a(k)}$, $\mathsf{Ed}(x_p, P_i)$ is monotonically decreasing on $x_p \in [x_k, x_{k+1})$.

  Now suppose p is located in any place on L with $x_p < x_k$. Let $x_{k'} \leq x_p < x_{k'+1}$ for some k' with $0 \leq k' < k$. By using the similar techniques as the above case (we omit the details), we can show that $x_{k'+1} \leq -\frac{b(k')}{2a(k')}$, and thus, $\mathsf{Ed}(x_p, P_i)$ is also monotonically decreasing on $x_p \in [x_{k'}, x_{k'+1})$.

- If $x_k < -\frac{b(k)}{2a(k)} < x_{k+1}$, then $\mathsf{Ed}(x_p, P_i)$ is a unimodal function that achieves the minimum at $x_p = -\frac{b(k)}{2a(k)}$.

  On one hand, suppose p is located in any place on L with $x_p \geq x_{k+1}$. Let $x_{k'} \leq x_p < x_{k'+1}$ for some k' with $k < k' \leq m$. Below, we prove that $x_{k'} \geq -\frac{b(k')}{2a(k')}$ must hold, which implies that $\mathsf{Ed}(x_p, P_i)$ is monotonically increasing on $x_p \in [x_{k'}, x_{k'+1})$. To this end, according to the analysis in the above first case, it is sufficient to show that $\beta(0, k'-1) \geq \beta(k', m)$.

  Since $x_{k+1} > -\frac{b(k)}{2a(k)}$, we have

$$x_{k+1} > -\frac{b(k)}{2a(k)} = \frac{\beta(k+1, m) - \beta(0, k-1)}{2y_k} + \frac{x_k + x_{k+1}}{2},$$

  which is equivalent to $\beta(0, k-1) + (x_{k+1} - x_k)y_k \geq \beta(k+1, m)$. Observe that $\beta(0, k-1) + (x_{k+1} - x_k)y_k = \beta(0, k)$. Thus, it holds that $\beta(0, k) \geq \beta(k+1, m)$.

  Due to $k' > k$ (i.e., $k' \geq k+1$), it follows that $\beta(0, k'-1) \geq \beta(0, k)$ and $\beta(k+1, m) \geq \beta(k', m)$. Therefore, we obtain $\beta(0, k'-1) \geq \beta(k', m)$. Hence, $\mathsf{Ed}(x_p, P_i)$ is monotonically increasing on $x_p \in [x_{k'}, x_{k'+1})$.

  On the other hand, suppose p is located in any place on L with $x_p < x_k$. Let $x_{k'} \leq x_p < x_{k'+1}$ for some k' with $0 \leq k' < k$. By using the similar techniques as above, we can show that $x_{k'+1} \leq -\frac{b(k')}{2a(k')}$, and thus, $\mathsf{Ed}(x_p, P_i)$ is monotonically decreasing on $x_p \in [x_{k'}, x_{k'+1})$.

As a summary, the above analysis proves the following. (1) $\mathsf{Ed}(x_p, P_i)$ on $x \in \mathbb{R}$ defines a function consists of $m+1$ pieces, and each piece is part of a parabola defined on the interval $[x_j, x_{j+1})$ for $j = 0, 1, \ldots, m$. (2) There is at most one interval $[x_k, x_{k+1})$ such that $\mathsf{Ed}(x_p, P_i)$ in the interval is neither monotonically increasing nor decreasing (but is a unimodal function). (3) Suppose $[x_k, x_{k+1})$ is the interval as discussed in the above (2); then for any interval $[x_{k'}, x_{k'+1})$, $\mathsf{Ed}(x_p, P_i)$ on $[x_{k'}, x_{k'+1})$ is monotonically increasing if $k < k'$ and monotonically decreasing if $k' < k$.

To prove that $\mathsf{Ed}(x_p, P_i)$ on $x_p \in \mathbb{R}$ is a unimodal function, it is sufficient to show that $\mathsf{Ed}(x_p, P_i)$ is continuous on each value $x_p \in \mathbb{R}$. Clearly, $\mathsf{Ed}(x_p, P_i)$ is continuous on each interval $x_p \in [x_k, x_{k+1})$, it remains to show that $\mathsf{Ed}(x_p, P_i)$ is continuous on $x_p = x_k$ for each k with $1 \leq k \leq m$ (note that since $x_0 = -\infty$ and $x_{m+1} = +\infty$, we do not need to prove the continuity of $\mathsf{Ed}(x_p, P_i)$ at these values).

Consider x_k for any $1 \leq k \leq m$. If $x_p = x_k$, we have $\mathsf{Ed}(x_k, P_i) = a(k) \cdot x_k^2 + b(k) \cdot x_k + c(k)$. To prove $\mathsf{Ed}(x_p, P_i)$ is continuous at $x_p = x_k$, it sufficient to show that for $x_p < x_k$, $\lim_{x_p \to x_k} \mathsf{Ed}(x_p, P_i) = \mathsf{Ed}(x_k, P_i)$. If $x_p < x_k$ and x_p is arbitrarily close to x_k, then we have $x_p \in [x_{k-1}, x_k)$ and $\mathsf{Ed}(x_p, P_i) = a(k-1) \cdot x_p^2 + b(k-1) \cdot x_p + c(k-1)$. Hence, $\lim_{x_p \to x_k} \mathsf{Ed}(x_p, P_i) = a(k-1) \cdot x_k^2 + b(k-1) \cdot x_k + c(k-1)$. It can be verified that $a(k) \cdot x_k^2 + b(k) \cdot x_k + c(k) = a(k-1) \cdot x_k^2 + b(k-1) \cdot x_k + c(k-1)$ from our definitions of $a(k)$, $b(k)$, and $c(k)$, and we omit the details.

Therefore, $\mathsf{Ed}(x_p, P_i)$ on $x_p \in \mathbb{R}$ is a unimodal function.

It remains to show that the function $\mathsf{Ed}(x_p, P_i)$ can be computed in $O(m)$ time. To this end, it is sufficient to compute the three values $a(k)$, $b(k)$, and $c(k)$ for all $k = 0, 1, \ldots, m$. We first compute $a(0)$, $b(0)$, and $c(0)$, which can be done in $O(m)$ time. In general, after we compute $a(k)$, $b(k)$, and $c(k)$, it is easy to compute the three values $a(k+1)$, $b(k+1)$, and $c(k+1)$ in constant time based on $a(k)$, $b(k)$, and $c(k)$. Therefore, $\mathsf{Ed}(x_p, P_i)$ can be computed in $O(m)$ time.

The lemma thus follows. □

In light of Lemma 2.2.1, we have the following corollary.

Corollary 2.2.2. *For each uncertain point P_i, with $O(m)$ time preprocessing, we can compute the value $\mathsf{Ed}(x_p, P_i)$ in $O(\log m)$ time for any query point p on L.*

Proof. Again, for each j with $0 \leq j \leq m+1$, we use x_j to refer to x_{ij}. As preprocessing, we first explicitly compute the function $\mathsf{Ed}(x_p, P_i)$ in $O(m)$ time by Lemma 2.2.1. For any query point $p \in L$, we first determine the interval $[x_k, x_{k+1})$ that contains x_p, which can be done in $O(\log m)$ time by binary search on the list $x_0, x_1, \ldots, x_{m+1}$. According to the proof of Lemma 2.2.1, we have computed the three values $a(k)$, $b(k)$, and $c(k)$, such that $\mathsf{Ed}(x_p, P_i) = a(k)x_p^2 + b(k)x_p + c(k)$. Hence, we can compute the value $\mathsf{Ed}(x_p, P_i)$ in additional constant time. □

Consider any uncertain point $P_i \in \mathcal{P}$. According to Lemma 2.2.1, there is a point $p \in L$ that minimizes the value $\mathsf{Ed}(p, P_i)$ and let p_i denote such a point; note that such a point may not be unique, in which case we let p_i denote any such point. We refer to p_i as the *centroid* of P_i. By Lemma 2.2.1, we can compute the centroids for all uncertain points of $\mathcal{P}$ in $O(nm)$ time.

2.3 The Decision k-Center Problem

In order to solve our k-center problem, we first solve the decision version of the problem in this section.

Recall that our goal for the k-center problem is to find a set Q of k points such that $\max_{P_i \in \mathcal{P}} \mathsf{Ed}(Q, P_i)$ is minimized, where $\mathsf{Ed}(Q, P_i) = \min_{q \in Q} \mathsf{Ed}(q, P_i)$. Below, for any set Q of points on L, let $\psi(Q) = \max_{P_i \in \mathcal{P}} \mathsf{Ed}(Q, P_i)$. Denote by ϵ^* the value $\psi(Q)$ in an optimal solution for the k-center problem.

Given any real value ϵ, the *decision k-center* problem is to determine whether there exist a set Q of k points on L such that $\psi(Q) \leq \epsilon$ (i.e., determine whether $\epsilon^* \leq \epsilon$), and if yes, then we say the decision problem is *feasible* and ϵ is a *feasible value*. To distinguish from the decision problem, we refer to our original k-center problem the *optimization problem*. Clearly, ϵ^* is the smallest feasible value.

Consider any value ϵ and any uncertain point $P_i \in \mathcal{P}$. Let Q be any set of k points on L. If $\mathsf{Ed}(Q, P_i) \leq \epsilon$, then there is at least one point q in Q with $\mathsf{Ed}(q, P_i) \leq \epsilon$. Let $\alpha(P_i, \epsilon)$ be the set of points p of L such that $\mathsf{Ed}(x_p, P_i) \leq \epsilon$. A line segment on L is also called an *interval* of L. By using Lemma 2.2.1, we obtain the following result.

Lemma 2.3.1. *For any uncertain point P_i and any value ϵ, $\alpha(P_i, \epsilon)$ is an interval of L ($\alpha(P_i, \epsilon) = \emptyset$ is possible); with $O(m)$ time preprocessing, we can compute $\alpha(P_i, \epsilon)$ in $O(\log m)$ time for any given ϵ.*

Proof. Since $\mathsf{Ed}(x_p, P_i)$ is a unimodal function, if the minimum value of $\mathsf{Ed}(x_p, P_i)$ is larger than ϵ, then $\alpha(P_i, \epsilon) = \emptyset$. Otherwise, $\alpha(P_i, \epsilon)$ is an interval of L. Specifically, let p_l be the leftmost point of L such that $\mathsf{Ed}(x_{p_l}, P_i) = \epsilon$ and let p_r be the rightmost point of L such that $\mathsf{Ed}(x_{p_r}, P_i) = \epsilon$. Then, $\alpha(P_i, \epsilon)$ is the line segment $\overline{p_l p_r}$.

To compute $\alpha(P_i, \epsilon)$, we compute the function $\mathsf{Ed}(x_p, P_i)$ for $x_p \in \mathbb{R}$ in $O(m)$ time by Lemma 2.2.1, as preprocessing. Also, we compute the centroid p_i of P_i in the processing.

Consider any query value ϵ. To compute $\alpha(P_i, \epsilon)$, we first check the y-coordinate of p_i, denoted by y_{p_i}. If $y_{p_i} > \epsilon$, then $\alpha(P_i, \epsilon) = \emptyset$. Otherwise, we find p_l and p_r, as follows. Observe that $x_{p_l} \leq x_{p_i}$ and $x_{p_r} \geq x_{p_i}$. We first find p_l in the following way.

Suppose $x_{ik} < p_i \leq x_{i,k+1}$ for some k with $0 \leq k$. Note that k is already determined in the preprocessing when we compute p_i. For the purpose of searching p_l, we temporarily set $x_{i,k+1} = x_{p_i}$. Since the function $\mathsf{Ed}(x_p, P_i)$ for $x_p \in (-\infty, x_{p_i}]$ is monotonically decreasing, we can determine the index j such that $\mathsf{Ed}(x_{ij}, P_i) > \epsilon \geq \mathsf{Ed}(x_{i,j+1,P_j})$ in $O(\log m)$ time by binary search. Hence, we have $x_{ij} < x_{p_l} \leq x_{i,j+1}$. Since the function $\mathsf{Ed}(x_p, P_i)$ for $x_p \in [x_{ij}, x_{i,j+1})$ is a parabola of constant complexity and the value $\mathsf{Ed}(x_{i,j+1}, P_i)$ is computed in the preprocessing, we can determine p_l in additional constant time. Therefore, we can find p_l in $O(\log m)$ time.

Similarly, we can also find p_r in $O(\log m)$ time. Thus, given any ϵ, we can compute $\alpha(P_i, \epsilon)$ in $O(\log m)$ time. □

We say that a point on L *covers* an interval of L if the interval contains the point. Let $\alpha(\mathcal{P}, \epsilon)$ is the set of all intervals $\alpha(P_i, \epsilon)$ for $i = 1 \ldots n$. We have the following observation.

Observation 2.3.2. *The value ϵ is a feasible value if and only if there exist a set Q of k points on L such that each interval of $\alpha(\mathcal{P}, \epsilon)$ is covered by at least one point in Q.*

By Observation 2.3.2, to determine whether ϵ is feasible, it is sufficient to solve the following *interval covering problem*: determine whether there exist a set Q of k points on L such that each interval of $\alpha(\mathcal{P}, \epsilon)$ is covered by at least one point in Q.

To solve the interval covering problem, we can compute the minimum number k^* of points that can cover all intervals of $\alpha(\mathcal{P}, \epsilon)$, and the problem can be solved in $O(n)$ time by a simple greedy algorithm after the endpoints of all intervals of $\alpha(\mathcal{P}, \epsilon)$ are sorted [47]. Specifically, we scan the sorted endpoints of intervals of $\alpha(\mathcal{P}, \epsilon)$ from left to right until we first encounter a right endpoint of an interval. We add this right endpoint into Q and removes all intervals that contain this point. This process is repeated until no intervals remain. However, due to the sorting procedure, the total time for computing k^* is $O(n \log n)$.

Snoeyink [48] gave an $O(n \log k^*)$ time algorithm for computing k^* without sorting. If $k^* \leq k$, then we have $n \log k^* = O(n \log k)$, which means that we can solve the interval covering problem in $O(n \log k)$ time. However, if $k^* > k$, since it is possible that $n \log k = o(n \log k^*)$ (e.g., $k = O(1)$ and $k^* = \Theta(n)$), we cannot bound the time by $O(n \log k)$. To ensure that the interval covering algorithm can still be solved in $O(n \log k)$ time even if $k^* > k$, we modify Snoeyink's algorithm [48] in the following way.

Observe that to solve the interval covering algorithm, it is sufficient to know whether $k^* \geq k$ holds. Snoeyink's algorithm finds a set Q of points one by one in $O(n \log k^*)$ time, with $k^* = |Q|$. Since the points of Q are computed one by one, when we run the algorithm, we simply stop the algorithm when there are $k + 1$ points in the current Q. In this way, the recursion tree of the algorithm has $k + 1$ leaves (instead of k^* leaves) and thus the running time is $O(n \log k)$ according to Lemma 1 in [48].

As a summary, given any ϵ, we solve the decision k-center as follows. First, we compute all intervals $\alpha(\mathcal{P}, \epsilon)$, in $O(n \log m)$ time by Lemma 2.3.1. Then, by modifying the algorithm in [48] as discussed above, we can solve the interval covering problem in $O(n \log k)$ time. The decision problem is thus solved. The total time is $O(n \log m + n \log k)$, after the $O(mn)$ time preprocessing in Lemma 2.3.1 for all uncertain points. By using fractional cascading [49, 50], we further reduce the running time in Lemma 2.3.3.

Lemma 2.3.3. *With $O(mn)$ time preprocessing, we can determine whether ϵ is a feasible value in $O(\log m + n \log k)$ time for any given ϵ.*

Proof. To prove the lemma, we show that with $O(mn)$ time preprocessing, we can compute the intervals of $\alpha(\mathcal{P}, \epsilon)$ in $O(n + \log m)$ time for any given ϵ.

We first do the preprocessing in Lemma 2.3.1 for all uncertain points. Consider any uncertain point $P_i \in \mathcal{P}$. Let p_i be the centroid of P_i, which has been computed in the preprocessing. Given any ϵ, as discussed in Lemma 2.3.1, if ϵ is smaller than the y-coordinate of p_i, then $\alpha(P_i, \epsilon) = \emptyset$. Below, we assume $\alpha(P_i, \epsilon) \neq \emptyset$. Let p_l be the left endpoint of $\alpha(P_i, \epsilon)$ and p_r the right endpoint of $\alpha(P_i, \epsilon)$. As in Lemma 2.3.1, $x_{p_l} \leq x_{p_i}$ and $x_{p_r} \geq x_{p_i}$. Let $x_{ik} < x_{p_i} \leq x_{i,k+1}$ for some k with $0 \leq k \leq m$, and again, k has been computed in the preprocessing. For the discussion of searching p_l, we temporarily set $x_{i,k+1} = x_{p_i}$.

As discussed in the proof of Lemma 2.3.1, to find p_l, we can first determine j with $0 \leq j \leq k$ such that $\mathsf{Ed}(x_{ij}, P_i) > \epsilon \geq \mathsf{Ed}(x_{i,j+1}, P_i)$, and subsequently compute p_l in additional constant time. Finding such an index j can be done in $O(\log m)$ by binary search, as in Lemma 2.3.1, and binary search on all uncertain points together takes $O(n \log m)$ time. To reduce the running time, a key observation is that each binary search uses the value ϵ, which allows us to use the fractional cascading technique [49,50], as follows.

Consider the above problem of searching the index j for p_l. We create a sorted lists L_i in descending order: $+\infty, \mathsf{Ed}(x_{i1}, P_i), \mathsf{Ed}(x_{i2}, P_i), \ldots, \mathsf{Ed}(x_{i,k+1}, P_i), -\infty$ (recall that $x_{i,k+1}$ has been set to x_{p_i} temporarily). Hence, the above index j can be found by doing binary search on the list L_i using ϵ.

Similarly, we can create a sorted list L'_i for P_i based on the function $\mathsf{Ed}(x_p, P_i)$ on $x_p \in [x_{p_i}, +\infty)$ and use L'_i to find p_r.

For each uncertain point $P_i \in \mathcal{P}$, we create two sorted lists L_i and L'_i as above. Given any ϵ, our goal is to find the interval in each sorted list that contains ϵ. By building a fractional cascading structure on the $2n$ sorted lists in $O(mn)$ time [49, 50], we can find the intervals containing ϵ in all sorted lists in overall $O(\log m + n)$ time, and consequently obtain all intervals of $\alpha(\mathcal{P}, \epsilon)$ in additional $O(n)$ time.

We summarize our discussion above. As preprocessing, we compute the functions $\mathsf{Ed}(x_p, P_i)$ for all uncertain points, in $O(mn)$ time by Lemma 2.2.1. Based on these functions, we create the above $2n$ sorted lists in $O(mn)$ time. Then, we build a fractional

cascading on these sorted lists in $O(mn)$ time [49, 50]. This finishes our preprocessing, which takes $O(mn)$ time in total. For any given value ϵ, we can compute all intervals of $\alpha(\mathcal{P}, \epsilon)$ in $O(\log m + n)$ time, as discussed above.

The lemma thus follows. □

2.4 The Optimization Problem

In this section, we present our algorithm for the original k-center problem, to which we refer as the *optimization* problem, and our goal is to find the smallest feasible value ϵ^* and the corresponding optimal facility set Q. Based on some observations and our decision algorithm in Lemma 2.3.3, we finally compute ϵ^* by modifying the parametric search technique [43, 44].

For any $\epsilon > 0$, for each $1 \leq i \leq n$, let $\alpha(P_i, \epsilon) = [l_i(\epsilon), r_i(\epsilon)]$, i.e., $l_i(\epsilon)$ is the x-coordinate of the left endpoint of $\alpha(P_i, \epsilon)$ and $r_i(\epsilon)$ is the x-coordinate of the right endpoint of $\alpha(P_i, \epsilon)$; below we will consider $l_i(\epsilon)$ and $r_i(\epsilon)$ as functions of ϵ. With a little abuse of notation, we also use $l_i(\epsilon)$ and $r_i(\epsilon)$ to denote the left and right endpoints of $\alpha(P_i, \epsilon)$, respectively. Define $E(\epsilon)$ to be the set of the endpoints of all intervals in $\alpha(\mathcal{P}, \epsilon)$. Notice that if we know the sorted order of the endpoints of $E(\epsilon^*)$ at the value ϵ^*, we can easily find an optimal facility set Q, e.g., by using the greedy algorithm mentioned before. Although we do not know ϵ^*, but we can still sort the values in $E(\epsilon^*)$ by making use of our decision algorithm to resolve comparisons, which is the key idea of parametric search [43, 44]. However, our problem does not allow us to apply the parametric search approaches in [43, 44] directly, because in our problem we cannot resolve each "comparison" by a single call on the decision algorithm (since a comparison may have multiple "roots"; refer to [43,44] for these standard terminology on parametric search). The details are given below.

Suppose in our sorting algorithm we want to resolve a comparison between two values in $E(\epsilon^*)$. Depending on whether the two values are left endpoints or right endpoints, there are two cases.

1. If a value is a left endpoint, say $l_i(\epsilon^*)$, and the other value is a right endpoint, say $r_j(\epsilon^*)$, then the comparison between them is called a *type-1* comparison. We resolve this type of comparison in the following way.

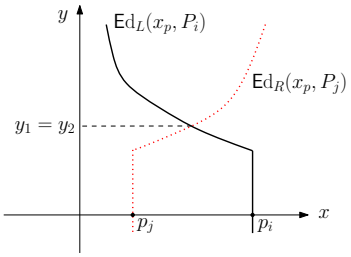

Figure 2.2. Illustrating the intersection of $\mathsf{Ed}_L(x_p, P_i)$ and $\mathsf{Ed}_R(x_p, P_j)$, where the intersection is a single point and thus $y_1 = y_2$. $l_i(\epsilon^*) \geq r_j(\epsilon^*)$ if and only if $\epsilon^* \leq y_1$.

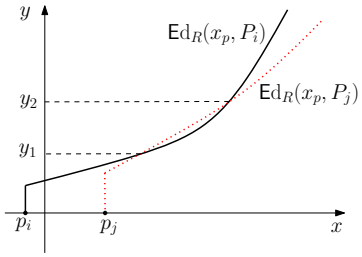

Figure 2.3. Illustrating two intersections of $\mathsf{Ed}_R(x_p, P_i)$ and $\mathsf{Ed}_R(x_p, P_j)$. For example, if $\epsilon^* \in [y_1, y_2]$, then $r_i(\epsilon^*) \geq r_j(\epsilon^*)$.

Recall that p_k is the centroid for each uncertain point $P_k \in \mathcal{P}$. We denote the function $\mathsf{Ed}(x_p, P_k)$ on $x_p \in (-\infty, x_{p_k}]$ by $\mathsf{Ed}_L(x_p, P_k)$ and denote $\mathsf{Ed}(x_p, P_k)$ on $x \in [x_{p_k}, +\infty)$ by $\mathsf{Ed}_R(x_p, P_k)$. By Lemma 2.2.1, $\mathsf{Ed}_L(x_p, P_k)$ is monotonically decreasing and $\mathsf{Ed}_R(x_p, P_k)$ is monotonically increasing. Further, $l_k(\epsilon) \leq x_{p_k} \leq r_k(\epsilon)$ holds. To simplify the discussion, for each $P_k \in \mathcal{P}$, we add a vertical half-line on the function $\mathsf{Ed}_L(x_p, P_k)$ from the point p_k downwards to $-\infty$ and we also add the same half-line to $\mathsf{Ed}_R(x_p, P_k)$. Note that each new $\mathsf{Ed}_L(x_p, P_k)$ is still monotonically decreasing and each new $\mathsf{Ed}_R(x_p, P_k)$ is still monotonically increasing.

To resolve the comparison between $l_i(\epsilon^*)$ and $r_j(\epsilon^*)$, our goal is to determine whether $l_i(\epsilon^*) \leq r_j(\epsilon^*)$ or $l_i(\epsilon^*) \geq r_j(\epsilon^*)$ holds. To this end, we first determine whether $\mathsf{Ed}_L(x_p, P_i)$ intersects $\mathsf{Ed}_R(x_p, P_j)$.

If $x_{p_i} < x_{p_j}$, then since $\mathsf{Ed}_L(x_p, P_i)$ is to the left of p_i and $\mathsf{Ed}_R(x_p, P_j)$ is to the right of p_j, the two functions do not intersect and $l_i(\epsilon^*) \leq r_j(\epsilon^*)$ always holds.

Otherwise, since $\mathsf{Ed}_L(x_p, P_i)$ is monotonically decreasing and $\mathsf{Ed}_R(x_p, P_j)$ is monotonically increasing, $\mathsf{Ed}_L(x_p, P_i)$ must intersect $\mathsf{Ed}_R(x_p, P_j)$ and the intersection is a line segment (may be degenerated into a single point) that spans an interval $[y_1, y_2]$ on y-coordinates (e.g., see Fig. 2.2). Observe that $l_i(\epsilon^*) < r_j(\epsilon^*)$ if $\epsilon^* > y_2$, $l_i(\epsilon^*) = r_j(\epsilon^*)$ if $\epsilon^* \in [y_1, y_2]$, and $l_i(\epsilon^*) > r_j(\epsilon^*)$ if $\epsilon^* < y_1$.

Hence, to resolve the comparison between $l_i(\epsilon^*)$ and $r_j(\epsilon^*)$, it sufficient to resolve the comparisons among ϵ^*, y_1, and y_2, which can be done by calling the decision

algorithm to determine whether y_1 and y_2 are feasible values. Specifically, if $\epsilon = y_2$ is not feasible, then $\epsilon^* > y_2$ and we obtain $l_i(\epsilon^*) < r_j(\epsilon^*)$. If $\epsilon = y_2$ is feasible, then $\epsilon^* \leq y_2$. We further check whether $\epsilon = y_1$ is feasible. If y_1 is not feasible, then we have $\epsilon^* \in (y_1, y_2]$ and thus obtain $l_i(\epsilon^*) = r_j(\epsilon^*)$; otherwise, we have $\epsilon^* \leq y_1$ and obtain $l_i(\epsilon^*) \geq r_j(\epsilon^*)$.

In summary, we can resolve the comparison between $l_i(\epsilon^*)$ and $r_j(\epsilon^*)$ by first finding the intersection of $\mathsf{Ed}_L(x_p, P_i)$ and $\mathsf{Ed}_R(x_p, P_j)$ and subsequently at most two calls on the decision algorithm. The intersection of $\mathsf{Ed}_L(x_p, P_i)$ and $\mathsf{Ed}_R(x_p, P_j)$ can be found in $O(m)$ time (it is possible to do better by binary search; however, this does not affect the overall result because the time for resolving each comparison is dominated by resolving a type-2 comparison, which will be given later and has a term $O(m)$ in its running time). The two calls on the decision algorithm takes $O(\log m + n \log k)$ time.

Hence, we can resolve each type-1 comparison in $O(m + n \log k)$ time.

2. If the two values involved in the comparison are both left endpoints or both right endpoints, then we call it a *type-2* comparison. It becomes more complex to resolve this type of comparison. Assume both values are two right endpoints, say $r_i(\epsilon^*)$ and $r_j(\epsilon^*)$, and the case where both values are two left endpoints can be handled similarly. In the sequel, we resolve the comparison in the following way.

   As in the type-1 case, we first compute the intersections between the two functions $\mathsf{Ed}_R(x_p, P_i)$ and $\mathsf{Ed}_R(s, P_j)$. Although both functions are monotonically increasing, there may be $\Theta(m)$ intersections as their complexities are $\Theta(m)$ in the worst case (e.g., see Fig. 2.3). All intersections can be computed in $O(m)$ time. If there is no intersection, then $r_i(\epsilon^*) \leq r_j(\epsilon^*)$ if and only if $x_{p_i} \leq x_{p_j}$, where p_i and p_j are the centroids.

   Otherwise, let $y_1, y_2, \ldots, y_h$ be the y-coordinates of all intersections, sorted in ascending order, with $h = O(m)$. We can compute this sorted list in $O(m)$ time as we compute the intersections. Using our decision algorithm, we can determine an interval $(y_k, y_{k+1}]$ that contains ϵ^*, by binary search with $O(\log m)$ calls on the decision algorithm. After finding the interval $(y_k, y_{k+1}]$, we can easily determine

whether $r_i(\epsilon^*) \leq r_j(\epsilon^*)$ or $r_i(\epsilon^*) \geq r_j(\epsilon^*)$ in the similar way as in the type-1 case (e.g., see Fig. 2.3).

Hence, we can resolve each type-2 comparison in $O(m + n \log k \log m)$ time.

The above shows that we can resolve each comparison in $O(m + n \log k \log m)$ time, which is dominated by the type-2 comparisons.

Now we apply the parametric search scheme to our problem by resolving comparisons in the above ways. We first consider Megiddo's approach [44]. We can use n processors to do the soring in $O(\log n)$ parallel steps. For each parallel step, we need to resolve n "independent" comparisons. Our problem is different from other problems in the sense that each type-2 comparison can have $O(m)$ "roots" (i.e., the y-coordinates of the intersections). Nevertheless, we can still be able to resolve all these comparisons in a simultaneous way, as follows.

First, for each comparison, we compute the coordinates of the $O(m)$ intersections as discussed above. The intersections of all n comparisons can be computed in $O(mn)$ time. Then, we have $O(mn)$ roots. Suppose $y_1, y_2, \ldots, y_h$ are the list of all $O(mn)$ roots sorted in ascending order, with $h = O(mn)$. Note that we only use this sorted list to explain our approach and our algorithm do not compute this sorted list. By using our decision algorithm, we determine the interval $(y_k, y_{k+1}]$ that contains ϵ, which can be done in $O(mn)$ time plus $O(\log mn)$ calls on the decision algorithm by using the linear time selection algorithm and binary search (without computing the above sorted list). Further, all n comparisons are resolved on the interval $(y_k, y_{k+1}]$. Therefore, we can resolve all these n independent comparisons in $O(mn + n \log k \log mn)$ time. Since there are $O(\log n)$ parallel steps, we can resolve all comparisons and compute the order for $E(\epsilon^*)$ in $O(mn \log n + n \log k \log n \log mn)$ time.

Once the order for $E(\epsilon^*)$ is determined, we can easily compute ϵ^* and obtain an optimal facility set Q by using the greedy algorithm discussed in Section 2.3. In fact, we can immediately determine ϵ^* after the above parametric search finishes. Specifically, after the parametric search finishes, the algorithm also gives us an interval $(y_k, y_{k+1}]$ that contains ϵ^*. We claim that $\epsilon^* = y_{k+1}$. Indeed, an observation is that ϵ^* is always equal to the y-coordinate of the intersection of two functions $\mathsf{Ed}(x_p, P_i)$ and $\mathsf{Ed}(x_p, P_j)$ since

otherwise we would always make ϵ^* smaller without changing the order of $E(\epsilon^*)$. On the other hand, the parametric search essentially finds y_{k+1} as the smallest y-coordinate of such function intersections that is feasible. Therefore, $\epsilon^* = y_{k+1}$.

In summary, we solve the k-center problem in $O(mn \log n + n \log k \log n \log mn)$ time. Note that this result is based on the assumption that x_{ij} for $j = 1, \ldots, m$ are given sorted for each uncertain point $P_i \in \mathcal{P}$. If they are not given sorted, then we need an extra step to sort them first, which takes $O(mn \log m)$ time in total. Therefore, we have the following lemma.

Theorem 2.4.1. *The optimization version of the k-center problem can be solved in $O(mn \cdot \log mn + n \log k \log n \log mn)$ time.*

One may wonder whether Cole's parametric search [43] can be used to further reduce the time complexity by a logarithmic factor, i.e., reduce the time to $O(mn \log mn + n \log k \log mn)$. However this is not the case because resolving each type-2 comparison needs to consider $O(m)$ roots. Specifically, in Cole's parametric search, calling the decision algorithm on the weighted median root of all roots in each comparison level can resolve a weighted-half comparisons in the level. However, in our problem, to resolve the each type-2 comparison, calling the decision algorithm once is not enough. Therefore, Cole's approach is not applicable to our problem.

Since even Megiddo's parametric search may not be quite practical, Van Oostrum and Veltkamp [45] showed that one can replace the parallel sorting scheme in Megiddo's parametric search by the randomized quicksort to obtain a practical solution with the same expected running time. By using the randomized quicksort, we can solve the k-center problem in expected $O(mn \log nm + n \log k \log n \log mn)$ time and the algorithm is relatively practical.

2.5 The Discrete k-Center Problem

In this section, we present an algorithm for the discrete version of the k-center problem, and due to some special properties of the discrete case, the algorithm is faster than the one in Theorem 2.4.1 for the general case.

In the discrete k-center problem, each uncertain point P_i has m possible locations, denoted by $p_{i1}, p_{i2}, \ldots, p_{im}$, each having a probability. Since this is a special case of the general k-center problem, the previous results on the general k-center problem (e.g., Lemma 2.3.3 and Theorem 2.4.1) are still applicable.

By Lemma 2.2.1, the function $\mathsf{Ed}(x_p, P_i)$ for $x_p \in \mathbb{R}$ is still a unimodal function, but in the discrete version, $\mathsf{Ed}(x_p, P_i)$ is a piecewise linear function. This can be obtained from the proof of Lemma 2.2.1 and we omit the details here. After the locations $p_{i1}, p_{i2}, \ldots, p_{im}$ are sorted in $O(m \log m)$ time, the function $\mathsf{Ed}(x_p, P_i)$ can be computed in additional $O(m)$ time by Lemma 2.2.1. In the following, we assume all functions $\mathsf{Ed}(x_p, P_i)$ for $i = 1, 2, \ldots, n$ have been computed.

We define the decision problem in the same way as before. Our goal is to find the smallest feasible value ϵ^*. As we discussed in the general k-center problem, ϵ^* is the y-coordinate of the intersection of two functions $\mathsf{Ed}(x_p, P_i)$ and $\mathsf{Ed}(x_p, P_j)$ for some i and j. Let $\mathcal{I}$ be the set of intersections of all functions $\mathsf{Ed}(x_p, P_i)$ for $i = 1, 2, \ldots, n$, and for simplicity of discussion, we assume each such intersection is a single point (the general case can be solved by the same techniques with more tedious discussion). Then, ϵ^* is the smallest feasible value among the y-coordinates of all points of $\mathcal{I}$. The algorithm of Theorem 2.4.1 uses parametric search to find ϵ^*. In the discrete version, due to the property that each $\mathsf{Ed}(x_p, P_i)$ is a piecewise linear function, we compute ϵ^* by using a technique for searching line arrangement [46], as follows.

We first define an arrangement $\mathcal{A}$. For each $1 \leq i \leq n$, since $\mathsf{Ed}(x_p, P_i)$ is a piecewise linear function, it consists of $O(m)$ line segments and two half-lines, and we let A_i denote the set of lines containing all line segments and half-lines of $\mathsf{Ed}(x_p, P_i)$. Hence, $|A_i| = O(m)$ for each $1 \leq i \leq n$. Note that we can explicitly compute each A_i in $O(m)$ time. Let $\mathcal{A}$ be the arrangement of the lines in $\bigcup_{i=1}^{n} A_i$. Note that our algorithm does not compute $\mathcal{A}$ explicitly. With a little abuse of notation, we also use $\mathcal{A}$ to denote the set of all *vertices* of $\mathcal{A}$ (i.e., all line intersections). Clearly, $\mathcal{I} \subseteq \mathcal{A}$. Hence, ϵ^* is also the smallest feasible value among the y-coordinates of the vertices of $\mathcal{A}$, and in other words, ϵ^* is the y-coordinate of the lowest vertex v^* of $\mathcal{A}$ whose y-coordinate is feasible for the decision problem. To search the particular vertex v^*, we use the decision algorithm in Lemma 2.3.3 and the following arrangement searching technique in [46].

Suppose there is a function $g : \mathbb{R} \to \{0, 1\}$, such that the description of g is unknown but it is known that g is monotonically increasing. Further, given any value y, we have a "black-box" that can evaluate $g(y)$ (i.e., determine whether $g(y)$ is 1 or 0) in $O(G)$ time, which we call the *g-oracle* (i.e., $O(G)$ is the query time for the g-oracle). Let B be a set of n lines in the plane and let $\mathcal{B}$ denote their arrangement. Note that $\mathcal{B}$ is not computed explicitly. For any vertex v of $\mathcal{B}$, let y_v be the y-coordinate of v. The *arrangement searching* is to find the lowest vertex vertex v of $\mathcal{B}$ such that $g(y_v) = 1$. An $O((n + G) \log n)$ time algorithm is given in [46] to solve the arrangement searching problem by modifying the slope selection algorithm [51, 52], without using parametric search.

In our problem, we are searching the vertex v^* in the arrangement $\mathcal{A}$. We can define such a function g as follows. For any value y, $g(y) = 1$ if and only if y is a feasible value. Clearly, g is monotonically increasing since for any feasible value y, any value larger than y is also feasible. Hence, v^* is the lowest point in $\mathcal{A}$ with $g(y_{v^*}) = 1$. We use our decision algorithm in Lemma 2.3.3 as the g-oracle with $G = O(\log m + n \log k)$. By the result in [46], after the $O(mn)$ lines of $\bigcup_{i=1}^{n} A_i$ are computed, we can compute v^* in $O((mn + \log m + n \log k) \log mn)$ time. It can be verified that $(mn + \log m + n \log k) \log mn = O(mn \log mn + n \log k \log n)$. Consequently, we can obtain ϵ^*. An optimal solution set Q can be found by using the decision algorithm on ϵ^* in additional $O(\log m + n \log k \log n)$ time.

Theorem 2.5.1. *The optimization version of the discrete k-center problem can be solved in $O(mn \log mn + n \log k \log n)$ time.*

Now we consider the case where $k = 1$ and the locations $p_{i1}, p_{i2}, \ldots, p_{im}$ are given sorted for each uncertain point P_i. Again, the function $\mathsf{Ed}(x_p, P_i)$ can be computed in additional $O(m)$ time by Lemma 2.2.1. Thus, all functions $\mathsf{Ed}(x_p, P_i)$ for $i = 1, 2, \ldots, n$ can be computed in $O(mn)$ time.

Consider any function $\mathsf{Ed}(x_p, P_i)$. Recall that $\mathsf{Ed}(x_p, P_i)$ is a unimodal piecewise linear function. The function $\mathsf{Ed}(x_p, P_i)$ divides the plane into two regions, one above $\mathsf{Ed}(x_p, P_i)$ and the other below $\mathsf{Ed}(x_p, P_i)$. Denote by R_i the region above $\mathsf{Ed}(x_p, P_i)$.

Denote by R the common intersection of all regions R_i for $i = 1, 2, \ldots, n$ and by q^* the lowest point in R. We have the following observation.

Observation 2.5.2. *The optimal value ϵ^* is the y-coordinate of q^* and the x-coordinate of the center in the optimal solution is the x-coordinate of q^*.*

Hence, to solve the one-center problem, it is sufficient to find the lowest point q^*. To this end, we reduce the problem to a linear programming problem as follows.

Consider any function $\mathsf{Ed}(x_p, P_i)$. We let each line segment (or half-line) of the function $\mathsf{Ed}(x_p, P_i)$ define an *upper half-plane* lower bounded by the line containing the line segment. Hence, the region R_i is the common intersection of the upper half-planes defined by all (at most $O(k)$) segments of the function $\mathsf{Ed}(x_p, P_i)$. Further, the common intersection R of all region R_i's is also the common intersection of the $O(mn)$ upper half-planes defined by the segments of all functions $\mathsf{Ed}(x_p, P_i)$'s. Hence, q^* is also the lowest point of the common intersection of all $O(mn)$ upper half-planes, which can be computed in $O(mn)$ time by applying the linear time algorithm in [27]. We thus obtain the following result.

Theorem 2.5.3. *If the locations of each uncertain point are given sorted, then the discrete one-center problem can be solved in $O(mn)$ time.*

CHAPTER 3

THE ONE-CENTER PROBLEM OF UNCERTAIN POINTS ON TREE NETWORKS

3.1 Introduction

In this chapter, we consider the one-center problem for uncertain data on tree networks, where the existence (presence) of each uncertain point is described probabilistically. The results in this chapter have been published in a conference [53, 54].

3.1.1 Problem Definitions

We borrow some terminology on trees from the literature (e.g., [27, 55]). Let T be a tree. Each edge $e = (u, v)$ of T has a positive length $l(e)$. We consider e as a line segment of length $l(e)$ so that we can talk about "points" on e. Formally, a point $p = (u, v, t)$ is characterized by being located at a distance of $t \leq l(e)$ from the vertex u. The *distance* of any two points p and q on T, denoted by $d(p, q)$, is defined as the length of the simple path from p to q on T.

Let $\mathcal{P} = \{P_1, P_2, \ldots, P_n\}$ be a set of n uncertain points on T. Each uncertain point P_i has m possible locations on T, denoted by $\{p_{i1}, p_{i2}, \cdots, p_{im}\}$, and each location p_{ij} is associated with a probability $f_{ij} \geq 0$ that is the probability of P_i being at p_{ij} (which is independent of other locations), with $\sum_{j=1}^{m} f_{ij} = 1$; e.g., see Fig. 3.1. Further, each uncertain point P_i has a weight $w_i > 0$.

Consider any point x on T. For any uncertain point P_i, the *(weighted) expected distance* from x to P_i, denoted by $Ed(x, P_i)$, is defined as

$$Ed(x, P_i) = w_i \cdot \sum_{j=1}^{m} \{f_{ij} \cdot d(x, p_{ij})\}.$$

In the following, for simplicity, we use "expected distance" to refer to "weighted expected distance". We define $R(x)$ as the maximum expected distance from x to all

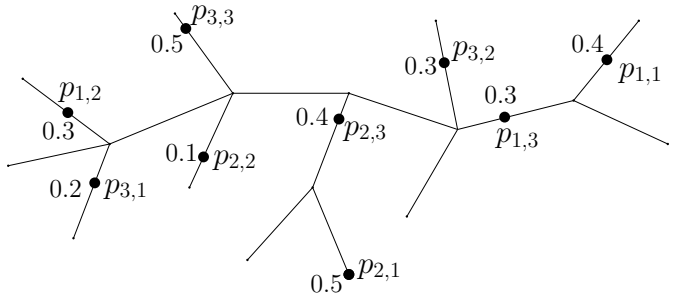

Figure 3.1. Illustrating three uncertain points P_1, P_2, P_3. Each of them has three possible locations (their probabilities are also shown).

uncertain points of $\mathcal{P}$, i.e., $R(x) = \max_{1 \le i \le n} Ed(x, P_i)$.

The *center* of T with respect to $\mathcal{P}$ is defined to be a point x^* that minimizes the value $R(x)$ among all points $x \in T$. Our goal is to compute x^*.

For any edge e of T, we assume the locations of the uncertain points of $\mathcal{P}$ on e are already given sorted on e. This means that if we traverse the edge e from one end to the other, then we can encounter those locations in order.

If T is a path network, the problem has been studied in Chapter 2, where a linear time is given for computing the center. However, if T is a tree, to the best of our knowledge, the problem has not been studied before. In this chapter, we give an $O(|T| + mn)$ time algorithm for the problem, where $|T|$ is the number of vertices of T. Note that since $\Theta(|T| + mn)$ is essentially the input size, the time complexity of our algorithm is linear, and thus our algorithm is optimal.

3.1.2 Related Work

Problems on uncertain data have been studied extensively. Two models have been commonly considered: the *existential* model [9, 11–14, 16] and the *locational* model [7, 8, 10, 15]. In the existential model, an uncertain point has a specific location but its existence is uncertain. In the locational model, an uncertain point always exists but its location is uncertain and follows a probability distribution function. Our one-center problem belongs to the locational model. In fact, the same problem under existential model is essentially the weighted one-center problem for deterministic data, which was solved in linear time [27].

As mentioned before, if T is a path network, the uncertain one-center problem has

been solved in linear time by the algorithm for the discrete k-Center Problem in Chapter 2. Algorithms for the more general uncertain k-center problems on path networks have also been given in Chapter 2.

The one-center and the more general k-center problems for the deterministic case where all data are certain have been studied extensively, as discussed below.

Megiddo [27] solved the (weighted) one-center problem on trees in linear time. For the more general k-center problem on trees, Megiddo and Tamir [31] presented an $O(n \log^2 n \log \log n)$ time algorithm for the weighted case, where n is the number of vertices of the tree, and later the running time of the algorithm was reduced to $O(n \log^2 n)$ by Cole [43]. The unweighted case was solved in linear time by Frederickson [30]. If all centers are required to be located at the vertices, then the weighted k-center problem on trees is solvable in $O(n \log^2 n)$ time [33] and the unweighted case is solvable in linear time [30].

In addition, the weighted k-center problem on the real line can be solved in $O(n \log n)$ time [31, 41, 43], and Bhattacharya and Shi [56] proposed an algorithm whose running time is linear in n but exponential in k.

If all points are on the two-dimensional plane, the unweighted one-center problem becomes the minimum enclosing circle problem, which is solvable in linear time [19]; the weighted one-center problem can be solved in $O(n \log n)$ time by the techniques in [43], where n is the number of input points (see also the discussions in [57], where an $O(n \log n)$ time randomized algorithm was given). The general k-center problem in the plane is NP-hard [19]. Efficient algorithms are known for some other special cases (e.g., [58–60] studied the *line-constrained* version where all centers are required to be on a line).

Facility location and related problems under other uncertain models have also been considered. Foul [61] studied the problem of finding the center in the plane to minimize the maximum expected distance from the center to all uncertain points, where each uncertain point has a uniform distribution in a given rectangle. Jørgenson et al. [62] considered the problem of computing the distribution of the radius of the smallest enclosing ball for a set of indecisive points each of which has multiple locations associated with probabilities in the plane. Löffler and van Kreveld [63] studied the problem of finding the

smallest enclosing circle and other related problems for imprecise points each of which is known to be contained in a planar region (e.g., a circle or a square). de Berg et al. [64] proposed an approximation algorithm to dynamically maintain Euclidean 2-centers for a set of moving points in the plane (the moving points are considered uncertain). See also the minmax regret problems, e.g., [37, 38, 65].

3.1.3 Our Approach

Note that the locations of the uncertain points of $\mathcal{P}$ may be in the interior of some edges of T. A *vertex-constrained case* happens when all locations of $\mathcal{P}$ are at vertices of T and each vertex of T contains at least one location of $\mathcal{P}$. We show (in Theorem 3.3.8) that the general problem can be reduced to the vertex-constrained case in $O(|T| + mn)$ time. In the following, unless otherwise stated, we focus our discussion on the vertex-constrained case (i.e., we assume our problem on T and $\mathcal{P}$ is a vertex-constrained case). Note that even in the vertex-constrained case, the center x^* do not have to be at a vertex of T.

To solve our problem, one immediate option is to see whether Meggido's prune-and-search techniques [27] for solving the deterministic one-center problem on trees can be applied. However, as will be discussed below, there are some "enormous" difficulties to apply Meggido's techniques directly. To overcome these difficulties, we propose new techniques, which can be viewed as a refinement of Megiddo's techniques and which we call the *refined prune-and-search*.

Megiddo's algorithm [27] is used to find the center x^* for a tree T of n vertices, where each vertex v has a weight w_v. Megiddo's algorithm first computes the centroid c of T (each of c's subtree has at most $n/2$ vertices), and then based on the weighted distances from all vertices to c, one can determine which subtree of c contains the center x^*. Suppose T' is a subtree containing x^*. The number of vertices outside T (i.e., those in $T \setminus T'$) is at least $n/2$. Consider any two vertices u and v in $T \setminus T'$. In general, by solving the equation $w_u(d(u,c) + t) = w_v(d(v,c) + t)$, one can obtain a value t_{uv} such that for every point x in T' at a distance t from c, $w_u(d(u,x)) \geq w_v(d(v,x))$ if and only if $0 \leq t \leq t_{uv}$. Based on this observation, the vertices in $T \setminus T'$ are arbitrarily arranged in roughly at least $n/4$ disjoint pairs, and for each pair u and v, the value t_{uv}

is computed. Let t^* be the median of these t_{uv} values. Depending on whether x^* is within distance t^* from c, at least $n/8$ vertices of T can be pruned. More specifically, suppose x^* is within distance t^* from c and $t^* \leq t_{uv}$ for two vertices u and v; then since $w_u(d(u,x)) \geq w_v(d(v,x))$ if and only if $0 \leq t \leq t_{uv}$, the vertex v can be pruned (i.e., the "influence" of v on x^* is "dominated" by that of u).

In our problem, the tree T has $O(mn)$ vertices, and we do the same thing and first find the centroid c of T. Although now we have uncertain points, by observations, we can still efficiently determine the subtree T' of c that contains the center x^*. However, we cannot proceed as above in Megiddo's algorithm. The reason is that for each uncertain point P_i, it may have locations in both T' and $T \setminus T'$, which prevents us from having an equation for two uncertain points and further prevents us from pruning uncertain points as in Megiddo's algorithm. Indeed, this is one of the major difficulties for us to apply Megiddo's algorithmic scheme. To overcome the difficulty, we continue to find the centroid c' of T' and determine which subtree of T' (rooted at c') containing x^*. One key idea is that we repeat this for $\log m + 1$ times, after which we obtain a subtree T'' with at most $nm/(2^{\log m+1}) = n/2$ vertices (one may wonder why not repeat this for $\log(mn)$ times so that we could obtain an edge containing x^*; the reason is that this would cost $\Omega(mn \log n)$ time). One observation is that there are at most $n/2$ uncertain points that have locations in T'', and thus, at least $n/2$ uncertain points have all locations outside T''. At this moment, we show that if T'' is connected with $T \setminus T''$ by only one vertex, then we can apply Megiddo's pruning scheme. However, another major difficulty is that T'' may be connected with $T \setminus T''$ by more than one vertex (indeed, there may be as many as $\log m + 1$ such vertices), in which case we introduce new pruning techniques to further reduce T'' to a smaller subtree T''' such that $x^* \in T'''$ and T''' is connected with $T \setminus T'''$ by either one or two vertices. For either case, we develop algorithms for the pruning. All above procedures are carefully implemented so that they together take $O(mn)$ time and eventually prune at least $n/8$ uncertain points. The total time for computing the center x^* is thus $O(mn)$.

Note that although we have assumed $\sum_{j=1}^{m} f_{ij} = 1$ for each $P_i \in \mathcal{P}$, our algorithm also works if $\sum_{j=1}^{m} f_{ij} \neq 1$. But for ease of exposition, our following discussion assumes $\sum_{j=1}^{m} f_{ij} = 1$.

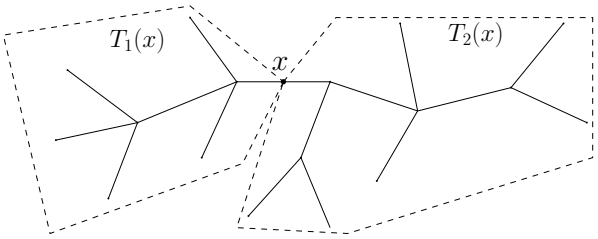

Figure 3.2. The point x has two split subtrees $T_1(x)$ and $T_2(x)$.

3.2 Preliminaries

In the following chapter, unless otherwise stated, we assume our problem is the vertex-constrained case, i.e., all locations of $\mathcal{P}$ are at vertices of T and each vertex of T has at least one location. Later in Theorem 3.3.8, we will show that the general problem can be reduced to this case in linear time. For ease of exposition, we further assume every vertex of T has only one location of $\mathcal{P}$, and thus $|T| = mn$.

In this section, we discuss a few observations, which are mainly related to determining which subtree of x contains the center x^* for any given point x on T. We begin with some notations.

For any two points p and q on T, denote by $\pi(p, q)$ the simple path on T from p to q. For any subtree T' of T and any uncertain point P_i, we call the sum of the probabilities of the locations of P_i in T' the *probability sum* of P_i in T'.

Consider any point x on T. Removing x from T will produce several subtrees of T, and we call them the *split subtrees* of x in T. More specifically, if x is in the interior of an edge, then there are two split subtrees (e.g., see Fig. 3.2); otherwise the number of subtrees is equal to the degree of x. We consider x as a vertex in each of these subtrees. However, we assign x to be contained in only one (and an arbitrary one) split subtree, but consider x as an "open vertex" in each of other subtrees. In this way, every point of T is in one and only one split subtree of x.

Let π be any simple path on T and x be any point on π. Consider any uncertain point P_i. For any location p_{ij} of P_i, the distance $d(x, p_{ij})$ is a convex (and piecewise linear) function as x changes on π [27]. Recall that the expected distance $Ed(x, P_i) = w_i \cdot \sum_{j=1}^{m} f_{ij} \cdot d(x, p_{ij})$. Since the sum of convex functions is also convex, $Ed(x, P_i)$ is convex (and piecewise linear) on π. Therefore, in general, as x moves from one end of π to

the other end, the value $Ed(x, P_i)$ first monotonically decreases and then monotonically increases. Further, recall that $R(x) = \max_{1 \le i \le n} Ed(x, P_i)$. Since the max of convex functions is also convex, $R(x)$ is convex (and piecewise linear) on π.

For each uncertain P_i, let p_i^* be a point $x \in T$ that minimizes $Ed(x, P_i)$. In fact, if we consider $w_i \cdot f_{ij}$ as the weight of p_{ij}, p_i^* is the *weighted median* of the points p_{ij} for all $j = 1, 2, \ldots, m$. Hence, we call p_i^* the *median* of P_i. Note that p_i^* may not be unique (in which case we use p_i^* to denote an arbitrary median of P_i). This case happens when there is an edge dividing T into two subtrees such that the probability sum of P_i in either subtree is exactly 0.5. Indeed, the above "degenerate case" also possibly makes the center x^* of T not unique, in which case we use x^* to refer to an arbitrary center of T.

The following lemma can be obtained readily from the results given by Kariv and Hakimi [55].

Lemma 3.2.1. *Consider any point x on T and any uncertain point P_i of $\mathcal{P}$.*

1. *If x has a split subtree whose probability sum of P_i is greater than 0.5, then p_i^* must be in that split subtree.*

2. *The point x is p_i^* if the probability sum of P_i in each of x's split subtree is less than 0.5.*

3. *The point x is p_i^* if x has a split subtree in which the probability sum of P_i is equal to 0.5.*

Consider any point x on T. If $R(x) = Ed(x, P_i)$ for some uncertain point P_i, then P_i is called a *dominating point* of x. Note that x may have multiple dominating points.

Lemma 3.2.2. *If x has a dominating point P_i whose median p_i^* is at x or x has two dominating points P_i and P_j whose medians p_i^* and p_j^* are in two different split subtrees of x, then x is x^*; otherwise, x^* is in the same split subtree of x as p_i^*.*

Proof. Suppose x has a dominating point P_i whose median p_i^* is at x. Then, imagine that x moves to any of its split subtrees to a new position x_1 and we use x_0 to denote x's original location. According to the definition of p_i^* and based on the complexity of

$Ed(x, P_i)$, as x moves, the value $Ed(x, P_i)$ is monotonically increasing, i.e, $Ed(x_1, P_i) \geq Ed(x_0, P_i)$. Clearly, $R(x_1) \geq Ed(x_1, P_i)$. Note that $R(x_0) = Ed(x_0, P_i)$ since P_i is a dominating point of x when x is at x_0. Thus, we obtain $R(x_1) \geq R(x_0)$, which proves that x_0 is a center.

Suppose x has two dominating points P_i and P_j such that the medians p_i^* and p_j^* are in two different split subtrees of x. We use a similar argument as above to prove that x is a center. Imagine that x moves to any of its split subtrees to a new position x_1 and we use x_0 to denote x's original location. Since p_i^* and p_j^* are in different split subtrees, as x moves, at least one of $Ed(x, P_i)$ and $Ed(x, P_j)$ must be monotonically increasing. Therefore, $R(x_1) \geq \max\{Ed(x_1, P_i), Ed(x_1, P_j)\} \geq Ed(x_0, P_i) = Ed(x_0, P_j) = R(x_0)$. This proves that x_0 is a center.

The remaining case is that all dominating points of x have their medians in the same split subtree T' of x and none of their medians is at x. Then, if we move x into T' infinitesimally, the value $R(x)$ will be strictly decreasing. Due to the convexity of $R(x)$, we obtain that x^* is in T'.

The lemma thus follows. □

The following corollary can be obtained from Lemmas 3.2.1 and 3.2.2.

Corollary 3.2.3. *If a point x on T has a dominating point P_i whose probability sum in a split subtree of x is less than* 0.5*, then unless x^* is at x, the split subtree does not contain x^*.*

Proof. Suppose the probability sum of P_i in a split subtree T' of x is less than 0.5. When x moves to T', the value $Ed(x, P_i)$ will be strictly increasing, and thus, p_i^* cannot be in the split subtree except at x. In other words, either p_i^* is at x or p_i^* is outside T'. Since P^* is a dominating point of x, by Lemma 3.2.2, x^* is either at x or outside T'. □

Based on the above observations, we can obtain the following result.

Lemma 3.2.4. *Given any point x on T, we can determine whether x is x^*, and if not, determine which split subtree of x contains x^* in $O(|T|)$ time.*

Proof. We first compute the expected distances $Ed(x, P_i)$ for all uncertain points $P_i \in \mathcal{P}$, which can be done in $O(mn)$ time by traversing the tree from x. Specifically, we maintain

an array $A[1 \cdots n]$ of size n, where $A[i]$ is used to compute $Ed(x, P_i)$ for each $1 \leq i \leq n$. During the traversal, we also maintain the distance from the current vertex to x. Suppose the traversal visits a vertex v of T for the first time and assume v holds a location p_{ij} of P_i. Then, we update $A[i] = A[i] + w_i \cdot f_{ij} \cdot d(x, p_{ij})$ in constant time since the distance $d(x, p_{ij})$ has been maintained during the traversal.

Next, we can find all dominating points of x in $O(n)$ time. Let P_i be an arbitrary dominating point. We compute the probability sums of P_i in all split subtrees of x by traversing the tree again, which takes $O(|T|)$ time.

If the probability sum of P_i in every split subtree of x is less than 0.5 or there is a split subtree in which the probability sum of P_i is equal to 0.5, then by Lemma 3.2.1, x is p_i^*, and further by Lemma 3.2.2, x is x^*.

Otherwise, there must be a split subtree T' in which the probability sum of P_i is greater than 0.5. By Lemma 3.2.1, T' contains p_i^*. If x does not have another dominating point, then by Lemma 3.2.2, the center x^* is in T'. Otherwise, we compute the probability sums of all dominating points in T' only, which can be done in $O(|T'|+n)$ time by traversing T'. Note that $|T'| + n = O(|T|)$ since $|T| = mn$. By Lemmas 3.2.1 and 3.2.2, if the probability sums of all dominating points in T' are greater than 0.5, then x^* is in T'; otherwise, x^* is x.

The lemma thus follows. □

Lemma 3.2.5. *If the edge of T that contains x^* is known, we can compute x^* in $O(|T|)$ time.*

Proof. We use similar techniques/observations that have been used in [21] for solving the one-dimensional version of the one-center problem.

Let $e = (u, v)$ be the edge of T that contains x^*. We first use the algorithm in Lemma 3.2.4 to determine whether u or v is x^*. If not, we know that x^* is in the interior of e. Below we assume x^* is in the interior of e.

Let x be any point in the interior of e and let t be the distance between x and u. Let $T(u)$ denote the subtree of T induced by u and the vertices $u' \in T$ such that the simple path from u' to x contains u.

We define an array $A[1 \cdots n]$ such that for each $1 \leq i \leq n$, $A[i]$ is the probability sum of P_i in $T(u)$. Since the interior of e do not contain any locations of $\mathcal{P}$ (and thus the probability sum of the locations of P_i not in $T(u)$ is $1 - A[i]$), it holds that $Ed(x, P_i) = Ed(u, P_i) + w_i \cdot (2 \cdot A[i] - 1) \cdot t$. Since $A[i]$ is a constant, $Ed(x, P_i)$ changes linearly as x moves on e. In other words, $Ed(x, P_i)$ defines a line on e. Since the center x^* is in the interior of e, we observe that x^* corresponds to the lowest point in the upper envelope of the lines defined by all uncertain points of $\mathcal{P}$.

Based on the above discussion, our algorithm for computing x^* works as follows. We first compute the array $A[1 \cdots n]$, which can be done in $O(|T|)$ time by traversing $T(u)$. Then, we compute the n lines as defined above. Finally, compute the lowest point in the upper envelope of the lines, which be done in $O(n)$ time using Meggido's linear time linear programming algorithm [27].

The lemma thus follows. □

3.3 The Refined Prune-and-Search

In this section, we give our refined prune-and-search algorithm for computing the center x^* of T. As discussed in Section 3.1.3, each round of our algorithm will prune at least $\frac{n}{8}$ uncertain points of $\mathcal{P}$ in $O(mn)$ time. After at most $O(\log n)$ rounds, only a constant number of uncertain points remain, in which case we can compute x^* in additional $O(m)$ time (see Lemma 3.3.7).

3.3.1 The Initial Pruning

A vertex c of T is called a *centroid* if every split subtree of c has no more than $|T|/2$ vertices. The centroid of T can be found in $O(|T|)$ time by traversing the tree [27, 55].

We first compute the centroid c of T. By Lemma 3.2.4, we determine whether c is x^*, and if not, determine which split subtree of c contains x^*. If c is x^*, we are done with the algorithm. Otherwise, let T_1 be the split subtree of c that contains x^*. To avoid repeatedly traversing $T \setminus T_1$ in future, we associate with c two *information arrays* $D_c[1 \cdots n]$ and $F_c[1 \cdots n]$, defined as follows. Note that according to our definition of split subtrees in Section 3.2, c may only be an "open vertex" of T_1. But from now on we consider c as a normal vertex of T_1. Note that $|T_1| \leq |T|/2 + 1$ (we assume $|T_1| \leq |T|/2$

for simplicity of the time analysis). Let $T(c) = (T \setminus T_1) \cup \{c\}$. For each $1 \leq i \leq n$, we define $F_c[i]$ to be the probability sum of P_i in $T(c)$, i.e., $F_c[i] = \sum_{p_{ij} \in P_i \cap T_1} f_{ij}$, and we define $D_c[i]$ to be the expected distance from c to the locations of P_i in T_1, i.e., $D_c[i] = w_i \cdot \sum_{p_{ij} \in P_i \cap T_1} f_{ij} \cdot d(c, p_{ij})$. We can compute the two information arrays in $O(mn)$ time by traversing $T(c)$.

Our following algorithm will continue to work on the split subtree T_1. Clearly, for any point $p \in T_1$ and $q \in T(c)$, the path $\pi(p, q)$ contains c. We call c a *connector* of T_1 since it connects T_1 with $T(c)$. We call $T(c)$ the *connector subtree* of c with respect to T_1.

As discussed in Section 3.1.3, since each uncertain point may have locations in both T_1 and $T \setminus T_1$, we cannot proceed as Megiddo's algorithm [27]. Instead, we continue to find the centroid of T_1, denoted by c_1, which can be done in $O(|T_1|)$ time. Similarly, c_1 has many split subtrees in T_1, and we want to determine whether c_1 is the center x^*, and if not, which split subtree of c_1 contains x^*. This can be done in $O(mn)$ time by Lemma 3.2.4. However, we can do faster in $O(|T_1|+n)$ time by using the two information arrays associated with the connector c without traversing the subtree $T(c)$ again. The algorithm is given in Lemma 3.3.1. Later we will generalize the algorithm to the more general case, which is necessary to bound the running time of our overall algorithm in $O(mn)$ time.

Lemma 3.3.1. *We can determine in $O(|T_1| + n)$ time whether c_1 is the center x^*, and if not, determine which split subtree of c_1 in T_1 contains x^*.*

Proof. As in Lemma 3.2.4, we first compute the expected distances $Ed(x, P_i)$ for all uncertain points $P_i \in \mathcal{P}$. To this end, we traverse T_1 starting from c_1. As in Lemma 3.2.4, during the traversal, we maintain an array $A[1 \cdots n]$ and the distance from the current vertex to c_1. Suppose the traversal visits a vertex v of T_1 for the first time; depending on whether v is the connector c, there are two cases. Note that the distance $d(c_1, v)$ is known.

If v is not c, then we proceed normally: Assume v holds a location p_{ij} of some uncertain point P_i; we update $A[i] = A[i] + w_i \cdot f_{ij} \cdot d(c_1, v)$.

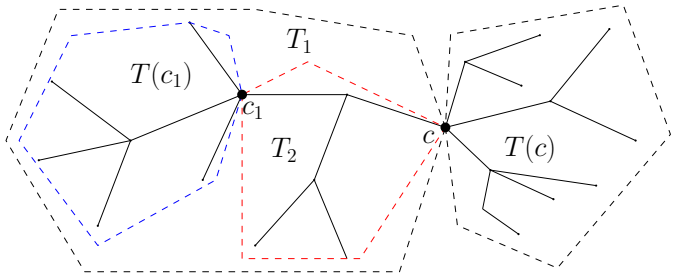

Figure 3.3. Illustrating the subtrees $T(c)$, T_1, T_2, and $T(c_1)$, where c is in T_2.

If v is c, then for each $1 \leq i \leq n$, we update $A[i] = A[i] + D_c[i] + w_i \cdot F_c[i] \cdot d(c_1, c)$. By the definition of the arrays D_c and F_c, the above correctly updates $A[i]$ since essentially $D_c[i] + w_i \cdot F_c[i] \cdot d(c_1, c)$ is the expected distance from c_1 to the locations of P_i in $T(c)$.

Once all vertices of T_1 are visited, the expected distances $Ed(c_1, P_i)$ for all $P_i \in \mathcal{P}$ are computed. It is easy to see that the algorithm spends $O(n)$ time on the connector c and spends constant time on each of other vertices of T_1. Therefore, the algorithm runs in $O(|T_1| + n)$ time.

The rest of the algorithm is similar to that of Lemma 3.2.4 with a difference that we need to use the array F_c associated with c to compute the probability sums, which can also be done in $O(|T_1| + n)$ time.

Hence, the total running time of the algorithm is $O(|T_1| + n)$. □

If c_1 is x^*, we are done. Otherwise let T_2 denote the split subtree of c_1 in T_1 that contains x^*. Note that c may or may not be in T_2. Define $T(c_1)$ to be the subtree of T induced by c_1 and the vertices $v \in T$ such that the simple path from v to any vertex of T_2 contains c_1. In fact, $T(c_1) = (T_1 \setminus T_2) \cup \{c_1\}$ if c is in T_2 (e.g., see Fig. 3.3) and $T(c_1) = T(c) \cup (T_1 \setminus T_2) \cup \{c_1\}$ otherwise.

Similarly, we associate c_1 with two information arrays $F_{c_1}[1 \cdots n]$ and $D_{c_1}[1 \cdots n]$, where $F_{c_1}[i] = \sum_{p_{ij} \in P_i \cap T(c_1)} f_{ij}$ and $D_{c_1}[i] = w_i \cdot \sum_{p_{ij} \in P_i \cap T(c_1)} f_{ij} d(c_1, p_{ij})$ for each $1 \leq i \leq n$. Similar to the algorithm in Lemma 3.3.1, we can compute the above two arrays in $O(|T_1| + n)$ time, regardless whether c is in T_2 or not. We call c_1 a *connector* of T_2 and call $T(c_1)$ the *connector subtree* of c_1. If c is in T_2, c is also a connector of T_2. Hence, T_2 may have at most two connectors. Note that each connector of T_2 must be a leaf of T_2.

Next we continue the above procedure recursively on T_2.

In general, suppose we have performed the above procedure for h recursive steps and obtain a subtree T_h, which may have at most h connectors. Each connector of T_h is a leaf of T_h and is associated with two information arrays. Then, we compute the centroid c_h of T_h in $|T_h|$ time. By generalizing the algorithm in Lemma 3.3.1 and using the information arrays associated at the connectors, in $O(|T_h| + nh)$ time we can determine whether c_h is x^*, and if not, which split subtree of c_h in T_h contains x^* (i.e., the algorithm spends $O(n)$ time on each connector and $O(1)$ time on each of other vertices of T_h). If c_h is x^*, we are done with the algorithm. Otherwise, let T_{h+1} be the split subtree of c_h containing x^*. The vertex c_h is a connector of T_{h+1}, and the connectors of T_h that are in T_{h+1} are also connectors of T_{h+1}. Hence, T_{h+1} has at most $h+1$ connectors, each of which is a leaf of T_{h+1}. We define the *connector subtree* $T(c_h)$ of c_h similarly (i.e., $T(c_h)$ is the subtree of T induced by c_h and the vertices $v \in T$ such that the simple path from v to any vertex of T_{h+1} contains c_h). Also, we associate two information arrays $F_{c_h}[1 \cdots n]$ and $D_{c_h}[1 \cdots n]$ with c_h (i.e., for each $1 \leq i \leq n$, $F_{c_h}[i] = \sum_{p_{ij} \in P_i \cap T(c_h)} f_{ij}$ and $D_{c_h}[i] = w_i \cdot \sum_{p_{ij} \in P_i \cap T(c_h)} f_{ij} d(c_h, p_{ij})$), and the two arrays can be computed in $O(|T_h| + nh)$ time.

We perform the above procedure for $h = 1 + \log m$ recursive steps, after which we obtain a tree T_h. By the definition of centroids, $|T_h| \leq |T|/2^h = (mn)/2^h = n/2$. Therefore, if we let $T_0 = T$, the running time of all above recursive steps is $O(\sum_{k=1}^{h}(|T_{k-1}| + n(k-1)))$, which is $O(mn + nh^2) = O(mn + n\log^2 m) = O(mn)$ since $|T| = mn$ and $|T_k| \leq |T_{k-1}|/2$ for each $1 \leq k \leq h$.

We refer to the above algorithm as *the initial pruning step.*

Since $|T_h| \leq n/2$, there are at most $n/2$ uncertain points of $\mathcal{P}$ that have locations in T_h. In other words, we have the following observation, which is crucial to our algorithm.

Observation 3.3.2. *There are at least $n/2$ uncertain points $P_i \in \mathcal{P}$ such that P_i does not have any location in T_h.*

Let C denote the number of connectors in T_h. Depending on the value of C, our algorithm will proceed accordingly for three cases: $C = 1$, $C = 2$, and $C > 2$. If $C = 1$, although the implementation details are quite different, we can still apply Meggido's

pruning scheme [27] due to the following *key property*: if an uncertain point P_i does not have any location in T_h, then all its locations must be in the connector subtree $T(\hat{c})$, where $\hat{c}$ is the only connector of T_h. However, if $C > 2$, although an uncertain point P_i may not have any location in T_h, the above key property does not hold any more, which makes Meggido's pruning scheme fail. To overcome the difficult, if $C > 2$, we will further process the subtree T_h using different techniques. For solving the case $C = 2$, we will reduce the problem to the case $C = 1$, and for solving the case $C > 2$, we will reduce the problem to either the case $C = 2$ or the case $C = 1$. In the following, we first present our algorithm for the case $C = 1$.

Let $\mathcal{P}'$ denote the set of uncertain points $P_i \in \mathcal{P}$ such that P_i does not have any location in T_h. We can easily find $\mathcal{P}'$ in $O(nm)$ time by traversing T_h. By Observation 3.3.2, $|\mathcal{P}'| \geq n/2$.

3.3.2 The Case $C = 1$

In this case, the subtree T_h has only one connector, denoted by $\hat{c}$. Hence, all vertices that are not in T_h are in the connector subtree $T(\hat{c})$. Recall that $\hat{c}$ is associated with two information arrays $D_{\hat{c}}[1 \cdots n]$ and $F_{\hat{c}}[1 \cdots n]$.

For each $P_i \in \mathcal{P}'$, since all locations of P_i are in $T(\hat{c})$, it holds that $Ed(x, P_i) = Ed(\hat{c}, P_i) + w_i \cdot t$, where x is a point in T_h at a distance t from $\hat{c}$. Note that $Ed(\hat{c}, P_i)$ is essentially $D_{\hat{c}}[i]$, and thus it is already known.

Consider any pair P_i and P_j of uncertain points in $\mathcal{P}'$. Without of loss generality, assume $Ed(\hat{c}, P_i) \geq Ed(\hat{c}, P_j)$. If $w_i < w_j$, then by solving the equation $Ed(\hat{c}, P_i) + w_i \cdot t = Ed(\hat{c}, P_j) + w_j \cdot t$, we can obtain a value t_{ij} such that for every x in T_h at a distance t from $\hat{c}$, $Ed(x, P_i) \geq Ed(x, P_j)$ if and only if $0 \leq t \leq t_{ij}$. If $w_i \geq w_j$, $Ed(x, P_i) \geq Ed(x, P_j)$ holds for any x in T_h, and thus P_j can be pruned immediately (since it is "dominated" by P_i).

Based on the above discussions, we arbitrarily arrange the uncertain points of $\mathcal{P}'$ into a set Σ of $|\mathcal{P}'|/2$ disjoint pairs, and for each pair (P_i, P_j), we compute the value t_{ij}. Let t^* denote the median of these t_{ij} values. Suppose we have already known whether x^* is within the distance t^* from $\hat{c}$ on T_h (we will discuss this step later); in either case, we can prune exactly one uncertain point from each pair of Σ. Since $|\mathcal{P}'| \geq n/2$ and

$|\Sigma| \geq n/4$, the total number of pruned uncertain points is at least $n/8$. At this point, we have reduced our problem to the same problem on a tree T^+ of at most $7n/8$ uncertain points, defined as follows. First, let $T^+ = T_h$. Then, consider any pair (P_i, P_j) of Σ. Without of loss of generality, assume P_i is not pruned. For each location p_{ij} of P_i, we create a vertex for T^+ connecting to $\hat{c}$ directly by an edge of length $d(p_{ij}, \hat{c})$. Also let w_i still be the weight of P_i. In this way, the tree T^+ has at most $7n/8$ uncertain points and at most $7nm/8$ vertices. Based on our above pruning procedure, the center of T^+ is also the center of the original tree T. Note that the above way of constructing T^+ can be easily done in $O(mn)$ time.

It remains to determine whether x^* is within the distance of t^* from $\hat{c}$ on T_h. As in [27], by traversing T_h from $\hat{c}$, in $|T_h|$ time we can find all points $q \in T_h$ such that $d(\hat{c}, q) = t^*$, and we use $Q(\hat{c}, T_h)$ to denote the set of these points. Note that each point $q \in Q(\hat{c}, T_h)$ has a split subtree that contains $\hat{c}$, and we assign q to be contained in that split subtree (recall that we allow q to be in only one of its split subtrees); for any point x in other split subtrees of q, it holds that $d(\hat{c}, x) > t^*$. Hence, the set of all points x of T_h with $d(\hat{c}, x) > t^*$ can be represented as the union of split subtrees of the points in $Q(\hat{c}, T_h)$ that do not contain $\hat{c}$, and we use $\mathcal{T}(\hat{c}, T_h)$ to denote the above set of split subtrees. Our goal is to determine whether x^* is in any subtree of $\mathcal{T}(\hat{c}, T_h)$. To this end, we solve a more general problem, called a *center-detecting problem*, defined as follows. Another reason for solving this more general problem is that our algorithms for the other two cases $C = 2$ and $C > 2$ will need it.

Consider the input tree T. Let y be any vertex of T. Consider any point $x \in T$ with $x \neq y$. One split subtree of x, denoted by $\tau(x)$, contains y, and we assign x to be in $\tau(x)$. We call other split subtrees of x than $\tau(x)$ the *y-exclusive* split subtrees of x. Note that since the y-exclusive split subtrees of x each contain x as an "open vertex", they are actually pairwise disjoint. Let Y be a set of points on T with $y \notin Y$ and $\mathcal{T}(Y)$ be any subset of the set of all y-exclusive subtrees of all points of Y with the following *disjoint property*: any two subtrees of $\mathcal{T}(Y)$ are disjoint (e.g., see Fig. 3.4; one can verify that this condition implies that $|Y| \leq |T|$). The *center-detecting problem* on $(y, Y, \mathcal{T}(Y))$ is to determine whether the center x^* is located in one of the subtrees of $\mathcal{T}(Y)$. In Section 3.3.3, we will give an $O(|T|)$ time algorithm to solve the center-detecting problem.

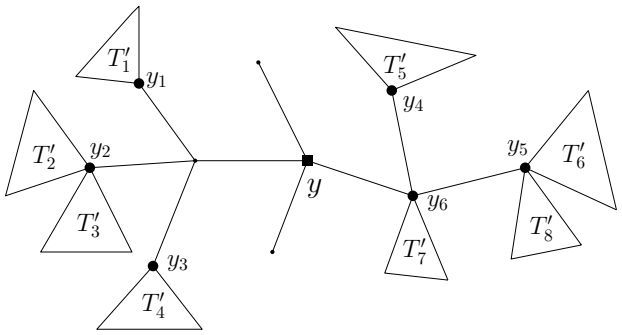

Figure 3.4. Illustrating an example for the center-detecting problem. In this example, $Y = \{y_1, y_2, \ldots, y_6\}$ and $\mathcal{T}(Y) = \{T'_1, \ldots, T'_8\}$ shown with triangles. Note that although T'_2 and T'_3 share a common point y_2, since y_2 is considered as an open vertex in each of them, T'_2 and T'_3 are disjoint. This is also the case for T'_6 and T'_8.

By using this result, in $O(|T|)$ time, we can solve the above problem of determining whether x^* is in the subtrees of $\mathcal{T}(\hat{c}, T_h)$ as follows. Recall that $T = T(\hat{c}) \cup T_h$ and $\hat{c}$ is a leaf of T_h. By the definitions of the points of $Q(\hat{c}, T_h)$, we observe that $\mathcal{T}(\hat{c}, T_h)$ is actually the set of the $\hat{c}$-exclusive subtrees of all points of $Q(\hat{c}, T_h)$ in T and any two subtrees in $\mathcal{T}(\hat{c}, T_h)$ are disjoint. Hence, although our problem is on the subtree T_h, we can actually work on the tree T. Therefore, to determine whether x^* is in the subtrees of $\mathcal{T}(\hat{c}, T_h)$ is to solve the center-detecting problem on $(\hat{c}, Q(\hat{c}, T_h), \mathcal{T}(\hat{c}, T_h))$ and T.

3.3.3 The Center-Detecting Problem

In this section, we give an $O(|T|)$ time algorithm to solve the center-detecting problem on $(y, Y, \mathcal{T}(Y))$, as defined above.

A similar (but somewhat constrained) problem also appears in [27] on the deterministic one-center problem, and Megiddo [27] gave an $O(|T|)$ time algorithm, which simply traverses all subtrees of $\mathcal{T}(Y)$. Our problem is more challenging, which is partially due to that each uncertain point may have locations either in or outside those subtrees of $\mathcal{T}(Y)$. In the following, we first give some observations, based on which our algorithm will be developed.

To simplify notation, let $\mathcal{T} = \mathcal{T}(Y)$. For each subtree $\tau \in \mathcal{T}$, let $y(\tau)$ be the point in Y such that τ is the y-exclusive split subtree of $y(\tau)$.

Consider any subtree $\tau \in \mathcal{T}$. Let $\mathcal{P}(\tau)$ be the set of uncertain points P_i whose probability sums in τ are greater than 0.5. Note that $\mathcal{P}(\tau) = \emptyset$ is possible. Define

$g(\tau) = \max_{P_i \in \mathcal{P}(\tau)} Ed(y(\tau), P_i)$, and $g(\tau) = 0$ if $\mathcal{P}(\tau) = \emptyset$. Define $g(\mathcal{T}) = \max_{\tau \in \mathcal{T}} g(\tau)$. Note that $\mathcal{P}(\tau) \cap \mathcal{P}(\tau') = \emptyset$ for any two subtrees τ and τ' of $\mathcal{T}$.

Lemma 3.3.3. *For any $\tau \in \mathcal{T}$, if $g(\tau) < g(\mathcal{T})$, then the center x^* cannot be in τ.*

Proof. Consider any subtree $\tau \in \mathcal{T}$ with $g(\tau) < g(\mathcal{T})$. Let P_j be a dominating point of $y(\tau)$, i.e., $R(y(\tau)) = Ed(y(\tau), P_j)$. We first show that P_j is not in $\mathcal{P}(\tau)$.

Let τ' be a subtree of $\mathcal{T}$ such that $g(\tau') = g(\mathcal{T})$, and let P_i be a point of $\mathcal{P}(\tau')$ such that $g(\tau') = Ed(y(\tau'), P_i)$. Based on the disjoint property of Y and $\mathcal{T}$, we can obtain that $y(\tau)$ is not in τ'. Since the probability sum of P_i in τ' is greater than 0.5 and $y(\tau)$ is not in τ', it holds that $Ed(y(\tau), P_i) \geq Ed(y(\tau'), P_i) = g(\tau') = g(\mathcal{T}) > g(\tau)$. On the other hand, recall that $R(y(\tau))$ is the maximum expected distance from all uncertain points of $\mathcal{P}$ to $y(\tau)$. Thus, $R(y(\tau)) \geq Ed(y(\tau), P_i)$, and we obtain $R(y(\tau)) > g(\tau)$. Therefore, P_j, as a dominating point of $y(\tau)$, cannot be in $\mathcal{P}(\tau)$, since otherwise we would obtain $g(\tau) = Ed(y(\tau), P_j) = R(y(\tau))$, incurring contradiction.

Since $P_j \notin \mathcal{P}(\tau)$, the probability sum of P_j in τ is less than or equal to 0.5.

If the probability sum of P_j in τ is less than 0.5, then since P_j is a dominating point of $y(\tau)$ and τ is a split subtree of $y(\tau)$, by Corollary 3.2.3 (recall that $y(\tau)$ is not in τ, i.e., $y(\tau)$ is an "open" vertex of τ), the center x^* cannot be in τ.

If the probability sum of P_j in τ is equal to 0.5, then by Lemma 3.2.1, $y(\tau)$ is the median p_j^* of P_j. Further, since P_j is a dominating point of $y(\tau)$, by Lemma 3.2.2, $y(\tau)$ is x^*. Recall that $y(\tau)$ is not in τ. Hence, x^* is not in τ.

The lemma thus follows. □

Lemma 3.3.4. *If there exist two subtrees τ_1 and τ_2 in $\mathcal{T}$ such that $g(\tau_1) = g(\tau_2) = g(\mathcal{T})$, then the center x^* cannot be in any subtree of $\mathcal{T}$.*

Proof. Let τ_1 and τ_2 be two subtrees in $\mathcal{T}$ such that $g(\tau_1) = g(\tau_2) = g(\mathcal{T})$. We first show that x^* cannot be in τ_1.

Let P_i be a point of $\mathcal{P}(\tau_2)$ such that $g(\tau_2) = Ed(y(\tau_2), P_i)$. As in the proof of Lemma 3.3.3, since the probability sum of P_i in τ_2 is greater than 0.5 and $y(\tau_1)$ is not in τ_2, it holds that $Ed(y(\tau_1), P_i) \geq Ed(y(\tau_2), P_i) = g(\tau_2) = g(\tau_1)$. On the other hand, since $R(y(\tau_1)) \geq Ed(y(\tau_1), P_i)$, we obtain that $R(y(\tau_1)) \geq g(\tau_1)$. Depending on whether $R(y(\tau_1)) = g(\tau_1)$, there are two cases.

If $R(y(\tau_1)) = g(\tau_1)$, then P_i is a dominating point of $y(\tau_1)$. Since $P_i \in \mathcal{P}(\tau_2)$, the probability sum of P_i in τ_2 is greater than 0.5, which implies that the probability sum of P_i in τ_1 is less than 0.5 because τ_1 and τ_2 are disjoint. Consequently, by Corollary 3.2.3, we obtain that x^* cannot be in τ_1 since P_i is a dominating point of $y(\tau_1)$.

If $R(y(\tau_1)) > g(\tau_1)$, then for any dominating point P_j of $y(\tau_1)$, P_j cannot be in $\mathcal{P}(\tau_1)$ since otherwise we would obtain $g(\tau_1) = Ed(y(\tau_1), P_j) = R(y(\tau_1))$, incurring contradiction. Hence, the probability sum of P_j in τ_1 is less than or equal to 0.5. By the similar analysis as that for Lemma 3.3.3, we can show that x^* is not in τ_1.

The above proves that x^* cannot be in τ_1. Analogously, we can show that x^* cannot be in any subtree τ with $g(\tau) = g(\mathcal{T})$. Combining with Lemma 3.3.3, we conclude that x^* cannot be in any subtree of $\mathcal{T}$. The lemma thus follows. □

Suppose we have computed the value $g(\tau)$ for each $\tau \in \mathcal{T}$; based on Lemmas 3.3.3 and 3.3.4, we can solve the center-detecting problem in $O(mn)$ time in the following way. First, we compute $g(\mathcal{T})$ and check whether there exist two subtrees τ_1 and τ_2 such that $g(\tau_1) = g(\tau_2) = g(\mathcal{T})$. If yes, by Lemma 3.3.4, x^* cannot be in any subtree of $\mathcal{T}$ and we are done. Otherwise, let τ be the subtree such that $g(\tau) = g(\mathcal{T})$. By Lemma 3.3.3, we only need to determine whether x^* is in τ, which can be done by applying Lemma 3.2.4 on $x = y(\tau)$. Since $|\mathcal{T}| \leq |T| = mn$ (recall that due to the disjoint property of Y and $\mathcal{T}$, it always holds that $|\mathcal{T}| \leq |T|$), the above can be done in $O(mn)$ time.

It remains to compute the value $g(\tau)$ for each $\tau \in \mathcal{T}$. Below we present an $O(|T|)$ time algorithm.

We first compute the set $\mathcal{P}(\tau)$ for all $\tau \in \mathcal{T}$. We use an array $A[1 \ldots n]$ of size n. Initially set $A[i] = 0$ for each $1 \leq i \leq n$. Consider any $\tau \in \mathcal{T}$. We traverse the subtree τ starting from $y(\tau)$. Whenever we visit a vertex v for the first time and suppose v holds a location p_{ij} of P_i, we update $A[i] = A[i] + f_{ij}$. After the update, if $A[i] > 0.5$, then we add P_i to $\mathcal{P}(\tau)$ (e.g., we can use a list to store $\mathcal{P}(\tau)$). In this way, we can obtain the set $\mathcal{P}(\tau)$ in time linear to the size of τ. After traversing τ and before traversing the next subtree of $\mathcal{T}$, we reset the non-zero elements of A to zero. However, to avoid the $\Omega(n)$ time, we do not scan the entire array A. Instead, we can traverse the subtree τ again and whenever we visit a location p_{ij} of P_i, we reset $A[i] = 0$. In this way, we can reset

A in time linear to the size of τ. We continue to traverse other subtrees of $\mathcal{T}$ as above. Hence, we can compute $\mathcal{P}(\tau)$ for all $\tau \in \mathcal{T}$ in time linear to the total size of all subtrees of $\mathcal{T}$, which is bounded by $O(mn)$ since the subtrees of $\mathcal{T}$ are pairwise disjoint.

We proceed to compute $g(\tau)$ for all $\tau \in \mathcal{T}$. Recall that for any two subtrees τ_1 and τ_2 of $\mathcal{T}$, $\mathcal{P}(\tau_1) \cap \mathcal{P}(\tau_2) = \emptyset$. Hence, each $P_i \in \mathcal{P}$ belongs to $\mathcal{P}(\tau)$ for at most one subtree τ of $\mathcal{T}$. We organize the set $\mathcal{P}(\tau)$ for all $\tau \in \mathcal{T}$ in a way that given P_i, if P_i is in $\mathcal{P}(\tau)$ for some τ, then we can obtain τ in constant time. This can be easily done by using an array $B[1 \dots n]$, i.e., $B[i] = \tau$ if P_i is in $\mathcal{P}(\tau)$. Note that B can be constructed in $O(n)$ time by scanning $\mathcal{P}(\tau)$ for all $\tau \in \mathcal{T}$.

Next we traverse the entire tree T to compute $g(\tau)$ for all $\tau \in \mathcal{T}$. During the traversal, we maintain another array $\alpha[1 \dots n]$. For each $1 \leq i \leq n$, initially we set $\alpha[i] = 0$, and after the algorithm finishes, if $B[i] = \tau$, then $\alpha[i]$ will be equal to $Ed(y(\tau), P_i)$. During the traversal, suppose we visit a vertex v for the first time and v holds a location p_{ij} from an uncertain pint P_i, we update $\alpha[i] = \alpha[i] + w_i \cdot f_{ij} \cdot d(v, y(\tau))$, where $\tau = B[i]$, provided that we know the distance $d(v, y(\tau))$. Below in Lemma 3.3.5 we will give a data structure that can compute $d(v, y(\tau))$ in constant time, with $O(mn)$ time preprocessing. After the entire tree T is traversed, the values $Ed(y(\tau), P_i)$ for all $P_i \in \mathcal{P}(\tau)$ and all $\tau \in \mathcal{T}$ are computed. Then, we can compute the values $g(\tau)$ for all $\tau \in \mathcal{T}$ in additional $O(n)$ time.

Lemma 3.3.5. *With $O(|T|)$ time preprocessing, given any two points p and q in $V(T) \cup Y$, where $V(T)$ is the set of all vertices of T, we can compute the distance $d(p, q)$ in $O(1)$ time.*

Proof. For each point in $Y \cup \{y\}$, if it is not a vertex of T, then it is in the interior of an edge of T, and we insert the point to T as a new vertex of T. In this way, since $|Y| \leq |T|$, the number of vertices in the new tree T is $O(mn)$.

We consider y as the root of the new tree T. For any two vertices u and v of T, let $lca(u, v)$ denote the lowest common ancestor of u and v. To determine the distance $d(u, v)$, we have the following easy observation: $d(u, v) = d(u, w) + d(w, v)$, where $w = lca(u, v)$, and further, $d(u, w) = d(y, u) - d(y, w)$ and $d(w, v) = d(y, v) - d(y, w)$; consequently, $d(u, v) = d(y, u) + d(y, v) - 2 \cdot d(y, w)$.

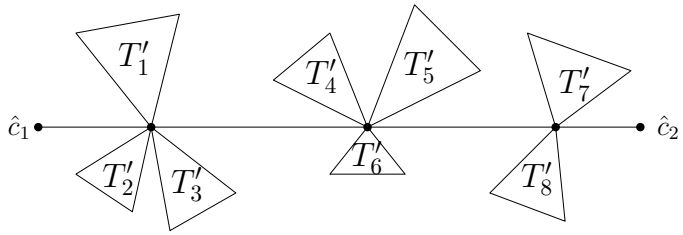

Figure 3.5. Illustrating the tree T_h for the case $C = 2$, where $\mathcal{T}(V) = \{T'_1, \ldots T'_8\}$ are shown with triangles.

Based on the above observation, in the preprocessing we compute $d(y, v)$ for each vertex v of T, which can be done in $O(|T|)$ time by traversing the tree. Further, in $|T|$ time we can build a lowest common ancestor data structure [66,67] on T such that given any two vertices u and v, $lca(u, v)$ can be found in constant time.

Given any two query vertices u and v of T, we first determine $w = lca(u, v)$ in constant time by using the lowest common ancestor data structure, and then compute $d(u, v) = d(y, u) + d(y, v) - 2 \cdot d(y, w)$ again in constant time.

The lemma thus follows. □

The above discussion leads to the following result.

Lemma 3.3.6. *The center-detecting problem on T can be solved in $O(|T|)$ time.*

3.3.4 The Case $C = 2$

In this case, the subtree T_h has two connectors, denoted by $\hat{c}_1$ and $\hat{c}_2$. Recall that each $\hat{c}_k$ is associated with two arrays $D_{\hat{c}_k}[1 \cdots n]$ and $F_{\hat{c}_k}[1 \cdots n]$ for $k = 1, 2$. The techniques for the previous case $C = 1$ do not work here. The reason is that although every uncertain point in $\mathcal{P}'$ does not have any location in T_h, it may have locations in both connector subtrees $T(\hat{c}_1)$ and $T(\hat{c}_2)$. We use a different approach given below.

Consider the path $\pi(\hat{c}_1, \hat{c}_2)$ from $\hat{c}_1$ to $\hat{c}_2$. Recall that $\hat{c}_1$ and $\hat{c}_2$ are leaves of T_h because they are connectors. Let V denote the set of vertices of $\pi(\hat{c}_1, \hat{c}_2)$ except $\hat{c}_1$ and $\hat{c}_2$. Consider any vertex v of V. Let $\mathcal{T}(v)$ be the set of the split subtrees of v that do not contain either $\hat{c}_1$ or $\hat{c}_2$. We assume v is not contained in any subtree of $\mathcal{T}(v)$. Note that $\mathcal{T}(v)$ is empty if the degree of v is two. Let $\mathcal{T}(V) = \cup_{v \in V} \mathcal{T}(v)$ (e.g., see Fig. 3.5). Thus, T_h is the union of the subtrees of $\mathcal{T}(V)$ and $\pi(\hat{c}_1, \hat{c}_2)$.

First of all, we want to determine whether the center x^* is in one of the subtrees of $\mathcal{T}(V)$. Indeed, this is an instance of the center-detecting problem. To see this, each subtree of $\mathcal{T}(V)$ is a $\hat{c}_1$-exclusive split subtree of some point in V, and any two subtrees of $\mathcal{T}(V)$ are disjoint. Further, $\hat{c}_1$ is a leaf of T_h. Hence, as discussed in Section 3.3.2, it is an instance of the center-detecting problem on $(\hat{c}_1, V, \mathcal{T}(V))$ and T, which can be solved in $O(mn)$ time by Lemma 3.3.6.

If x^* is contained in a subtree τ of $\mathcal{T}(V)$, and assume τ is the split subtree of $v \in V$. Then, the problem essentially becomes the first case where $C = 1$. Indeed, we can consider v as the "connector" of τ. Recall that every uncertain point in $\mathcal{P}'$ does not have any location in T_h, since τ is a subtree of T_h, every uncertain point in $\mathcal{P}'$ does not have any location in τ as well. Since τ has only one connector, by using the same techniques as for the case $C = 1$, we can prune at least $n/8$ uncertain points.

If x^* is not contained in any subtree of $\mathcal{T}(V)$, then x^* must be in the path $\pi(\hat{c}_1, \hat{c}_2)$. Consider any uncertain point $P_i \in \mathcal{P}'$. Since P_i does not have any location in T_h, P_i does not have any location in $\pi(\hat{c}_1, \hat{c}_2)$. Hence, if x is a point on $\pi(\hat{c}_1, \hat{c}_2)$ at a distance t from $\hat{c}_1$, then it is not difficult to see that $Ed(x, P_i) = Ed(\hat{c}_1, P_i) + w_i \cdot (F_{\hat{c}_1}[i] - F_{\hat{c}_2}[i]) \cdot t$ (recall that $F_{\hat{c}_k}[i]$ is the probability sum of P_i in $T(\hat{c}_k)$ for $k = 1, 2$). Note that $F_{\hat{c}_1}[i] - F_{\hat{c}_2}[i]$ is constant as long as x is in $\pi(\hat{c}_1, \hat{c}_2)$ since P_i does not have any location in $\pi(\hat{c}_1, \hat{c}_2)$. Hence, as x moves from $\hat{c}_1$ to $\hat{c}_2$ along $\pi(\hat{c}_1, \hat{c}_2)$, the value $Ed(x, P_i)$ changes linearly.

The above observation leads to the following algorithm for pruning the uncertain points of $\mathcal{P}'$. We again arbitrarily arrange the points of $\mathcal{P}'$ into $|\mathcal{P}'|/2$ pairs. In general, for each such pair (P_i, P_j), by solving the equation $Ed(\hat{c}_1, P_i) + w_i \cdot (F_{\hat{c}_1}[i] - F_{\hat{c}_2}[i]) \cdot t = Ed(\hat{c}_1, P_j) + w_j \cdot (F_{\hat{c}_1}[j] - F_{\hat{c}_2}[j]) \cdot t$, we can determine a value t_{ij} such that for a point x in $\pi(\hat{c}_1, \hat{c}_2)$ at a distance t from $\hat{c}_1$, $Ed(x, P_i) \geq Ed(x, P_j)$ if and only if $0 \leq t \leq t_{ij}$. In this way, we can obtain $|\mathcal{P}'|/2$ such values t_{ij}, and let t^* be the median of them. Let q^* be the point on $\pi(\hat{c}_1, \hat{c}_2)$ at distance t^* from $\hat{c}_1$. Again, by Lemma 3.2.4, we can determine in $O(mn)$ time whether q^* is x^*, and if not, which split subtree of q^* contains x^* (and thus determine whether x^* is within the distance t^* from $\hat{c}_1$). In either case, we can prune an uncertain point from each of the above pairs of $\mathcal{P}'$, and thus prune a total of at least $n/8$ uncertain points due to $|\mathcal{P}'| \geq n/2$.

3.3.5 The Case $C > 2$

In this case, T_h has more than two connectors. Indeed, this is the most general case. Clearly, the techniques for the case $C = 2$, which reply on a path $\pi(\hat{c}_1, \hat{c}_2)$, are not applicable any more. We use a new approach by "shrinking" T_h until the problem is reduced to one of the previous two cases.

A vertex z of T_h is called a *connector-centroid* if each split subtree of z has no more than $C/2$ connectors (such a vertex may not be unique). The main idea of our algorithm is similar to the scheme of the initial pruning in Section 3.3.1. We first find a connector-centroid z of T_h and then remove the split subtrees of z that do not contain the center x^*. We work on the remaining split subtree of z recursively until there are at most two connectors left, at which moment we have reduced the problem to the case of either $C = 2$ or $C = 1$. The details are given below.

We first find a connector-centroid z of T_h. This can be done in $O(|T_h|)$ time by a traversal of T_h, always moving in the direction in which the number of connectors, in the subtree entered into, is being maximized (or by modifying the algorithm in [55] for finding the ordinary centroid). As the algorithm in Section 3.3.1, by traversing T_h and using the information arrays associated with the connectors, we can determine in $O(Cn+|T_h|)$ time whether z is the center x^*, and if not, which split subtree of z contains x^*. If z is x^*, we are done with the algorithm. Otherwise, let $T_h^1(z)$ denote the split subtree of z in T_h that contains x^*. Further, we consider z as a "connector" of $T_h^1(z)$. By the definition of the connector-centroid, $T_h^1(z)$ has at most $C/2+1$ connectors. Also as in Section 3.3.1, we define the *connector subtree* $T(z)$ for z as the subtree of T induced by z and the vertices $v \in T$ such that the simple path from v to any vertex of $T_h^1(z)$ contains z. Similarly, we associate two information arrays $F_z[1 \cdots n]$ and $D_z[1 \cdots n]$ with z (i.e., for each $1 \leq i \leq n$, $F_z[i] = \sum_{p_{ij} \in P_i \cap T(z)} f_{ij}$ and $D_z[i] = w_i \cdot \sum_{p_{ij} \in P_i \cap T(z)} f_{ij} d(z, p_{ij})$). As in Section 3.3.1, the two arrays can be computed in $O(Cn + |T_h|)$ time by traversing T_h.

We continue the above algorithm recursively on $T_h^1(z)$ until after l steps we obtain a subtree $T_h^l(z)$ of at most two connectors, at which moment we have reduced the problem to one of the previous two cases (and thus we can use the algorithms for the previous case to proceed). Clearly, $l = O(\log C)$. For the running time, suppose for

each $1 \leq k \leq l$, we refer to the *k-th step* as for determining the subtree $T_h^k(z)$ and computing the corresponding information (e.g., the information arrays). As discussed above, the first step takes $O(nC + |T_h|)$ time, and similarly, the second step can be done in $O(nC/2 + |T_h|)$ time because there are only at most $C/2 + 1$ connectors in $T_h^1(z)$. In general, the k-th step can be done in $O(nC/2^{k-1} + |T_h|)$ time for each $1 \leq k \leq l$. Recall that T_h was obtained in the initial pruning step in Section 3.3.1 and $|T_h| \leq n/2$. Since $l = O(\log C)$ and $C = O(\log m)$, the total running time for obtaining the subtree $T_h^l(z)$ (and thus reducing the problem to the previous two cases) is bounded by $\sum_{k=1}^{l}(nC/2^{k-1} + n/2) = O(nC + nl) = O(n \log m)$.

As a summary, for the case $C > 2$, within $O(mn)$ time we can also prune at least $n/8$ uncertain points.

3.3.6 Wrapping Things Up

The above gives an $O(mn)$ time algorithm that computes a tree T^+ of at most $7n/8$ uncertain points and at most $7mn/8$ vertices, such that the center of T^+ is x^*. We continue the same procedure recursively on T^+ until we obtain a tree T^* with only a constant number of uncertain points (hence T^* has $O(m)$ vertices). The total time is $O(mn)$. The following lemma computes the center x^* on T^* in additional $O(m)$ time, and the algorithm is similar to the algorithmic scheme of the initial pruning in Section 3.3.1.

Lemma 3.3.7. *The center x^* on T^* can be computed in $O(m)$ time.*

Proof. Let H denote the number of uncertain points on T^*, and $H = O(1)$. Note that $|T^*| = Hm$. We reorder the uncertain points of T^* as $P_1, P_2, \ldots, P_H$.

We first compute the centroid b of T^*. By Lemma 3.2.4, in $O(m)$ time we can determine whether b is x^*, and if not, determine which split subtree of b contains x^*. If b is x^*, we are done. Otherwise, assume T_1^* is the split subtree of b that contains x^*. As in Section 3.3.1, we consider b as the connector of T_1^*. Similarly, we define the connector subtree $T^*(b)$ as the subtree of T^* induced by b and the vertices $v \in T^*$ such that the simple path from v to any vertex of T_1^* contains b. Also, we associate b with two information arrays $F_b[1 \cdots H]$ and $D_b[1 \cdots H]$ size H, i.e., for each $1 \leq i \leq H$,

$F_b[i] = \sum_{p_{ij} \in P_i \cap T^*(b)} f_{ij}$ and $D_b[i] = w_i \cdot \sum_{p_{ij} \in P_i \cap T^*(b)} f_{ij} d(b, p_{ij})$. Note that the size of T_1^* is at most $Hm/2$.

Next we find the centroid b_1 of T_1^*. As in Section 3.3.1, by traversing T_1^* and using the two information arrays associated with the connector b, in $O(H + |T_1^*|)$ time we can determine whether b_1 is x^*, and if not, which split subtree of b_1 contains x^*. If b_1 is x^*, then we are done. Otherwise, we proceed on the split subtree of b_1 that contains x^* recursively until after l steps we obtain a subtree T_l^* that is an edge of T^* with $x^* \in T_l^*$. Since $|T^*| = O(m)$, we have $l = O(\log m)$.

To analyze the total running time of all above recursive steps, suppose for each $1 \leq k \leq l$, we refer to the *k-th step* as for determining the subtree T_k^* and computing the corresponding information (e.g., the information arrays). Note that $|T_k^*| \leq Hm/2^k$ and T_k^* has at most k connectors. As discussed above, the first step takes $O(|T^*|)$ time, and similarly, the second step takes $O(H + |T_1^*|)$ time because there is a connector in T_1^*. In general, the k-th step can be done in $O((k-1)H + |T_{k-1}^*|)$ time. Therefore, if we let $T_0^* = T^*$, the total running time for obtaining T_l^* is $O(\sum_{k=1}^{l}((k-1)H + |T_{k-1}^*|)) = O(H \cdot l^2 + H \cdot m \cdot \sum_{k=1}^{l}(1/2^k))$, which is $O(m)$ since $H = O(1)$ and $l = O(\log m)$.

Finally, to compute x^* in T_l^*, which is an edge of T^*, we can use Lemma 3.2.5 (replacing T with T^*) and find x^* in $O(m)$ time.

The lemma thus follows. □

We conclude that the center x^* on T can be found in $O(|T|) = O(mn)$ time.

Recall that the above only considered the vertex-constrained case, i.e., all locations of $\mathcal{P}$ are at vertices of T and each vertex of T contains at least one location of $\mathcal{P}$. For the general problem where a location of $\mathcal{P}$ may be in the interior of an edge of T and a vertex of T may not contain any location of $\mathcal{P}$, the following theorem solves it in $O(mn + |T|)$ time by reducing it to the vertex-constrained case.

Theorem 3.3.8. *The center x^* of T and $\mathcal{P}$ can be found in $O(mn + |T|)$ time.*

Proof. Recall that $\mathcal{P}$ consists of n uncertain points, each of which has m locations on T. We reduce the problem to a problem instance of the vertex-constrained case and then apply our algorithm for the vertex-constrained case. More specifically, we will modify the tree T to obtain another tree T' of size $O(mn)$. We will also compute another set

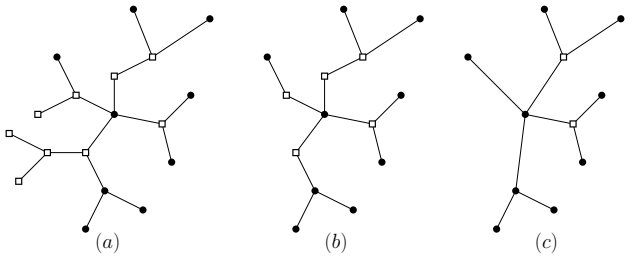

Figure 3.6. Illustrating the three trees: (a) T_1, (b) T_2, and (c) T'. Note that the empty and non-empty vertices are shown with squares and disks, respectively.

$\mathcal{P}'$ of n uncertain points on T', which correspond to the uncertain points of $\mathcal{P}$ with the same weights, but each uncertain point of $\mathcal{P}'$ has at most $2m$ locations on T'. Further, each location of $\mathcal{P}'$ is at a vertex of T' and each vertex of T' holds at least one location of $\mathcal{P}'$. We will show that we can obtain T' and $\mathcal{P}'$ in $O(mn + |T|)$ time based on T and $\mathcal{P}$. Finally, given the center x' of T' and $\mathcal{P}'$, we can find x^* on T in $O(mn + |T|)$ time. The details are given below.

Recall that for each edge e of T, all locations of $\mathcal{P}$ on e have been sorted. We traverse T, and for each edge e, if e contains some locations of $\mathcal{P}$ in its interior, then we create a new vertex in T for each such location. In this way, we create at most mn new vertices for T. The above can be done in $O(mn + |T|)$ time. We use T_1 to denote the new tree. Note that $|T_1| = O(mn + |T|)$. For each vertex v of T_1, if v does not hold any location of $\mathcal{P}$, then we call v an *empty* vertex.

Next, we modify T_1 in the following way. First, for each leaf v of T_1, if v is empty, then we remove v from T_1. We keep doing this until every leaf of the remaining tree is not empty. Let T_2 denote the tree after the above step (e.g., see Fig. 3.6). Second, for each internal vertex v of T_2, if the degree of v is 2 and v is empty, then we remove v from T_2 and merge its two incident edges as a single edge whose length is equal to the sum of the lengths of the two incident edges of v. We keep doing this until every degree-2 vertex of the remaining tree is not empty. Let T' be the remaining tree (e.g., see Fig. 3.6).

The above two steps can be implemented in $O(|T_1|)$ time as follows. We pick an arbitrary non-empty vertex of T_1 as the root. The first step can be done by a post-order traversal of T_1 from the root. Indeed, during the traversal, suppose a vertex v is

currently being visited; if all vertices in its subtree are empty, then v should be removed from T_1. In this way, we can obtain T_2 in $O(|T_1|)$ time. The second step can be done by traversing T_2 and checking every degree-2 vertices.

We have the following observation on T': Every location of $\mathcal{P}$ is at a vertex of T' and every vertex of T' except those whose degrees are larger than two holds a location of $\mathcal{P}$. Let V denote the set of all vertices of T' and let V_3 denote the set of the vertices of T' whose degrees are at least three. Clearly, $|V_3| \leq |V \setminus V_3|$. Since each vertex in $V \setminus V_3$ holds a location of $\mathcal{P}$, we have $|V \setminus V_3| \leq mn$, and thus $|V_3| \leq mn$.

To make sure that every vertex of T' contains a location of an uncertain point, we arbitrarily pick m vertices from V_3 and remove them from V_3, and then set a "dummy" location for P_1 at each of these vertices with zero probability. We keep picking another m vertices from V_3 for P_2 and continue this procedure until V_3 becomes empty. Note that since $|V_3| \leq mn$, the above procedure will eventually make V_3 empty before we "use up" all n uncertain points of $\mathcal{P}$. We let $\mathcal{P}'$ be the set of new uncertain points, each of which has at most $2m$ locations.

Clearly, now every vertex of T' holds a location of $\mathcal{P}'$ and every location of $\mathcal{P}'$ is at a vertex of T', and therefore, we have obtained an instance of the vertex-constrained case on T' and $\mathcal{P}'$. Hence, we can use our algorithm for the vertex-constrained case to compute the center x' of T' in $O(mn)$ time. Finally, we determine the center x^* for our original problem on T and $\mathcal{P}$ in the following way.

Observe that every vertex v of T' also exists as a vertex in T_1, and every edge (u, v) of T' corresponds to the simple path in T_1 between u and v. Suppose x' is on an edge (u, v) of T' and let δ be the distance from u to x'. Then, we can locate the point x_1 in T_1 in the simple path from u to v and at distance δ from u. The above way of computing x_1 can be done in $O(|T_1|) = O(mn + |T|)$ time. By our construction from T_1 to T', x_1 is a center of T_1. Further, by our construction from T to T_1, if an edge e of T does not appear in T_1, then e is broken into several edges in T_1 whose total length is equal to that of e. Hence, every point of T corresponds a unique point on T_1. The point of T that corresponds to x' is the center x^* of T and $\mathcal{P}$. Therefore, given x', x^* can be found in $O(mn + |T|)$ time.

As a summary, we can compute the center x^* of T and $\mathcal{P}$ in $O(mn+|T|)$ time. The theorem thus follows. □

CHAPTER 4

THE ONE-CENTER PROBLEM OF UNCERTAIN POINTS ON THE REAL LINE

In this chapter, we consider the one-dimensional one-center problem on uncertain data under the locational model. Our results in this chapter have been published in [68].

4.1 Introduction

Let L be a real line. Without loss of generality, we assume L is the x-axis. Let $\mathcal{P}$ be a set of n uncertain points $\{P_1, P_2, \ldots, P_n\}$ on L, where each uncertain point $P_i \in \mathcal{P}$ is specified by its pdf f_i: $\mathbb{R} \to \mathbb{R}^+ \cup \{0\}$, which is a piecewise-uniform function (i.e., a histogram), consisting of at most $m+1$ pieces (e.g., see Fig. 2.1 in Chapter 2). More specifically, for each P_i, there are m sorted x-coordinates $x_{i1} < x_{i2} < \ldots < x_{im}$ and $m-1$ nonnegative values $y_{i1}, y_{i2}, \ldots, y_{i,m-1}$ such that $f_i(x) = y_{ij}$ for $x_{ij} \leq x < x_{i,j+1}$ with $1 \leq j \leq m-1$. For convenience of discussion, we assume $x_{i0} = -\infty$, $x_{i,m+1} = +\infty$, $y_{i0} = y_{im} = 0$, and $f_i(x) = 0$ for $x \in (-\infty, x_{i1}) \cup [x_{im}, +\infty)$.

As discussed in [7], such a histogram function f_i can be used to approximate any pdf with arbitrary precision. In particular, for the *discrete* case where each uncertain point has a finite number of discrete locations, each with a probability, it can also be incorporated by our histogram model using infinitesimal pieces at these locations. Thus, the discrete case is a special case of our histogram model.

With a little abuse of notation, for any (deterministic) point q on L, we also use q to denote the x-coordinate of q. For any uncertain point $P_i \in \mathcal{P}$, the *expected distance* from q to P_i is

$$\mathsf{Ed}(q, P_i) = \int_{-\infty}^{+\infty} f_i(x)|x - q|dx.$$

The goal of our *one-center* problem on $\mathcal{P}$ is to find a (deterministic) point c^* on L such that the maximum expected distance from c^* to all uncertain points of $\mathcal{P}$ is minimized, and c^* is called a *center* of $\mathcal{P}$.

The algorithm proposed in Chapter 2 can solve the problem in $O(mn \log mn + n \log n \log mn)$ time. In this paper, we present an $O(mn)$ time algorithm. Since the input size of the problem is $\Theta(mn)$, our algorithm runs in linear time, which is optimal.

We should point out that our algorithm is applicable to the *weighted case* of this problem where each uncertain point $P_i \in \mathcal{P}$ has a nonnegative multiplicative weight w_i and the weighted expected distance is considered (i.e., $\mathsf{Ed}(q, P_i) = w_i \cdot \int_{-\infty}^{+\infty} f_i(x)|x - q|dx$). To solve the weighted case, we can first reduce it to the above unweighted case by changing each value y_{ij} to $w_i \cdot y_{ij}$ for every $1 \leq i \leq n$ and $1 \leq j \leq m-1$, and then apply our algorithm for the unweighted case. The running time is still linear. Hence, we will focus our discussion on the unweighted case.

4.1.1 Related Work

In Chapter 2, the problem of finding k centers for a set $\mathcal{P}$ of uncertain points on L was studied, and an algorithm of $O(mn \log mn + n \log k \log n \log mn)$ time was proposed. Therefore, when $k = 1$, the algorithm runs in $O(mn \log mn + n \log n \log mn)$ time as mentioned above. In addition, the discrete case of the above k-center problem (where each uncertain point P_i has m discrete locations, each with a probability) was solved in a faster way in $O(mn \log mn + n \log k \log mn)$ time, and the discrete one-center problem was solved in $O(mn)$ time. Therefore, our new result in this chapter for the one-center problem under the more general histogram model matches the previous result for the discrete case. We also studied the discrete one-center problem for uncertain points on tree networks and proposed a linear-time algorithm in Chapter 3.

The deterministic k-center problems are classical facility location problems have been extensively studied. The problem is NP-hard in the plane [19]. Efficient algorithms were known for special cases, e.g., finding the smallest enclosing circle (i.e., the case $k = 1$) [27], k-center on trees [29–31]. The deterministic k-center problem in the one-dimensional space is solvable in $O(n \log n)$ time [31, 43, 69]. As shown in Chapter 2, the deterministic one-center problem in the one-dimensional space can be solved in linear time.

The k-center problems on uncertain data in high-dimensional spaces have also been considered. For example, approximation algorithms were given in [34] for different prob-

lem models, e.g., the assigned model that is somewhat similar to our problem model and the unassigned model which was relatively easy because it can be reduced to the corresponding deterministic problem [34]. Foul [61] studied the problem of finding the center in the plane to minimize the maximum expected distance from the center to all uncertain points, where each uncertain point has a uniform distribution in a given rectangle. Other facility location problems on uncertain data under various models, e.g., the minmax regret [37, 38, 65], have also been studied (see [40] for a survey).

4.1.2 Our Techniques

To solve our one-center problem, based on observations, we reduce it to the following geometric problem. Let $\mathcal{H}$ be a set of n unimodal functions in the plane (i.e., when x changes from $-\infty$ to $+\infty$, it first monotonically decreases and then increases) and each function consists of m pieces with each piece being a parabolic arc. We wish to find the lowest point v^* in the upper envelope of the functions of $\mathcal{H}$. In the discrete version of the one-center problem, as shown in Chapter 2, each parabolic arc of every function of $\mathcal{H}$ is simply a line segment, and thus, v^* can be found by applying Megiddo's linear time linear programming algorithm [27]. In our problem, however, since the parabolic arcs of the functions of $\mathcal{H}$ may not be line segments, Megiddo's algorithm in [27] does not work any more. We present a new prune-and-search technique that can compute v^* in $O(mn)$ time. This result immediately leads to a linear time algorithm for our one-center problem on $\mathcal{P}$.

Comparing with the linear programming problem, the above geometric problem is more general (i.e., the linear programming problem is a special case of our problem). Our linear time algorithm for the problem may be interesting in its own right and may find other applications as well. In fact, our result can be extended to more general unimodal functions.

4.2 Preliminaries

A function $g : \mathbb{R} \rightarrow \mathbb{R}$ is a *unimodal* if there exists a value x' such that for any $x_1 < x_2$, $g(x_1) \geq g(x_2)$ holds if $x_2 \leq x'$ and $g(x_1) \leq g(x_2)$ holds if $x' \leq x_1$, i.e., $g(x)$ is monotonically decreasing on $x \in (-\infty, x']$ and increasing on $x \in [x', +\infty)$.

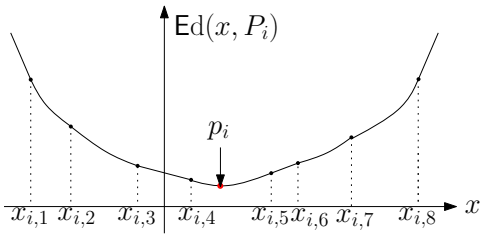

Figure 4.1. Illustrating the expected distance function $\mathsf{Ed}(x, P_i)$ for an uncertain point P_i with $m = 8$. $\mathsf{Ed}(x, P_i)$ is monotonically decreasing for $x \in (-\infty, p_i]$, and increasing for $x \in [p_i, -\infty)$.

Recall that L is the x-axis, and for any point q on L, we also use q to denote the x-coordinate of q. Therefore, the values of $\mathbb{R}$ correspond to the points of L. In the following, we will use "the values of $\mathbb{R}$" and "the points of L" interchangeably.

Consider any uncertain point P_i of $\mathcal{P}$. If we consider the expected distance $\mathsf{Ed}(x, P_i)$ as a function of the points x on L (or $x \in \mathbb{R}$), the following observation has been proved in Chapter 2. Note that the m coordinates $x_{i1}, \ldots, x_{im}$ of P_i are already given sorted.

Lemma 4.2.1. *The function* $\mathsf{Ed}(x, P_i)$ *for* $x \in \mathbb{R}$ *is unimodal. More specifically, there exists a point* $p_i \in L$ *such that* $\mathsf{Ed}(x, P_i)$ *is monotonically decreasing on* $x \in (-\infty, p_i]$ *and increasing on* $x \in [p_i, +\infty)$ *(e.g., see Fig. 4.1). In addition,* $\mathsf{Ed}(x, P_i)$ *is a parabolic arc (of constant complexity) on the interval* $[x_{ik}, x_{i,k+1})$ *for each* $0 \leq k \leq m$*, and can be explicitly computed in* $O(m)$ *time.*

The point p_i in Lemma 4.2.1 is referred to as a *centroid* of P_i and can be easily computed in $O(m)$ time after $\mathsf{Ed}(x, P_i)$ is explicitly computed in $O(m)$ time, proved in Chapter 2. In fact, a point $q \in L$ is a centroid of P_i if and only if $\int_{-\infty}^{q} f_i(x) = 0.5$ and $\int_{q}^{+\infty} f_i(x) = 0.5$. Note that the centroid of P_i may not be unique. This case happens when there exists an interval on the x-axis such that for any point q in this interval both $\int_{-\infty}^{q} f_i(x) = 0.5$ and $\int_{q}^{+\infty} f_i(x) = 0.5$ hold, and thus $\mathsf{Ed}(x, P_i)$ is a constant when x is in this interval and any point q in this interval is a centroid. If the centroid of P_i is not unique, then we use p_i to refer to an arbitrary such centroid. For a similar reason, the center of $\mathcal{P}$ may also not be unique, in which case our algorithm will find one such center.

The following corollary can be easily obtained based on Lemma 4.2.1 and binary search on the sorted list of $x_{i1}, x_{i2}, \ldots, x_{im}$. Refer to Chapter 2 for the proof.

Corollary 4.2.2. *For each uncertain point P_i, with $O(m)$ time preprocessing, the value $\mathsf{Ed}(x', P_i)$ for any query value x' can be computed in $O(\log m)$ time.*

Let $\mathcal{H}$ denote the set of all functions $\mathsf{Ed}(x, P_i)$ for $i = 1, 2, \ldots, n$. Since the center c^* of $\mathcal{P}$ is a point on L that minimizes the value $\max_{i=1}^{n} \mathsf{Ed}(x, P_i)$, i.e., $c^* = \text{argmin}_{x \in L} \max_{i=1}^{n} \mathsf{Ed}(x, P_i)$, c^* is equal to the x-coordinate of the lowest point v^* in the upper envelope of $\mathcal{H}$. Therefore, to compute c^*, it is sufficient to find v^*.

As shown in Chapter 2 and mentioned in Section 4.1.2, in the discrete case where each uncertain point P_i has m discrete locations (each with a probability), each function $\mathsf{Ed}(x, P_i)$ consists of $m + 1$ pieces with each piece being a special parabolic arc: a line segment. Consequently, to compute v^* is equivalent to computing the lowest point in the upper envelope of the $n(m + 1)$ extending lines of all line segments of all functions of $\mathcal{H}$, which can be done in $O(mn)$ time by Megiddo's linear time linear programming algorithm [27].

In our problem, since each piece of every function of $\mathcal{H}$ may be a general parabolic arc, Megiddo's algorithm [27], which heavily relies on the properties of lines, does not work any more. In the next section, we present a new prune-and-search algorithm that can compute v^* in $O(mn)$ time, and it can be considered as an extension of Megiddo's algorithm [27].

4.3 Compute the Lowest Point v^* in the Upper Envelope of $\mathcal{H}$

In this section, we present an algorithm that can compute v^* in $O(mn)$ time. Note that the lowest point of the upper envelope of $\mathcal{H}$ may not be unique, in which case we use v^* to represent an arbitrary such lowest point. We assume the preprocessing in Corollary 4.2.2 has been done for every uncertain point $P_i \in \mathcal{P}$, which takes $O(mn)$ time in total. Denote by x^* the x-coordinate of v^*. We first present an $O(n \log m)$ time algorithm for solving the following *decision problem*: given any value x', determine whether $x' < x^*$, $x' = x^*$, or $x' > x^*$. Let $\mathcal{U}$ denote the upper envelope of $\mathcal{H}$.

4.3.1 A Decision Algorithm

Consider any value x'. To solve the decision problem, the main idea is to compute the intersection of $\mathcal{U}$ and the vertical line $x = x'$, and then determine whether $x' < x^*$, $x' = x^*$, or $x' > x^*$ based on the local information of $\mathcal{U}$ at the above intersection. The details are given below. Denote by $l(x')$ the vertical line $x = x'$.

For any uncertain point $P_i \in \mathcal{P}$, by Corollary 4.2.2, we can compute the intersection of the function $\mathsf{Ed}(x, P_i)$ and $l(x')$ in $O(\log m)$ time, and let q_i denote the above intersection. In fact, the parabolic arc of $\mathsf{Ed}(x, P_i)$ that contains q_i can also be determined. We let S_i denote the above parabolic arc. In the case that q_i is the common endpoint of two parabolic arcs, we let S_i denote the set of both parabolic arcs. Hence $|S_i| \leq 2$ holds in either case. In summary, q_i and S_i can be determined in $O(\log m)$ time.

We compute q_i and S_i for every uncertain point $P_i \in \mathcal{P}$, which takes $O(n \log m)$ time in total with the preprocessing in Corollary 4.2.2. Let $Q = \{q_i \mid 1 \leq i \leq n\}$. Next, we find the highest point $q(x')$ of Q, and clearly, $q(x')$ is the intersection of $l(x')$ and the upper envelope $\mathcal{U}$. In case $q(x')$ is equal to multiple points of Q, we let $I(x')$ denote the index set such that for each $i \in I(x')$, $q_i = q(x')$. Let $S(x') = \cup_{i \in I(x')} S_i$. Note that after q_i and S_i for every $P_i \in \mathcal{P}$ are computed, we can obtain $q(x')$, $I(x')$ and $S(x')$ in $O(n)$ time.

Based on $q(x')$ and the set $S(x')$ of parabolic arcs , it is not difficult to see that we can solve the decision problem in the following way. If all parabolic arcs of $S(x')$ are strictly decreasing at $q(x')$, then $x' < x^*$. If all parabolic arcs of $S(x')$ are strictly increasing at $q(x')$, then $x' > x^*$. Otherwise, $x' = x^*$. Clearly, with $q(x')$ and $S(x')$, the above can be determined in $O(n)$ time.

Therefore, we obtain the following result.

Lemma 4.3.1. *With $O(mn)$ time preprocessing, given any value x', we can determine whether $x' < x^*$, $x' = x^*$, or $x' > x^*$, in $O(n \log m)$ time.*

4.3.2 Observations

We first give the following lemma, which will help us to prune uncertain points of $\mathcal{P}$ in our prune-and-search algorithm.

Lemma 4.3.2. *For any two uncertain points P_i and P_j, the upper envelope of $\mathsf{Ed}(x, P_i)$ and $\mathsf{Ed}(x, P_j)$ has $O(m)$ complexity and can be computed in $O(m)$ time.*

Proof. Consider two uncertain points P_i and P_j. We first analyze the intersections of the functions $\mathsf{Ed}(x, P_i)$ and $\mathsf{Ed}(x, P_j)$. Recall that each of these two functions has $m+1$ parabolic arcs.

We use $\mathsf{Ed}_L(x, P_i)$ to denote the part of $\mathsf{Ed}(x, P_i)$ to the left of the centroid p_i (i.e., $x \in (-\infty, p_i]$), and use $\mathsf{Ed}_R(x, P_i)$ to denote the part of $\mathsf{Ed}(x, P_i)$ to the right of p_i (i.e., $x \in [p_i, +\infty)$). Hence, $\mathsf{Ed}_L(x, P_i)$ is monotonically decreasing and $\mathsf{Ed}_R(x, P_i)$ is monotonically increasing. We define $\mathsf{Ed}_L(x, P_j)$ and $\mathsf{Ed}_R(x, P_j)$ for P_j similarly.

Since $\mathsf{Ed}_L(x, P_i)$ is monotonically increasing and $\mathsf{Ed}_R(x, P_j)$ is monotonically decreasing, $\mathsf{Ed}_L(x, P_i)$ can intersect $\mathsf{Ed}_R(x, P_j)$ transversally at most once. Similarly, $\mathsf{Ed}_R(x, P_i)$ can intersect $\mathsf{Ed}_L(x, P_j)$ transversally at most once.

Consider $\mathsf{Ed}_L(x, P_i)$ and $\mathsf{Ed}_L(x, P_j)$. Since both of them are monotonically decreasing, a parabolic arc of $\mathsf{Ed}_L(x, P_i)$ can intersect any parabolic arc of $\mathsf{Ed}_L(x, P_j)$ transversally at most twice (this is because both parabolic arcs are decreasing). Therefore, $\mathsf{Ed}_L(x, P_i)$ can intersect $\mathsf{Ed}_L(x, P_j)$ transversally $O(m)$ times. Similarly, $\mathsf{Ed}_R(x, P_i)$ can intersect $\mathsf{Ed}_R(x, P_j)$ transversally $O(m)$ times.

The above discussion leads to the conclusion that the two functions $\mathsf{Ed}(x, P_i)$ and $\mathsf{Ed}(x, P_j)$ can intersect each other transversally at most $O(m)$ times. This implies that the upper envelope of $\mathsf{Ed}(x, P_i)$ and $\mathsf{Ed}(x, P_j)$ is of complexity $O(m)$. Since both $\mathsf{Ed}(x, P_i)$ and $\mathsf{Ed}(x, P_j)$ can be computed in $O(m)$ time by Lemma 4.2.1, their upper envelope can be computed by a simple sweeping algorithm in $O(m)$ time. The lemma thus follows. □

Lemma 4.3.2 implies the following corollary.

Corollary 4.3.3. *For any two uncertain points P_i and P_j, in $O(m)$ time we can compute a sorted list of values $-\infty = x_0 < x_1 < x_2 < \ldots < x_t < x_{t+1} = +\infty$ with $t = O(m)$ such that either $\mathsf{Ed}(x, P_i) \leq \mathsf{Ed}(x, P_j)$ or $\mathsf{Ed}(x, P_i) > \mathsf{Ed}(x, P_j)$ holds when $x \in [x_k, x_{k+1}]$ for any $0 \leq k \leq t$.*

Recall that x^* is the x-coordinate of v^*. Corollary 4.3.3 implies that if we know x^* is contained in some interval $[x_k, x_{k+1}]$ as defined in Corollary 4.3.3, then at least

one of P_i and P_j can be pruned, and more specifically, if $\mathsf{Ed}(x, P_i) \leq \mathsf{Ed}(x, P_j)$ when $x \in [x_k, x_{k+1}]$, then for computing v^*, P_i can be discarded. In the following, let $X(i, j)$ denote the set of sorted x-coordinates as defined in Corollary 4.3.3. Thus, it holds that $|X(i, j)| \leq Cm$ for some constant C for any two uncertain points P_i and P_j of $\mathcal{P}$.

4.3.3 Computing the Lowest Point v^*

In the sequel, we present our algorithm for computing v^* based on the prune-and-search techniques. Below, we show that our algorithm can prune $n/4$ uncertain points of $\mathcal{P}$ in $O(mn)$ time.

We arbitrarily assign the uncertain points of $\mathcal{P}$ into $n/2$ pairs. Let Σ denote all such pairs. For each pair $(P_i, P_j) \in \Sigma$, we compute $X(i, j)$ and the upper envelope of $\mathsf{Ed}(x, P_i)$ and $\mathsf{Ed}(x, P_j)$ in $O(m)$ time by Lemma 4.3.2. Let $\mathcal{X} = \bigcup_{(P_i,P_j)\in\Sigma} X(i, j)$. Since Σ has $n/2$ pairs, $\mathcal{X}$ can be computed in $O(mn)$ time by Lemma 4.3.2 and $|\mathcal{X}| \leq Cmn/2$.

We find the median x_m of $\mathcal{X}$ in $O(mn)$ time. By using our decision algorithm in Lemma 4.3.1, we can determine whether $x_m < x^*$, $x_m = x^*$, or $x_m > x^*$ in $O(n \log m)$ time. If $x_m = x^*$, then x_m is the center of $\mathcal{P}$ and we are done with the algorithm. Otherwise, if $x_m < x^*$, then we eliminate all values in $\mathcal{X}$ that are smaller than or equal to x_m; if $x_m > x^*$, then we eliminate all values in $\mathcal{X}$ that are larger than or equal to x_m. In either case, no more than $|\mathcal{X}|/2$ values of $\mathcal{X}$ will remain. Next, we continue the above procedure recursively on the remaining values of $\mathcal{X}$ until the number of remaining values is no more than $n/4$. Since initially $|\mathcal{X}| \leq Cmn/2$, the number of recursive steps is bounded by $O(\log(2Cm)) = O(\log m + \log C + 1)$.

We claim that the total running time of the above recursive procedure is $O(mn)$. Indeed, the total time for finding the medians in all recursive steps is bounded by $O(|\mathcal{X}| + |\mathcal{X}|/2 + |\mathcal{X}|/4 + \cdots) = O(|\mathcal{X}|) = O(mn)$. In each recursive step, we need to call the decision algorithm once, and therefore, the total time of the decision algorithm in all recursive steps is $O(n \log^2 m)$. Hence, the above claim is proved.

We should point out that one may want to keep doing the above recursive procedure until X has at most one or two remaining values, which would need $\Theta(\log mn)$ recursive steps; but according to our above time complexity analysis, it would take $O(mn + n \log m \log mn)$ time, which may not necessarily be bounded by $O(mn)$ (e.g., when

$m = o(\log n)$). In fact, performing the above recursive steps until the remaining values of $\mathcal{X}$ is at most $n/4$ is an interesting and crucial part of our techniques.

Let $\mathcal{X}'$ be the set of remaining values of $\mathcal{X}$. Hence, $|\mathcal{X}'| \leq n/4$. Let x' and x'' denote the smallest and largest values of $\mathcal{X}'$, respectively. According to our above way of computing $\mathcal{X}'$, $\mathcal{X}'$ consists of all values of $\mathcal{X}$ in $[x', x'']$, and further, $x^* \in [x', x'']$. Since $|\mathcal{X}'| \leq n/4$, there are at most $n/4$ pairs (P_i, P_j) of Σ such that $X(i,j)$ contains a value in $\mathcal{X}'$. In other words, we have the following observation.

Observation 4.3.4. *There are at least $n/4$ pairs (P_i, P_j) of Σ such that $X(i,j)$ does not contain any value of $\mathcal{X}'$ (i.e., no value of $X(i,j)$ is in the interval $[x', x'']$).*

Let Σ' denote the set of pairs of Σ that satisfy the condition in Observation 4.3.4 (i.e., for each pair (P_i, P_j) of Σ', no value of $X(i,j)$ is in the interval $[x', x'']$). Hence, $|\Sigma'| \geq n/4$. Consider any pair $(P_i, P_j) \in \Sigma'$. Let $X(i,j) = x_1, x_2, \ldots, x_t$ with $x_0 = -\infty$ and $x_{t+1} = +\infty$. By Observation 4.3.4, since no value of $X(i,j)$ is in the interval $[x', x'']$, $[x', x'']$ is contained in $[x_k, x_{k+1}]$ for some k with $0 \leq k \leq t$. Thus, either $\mathsf{Ed}(x, P_i) \leq \mathsf{Ed}(x, P_j)$ or $\mathsf{Ed}(x, P_i) > \mathsf{Ed}(x, P_j)$ holds for $x \in [x', x'']$. Without loss of generality, we assume $\mathsf{Ed}(x, P_i) \leq \mathsf{Ed}(x, P_j)$ holds for $x \in [x', x'']$. Since $x^* \in [x', x'']$, we obtain that $\mathsf{Ed}(x^*, P_i) \leq \mathsf{Ed}(x^*, P_j)$. Therefore, P_i can be discarded. Since $|\Sigma'| \geq n/4$, we can discard at least $n/4$ uncertain points of $\mathcal{P}$.

Recall that $\mathcal{X}'$ has been computed and $|\mathcal{X}'| \leq n/4$. The above pruning procedure can be done in $O(mn)$ time. Indeed, we can find x' and x'' in $\mathcal{X}'$ in $O(n)$ time. To determine the set Σ', suppose for each element $x \in \mathcal{X}$, we have associated the pair (P_i, P_j) with x such that $x \in X(i,j)$; then Σ' can be determined in $O(n)$ time by scanning $\mathcal{X}'$. Next, for each pair (P_i, P_j) of Σ', we determine the interval $[x_k, x_{k+1}]$ of X_{ij} (as defined above) that contains $[x', x'']$, which can be easily done in $O(m)$ since the values of $X(i,j)$ have already been sorted and $|X(i,j)| = O(m)$ by Corollary 4.3.3. Since the upper envelope of P_i and P_j has been computed, we can also determine whether $\mathsf{Ed}(x, P_i) \leq \mathsf{Ed}(x, P_j)$ or $\mathsf{Ed}(x, P_i) > \mathsf{Ed}(x, P_j)$ holds for $x \in [x', x'']$ in $O(m)$ time. Consequently, we can prune one of P_i and P_j. Hence, the pruning procedure can be done in $O(mn)$ time.

The above discussion shows that within $O(mn)$ time, we can prune at least $n/4$ uncertain points of $\mathcal{P}$ such that the center c^* is determined only by the remaining at most $3n/4$ uncertain points of $\mathcal{P}$.

We continue the above procedure recursively on the remaining uncertain points of $\mathcal{P}$ until a constant number of uncertain points of $\mathcal{P}$ remain. The total running time satisfies the recurrence $T(m,n) = T(m,3n/4) + O(mn)$. Solving the recurrence yields $T(m,n) = O(mn)$. Let $\mathcal{P}'$ denote the set of remaining uncertain points of $\mathcal{P}$. We have $|\mathcal{P}'| = O(1)$. Let $\mathcal{H}'$ denote the set of functions $\mathsf{Ed}(x, P_i)$ for all $P_i \in \mathcal{P}'$, and let $\mathcal{U}'$ be the upper envelope of the functions of $\mathcal{H}'$. According to our above pruning algorithm, v^* is also the lowest point of $\mathcal{U}'$. Note that in the case that the lowest point of $\mathcal{U}'$ is not unique, our pruning algorithm above makes sure that every lowest point of $\mathcal{U}'$ is a lowest point of $\mathcal{U}$ (we omit the detailed discussions).

Finally, we find the lowest point of $\mathcal{U}'$ by constructing it explicitly, which can be done in $O(m)$ time by a sweeping algorithm, as follows. We sweep a vertical line l from left to right. During the sweeping, we maintain the intersections of l with all functions of $\mathcal{H}'$ in the order sorted by their y-coordinates (note that the number of such intersections is $O(1)$ since $|\mathcal{H}'| = O(1)$). An event happens when l hits a vertex of some function or an intersection of two functions. Clearly, all functions of $\mathcal{H}'$ have $O(m)$ vertices since $|\mathcal{H}'| = O(1)$. As discussed in the proof of Lemma 4.3.2, every two functions of $\mathcal{H}'$ can intersect (transversally) $O(m)$ times. Therefore, the number of (transversal) intersections among all functions of $\mathcal{H}'$ is $O(m)$ since $|\mathcal{H}'| = O(1)$. Hence, the total number of events is $O(m)$. Processing each event can be done in constant time since $|\mathcal{H}'| = O(1)$. Therefore, we can explicitly construct $\mathcal{U}'$ and thus find its lowest point in $O(m)$ time.

Combining all above efforts, the lowest vertex v^* of $\mathcal{H}$ can be computed in $O(mn)$ time. Thus, the center of $\mathcal{P}$ can be computed in $O(mn)$ time.

Theorem 4.3.5. *The center of $\mathcal{P}$ can be computed in $O(mn)$ time.*

CHAPTER 5

THE RECTILINEAR CENTER OF UNCERTAIN POINTS IN THE PLANE

In this chapter, we study the one-center problem on uncertain points in the plane with respect to the rectilinear distance. The results in this chapter have been submitted to *International Journal of Computational Geometry and Applications*.

5.1 Introduction

Let $\mathcal{P} = \{P_1, P_2, \ldots, P_n\}$ be a set of n uncertain points in the plane, where each uncertain point $P_i \in \mathcal{P}$ has m possible locations $p_{i1}, p_{i2}, \cdots, p_{im}$ and for each $1 \leq j \leq m$, p_{ij} is associated with a probability $f_{ij} \geq 0$ for P_i being at p_{ij} (which is independent of other locations). Technically, we should assume $\sum_{j=1}^{m} f_{ij} \leq 1$ for each P_i. However, our algorithm also works when the assumption does not hold, in which case we may simply consider f_{ij} as some kind of weight for the location p_{ij}.

For any (deterministic) point p in the plane, we use x_p and y_p to denote the x- and y-coordinates of p, respectively. For any two points p and q, we use $d(p, q)$ to denote the *rectilinear distance* between p and q, i.e., $d(p, q) = |x_p - x_q| + |y_p - y_q|$.

Consider a point q in the plane. For any uncertain point $P_i \in \mathcal{P}$, the *expected rectilinear distance* between q and P_i is defined as

$$\mathsf{Ed}(P_i, q) = \sum_{j=1}^{m} f_{ij} \cdot d(p_{ij}, q).$$

Let $\mathsf{Ed}_{\max}(q) = \max_{P_i \in \mathcal{P}} \mathsf{Ed}(P_i, q)$. A point q^* is called a *rectilinear center* of $\mathcal{P}$ if it minimizes the value $\mathsf{Ed}_{\max}(q^*)$ among all points in the plane. Our goal is to compute q^*. Note that such a point q^* may not be unique, in which case we let q^* denote an arbitrary such point.

We assume that for each uncertain point P_i of $\mathcal{P}$, its m locations are given in two sorted lists, one by x-coordinates and the other by y-coordinates. To the best of our knowledge, this problem has not been studied before. In this chapter, we present an

$O(mn)$ time algorithm. Since the input size of the problem is $\Theta(nm)$, our algorithm essentially runs in linear time, which is optimal.

Further, our algorithm is applicable to the weighted version of this problem in which each $P_i \in \mathcal{P}$ has a weight $w_i \geq 0$ and the *weighted expected distance*, i.e., $w_i \cdot \mathsf{Ed}(P_i, q)$, is considered. To solve the weighted version, we can first reduce it to the unweighted version by changing each f_{ij} to $w_i \cdot f_{ij}$ for all $1 \leq i \leq n$ and $1 \leq j \leq m$, and then apply our algorithm for the unweighted version. The running time is still $O(mn)$.

5.1.1 Related Work

The problem of finding one-center among uncertain points on a line has been considered in Chapter 2 where an $O(nm)$ time algorithm was given. An algorithm for computing k centers for general k was also given in Chapter 2 with the running time $O(mn \log mn + n \log n \log k)$. In fact, in Chapter 2 we considered the k-center problem under a more general uncertain model where each uncertain point can appear in m intervals, and in Chapter 4 we solve the one-center problem on a line in linear time under the more general uncertain model. We also studied the one-center problem for uncertain points on a tree in Chapter 3, where a linear-time algorithm was proposed.

There is also a lot of other work on facility location problems for uncertain data. For instance, Cormode and McGregor [34] proved that the k-center problem on uncertain points each associated with multiple locations in high-dimension space is NP-hard and gave approximation algorithms for different problem models. Foul [61] considered the Euclidean one-center problem on uncertain points each of which has a uniform distribution in a given rectangle in the plane. de Berg et al. [64] studied the Euclidean 2-center problem for a set of moving points in the plane (the moving points can be considered uncertain).

The k-center problems on deterministic points are classical problems and have been studied extensively. When all points are in the plane, the problems on most distance metrics are NP-hard [19]. However, some special cases can be solved in polynomial time, e.g., the one-center problem [27], the two-center problem [70], the rectilinear three-center problem [71], the line-constrained k-center problems (where all centers are restricted to be on a given line in the plane) [58–60].

5.1.2 Our Techniques

Consider any uncertain point $P_i \in \mathcal{P}$ and any (deterministic) point q in the plane $\mathbb{R}^2$. We first show that $\mathsf{Ed}(P_i, q)$ is a convex piecewise linear function with respect to $q \in \mathbb{R}^2$. More specifically, if we draw a horizontal line and a vertical line from each location of P_i, these lines partition the plane into a grid G_i of $(m+1) \times (m+1)$ cells. Then, $\mathsf{Ed}(P_i, q)$ is a linear function (in both the x- and y-coordinates of q) in each cell of G_i. In other words, $\mathsf{Ed}(P_i, q)$ defines a plane surface patch in 3D on each cell of G_i. Then, finding $q^* \in \mathbb{R}^2$ is equivalent to finding a lowest point p^* in the upper envelope of the n graphs in 3D defined by $\mathsf{Ed}(P_i, q)$ for all $P_i \in \mathcal{P}$ (specifically, q^* is the projection of p^* onto the xy-plane).

The problem of finding p^*, which may be interesting in its own right, can be solved in $O(nm^2)$ time by the linear-time algorithm for the 3D linear programming (LP) problem [27]. Indeed, for a plane surface patch, we call the plane containing it the *supporting plane*. Let $\mathcal{H}$ be the set of the supporting planes of the surface patches of the functions $\mathsf{Ed}(P_i, q)$ for all $P_i \in \mathcal{P}$. Since each function $\mathsf{Ed}(P_i, q)$ is convex, p^* is also a lowest point in the upper envelope of the planes of $\mathcal{H}$. Thus, finding p^* is an LP problem in $\mathbb{R}^3$ and can be solved in $O(|\mathcal{H}|)$ time [27]. Note that $|\mathcal{H}| = \Theta(nm^2)$ since each grid G_i has $(m+1)^2$ cells.

We give an $O(mn)$ time algorithm without computing the functions $\mathsf{Ed}(P_i, q)$ explicitly. We use a prune-and-search technique that can be considered as an extension of Megiddo's technique for the 3D LP problem [27]. In each recursive step, we prune at least $n/32$ uncertain points from $\mathcal{P}$ in linear time. In this way, q^* can be found after $O(\log n)$ recursive steps.

Unlike Megiddo's algorithm [27], each recursive step of our algorithm itself is a recursive algorithm of $O(\log m)$ recursive steps. Therefore, our algorithm has $O(\log n)$ "outer" recursive steps and each outer recursive step has $O(\log m)$ "inner" recursive steps. In each outer recursive step, we maintain a rectangle R that always contains q^* in the xy-plane. Initially, R is the entire plane. Each inner recursive step shrinks R with the help of a *decision algorithm*. The key idea is that after $O(\log m)$ steps, R is so small that there is a set $\mathcal{P}^*$ of at least $n/2$ uncertain points such that R is contained inside a single cell of the grid G_i of each uncertain point P_i of $\mathcal{P}^*$ (i.e., R does not intersect the

extension lines from the locations of P_i). At this point, with the help of our decision algorithm, we can use a pruning procedure similar to Megiddo's algorithm [27] to prune at least $|\mathcal{P}^*|/16 \geq n/32$ uncertain points of $\mathcal{P}^*$. Each outer recursive step is carefully implemented so that it takes only linear time.

In particular, our decision algorithm is for the following *decision problem*. Let R be a rectangle in the plane and R contains q^* (but the exact location of q^* is unknown). Given an arbitrary line L that intersects R, the decision problem is to determine which side of L contains q^*. Megiddo's technique [27] gave an algorithm that can solve our decision problem in $O(m^2n)$ time. We give a decision algorithm of $O(mn)$ time. In fact, in order to achieve the overall $O(mn)$ time for computing q^*, our decision algorithm has the following performance. For each $1 \leq i \leq n$, let a_i and b_i be the number of columns and rows of the grid G_i intersecting R, respectively. Our decision algorithm runs in $O(\sum_{i=1}^{n}(a_i + b_i))$ time.

The rest of this chapter is organized as follows. In Section 5.2, we introduce some observations. In Section 5.3, we present our decision algorithm. Section 5.4 gives the overall algorithm for computing the rectilinear center q^*.

5.2 Observations

Let p be a point in the plane $\mathbb{R}^2$. The vertical line and the horizontal line through p partition the plane into four (unbounded) rectangles. Consider another point $q \in \mathbb{R}^2$. We consider $d(p, q)$ as a function of $q \in \mathbb{R}^2$. For each of the above rectangles R, $d(p, q)$ on $q \in R$ is a linear function in both the x- and y-coordinates of q, and thus $d(p, q)$ on $q \in R$ defines a plane surface patch in $\mathbb{R}^3$. Further, $d(p, q)$ on $q \in \mathbb{R}^2$ is a convex piecewise linear function.

For ease of exposition, we make a general position assumption that no two locations of the uncertain points of $\mathcal{P}$ have the same x- or y-coordinate.

Consider an uncertain point P_i of $\mathcal{P}$. We extend a horizontal line and a vertical line through each location of P_i to obtain a grid, denoted by G_i, which has $(m+1) \times (m+1)$ cells (and each cell is a rectangle). According to the above discussion, for each location p_{ij} of $\mathcal{P}$, the function $d(p_{ij}, q)$ of q in each cell of G_i is linear and defines a plane surface patch in $\mathbb{R}^3$. Therefore, if we consider $\mathsf{Ed}(P_i, q)$ as a function of q, since $\mathsf{Ed}(P_i, q)$ is the

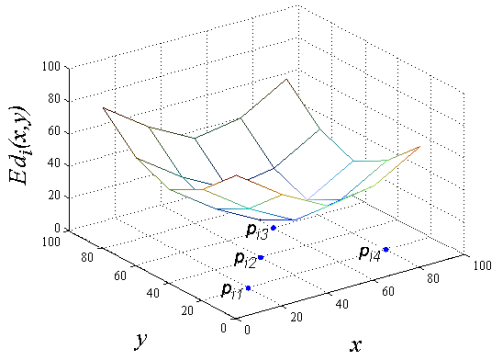

Figure 5.1. Illustrating the function $\mathsf{Ed}_i(x, y)$ of an uncertain point P_i with $m = 4$.

sum of $f_{ij} \cdot d(p_{ij}, q)$ for all $1 \leq j \leq m$, $\mathsf{Ed}(P_i, q)$ of q in each cell of G_i is also linear and defines a plane surface patch in $\mathbb{R}^3$. Further, since each $d(p_{ij}, q)$ for $q \in \mathbb{R}^2$ is convex, the function $\mathsf{Ed}(P_i, q)$, as the sum of convex functions, is also convex.

In the following, since $\mathsf{Ed}(P_i, q)$ is normally considered as function of q, for convenience, we will use $\mathsf{Ed}_i(x, y)$ to denote it for $q = (x, y) \in \mathbb{R}^2$.

The above discussion leads to the following observation.

Observation 5.2.1. *For each uncertain point $P_i \in \mathcal{P}$, the function $\mathsf{Ed}_i(x, y)$ is convex piecewise linear. More specifically, $\mathsf{Ed}_i(x, y)$ on each cell of the grid G_i is linear and defines a plane surface patch in $\mathbb{R}^3$ (e.g., see Fig. 5.1).*

Consider the function $\mathsf{Ed}_i(x, y)$ of any $P_i \in \mathcal{P}$. Clearly, the size of $\mathsf{Ed}_i(x, y)$ is $\Theta(m^2)$. However, since $\mathsf{Ed}_i(x, y)$ on each cell C of G_i is a plane surface patch in $\mathbb{R}^3$, $\mathsf{Ed}_i(x, y)$ on C is of constant complexity. We use $\mathsf{Ed}_i(x, y, C)$ to denote the linear function of $\mathsf{Ed}_i(x, y)$ on C. Note that $\mathsf{Ed}_i(x, y, C)$ is also the function of the supporting plane of the surface patch of $\mathsf{Ed}_i(x, y)$ on C.

As discussed in Section 5.1.2, our algorithm will not compute the function $\mathsf{Ed}_i(x, y)$ explicitly. Instead, we will compute it implicitly. More specifically, we will do some preprocessing such that given any cell C of G_i, the function $\mathsf{Ed}_i(x, y, C)$ can be determined efficiently. We first introduce some notation.

Let $X_i = \{x_{i1}, x_{i2}, \cdots, x_{im}\}$ be the set of the x-coordinates of all locations of P_i sorted in ascending order. Let $Y_i = \{y_{i1}, y_{i2}, \cdots, y_{im}\}$ be the set of their y-coordinates in ascending order. Note that X_i and Y_i can be obtained in $O(m)$ time from the input

(recall that the locations of P_i are given in two sorted lists in the input). For convenience of discussion, we let $x_{i0} = -\infty$, and let X_i also include x_{i0}. Similarly, let $y_{i0} = -\infty$, and let Y_i also include y_{i0}. Note that due to our general position assumption, the values in X_i (resp., Y_i) are distinct.

For any value z, we refer to the largest value in X_i that is smaller than or equal to z the *predecessor* of z in X_i, and we use $I_z(X_i)$ to denote the index of the predecessor. Similarly, $I_z(Y_i)$ is the index of the predecessor of z in Y_i.

Consider any point q in the plane. The predecessor of the x-coordinate of q in X_i is also called the *predecessor of* q in X_i. Similarly, the predecessor of the y-coordinate of q in Y_i is also called the *predecessor of* q in Y_i. We use $I_q(X_i)$ and $I_q(Y_i)$ to denote their indices, respectively.

Consider any cell C of the grid G_i. For convenience of discussion, we assume C contains its left and bottom sides, but does not contain its top and right sides. In this way, any point in the plane is contained in one and only one cell of G_i. Further, all points of C have the same predecessor in X_i and also have the same predecessor in Y_i. This allows us to define the *predecessor of* C in X_i as the predecessor of any point in X_i, and we use $I_C(X_i)$ to denote the index of the predecessor. We define $I_C(Y_i)$ similarly. We have the following lemma.

Lemma 5.2.2. *For any uncertain point $P_i \in \mathcal{P}$, after $O(m)$ time preprocessing, for any cell C of the grid G_i, if $I_C(X_i)$ and $I_C(Y_i)$ are known, then the function $\mathsf{Ed}_i(x, y, C)$ can be computed in constant time.*

Proof. For each location $p \in P_i$, let x_p and y_q be the x- and y-coordinates of p, respectively, and let f_p be the probability associated with p.

For any point $q = (x, y)$ in $\mathbb{R}^2$, recall that the expected distance function $\mathsf{Ed}_i(x, y) = \sum_{p \in P_i} f_p \cdot d(p, q) = \sum_{p \in P_i} f_p \cdot (|x_p - x| + |y_p - y|)$. Therefore, we can write $\mathsf{Ed}_i(x, y) = \sum_{p \in P_i} f_p \cdot |x_p - x| + \sum_{p \in P_i} f_p \cdot |y_p - y|$. In the following, we first discuss how to compute $\sum_{p \in P_i} f_p \cdot |x_p - x|$ and the case for $\sum_{p \in P_i} f_p \cdot |y_p - y|$ is very similar.

Let S_1 denote the set of all locations of P_i whose x-coordinates are smaller than or equal to x, i.e., the x-coordinate of q. Let $S_2 = P_i \setminus S_1$. Then, we have the following:

$$\begin{aligned}\sum_{p\in P_i} f_p \cdot |x_p - x| &= \sum_{p\in S_1} f_p \cdot (x - x_p) + \sum_{p\in S_2} f_p \cdot (x_p - x) \\ &= x \cdot \Big(\sum_{p\in S_1} f_p - \sum_{p\in S_2} f_p\Big) - \sum_{p\in S_1} f_p \cdot x_p + \sum_{p\in S_2} f_p \cdot x_p \\ &= x \cdot \Big(2 \cdot \sum_{p\in S_1} f_p - \sum_{p\in P_i} f_p\Big) - 2\sum_{p\in S_1} f_p \cdot x_p + \sum_{p\in P_i} f_p \cdot x_p.\end{aligned} \tag{5.1}$$

Thus, in order to compute $\sum_{p\in P_i} f_p \cdot |x_p - x|$, it is sufficient to know the four values $\sum_{p\in S_1} f_p$, $\sum_{p\in P_i} f_p$, $\sum_{p\in S_1} f_p \cdot x_p$, and $\sum_{p\in P_i} f_p \cdot x_p$. To this end, we do the following preprocessing.

First, we compute $\sum_{p\in P_i} f_p$ and $\sum_{p\in P_i} f_p \cdot x_p$, which can be done in $O(m)$ time. Second, recall that $X_i = \{x_{i0}, x_{i1}, \ldots, x_{im}\}$ maintains the x-coordinates of the locations of P_i sorted in ascending order. Note that given any index j with $1 \le j \le m$, we can access the information of the location of P_i whose x-coordinate is x_{ij} in constant time, and this can be done by linking each x_{ij} to the corresponding location of P_i when we create the list X_i from the input. For each j with $1 \le j \le n$, we let $f(j)$ be the probability associated with the location of P_i whose x-coordinate is x_{ij}.

In the preprocessing, we compute two arrays $A[0 \cdots m]$ and $B[0 \cdots m]$. Specifically, for each $1 \le j \le n$, $A[j] = \sum_{k=1}^{j} f(k)$ and $B[j] = \sum_{k=1}^{j} f(k) \cdot x_{ik}$. For $j = 0$, we let $A[0] = B[0] = 0$. As discussed above, since we can access $f(j)$ in constant time for any $1 \le j \le m$, the two arrays A and B can be computed in $O(m)$ time.

Let $t = I_q(X_i)$, i.e., the index of the predecessor of q in X_i. Note that $t \in [0, m]$. To compute $\sum_{p\in P_i} f_p \cdot |x_p - x|$, an easy observation is that $\sum_{p\in S_1} f_p$ is exactly equal to $A[t]$ and $\sum_{p\in S_1} f_p \cdot x_p$ is exactly equal to $B[t]$. Therefore, with the above preprocessing, if t is known, according to Equation (5.1), $\sum_{p\in P_i} f_p \cdot |x_p - x|$ can be computed in $O(1)$ time.

The above shows that with $O(m)$ time preprocessing, given $I_q(X_i)$, we can compute the function $\sum_{p\in P_i} f_p \cdot |x_p - x|$ of x at $q = (x, y)$ in constant time.

In a similar way, with $O(m)$ time preprocessing, given $I_q(Y_i)$, we can compute the function $\sum_{p\in P_i} f_p \cdot |y_p - y|$ of y at $p = (x, y)$ in constant time.

Let q be any point in the cell C. Hence, $I_q(X_i) = I_C(X_i)$ and $I_q(Y_i) = I_C(Y_i)$. Further, the function $\mathsf{Ed}_i(x, y)$ on $q = (x, y) \in C$ is exactly the function $\mathsf{Ed}_i(x, y, C)$. Therefore, with $O(m)$ time preprocessing, given $I_C(X_i)$ and $I_C(Y_i)$, we can compute the function $\mathsf{Ed}_i(x, y, C)$ in constant time.

The lemma thus follows. □

Due to Lemma 5.2.2, we have the following corollary.

Corollary 5.2.3. *For each uncertain point $P_i \in \mathcal{P}$, after $O(m)$ time preprocessing, given any point q in the plane, the expected distance $\mathsf{Ed}(P_i, q)$ can be computed in $O(\log m)$ time.*

Proof. Given any point $q \in \mathbb{R}^2$, we can compute $I_q(X_i)$ in $O(\log m)$ time by doing binary search on X_i. Similarly, we can compute $I_q(Y_i)$ in $O(\log m)$ time. Let C be the cell containing q. Recall that $I_C(X_i) = I_q(X_i)$ and $I_C(Y_i) = I_q(Y_i)$. Hence, by Lemma 5.2.2, we can compute the function $\mathsf{Ed}_i(x, y, C)$ in constant time. Then, $\mathsf{Ed}(P_i, q)$ is equal to $\mathsf{Ed}_i(q_x, q_y, C)$, where q_x and q_y are the x- and y-coordinates of q, respectively. Thus, after $\mathsf{Ed}_i(x, y, C)$ is known, $\mathsf{Ed}(P_i, q)$ can be computed in constant time. The corollary thus follows. □

Recall that $\mathsf{Ed}_{\max}(q) = \max_{P_i \in \mathcal{P}} \mathsf{Ed}(P_i, q)$ for any point q in the plane. For convenience, we use $\mathsf{Ed}_{\max}(x, y)$ to represent $\mathsf{Ed}_{\max}(q)$ as a function of $q = (x, y) \in \mathbb{R}^2$. Note that $\mathsf{Ed}_{\max}(x, y)$ is the upper envelope of the functions $\mathsf{Ed}_i(x, y)$ for all $i = 1, 2, \ldots, n$. Since each $\mathsf{Ed}_i(x, y)$ is convex on $\mathbb{R}^2$, $\mathsf{Ed}_{\max}(x, y)$ is also convex on $\mathbb{R}^2$. Further, the rectilinear center q^* corresponds to a lowest point p^* on $\mathsf{Ed}_{\max}(x, y)$. Specifically, q^* is the projection of p^* on the xy-plane. Therefore, computing q^* is equivalent to computing a lowest point in the upper envelope of all functions $\mathsf{Ed}_i(x, y)$ for all $i = 1, 2, \ldots, n$.

For each $1 \leq i \leq n$, let H_i denote the set of supporting planes of all surface patches of the function $\mathsf{Ed}_i(x, y)$. Let $\mathcal{H} = \cup_{i=1}^{n} H_i$. Since $\mathsf{Ed}_i(x, y)$ is convex, $\mathsf{Ed}_i(x, y)$ is essentially the upper envelope of the planes in H_i. Hence, $\mathsf{Ed}_{max}(x, y)$ is also the upper envelope of all planes in $\mathcal{H}$. Therefore, as discussed in Section 5.1.2, finding p^* is essentially a 3D LP problem on $\mathcal{H}$, which can be solved in $O(|\mathcal{H}|)$ time by Megiddo's technique [27]. Since the size of each H_i is $\Theta(m^2)$, $|\mathcal{H}| = \Theta(nm^2)$. Therefore, applying

the algorithm in [27] directly can solve the problem in $O(nm^2)$ time. In the following, we give an $O(nm)$ time algorithm.

In the following sections, we assume we have done the preprocessing of Lemma 5.2.2 for each $P_i \in \mathcal{P}$, which takes $O(mn)$ time in total.

5.3 The Decision Algorithm

In this section, we present a decision algorithm that solves a decision problem, which is needed later in Section 5.4. We first introduce the decision problem.

Let $R = [x_1, x_2] \times [y_1, y_2]$ be an axis-parallel rectangle in the plane, where x_1 and x_2 are the x-coordinates of the left and right sides of R, respectively, and y_1 and y_2 are the y-coordinates of the bottom and top sides of R, respectively. Suppose it is known that q^* is in R (but the exact location of q^* is not known). Let L be an arbitrary line that intersects the interior of R. The *decision problem* asks whether q^* is on L, and if not, which side of L contains q^*. We assume the two predecessor indices $I_{x_1}(X_i)$ and $I_{y_1}(Y_i)$ are already known.

For each $1 \leq i \leq n$, let $a_i = I_{x_2}(X_i) - I_{x_1}(X_i) + 1$ and $b_i = I_{y_2}(Y_i) - I_{y_1}(Y_i) + 1$. In fact, a_i and b_i are the numbers of columns and rows of G_i that intersect R, respectively. Below, we give a *decision algorithm* that solves the decision problem in $O(\sum_{i=1}^{n}(a_i+b_i))$ time. Note that for each $1 \leq i \leq n$, it holds that $1 \leq a_i, b_i \leq m+1$, and thus $2n \leq \sum_{i=1}^{n}(a_i + b_i) \leq 2(m+1)n$.

We first show that the decision problem can be solved in $O(\sum_{i=1}^{n} a_i \cdot b_i)$ time by using the decision algorithm for the 3D LP problem [27]. Later we will reduce the running time to $O(\sum_{i=1}^{n}(a_i + b_i))$ time.

Recall that p^* is a lowest point in the upper envelope of the functions $\mathsf{Ed}_i(x, y)$ for $i = 1, 2, \ldots, n$. Since q^* is in R and each function $\mathsf{Ed}_i(x, y)$ is convex, an easy observation is that p^* is also a lowest point in the upper envelope of $\mathsf{Ed}_i(x, y)$ for $i = 1, 2, \ldots, n$ restricted on $(x, y) \in R$. This implies that we only need to consider each function $\mathsf{Ed}_i(x, y)$ restricted on R.

For each $1 \leq i \leq n$, let $G_i(R)$ be the set of cells of G_i that intersect R, and let $H_i(R)$ be the set of supporting planes of the surface patches of $\mathsf{Ed}(P_i, q)$ defined on the cells of $G_i(R)$. Let $\mathcal{H}(R) = \cup_{i=1}^{n} H_i(R)$. By our above analysis, p^* is a lowest point of

the upper envelope of all planes in $\mathcal{H}(R)$. Note that $|H_i(R)| = a_i \cdot b_i$ for each $1 \leq i \leq n$. Thus, $|\mathcal{H}(R)| = \sum_{i=1}^{n} a_i \cdot b_i$. Then, we can apply the decision algorithm in [27] (Section 5.2) on $\mathcal{H}(R)$ to determine which side of L contains q^* in $O(|\mathcal{H}(R)|)$ time. In order to explain our improved algorithm later, we sketch this algorithm below.

We consider each plane of $\mathcal{H}(R)$ as a function of the points q on the xy-plane $\mathbb{R}^2$. In the first step, the algorithm finds a point q' on L that minimizes the maximum value of all functions in $\mathcal{H}(R)$ restricted on $q \in L$. This is essentially a 2D LP problem because each function of $\mathcal{H}(R)$ restricted on L is a line, and thus the problem can be solved in $O(|\mathcal{H}(R)|)$ time [27]. Let $\Phi_{q'}$ be the set of functions of $\mathcal{H}(R)$ whose values at q' are equal to the above maximum value. The set $\Phi_{q'}$ can be found in $O(|\mathcal{H}(R)|)$ time after q' is computed. This finishes the first step, which takes $O(|\mathcal{H}(R)|) = O(\sum_{i=1}^{n} a_i \cdot b_i)$ time.

The second step solves another two instances of the 2D LP problem on the planes of $\Phi_{q'}$, which takes $O(|\Phi_{q'}|)$ time. An easy upper bound for $|\Phi_{q'}|$ is $\sum_{i=1}^{n} a_i \cdot b_i$. A close analysis can show that $|\Phi_{q'}| = O(n)$. Indeed, for each $1 \leq i \leq n$, since the function $\mathsf{Ed}_i(x, y)$ is convex, among all $a_i \cdot b_i$ planes in $H_i(R)$, at most four of them are in $\Phi_{q'}$. Therefore, $|\Phi_{q'}| = O(n)$. Hence, the second step runs in $O(n)$ time. Since in our problem there always exists a solution, according to [27], the second step will either conclude that q' is q^* or tell which side of L contains q^*, which solves the decision problem. The algorithm takes $O(\sum_{i=1}^{n} a_i \cdot b_i)$ time in total, which is dominated by the first step.

In the sequel, we reduce the running time of the above algorithm, in particular, the first step, to $O(\sum_{i=1}^{n}(a_i + b_i))$. Our goal is to compute q' and $\Phi_{q'}$. By the definition, q' is a lowest point in the upper envelope of all functions of $\mathcal{H}(R)$ restricted on the line L. Consider any uncertain point $P_i \in \mathcal{P}$. Let $H_i(R, L)$ be the set of supporting planes of the surface patches defined on the cells of $G_i(R)$ intersecting L. Observe that since $\mathsf{Ed}_i(x, y)$ is convex, the upper envelope of all the functions of $H_i(R)$ restricted on L is exactly the upper envelope of the functions of $H_i(R, L)$ restricted on L. Therefore, q' is also a lowest point in the upper envelope of the functions of $\mathcal{H}(R, L)$ restricted on L, where $\mathcal{H}(R, L) = \cup_{i=1}^{n} H_i(R, L)$. In other words, among all planes in $\mathcal{H}(R)$, only the planes of $\mathcal{H}(R, L)$ are relevant for determining q'. Thus, suppose $\mathcal{H}(R, L)$ has been computed; then q' can be computed based on the planes of $\mathcal{H}(R, L)$ in $O(|\mathcal{H}(R, L)|)$ time

by the 2D LP algorithm [27]. After q' is computed, the set $\Phi_{q'}$ can also be determined in $O(|\mathcal{H}(R, L)|)$ time.

Note that $|\mathcal{H}(R, L)| = O(\sum_{i=1}^{n}(a_i + b_i))$, since for each $1 \leq i \leq n$, $|H_i(R, L)|$, which is equal to the number of cells of $G_i(R)$ intersecting L, is $O(a_i + b_i)$.

It remains to compute $\mathcal{H}(R, L)$, i.e., compute $H_i(R, L)$ for each $1 \leq i \leq n$. Recall that $R = [x_1, x_2] \times [y_1, y_2]$ and the two predecessor indices $I_{x_1}(X_i)$ and $I_{y_1}(Y_i)$ for each $1 \leq i \leq n$ are already known. The following lemma gives an $O(a_i + b_i)$-time algorithm to compute $H_i(R, L)$.

Lemma 5.3.1. *For each $1 \leq i \leq n$, $H_i(R, L)$ can be computed in $O(a_i + b_i)$ time.*

Proof. Let $G_i(R, L)$ be the set of cells of $G_i(R)$ intersecting L. To compute the planes in $H_i(R, L)$, it is sufficient to determine the plane surface patches of $\mathsf{Ed}_i(x, y)$ defined on the cells of $G_i(R, L)$. By Lemma 5.2.2, this amounts to determine the indices of the predecessors of these cells in X_i and Y_i, respectively. In the following, we give an algorithm to compute the cells of $G_i(R, L)$ and determine their predecessor indices in X_i and Y_i, respectively, and the algorithm runs in $O(a_i + b_i)$ time.

The main idea is that we first pick a particular point p on $L \cap R$ and locate the cell of $G_i(R)$ containing p (clearly this cell belongs to $G_i(R, L)$), and then starting from p, we traverse on L and $G_i(R)$ simultaneously to trace other cells of $G_i(R, L)$ until we move out of R. The details are given below.

We focus on the case where L has a positive slope. The other cases can be handled similarly. Recall that L intersects the interior of R. Let p be the leftmost intersection of L with the boundary of R. Hence, p is either on the left side or the bottom side of R.

Let C be the cell of G_i that contains p. We first determine the two indices $I_p(X_i)$ and $I_p(Y_i)$. Note that $I_C(X_i) = I_p(X_i)$ and $I_C(Y_i) = I_p(Y_i)$.

Since $p \in R$, the index $I_p(X_i)$ can be found in $O(a_i)$ time by scanning the list X_i from the index $I_{x_1}(X_i)$. Similarly, $I_p(Y_i)$ can be found in $O(b_i)$ time by scanning the list Y_i from the index $I_{y_1}(Y_i)$. After $I_C(X_i) = I_p(X_i)$ and $I_C(Y_i) = I_p(Y_i)$ are computed, by Lemma 5.2.2, the function $\mathsf{Ed}_i(x, y, C)$ can be computed in constant time, and we add the function to $H_i(R, L)$.

Next, we move p on L rightwards. We will show that when p crosses the boundary of C, we can determine the new cell containing p and update the two indices $I_p(X_i)$ and $I_p(Y_i)$ in constant time. This process continues until p moves out of R. Specifically, when p moves on L rightwards, p will cross the boundary of C either from the top side or the right side.

First, we determine whether p will move out of R before p crosses the boundary of C. If yes, then we terminate the algorithm. Otherwise, we determine whether p moves out of C from its right side or left side. All above can be easily done in constant time. Depending on whether p crosses the boundary of C from its top side, right side, or from both sides simultaneously, there are three cases.

1. If p crosses the boundary of C from the top side and p does not cross the right side of C, then p enters into a new cell that is on top of C. We update C to the new cell. We increase the index $I_p(Y_i)$ by one, but keep $I_p(X_i)$ unchanged. Clearly, the above two indices are correctly updated and $I_C(X_i) = I_p(X_i)$ and $I_C(Y_i) = I_p(Y_i)$ for the new cell C. Again, by Lemma 5.2.2, the function $\mathsf{Ed}_i(x, y, C)$ for the new cell C can be computed in constant time. We add the new function to $H_i(R, L)$.

2. If p crosses the boundary of C from the right side and p does not cross the top side of C, then p enters into a new cell that is on right of C. The algorithm in this case is similar to the above case and we omit the discussions.

3. The remaining case is when p crosses the boundary of C through the top right corner of C. In this case, p enters into the northeast neighboring cell of C. We first add to $H_i(R, L)$ the supporting planes of the surface patches of $\mathsf{Ed}_i(x, y)$ defined on the top neighboring cell and the right neighboring cell of C, which can be computed in constant time as the above two cases. Then, we update C to the new cell p is entering. We increase each of $I_p(X_i)$ and $I_p(Y_i)$ by one. Again, the two indices are correctly updated for the new cell C. Finally, we compute the new function $\mathsf{Ed}_i(x, y, C)$ and add it to $H_i(R, L)$.

When the algorithm stops, $H_i(R, L)$ is computed. In general, during the procedure of moving p on L, we spend constant time on finding each supporting plane of $H_i(R, L)$.

Therefore, the total running time of the entire algorithm is $O(a_i + b_i)$. The lemma thus follows. □

With the preceding lemma, we have the following result.

Theorem 5.3.2. *The decision problem can be solved in* $O(\sum_{i=1}^{n}(a_i + b_i))$ *time.*

5.4 Computing the Rectilinear Center

In this section, with the help of our decision algorithm in Section 5.3, we compute the rectilinear center q^* in $O(mn)$ time.

As discussed in Section 5.1.2, our algorithm is a prune-and-search algorithm that has $O(\log n)$ "outer" recursive steps each of which has $O(\log m)$ "inner" recursive steps. In each outer recursive step, the algorithm prunes at least $|\mathcal{P}|/32$ uncertain points of $\mathcal{P}$ such that these uncertain points are not relevant for computing q^*. After $O(\log n)$ outer recursive steps, there will be only a constant number of uncertain points remaining in $\mathcal{P}$. Each outer recursive step runs in $O(m|\mathcal{P}|)$ time, where $|\mathcal{P}|$ is the number of uncertain points remaining in $\mathcal{P}$. In this way, the total running time of the algorithm is $O(mn)$.

Each outer recursive step is another recursive prune-and-search algorithm, which consists of $2 + \log m$ inner recursive steps. Let $\mathcal{X} = \cup_{i=1}^{n} X_i$ and $\mathcal{Y} = \cup_{i=1}^{n} Y_i$. Hence, $|\mathcal{X}| = |\mathcal{Y}| = mn$. We maintain a rectangle $R = [x_1, x_2] \times [y_1, y_2]$ that contains q^*. Initially, R is the entire plane. In each inner recursive step, we shrink R such that the *x-range* $[x_1, x_2]$ (resp., *y-range* $[y_1, y_2]$) of the new R only contains half of the values of $\mathcal{X}$ (resp., $\mathcal{Y}$) in the x-range (resp., y-range) of the previous R. In this way, after $\log m + 2$ inner recursive steps, the x-range (resp., y-range) of R only contains at most $n/4$ values of $\mathcal{X}$ (resp., $\mathcal{Y}$). At this moment, a *key observation* is that there is a subset $\mathcal{P}^*$ of at least $n/2$ uncertain points, such that for each $P_i \in \mathcal{P}^*$, R is contained in the interior of a single cell of the grid G_i, i.e., the x-range (resp., y-range) of R does not contain any value of X_i (resp., Y_i). Due to the observation, we can use a pruning procedure similar to that in [27] to prune at least $|\mathcal{P}^*|/16 \geq n/32$ uncertain points.

In the following, in Section 5.4.1, we give our algorithm on pruning the values of $\mathcal{X}$ and $\mathcal{Y}$ to obtain $\mathcal{P}^*$. In Section 5.4.2, we prune uncertain points of $\mathcal{P}^*$.

5.4.1 Pruning the Coordinate Values of $\mathcal{X}$ and $\mathcal{Y}$

Consider a general step of the algorithm where we are about to perform the j-th inner recursive step for $1 \leq j \leq \log m + 2$. Our algorithm maintains the following *algorithm invariants*. (1) We have a rectangle $R^{j-1} = [x_1^{j-1}, x_2^{j-1}] \times [y_1^{j-1}, y_2^{j-1}]$ that contains q^*. (2) For each $1 \leq i \leq n$, the index $I_{x_1^{j-1}}(X_i)$ of the predecessor of x_1^{j-1} in X_i is known, and so is the index $I_{y_1^{j-1}}(Y_i)$. (3) We have a sublist X_i^{j-1} of X_i that consists of all values of X_i in $[x_1^{j-1}, x_2^{j-1}]$ and a sublist Y_i^{j-1} of Y_i that consists of all values of Y_i in $[y_1^{j-1}, y_2^{j-1}]$. Note that these sublists can be empty. (4) $|\mathcal{X}^{j-1}| \leq mn/2^{j-1}$ and $|\mathcal{Y}^{j-1}| \leq mn/2^{j-1}$, where $\mathcal{X}^{j-1} = \cup_{i=1}^{n} X_i^{j-1}$ and $\mathcal{Y}^{j-1} = \cup_{i=1}^{n} Y_i^{j-1}$.

Initially, we set $R^0 = [-\infty, +\infty] \times [-\infty, +\infty]$, $X_i^0 = X_i$ and $Y_i^0 = Y_i$ for each $1 \leq i \leq n$, with $\mathcal{X}^0 = \mathcal{X}$ and $\mathcal{Y}^0 = \mathcal{Y}$. It is easy to see that before we start the first inner recursive step for $j = 1$, all the algorithm invariants hold.

In the sequel, we give the details of the j-th inner recursive step. We will show that its running time is $O(mn/2^j + n)$ and all algorithm invariants are still maintained after the step.

Let x_m be the median of $\mathcal{X}^{j-1}$ and y_m be the median of $\mathcal{Y}^{j-1}$. Both x_m and y_m can be found in $O(|\mathcal{X}^{j-1}| + |\mathcal{Y}^{j-1}|)$ time.

For each $1 \leq i \leq n$, let $a_i^{j-1} = I_{x_2^{j-1}}(X_i) - I_{x_1^{j-1}}(X_i) + 1$ and $b_i^{j-1} = I_{y_2^{j-1}}(Y_i) - I_{y_1^{j-1}}(Y_i) + 1$. Observe that $a_i^{j-1} = |X_i^{j-1}| + 1$ and $b_i^{j-1} = |Y_i^{j-1}| + 1$.

Let x^* and y^* be the x- and y-coordinates of q^*, respectively.

We first determine whether $x^* > x_m$, $x^* < x_m$, or $x^* = x_m$. This can be done by applying our decision algorithm on R^{j-1} and L with L being the vertical line $x = x_m$. By Theorem 5.3.2, the running time of our decision algorithm is $O(\sum_{i=1}^{n}(a_i^{j-1} + b_i^{j-1}))$, which is $O(n + |\mathcal{X}^{j-1}| + |\mathcal{Y}^{j-1}|)$.

Note that if $x^* = x_m$, then q^* will be found by the decision algorithm and we can terminate the entire algorithm. Otherwise, without loss of generality, we assume $x^* > x_m$. We proceed to determine whether $y^* > y_m$ or $y^* < y_m$, or $y^* = y_m$ by applying our decision algorithm on R^{j-1} and L with L being the horizontal line $y = y_m$. Similarly, if $y^* = y_m$, then the decision algorithm will find q^* and we are done. Otherwise, without loss of generality we assume $y^* > y_m$. The above calls our decision algorithm twice, which takes $O(n + |\mathcal{X}^{j-1}| + |\mathcal{Y}^{j-1}|)$ time in total.

Now we know that q^* is in the rectangle $[x_m, x_2^{j-1}] \times [y_m, y_2^{j-1}]$. We let $R^j = [x_1^j, x_2^j] \times [y_1^j, y_2^j]$ be the above rectangle, i.e., $x_1^j = x_m$, $x_2^j = x_2^{j-1}$, $y_1^j = y_m$, and $y_2^j = y_2^{j-1}$. Clearly, the first algorithm invariant is maintained.

We further proceed as follows to maintain the other three invariants.

For each $1 \leq i \leq n$, by scanning the sorted list X_i^{j-1}, we compute the index $I_{x_1^j}(X_i)$ of the predecessor of x_1^j in X_i (each element of X_i^{j-1} maintains its original index in X_i), and similarly, by scanning the sorted list Y_i^{j-1}, we compute the index $I_{y_1^j}(Y_i)$. Computing these indices in all X_i and Y_i for $i = 1, 2, \ldots, n$ can be done in $O(|\mathcal{X}^{j-1}| + |\mathcal{Y}^{j-1}|)$ time. This maintains the second algorithm invariant.

Next, for each $1 \leq i \leq n$, we scan X_i^{j-1} to compute a sublist X_i^j, which consists of all values of X_i^{j-1} in $[x_1^j, x_2^j]$, and similarly, we scan Y_i^{j-1} to compute a sublist Y_i^j, which consists of all values of Y_i^{j-1} in $[y_1^j, y_2^j]$. Computing the lists X_i^j and Y_i^j for all $i = 1, 2, \ldots, n$ as above can be done in overall $O(|\mathcal{X}^{j-1}| + |\mathcal{Y}^{j-1}|)$ time. This maintains the third algorithm invariant.

Let $\mathcal{X}^j = \cup_{i=1}^n X_i^j$ and $\mathcal{Y}^j = \cup_{i=1}^n Y_i^j$. According to our above algorithm, $|\mathcal{X}^j| \leq |\mathcal{X}^{j-1}|/2$ and $|\mathcal{Y}^j| \leq |\mathcal{Y}^{j-1}|/2$. Since $|\mathcal{X}^{j-1}| \leq nm/2^{j-1}$ and $|\mathcal{Y}^{j-1}| \leq nm/2^{j-1}$, we obtain $|\mathcal{X}^j| \leq nm/2^j$ and $|\mathcal{Y}^j| \leq nm/2^j$. Hence, the fourth algorithm invariant is maintained.

In summary, after the j-th inner recursive step, all four algorithm invariants are maintained. Our above analysis also shows that the total running time is $O(n + |\mathcal{X}^{j-1}| + |\mathcal{Y}^{j-1}|)$, which is $O(nm/2^j + n)$.

We stop the algorithm after the t-th inner recursive step, for $t = 2 + \log m$. The total time for all t steps is thus $O(\sum_{j=1}^t (n + mn/2^j)) = O(mn)$.

After the t-th step, by our algorithm invariants, the rectangle R^t contains q^*, and $|\mathcal{X}^t| \leq mn/2^t = n/4$ and $|\mathcal{Y}^t| \leq mn/2^t = n/4$.

We say that an uncertain point P_i is *prunable* if both X_i^t and Y_i^t are empty (and thus R^t is contained in the interior of a cell of G_i). Let $\mathcal{P}^*$ denote the set of all prunable uncertain points of $\mathcal{P}$. The following is an easy but crucial observation.

Observation 5.4.1. $|\mathcal{P}^*| \geq n/2$.

Proof. Since $\mathcal{X}^t \leq n/4$, among the n sets X_i^t for $i = 1, 2, \ldots, n$, at most $n/4$ of them are non-empty. Similarly, since $\mathcal{Y}^t \leq n/4$, among the n sets Y_i^t for $i = 1, 2, \ldots, n$, at most $n/4$ of them are non-empty. Therefore, there are at most $n/2$ uncertain points $P_i \in \mathcal{P}$ such that either X_i^t or Y_i^t is non-empty. This implies that there are at least $n/2$ prunable uncertain points in $\mathcal{P}$. □

After the t-th inner recursive step, the set $\mathcal{P}^*$ can be obtained in $O(n)$ time by checking all sets X_i^t and Y_i^t for $i = 1, 2, \ldots, n$ and see whether they are empty.

The reason we are interested in prunable uncertain points is that for each prunable uncertain point P_i of $\mathcal{P}^*$, since R^t contains q^* and R^t is contained in a single cell C_i of G_i, there is only one surface patch of $\mathsf{Ed}_i(x, y)$ (i.e., the one defined on C_i) that is relevant for computing q^*. Let h_i denote the supporting plane of the above surface patch. We call h_i the *relevant plane* of P_i. Note that we can obtain h_i in constant time. Indeed, observe that the predecessor index $I_{C_i}(X_i)$ is exactly $I_{x_1^t}(X_i)$, which is known by our algorithm invariants. Similarly, the index $I_{C_i}(Y_i)$ is also known. By Lemma 5.2.2, the function $\mathsf{Ed}_i(x, y, C_i)$, which is also the function of h_i, can be obtained in constant time. Hence, the relevant planes of all prunable uncertain points of $\mathcal{P}^*$ can be obtained in $O(n)$ time.

Remark.. One may wonder why we did not perform the inner recursive steps for $t = \log mn$ times (instead of $t = 2 + \log m$ time) so that $\mathcal{X}^t$ and $\mathcal{Y}^t$ would each have a constant number of values in the range of R. The reason is that based on our analysis, that would take $O(mn + n \log nm)$ time, which may not be bounded by $O(mn)$ (e.g., when $m = o(\log n)$). In fact, performing the inner recursive steps for $t = 2 + \log m$ times such that $\mathcal{X}^t$ and $\mathcal{Y}^t$ each have at most $\frac{n}{4}$ values in the range of R is an interesting and crucial ingredient of our techniques.

5.4.2 Pruning Uncertain Points from $\mathcal{P}^*$

Consider a prunable uncertain point P_i of $\mathcal{P}^*$. Recall that H_i is the set of supporting planes of all surface patches of $\mathsf{Ed}_i(x, y)$. The above analysis shows that among all planes in H_i, only the relevant plane h_i is useful for determining q^*. In other words, the point p^*, as a lowest point of all planes in $\mathcal{H} = \cup_{i=1}^n H_i$, is also a lowest point of the planes

in the union of $\cup_{P_i \in \mathcal{P}^*} h_i$ and $\cup_{P_i \in \mathcal{P} \setminus \mathcal{P}^*} H_i$. This will allow us to prune at least $|\mathcal{P}^*|/16$ uncertain points from $\mathcal{P}^*$. The idea is similar to Megiddo's pruning scheme for the 3D LP algorithm in [27].

For each $P_i \in \mathcal{P}^*$, its relevant plane h_i is also considered as a function in the xy-plane. Arrange all uncertain points of $\mathcal{P}^*$ into $|\mathcal{P}^*|/2$ disjoint pairs. Let $D(\mathcal{P}^*)$ denote the set of all these pairs. For each pair $(P_i, P_j) \in D(\mathcal{P}^*)$, if the value of the function h_i at any point of R^t is greater than or equal to that of h_j, then P_j can be pruned immediately; otherwise, we project the intersection of h_i and h_j on the xy-plane to obtain a line L_{ij} dividing R^t into two parts, such that $h_i \geq h_j$ on one part and $h_i \leq h_j$ on the other.

Let $\mathcal{L}$ denote the set of the dividing lines L_{ij} for all pairs of $D(\mathcal{P}^*)$. Let L_m be the line whose slope has the median value among the lines of $\mathcal{L}$. We transform the coordinate system by rotating the x-axis to be parallel to L_m (the y-axis does not change). For ease of discussion, we assume no other lines of $\mathcal{L}$ are parallel to L_m (the assumption can be easily lifted; see [27]). In the new coordinate system, half the lines of $\mathcal{L}$ have negative slopes and the other half have positive slopes. We now arrange all lines of $\mathcal{L}$ into disjoint pairs such that each pair has a line of a negative slope and a line of a positive slope. Let $D(\mathcal{L})$ denote the set of all these line pairs.

Let x^* and y^* respectively be the x- and y-coordinate of the center q^* in the new coordinate system. For each pair $(L_i, L_j) \in D(\mathcal{L})$, we define y_{ij} as the y-coordinate of the intersection of L_i and L_j. We find the median y_m of the values y_{ij} for all pairs in $D(\mathcal{L})$. We determine in $O(mn)$ time whether $y^* > y_m$, $y^* < y_m$ or $y^* = y_m$ by using our decision algorithm (here an $O(mn)$ time decision algorithm is sufficient for our purpose). If $y^* = y_m$, then our decision algorithm will find q^* and we can terminate the algorithm. Otherwise, without of loss generality, we assume $y^* < y_m$.

Let $D'(\mathcal{L})$ denote the set of all pairs (L_i, L_j) of $D(\mathcal{L})$ such that $y_{ij} \geq y_m$. Note that $|D'(\mathcal{L})| \geq |D(\mathcal{L})|/2$. For each pair $(L_i, L_j) \in D'(\mathcal{L})$, let x_{ij} be the x-coordinate of the intersection of L_i and L_j. We find the median x_m of all such x_{ij}'s. By using our decision algorithm, we can determine in $O(mn)$ time whether $x^* > x_m$, $x^* < x_m$, or $x^* = x_m$. If $x^* = x_m$, our decision algorithm will find q^* and we are done. Otherwise, without loss of generality, we assume $x^* < x_m$.

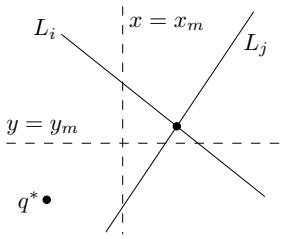

Figure 5.2. The intersection of L_i and L_j is in the first quarter of the intersection of $x = x_m$ and $y = y_m$. Note that q^* is in the interior of the third quarter.

Now for each pair (L_i, L_j) of $D'(\mathcal{L})$ with $x_{ij} \geq x_m$ and $y_{ij} \geq y_m$ (there are at least $|D'(\mathcal{L})|/2$ such pairs), we can prune either P_i or P_j, as follows. Indeed, one of the lines in such a pair (L_i, L_j), say L_i, has a negative slope and does not intersect the region $A = \{(x, y) \mid x < x_m,\ y < y_m\}$ (e.g., see Fig. 5.2). Suppose L_i is the dividing line of the relevant planes h_{k_1} and h_{k_2} of two uncertain points P_{k_1} and P_{k_2} of $\mathcal{P}^*$. It follows that either $h_{k_1} \geq h_{k_2}$ or $h_{k_1} \leq h_{k_2}$ holds on the region A. Since $q^* \in A$, one of P_{k_1} and P_{k_2} can be pruned.

As a summary, the above pruning algorithm prunes at least $|\mathcal{P}^*|/16 \geq n/32$ uncertain points and the total time is $O(mn)$.

5.4.3 Wrapping Things Up

The algorithm in the above two subsections either computes q^* or prunes at least $n/32$ uncertain points from $\mathcal{P}$ in $O(mn)$ time. We assume the latter case happens. Then we apply the same algorithm recursively on the remaining uncertain points for $O(\log n)$ steps, after which only a constant number of uncertain points remain. The total running time can be described by the following recurrence: $T(m, n) = T(m, \frac{31 \cdot n}{32}) + O(mn)$. Solving the recurrence gives $T(m, n) = O(mn)$.

Let $\mathcal{P}'$ be the set of the remaining uncertain points, with $|\mathcal{P}'| = O(1)$. Hence, the rectilinear center q^* is determined by $\mathcal{P}'$. In other words, q^* is also a rectilinear center of $\mathcal{P}'$. In fact, like other standard prune-and-search algorithms, the way we prune uncertain points of $\mathcal{P}$ guarantees that any rectilinear center of $\mathcal{P}$ is also a rectilinear center of $\mathcal{P}'$, and vice versa. By using an approach similar to that in Section 5.4.1, Lemma 5.4.2 finally computes q^* based on $\mathcal{P}'$ in additional $O(m)$ time.

Lemma 5.4.2. *The rectilinear center q^* of $\mathcal{P}'$ can be computed in $O(m)$ time.*

Proof. Let $c = |\mathcal{P}'|$, which is a constant. Let $\mathcal{X}' = \cup_{P_i \in \mathcal{P}'} X_i$ and $\mathcal{Y}' = \cup_{P_i \in \mathcal{P}'} Y_i$. We apply the same recursive algorithm in Section 5.4.1 on $\mathcal{X}'$ and $\mathcal{Y}'$ for $O(\log m)$ steps, after which we will obtain a rectangle R such that R contains q^* and for each $P_i \in \mathcal{P}'$, the x-range (resp., y-range) of R only contains a constant number of values of X_i (resp., Y_i), and thus R intersects a set $G_i(R)$ of only a constant number of cells of G_i. Therefore, for each $P_i \in \mathcal{P}'$, only the surface patches of $\mathsf{Ed}_i(x, y)$ defined on the cells of $G_i(R)$ are relevant for computing q^*. The supporting planes of these surface patches can be determined immediately after the above $O(\log m)$ recursive steps. By the same analysis as in Section 5.4.1, all above can be done in $O(c \cdot m)$ time.

The above found $O(c)$ "relevant" supporting planes such that q^* corresponds to a lowest point in the upper envelope of them. Consequently, q^* can be found in $O(c)$ time by applying the linear-time algorithm for the 3D LP problem [27] on these $O(c)$ relevant supporting planes. □

This finishes our algorithm for computing q^*, which runs in $O(mn)$ time.

Theorem 5.4.3. *A rectilinear center q^* of the uncertain points of $\mathcal{P}$ in the plane can be computed in $O(mn)$ time.*

CHAPTER 6

THE K-CENTER PROBLEM OF UNCERTAIN POINTS ON TREE NETWORKS

In this chapter, we study the k-center problem of uncertain points on a tree. To solve this problem, we first present an algorithm to solve a *center-coverage* problem, which is the decision or dual version of the k-center problem. We then use it to solve the k-center problem. A preliminary version of this chapter will appear in the 14th Algorithms and Data Structures Symposium (WADS 2017).

6.1 Introduction

As in Chapter 3, let T be a tree, and we consider each edge e of T as a line segment so that we can talk about "points" on e. Formally, we specify a point x of T by an edge e of T that contains x and the distance between x and an incident vertex of e. The distance of any two points p and q on T, denoted by $d(p, q)$, is defined as the length of the simple path from p to q in T. Let $\mathcal{P} = \{P_1, \ldots, P_n\}$ be a set of n uncertain (demand) points on T. Each $P_i \in \mathcal{P}$ has m_i possible locations on T, denoted by $\{p_{i1}, p_{i2}, \cdots, p_{im_i}\}$, and each location p_{ij} of P_i is associated with a probability $f_{ij} \geq 0$ for P_i appearing at p_{ij} (which is independent of other locations), with $\sum_{j=1}^{m_i} f_{ij} = 1$; e.g., see Fig. 3.1 in Chapter 3. In addition, each $P_i \in \mathcal{P}$ has a weight $w_i \geq 0$. For any point x on T, the (weighted) *expected distance* from x to P_i, denoted by $\mathsf{Ed}(x, P_i)$, is defined as

$$\mathsf{Ed}(x, P_i) = w_i \cdot \sum_{j=1}^{m_i} f_{ij} \cdot d(x, p_{ij}).$$

Given a value $\lambda \geq 0$, called the *covering range*, we say that a point x on T *covers* an uncertain point P_i if $\mathsf{Ed}(x, P_i) \leq \lambda$. The *center-coverage problem* is to compute a minimum number of points on T, called *centers*, such that every uncertain point of $\mathcal{P}$ is covered by at least one center (hence we can build facilities on these centers to "serve" all demand points).

Let M denote the total number of locations all uncertain points, i.e., $M = \sum_{i=1}^{n} m_i$. In this chapter, we present an algorithm that solves the problem in $O(|T| + M \log^2 M)$ time, which is nearly linear as the input size of the problem is $\Theta(|T| + M)$.

As an application of our algorithm, we can solve the *k-center* problem, which computes a given number of k centers on T such that the covering range can be minimized. Our algorithm solves it in $O(|T| + n^2 \log n \log M + M \log^2 M \log n)$ time.

6.1.1 Related Work

If each $P_i \in \mathcal{P}$ has a single "certain" location, this problem is essentially the weighted case for "deterministic" points and the center-coverage problem is solvable in linear time [72] and the k-center problem is solvable in $O(n \log^2 n)$ time [31, 43].

If T is a path, both the center-coverage problem and the k-center problem on uncertain points have been studied in Chapter 2, but under a somewhat special problem setting where m_i is the same for all $1 \leq i \leq n$. The two problems were solved in $O(M + n \log k)$ and $O(M \log M + n \log k \log n)$ time, respectively. If T is tree, an $O(|T| + M)$ time algorithm was given in Chapter 3 for the one-center problem under the above special problem setting.

As mentioned above, the "deterministic" version of the center-coverage problem is solvable in linear time [72], where all demand points are on the vertices. For the k-center problem, Megiddo and Tamir [31] presented an $O(n \log^2 n \log \log n)$ time algorithm (n is the size of the tree), which was improved to $O(n \log^2 n)$ time by Cole [43]. The unweighted case was solved in linear time by Frederickson [30].

Facility location problems in other uncertain models have also been considered. For example, Löffler and van Kreveld [63] gave algorithms for computing the smallest enclosing circle for imprecise points each of which is contained in a planar region (e.g., a circle or a square). Jørgenson et al. [62] studied the problem of computing the distribution of the radius of the smallest enclosing circle for uncertain points each of which has multiple locations in the plane. de Berg et al. [64] proposed algorithms for dynamically maintaining Euclidean 2-centers for a set of moving points in the plane (the moving points are considered uncertain). See also the problems for minimizing the maximum regret, e.g., [37, 38, 65].

Our center-coverage problem can also be considered as a geometric coverage problem, which has been studied in various settings. For example, the unit disk coverage problem is to compute a minimum number of unit disks to cover a given set of points in the plane. The problem is NP-hard and a polynomial-time approximation scheme was known [73]. The discrete case where the disks must be selected from a given set was also studied [74]. See [75–78] and the references therein for various problems of covering points using squares. Refer to a survey [79] for more geometric coverage problems.

6.1.2 Our Techniques

We first discuss our techniques for solving the center-coverage problem.

For each uncertain point $P_i \in \mathcal{P}$, we find a point p_i^* on T that minimizes the expected distance $\mathsf{Ed}(p_i, P_i)$, and p_i^* is actually the weighted median of all locations of P_i. We observe that if we move a point x on T away from p_i^*, the expected distance $\mathsf{Ed}(x, P_i)$ is monotonically increasing. We compute the medians p_i^* for all uncertain points in $O(M \log M)$ time. Then we show that there exists an optimal solution in which all centers are in T_m, where T_m is the minimum subtree of T that connects all medians p_i^* (so every leaf of T_m is a median p_i^*). Next we find centers on T_m. To this end, we propose a simple greedy algorithm, but the challenge is on developing efficient data structures to perform certain operations. We briefly discuss it below.

We pick an arbitrary vertex r of T_m as the root. Starting from the leaves, we consider the vertices of T_m in a bottom-up manner and place centers whenever we "have to". For example, consider a leaf v holding a median p_i^* and let u be the parent of v. If $\mathsf{Ed}(u, P_i) > \lambda$, then we have to place a center c on the edge $e(u, v)$ in order to cover P_i. The location of c is chosen to be at a point of $e(u, v)$ with $\mathsf{Ed}(c, P_i) = \lambda$ (i.e., on the one hand, c covers P_i, and on the other hand, c is as close to u as possible in the hope of covering other uncertain points as many as possible). After c is placed, we find and remove all uncertain points that are covered by c. Performing this operation efficiently is a key difficulty for our approach. We solve the problem in an output-sensitive manner by proposing a dynamic data structure that also supports the remove operations.

We also develop data structures for other operations needed in the algorithm. For example, we build a data structure in $O(M \log M)$ time that can compute the expected

distance $\mathsf{Ed}(x, P_i)$ in $O(\log M)$ time for any point x on T and any $P_i \in \mathcal{P}$. These data structures may be of independent interest.

For solving the k-center problem, by observations, we first identify a set of $O(n^2)$ "candidate" values such that the covering range in the optimal solution must be in the set. Consequently, we use our algorithm for the center-coverage problem as a decision procedure to find the optimal covering range in the set.

We should point out that although we have assumed $\sum_{j=1}^{m_i} f_{ij} = 1$ for each $P_i \in \mathcal{P}$, it is quite straightforward to adapt our algorithm to the general case where the assumption does not hold.

The rest of the chapter is organized as follows. We introduce some notation in Section 6.2. In Section 6.3, we describe our algorithmic scheme for the center-coverage problem but leave the implementation details in the subsequent two sections. Specifically, the algorithm for computing all medians p_i^* is given in Section 6.4, and in the same section we also propose a connector-bounded centroid decomposition of T, which is repeatedly used in the paper and may be interesting in its own right. The data structures used in our algorithmic scheme are given in Section 6.5. We finally solve the k-center problem in Section 6.6.

6.2 Preliminaries

Note that the locations of the uncertain points of $\mathcal{P}$ may be in the interior of the edges of T. A *vertex-constrained case* happens if all locations of $\mathcal{P}$ are at vertices of T and each vertex of T holds at least one location of $\mathcal{P}$ (but the centers we seek can still be in the interior of edges). As in Chapter 3, we will show later in Section 6.5.4 that the general problem can be reduced to the vertex-constrained case in $O(|T| + M)$ time. In the following paper, unless otherwise stated, we focus our discussion on the vertex-constrained case and assume our problem on $\mathcal{P}$ and T is a vertex-constrained case. For ease of exposition, we further make a general position assumption that every vertex of T has only one location of $\mathcal{P}$ (we explain in Section 6.5.4 that our algorithm easily extends to the degenerate case). Under this assumption, it holds that $|T| = M \geq n$.

Let $e(u, v)$ denote the edge of T incident to two vertices u and v. For any two points p and q on T, denote by $\pi(p, q)$ the simple path from p to q on T.

Let π be any simple path on T and x be any point on π. For any location p_{ij} of an uncertain point P_i, the distance $d(x, p_{ij})$ is a convex (and piecewise linear) function as x changes on π [27]. As a sum of multiple convex functions, $\mathsf{Ed}(x, P_i)$ is also convex (and piecewise linear) on π, that is, in general, as x moves on π, $\mathsf{Ed}(x, P_i)$ first monotonically decreases and then monotonically increases. In particular, for each edge e of T, $\mathsf{Ed}(x, P_i)$ is a linear function for $x \in e$.

For any subtree T' of T and any $P_i \in \mathcal{P}$, we call the sum of the probabilities of the locations of P_i in T' the *probability sum* of P_i in T'.

For each uncertain point P_i, let p_i^* be a point $x \in T$ that minimizes $\mathsf{Ed}(x, P_i)$. If we consider $w_i \cdot f_{ij}$ as the weight of p_{ij}, p_i^* is actually the *weighted median* of all points $p_{ij} \in P_i$. We call p_i^* the *median* of P_i. Although p_i^* may not be unique (e.g., when there is an edge e dividing T into two subtrees such that the probability sum of P_i in either subtree is exactly 0.5), P_i always has a median located at a vertex v of T, and we let p_i^* refer to such a vertex.

Recall that λ is the given covering range for the center-coverage problem. If $\mathsf{Ed}(p_i^*, P_i) > \lambda$ for some $i \in [1, n]$, then there is no solution for the problem since no point of T can cover P_i. Henceforth, we assume $\mathsf{Ed}(p_i^*, P_i) \leq \lambda$ for each $i \in [1, n]$.

6.3 The Algorithmic Scheme

In this section, we describe our algorithmic scheme for the center-coverage problem, and the implementation details will be presented in the subsequent two sections.

We start with computing the medians p_i^* of all uncertain points of $\mathcal{P}$. We have the following lemma, whose proof is deferred to Section 6.4.2.

Lemma 6.3.1. *The medians p_i^* of all uncertain points P_i of $\mathcal{P}$ can be computed in $O(M \log M)$ time.*

6.3.1 The Medians-Spanning Tree T_m

Denote by P^* the set of all medians p_i^*. Let T_m be the minimum connected subtree of T that spans/connects all medians. Note that each leaf of T_m must hold a median. We pick an arbitrary median as the root of T, denoted by r. The subtree T_m can be easily computed in $O(M)$ time by a post-order traversal on T (with respect to the root

r), and we omit the details. The following lemma is based on the fact that $\mathsf{Ed}(x, P_i)$ is convex for x on any simple path of T and $\mathsf{Ed}(x, P_i)$ minimizes at $x = p_i^*$.

Lemma 6.3.2. *There exists an optimal solution for the center-coverage problem in which every center is on T_m.*

Proof. Consider an optimal solution and let C be the set of all centers in it. Assume there is a center $c \in C$ that is not on T_m. Let v be the vertex of T_m that holds a median and is closest to c. Then v decompose T into two subtrees T_1 and T_2 with the only common vertex v such that c is in one subtree, say T_1, and all medians are in T_2. If we move a point x from c to v along $\pi(c, v)$, then $\mathsf{Ed}(x, P_i)$ is non-increasing for each $i \in [1, n]$. This implies that if we move the center c to v, we can obtain an optimal solution in which c is in T_m.

If C has other centers that are not on T_m, we do the same as above to obtain an optimal solution in which all centers are on T_m. The lemma thus follows. □

Due to Lemma 6.3.2, we will focus on finding centers on T_m. We also consider r as the root of T_m. With respect to r, we can talk about ancestors and descendants of the vertices in T_m. Note that for any two vertices u and v of T_m, $\pi(u, v)$ is in T_m.

We reindex all medians and the corresponding uncertain points so that the new indices will facilitate our algorithm, as follows. Starting from an arbitrary child of r in T_m, we traverse down the tree T_m by always following the leftmost child of the current node until we encounter a leaf, denoted by v^*. Starting from v^*, we perform a post-order traversal on T_m and reindex all medians of P^* such that $p_1^*, p_2^*, \ldots, p_n^*$ is the list of points of P^* encountered in order in the above traversal. Recall that the root r contains a median, which is p_n^* after the reindexing. Accordingly, we also reindex all uncertain points of $\mathcal{P}$ and their corresponding locations on T, which can be done in $O(M)$ time. In the following paper, we will always use the new indices.

For each vertex v of T_m, we use $T_m(v)$ to represent the subtree of T_m rooted at v. The reason we do the above reindexing is that for any vertex v of T_m, the new indices of all medians in $T_m(v)$ must form a range $[i, j]$ for some $1 \leq i \leq j \leq n$, and we use $R(v)$ to denote the range. It will be clear later that this property will facilitate our algorithm.

6.3.2 The Algorithm

Our algorithm for the center-coverage problem works as follows. Initially, all uncertain points are "active". During the algorithm, we will place centers on T_m, and once an uncertain point P_i is covered by a center, we will "deactivate" it (it then becomes "inactive"). The algorithm visits all vertices of T_m following the above post-order traversal of T_m starting from leaf v^*. Suppose v is currently being visited. Unless v is the root r, let u be the parent of v. Below we describe our algorithm for processing v. There are two cases depending on whether v is a leaf or an internal node, although the algorithm for them is essentially the same.

The Leaf Case

If v is a leaf, then we process it as follows. Since v is leaf, it holds a median p_i^*. If P_i is inactive, we do nothing; otherwise, we proceed as follows.

We compute a point c (called a *candidate center*) on the path $\pi(v, r)$ closest to r such that $\mathsf{Ed}(c, P_i) \leq \lambda$. Note that if we move a point x from v to r along $\pi(v, r)$, $\mathsf{Ed}(x, P_i)$ is monotonically increasing. By the definition of c, if $\mathsf{Ed}(r, P_i) \leq \lambda$, then $c = r$; otherwise, $\mathsf{Ed}(c, P_i) = \lambda$. If c is in $\pi(u, r)$, then we do nothing and finish processing v. Below we assume that c is not in $\pi(u, r)$ and thus is in $e(u, v) \setminus \{u\}$ (i.e., $c \in e(u, v)$ but $c \neq u$).

In order to cover P_i, by the definition of c, we must place a center in $e(u, v) \setminus \{u\}$. Our strategy is to place a center at c. Indeed, this is the best location for placing a center since it is the location that can cover P_i and is closest to u (and thus is closest to every other active uncertain point). We use a *candidate-center-query* to compute c in $O(\log n)$ time, whose details will be discussed later. Next, we report all active uncertain points that can be covered by c, and this is done by a *coverage-report-query* in output-sensitive $O(\log M \log n + k \log n)$ amortized time, where k is the number of uncertain points covered by c. The details for the operation will be discussed later. Further, we deactivate all these uncertain points. We will show that deactivating each uncertain point P_j can be done in $O(m_j \log M \log n)$ amortized time. This finishes processing v.

The Internal Node Case

If v is an internal node, since we process the vertices of T_m following a post-order traversal, all descendants of v have already been processed. Our algorithm maintains an invariant that if the subtree $T_m(v)$ contains any active median p_i^* (i.e., P_i is active), then $\mathsf{Ed}(v, P_i) \leq \lambda$. When v is a leaf, this invariant trivially holds. Our way of processing a leaf discussed above also maintains this invariant.

To process v, we first check whether $T_m(v)$ has any active medians. This is done by a *range-status-query* in $O(\log n)$ time, whose details will be given later. If $T_m(v)$ does not have any active median, then we are done with processing v. Otherwise, by the algorithm invariant, for each active median p_i^* in $T_m(v)$, it holds that $\mathsf{Ed}(v, P_i) \leq \lambda$. If $v = r$, we place a center at v and finish the entire algorithm. Below, we assume v is not r and thus u is the parent of v.

We compute a point c on $\pi(v, r)$ closest to r such that $\mathsf{Ed}(c, P_i) \leq \lambda$ for all active medians $p_i^* \in T_m(v)$, and we call c the *candidate center*. By the definition of c, if $\mathsf{Ed}(r, P_i) \leq \lambda$ for all active medians $p_i^* \in T_m(v)$, then $c = r$; otherwise, $\mathsf{Ed}(c, P_i) = \lambda$ for some active median $p_i^* \in T_m(v)$. As in the leaf case, finding c is done in $O(\log n)$ time by a *candidate-center-query*. If c is on $\pi(u, r)$, then we finish processing v. Note that this implies $\mathsf{Ed}(u, P_i) \leq \lambda$ for each active median $p_i^* \in T_m(v)$, which maintains the algorithm invariant for u.

If $c \notin \pi(u, r)$, then $c \in e(u, v) \backslash \{u\}$. In this case, by the definition of c, we must place a center in $e(u, v) \setminus \{u\}$ to cover P_i. As discussed in the leaf case, the best location for placing a center is c and thus we place a center at c. Then, by using a coverage-report-query, we find all active uncertain points covered by c and deactivate them. Note that by the definition of c, c covers P_j for all medians $p_j^* \in T_m(v)$. This finishes processing v.

Once the root r is processed, the algorithm finishes.

6.3.3 The Time Complexity

To analyze the running time of the algorithm, it remains to discuss the three operations: range-status-queries, coverage-report-queries, and candidate-center-queries. For answering range-status-queries, it is trivial, as shown in Lemma 6.3.3.

Lemma 6.3.3. *We can build a data structure in $O(M)$ time that can answer each range-status-query in $O(\log n)$ time. Further, once an uncertain point is deactivated, we can remove it from the data structure in $O(\log n)$ time.*

Proof. Initially we build a balanced binary search tree Φ to maintain all indices $1, 2, \ldots, n$. If an uncertain point P_i is deactivated, then we simply remove i from the tree in $O(\log n)$ time.

For each range-status-query, we are given a vertex v of T_m, and the goal is to decide whether $T_m(v)$ has any active medians. Recall that all medians in $T_m(v)$ form a range $R(v) = [i, j]$. As preprocessing, we compute $R(v)$ for all vertices v of T_m, which can be done in $O(|T_m|)$ time by the post-order traversal of T_m starting from leaf v^*. Note that $|T_m| = O(M)$.

During the query, we simply check whether Φ still contains any index in the range $R(v) = [i, j]$, which can be done in $O(\log n)$ time by standard approaches (e.g., by finding the successor of i in Φ). □

For answering the coverage-report-queries and the candidate-center-queries, we have the following two lemmas. Their proofs are deferred to Section 6.5.

Lemma 6.3.4. *We can build a data structure $\mathcal{A}_1$ in $O(M \log^2 M)$ time that can answer in $O(\log M \log n + k \log n)$ amortized time each coverage-report-query, i.e., given any point $x \in T$, report all active uncertain points covered by x, where k is the output size. Further, if an uncertain point P_i is deactivated, we can remove P_i from $\mathcal{A}_1$ in $O(m_i \cdot \log M \cdot \log n)$ amortized time.*

Lemma 6.3.5. *We can build a data structure $\mathcal{A}_2$ in $O(M \log M + n \log^2 M)$ time that can answer in $O(\log n)$ time each candidate-center-query, i.e., given any vertex $v \in T_m$, find the candidate center c for the active medians of $T_m(v)$. Further, if an uncertain point P_i is deactivated, we can remove P_i from $\mathcal{A}_2$ in $O(\log n)$ time.*

Using these results, we obtain the following.

Theorem 6.3.6. *We can find a minimum number of centers on T to cover all uncertain points of $\mathcal{P}$ in $O(M \log^2 M)$ time.*

Proof. First of all, the total preprocessing time of Lemmas 6.3.3, 6.3.4, and 6.3.5 is $O(M \log^2 M)$. Computing all medians takes $O(M \log M)$ time by Lemma 6.3.1. Below we analyze the total time for computing centers on T_m.

The algorithm processes each vertex of T_m exactly once. The processing of each vertex calls each of the following three operations at most once: coverage-report-queries, range-status-queries, and candidate-center-queries. Since each of the last two operations runs in $O(\log n)$ time, the total time of these two operations in the entire algorithm is $O(M \log n)$. For the coverage-report-queries, each operation runs in $O(\log M \log n + k \log n)$ amortized time. Once an uncertain point P_i is reported by it, P_i will be deactivated by removing it from all three data structures (i.e., those in Lemmas 6.3.3, 6.3.4, and 6.3.5) and P_i will not become active again. Therefore, each uncertain point will be reported by the coverage-report-query operations at most once. Hence, the total sum of the value k in the entire algorithm is n. Further, notices that there are at most n centers placed by the algorithm. Hence, there are at most n coverage-report-query operations in the algorithm. Therefore, the total time of the coverage-report-queries in the entire algorithm is $O(n \log M \log n)$. In addition, since each uncertain point P_i will be deactivated at most once, the total time of the remove operations for all three data structures in the entire algorithm is $O(M \log M \log n)$ time.

The theorem thus follows. □

In addition, Lemma 6.3.7 will be used to build the data structure $\mathcal{A}_2$ in Lemma 6.3.5, and it will also help to solve the k-center problem in Section 6.6. Its proof is given in Section 6.5.

Lemma 6.3.7. *We can build a data structure $\mathcal{A}_3$ in $O(M \log M)$ time that can compute the expected distance $\mathsf{Ed}(x, P_i)$ in $O(\log M)$ time for any point $x \in T$ and any uncertain point $P_i \in \mathcal{P}$.*

6.4 A Tree Decomposition and Computing the Medians

In this section, we first introduce a decomposition of T, which will be repeatedly used in our algorithms (e.g., for Lemmas 6.3.1, 6.3.4, 6.3.7). Later in Section 6.4.2 we will compute the medians with the help of the decomposition.

6.4.1 A Connector-Bounded Centroid Decomposition

We propose a tree decomposition of T, called a *connector-bounded centroid decomposition*, which is different from the centroid decompositions used before, e.g., [31,33,72,80] and has certain properties that can facilitate our algorithms.

A vertex v of T is called a *centroid* if T can be represented as a union of two subtrees with v as their only common vertex and each subtree has at most $\frac{2}{3}$ of the vertices of T [33, 72], and we say the two subtrees are *decomposed* by v. Such a centroid always exists and can be found in linear time [33,72]. For convenience of discussion, we consider v to be contained in only one subtree but an "open vertex" in the other subtree (thus, the location of $\mathcal{P}$ at v only belongs to one subtree).

Our decomposition of T corresponds to a *decomposition tree*, denoted by Υ and defined recursively as follows. Each internal node of Υ has two, three, or four children. The root of Υ corresponds to the entire tree T. Let v be the centroid of T, and let T_1 and T_2 be the subtrees of T decomposed by v. Note that T_1 and T_2 are disjoint since we consider v to be contained in only one of them. Further, we call v a *connector* in both T_1 and T_2. Correspondingly, in Υ, its root has two children corresponding to T_1 and T_2, respectively.

In general, consider a node μ of Υ. Let $T(\mu)$ represent the subtree of T corresponding to μ. We assume $T(\mu)$ has at most two connectors (initially this is true when μ is the root). We further decompose $T(\mu)$ into subtrees that correspond to the children of μ in Υ, as follows. Let v be the centroid of $T(\mu)$ and let $T_1(\mu)$ and $T_2(\mu)$ respectively be the two subtrees of $T(\mu)$ decomposed by v. We consider v as a *connector* in both $T_1(\mu)$ and $T_2(\mu)$.

If $T(\mu)$ has at most one connector, then each of $T_1(\mu)$ and $T_2(\mu)$ has at most two connectors. In this case, μ has two children corresponding to $T_1(\mu)$ and $T_2(\mu)$, respectively.

If $T(\mu)$ has two connectors but each of $T_1(\mu)$ and $T_2(\mu)$ still has at most two connectors (with v as a new connector), then μ has two children corresponding to $T_1(\mu)$ and $T_2(\mu)$, respectively. Otherwise, one of them, say, $T_2(\mu)$, has three connectors and the other $T_1(\mu)$ has only one connector (e.g., see Fig. 6.1). In this case, μ has a child in Υ corresponding to $T_1(\mu)$, and we further perform a *connector-reducing decomposition*

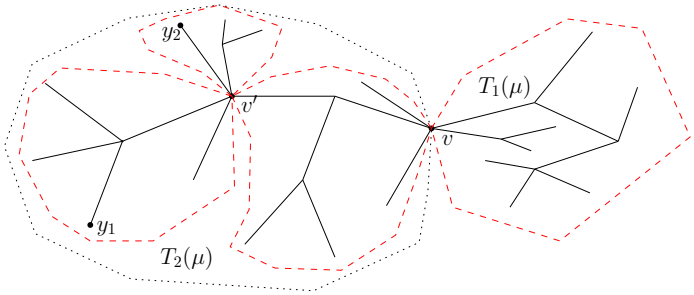

Figure 6.1. Illustrating the decomposition of $T(\mu)$ into four subtrees enclosed by the (red) dashed cycles. y_1 and y_2 are two connectors of $T(\mu)$; $T(\mu)$ is first decomposed into two subtrees $T_1(\mu)$ and $T_2(\mu)$. However, since $T_2(\mu)$ has three connectors, we further decompose it into three subtrees each of which has at most two connectors.

on $T_2(\mu)$, as follows (this is the main difference between our decomposition and the traditional centroid decomposition used before [31, 33, 72, 80]). Depending on whether the three connectors of $T_2(\mu)$ are in a simple path, there are two cases.

1. If they are in a simple path, without loss of generality, we assume v is the one between the other two connectors in the path. We decompose $T_2(\mu)$ into two subtrees at v such that they contain the two connectors respectively. In this way, each subtree contains at most two connectors. Correspondingly, μ has another two children corresponding the two subtrees of $T_2(\mu)$, and thus μ has three children in total.

2. Otherwise, there is a unique vertex v' in $T_2(\mu)$ that decomposes $T_2(\mu)$ into three subtrees that contain the three connectors respectively (e.g., see Fig. 6.1). Note that v' and the three subtrees can be easily found in linear time by traversing $T_2(\mu)$. Correspondingly, μ has another three children corresponding to the above three subtrees of $T_2(\mu)$, respectively, and thus μ has four children in total. Note that we consider v' as a connector in each of the above three subtrees. Thus, each subtree contains at most two connectors.

We continue the decomposition until each subtree $T(\mu)$ of $\mu \in \Upsilon$ becomes an edge $e(v_1, v_2)$ of T. According to our decomposition, both v_1 and v_2 are connectors of $T(\mu)$, but they may only open vertices of $T(\mu)$. If both v_1 and v_2 are open vertices of $T(\mu)$, then we will not further decompose $T(\mu)$, so μ is a leaf of Υ. Otherwise, we further

decompose $T(\mu)$ into an open edge and a closed vertex v_i if v_i is contained in $T(\mu)$ for each $i = 1, 2$. Correspondingly, μ has either two or three children that are leaves of Υ. In this way, for each leaf μ of Υ, $T(\mu)$ is either an open edge or a closed vertex of T. In the former case, $T(\mu)$ has two connectors that are its incident vertices, and in the latter case, $T(\mu)$ has one connector that is itself.

This finishes the decomposition of T. A major difference between our decomposition and the traditional centroid decomposition [31, 33, 72, 80] is that the subtree in our decomposition has at most two connectors. As will be clear later, this property is crucial to guarantee the runtime of our algorithms.

Lemma 6.4.1. *The height of Υ is $O(\log M)$ and Υ has $O(M)$ nodes. The connector-bounded centroid decomposition of T can be computed in $O(M \log M)$ time.*

Proof. Consider any node μ of Υ. Let $T(\mu)$ be the subtree of T corresponding to μ. According to our decomposition, $|T(\mu)| = O(M \cdot (\frac{2}{3})^t)$, where t is the depth of μ in Υ. This implies that the height of Υ is $O(\log M)$.

Since each leaf of Υ corresponds to either a vertex or an open edge of T, the number of leaves of Υ is $O(M)$. Since each internal node of Υ has at least two children, the number of internal nodes is no more than the number of leaves. Hence, Υ has $O(M)$ nodes.

According to our decomposition, all subtrees of T corresponding to all nodes in the same level of Υ (i.e., all nodes with the same depth) are pairwise disjoint, and thus the total size of all these subtrees is $O(M)$. Decomposing each subtree can be done in linear time (e.g., finding a centroid takes linear time). Therefore, decomposing all subtrees in each level of Υ takes $O(M)$ time. As the height of Υ is $O(\log M)$, the total time for computing the decomposition of T is $O(M \log M)$. □

In the following, we assume our decomposition of T and the decomposition tree Υ have been computed. In addition, we introduce some notation that will be used later. For each node μ of Υ, we use $T(\mu)$ to represent the subtree of T corresponding to μ. If y is a connector of $T(\mu)$, then we use $T(y, \mu)$ to represent the subtree of T consisting of all points q of $T \setminus T(\mu)$ such that $\pi(q, p)$ contains y for any point $p \in T(\mu)$ (i.e., $T(y, \mu)$ is the "outside world" connecting to $T(\mu)$ through y; e.g., see Fig. 6.2). By this definition,

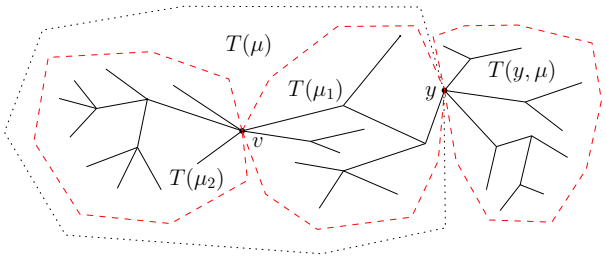

Figure 6.2. Illustrating the subtrees $T(\mu_1), T(\mu_2)$, and $T(y,\mu)$, where y is a connector of $T(\mu) = T(\mu_1) \cup T(\mu_2)$. Note that $T(y,\mu)$ is also $T(y,\mu_1)$ as $y \in T(\mu_1)$.

if y is the only connector of $T(\mu)$, then $T = T(\mu) \cup T(y,\mu)$; if $T(\mu)$ has two connectors y_1 and y_2, then $T = T(\mu) \cup T(y_1,\mu) \cup T(y_2,\mu)$.

6.4.2 Computing the Medians

In this section, we compute all medians. It is easy to compute the median p_i^* for a single uncertain point P_i in $O(M)$ time by traversing the tree T. Hence, a straightforward algorithm can compute all n medians in $O(nM)$ time. Instead, we present an $O(M \log M)$ time algorithm, which will prove Lemma 6.3.1.

For any vertex v (e.g., the centroid) of T, let T_1 and T_2 be two subtrees of T decomposed by v (i.e., v is their only common vertex and $T = T_1 \cup T_2$), such that v is contained in only one subtree and is an open vertex in the other. The following lemma can be readily obtained from Kariv and Hakimi [55], and similar results were also given in Chapter 3.

Lemma 6.4.2. *For any uncertain point P_i of $\mathcal{P}$, we have the following.*

1. *If the probability sum of P_i in T_j is greater than 0.5 for some $j \in \{1,2\}$, then the median p_i^* must be in T_j.*

2. *The vertex v is p_i^* if the probability sum of P_i in T_j is equal to 0.5 for some $j \in \{1,2\}$.*

Consider the connector-bounded centroid decomposition Υ of T. Starting from the root of Υ, our algorithm will process the nodes of Υ in a top-down manner. Suppose we are processing a node μ. Then, we maintain a sorted list of indices for μ, called the *index list* of μ and denoted by $L(\mu)$, which consists of all indices $i \in [1,n]$ such

that p_i^* is not found yet but is known to be in the subtree $T(\mu)$. Since each index i of $L(\mu)$ essentially refers to P_i, for convenience, we also say that $L(\mu)$ is a set of uncertain points. Let $F[1 \cdots n]$ be an array, which will help to compute the probability sums in our algorithm.

The Root Case

Initially, μ is the root and we process it as follows. We present our algorithm in a way that is consistent with that for the general case.

Since μ is the root, we have $T(\mu) = T$ and $L(\mu) = \{1, 2, \ldots, n\}$. Let μ_1 and μ_2 be the two children of μ. Let v be the centroid of T that is used to decompose $T(\mu)$ into $T(\mu_1)$ and $T(\mu_2)$ (e.g., see Fig. 6.2). We compute in $O(|T(\mu)|)$ time the probability sums of all uncertain points of $L(\mu)$ in $T(\mu_1)$ by using the array F and traversing $T(\mu_1)$. Specifically, we first perform a *reset procedure* on F to reset $F[i]$ to 0 for each $i \in L(\mu)$, by scanning the list $L(\mu)$. Then, we traverse $T(\mu_1)$, and for each visited vertex, which holds some uncertain point location p_{ij}, we update $F[i] = F[i] + f_{ij}$. After the traversal, for each $i \in L(\mu)$, $F[i]$ is equal to the probability sum of P_i in $T(\mu_1)$. By Lemma 6.4.2, if $F[i] = 0.5$, then p_i^* is v and we report $p_i^* = v$; if $F[i] > 0.5$, then p_i^* is in $T_1(\mu)$ and we add i to the end of the index list $L(\mu_1)$ for μ_1 (initially $L(\mu_1) = \emptyset$); if $F[i] < 0.5$, then p_i^* is in $T_2(\mu)$ and add i to the end of $L(\mu_2)$ for μ_2. The above has correctly computed the index lists for μ_1 and μ_2.

Recall that v is a connector in both $T(\mu_1)$ and $T(\mu_2)$. In order to efficiently compute medians in $T(\mu_1)$ and $T(\mu_2)$ recursively, we compute a *probability list* $L(v, \mu_j)$ at v for μ_j for each $j = 1, 2$. We discuss $L(v, \mu_1)$ first.

The list $L(v, \mu_1)$ is the same as $L(\mu_1)$ except that each index $i \in L(v, \mu_1)$ is also associated with a value, denoted by $F(i, v, \mu_1)$, which is the probability sum of P_i in $T(v, \mu_1)$ (recall the definition of $T(v, \mu_1)$ at the end of Section 6.4.1; note that $T(v, \mu_1) = T(\mu_2)$ in this case). The list $L(v, \mu_1)$ can be built in $O(|T(\mu)|)$ time by traversing $T(\mu_2)$ and using the array F. Specifically, we scan the list $L(\mu_1)$, and for each index $i \in L(\mu_1)$, we reset $F[i] = 0$. Then, we traverse the subtree $T(\mu_2)$, and for each location p_{ij} in $T(\mu_2)$, we update $F[i] = F[i] + f_{ij}$ (if i is not in $L(\mu_1)$, this step is actually redundant but does not affect anything). After the traversal, for each index $i \in L(\mu_1)$, we copy it

to $L(v, \mu_1)$ and set $F(i, v, \mu_1) = F[i]$.

Similarly, we compute the probability list $L(v, \mu_2)$ at v for μ_2 in $O(|T(\mu)|)$ time by traversing $T(\mu_1)$. This finishes the processing of the root μ. The total time is $O(|T(\mu)|)$ since $|L(\mu)| \leq |T(\mu)|$. Note that our algorithm guarantees that for each $i \in L(\mu_1)$, P_i must have at least one location in $T(\mu_1)$, and thus $|L(\mu_1)| \leq |T(\mu_1)|$. Similarly, for each $i \in L(\mu_2)$, P_i must have at least one location in $T(\mu_2)$, and thus $|L(\mu_2)| \leq |T(\mu_2)|$.

The General Case

Let μ be an internal node of Υ such that the ancestors of μ have all been processed. Hence, we have a sorted index list $L(\mu)$. If $L(\mu) = \emptyset$, then we do not need to process μ and any of its descendants. We assume $L(\mu) \neq \emptyset$. Thus, for each $i \in L(\mu)$, p_i^* is in $T(\mu)$ and P_i has at least one location in $T(\mu)$ (and thus $|L(\mu)| \leq |T(\mu)|$). Further, for each connector y of $T(\mu)$, the algorithm maintains a probability list $L(y, \mu)$ that is the same as $L(\mu)$ except that each index $i \in L(y, \mu)$ is associated with a value $F(i, y, \mu)$, which is the probability sum of P_i in the subtree $T(y, \mu)$. Our processing algorithm for μ works as follows, whose total time is $O(|T(\mu)|)$.

According to our decomposition, $T(\mu)$ has at most two connectors and μ may have two, three, or four children. We first discuss the case where μ has two children, and other cases can be handled similarly.

Let μ_1 and μ_2 be the two children of μ, respectively. Let v be the centroid of $T(\mu)$ that is used to decompose it. We discuss the subtree $T(\mu_1)$ first, and $T(\mu_2)$ is similar. Since v is a connector of $T(\mu_1)$ and $T(\mu_1)$ has at most two connectors, $T(\mu_1)$ has at most one connector y other than v. We consider the general situation where $T(\mu_1)$ has such a connector y (the case where such a connector does not exist can be handled similarly but in a simpler way). Note that y must be a connector of $T(\mu)$.

We first compute the probability sums of P_i's for all $i \in L(\mu)$ in the subtree $T(\mu_1) \cup T(y, \mu)$ (e.g., see Fig. 6.2), which can be done in $O(|T(\mu)|)$ time by traversing $T(\mu_1)$ and using the array F and the probability list $L(y, \mu)$ at y, as follows. We scan the list $L(\mu)$ and for each index $i \in L(\mu)$, we reset $F[i] = 0$. Then, we traverse $T(\mu_1)$ and for each location p_{ij}, we update $F[i] = F[i] + f_{ij}$ (it does not matter if $i \notin L(\mu)$). When the traversal visits y, we scan the list $L(y, \mu)$ and for each index $i \in L(y, \mu)$, we update

$F[i] = F[i] + F(i, y, \mu)$. After the traversal, for each $i \in L(\mu)$, $F[i]$ is the probability sum of P_i in $T(\mu_1) \cup T(y, \mu)$. For each $i \in L(\mu)$, if $F[i] = 0.5$, we report $p_i^* = v$; if $F[i] > 0.5$, we add i to $L(\mu_1)$; if $F[i] < 0.5$, we add i to $L(\mu_2)$. This builds the two lists $L(\mu_1)$ and $L(\mu_2)$, which are initially $\emptyset$. Note that since for each $i \in L(\mu)$, P_i has at least one location in $T(\mu)$, the above way of computing $L(\mu_1)$ (resp., $L(\mu_2)$) guarantees that for each i in $L(\mu_1)$ (resp., $L(\mu_2)$), P_i has at least one location in $T(\mu_1)$ (resp., $T(\mu_2)$), which implies $|L(\mu_1)| \leq |T(\mu_1)|$ (resp., $|L(\mu_2)| \leq |T(\mu_2)|$).

Next we compute the probability lists for the connectors of $T(\mu_1)$. Note that $T(\mu_1)$ has two connectors v and y. For v, we compute the probability list $L(v, \mu_1)$ that is the same as $L(\mu_1)$ except that each $i \in L(v, \mu_1)$ is associated with a value $F(i, v, \mu_1)$, which is the probability sum of P_i in the subtree $T(v, \mu_1)$. To compute $L(v, \mu_1)$, we first reset $F[i] = 0$ for each $i \in L(\mu_1)$. Then we traverse $T(\mu_2)$ and for each location $p_{ij} \in T(\mu_2)$, we update $F[i] = F[i] + f_{ij}$. If $T(\mu_2)$ has a connector y' other than v, then y' is also a connector of $T(\mu)$ (note that there is at most one such connector); we scan the probability list $L(y', \mu)$ and for each $i \in L(y', \mu)$, we update $F[i] = F[i] + F(i, y', \mu)$. Finally, we scan $L(\mu_1)$ and for each $i \in L(\mu_1)$, we copy it to $L(v, \mu_1)$ and set $F(i, v, \mu_1) = F[i]$. This computes the probability list $L(v, \mu_1)$.

Further, we also need to compute the probability list $L(y, \mu_1)$ at y for $T(\mu_1)$. The list $L(y, \mu_1)$ is the same as $L(\mu_1)$ except that each $i \in L(y, \mu_1)$ also has a value $F(i, y, \mu_1)$, which is the probability sum of P_i in $T(y, \mu_1)$. To compute $L(y, \mu_1)$, we first copy all indices of $L(\mu_1)$ to $L(y, \mu_1)$, and then compute the values $F(i, y, \mu_1)$, as follows. Note that $T(y, \mu_1)$ is exactly $T(y, \mu)$ (e.g., see Fig. 6.2). Recall that as a connector of $T(\mu)$, y has a probability list $L(y, \mu)$ in which each $i \in L(y, \mu)$ has a value $F(i, y, \mu)$. Notice that $L(y, \mu_1) \subseteq L(y, \mu)$. Due to $T(y, \mu_1) = T(y, \mu)$, for each $i \in L(y, \mu_1)$, $F(i, y, \mu_1)$ is equal to $F(i, y, \mu)$. Since indices in each of $L(y, \mu_1)$ and $L(y, \mu)$ are sorted, we scan $L(y, \mu_1)$ and $L(y, \mu)$ simultaneously (like merging two sorted lists) and for each $i \in L(y, \mu_1)$, if we encounter i in $L(y, \mu)$, then we set $F(i, y, \mu_1) = F(i, y, \mu)$. This computes the probability list $L(y, \mu_1)$ at y for $T(\mu_1)$.

The above has processed the subtree $T(\mu_1)$. Using the similar approach, we can process $T(\mu_2)$ and we omit the details.

This finishes the processing of μ for the case where μ has two children. The total

time is $O(|T(\mu)|)$. To see this, the algorithm traverses $T(\mu)$ for a constant number of times. The algorithm also visits the list $L(\mu)$ and the probability list of each connector of $T(\mu)$ for a constant number of times. Recall that $|L(\mu)| \leq |T(\mu)|$ and $|L(\mu)| = |L(\mu, y)|$ for each connector y of $T(\mu)$. Also recall that $T(\mu)$ has at most two connectors. Thus, the total time for processing μ is $O(|T(\mu)|)$.

Remark. If the number of connectors of $T(\mu)$ were not bounded by a constant, then we could not bound the processing time for μ as above. This is one reason our decomposition on T requires each subtree $T(\mu)$ to have at most two connectors.

If μ has three children, μ_1, μ_2, μ_3, then $T(\mu)$ is decomposed into three subtrees $T(\mu_j)$ for $j = 1, 2, 3$. In this case, $T(\mu)$ has two connectors. To process μ, we apply the above algorithm for the two-children case twice. Specifically, we consider the procedure of decomposing $T(\mu)$ into three subtrees consisting of two "intermediate decomposition steps". According to our decomposition, $T(\mu)$ was first decomposed into two subtrees by its centroid such that one subtree $T_1(\mu)$ contains at most two connectors while the other one $T_2(\mu)$ contains three connectors, and we consider this as the first intermediate step. The second intermediate step is to further decompose $T_2(\mu)$ into two subtrees each of which contains at most two connectors. To process μ, we apply our two-children case algorithm on the first intermediate step and then on the second intermediate step. The total time is still $O(|T(\mu)|)$. We omit the details.

Similarly, if μ has four children, then the decomposition can be considered as consisting of three intermediate steps (e.g., in Fig. 6.2, the first step is to decompose $T(\mu)$ into $T_1(\mu)$ and $T_2(\mu)$, and then decomposing $T_2(\mu)$ into three subtrees can be considered as consisting of two steps each of which decomposes a subtree into two subtrees), and we apply our two-children case algorithm three times. The total processing time for μ is also $O(|T(\mu)|)$.

The above describes the algorithm for processing μ when μ is an internal node of Υ.

If μ is a leaf, then $T(\mu)$ is either a vertex or an open edge of T. If $T(\mu)$ is an open edge, the index list $L(\mu)$ must be empty since our algorithm only finds medians on vertices. Otherwise, $T(\mu)$ is a vertex v of T. If $L(\mu)$ is not empty, then for each

$i \in L(\mu)$, we simply report $p_i^* = v$.

The running time of the entire algorithm is $O(M \log M)$. To see this, processing each node μ of Υ takes $O(|T(\mu)|)$ time. For each level of Υ, the total sum of $|T(\mu)|$ of all nodes μ in the level is $O(|T|)$. Since the height of Υ is $O(\log M)$, the total time of the algorithm is $O(M \log M)$. This proves Lemma 6.3.1.

6.5 The Data Structures $\mathcal{A}_1$, $\mathcal{A}_2$, and $\mathcal{A}_3$

In this section, we present the three data structures $\mathcal{A}_1$, $\mathcal{A}_2$, and $\mathcal{A}_3$, for Lemmas 6.3.4, 6.3.5, and 6.3.7, respectively. In particular, $\mathcal{A}_3$ will be used to build $\mathcal{A}_2$ and it will also be needed for solving the k-center problem in Section 6.6. Our connector-bounded centroid decomposition Υ will play an important role in constructing both $\mathcal{A}_1$ and $\mathcal{A}_3$. In the following, we present them in the order of $\mathcal{A}_1, \mathcal{A}_3$, and $\mathcal{A}_2$.

6.5.1 The Data Structure $\mathcal{A}_1$

The data structure $\mathcal{A}_1$ is for answering the coverage-report-queries, i.e., given any point $x \in T$, find all active uncertain points that are covered by x. Further, it also supports the operation of removing an uncertain point once it is deactivated.

Consider any node $\mu \in \Upsilon$. If μ is the root, let $L(\mu) = \emptyset$; otherwise, define $L(\mu)$ to be the sorted list of all such indices $i \in [1, n]$ that P_i does not have any locations in the subtree $T(\mu)$ but has at least one location in $T(\mu')$, where μ' is the parent of μ. Let y be any connector of $T(\mu)$. Let $L(y, \mu)$ be an index list the same as $L(\mu)$ and each index $i \in L(y, \mu)$ is associated with two values: $F(i, y, \mu)$, which is the probability sum of P_i in the subtree $T(y, \mu)$, and $D(i, y, \mu)$, which is the expected distance from y to the locations of P_{ij} in $T(y, \mu)$, i.e., $D(i, y, \mu) = w_i \cdot \sum_{p_{ij} \in T(y,\mu)} f_{ij} \cdot d(p_{ij}, y)$. We refer to $L(\mu)$ and $L(y, \mu)$ for each connector $y \in T(\mu)$ the *information lists* of μ.

Lemma 6.5.1. *Suppose $L(\mu) \neq \emptyset$ and the information lists of μ are available. Let t_μ be the number of indices in $L(\mu)$. Then, we can build a data structure of $O(t_\mu)$ size in $O(|T(\mu)| + t_\mu \log t_\mu)$ time on $T(\mu)$, such that given any point $x \in T(\mu)$, we can report all indices i of $L(\mu)$ such that P_i is covered by x in $O(\log n + k \log n)$ amortized time, where k is the output size; further, if P_i is deactivated with $i \in L(\mu)$, then we can remove i from the data structure and all information lists of μ in $O(\log n)$ amortized time.*

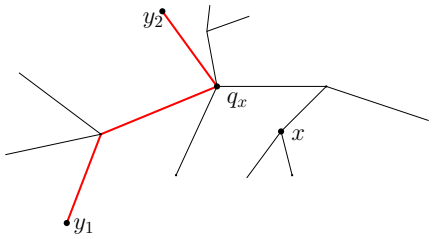

Figure 6.3. Illustrating the definition of q_x in the subtree $T(\mu)$ with two connectors y_1 and y_2. The path $\pi(y_1, y_2)$ is highlighted with thicker (red) segments.

Proof. As $L(\mu) \neq \emptyset$, μ is not the root. Thus, $T(\mu)$ has one or two connectors. We only discuss the most general case where $T(\mu)$ has two connectors since the other case is similar but easier. Let y_1 and y_2 denote the two connectors of $T(\mu)$, respectively. So the lists $L(y_1, \mu)$ and $L(y_2, \mu)$ are available.

Note that for any two points p and q in $T(\mu)$, $\pi(p, q)$ is also in $T(\mu)$ since $T(\mu)$ is connected.

Consider any point $x \in T(\mu)$. Suppose we traverse on $T(\mu)$ from x to y_1, and let q_x be the first point on $\pi(y_1, y_2)$ we encounter (e.g. see Fig 6.3; so q_x is x if $x \in \pi(y_1, y_2)$). Let $a_x = d(x, q_x)$ and $b_x = d(q_x, y_1)$. Thus, $d(y_1, x) = a_x + b_x$ and $d(y_2, x) = a_x + d(y_1, y_2) - b_x$.

For any $i \in L(\mu)$, since P_i does not have any location in $T(\mu)$, we have $F(i, y_1, \mu) + F(i, y_2, \mu) = 1$, and thus the following holds for $\mathsf{Ed}(x, P_i)$:

$$\mathsf{Ed}(x, P_i) = w_i \cdot \sum_{p_{ij} \in T} f_{ij} \cdot d(x, p_{ij})$$

$$= w_i \cdot \sum_{p_{ij} \in T(y_1, \mu)} f_{ij} \cdot d(x, p_{ij}) + w_i \cdot \sum_{p_{ij} \in T(y_2, \mu)} f_{ij} \cdot d(x, p_{ij})$$

$$= w_i \cdot [F(i, y_1, \mu) \cdot (a_x + b_x) + D(i, y_1, \mu)] + w_i \cdot [F(i, y_2, \mu) \cdot (a_x + d(y_1, y_2) - b_x) + D(i, y_2, \mu)]$$

$$= w_i \cdot [a_x + (F(i, y_1, \mu) - F(i, y_2, \mu)) \cdot b_x + D(i, y_1, \mu) + D(i, y_2, \mu) + F(i, y_2, \mu) \cdot d(y_1, y_2)].$$

Notice that for any $x \in T(\mu)$, all above values are constant except a_x and b_x. Therefore, if we consider a_x and b_x as two variables of x, $\mathsf{Ed}(x, P_i)$ is a linear function of them. In other words, $\mathsf{Ed}(x, P_i)$ defines a plane in $\mathbb{R}^3$, where the z-coordinates correspond to the values of $\mathsf{Ed}(x, P_i)$ and the x- and y-coordinates correspond to a_x and b_x respectively. In the following, we also use $\mathsf{Ed}(x, P_i)$ to refer to the plane defined

by it in $\mathbb{R}^3$.

Remark. This nice property for calculating $\mathsf{Ed}(x, P_i)$ is due to that μ has at most two connectors. This is another reason our decomposition requires every subtree $T(\mu)$ to have at most two connectors.

Recall that x covers P_i if $\mathsf{Ed}(x, P_i) \leq \lambda$. Consider the plane $H_\lambda : z = \lambda$ in $\mathbb{R}^3$. In general the two planes $\mathsf{Ed}(x, P_i)$ and H_λ intersect at a line l_i and we let h_i represent the closed half-plane of H_λ bounded by l_i and above the plane $\mathsf{Ed}(x, P_i)$. Let x_λ be the point (a_x, b_x) in the plane H_λ. An easy observation is that $\mathsf{Ed}(x, P_i) \leq \lambda$ if and only if $x_\lambda \in h_i$. Further, we say that l_i is an *upper bounding line* of h_i if h_i is below l_i and a *lower bounding line* otherwise. Observe that if l_i is an upper bounding line, then $\mathsf{Ed}(x, P_i) \leq \lambda$ if and only if x_λ is below l_i; if l_i is a lower bounding line, then $\mathsf{Ed}(x, P_i) \leq \lambda$ if and only if x_λ is above l_i.

Given any query point $x \in T(\mu)$, our goal for answering the query is to find all indices $i \in L(\mu)$ such that P_i is covered by x. Based on the above discussions, we do the following preprocessing. After $d(y_1, y_2)$ is computed, by using the information lists of y_1 and y_2, we compute all functions $\mathsf{Ed}(x, P_i)$ for all $i \in L(\mu)$ in $O(t_\mu)$ time. Then, we obtain a set U of all upper bounding lines and a set of all lower bounding lines on the plane H_λ defined by $\mathsf{Ed}(x, P_i)$ for all $i \in L(\mu)$. In the following, we first discuss the upper bounding lines. Let S_U denote the indices $i \in L(\mu)$ such that P_i defines an upper bounding line in U.

Given any point $x \in T(\mu)$, we first compute a_x and b_x. This can be done in constant time after $O(|T(\mu)|)$ time preprocessing, as follows. In the preprocessing, for each vertex v of $T(\mu)$, we compute the vertex q_v (defined in the similar way as q_x with respect to x) as well as the two values a_v and b_v (defined similarly as a_x and b_x, respectively). This can be easily done in $O(|T(\mu)|)$ time by traversing $T(\mu)$ and we omit the details. Given the point x, which is specified by an edge e containing x, let v be the incident vertex of e closer to y_1 and let δ be the length of e between v and x. Then, if e is on $\pi(y_1, y_2)$, we have $a_x = 0$ and $b_x = b_v + \delta$. Otherwise, $a_x = a_v + \delta$ and $b_x = b_v$.

After a_x and b_x are computed, the point $x_\lambda = (a_x, b_x)$ on the plane H_λ is also obtained. Then, according to our discussion, all uncertain points of S_U that are covered

by x correspond to exactly those lines of U above x_λ. Finding the lines of U above x_λ is actually the dual problem of half-plane range reporting query in $\mathbb{R}^2$ [81]. By using the dynamic convex hull maintenance data structure of Brodal and Jacob [82], with $O(|U| \log |U|)$ time and $O(|U|)$ space preprocessing, for any point x_λ, we can easily report all lines of U above x_λ in $O(\log |U| + k \log |U|)$ amortized time (i.e., by repeating k deletions), where k is the output size, and deleting a line from U can be done in $O(\log |U|)$ amortized time. Clearly, $|U| \leq t_\mu$.

On the set of all lower bounding lines, we do the similar preprocessing, and the query algorithm is symmetric.

Hence, the total preprocessing time is $O(|T(\mu)| + t_\mu \log t_\mu)$ time. Each query takes $O(\log t_\mu + k \log^2 t_\mu)$ amortized time and each remove operation can be performed in $O(\log t_\mu)$ amortized time. Note that $t_\mu \leq n$. The lemma thus follows.

□

The preprocessing algorithm for our data structure $\mathcal{A}_1$ consists of the following four steps. First, we compute the information lists for all nodes μ of Υ. Second, for each node $\mu \in \Upsilon$, we compute the data structure of Lemma 6.5.1. Third, for each $i \in [1, n]$, we compute a *node list* $L_\mu(i)$ containing all nodes $\mu \in \Upsilon$ such that $i \in L(\mu)$. Fourth, for each leave μ of Υ that is a vertex v of T holding a location p_{ij}, we maintain at μ the value $\mathsf{Ed}(v, P_i)$. Before giving the details of the above processing algorithm, we first assume the preprocessing work has been done and discuss the algorithm for answering the coverage-report-queries.

Given any point $x \in T$, we answer the coverage-report-query as follows. Note that x is in $T(\mu_x)$ for some leaf μ_x of Υ. For each node μ in the path of Υ from the root to μ_x, we apply the query algorithm in Lemma 6.5.1 to report all indices $i \in L(\mu)$ such that x covers P_i. In addition, if μ_x is a vertex of T holding a location p_{ij} such that P_i is active, then we report i if $\mathsf{Ed}(v, P_i)$, which is maintained at v, is at most λ. The following lemma proves the correctness and the performance of our query algorithm.

Lemma 6.5.2. *Our query algorithm correctly finds all active uncertain points that are covered by x in $O(\log M \log n + k \log n)$ amortized time, where k is the output size.*

Proof. Let π_x represent the path of Υ from the root to the leaf μ_x. To show the correctness of the algorithm, we argue that for each active uncertain point P_i that is covered by x, i will be reported by our query algorithm.

Indeed, if $T(\mu_x)$ is a vertex v of T holding a location p_{ij} of P_i, then the leaf μ_x maintains the value $\mathsf{Ed}(v, P_i)$, which is equal to $\mathsf{Ed}(x, P_i)$ as $x = v$. Hence, our algorithm will report i when it processes μ_x. Otherwise, no location of P_i is in $T(\mu_x)$. Since P_i has locations in T, if we go from the root to μ_x along π, we will eventually meet a node μ such that $T(\mu)$ does not have any location of P_i while $T(\mu')$ has at least one location of P_i, where μ' is the parent of μ. This implies that i is in $L(\mu)$, and consequently, our query algorithm will report i when it processes μ. This establishes the correctness of our query algorithm.

For the runtime, as the height of Υ is $O(\log M)$, we make $O(\log M)$ calls on the query algorithm in Lemma 6.5.1. Hence, the total time is $O(\log M \log n + k \log n)$. □

If an uncertain point P_i is deactivated, then we scan the node list $L_\mu(i)$ and for each node $\mu \in L_\mu(i)$, we remove i from the data structure by Lemma 6.5.1. The following lemma implies that the total time is $O(m_i \log M \log n)$.

Lemma 6.5.3. *For each $i \in [1, n]$, the number of nodes in $L_\mu(i)$ is $O(m_i \log M)$.*

Proof. Let α denote the number of nodes of $L_\mu(i)$. Our goal is to argue that i appears in $L(\mu)$ for $O(m_i \log M)$ nodes μ of Υ. Recall that if i is in $L(\mu)$ for a node $\mu \in \Upsilon$, then P_i has at least one location in $T(\mu')$, where μ' is the parent of μ. Since each node of Υ has at most four children, if N is the total number of nodes μ' such that P_i has at least one location in $T(\mu')$, then it holds that $\alpha \leq 4N$. Below we show that $N = O(m_i \log M)$, which will prove the lemma.

Consider any location p_{ij} of P_i. According to our decomposition, the subtrees $T(\mu)$ for all nodes μ in the same level of Υ are pairwise disjoint. Let v be the vertex of T that holds p_{ij}, and let μ_v be the leaf of Υ with $T(\mu_v) = v$. Observe that for any node $\mu \in \Upsilon$, p_{ij} appears in $T(\mu)$ if and only if μ is in the path of Υ from μ_v to the root. Hence, there are $O(\log M)$ nodes $\mu \in \Upsilon$ such that p_{ij} appears in $T(\mu)$. As P_i has m_i locations, we obtain $N = O(m_i \log M)$. □

The following lemma gives our preprocessing algorithm for building $\mathcal{A}_1$.

Lemma 6.5.4. $\sum_{\mu \in \Upsilon} t_\mu = O(M \log M)$, *and the preprocessing time for constructing the data structure* $\mathcal{A}_1$ *excluding the second step is* $O(M \log M)$.

Proof. We begin with the first step of the preprocessing algorithm for $\mathcal{A}_1$, i.e., computing the information lists for all nodes μ of Υ.

In order to do so, for each node $\mu \in \Upsilon$, we will also compute a sorted list $L'(\mu)$ of all such indices $i \in [1, n]$ that P_i has at least one location in $T(\mu)$, and further, for each connector y of $T(\mu)$, we will compute a list $L'(y, \mu)$ that is the same as $L'(\mu)$ except that each $i \in L'(y, \mu)$ is associated with two values: $F(i, y, \mu)$, which is equal to the probability sum of P_i in the subtree $T(y, \mu)$, and $D(i, y, \mu)$, which is equal to the expected distance from y to the locations of P_i in $T(y, \mu)$, i.e., $D(i, y, \mu) = w_i \cdot \sum_{p_{ij} \in T(y,\mu)} f_{ij} \cdot d(y, p_{ij})$. With a little abuse of notation, we call all above the *information lists* of μ (including its original information lists). In the following, we describe our algorithm for computing the information lists of all nodes μ of Υ. Let $F[1 \cdots n]$ and $D[1 \cdots n]$ be two arrays that we are going to use in our algorithm (they will mostly be used to compute the F values and D values of the information lists of connectors).

Initially, if μ is the root of Υ, we have $L'(\mu) = \{1, 2, \ldots, n\}$ and $L(\mu) = \emptyset$. Since $T(\mu)$ does not have any connectors, we do not need to compute the information lists for connectors.

Consider any internal node μ. We assume all information lists for μ has been computed (i.e., $L(\mu)$, $L'(\mu)$, and $L'(y, \mu)$, $L(y, \mu)$ for each connector y of $T(\mu)$). In the following we present our algorithm for processing μ, which will compute the information lists of all children of μ in $O(|T(\mu)|)$ time.

We first discuss the case where μ has two children, denoted by μ_1 and μ_2, respectively. Let v be the centroid of $T(\mu)$ that is used to decompose $T(\mu)$ into $T(\mu_1)$ and $T(\mu_2)$ (e.g., see Fig. 6.4). We first compute the information lists of μ_1, as follows.

We begin with computing the two lists $L(\mu_1)$ and $L'(\mu_1)$. Initially, we set both of them to $\emptyset$. We scan the list $L'(\mu)$ and for each $i \in L'(\mu)$, we reset $F[i] = 0$. Then, we scan the subtree $T(\mu_1)$, and for each location p_{ij}, we set $F[i] = 1$ as a flag showing that P_i has locations in $T(\mu_1)$. Afterwards, we scan the list $L'(\mu)$ again, and for each $i \in L'(\mu)$, if $F[i] = 1$, then we add i to $L'(\mu_1)$; otherwise, we add i to $L(\mu_1)$. This

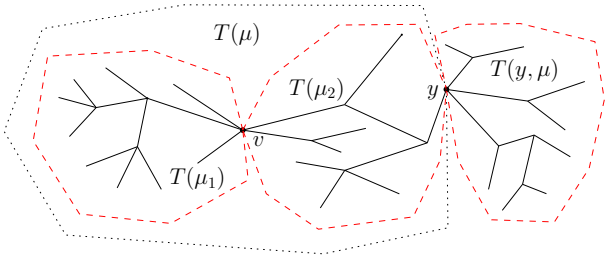

Figure 6.4. Illustrating the subtrees $T(\mu_1), T(\mu_2)$, and $T(y,\mu)$, where y is a connector of $T(\mu) = T(\mu_1) \cup T(\mu_2)$. Note that $T(y,\mu)$ is also $T(y,\mu_2)$ as $y \in T(\mu_2)$.

computes the two index lists $L(\mu_1)$ and $L'(\mu_1)$ for μ_1. The running time is $O(|T(\mu)|)$ since the size of $L'(\mu)$ is no more than $|T(\mu)|$.

We proceed to compute the information lists for the connectors of $T(\mu_1)$. Recall that v is a connector of $T(\mu_1)$. So we need to compute the two lists $L(v,\mu_1)$ and $L'(v,\mu_1)$, such that each index i in either list is associated with the two values $F(i,v,\mu_1)$ and $D(i,v,\mu_1)$. We first copy all indices of $L(\mu_1)$ to $L(v,\mu_1)$ and copy all indices of $L'(\mu_1)$ to $L'(v,\mu_1)$. Next we compute their F and D values as follows.

We first scan $L'(\mu)$ and for each $i \in L'(\mu)$, we reset $F[i] = 0$ and $D[i] = 0$. Next, we traverse $T(\mu_2)$ and for each location p_{ij}, we update $F[i] = F[i] + f_{ij}$ and $D[i] = D[i] + d(v,p_{ij})$ ($d(v,p_{ij})$ can be computed in constant time after $O(T(\mu))$-time preprocessing that computes $d(v,v')$ for every vertex $v' \in T(\mu)$ by traversing $T(\mu)$). Further, if $T(\mu_2)$ has a connector y other than v, then y must be a connector of $T(\mu)$ (e.g., see Fig. 6.4; there exists at most one such connector y); we scan the list $L'(y,\mu_2)$, and for each $i \in L'(y,\mu_2)$, we update $F[i] = F[i]+F(i,y,\mu)$ and $D[i] = D[i]+D(i,y,\mu)+d(v,y)\cdot F(i,y,\mu)$ ($d(v,y)$ is already computed in the preprocessing discussed above). Finally, we scan $L(v,\mu_1)$ (resp., $L'(v,\mu_1)$) and for each index i in $L(v,\mu_1)$ (resp., $L'(v,\mu_1)$), we set $F(i,v,\mu_1) = F[i]$ and $D(i,v,\mu_1) = D[i]$. This computes the two information lists $L(v,\mu_1)$ and $L'(v,\mu_1)$. The total time is $O(|T(\mu)|)$.

In addition, if $T(\mu_1)$ has a connector y other than v, then y must be a connector of $T(\mu)$ (e.g., see Fig. 6.2; there is only one such connector), and we further compute the two information lists $L(y,\mu_1)$ and $L'(y,\mu_1)$. To do so, we first copy all indices of $L(\mu_1)$ to $L(y,\mu_1)$ and copy all indices of $L'(\mu_1)$ to $L'(y,\mu_1)$. Observe that $L(y,\mu_1)$ and $L'(y,\mu_1)$ form a partition of the indices of $L'(y,\mu)$. For each index i in $L(y,\mu_1)$ (resp.,

$L'(y, \mu_1)$), we have $F(i, y, \mu_1) = F(i, y, \mu)$ and $D(i, y, \mu_1) = D(i, y, \mu)$. Therefore, the F and D values for $L'(y, \mu_1)$ and $L(y, \mu_1)$ can be obtained from $L'(y, \mu)$ by scanning the three lists $L'(y, \mu_1)$, $L(y, \mu_1)$, and $L'(y, \mu)$ simultaneously, as they are all sorted lists.

The above has computed the information lists for μ_1 and the total time is $O(|T(\mu)|)$. Using the similar approach, we can compute the information lists for μ_2, and we omit the details. This finishes the algorithm for processing μ where μ has two children.

If μ has three children, then $T(\mu)$ is decomposed into three subtrees in our decomposition. As discussed in Section 6.4.2 on the algorithm for Lemma 6.3.1, we can consider the decomposition of $T(\mu)$ consisting of two intermediate decomposition steps each of which decompose a subtree into two subtrees. For each intermediate step, we apply the above processing algorithm for the two-children case. In this way, we can compute the information lists for all three children of μ in $O(|T(\mu)|)$ time. If μ has four children, then similarly there are four intermediate decomposition steps and we apply the two-children case algorithm three times. The total processing time for μ is still $O(|T(\mu)|)$.

Once all internal nodes of Υ are processed, the information lists of all nodes are computed. Since processing each node μ of Υ takes $O(|T(\mu)|)$ time, the total time of the algorithm is $O(M \log M)$. This also implies that the total size of the information lists of all nodes of Υ is $O(M \log M)$, i.e., $\sum_{\mu \in \Upsilon} t_\mu = O(M \log M)$.

This above describes the first step of our preprocessing algorithm for $\mathcal{A}_1$. For the third step, the node lists $L_\mu(i)$ can be built during the course of the above algorithm. Specifically, whenever an index i is added to $L(\mu)$ for some node μ of Υ, we add μ to the list $L_\mu(i)$. This only introduces constant extra time each. Therefore, the overall algorithm has the same runtime asymptotically as before.

For the fourth step, for each leaf μ of Υ such that $T(\mu)$ is a vertex v of T, we do the following. Let p_{ij} be the uncertain point location at v. Based on our above algorithm, we have $L'(\mu) = \{i\}$. Since v is a connector, we have a list $L'(v, \mu)$ consisting of i itself and two values $F(i, v, \mu)$ and $D(i, v, \mu)$. Notice that $\mathsf{Ed}(v, P_i) = D(i, v, \mu)$. Hence, once the above algorithm finishes, the value $\mathsf{Ed}(v, P_i)$ is available.

As a summary, the preprocessing algorithm for $\mathcal{A}_1$ except the second step runs in $O(M \log M)$ time. The lemma thus follows. □

For the second step of the preprocessing of $\mathcal{A}_1$, since $\sum_{\mu \in \Upsilon} t_\mu = O(M \log M)$ by Lemma 6.5.4, applying the preprocessing algorithm of Lemma 6.5.1 on all nodes of Υ takes $O(M \log^2 M)$ time and $O(M \log M)$ space in total. Hence, the total preprocessing time of $\mathcal{A}_1$ is $O(M \log^2 M)$ and the space is $O(M \log M)$. This proves Lemma 6.3.4.

6.5.2 The Data Structure $\mathcal{A}_3$

In this section, we present the data structure $\mathcal{A}_3$. Given any point x and any uncertain point P_i, $\mathcal{A}_3$ is used to compute the expected distance $\mathsf{Ed}(x, P_i)$. Note that we do not need to consider the remove operations for $\mathcal{A}_3$.

We follow the notation defined in Section 6.5.1. As preprocessing, for each node $\mu \in \Upsilon$, we compute the information lists $L(\mu)$ and $L(y, \mu)$ for each connector y of $T(\mu)$. This is actually the first step of the preprocessing algorithm of $\mathcal{A}_1$ in Section 6.5.1. Further, we also preform the fourth step of the preprocessing algorithm for $\mathcal{A}_1$. The above can be done in $O(M \log M)$ time by Lemma 6.5.4.

Consider any node $\mu \in \Upsilon$ with $L(\mu) \neq \emptyset$. Given any point $x \in T(\mu)$, we have shown in the proof of Lemma 6.5.1 that $\mathsf{Ed}(x, P_i)$ is a function of two variables a_x and b_x. As preprocessing, we compute these functions for all $i \in L(\mu)$, which takes $O(t_\mu)$ time as shown in the proof of Lemma 6.5.1. For each $i \in L(\mu)$, we store the function $\mathsf{Ed}(x, P_i)$ at μ. The total preprocessing time for $\mathcal{A}_3$ is $O(M \log M)$.

Consider any query on a point $x \in T$ and $P_i \in \mathcal{P}$. Note that x is specified by an edge e and its distance to a vertex of e. Let μ_x be the leaf of Υ with $x \in T(\mu_x)$. If x is in the interior of e, then $T(\mu_x)$ is the open edge e; otherwise, $T(\mu_x)$ is a single vertex $v = x$.

We first consider the case where x is in the interior of e. In this case, P_i does not have any location in $T(\mu_x)$ since $T(\mu_x)$ is an open edge. Hence, if we go along the path of Υ from the root to μ_x, we will encounter a first node μ' with $i \in L(\mu')$. After finding μ', we compute a_x and b_x in $T(\mu')$, which can be done in constant time after $O(|T(\mu')|)$ time preprocessing on $T(\mu')$, as discussed in the proof of Lemma 6.5.1 (so the total preprocessing time for all nodes of Υ is $O(M \log M)$). After a_x and b_x are computed, we can obtain the value $\mathsf{Ed}(x, P_i)$.

Remark. One can verify (from the proof of Lemma 6.5.1) that as x changes on e, $\mathsf{Ed}(x, P_i)$ is a linear function of x because one of a_x and b_x is constant and the other linearly changes as x changes in e. Hence, the above also computes the linear function $\mathsf{Ed}(x, P_i)$ for $x \in e$.

To find the above node μ', for each node μ in the path of Υ from the root to μ_x, we need to determine whether $i \in L(\mu)$. If we represented the sorted index list $L(\mu)$ by a binary search tree, then we could spend $O(\log n)$ time on each node μ and thus the total query time would be $O(\log n \log M)$. To remove the $O(\log n)$ factor, we further enhance our preprocessing work by building a fractional cascading structure [49] on the sorted index lists $L(\mu)$ for all nodes μ of Υ. The total preprocessing time for building the structure is linear in the total number of nodes of all lists, which is $O(M \log M)$ by Lemma 6.5.4. For each node μ, the fractional cascading structure will create a new list $L^*(\mu)$ such that $L(\mu) \subseteq L^*(\mu)$. Further, for each index $i \in L^*(\mu)$, if it is also in $L(\mu)$, then we set a flag as an indicator. Setting the flags for all nodes of Υ can be done in $O(M \log M)$ time as well. Using the fractional cascading structure, we only need to do binary search on the list in the root and then spend constant time on each subsequent node [49], and thus the total query time is $O(\log M)$.

If x is a vertex v of T, then depending on whether the location at v is P_i's or not, there are two subcases. If it is not, then we apply the same query algorithm as above. Otherwise, let p_{ij} be the location at v. Recall that in our preprocessing, the value $\mathsf{Ed}(v, P_i)$ has already been computed and stored at μ_x as $T(\mu_x) = v$. Due to $v = x$, we obtain $\mathsf{Ed}(x, P_i) = \mathsf{Ed}(v, P_i)$.

Hence, in either case, the query algorithm runs in $O(\log M)$ time. This proves Lemma 6.3.7.

6.5.3 The Data Structure $\mathcal{A}_2$

The data structure $\mathcal{A}_2$ is for answering candidate-center-queries: Given any vertex $v \in T_m$, the query asks for the candidate center c for the active medians in $T_m(v)$, which is the subtree of T_m rooted at v. Once an uncertain point is deactivated, $\mathcal{A}_2$ can also support the operation of removing it.

Consider any vertex $v \in T_m$. Recall that due to our reindexing, the indices of all

medians in $T_m(v)$ exactly form the range $R(v)$. Recall that the candidate center c is the point on the path $\pi(v, r)$ closest to r with $\mathsf{Ed}(v, P_i) \leq \lambda$ for each active uncertain point P_i with $i \in R(v)$. Also recall that our algorithm invariant guarantees that whenever a candidate-center-query is called at a vertex v, then it holds that $\mathsf{Ed}(v, P_i) \leq \lambda$ for each active uncertain point P_i with $i \in R(v)$. However, we actually give a result that can answer a more general query. Specifically, given a range $[k, j]$ with $1 \leq k \leq j \leq n$, let v_{kj} be the lowest common ancestor of all medians p_i^* with $i \in [k, j]$ in T_m; if $\mathsf{Ed}(v_{kj}, P_i) > \lambda$ for some active P_i with $i \in [k, j]$, then our query algorithm will return $\emptyset$; otherwise, our algorithm will compute a point c on $\pi(v_{kj}, r)$ closest to r with $\mathsf{Ed}(c, P_i) \leq \lambda$ for each active P_i with $i \in [k, j]$. We refer to it as the *generalized* candidate-center-query.

In the preprocessing, we build a complete binary search tree $\mathcal{T}$ whose leaves from left to right correspond to indices $1, 2, \ldots, n$. For each node u of $\mathcal{T}$, let $R(u)$ denote the set of indices corresponding to the leaves in the subtree of $\mathcal{T}$ rooted at u. For each median p_i^*, define q_i to be the point x on the path $\pi(p_i^*, r)$ of T_m closest to r with $\mathsf{Ed}(x, P_i) \leq \lambda$.

For each node u of $\mathcal{T}$, we define a node $q(u)$ as follows. If u is a leaf, define $q(u)$ to be q_i, where i is the index corresponding to leaf u. If u is an internal node, let v_u denote the vertex of T_m that is the lowest common ancestor of the medians p_i^* for all $i \in R(u)$. If $\mathsf{Ed}(v_u, P_i) \leq \lambda$ for all $i \in R(u)$ (or equivalently, q_i is in $\pi(v_u, r)$ for all $i \in R(u)$), then define $q(u)$ to be the point x on the path $\pi(v_u, r)$ of T_m closest to r with $\mathsf{Ed}(x, P_i) \leq \lambda$ for all $i \in R(u)$; otherwise, $q(u) = \emptyset$.

Lemma 6.5.5. *The points $q(u)$ for all nodes $u \in \mathcal{T}$ can be computed in $O(M \log M + n \log^2 M)$ time.*

Proof. Assume the data structure $\mathcal{A}_3$ for Lemma 6.3.7 has been computed in $O(M \log M)$ time. In the following, by using $\mathcal{A}_3$ we compute $q(u)$ for all nodes $u \in \mathcal{T}$ in $O(M + n \log^2 M)$ time.

We first compute q_i for all medians p_i^*. Consider the depth-first-search on T_m starting from the root r. During the traversal, we use a stack S to maintain all vertices in order along the path $\pi(r, v)$ whenever a vertex v is visited. Such a stack can be easily maintained by standard techniques (i.e., push new vertices into S when we go

"deeper" and pop vertices out of S when backtrack), without affecting the linear-time performance of the traversal asymptotically. Suppose the traversal visits a median p_i^*. Then, the vertices of S essentially form the path $\pi(r, p_i^*)$. To compute q_i, we do binary search on the vertices of S, as follows.

We implement S by using an array of size M. Since the order of the vertices of S is the same as their order along $\pi(r, p_i^*)$, the expected distances $\mathsf{Ed}(v, P_i)$ of the vertices $v \in S$ along their order in S are monotonically changing. Consider a middle vertex v of S. The vertex v partitions S into two subarrays such that one subarray contains all vertices of $\pi(r, v)$ and the other contains vertices of $\pi(v, p_i^*)$. We compute $\mathsf{Ed}(v, P_i)$ by using data structure $\mathcal{A}_3$. Depending on whether $\mathsf{Ed}(v, P_i) \leq \lambda$, we can proceed on only one subarray of M. The binary search will eventually locate an edge $e = (v, v')$ such that $\mathsf{Ed}(v, P_i) \leq \lambda$ and $\mathsf{Ed}(v', P_i) > \lambda$. Then, we know that q_i is located on $e \setminus \{v'\}$. We further pick any point x in the interior of e and the data structure $\mathcal{A}_3$ can also compute the function $\mathsf{Ed}(x, P_i)$ for $x \in e$ as remarked in Section 6.5.2. With the function $\mathsf{Ed}(x, P_i)$ for $x \in e$, we can compute q_i in constant time. Since the binary search calls $\mathcal{A}_3$ $O(\log M)$ times, the total time of the binary search is $O(\log^2 M)$.

In this way, we can compute q_i for all medians p_i^* with $i \in [1, n]$ in $O(M + n \log^2 M)$ time, where the $O(n \log^2 M)$ time is for the binary search procedures in the entire algorithm and the $O(M)$ time is for traversing the tree T_m. Note that this also computes $q(u)$ for all leaves u of $\mathcal{T}$.

We proceed to compute the points $q(u)$ for all internal nodes μ of $\mathcal{T}$ in a bottom-up manner. Consider an internal node u such that $q(u_1)$ and $q(u_2)$ have been computed, where u_1 and u_2 are the children of u, respectively. We compute $q(u)$ as follows.

If either one of $q(u_1)$ and $q(u_2)$ is $\emptyset$, then we set $q(u) = \emptyset$. Otherwise, we do the following. Let i (resp., j) be the leftmost (resp., rightmost) leaf in the subtree $\mathcal{T}(u)$ of $\mathcal{T}$ rooted at u. We first find the lowest comment ancestor of p_i^* and p_j^* in the tree T_m, denoted by v_{ij}. Due to our particular way of defining indices of all medians, v_{ij} is the lowest common ancestor of the medians p_k^* for all $k \in [i, j]$. We determine whether $q(u_1)$ and $q(u_2)$ are both on $\pi(r, v_{ij})$. If either one is not on $\pi(r, v_{ij})$, then we set $q(u) = \emptyset$; otherwise, we set $q(u)$ to the one of $q(u_1)$ and $q(u_2)$ closer to v_{ij}.

The above for computing $q(u)$ can be implemented in $O(1)$ time, after $O(M)$ time

preprocessing on T_m. Specifically, with $O(M)$ time preprocessing on T_m, given any two vertices of T_m, we can compute their lowest common ancestor in $O(1)$ time [66, 67]. Hence, we can compute v_{ij} in constant time. To determine whether $q(u_1)$ is on $\pi(r, v_{ij})$, we use the following approach. As a point on T_m, $q(u_1)$ is specified by an edge e_1 and its distance to one incident vertex of e_1. Let v_1 be the incident vertex of e_1 that is farther from the root r. Observe that $q(u_1)$ is on $\pi(r, v_{ij})$ if and only if the lowest common ancestor of v_1 and v_{ij} is v_1. Hence, we can determine whether $q(u_1)$ is on $\pi(r, v_{ij})$ in constant time by a lowest common ancestor query. Similarly, we can determine whether $q(u_2)$ is on $\pi(r, v_{ij})$ in constant time. Assume both $q(u_1)$ and $q(u_2)$ are on $\pi(r, v_{ij})$. To determine which one of $q(u_1)$ and $q(u_2)$ is closer to v_{ij}, if they are on the same edge e of T_m, then this can be done in constant time since both points are specified by their distances to an incident vertex of e. Otherwise, let e_1 be the edge of T_m containing $q(u_1)$ and let v_1 be the incident vertex of e_1 farther to r; similarly, let e_2 be the edge of T_m containing $q(u_2)$ and let v_2 be the incident vertex of e_2 farther to r. Observe that $q(u_1)$ is closer to v_{ij} if and only if the lowest common ancestor of v_1 and v_2 is v_2, which can be determined in constant time by a lowest common ancestor query.

The above shows that we can compute $q(u)$ in constant time based on $q(u_1)$ and $q(u_2)$. Thus, we can compute $q(u)$ for all internal nodes u of $\mathcal{T}$ in $O(n)$ time. The lemma thus follows. □

In addition to constructing the tree $\mathcal{T}$ as above, our preprocessing for $\mathcal{A}_2$ also includes building a lowest common ancestor query data structure on T_m in $O(M)$ time, such that given any two vertices of T_m, we can compute their lowest common ancestor in $O(1)$ time [66,67]. This finishes the preprocessing for $\mathcal{A}_2$. The total time is $O(M \log M + n \log^2 M)$.

The following lemma gives our algorithm for performing operations on $\mathcal{T}$.

Lemma 6.5.6. *Given any range $[k, j]$, we can answer each generalized candidate-center-query in $O(\log n)$ time, and each remove operation (i.e., deactivating an uncertain point) can be performed in $O(\log n)$ time.*

Proof. We first describe how to perform the remove operations. Suppose an uncertain point P_i is deactivated. Let u_i be the leaf of $\mathcal{T}$ corresponding to the index i. We first

set $q(u_i) = \emptyset$. Then, we consider the path of $\mathcal{T}$ from u_i to the root in a bottom-up manner, and for each node u, we update $q(u)$ based on $q(u_1)$ and $q(u_2)$ in constant time in exactly the same way as in the Lemma 6.5.5, where u_1 and u_2 are the two children of u, respectively. In this way, each remove operation can be performed in $O(\log n)$ time.

Next we discuss the generalized candidate-center-query on a range $[k, j]$. By standard techniques, we can locate a set S of $O(\log n)$ nodes of $\mathcal{T}$ such that the descendant leaves of these nodes exactly correspond to indices in the range $[k, j]$. We find the lowest common ancestor v_{kj} of p_k^* and p_j^* in T_m in constant time. Then, for each node $u \in S$, we check whether $q(u)$ is on $\pi(r, v_{kj})$, which can be done in constant time by using the lowest common ancestor query in the same way as in the proof of Lemma 6.5.5. If $q(u)$ is not on $\pi(r, v_{kj})$ for some $u \in S$, then we simply return $\emptyset$. Otherwise, $q(u)$ is on $\pi(r, v_{kj})$ for every $u \in S$. We further find the point $q(u)$ that is closest to v_{kj} among all $u \in S$, and return it as the answer to the candidate-center-query on $[k, j]$. Such a $q(u)$ can be found by comparing the nodes of S in $O(\log n)$ time. Specifically, for each pair u and u' in a comparison, we find among $q(u)$ and $q(u')$ the one closer to v_{kj}, which can be done in constant time by using the lowest common ancestor query in the same way as in the proof of Lemma 6.5.5, and then we keep comparing the above closer one to the rest of the nodes in S. In this way, the candidate-center-query can be handled in $O(\log n)$ time. □

This proves Lemma 6.3.5.

6.5.4 Handling the Degenerate Case and Reducing the General Case to the Vertex-Constrained Case

The above has solved the vertex-constrained case, i.e., all locations of $\mathcal{P}$ are at vertices of T and each vertex of T contains at least one location of $\mathcal{P}$. Recall that we have made a general position assumption that every vertex of T has only one location of $\mathcal{P}$. For the degenerate case, our algorithm still works in the same way as before with the following slight change. Consider a subtree $T(\mu)$ corresponding to a node μ of Υ. In the degenerate case, since a vertex of $T(\mu)$ may hold multiple uncertain point locations of $\mathcal{P}$, we define the size $|T(\mu)|$ to be the total number of all uncertain point

locations in $T(\mu)$. In this way, the algorithm and the analysis follow similarly as before. In fact, the performance of the algorithm becomes even better in the degenerate case since the height of the decomposition tree Υ becomes smaller (specifically, it is bounded by $O(\log t)$, where t is the number of vertices of T, and $t < M$ in the degenerate case).

The above has solved the vertex-constrained case. In the general case, a location of $\mathcal{P}$ may be in the interior of an edge of T and a vertex of T may not hold any location of $\mathcal{P}$. The following theorem solves the general case by reducing it to the vertex-constrained case. The reduction is almost the same as the one given in Chapter 3 for the one-center problem and we include it here for the completeness of this paper.

Lemma 6.5.7. *The center-coverage problem on $\mathcal{P}$ and T is solvable in $O(\tau + M + |T|)$ time, where τ is the time for solving the same problem on $\mathcal{P}$ and T if this were a vertex-constrained case.*

Proof. We reduce the problem to an instance of the vertex-constrained case and then apply our algorithm for the vertex-constrained case. More specifically, we will modify the tree T to obtain another tree T' of size $\Theta(M)$. We will also compute another set $\mathcal{P}'$ of n uncertain points on T', which correspond to the uncertain points of $\mathcal{P}$ with the same weights, but each uncertain point P_i of $\mathcal{P}'$ has at most $2m_i$ locations on T'. Further, each location of $\mathcal{P}'$ is at a vertex of T' and each vertex of T' holds at least one location of $\mathcal{P}'$, i.e., it is the vertex-constrained case. We will show that we can obtain T' and $\mathcal{P}'$ in $O(M + |T|)$ time. Finally, we will show that given a set of centers on T' for $\mathcal{P}'$, we can find a corresponding set of the same number of centers on T for $\mathcal{P}$ in $O(M + |T|)$ time. The details are given below.

We assume that for each edge e of T, all locations of $\mathcal{P}$ on e have been sorted (otherwise we sort them first, which would introduce an additional $O(M \log M)$ time on the problem reduction). We traverse T, and for each edge e, if e contains some locations of $\mathcal{P}$ in its interior, we create a new vertex in T for each such location. In this way, we create at most M new vertices for T. The above can be done in $O(M + |T|)$ time. We use T_1 to denote the new tree. Note that $|T_1| = O(M + |T|)$. For each vertex v of T_1, if v does not hold any location of $\mathcal{P}$, we call v an *empty* vertex.

Next, we modify T_1 in the following way. First, for each leaf v of T_1, if v is empty, then we remove v from T_1. We keep doing this until every leaf of the remaining tree is not empty. Let T_2 denote the tree after the above step (e.g., see Fig. 3.6 in Chapter 3). Second, for each internal vertex v of T_2, if the degree of v is 2 and v is empty, then we remove v from T_2 and merge its two incident edges as a single edge whose length is equal to the sum of the lengths of the two incident edges of v. We keep doing this until every degree-2 vertex of the remaining tree is not empty. Let T' represent the remaining tree (e.g., see Fig. 3.6 in Chapter 3). The above two steps can be implemented in $O(|T_1|)$ time, e.g., by a post-order traversal of T_1. We omit the details.

Notice that every location of $\mathcal{P}$ is at a vertex of T' and every vertex of T' except those whose degrees are at least three holds a location of $\mathcal{P}$. Let V denote the set of all vertices of T' and let V_3 denote the set of the vertices of T' whose degrees are at least three. Clearly, $|V_3| \leq |V \setminus V_3|$. Since each vertex in $V \setminus V_3$ holds a location of $\mathcal{P}$, we have $|V \setminus V_3| \leq M$, and thus $|V_3| \leq M$.

To make every vertex of T' contain a location of an uncertain point, we first arbitrarily pick m_1 vertices from V_3 and remove them from V_3, and set a "dummy" location for P_1 at each of these vertices with zero probability. We keep picking next m_2 vertices from V_3 for P_2 and continue this procedure until V_3 becomes empty. Since $|V_3| \leq M$, the above procedure will eventually make V_3 empty before we "use up" all n uncertain points of $\mathcal{P}$. We let $\mathcal{P}'$ be the set of new uncertain points. For each $P_i \in \mathcal{P}$, it has at most $2m_i$ locations on T'.

Since now every vertex of T' holds a location of $\mathcal{P}'$ and every location of $\mathcal{P}'$ is at a vertex of T', we obtain an instance of the vertex-constrained case on T' and $\mathcal{P}'$. Hence, we can use our algorithm for the vertex-constrained case to compute a set C' of centers on T' in $O(\tau)$ time. In the following, for each center $c' \in C'$, we find a corresponding center c on the original tree T such that P_i is covered by c on T if and only if P_i' is covered by c' on T'.

Observe that every vertex v of T' also exists as a vertex in T_1, and every edge (u, v) of T' corresponds to the simple path in T_1 between u and v. Suppose c' is on an edge (u, v) of T' and let δ be the length of e between u and c'. We locate a corresponding c_1 in T_1 in the simple path from u to v at distance δ from u. On the other hand, by

our construction from T to T_1, if an edge e of T does not appear in T_1, then e is broken into several edges in T_1 whose total length is equal to that of e. Hence, every point of T corresponds a point on T_1. We find the point on T that corresponds to c_1 of T_1, and let the point be c.

Let C be the set of points c on T corresponding to all $c' \in C'$ on T, as defined above. Let C_1 be the set of points c_1 on T_1 corresponding to all $c' \in C'$ on T'. To compute C, we first compute C_1. This can be done by traversing both T' and T_1, i.e., for each edge e of T' that contains centers c' of C', we find the corresponding points c_1 in the path of T_1 corresponding to the edge e. Since the paths of T_1 corresponding to the edges of T' are pairwise edge-disjoint, the runtime for computing C_1 is $O(|T_1| + |T'|)$. Next we compute C, and similarly this can be done by traversing both T_1 and T in $O(|T_1| + |T|)$ time. Hence, the total time for computing C is $O(|T| + M)$ since both $|T_1|$ and $|T'|$ are bounded by $O(|T| + M)$.

As a summary, we can find an optimal solution for the center-coverage problem on T and $\mathcal{P}$ in $O(\tau + M + |T|)$ time. The lemma thus follows. □

6.6 The k-Center Problem

The k-center problem is to find a set C of k centers on T minimizing the value $\max_{1 \le i \le n} d(C, P_i)$, where $d(C, P_i) = \min_{c \in C} d(c, P_i)$. Let $\lambda_{opt} = \max_{1 \le i \le n} d(C, P_i)$ for an optimal solution C, and we call λ_{opt} the *optimal covering range*.

As the center-coverage problem, we can also reduce the general k-center problem to the vertex-constrained case. The reduction is similar to the one in Lemma 6.5.7 and we omit the details. In the following, we only discuss the vertex-constrained case and we assume the problem on T and $\mathcal{P}$ is a vertex-constrained case. Let τ denote the running time for solving the center-coverage algorithm on T and $\mathcal{P}$.

To solve the k-center problem, the key is to compute λ_{opt}, after which we can compute k centers in additional $O(\tau)$ time using our algorithm for the center-coverage problem with $\lambda = \lambda_{opt}$. To compute λ_{opt}, there are two main steps. In the first step, we find a set S of $O(n^2)$ *candidate values* such that λ_{opt} must be in S. In the second step, we compute λ_{opt} in S. Below we first compute the set S.

For any two medians p_i^* and p_j^* on T_m, observe that as x moves on $\pi(p_i^*, p_j^*)$ from p_i^*

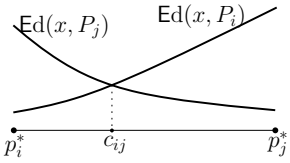

Figure 6.5. Illustrating c_{ij} and the two functions $\mathsf{Ed}(x, P_i)$ and $\mathsf{Ed}(x, P_j)$ as x changes in the path $\pi(p_i^*, p_j^*)$. $\pi(p_i^*, p_j^*)$ is shown as a segment.

to p_j^*, $\mathsf{Ed}(x, P_i)$ is monotonically increasing and $\mathsf{Ed}(x, P_j)$ is monotonically decreasing (e.g., see Fig. 6.5); we define c_{ij} to be a point on the path $\pi(p_i^*, p_j^*)$ with $\mathsf{Ed}(c_{ij}, P_i) = \mathsf{Ed}(c_{ij}, P_j)$, and we let $c_{ij} = \emptyset$ if such a point does not exist on $\pi(p_i^*, p_j^*)$. We have the following lemma.

Lemma 6.6.1. *Either* $\lambda_{opt} = \mathsf{Ed}(p_i^*, P_i)$ *for some uncertain point* P_i *or* $\lambda_{opt} = \mathsf{Ed}(c_{ij}, P_i) = \mathsf{Ed}(c_{ij}, P_j)$ *for two uncertain points* P_i *and* P_j.

Proof. Consider any optimal solution and let C be the set of all centers. For each $c \in C$, let $Q(c)$ be the set of uncertain points that are covered by c with respect to λ_{opt}, i.e., for each $P_i \in Q(c)$, $\mathsf{Ed}(c, P_i) \leq \lambda_{opt}$. Let C' be the subset of all centers $c \in C$ such that $Q(c)$ has an uncertain point P_i with $\mathsf{Ed}(c, P_i) = \lambda_{opt}$ and there is no other center $c' \in C$ with $\mathsf{Ed}(c', P_i) < \lambda_{opt}$. For each $c \in C'$, let $Q'(c)$ be the set of all uncertain points P_i such that $\mathsf{Ed}(c, P_i) = \lambda_{opt}$.

If there exists a center $c \in C'$ with an uncertain point $P_i \in Q'(c)$ such that c is at p_i^*, then the lemma follows since $\lambda_{opt} = \mathsf{Ed}(c, P_i) = \mathsf{Ed}(p_i^*, P_i)$. Otherwise, if there exists a center $c \in C'$ with two uncertain points P_i and P_j in $Q'(c)$ such that c is at c_{ij}, then the lemma also follows since $\lambda_{opt} = \mathsf{Ed}(c_{ij}, P_i) = \mathsf{Ed}(c_{ij}, P_j)$. Otherwise, if we move each $c \in C'$ towards the median p_j^* for any $P_j \in Q'(c)$, then $\mathsf{Ed}(c, P_i)$ for every $P_i \in Q'(c)$ becomes non-increasing. During the above movements of all $c \in C'$, one of the following two cases must happen (since otherwise we would obtain another set C'' of k centers with $\max_{1 \leq i \leq n} d(C'', P_i) < \lambda_{opt}$, contradicting with that λ_{opt} is the optimal covering range): either a center c of C' arrives at a median p_i^* with $\lambda_{opt} = \mathsf{Ed}(c, P_i) = \mathsf{Ed}(p_i^*, P_i)$ or a center c of C' arrives at c_{ij} for two uncertain points P_i and P_j with $\lambda_{opt} = \mathsf{Ed}(c_{ij}, P_i) = \mathsf{Ed}(c_{ij}, P_j)$. In either case, the lemma follows. □

In light of Lemma 6.6.1, we let $S = S_1 \cup S_2$ with $S_1 = \{\mathsf{Ed}(p_i^*, P_i) \mid 1 \leq i \leq n\}$ and $S_2 = \{\mathsf{Ed}(c_{ij}, P_i) \mid 1 \leq i, j \leq n\}$ (if $c_{ij} = \emptyset$ for a pair i and j, then let $\mathsf{Ed}(c_{ij}, P_i) = 0$). Hence, λ_{opt} must be in S and $|S| = O(n^2)$.

We assume the data structure $\mathcal{A}_3$ has been computed in $O(M \log M)$ time. Then, computing the values of S_1 can be done in $O(n \log M)$ time by using $\mathcal{A}_3$. The following lemma computes S_2 in $O(M + n^2 \log n \log M)$ time.

Lemma 6.6.2. *After $O(M)$ time preprocessing, we can compute $\mathsf{Ed}(c_{ij}, P_i)$ in $O(\log n \cdot \log M)$ time for any pair i and j.*

Proof. As preprocessing, we do the following. First, we compute a lowest common ancestor query data structure on T_m in $O(M)$ time such that given any two vertices of T_m, their lowest common ancestor can be found in $O(1)$ time [66, 67]. Second, for each vertex v of T_m, we compute the length $d(v, r)$, i.e., the number of edges in the path of T_m from v to the root r of T_m. Note that $d(v, r)$ is also the depth of v. Computing $d(v, r)$ for all vertices v of T_m can be done in $O(M)$ time by a depth-first-traversal of T_m starting from r. For each vertex $v \in T_m$ and any integer $d \in [0, d(v, r)]$, we use $\alpha(v, d)$ to denote the ancestor of v whose depth is d. We build a *level ancestor query* data structure on T_m in $O(M)$ time that can compute $\alpha(v, d)$ in constant time for any vertex v and any $d \in [0, d(v, r)]$ [83]. The total time of the above processing is $O(M)$.

Consider any pair i and j. We present an algorithm to compute c_{ij} in $O(\log n \cdot \log M)$ time, after which $\mathsf{Ed}(c_{ij}, P_i)$ can be computed in $O(\log M)$ time by using the data structure $\mathcal{A}_3$.

Observe that $c_{ij} \neq \emptyset$ if and only if $\mathsf{Ed}(p_i^*, P_i) \leq \mathsf{Ed}(p_i^*, P_j)$ and $\mathsf{Ed}(p_j^*, P_j) \leq \mathsf{Ed}(p_j^*, P_i)$. Using $\mathcal{A}_3$, we can compute the four expected distances in $O(\log M)$ time and thus determine whether $c_{ij} = \emptyset$. If yes, we simply return zero. Otherwise, we proceed as follows.

Note that c_{ij} is a point $x \in \pi(p_i^*, p_j^*)$ minimizing the value $\max\{\mathsf{Ed}(x, P_i), \mathsf{Ed}(x, P_j)\}$ (e.g., see Fig. 6.5). To compute c_{ij}, by using a lowest common ancestor query, we find the lowest common ancestor v_{ij} of p_i^* and p_j^* in constant time. Then, we search c_{ij} on the path $\pi(p_i^*, v_{ij})$, as follows (we will search the path $\pi(p_j^*, v_{ij})$ later). To simplify the notation, let $\pi = \pi(p_i^*, v_{ij})$. By using the level ancestor queries, we can find the

middle edge of π in $O(1)$ time. Specifically, we find the two vertices $v_1 = \alpha(p_i^*, k)$ and $v_2 = \alpha(p_i^*, k+1)$, where $k = \lfloor (d(p_i^*, r) + d(v_{ij}, r))/2 \rfloor$. Note that the two values $d(p_i^*, r)$ and $d(v_{ij}, r)$ are computed in the preprocessing. Hence, v_1 and v_2 can be found in constant time by the level ancestor queries. Clearly, the edge $e = (v_1, v_2)$ is the middle edge of π.

After e is the obtained, by the data structure $\mathcal{A}_3$ and as remarked in Section 6.5.2, we can obtain the two functions $\mathsf{Ed}(x, P_i)$ and $\mathsf{Ed}(x, P_j)$ on $x \in e$ in $O(\log M)$ time, and both functions are linear in x for $x \in e$. As x moves in e from one end to the other, one of $\mathsf{Ed}(x, P_i)$ and $\mathsf{Ed}(x, P_j)$ is monotonically increasing and the other is monotonically decreasing. Therefore, we can determine in constant time whether c_{ij} is on π_1, π_2, or e, where π_1 and π_2 are the sub-paths of π partitioned by e. If c_{ij} is on e, then c_{ij} can be computed immediately by the two functions and we can finish the algorithm. Otherwise, the binary search proceeds on either π_1 or π_2 recursively.

For the runtime, the binary search has $O(\log n)$ iterations and each iteration runs in $O(\log M)$ time. So the total time of the binary search on $\pi(p_i^*, v_{ij})$ is $O(\log n \log M)$. The binary search will either find c_{ij} or determine that c_{ij} is at v_{ij}. The latter case actually implies that c_{ij} is in the path $\pi(p_j^*, v_{ij})$, and thus we apply the similar binary search on $\pi(p_j^*, v_{ij})$, which will eventually compute c_{ij}. Thus, the total time for computing c_{ij} is $O(\log n \log M)$.

The lemma thus follows. □

The following theorem summarizes our algorithm.

Theorem 6.6.3. *An optimal solution for the k-center problem can be found in $O(n^2 \log n \log M + M \log^2 M \log n)$ time.*

Proof. Assume the data structure $\mathcal{A}_3$ has been computed in $O(M \log M)$ time. Computing S_1 can be done in $O(n \log M)$ time. Computing S_2 takes $O(M + n^2 \log n \log M)$ time. After S is computed, we find λ_{opt} from S as follows.

Given any λ in S, we can use our algorithm for the center-coverage problem to find a minimum number k' of centers with respect to λ. If $k' \leq k$, then we say that λ is *feasible*. Clearly, λ_{opt} is the smallest feasible value in S. To find λ_{opt} from S, we first

sort all values in S and then do binary search using our center-coverage algorithm as a decision procedure. In this way, λ_{opt} can be found in $O(n^2 \log n + \tau \log n)$ time.

Finally, we can find an optimal solution using our algorithm for the covering problem with $\lambda = \lambda_{opt}$ in $O(\tau)$ time. Therefore, the total time of the algorithm is $O(n^2 \log n \log M + \tau \log n)$, which is $O(n^2 \log n \log M + M \log^2 M \log n)$ by Theorem 6.3.6. □

CHAPTER 7

THE LINE-CONSTRAINED K-MEDIAN, K-MEANS, AND K-CENTER PROBLEMS IN THE PLANE

It has been known that the (weighted) k-median, k-means, and k-center problems in the plane are NP-hard [17–19]. In this chapter, we study these problems with an additional constraint that the sought k facilities must be on a given line. We present efficient algorithms for various distance metrics such as L_1, L_2, L_∞. The results in this chapter have been published in [60, 84].

7.1 Introduction

For any point p, denote by $x(p)$ and $y(p)$ its x- and y-coordinates, respectively. For any two points p and q, denote by $d(p, q)$ the distance between p and q. Depending on the distance metrics, $d(p, q)$ may refer to the L_1 distance, i.e., $|x(p) - x(q)| + |y(p) - y(q)|$, or the L_2 distance, i.e., $\sqrt{(x(p) - x(q))^2 + (y(p) - y(q))^2}$, or the L_∞ distance, i.e., $\max\{|x(p) - x(q)|, |y(p) - y(q)|\}$. For convenience, we define the L_2^2 *distance metric*[1] as $(x(p) - x(q))^2 + (y(p) - y(q))^2$.

Let P be a set of n points in the plane, and each point $p \in P$ has a weight $w(p) > 0$. The goal of the k-median (resp., k-center) problem is to find a set Q of k points (called *facilities*) in the plane such that $\sum_{p \in P}[w(p) \cdot \min_{q \in Q} d(p, q)]$ (resp., $\max_{p \in P}[w(p) \cdot \min_{q \in Q} d(p, q)]$) is minimized. With our above definition on the L_2^2 metric, the k-means problem is actually the k-median problem under the L_2^2 metric.

If all points of Q are required to be on a given line, denoted by χ, then we refer to the corresponding problems as *line-constrained* or simply *constrained* k-median, k-means, and k-center problems. In the following, we assume χ is the x-axis and the points of P have been sorted by their x-coordinates.

[1] Note that the L_2^2 distance is actually not a metric because the triangle inequality does not hold; we use "metric" here merely for the reference purpose.

Table 7.1. Summary of our results, where $\tau = \min\{n\sqrt{k \log n}, n2^{O(\sqrt{\log k \log\log n})}\}$.

	constrained k-median	**constrained k-center**
L_1	$O(\min\{nk, \tau \log n\})$, and $O(\tau)$ for the unweighted case	$O(n \log n)$, and $O(n)$ for the unweighted case
L_2	unsolvable	$O(n \log n)$
L_∞	$O(\min\{nk \log n, \tau \log^2 n\})$	$O(n \log n)$, and $O(n)$ for the unweighted case
L_2^2	$O(\min\{nk, \tau\})$ (i.e., the constrained k-means)	not applicable

Table 7.1 summarizes our results in this chapter. Throughout the chapter, we always let $\tau = \min\{n\sqrt{k \log n}, n2^{O(\sqrt{\log k \log\log n})}\}$. For the constrained k-median, we give algorithms for the L_1 and L_∞ metrics, with running time $O(\min\{nk, \tau \log n\})$ and $O(\min\{nk \log n, \tau \log^2 n\})$, respectively. The L_1 unweighted version where all points of P have the same weight can also be solved in $O(\tau)$ time. These time bounds almost match those of the best algorithms for the one-dimensional k-median problems. Note that the L_2 version of the constrained k-median has been shown unsolvable due to the computation challenge even for $k = 1$ [85]. For the constrained k-means, we give an $O(\min\{nk, \tau\})$ time algorithm. For the constrained k-center, our algorithms run in $O(n \log n)$ time for all three metrics, and in $O(n)$ time for the unweighted version under L_1 and L_∞ metrics. These k-center results are optimal.

Our results show that although the 2D versions of these problems are hard, their "1.5D" versions are "easy". A practical example in which the facilities are restricted to lie along a line is that we want to build some supply centers along a highway or railway (although a highway or railway may not be a straight line, it may be considered straight in each local area). Other relevant examples may include building partial delivery stations along an oil or gas transportation pipeline.

7.1.1 Previous work

The L_1 and L_2 k-median and k-center problems in the plane are NP-hard [19], and so as the L_∞ k-center problem [17]. In the one-dimensional space, however, both problems are solvable in polynomial time: For k-median, the best-known algorithms run

in $O(nk)$ time [86,87] or in $O(\tau \log n)$ time [88]; for k-center, the best-known algorithms run in $O(n \log n)$ time [41,43,89].

The k-means problem in the plane is also NP-hard [18]. Heuristic and approximation algorithms have been proposed, e.g., see [90–93].

The unweighted versions of the constrained k-center were studied before. The L_2 case was first proposed and solved in $O(n \log^2 n)$ time by Brass *et al.* [58] and later was improved to $O(n \log n)$ time by Karmakar *et al.* [59]. Algorithms of $O(n \log n)$ time were also given in [58] for L_1 and L_∞ metrics; note that even the points are given sorted, the above algorithms [58] still run in $O(n \log n)$ time. In addition, Brass *et al.* [58] also gave interesting and efficient algorithms for other two variations of the unweighted k-center problems, i.e., the line χ is not fixed but its slope is fixed, or χ is arbitrary. To the best of our knowledge, we are not aware of any previous work on the weighted versions of the constrained k-median and k-center problems studied in this chapter.

Efficient algorithms have been given for other special cases. When $k = 1$, Megiddo [27] solved the unweighted L_2 1-center problem in $O(n)$ time. Hurtado [94] gave an $O(n + m)$ time algorithm for the unweighted L_2 1-center problem with the center restricted in a given convex polygon of m vertices. For $k = 2$, Chan [70] proposed an $O(n \log^2 n \log^2 \log n)$ time for the unweighted L_2 2-center problem and another randomized algorithm; if the points are in convex positions, the same problem can be solved in $O(n \log^2 n)$ time [95]. The L_2 1-median problem is also known as the Weber problem and no exact algorithm is known for it (and even for the constrained version) [85].

Alt *et al.* [96] studied a somewhat similar problem as our unweighted constrained problems, where the goal is to find a set of disks whose union covers all points and whose centers must be on a given line such that the *sum* of the radii of all disks is minimized, and they gave an $O(n^2 \log n)$ time algorithm [96]. Note that this problem is different from our k-median, k-means, or k-center problems.

Note that the unweighted k-center and k-median problems can be considered as geometric covering problems, i.e., cover the points in P by using k diamonds, discs, and squares corresponding to the L_1, L_2, and L_∞ metrics, respectively.

7.1.2 Our Approaches

Suppose $p_1, p_2, \ldots, p_n$ are the points of P ordered by increasing x-coordinate. We discover an easy but crucial observation: For every problem studied in this chapter, there always exists an optimal solution in which the points of P "served" by the same facility are consecutive in the above index order.

For convenience of discussion, in the following we will refer to the k-means problem as the k-median problem under the L_2^2 metric.

Based on the above observation, for the constrained k-median, for all metrics (i.e., L_1, L_2, L_2^2, and L_∞), by modeling the problem as finding a minimum weight k-link path in a DAG G. Furthermore, we prove that the weights of the edges of G satisfy the concave Monge property and thus efficient techniques [97, 98] can be used. One challenging problem for the above algorithmic scheme is that we need to design a data structure to quickly compute any graph edge weight (i.e., given any i and j with $i \leq j$, compute the optimal objective value for the constrained 1-median problem on the points $p_i, p_{i+1}, \ldots, p_j$).

For L_2^2 metric (i.e., the k-means), we build such a data structure in $O(n)$ time that can answer each query in $O(1)$ time. For L_∞ metric, we build such a data structure in $O(n \log n)$ time that can answer each query in $O(\log^2 n)$ time. Combining these data structures with the above algorithmic scheme, we can solve the L_2^2 and L_∞ cases. In addition, based on interesting observations, we give another algorithm for the L_∞ case that is faster than the above scheme for a certain range of values of k. For L_1 metric, instead of using the above algorithmic scheme, we reduce the problem to the one-dimensional k-median problem and then the algorithms in [86–88] can be applied.

For the constrained k-center, to solve the L_2 case, we generalize the $O(n \log n)$ time algorithm in [59] for the unweighted version. In fact, similar approaches can also solve the L_1 and L_∞ cases. However, since the algorithm uses Cole's parametric search [43], which is complicated and involves large constants and thus is only of theoretical interest, we design another $O(n \log n)$ time algorithms for the L_1 and L_∞ cases, without using parametric search.

In addition, for the unweighted L_1 and L_∞ cases, due to the above crucial observation, our linear time algorithm hinges on the following efficient data structures. With

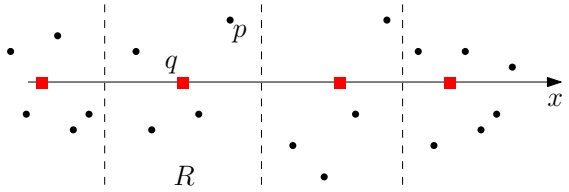

Figure 7.1. Illustrating the proof of Lemma 7.2.1. The (red) squared points are facilities and the vertical dashed lines pass through the midpoints of adjacent facilities.

$O(n)$ time preprocessing, for any query $i \leq j$, we can solve in $O(1)$ time the constrained L_1 and L_∞ 1-median problems on the points $p_i, p_{i+1}, \ldots, p_j$.

Note that our algorithms for the L_2 and L_2^2 metrics work for any arbitrary line χ (but χ must be given as input). However, since the distances under L_1 and L_∞ metrics are closely related to the orientation of the coordinate system, our algorithms for them only work for horizontal lines χ.

The rest of the chapter is organized as follows. In Section 7.2, we introduce some notations and observations. In Sections 7.3 and 7.4, we present our algorithms for the constrained k-median (including the k-means) and k-center problems, respectively. Again, in the following we will consider the k-means problem as the k-median problem under the L_2^2 metric.

7.2 Preliminaries

For simplicity of discussion, we assume no two points in P have the same x-coordinate. Let $p_1, p_2, \ldots, p_n$ be the points of P ordered by increasing x-coordinate. Define $P(i,j) = \{p_i, p_{i+1}, \ldots, p_j\}$ for any $i \leq j$. For any $1 \leq i \leq n$, we also use x_i, y_i, and w_i to refer to $x(p_i)$, $y(p_i)$, and $w(p_i)$, respectively.

For any facility set Q and any point p, let $d(p,Q) = \min_{q \in Q} d(p,q)$. For any point $p \in P$, if $d(p,Q) = d(p,q)$ for some facility point $q \in Q$, then we say p is "served" by q. We call $\sum_{p \in P}[w(p) \cdot d(p,Q)]$ and $\max_{p \in P}[w(p) \cdot d(p,Q)]$ the *objective value* of the k-median and k-center problems, respectively.

The following is the crucial lemma mentioned in Section 7.1.2.

Lemma 7.2.1. *For each of the constrained k-median and k-center problems of any metric (i.e., L_1, L_2, L_2^2, or L_∞), there must exist an optimal solution in which the points of P*

served by the same facility are consecutive in their index order.

Proof. We first consider the constrained k-median, for all metrics. Let Q be the facility set in any optimal solution. Let $q_1, q_2, \ldots, q_k$ be the facilities of Q sorted from left to right. For any $1 \leq i \leq k-1$, let l_i be the vertical line through the midpoint of the line segment $\overline{q_i q_{i+1}}$ (e.g., see Fig. 7.1). The above $k-1$ lines partition the plane into k strips (the leftmost and rightmost strips are two half-planes; in fact, this is exactly the Voronoi diagram of the k facilities) and each strip contains a single facility. Consider any strip R. Suppose R contains a facility q. It is easy to see that for any point $p \in P$ that is contained in R, it holds that $d(p, Q) = d(p, q)$ for any metric (e.g., see Fig. 7.1), implying that p should be served by q. Of course, if p is on the boundary of two strips, p can be served by either of the two facilities contained in the two strips. Clearly, the points of P contained in R are consecutive in their index order. Therefore, the lemma is proved for the constrained k-center problem for any metric.

For the constrained k-center, the proof is exactly the same as above and we omit it. The lemma thus follows. □

For any $i \leq j$, consider the constrained 1-median problem on $P(i, j)$; denote by $f(i, j)$ the facility in an optimal solution and define $\alpha(i, j)$ to be the objective value of the optimal solution, i.e., $\alpha(i, j) = \sum_{t=i}^{j}[w_t \cdot d(p_t, f(i, j))]$. We call $f(i, j)$ the *constrained median* of $P(i, j)$. In the case that $f(i, j)$ is not unique, we let $f(i, j)$ refer to the leftmost such point.

According to Lemma 7.2.1, solving the constrained k-median problem is equivalent to partitioning the sequence $p_1, p_2, \ldots, p_n$ into k subsequences such that the sum of the α values of all these subsequences is minimized. Formally, we want to find $k-1$ indices $i_0 < i_1 < i_2 < \cdots < i_{k-1} < i_k$, with $i_0 = 0$ and $i_k = n$, such that $\sum_{j=1}^{k} \alpha(i_{j-1} + 1, i_j)$ is minimized. There are also similar observations for the constrained k-center problem. As will be seen later, these observations are quite useful for our algorithms.

For any point p on the x-axis, for convenience, we also use p to denote its x-coordinate. For example, if two points p and q are on the x-axis, then $p < q$ means that p is strictly to the left of q. For any value x, we sometime also use x to refer to the point on the x-axis with x-coordinate x.

7.3 The Constrained k-Median

This section presents our algorithms for the constrained k-median under L_1, L_∞, and L_2^2 metrics.

We first propose an algorithmic scheme in Section 7.3.1 that works for any metric. To use the scheme, one has to design a data structure for computing $\alpha(i, j)$ for any query $i \leq j$. In Section 7.3.2, we design such a data structure for L_∞ metric, and thus solves the L_∞ case. In addition, we give another algorithm that is faster than the scheme for a certain range of values of k. In Section 7.3.3, we solve the L_1 case; instead of using the above scheme, we get a better result by reducing it to the one-dimensional problem. In Section 7.3.4, we solve the L_2^2 case (i.e., the k-means) by designing an efficient data structure for the above algorithmic scheme.

7.3.1 An Algorithmic Scheme for All Metrics

In this subsection, unless otherwise stated, all notations involving distances, e.g., $d(p, q)$, $\alpha(i, j)$, can use any distance metric (i.e., L_1, L_2, L_2^2, and L_∞).

In light of our observations in Section 7.2, we will reduce the problem to finding a minimum weight k-link path in a DAG G. Further, we will show that the edge weights of G satisfy the concave Monge property and then efficient algorithms [97–99] can be used. Below, we first define the graph G.

For each point $p_i \in P$, recall that $x_i = x(p_i)$ and we also use x_i to denote the projection of p_i on the x-axis. The vertex set of G consists of $n+1$ vertices $v_0, v_1, \ldots, v_n$ and one can consider each v_i corresponding to a point between x_i and x_{i+1} (v_0 is to the left of x_1 and v_n is to the right of x_n); e.g., see Fig.7.2. For any i and j with $0 \leq i \leq j \leq n$, we define a directed edge $e(i, j)$ from v_i to v_j, and the weight of the edge, denoted by $w(i, j)$, is defined to be $\alpha(i + 1, j)$ (if we view v_i and v_j as two points on the x-axis as above, then $\overline{v_i v_j}$ contains the points $x_{i+1}, x_{i+2}, \ldots, x_j$). Clearly, G is a directly acyclic graph (DAG).

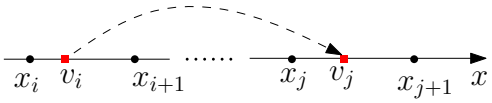

Figure 7.2. Illustrating an edge of G from v_i to v_j. It is the two (red) squared points.

A path in G is a *k-link* path if it has k edges. The *weight* of any path is the sum of the weights of all edges of the path. A *minimum weight k-link path* from v_0 to v_n in G is a k-link path that has the minimum weight among all k-link paths from v_0 to v_n. Note that any k-link path from v_0 to v_n in G corresponds to a partition of the points in P into k subsequences. According to our observations in Section 7.2 and the definition of G, the following lemma is self-evident.

Lemma 7.3.1. *A minimum weight k-link path π from v_0 to v_n in G corresponding to an optimal solution OPT of the constrained k-median problem on P. Specifically, the objective value of OPT is equal to the weight of π, and for each edge $e(v_i, v_j)$ of π, there is a corresponding facility serving all points of $P(i+1, j)$ in OPT.*

Recall that $f(i, j)$ is the leftmost constrained median of $P(i, j)$ for any $i \leq j$.

Lemma 7.3.2. *For any metric, the weights of the edges of G satisfy the concave Monge property, i.e., $w(i,j)+w(i+1,j+1) \leq w(i,j+1)+w(i+1,j)$ holds for any $1 \leq i < j \leq n$.*

Proof. Recall that $w(i, j) = \alpha(i+1, j)$ for any $i < j$. To facilitate using indices, we will prove that for any $i \leq j$, $w(i-1, j) + w(i, j+1) \leq w(i-1, j+1) + w(i, j)$ holds, i.e., $\alpha(i, j) + \alpha(i+1, j+1) \leq \alpha(i, j+1) + \alpha(i+1, j)$, for any metric.

Note that both $f(i, j+1) \leq f(i+1, j)$ and $f(i, j+1) > f(i+1, j)$ are possible. In the following, we will show that $\alpha(i, j) + \alpha(i+1, j+1) \leq \alpha(i, j+1) + \alpha(i+1, j)$ holds in either case.

1. Consider the case $f(i, j+1) \leq f(i+1, j)$.

   According to the definition, $\alpha(i, j)$ is the optimal objective solution for the 1-median problem on the points in $P(i, j)$. If we place a facility at $f(i, j+1)$ to serve all points in $P(i, j)$, then the objective value is $\sum_{t=i}^{j} w_t d(p_t, f(i, j+1))$, which is equal to $\alpha(i, j+1) - w_{j+1} d(p_{j+1}, f(i, j+1))$ since $\alpha(i, j+1) = \sum_{t=i}^{j+1} w_t d(p_t, f(i, j+1))$. Because $\alpha(i, j)$ is the optimal objective value for $P(i, j)$, it holds that $\alpha(i, j) \leq \sum_{t=i}^{j} w_t d(p_t, f(i, j+1))$. Therefore, we obtain

$$\alpha(i, j) \leq \alpha(i, j+1) - w_{j+1} d(p_{j+1}, f(i, j+1)). \tag{7.1}$$

Similarly, $\alpha(i+1, j+1)$ is the optimal objective value for the 1-median problem on $P(i+1, j+1)$. If we place a facility at $f(i+1, j)$ to serve all points of $P(i+1, j+1)$, then the objective value is $\sum_{t=i+1}^{j+1} w_t d(p_t, f(i+1, j))$, which is exactly equal to $\alpha(i+1, j) + w_{j+1} d(p_{j+1}, f(i+1, j))$ since $\alpha(i+1, j) = \sum_{t=i+1}^{j} w_t d(p_t, f(i+1, j))$. Because $\alpha(i+1, j+1)$ is the optimal objective value for $P(i+1, j+1)$, it holds that $\alpha(i+1, j+1) \leq \sum_{t=i+1}^{j+1} w_t d(p_t, f(i+1, j))$. Therefore, we obtain

$$\alpha(i+1, j+1) \leq \alpha(i+1, j) + w_{j+1} d(p_{j+1}, f(i+1, j)). \tag{7.2}$$

Recall that $f(i, j+1) \leq f(i+1, j)$ in this case. Since $f(i+1, j)$ is the leftmost constrained k-median of $P(i+1, j)$, it can be easily shown that $f(i+1, j) \leq x_j \leq x_{j+1}$ and we omit the detailed proof. Due to $f(i, j+1) \leq f(i+1, j) \leq x_{j+1}$, we obtain the following for any metric

$$d(p_{j+1}, f(i+1, j)) \leq d(p_{j+1}, f(i, j+1)). \tag{7.3}$$

Combining the three inequalities (7.1), (7.2), and (7.3), since all point weights are positive, we obtain $\alpha(i, j) + \alpha(i+1, j+1) \leq \alpha(i, j+1) + \alpha(i+1, j)$.

2. Consider the other case $f(i, j+1) > f(i+1, j)$. Some proof techniques are similar to the first case and we briefly discuss them.

   Note that $\alpha(i, j)$ is the optimal objective value for the 1-median problem on $P(i, j)$. If we place a facility at $f(i+1, j)$ to serve all points of $P(i, j)$, then the objective value is $\sum_{t=i}^{j} w_t d(p_t, f(i+1, j))$, which is exactly equal to $\alpha(i+1, j) + w_i d(p_i, f(i+1, j))$. Therefore, we obtain

   $$\alpha(i, j) \leq \alpha(i+1, j) + w_i d(p_i, f(i+1, j)). \tag{7.4}$$

   Similarly, $\alpha(i+1, j+1)$ is the optimal objective value for the 1-median problem on $P(i+1, j+1)$. If we place a facility at $f(i, j+1)$ to serve all points in $P(i+1, j+1)$,

then the objective value is $\sum_{t=i+1}^{j+1} w_t d(p_t, f(i, j+1))$, which is exactly equal to $\alpha(i, j+1) - w_i d(p_i, f(i, j+1))$. Therefore, we obtain

$$\alpha(i+1, j+1) \leq \alpha(i, j+1) - w_i d(p_i, f(i, j+1)). \tag{7.5}$$

Hence, if we can obtain the following

$$d(p_i, f(i+1, j)) \leq d(p_i, f(i, j+1)), \tag{7.6}$$

then combining Inequalities (7.4) and (7.5), we can prove that $\alpha(i, j) + \alpha(i+1, j+1) \leq \alpha(i, j+1) + \alpha(i+1, j)$.

Recall that $f(i+1, j) < f(i, j+1)$ in this case. One can easily prove that $x_i \leq x_{i+1} \leq f(i+1, j)$ holds for the L_1, L_2, and L_2^2 metrics, and thus Inequality (7.6) holds.

In the L_∞ metric, however, $x_i \leq f(i+1, j)$ may not be true. The reason is as follows. For any point p_t, let I_t be the interval on the x-axis centered at x_t with length $|y_t|$ (i.e., the absolute value of the y-coordinate of p_t). It can be easily seen that the points of I_t have the same L_∞ distance to p_t and the points of I_t are also the closest points to p_t on the x-axis. Let $I(i+1, j)$ be the intersections of all intervals I_t for $t = i+1, i+2, \ldots, j$. Hence, if $I(i+1, j) \neq \emptyset$, then $f(i+1, j)$ is the leftmost point of $I(i+1, j)$. One can easily draw an example in which $x_i > f(i+1, j)$.

If $x_i \leq f(i+1, j)$ holds, then we can obtain Inequality (7.6) and thus prove the lemma. In the following, we assume that $x_i > f(i+1, j)$.

First, we claim $I(i+1, j) \neq \emptyset$. Suppose to the contrary that $I(i+1, j) = \emptyset$. Then, there is at least one point $p_h \in P(i+1, j)$ such that I_h does not contain $f(i+1, j)$. Due to $x_i > f(i+1, j)$, all points of $P(i+1, j)$ have x-coordinates strictly larger than $f(i+1, j)$. Hence, if we move $f(i+1, j)$ rightwards for an infinitesimal distance, p_h will have the distance $d(p_h, f(i+1, j))$ strictly decrease, which makes the value $\sum_{t=i+1}^{t} w_t d(p_t, f(i+1, j))$ strictly decrease, contradicting

with that the original $f(i+1,j)$ is the constrained median of $P(i+1,j)$. The claim is thus proved.

Since $I(i+1,j) \neq \emptyset$, according to our above discussion, $f(i+1,j)$ is the leftmost point of $I(i+1,j)$. In fact, any point of $I(i+1,j)$ is a constrained median of $P(i+1,j)$. By its definition, the interval $I(i+1,j)$ must contain x_i due to $f(i+1,j) < x_i$. Therefore, x_i is also a constrained median of $P(i+1,j)$. Now we let $f(i+1,j)$ temporarily refer to x_i. By exactly the same analysis as above, we can still obtain Inequality (7.4).

Now consider $f(i,j+1)$. If $f(i,j+1) \geq x_i$, then Inequality (7.6) is obtained since $f(i+1,j)$ is now x_i, and thus the lemma is proved.

If $f(i,j+1) < x_i$, then using the similar analysis as above, we can also obtain that x_i is a constrained median of $P(i,j+1)$. Hence, by letting $f(i,j+1)$ refer to x_i, we can still obtain Inequality (7.5). Since now both $f(i+1,j)$ and $f(i,j+1)$ are x_i, Inequality (7.6) trivially holds and the lemma is proved.

Therefore, for the L_∞ metric, we also obtain $\alpha(i,j)+\alpha(i+1,j+1) \leq \alpha(i,j+1)+\alpha(i+1,j)$.

In summary, in any case, we obtain $\alpha(i,j)+\alpha(i+1,j+1) \leq \alpha(i,j+1)+\alpha(i+1,j)$ for any metric, and the lemma is thus proved. □

By Lemma 7.3.2, we can apply the algorithm in [97–99]. Assuming the weight of each graph edge $w(i,j)$ can be obtained in $O(1)$ time, the algorithms in [97] and [98] can compute a minimum weight k-link path from v_0 to v_n in $O(n\sqrt{k \log n})$ time and $O(n2^{O(\sqrt{\log k \log\log n})})$ time, respectively. Further, as indicated in [97], by using dynamic programming and applying the technique in [99], such a path can also be computed in $O(nk)$ time. In our problem, to compute each $w(i,j)$ is essentially to compute $\alpha(i+1,j)$. Therefore, we can obtain the following result.

Theorem 7.3.3. *For any metric (i.e., L_1, L_2, L_2^2, or L_∞), if we can build a data structure in $O(T)$ time that can compute $\alpha(i,j)$ in $O(\sigma)$ time for any query $i \leq j$, then we can solve the constrained k-median problem in $O(T + \sigma \cdot \min\{nk, \tau\})$ time, where $\tau = \min\{\sqrt{nk \log n}, n2^{O(\sqrt{\log k \log\log n})}\}$.*

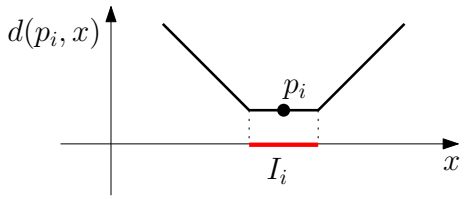

Figure 7.3. Illustrating the function $d(p_i, x)$. The (red) thick segment on the x-axis is I_i.

7.3.2 The Constrained k-Median under the L_∞-Metric

In this section, we present two algorithms for the L_∞ metric. The first algorithm, which runs in $O(\min\{nk, \tau\} \cdot \log^2 n)$ time, utilizes our result in Theorem 7.3.3, and the second algorithm, which runs in $O(nk \log n)$ time, uses a dynamic programming approach. Our second algorithm is actually based on some observations we found for the first algorithm and also uses some results of the first algorithm. In the sequel, we successively present the two algorithms.

In this section, all notations related to distances use the L_∞ metric.

The first algorithm

For the first algorithm, our main goal is to prove the following Lemma 7.3.4. Consequently, by Theorem 7.3.3, we can solve the L_∞ case in $O(\min\{nk, \tau\} \cdot \log^2 n)$ time.

Lemma 7.3.4. *For the L_∞ metric, a data structure can be constructed in $O(n \log n)$ time that can answer each $\alpha(i, j)$ query in $O(\log^2 n)$ time.*

We first introduce some notations. For any point p_i, let I_i denote the interval on the x-axis centered at x_i with length $|y_i|$ (i.e., the absolute value of the y-coordinate of p_i). Note that the points of I_i have the same (L_∞) distance to p_i. Consider $d(p_i, x)$ as a function of a point x on the x-axis. As x changes from $-\infty$ to $+\infty$, $d(p_i, x)$ first decreases and then does not change when $x \in I_i$ and finally increases (e.g., see Fig. 7.3). Consider any two indices $i \leq j$. Let $E(i, j)$ be the set of the endpoints of all intervals I_t for $i \leq t \leq j$. For any point x on the x-axis, define $\phi(i, j, x) = \sum_{t=i}^{j} w_t d(p_t, x)$. By the definition of $f(i, j)$, $\phi(i, j, x)$ is minimized at $x = f(i, j)$ and $\alpha(i, j) = \phi(i, j, f(i, j))$. Lemma 7.3.5 is crucial for computing $\alpha(i, j)$.

Lemma 7.3.5. *The function $\phi(i,j,x)$ is a continuous piecewise linear function whose slopes change only at the points of $E(i,j)$. Further, there exist two points in $E(i,j)$, denoted by x' and x'' with $x' \leq x''$ ($x' = x''$ is possible), such that as x increases from $-\infty$ to $+\infty$, $\phi(i,j,x)$ will strictly decrease when $x \leq x'$, and will be constant when $x \in [x', x'']$, and will strictly increase when $x \geq x''$.*

Proof. Consider any point x on the x-axis. We define three sets

$P_L(i,j,x) = \{p_t \mid i \leq t \leq j$ and I_t is strictly to the left of $x\}$,

$P_M(i,j,x) = \{p_t \mid i \leq t \leq j$ and I_t contains $x\}$, and

$P_R(i,j,x) = \{p_t \mid i \leq t \leq j$ and I_t is strictly to the right of $x\}$.

It is not difficult to see that the above three sets form a partition of $P(i,j)$. For any point p_t and any x, it can be verified that if $p_t \in P_L(i,j,x)$, then $d(p_t,x) = x - x_t$, if $x \in P_M(i,j,x)$, then $d(p_t,x) = |y_t|$, and if $I_t \in P_R(i,j,x)$, then $d(p_t,x) = -x + x_t$.

To simplify notations in this proof, when the context is clear, we will omit the indices i and j from some notations, e.g., we use $P_L(x)$, $P_M(x)$, $P_R(x)$, $\phi(x)$ to refer to $P_L(i,j,x)$, $P_M(i,j,x)$, $P_R(i,j,x)$, $\phi(i,j,x)$, respectively.

We have the following way to compute $\phi(x)$ (recall that for any point p, $x(p)$ and $y(p)$ are its x- and y-coordinates, respectively):

$$\begin{aligned}
\phi(x) &= \sum_{p \in P(i,j)} w(p)d(p,x) \\
&= \sum_{p \in P_L(x)} w(p)d(p,x) + \sum_{p \in P_M(x)} w(p)d(p,x) + \sum_{p \in P_R(x)} w(p)d(p,x) \\
&= \sum_{p \in P_L(x)} w(p)(x - x(p)) + \sum_{p \in P_M(x)} w(p)|y(p)| + \sum_{p \in P_R(x)} w(p)(-x + x(p)) \qquad (7.7) \\
&= x \cdot \Big[\sum_{p \in P_L(x)} w(p) - \sum_{p \in P_R(x)} w(p) \Big] \\
&\quad - \sum_{p \in P_L(x)} w(p)x(p) + \sum_{p \in P_M(x)} w(p)|y(p)| + \sum_{p \in P_R(x)} w(p)x(p).
\end{aligned}$$

For simplicity, we assume no two points in $E(i,j)$ are the same. Let $m = |E(i,j)|$. Suppose $x'_1, x'_2, \ldots, x'_m$ is the sorted list of $E(i,j)$. Let $x'_0 = -\infty$ and $x'_{m+1} = +\infty$. Consider any open interval (x'_t, x'_{t+1}) for $0 \leq t \leq m$. For any $x \in (x'_t, x'_{t+1})$, the three sets $P_L(x)$, $P_M(x)$, and $P_R(x)$ are the same, and thus, the coefficient of x in Equality

(7.7) is a constant and the last three terms are also constants. Hence, $\phi(x)$ is a linear function for $x \in (x'_t, x'_{t+1})$. To prove that $\phi(x)$ is a continuous function, it is sufficient to show that $\lim_{x \to x'_t} \phi(x) = \phi(x'_t)$ for any $1 \leq t \leq m$, which can be verified using Equation (7.7), and we omit the details.

The above proves $\phi(i, j, x)$ is a continuous piecewise linear function whose slopes change only at the points of $E(i, j)$.

As x increases from $-\infty$ to $+\infty$, $P_L(x)$ is monotonically increasing and $P_R(x)$ is monotonically decreasing, and thus, by Equality (7.7) the slope of $\phi(x)$ is monotonically increasing. Note that when $x \in (x'_0, x'_1)$, $P_L(x) = \emptyset$ and $P_R(x) = P(i, j)$, and thus the slope of $\phi(x)$ is negative; when $x \in (x'_m, x'_{m+1})$, $P_L(x) = P(i, j)$ and $P_R(x) = \emptyset$, and thus the slope of $\phi(x)$ is positive. Hence, as x increases from $-\infty$ to $+\infty$, at some moment, the slope of $\phi(x)$ changes from negative to zero and then from zero to positive, or from negative directly to positive. In either case, as the slope of $\phi(x)$ is monotonically increasing and only changes at the points of $E(i, j)$, there exists two points x' and x'' in $E(i, j)$ with $x' \leq x''$, such that the slope of $\phi(x)$ is negative for $x \in (-\infty, x')$, zero for $x \in (x', x'')$, and positive for $x \in (x'', +\infty)$. Since the function $\phi(x)$ is continuous, the lemma is proved. □

By Lemma 7.3.5, to compute $\alpha(i, j)$, which is the minimum value of $\phi(x, i, j)$, we can do binary search on the sorted list of $E(i, j)$, provided that we can compute $\phi(i, j, x)$ efficiently for any x. Clearly, $E(i, j) \subseteq E(1, n)$, and thus, we can also do binary search on the sorted list of $E(1, n)$ to compute $\alpha(i, j)$ for any query $i \leq j$. Hence, as preprocessing, we compute the sorted list of $E(1, n)$ in $O(n \log n)$ time since $|E(1, n)| = 2n$.

According to the above discussion, for any query (i, j) with $i \leq j$, if we can compute $\phi(i, j, x)$ in $O(\sigma')$ time for any x, then we can compute $\alpha(i, j)$ in $O(\sigma' \log n)$ time. The following Lemma 7.3.6 gives a data structure for answering $\phi(i, j, x)$ queries, which immediately leads to Lemma 7.3.4.

Lemma 7.3.6. *We can construct a data structure in $O(n \log n)$ time that can compute $\phi(i, j, x)$ in $O(\log n)$ time for any $i \leq j$ and x.*

Proof. Let T be a complete binary tree whose leaves correspond to the points of P from left to right. For each $1 \leq i \leq n$, the i-th leaf is associated with the function $w_i d(p_i, x)$,

which is actually $\phi(i,i,x)$. Consider any internal node v. Let the leftmost (resp., rightmost) leaf of the subtree rooted at v be the i-th (resp., j-th) leaf. We associate with v the function $\phi(i,j,x)$, and use $\phi_v(x)$ to denote the function. By Lemma 7.3.5, the combinatorial complexity of the function $\phi_v(x)$ is $O(j-i+1)$. Let u and w be v's two children. Suppose we have already computed the two functions $\phi_u(x)$ and $\phi_w(x)$; since essentially $\phi_v(x) = \phi_u(x) + \phi_w(x)$, we can compute $\phi_v(x)$ in $O(j-i+1)$ time. Therefore, we can compute the tree T in $O(n \log n)$ time in a bottom-up fashion.

Consider any query $i \leq j$ and $x = x'$ and the goal is to compute $\phi(i,j,x')$. By standard approaches, we first find $O(\log n)$ maximum subtrees such that the leaves of these subtrees are exactly the leaves from the i-th leaf to the j-th one. Let V be the set of the roots of these subtrees, and V can be found in $O(\log n)$ time by following the two paths from the root of T to the i-th leaf and the j-th leaf, respectively. Notice that $\phi(i,j,x') = \sum_{v \in V} \phi_v(x')$. For each $v \in V$, to compute the value $\phi_v(x')$, we can do binary search on the function $\phi_v(x)$ associated with v, which can be done in $O(\log n)$ time. In this way, since $|V| = O(\log n)$, we can compute $\phi(i,j,x')$ in overall $O(\log^2 n)$ time. We can avoid doing binary search on each node v of V by constructing a fractional cascading structure [49] on the functions $\phi_v(x)$ of the nodes of T. Using fractional cascading, we only need to do one binary search on the root of T, and then the values $\phi_v(x')$ for all nodes v of V can be computed in constant time each. The fractional cascading structure can be built in additional $O(n \log n)$ time [49].

As a summary, we can construct a data structure in $O(n \log n)$ time that can compute $\phi(i,j,x)$ in $O(\log n)$ time for any $i \leq j$ and x. □

By Theorem 7.3.3 and Lemma 7.3.4, we have the following result.

Lemma 7.3.7. *The L_∞ constrained k-median can be solved in $O(\min\{nk \log^2 n, \tau \log^2 n\}$ time.*

The second algorithm

In the sequel, we present our second algorithm for the L_∞ constrained k-median problem, which runs in $O(nk \log n)$ time. Our algorithm is based on the following Lemma 7.3.8.

Lemma 7.3.8. *For the L_∞ constrained k-median problem on P, there must exist an optimal solution in which the facility set Q is a subset of $E(1,n)$.*

Proof. As discussed in Section 7.2, solving the k-median problem is equivalent to partitioning the sequence of the points of P into k subsequences such that the sum of the values $\alpha(i,j)$ for all subsequences $P(i,j)$ is minimized. By Lemma 7.3.5, for each subsequence $P(i,j)$, there must be a point of $E(i,j)$ that is a constrained 1-median of $P(i,j)$. This implies that there must exist an optimal solution for the constrained k-median problem on P such that each facility is a point of $E(1,n)$. □

Based on Lemma 7.3.8, we develop a dynamic programming algorithm. Intuitively, we have a set of "candidate" facilities, and further, by Lemma 7.2.1, we only need to check these candidates from left to right. Our algorithm is similar to the algorithm in [87] for the one-dimensional k-median problem. Below, we first do a problem transformation.

For each point $p \in E(1,n)$, we assign a weight of zero to p. Let $S = E(1,n) \cup P$. Consider the L_∞ constrained k-median problem on S. Since the weight of each point of $E(1,n)$ is zero, an optimal solution to the problem on S is also an optimal solution to the problem on P, and vice versa. In the following, we will focus on solving the problem on S.

Note that $|S| = 3n$. To simplify the notation, we still use n to denote the size of S. For simplicity of discussion, we assume no two points of S have the same x-coordinate. Let $s_1, s_2, \ldots, s_n$ be the sorted list of S by their x-coordinates. For each $s_i \in S$, let s'_i be the projection of s_i on the x-axis. For any $i \leq j$, let $S(i,j) = \{s_i, s_{i+1}, \ldots, s_j\}$ and $S'(i,j) = \{s'_i, s'_{i+1}, \ldots, s'_j\}$. Let $S' = \{s'_i \mid 1 \leq i \leq n\}$. Note that $E(1,n) \subseteq S'$. According to Lemma 7.3.8, there must exist an optimal facility set Q that is a subset of S'.

For any $1 \leq r \leq k$ and $1 \leq j \leq n$, let $A(r,j)$ denote the optimal objective value of the subproblem on $S(j,n)$ using r facilities, i.e., $A(r,j)$ is the minimized value of $\sum_{p \in S(j,n)}[w(p) \cdot \min_{q \in Q'} d(q,p)]$ subject to $Q' \subseteq S'(j,n)$ and $|Q'| \leq r$; let $B(r,j)$ be the optimal objective value on the same subproblem with an additional condition that a facility must be placed at s'_j, i.e., $B(r,j)$ is the minimized value of $\sum_{p \in S(j,n)}[w(p) \cdot$

$\min_{q\in Q'} d(q,p)]$ subject to $s'_j \in Q' \subseteq S'(j,n)$ and $|Q'| \le r$. Then, we obtain the following recursions

$$B(r,j) = \min_{j<t\le n+1} \left\{ \sum_{i=j}^{t-1} w(s_i)d(s'_j,s_i) + A(r-1,t) \right\}, \quad r \ge 2,\ 1 \le j \le n, \tag{7.8}$$

$$A(r,j) = \min_{j\le t\le n} \left\{ \sum_{i=j}^{t} w(s_i)d(s'_t,s_i) + B(r,t) \right\}, \quad r \ge 1,\ 1 \le j \le n. \tag{7.9}$$

The base case is $B(1,j) = \sum_{i=j}^{n} d(s'_j,s_i)$, for $j = 1,2,\ldots,n$, and $A(r,n+1) = 0$ for any $r \ge 1$. Our goal is to compute $A(1,n)$.

By the above recursions, computing $A(1,n)$ can be easily done in $O(n^2k)$ time. As in [87], by showing a concave quadrangle inequality (note that it is not the same Monge property as in Lemma 7.3.2), we can solve the recursions in a more efficient way [100]. The details are given below.

For any $i \le j$, define $g(i,j) = \sum_{t=i}^{j} w(s_t)d(s'_i,s_t)$ and $g'(i,j) = \sum_{t=i}^{j} w(s_t)d(s'_j,s_t)$. Note that we can replace the first term in Equality (7.8) by $g(j,t-1)$ and replace the first term in Equality (7.9) by $g'(j,t)$. We can easily prove that the following concave quadrangle inequality: for any $1 \le a \le b \le c \le d \le n$, $g(a,c) + g(b,d) \le g(b,c) + g(a,d)$. To see this, we have $g(a,d) - g(a,c) = \sum_{t=c+1}^{d} w(s_t)d(s'_a,s_t)$ and $g(b,d) - g(b,c) = \sum_{t=c+1}^{d} w(s_t)d(s'_b,s_t)$. Clearly, due to $a \le b \le c$, $d(s'_a,s_t) \ge d(s'_b,s_t)$ for any $c+1 \le t$. Therefore, we obtain $g(b,d) - g(b,c) \le g(a,d) - g(a,c)$ and the above inequality holds. Similarly, we can prove that for any $1 \le a \le b \le c \le d \le n$, $g'(a,c) + g'(b,d) \le g'(b,c) + g'(a,d)$.

Therefore, if we can compute $g(i,j)$ and $g'(i,j)$ in $O(\sigma')$ time for any $i \le j$, then as in [87], by using the algorithm in [100] we can solve the recursions and compute $A(1,n)$ in $O(nk\sigma')$ time. To compute $g(i,j)$, an easy observation is that $g(i,j)$ is exactly $\phi(i,j,x)$ for $x = s'_i$, which can be computed in $O(\log n)$ time by Lemma 7.3.6 with $O(n \log n)$ time preprocessing. Similarly, $g'(i,j)$ is exactly $\phi(i,j,x)$ for $x = s'_j$ and can be computed in $O(\log n)$ time.

Therefore, we can solve the k-median problem on S in $O(nk \log n)$ time. Not that the algorithm in [87] for the one-dimensional k-median problem runs in $O(nk)$ time because in the 1D space we can easily compute each $g(i,j)$ or $g'(i,j)$ in constant time

with $O(n)$ time preprocessing.

Combining with Lemma 7.3.7, we obtain the following theorem.

Theorem 7.3.9. *The L_∞ constrained k-median can be solved in $O(\min\{nk \log n, \tau \log^2 n\})$ time.*

7.3.3 The Constrained k-Median Problem under the L_1-Metric

To solve the L_1 case, we could use the algorithmic scheme in Theorem 7.3.3. However, we get a better result by reducing the problem to the one-dimensional problem and then applying the algorithms in [86–88]. In this section, all notations related to distances use the L_1 metric.

Recall that our goal is to minimize $\sum_{i=1}^{n}[w_i \cdot d(p_i, Q)]$. Consider any point $p_i \in P$. For any point q on the x-axis, since $d(p_i, q)$ is the L_1 distance, we have $d(p_i, q) = d(x_i, q) + |y_i|$. Since all points of Q are on the x-axis, it holds that $d(p_i, Q) = \min_{q \in Q} d(p_i, q) = |y_i| + \min_{q \in Q} d(x_i, q)$. Therefore, we obtain $\sum_{i=1}^{n}[w_i \cdot d(p_i, Q)] = \sum_{i=1}^{n} w_i|y_i| + \sum_{i=1}^{n}[w_i \cdot d(x_i, Q)]$.

Note that once P is given, $\sum_{i=1}^{n} w_i|y_i|$ is constant, and thus, to minimize $\sum_{i=1}^{n}[w_i \cdot d(p_i, Q)]$ is to minimize $\sum_{i=1}^{n}[w_i \cdot d(x_i, Q)]$, which is the following *one-dimensional k-median problem*: Given a set of n points $P' = \{x_1, x_2, \ldots, x_n\}$ on the x-axis with each x_i having a weight $w(x_i) = w_i \geq 0$, find a set Q of k points on the x-axis to minimize $\sum_{i=1}^{n}[w_i \cdot d(x_i, Q)]$.

The above 1D k-median problem is a *continuous version* because each point of our facility set Q can be any point on the x-axis. There is also a *discrete version*, where Q is required to be a subset of P'. The algorithms given in [86–88] are for the discrete version and therefore we cannot apply their algorithms directly. Fortunately, due to some observations, we prove below that for our continuous version there always exists an optimal solution in which the set Q is a subset of P', and consequently we can apply the discrete version algorithms.

Consider any indices $i \leq j$. Let $P'(i, j) = \{x_i, x_{i+1}, \ldots, x_j\}$. As in the L_∞ case, for any point x on the x-axis, define $\phi(i, j, x) = \sum_{t=i}^{j} w_t d(x_t, x)$. The following lemma is similar in spirit to Lemma 7.3.5.

Lemma 7.3.10. *The function $\phi(i,j,x)$ is a continuous piecewise linear function whose slopes change only at the points of $P'(i,j)$. Further, there exist two points, denoted by x' and x'', in $P'(i,j)$ with $x' \leq x''$ ($x' = x''$ is possible), such that as x increases from $-\infty$ to $+\infty$, $\phi(i,j,x)$ will strictly decrease when $x \leq x'$, and will be constant when $x \in [x', x'']$, and will strictly increase when $x \geq x''$.*

Proof. Consider any point x on the x-axis. Define $P'_L(i,j,x) = \{x_t \mid i \leq t \leq j \text{ and } x_t \leq x\}$, and $P'_R(i,j,x) = \{x_t \mid i \leq t \leq j \text{ and } x_t > x\}$. Hence, the above two sets form a partition of $P'(i,j)$. For any point x_t, if $x_t \in P'_L(i,j,x)$, then $d(x_t,x) = x - x_t$, and $d(x_t,x) = x_t - x$ otherwise.

To simplify notations in this proof, when the context is clear, we will omit the indices i and j from some notations, e.g., we use $P'_L(x)$, $P'_R(x)$, $\phi(x)$ to refer to $P'_L(i,j,x)$, $P'_R(i,j,x)$, $\phi(i,j,x)$, respectively.

We have the following way to compute $\phi(x)$:

$$\begin{aligned}\phi(x) &= \sum_{p \in P'(i,j)} w(p)d(p,x) = \sum_{p \in P'_L(x)} w(p)(x-p) + \sum_{p \in P'_R(x)} w(p)(p-x) \\ &= x \cdot \Big[\sum_{p \in P'_L(x)} w(p) - \sum_{p \in P'_R(x)} w(p) \Big] - \sum_{p \in P'_L(x)} w(p)p + \sum_{p \in P'_R(x)} w(p)p.\end{aligned} \tag{7.10}$$

For ease of discussion, we let $x_{i-1} = -\infty$ and $x_{j+1} = +\infty$ temporarily for this proof. Consider any interval $[x_t, x_{t+1})$ for $i-1 \leq t \leq j$. For any $x \in [x_t, x_{t+1})$, the two sets $P'_L(x)$ and $P'_R(x)$ are the same, and thus, the coefficient of x in Equality (7.10) is a constant and the last two terms are also constants, and consequently $\phi(x)$ is a linear function for $x \in [x_t, x_{t+1})$. To prove that $\phi(x)$ is a continuous function, it is sufficient to show that $\lim_{x \to x_{t+1}} \phi(x) = \phi(x_{t+1})$, which can be verified by using Equation (7.10), and we omit the details.

The above proves that $\phi(x)$ is a continuous piecewise linear function whose slopes change only at the points of $P'(i,j)$.

As x increases from $-\infty$ to $+\infty$, $P'_L(x)$ is monotonically increasing and $P'_R(x)$ is monotonically decreasing, and thus the slope of $\phi(x)$ is monotonically increasing. When $x \in [x_{i-1}, x_i)$, then $P'_L(x) = \emptyset$ and $P'_R(x) = P'(i,j)$, and thus, the slope of $\phi(x)$ is negative; if $x \in [x_j, x_{j+1})$, then $P'_L(x) = P'(i,j)$ and $P'_R(x) = \emptyset$, and thus, the slope of

$\phi(x)$ is positive. Hence, as x increases from $-\infty$ to $+\infty$, at some moment, the slope of $\phi(x)$ changes from negative to zero and then from zero to positive, or from negative directly to positive. In either case, as the slope of $\phi(x)$ is monotonically increasing and only changes at the points of $P'(i,j)$, there exist two points x' and x'' in $P'(i,j)$ with $x' \leq x''$, such that the slope of $\phi(x)$ is negative when $x \in (-\infty, x')$, zero if when $x \in [x', x'')$, and positive when $x \in [x'', +\infty)$. Since the function $\phi(x)$ is continuous, the lemma is proved. □

Lemma 7.3.11. *For the 1D k-median problem on P', there must exist an optimal solution in which the facility set Q is a subset of P'.*

Proof. Similar as we discussed in Section 7.2, solving the 1D k-median problem is equivalent to partitioning the sequence of the points of P' into k subsequences such that the sum of the values $\alpha(i,j)$ for all subsequences $P'(i,j)$ is minimized.

By Lemma 7.3.10, for each subsequence $P'(i,j)$, there must be a point of $P'(i,j)$ that is a 1-median of $P'(i,j)$. This implies that there must exist an optimal solution for the k-median problem on P' in which each facility is a point of P', which leads to the lemma. □

In light of Lemma 7.3.11, we can apply the algorithms in [86–88] to solve the k-median problem on P'. The algorithms in [86, 87] run in $O(nk)$ time and the algorithm in [88] runs in $O(\tau \log n)$ time and $O(\tau)$ time for the unweighted case. As a summary, we have the following result.

Theorem 7.3.12. *The L_1 constrained k-median can be solved in $O(\min\{nk, \tau \log n\})$ time and the unweighted case can be solved in $O(\min\{nk, \tau\})$ time.*

7.3.4 The Constrained k-Median under the L_2^2-Metric (i.e., the Constrained k-Means)

In this section, we use the algorithmic scheme in Theorem 7.3.3 to solve the L_2^2 case in $O(\min\{nk, \tau\})$ time, by designing a data structure for answering the $\alpha(i,j)$ queries in the following Lemma 7.3.13.

Lemma 7.3.13. *For the L_2^2 metric, a data structure can be built in $O(n)$ time that can answer each $\alpha(i,j)$ query in $O(1)$ time.*

Proof. Consider any two indices $i \leq j$. For any point x on the x-axis, define $\phi(i,j,x) = \sum_{t=i}^{j} w_t d(p_t, x)$. By the definition of $f(i,j)$, $\phi(i,j,x)$ is minimized at $x = f(i,j)$ and $\alpha(i,j) = \phi(i,j,f(i,j))$. We further obtain the following

$$\begin{aligned}\phi(i,j,x) &= \sum_{t=i}^{j} w_t d(p_t,x) = \sum_{t=i}^{j} w_t[(x-x_t)^2 + y_t^2] = \sum_{t=i}^{j} w_t(x^2 - 2x_t x + x_t^2 + y_t^2) \\ &= \sum_{t=i}^{j} w_t \cdot x^2 - 2\sum_{t=i}^{j} w_t x_t \cdot x + \sum_{t=i}^{j} w_t(x_t^2 + y_t^2).\end{aligned} \tag{7.11}$$

Hence, $\phi(i,j,x)$ is a parabola opening up, which is minimized at $x = \frac{\sum_{t=i}^{j} w_t x_t}{\sum_{t=i}^{j} w_t}$, implying that $f(i,j) = \frac{\sum_{t=i}^{j} w_t x_t}{\sum_{t=i}^{j} w_t}$.

By Equality (7.11), to compute $\alpha(i,j)$, it is sufficient to compute $f(i,j)$ and the parabola function $\phi(i,j,x)$, and specifically, it is sufficient to compute the three values $\sum_{t=i}^{j} w_t$, $\sum_{t=i}^{j} w_t x_t$, and $\sum_{t=i}^{j} w_t(x_t^2 + y_t^2)$. To this end, we do the following preprocessing. For each $1 \leq i \leq n$, we compute $\sum_{t=1}^{i} w_t$, $\sum_{t=1}^{i} w_t x_t$, and $\sum_{t=1}^{i} w_t(x_t^2 + y_t^2)$. In this way, given any two indices $i \leq j$, we can obtain the above three values in constant time and thus compute $\alpha(i,j)$ in constant time. The preprocessing can be done in $O(n)$ time.

The lemma is thus proved. □

Hence, by Theorem 7.3.3 we obtain the following result.

Theorem 7.3.14. *The constrained k-median problem under the L_2^2 metric, which is also the constrained k-means problem, can be solved in $O(\min\{nk, \tau\})$ time.*

7.4 The Constrained k-Center

This section presents our algorithms for the constrained k-center for all metrics. We first give in Section 7.4.1 a linear time algorithm to solve the decision version of the problem for all metrics, which will be used in both Sections 7.4.2 and 7.4.3. In Section 7.4.2, we present an $O(n \log n)$ time algorithm for L_2 metric. In fact, similar algorithms also work for the other two metrics. However, since the algorithm uses Cole's parametric search [43], which is complicated and involves large constants and thus is

only of theoretical interest, in Section 7.4.3, we give another $O(n \log n)$ time algorithm for L_1 and L_∞ metrics, without using parametric search. Finally, in Section 7.4.4, we give an $O(n)$ time algorithm for the unweighted case under L_1 and L_∞ metrics.

7.4.1 A Decision Algorithm for All Metrics

In this subsection, unless otherwise stated, all notations related to distances are applicable to all metrics, i.e., L_1, L_2, and L_∞.

The *decision version* of the problem is as follows: given any value ϵ, determine whether there are a set Q of k facilities such that $\max_{p \in P}[w(p) \cdot d(p, Q)] \leq \epsilon$, and if yes, we call ϵ a *feasible* value. We let ϵ^* denote the optimal objective value, i.e., $\epsilon^* = \max_{p \in P}[w(p) \cdot d(p, Q)]$ for the facility set Q in any optimal solution. Hence, for any ϵ, it is a feasible value if and only $\epsilon \geq \epsilon^*$.

For any point $p_i \in P$, denote by $I(p_i, \epsilon)$ the set of points q on the x-axis such that $w_i d(p_i, q) \leq \epsilon$. Note that $I(p_i, \epsilon)$ is the intersection of the "disk" centered at p_i with radius ϵ / w_i (the "disk" is a diamond, a real circular disk, and a square under L_1, L_2, and L_∞ metrics, respectively). Hence, $I(p_i, \epsilon)$ is an interval and we refer to $I(p_i, \epsilon)$ as the *facility location interval* of p_i. Note that for any subset $P(i, j)$, if the intersection of all facility location intervals of $P(i, j)$ is not empty, then any point in the above intersection can be used as a facility to serve all points of $P(i, j)$ within weighted distance ϵ.

We say a point *covers* an interval on the x-axis if the interval contains the point. Let $I(P, \epsilon)$ be the set of all facility location intervals of P. According to the above discussion, to determine whether ϵ is a feasible value, it is sufficient to compute a minimum number of points that can cover all intervals of $I(P, \epsilon)$, which can be done in $O(n)$ time after the endpoints of all intervals of $I(P, \epsilon)$ are sorted [47]. The overall time for solving the decision problem is $O(n \log n)$ due to the sorting. Below, we give an $O(n)$ time algorithm, without sorting.

Similar to Lemma 7.2.1, if ϵ is a feasible value, then there exists a feasible solution in which each facility serves a set of consecutive points of P. Using this observation, our algorithm works as follows. We consider the intervals of $I(P, \epsilon)$ from $I(p_1, \epsilon)$ in the index order of p_i. We find the largest index j such that $\bigcap_{i=1}^{j} I(p_i, \epsilon)$ is not empty, and then we place a facility at any point in the above intersection to serve all points in $P(1, j)$.

Next, from $I(p_{j+1}, \epsilon)$, we find the next maximal subset of intervals whose intersection is not empty to place a facility. We continue this procedure until the last interval $I(p_n, \epsilon)$ has been considered. Clearly, the running time of the algorithm is $O(n)$. Let k' be the number of facilities that are placed in the above procedure. The value ϵ is a feasible value if and only if $k' \leq k$. Hence, we have the following result.

Lemma 7.4.1. *Given any value ϵ, we can determine whether ϵ is a feasible value in $O(n)$ time for any metric.*

7.4.2 The Algorithm for the L_2 Metric

In this subsection, all notations related to distances are for the L_2 metric. Our algorithm uses Cole's parametric search [43], which generalizes the $O(n \log n)$ time algorithm for the unweighted case [59].

For any ϵ and each $1 \leq i \leq n$, let $l_i(\epsilon)$ and $r_i(\epsilon)$ denote the left and right endpoints of $I(p_i, \epsilon)$, respectively. Recall that ϵ^* is the optimal objective value. Let $S = \{l_i(\epsilon^*), r_i(\epsilon^*) \mid 1 \leq i \leq n\}$. If we know the sorted lists of the values of S, then we can use our decision algorithm to compute an optimal facility set in $O(n)$ time. Although we do not know ϵ^*, we can still sort S by parametric search and our decision algorithm. In the parametric search, we will need to compare two values of S. Although we do not know ϵ^*, we can still resolve the comparison by using our decision algorithm in Lemma 7.4.1. The details are given below.

Let γ_1 and γ_2 be any two values of S whose relative order needs to be determined (i.e., determine whether $\gamma_1 \geq \gamma_2$ or $\gamma_1 \leq \gamma_2$). There are several cases.

1. If γ_1 and γ_2 are $l_i(\epsilon^*)$ and $r_i(\epsilon^*)$ for the same index i, then $l_i(\epsilon^*) \leq r_i(\epsilon^*)$ always holds.

2. If γ_1 and γ_2 are $r_i(\epsilon^*)$ and $r_j(\epsilon^*)$ for $i \neq j$, then we use the following approach. We consider $r_i(\epsilon)$ as a function of ϵ (e.g., see Fig. 7.4). First of all, it must hold that $\epsilon^*/w_i \geq |y_i|$ since otherwise the interval $I(p_i, \epsilon)$ would be empty and no facility would be able to sever p_i within the weighted distance ϵ^*. One can verify that for $\epsilon \geq w_i \cdot |y_i|$, we have $r_i(\epsilon) = x_i + \sqrt{(\epsilon/w_i)^2 - y_i^2}$. It can be verify that there is at most one root for $r_i(\epsilon) = r_j(\epsilon)$. Note that we can determine whether there

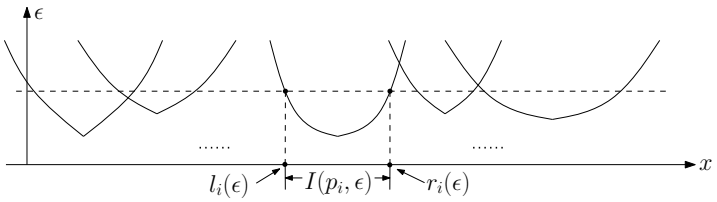

Figure 7.4. Illustrating the relationship between ϵ and $l_i(\epsilon)$ and $r_i(\epsilon)$ under the L_2 metric. Suppose the y-coordinate of the dashed horizontal line is ϵ; the interval $I(p_i, \epsilon)$ is shown on the x-axis.

is a root for $r_i(\epsilon) = r_j(\epsilon)$, and if yes, find the root, in constant time. If there is no root, then we can determine the relative order of $r_i(\epsilon^*)$ and $r_j(\epsilon^*)$ by assigning any value at least $\max\{w_i \cdot |y_i|, w_j \cdot |y_j|\}$ to ϵ, i.e., $r_i(\epsilon^*) \leq r_j(\epsilon^*)$ if and only if $r_i(\epsilon') \leq r_j(\epsilon')$ for any value $\epsilon' \geq \max\{w_i \cdot |y_i|, w_j \cdot |y_j|\}$. Otherwise, let ϵ' be the root. We can determine the relative order by using our decision algorithm to determine whether ϵ' is a feasibility value. Hence, the relative order of $r_i(\epsilon^*)$ and $r_j(\epsilon^*)$ can be determined by at most one feasibility test using our decision algorithm.

3. If γ_1 and γ_2 are $l_i(\epsilon^*)$ and $l_j(\epsilon^*)$ for $i \neq j$, then we use the same approach as the above second case to resolve the comparison.

4. Finally, suppose γ_1 and γ_2 are $l_i(\epsilon^*)$ and $r_j(\epsilon^*)$ for $i \neq j$. If $i < j$, then $l_i(\epsilon^*) \leq r_j(\epsilon^*)$ holds since $l_i(\epsilon^*) \leq x_i \leq x_j \leq r_j(\epsilon^*)$. Otherwise, it can be verified that there is one root for $l_i(\epsilon) = r_j(\epsilon)$. Hence, by one feasibility test using our decision algorithm, we can determine the relative order of $l_i(\epsilon^*)$ and $r_j(\epsilon^*)$.

The above shows that we can resolve the comparison of any two values in S by at most one feasibility test using our decision algorithm. Using Cole's parametric search [43], we can determine the sorted list of S by using only $O(\log n)$ feasibility test. Hence, the total running time of the algorithm is $O(n \log n)$.

Theorem 7.4.2. *The constrained k-center problem under the L_2 metric can be solved in $O(n \log n)$ time.*

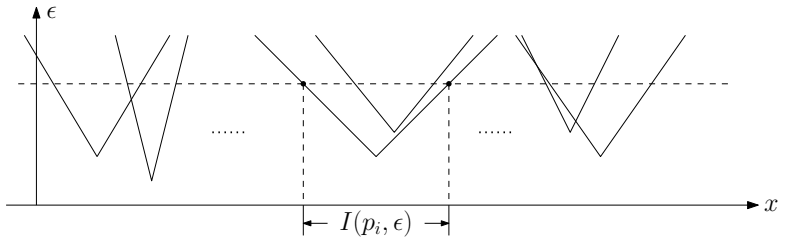

Figure 7.5. Illustrating the relationship between ϵ and $l_i(\epsilon)$ and $r_i(\epsilon)$ under the L_1 metric. Suppose the y-coordinate of the dashed horizontal line is ϵ; the interval $I(p_i, \epsilon)$ is shown on the x-axis.

7.4.3 The Algorithms for L_1 and L_∞ Metrics

We present $O(n \log n)$ time algorithms for the L_1 and L_∞ metrics, without using parametric search. We consider the L_1 case first.

We define $l_i(\epsilon)$ and $r_i(\epsilon)$ in the same way as in Section 7.4.2. We consider $l_i(\epsilon)$ and $r_i(\epsilon)$ as functions of ϵ (e.g., see Fig. 7.5). It can be verified that $r_i(\epsilon) = x_i + \epsilon/w_i - |y_i|$ defined on $\epsilon \geq w_i \cdot |y_i|$. Similarly, we have $l_i(\epsilon) = x_i - \epsilon/w_i + |y_i|$ defined on $\epsilon \geq w_i \cdot |y_i|$. Hence, each of $l_i(\epsilon)$ and $r_i(\epsilon)$ defines a half-lines. Let A be the set of the half-lines defined by $l_i(\epsilon)$ and $r_i(\epsilon)$ for all $i = 1, 2, \ldots, n$. As analyzed in [58] for the unweighted case, the optimal objective value ϵ^* must be the y-coordinate of an intersection of two half-lines of A. In fact, ϵ^* is the smallest feasible value among the y-coordinates of all intersections of the half-lines of A. Let $\mathcal{A}$ be the arrangement of the lines containing the half-lines of A. The intersection of two lines of $\mathcal{A}$ is called a *vertex*. Hence, ϵ^* is the smallest feasible value among the y-coordinates of the vertices of $\mathcal{A}$. Therefore, to solve the constrained k-center problem on P, it is sufficient to find the lowest vertex (denoted by v^*) of $\mathcal{A}$ whose y-coordinate is a feasible value (which is ϵ^*) and then apply our decision algorithm in Lemma 7.4.1 on ϵ^* to find an optimal facility set in additional $O(n)$ time.

To find such a vertex v^*, we use our decision algorithm and a line arrangement searching technique given in [46]. The technique gives the following result. Suppose there is a function $g : \mathbb{R} \to \{0, 1\}$, such that the description of g is unknown but it is known that g is monotonically increasing. Further, given any value y, we have a "black-box" that can evaluate $g(y)$ (i.e., determine whether $g(y)$ is 1 or 0) in $O(G)$ time, which we call the *g-oracle*. Let B be a set of n lines in the plane and let $\mathcal{B}$ denote

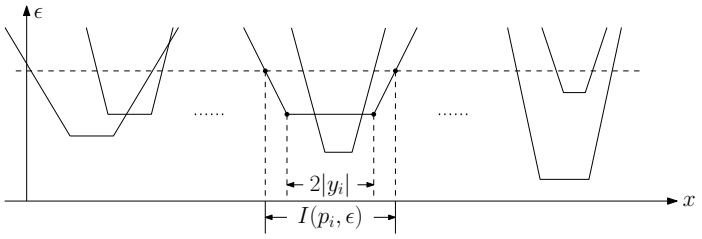

Figure 7.6. Illustrating the relationship between ϵ and $l_i(\epsilon)$ and $r_i(\epsilon)$ under the L_∞ metric. Suppose the y-coordinate of the dashed horizontal line is ϵ; the interval $I(p_i, \epsilon)$ is shown on the x-axis.

their arrangement. Note that $\mathcal{B}$ is not computed explicitly. For any vertex v of $\mathcal{B}$, let y_v be the y-coordinate of v. The *arrangement searching* is to find the lowest vertex v of $\mathcal{B}$ such that $g(y_v) = 1$. An $O((n + G) \log n)$ time algorithm is given in [46] to solve the arrangement searching problem by modifying the slope selection algorithm [51, 52], without using parametric search.

For our problem, to search v^* in the arrangement $\mathcal{A}$, we can define such a function g as follows. For any value y, $g(y) = 1$ if and only if y is a feasible value. Clearly, g is monotonically increasing since for any feasible value y, any value larger than y is also feasible. Hence, v^* is the lowest point in $\mathcal{A}$ with $g(y_{v^*}) = 1$. We use our decision algorithm in Lemma 7.4.1 as the g-oracle with $G = O(n)$. By the result in [46], we can compute v^* in $O(n \log n)$ time. Consequently, we obtain ϵ^*. An optimal facility set Q can be found by using our decision algorithm on ϵ^* in additional $O(n)$ time.

As a summary, we can solve the L_1 constrained k-center problem on P in $O(n \log n)$ time.

For the L_∞ case, the algorithm is similar. Under L_∞ metric, it can be verified that $r_i(\epsilon) = x_i + \epsilon/w_i$ and $l_i(\epsilon) = x_i - \epsilon/w_i$, both defined on $\epsilon \geq w_i \cdot |y_i|$ (e.g., see Fig. 7.6). Hence, each of $r_i(\epsilon)$ and $l_i(\epsilon)$ still defines a half-line, as in the L_1 case. Therefore, we can use the similar algorithm as in the L_1 case and solve the L_∞ case problem in $O(n \log n)$ time.

Theorem 7.4.3. *The L_1 and L_∞ constrained k-center problems can be solved in $O(n \log n)$ time, without using parametric search.*

7.4.4 The Unweighted Case under L_1 and L_∞ Metrics

We give an $O(n)$ time algorithm for the unweighted case under L_1 and L_∞ metrics.

We use the similar idea as discussed in Section 7.2 for the k-median problem. For any $i \leq j$, consider the constrained 1-center problem on the points in $P(i,j)$; denote by $g(i,j)$ the facility in an optimal solution and define $\beta(i,j)$ to be the objective value of the optimal solution, i.e., $\beta(i,j) = \max_{i \leq t \leq j} w_t d(p_t, g(i,j))$. We call $g(i,j)$ the *constrained center* of $P(i,j)$.

By Lemma 7.2.1, solving the constrained k-median problem is equivalent to partitioning the sequence $p_1, p_2, \ldots, p_n$ into k subsequences such that the maximum of the β values of all these subsequences is minimized. Formally, we want to find $k-1$ indices $i_0 < i_1 < i_2 < \cdots < i_{k-1} < i_k$, with $i_0 = 0$ and $i_k = n$, such that $\max_{j=1}^{k} \beta(i_{j-1}+1, i_j)$ is minimized. This is exactly the MIN-MAX PARTITION problem proposed in [101]. Based on Frederickson's algorithm [102], the following result is a re-statement of Theorem 2 in [101] with respect to our problem.

Lemma 7.4.4. [101] *If $\beta(i,j) \leq \beta(i',j')$ holds for any $1 \leq i' \leq i \leq j \leq j' \leq n$, then we have the following result. For any metric, suppose after $O(T)$ time preprocessing, we can compute $\beta(i,j)$ in $O(\sigma)$ time for any query $i \leq j$; then the constrained k-center problem can be solved in $O(T + n\sigma)$ time.*

Clearly, the condition in Lemma 7.4.4 holds for our problem, i.e., $\beta(i,j) \leq \beta(i',j')$ for any $1 \leq i' \leq i \leq j \leq j' \leq n$.

In Lemma 7.4.5, we give data structures for $\beta(i,j)$ queries under L_1 and L_∞ metrics.

Lemma 7.4.5. *For L_1 and L_∞ metrics, with $O(n)$ time preprocessing, we can compute $\beta(i,j)$ in constant time for any query $i \leq j$.*

Proof. Since we only consider the unweighted case, without loss of generality, we assume the weights of all points of P are 1. We begin with the L_1 case.

Consider the constrained 1-center problem on $P(i,j)$ for any $i \leq j$. We want to find a point q on the x-axis to minimize $\max_{i \leq t \leq j} d(p_t, q)$, or equivalently, we want to find a smallest diamond centered at a point on the x-axis such that all points of $P(i,j)$ are contained in the diamond (and the center of such a smallest diamond is $g(i,j)$, e.g., see Fig. 7.7). (We remark that this observation does not apply to the weighted case of

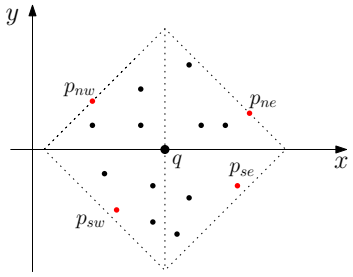

Figure 7.7. Illustrating the smallest diamond centered at the x-axis containing all points. The four red points are the four extreme points of the point set along the northeast, northwest, southwest, and southeast directions, and q is the center of the diamond.

the problem.) Note that such a smallest diamond must be determined by the extreme points of $P(i, j)$ along the following four directions: northeast, northwest, southwest, southeast. Specifically, let p_{ne} an extreme point of $P(i, j)$ along the northeast direction, and similarly, define p_{nw}, p_{sw}, and p_{se} for the other three directions. Then, the smallest diamond containing the above four points and centered at the x-axis is also the smallest diamond containing all points of $P(i, j)$ and centered at the x-axis. Since we only have four points, finding such a smallest diamond can be done in constant time and we omit the details.

According to the above discussion, to solve the 1-center problem on $P(i, j)$ and compute $\beta(i, j)$ for the L_1 metric, it is sufficient to find the four extreme points of $P(i, j)$ (after which $\beta(i, j)$ can be computed in constant time). To this end, we will use the range-maxima data structure [66, 67]. Given any arbitrary array A of size n, the data structure [66, 67] can compute in constant time the largest element and its index in the sub-array $A[i, \ldots, j]$ for any query $i \leq j$, with $O(n)$ time preprocessing. To solve our problem, we create an array A of size n, with $A[i]$ equal to the y-coordinate of the projection point of p_i on the line $y = x$. Given any query $i \leq j$, the extreme point p_{ne} corresponds to the largest element in the sub-array $A[i, \ldots, j]$. Therefore, the point p_{ne} can be found by a range-maxima query on A with $i \leq j$. We can find the other three extreme points in a similar manner.

As a summary, for the L_1 case, we can build a data structure in $O(n)$ time that can answer each $\beta(i, j)$ query in $O(1)$ time.

Next we discuss the L_∞ case. The idea is similar. Consider the constrained 1-center

problem on $P(i,j)$ for any $i \leq j$. We want to find a point q on the x-axis to minimize $\max_{i \leq t \leq j} d(p_t, q)$, or equivalently, we want to find a smallest axis-parallel square centered at a point on the x-axis such that all points of $P(i,j)$ are contained in the square (again, this does not apply to the weighted case of the problem). Note that such a smallest square is be determined by topmost, bottommost, leftmost, and rightmost points of $P(i,j)$. The leftmost and rightmost points of $P(i,j)$ are p_i and p_j, respectively. The topmost and bottommost points of $P(i,j)$ can be found using the range-maxima data structure as in the L_1 case, and we omit the details. Therefore, for the L_∞ case, we can also build a data structure in $O(n)$ time that can answer each $\beta(i,j)$ query in $O(1)$ time. □

Our linear time algorithm follows immediately from Lemmas 7.4.4 and 7.4.5.

Theorem 7.4.6. *The unweighted L_∞ and L_∞ constrained k-center problem can be solved in $O(n)$ time.*

CHAPTER 8

CONCLUSION AND FUTURE WORK

In this dissertation, we study a number of facility location problems, mainly on uncertain data. We present efficient algorithms for solving these problems.

In Chapter 2, we present an algorithm for solving the one-dimensional k-center problem on uncertain data. We consider a commonly used pdf—a piecewise constant function—for each uncertain point, which can be used to approximate other distributions. A useful observation discovered in Chapter 2 is that the expected distance from each uncertain point is a unimodal function (i.e., first monotonically decreasing and then increasing). Accordingly, we can use this property to design an algorithm based on parametric search. Further, we consider the discrete pdf for uncertain points. Our algorithm is based on a line arrangement searching technique. As shown in Chapter 2, our results almost match those for the corresponding deterministic k-center problems.

In Chapter 3, we present a linear-time algorithm for computing the center of uncertain points on tree networks. To this end, we propose a refined prune-and-search technique that generalizes Megiddos prune-and-search technique on the deterministic version of this problem. This general algorithmic framework has also been used to solved other problems on uncertain data. We suspect that it will find more applications.

In Chapter 4, we study the one-center problem of uncertain points on a line in which each uncertain point has a continuous pdf—a piecewise constant function. We model this problem as a geometric problem that is to find the lowest point in the upper envelope of unimodal functions in the plane. Based on the refined prune-and-search technique, we compute this lowest point in linear time. Subsequently, we solve the uncertain one-center problem in linear time, which improves our previous result presented in Chapter 2 for the more general k-center problem when $k = 1$. In fact, we can easily extend this algorithm for the above geometric problem to more general functions where each function consists of m pieces with each piece being a constant-sized algebraic curve

segment and any two such curve segments of all functions intersect (transversally) at most a constant number of times. It would be interesting to see whether the techniques can be used to solve other related problems.

In Chapter 5, we solve in linear time the rectilinear (or L_1) one-center problem of uncertain points in the plane. We observe that the expected rectilinear distance function for each uncertain point is a convex piecewise linear function in the three dimensional space. Further, the lowest point in the upper envelop of graphs defined by all expected rectilinear distance functions corresponds to the center. By using the refined prune-and-search technique, we solve this computational geometry problem in linear time. Since the L_1 and L_∞ metrics are closely related to each other (by rotating the coordinate axes by 45°), the same problem under the L_∞ metric can be solved in linear time as well.

In Chapter 6, we first consider the center-coverage problem for uncertain points in a tree. We solve this problem by a simple greedy algorithm. However, the challenging work is to develop efficient data structures to perform certain operations that are needed in our algorithm. These data structures are based on a new tree decomposition with the following property: each subtree of the decomposition has at most two so-called "connectors". This property helps guarantee the running time of our algorithm. With our algorithm for the center-coverage problem as a decision procedure, we solve the k-center problem of uncertain points in the tree. Our tree decomposition and data structures are interesting in their own right and might find other applications as well. As future work, it would be interesting to see whether the algorithm for the k-center problem can be further improved.

In Chapter 7, we investigate the line-constrained k-center, k-means, and k-median problems in the plane (under various distance metrics) where the facilities are required to be located on a given line. We provide efficient algorithms for these problems, although the general versions (without the line constraint) of these problems are NP-hard. Since the one-dimensional k-center problem has an $\Omega(n \log n)$ time lower bound, our algorithms for the line constrained k-center are optimal. The time complexities of our k-median algorithms almost match those of the currently best algorithms of the one-dimensional k-median problems. As future work, it might be interesting to consider the same problems on uncertain points. In addition, two problem variations of the line-

constrained problems have been studied in the literature. In the first one, the slope of the line is given in the input and we want to find the best line with that slope to place facilities to minimize the objective value. In the second problem, we want to find an arbitrary line to place facilities to minimize the objective value. Efficient algorithms were given in the previous work for these problems. It might be interesting to see whether the techniques given in this dissertation can be extended to solve these two problems more efficiently than those in the previous work.

REFERENCES

[1] F. Preparata and M. Shamos, *Computational Geometry: An Introduction*, 2nd ed. Springer-Verlag, 1988.

[2] M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars, *Computational Geometry — Algorithms and Applications*, 3rd ed. Berlin: Springer-Verlag, 2008.

[3] S. Devadoss and J. O'Rourke, *Discrete and Computational Geometry.* New Jersey: Princeton University Press, 2011.

[4] J. Sack and J. Urrutia, Eds., *Handbook of Computational Geometry.* Elsevier, Amsterdam, The Netherlands, 2000.

[5] R. Cheng, J. Chen, and X. Xie, "Cleaning uncertain data with quality guarantees," *Proceedings of the VLDB Endowment*, vol. 1, no. 1, pp. 722–735, 2008.

[6] X. Dong, A. Halevy, and C. Yu, "Data integration with uncertainty," in *Proceedings of the 33rd International Conference on Very Large Data Bases*, 2007, pp. 687–698.

[7] P. Agarwal, S.-W. Cheng, Y. Tao, and K. Yi, "Indexing uncertain data," in *Proc. of the 28th Symposium on Principles of Database Systems (PODS)*, 2009, pp. 137–146.

[8] P. Agarwal, A. Efrat, S. Sankararaman, and W. Zhang, "Nearest-neighbor searching under uncertainty," in *Proc. of the 31st Symposium on Principles of Database Systems (PODS)*, 2012, pp. 225–236.

[9] P. Agarwal, S. Har-Peled, S. Suri, H. Yıldız, and W. Zhang, "Convex hulls under uncertainty," in *Proc. of the 22nd Annual European Symposium on Algorithms*, 2014, pp. 37–48.

[10] R. Cheng, Y. Xia, S. Prabhakar, R. Shah, and J. Vitter, "Efficient indexing methods for probabilistic threshold queries over uncertain data," in *Proc. of the 30th International Conference on Very Large Data Bases (VLDB)*, 2004, pp. 876–887.

[11] P. Kamousi, T. Chan, and S. Suri, "Closest pair and the post office problem for stochastic points," in *Proc. of the 12nd Workshop on Algorithms and Data Structures (WADS)*, 2011, pp. 548–559.

[12] ——, "Stochastic minimum spanning trees in Euclidean spaces," in *Proc. of the 27th Annual Symposium on Computational Geometry (SoCG)*, 2011, pp. 65–74.

[13] S. Suri and K. Verbeek, "On the most likely voronoi diagram and nearest neighbor searching," in *Proc. of the 25th International Symposium on Algorithms and Computation (ISAAC)*, 2014, pp. 338–350.

[14] S. Suri, K. Verbeek, and H. H. Yıldız, "On the most likely convex hull of uncertain points," in *Proc. of the 21st European Symposium on Algorithms*, 2013, pp. 791–802.

[15] Y. Tao, X. Xiao, and R. Cheng, "Range search on multidimensional uncertain data," *ACM Transactions on Database Systems*, vol. 32, no. 3, pp. 15–es, 2007.

[16] M. Yiu, N. Mamoulis, X. Dai, Y. Tao, and M. Vaitis, "Efficient evaluation of probabilistic advanced spatial queries on existentially uncertain data," *IEEE Transactions on Knowledge and Data Engineering*, vol. 21, pp. 108–122, 2009.

[17] R. Fowler, M. Paterson, and S. Tanimoto, "Optimal packing and covering in the plane are NP-complete," *Information Processing Letters*, vol. 12, pp. 133–137, 1981.

[18] M. Mahajan, P. Nimbhorkar, and K. Varadarajan, "The planar k-means problem is NP-hard," *Theoretical Computer Science*, vol. 442, pp. 13–21, 2012.

[19] N. Megiddo and K. Supowit, "On the complexity of some common geometric location problems," *SIAM Journal on Comuting*, vol. 13, pp. 182–196, 1984.

[20] H. Wang and J. Zhang, "One-dimensional k-center on uncertain data," in *Proc. of the 20th International Computing and Combinatorics Conference (COCOON)*, 2014, pp. 104–115.

[21] ——, "One-dimensional k-center on uncertain data," *Theoretical Computer Science*, vol. 602, pp. 114–124, 2015.

[22] V. Arya, N. Garg, R. Khandekar, A. Meyerson, K. Munagala, and V. Pandit, "Local search heuristics for k-median and facility location problems," *SIAM Journal on Computing*, vol. 33, pp. 544–562, 2004.

[23] M. Badoiu, S. Har-Peled, and P. Indyk, "Approximate clustering via core-sets," in *Proc. of the 34th Annual Symposium on Theory of Computing (STOC)*, 2002, pp. 250–257.

[24] S. Har-Peled and S. Mazumdar, "On coresets for k-means and k-median clustering," in *Proc. of the 36th Annual Symposium on Theory of Computing (STOC)*, 2004, pp. 291–300.

[25] M. Dyer, "On a multidimensional search technique and its application to the Euclidean one centre problem," *SIAM Journal on Computing*, vol. 15, no. 3, pp. 725–738, 1986.

[26] D. Lee and Y. Wu, "Geometric complexity of some location problems," *Algorithmica*, vol. 1, no. 1, pp. 193–211, 1986.

[27] N. Megiddo, "Linear-time algorithms for linear programming in R^3 and related problems," *SIAM Journal on Computing*, vol. 12, no. 4, pp. 759–776, 1983.

[28] R. Chandrasekaran and A. Tamir, "Algebraic optimization: The Fermat-Weber location problem," *Mathematical Programming*, vol. 46, no. 2, pp. 219–224, 1990.

[29] ——, "Polynomially bounded algorithms for locating p-centers on a tree," *Mathematical Programming*, vol. 22, no. 1, pp. 304–315, 1982.

[30] G. Frederickson, "Parametric search and locating supply centers in trees," in *Proc. of the 2nd International Workshop on Algorithms and Data Structures (WADS)*, 1991, pp. 299–319.

[31] N. Megiddo and A. Tamir, "New results on the complexity of p-centre problems," *SIAM J. on Computing*, vol. 12, no. 4, pp. 751–758, 1983.

[32] Z. Drezner and H. Hamacher, *Facility Location: Applications and Theory.* New York: Springer, 2004.

[33] N. Megiddo, A. Tamir, E. Zemel, and R. Chandrasekaran, "An $O(n \log^2 n)$ algorithm for the k-th longest path in a tree with applications to location problems," *SIAM J. on Computing*, vol. 10, pp. 328–337, 1981.

[34] G. Cormode and A. McGregor, "Approximation algorithms for clustering uncertain data," in *Proc. of the 27t Symposium on Principles of Database Systems (PODS)*, 2008, pp. 191–200.

[35] C. Aggarwal and P. Yu, "A framework for clustering uncertain data streams," in *Proc. of the 24th International Conference on Data Engineering (ICDE)*, 2008, pp. 150–159.

[36] W. Ngai, B. Kao, C. Chui, R. Cheng, M. Chau, and K. Yip, "Efficient clustering of uncertain data," in *Proc. of the 6th International Conference on Data Mining (ICDM)*, 2006, pp. 436–445.

[37] I. Averbakh and S. Bereg, "Facility location problems with uncertainty on the plane," *Discrete Optimization*, vol. 2, pp. 3–34, 2005.

[38] H. Wang, "Minmax regret 1-facility location on uncertain path networks," *European Journal of Operational Research*, vol. 239, pp. 636–643, 2014.

[39] H.-I. Yu, T.-C. Lin, and B.-F. Wang, "Improved algorithms for the minmax-regret 1-center and 1-median problems," *ACM Transactions on Algorithms*, vol. 4, no. 3, pp. 36–es, 2008.

[40] L. Snyder, "Facility location under uncertainty: a review," *IIE Transactions*, vol. 38, pp. 537–554, 2006.

[41] D. Chen, J. Li, and H. Wang, "Efficient algorithms for one-dimensional k-center problems," *Theoretical Computer Science*, vol. 592, pp. 135–142, 2015.

[42] D. Chen and H. Wang, "Efficient algorithms for the weighted k-center problem on a real line," in *Proc. of the 22nd International Symposium on Algorithms and Computation (ISAAC)*, 2011, pp. 584–593.

[43] R. Cole, "Slowing down sorting networks to obtain faster sorting algorithms," *Journal of the ACM*, vol. 34, no. 1, pp. 200–208, 1987.

[44] N. Megiddo, "Applying parallel computation algorithms in the design of serial algorithms," *Journal of the ACM*, vol. 30, no. 4, pp. 852–865, 1983.

[45] R. van Oostrum and R. Veltkamp, "Parametric search made practical," *Computational Geometry: Theory and Applications*, vol. 28, pp. 75–88, 2004.

[46] D. Chen and H. Wang, "A note on searching line arrangements and applications," *Information Processing Letters*, vol. 113, pp. 518–521, 2013.

[47] U. Gupta, D. Lee, and J.-T. Leung, "Efficient algorithms for interval graphs and circular-arc graphs," *Networks*, vol. 12, pp. 459–467, 1982.

[48] J. Snoeyink, "Maximum independent set for intervals by divide and conquer with pruning," *Networks*, vol. 49, pp. 158–159, 2007.

[49] B. Chazelle and L. Guibas, "Fractional cascading: I. A data structuring technique," *Algorithmica*, vol. 1, no. 1, pp. 133–162, 1986.

[50] ——, "Fractional cascading: II. Applieacations," *Algorithmica*, vol. 1, no. 1, pp. 163–191, 1986.

[51] H. Brönnimann and B. Chazelle, "Optimal slope selection via cuttings," *Computational Geometry: Theory and Applications*, vol. 10, no. 1, pp. 23–29, 1998.

[52] M. Katz and M. Sharir, "Optimal slope selection via expanders," *Information Processing Letters*, vol. 47, no. 3, pp. 115–122, 1993.

[53] H. Wang and J. Zhang, "Computing the center of uncertain points on tree networks," in *Proc. of the 14th Algorithms and Data Structures Symposium (WADS)*, 2015, pp. 606–618.

[54] ——, "Computing the center of uncertain points on tree networks," *Algorithmica*, vol. 78, no. 1, pp. 232–254, 2017.

[55] O. Kariv and S. Hakimi, "An algorithmic approach to network location problems. I: The p-centers," *SIAM J. on Applied Mathematics*, vol. 37, no. 3, pp. 513–538, 1979.

[56] B. Bhattacharya and Q. Shi, "Optimal algorithms for the weighted p-center problems on the real line for small p," in *Proc. of the 10th International Workshop on Algorithms and Data Structures*, 2007, pp. 529–540.

[57] N. Megiddo and E. Zemel, "An $O(n \log n)$ randomizing algorithm for the weighted Euclidean 1-center problem," *Journal of Algorithms*, vol. 7, pp. 358–368, 1986.

[58] P. Brass, C. Knauer, H.-S. Na, C.-S. Shin, and A. Vigneron, "The aligned k-center problem," *International Journal of Computational Geometry and Applications*, vol. 21, no. 02, pp. 157–178, 2011.

[59] A. Karmakar, S. Das, S. Nandy, and B. Bhattacharya, "Some variations on constrained minimum enclosing circle problem," *Journal of Combinatorial Optimization*, vol. 25, no. 2, pp. 176–190, 2013.

[60] H. Wang and J. Zhang, "Line-constrained k-median, k-means, and k-center problems in the plane," in *Proc. of the 25th International Symposium on Algorithms and Computation (ISAAC)*, 2014, pp. 3–14.

[61] A. Foul, "A 1-center problem on the plane with uniformly distributed demand points," *Operations Research Letters*, vol. 34, no. 3, pp. 264–268, 2006.

[62] A. Jørgensen, M. Löffler, and J. Phillips, "Geometric computations on indecisive points," in *Proc. of the 12nd Algorithms and Data Structures Symposium (WADS)*, 2011, pp. 536–547.

[63] M. Löffler and M. van Kreveld, "Largest bounding box, smallest diameter, and related problems on imprecise points," *Computational Geometry: Theory and Applications*, vol. 43, no. 4, pp. 419–433, 2010.

[64] M. de Berg, M. Roeloffzen, and B. Speckmann, "Kinetic 2-centers in the black-box model," in *Proc. of the 29th Annual Symposium on Computational Geometry (SoCG)*, 2013, pp. 145–154.

[65] I. Averbakh and O. Berman, "Minimax regret p-center location on a network with demand uncertainty," *Location Science*, vol. 5, pp. 247–254, 1997.

[66] M. Bender and M. Farach-Colton, "The LCA problem revisited," in *Proc. of the 4th Latin American Symposium on Theoretical Informatics*, 2000, pp. 88–94.

[67] D. Harel and R. Tarjan, "Fast algorithms for finding nearest common ancestors," *SIAM Journal on Computing*, vol. 13, pp. 338–355, 1984.

[68] H. Wang and J. Zhang, "A note on computing the center of uncertain data on the real line," *Operations Research Letters*, vol. 44, pp. 370–373, 2016.

[69] D. Chen, J. Li, and H. Wang, "Efficient algorithms for the one-dimensional k-center problem." *Theoretical Computer Science*, vol. 592, pp. 135–142, 2015.

[70] T. Chan, "More planar two-center algorithms," *Computational Geometry: Theory and Applications*, vol. 13, pp. 189–198, 1999.

[71] M. Hoffmann, "A simple linear algorithm for computing rectilinear 3-centers," *Computational Geometry*, vol. 31, no. 3, pp. 150–165, 2005.

[72] O. Kariv and S. Hakimi, "An algorithmic approach to network location problems. I: The p-centers," *SIAM J. on Applied Mathematics*, vol. 37, no. 3, pp. 513–538, 1979.

[73] D. Hochbaum and W. Maass, "Approximation schemes for covering and packing problems in image processing and vlsi," *Journal of the ACM*, vol. 32, no. 1, pp. 130–136, 1985.

[74] N. Mustafa and S. Ray, "PTAS for geometric hitting set problems via local search," in *Proc. of the 25th Annual Symposium on Computational Geometry (SoCG)*, 2009, pp. 17–22.

[75] S. Bereg, B. Bhattacharya, S. Das, T. Kameda, P. Mahapatra, and Z. Song, "Optimizing squares covering a set of points," *Theoretical Computer Science*, in press, 2015.

[76] T. Chan and N. Hu, "Geometric redblue set cover for unit squares and related problems," *Computational Geometry*, vol. 48, no. 5, pp. 380–385, 2015.

[77] T. F. Gonzalez, "Covering a set of points in multidimensional space," *Information Processing Letters*, vol. 40, no. 4, pp. 181–188, 1991.

[78] S.-S. Kim, S. Bae, and H.-K. Ahn, "Covering a point set by two disjoint rectangles," *International Journal of Computational Geometry and Applications*, vol. 21, pp. 313–330, 2011.

[79] P. Agarwal and M. Sharir, "Efficient algorithms for geometric optimization," *ACM Computing Surveys*, vol. 30, no. 4, pp. 412–458, 1998.

[80] G. Frederickson and D. Johnson, "Finding kth paths and p-centers by generating and searching good data structures," *Journal of Algorithms*, vol. 4, no. 1, pp. 61–80, 1983.

[81] P. Agarwal and J. Matoušek, "Dynamic half-space range reporting and its applications," *Algorithmica*, vol. 13, no. 4, pp. 325–345, 1995.

[82] G. Brodal and R. Jacob, "Dynamic planar convex hull," in *Proc. of the 43rd IEEE Symposium on Foundations of Computer Science (FOCS)*, 2002, pp. 617–626.

[83] M. Bender and M. Farach-Colton, "The level ancestor problem simplied," *Theoretical Computer Science*, vol. 321, pp. 5–12, 2004.

[84] H. Wang and J. Zhang, "Line-constrained k-median, k-means, and k-center problems in the plane," *International Journal of Computational Geometry and Applications*, vol. 26, no. 3 & 4, pp. 185–210, 2016.

[85] C. Bajaj, "The algebraic degree of geometric optimization problems," *Discrete and Computational Geometry*, vol. 3, pp. 177–191, 1988.

[86] V. Auletta, D. Parente, and G. Persiano, "Placing resources on a growing line," *Journal of Algorithms*, vol. 26, no. 1, pp. 87–100, 1998.

[87] R. Hassin and A. Tamir, "Improved complexity bounds for location problems on the real line," *Operations Research Letters*, vol. 10, pp. 395–402, 1991.

[88] D. Chen and H. Wang, "New algorithms for facility location problems on the real line," *Algorithmica*, vol. 69, pp. 370–383, 2014.

[89] N. Megiddo, "Linear programming in linear time when the dimension is fixed," *Journal of the ACM*, vol. 31, no. 1, pp. 114–127, 1984.

[90] W. F. de la Vega, M. Karpinski, C. Kenyon, and Y. Rabani, "Approximation schemes for clustering problems," in *Proc. of the 25th Annual ACM symposium on Theory of Computing (STOC)*, 2003, pp. 50–58.

[91] S. Floyd, "Least squares quantization in PCM," *IEEE Transactions on Information Theory*, vol. 28, pp. 129–137, 1982.

[92] T. Kanungo, D. Mount, N. Netanyahu, C. Piatko, R. Silverman, and A. Wu, "A local search approximation algorithm for k-means clustering," *Computational Geometry: Theory and Applications*, vol. 28, pp. 89–112, 2004.

[93] A. Kumar, Y. Sabharwal, and S. Sen, "A simple linear time $(1+\epsilon)$-approximation algorithm for k-means clustering in any dimensions," in *Proc. of the 45th IEEE Symposium on Foundations of Computer Science (FOCS)*, 2004, pp. 454–462.

[94] F. Hurtado, V. Sacristn, and G. Toussaint, "Some constrained minimax and maximin location problems," *Studies in Locational Analysis*, vol. 5, pp. 17–35, 2000.

[95] S. Kim and C.-S. Shin, "Efficient algorithms for two-center problems for a convex polygon," in *Proc. of the 6th Annual International Computing and Combinatorics Conference (COCOON)*, 2000, pp. 299–309.

[96] H. Alt, E. Arkin, H. Brönnimann, J. Erickson, S. Fekete, C. Knauer, J. Lenchner, J. Mitchell, and K. Whittlesey, "Minimum-cost coverage of point sets by disks," in

Proc, of the 22nd Annual Symposium on Computational Geometry (SoCG), 2006, pp. 449–458.

[97] A. Aggarwal, B. Schieber, and T. Tokuyama, "Finding a minimum weight k-link path in graphs with concave monge property and applications," *Discrete & Computational Geometry*, vol. 12, pp. 263–280, 1994.

[98] B. Schieber, "Computing a minimum weight k-link path in graphs with the concave monge property," *Journal of Algorithms*, vol. 29, no. 2, pp. 204–222, 1998.

[99] A. Aggarwal, M. Klawe, S. Moran, P. Shor, and R. Wilbur, "Geometric applications of a matrix-searching algorithm," *Algorithmica*, vol. 2, pp. 195–208, 1987.

[100] Z. Galil and K. Park, "A linear-time algorithm for concave one-dimensional dynamic programming," *Information Processing Letters*, vol. 33, no. 6, pp. 309–311, 1990.

[101] H. Fournier and A. Vigneron, "Fitting a step function to a point set," *Algorithmica*, vol. 60, no. 1, pp. 95–109, 2011.

[102] G. Frederickson, "Optimal algorithms for tree partitioning," in *Proc. of the 2nd Annual ACM-SIAM Symposium of Discrete Algorithms (SODA)*, 1991, pp. 168–177.

CURRICULUM VITAE

Jingru Zhang

EDUCATION

Ph.D., Computer Science. Utah State University, Logan, Utah. Expected in May 2017.

M.S., Traffic Information Engineering and Control. Chang'an University, Xi'an, China. June 2012.

B.S., Automation. Chang'an University, Xi'an, China. June 2009.

RESEARCH INTERESTS

Algorithms and Theory, Computational Geometry, Combinatorial Optimization, Operations Research, etc.

JOURNAL PUBLICATIONS

Haitao Wang and Jingru Zhang. Computing the Rectilinear Center of Uncertain Points in the Plane, submitted to *International Journal of Computational Geometry and Applications*, 2017. (**Under Review**)

Haitao Wang and Jingru Zhang. Computing the Center of Uncertain Points on Tree Networks. *Algorithmica*, vol. 78, no. 1, pages 232–254, 2017.

Haitao Wang and Jingru Zhang. Line-Constrained k-Median, k-Means, and k-Center Problems in the Plane, *International Journal of Computational Geometry and Applications*, vol. 26, no. 3 & 4, pages 185–210, 2016.

Haitao Wang and Jingru Zhang. A Note on Computing the Center of Uncertain Data on the Real Line. *Operations Research Letters*, vol. 44, pages 370–373, 2016.

Haitao Wang and Jingru Zhang, One-Dimensional k-Center on Uncertain Data, *Theoretical Computer Science*, vol. 602, no. 3, pages 114–124, 2015.

Shuguang Li, Hongkai Yu, Jingru Zhang, Kaixin Yang, Ran Bin. A Video-based Traffic Data Collection System for Multiple Vehicle Types. *Journal of IET Intelligent Transport Systems*, vol. 8, no. 2, pages 164–174, 2014.

CONFERENCE PUBLICATIONS

Haitao Wang and Jingru Zhang. Covering Uncertain Points in a Tree, To appear in *Proceedings of the 15th Algorithms and Data Structures Symposium* (WADS), St. John's, Canada, 2017.

Haitao Wang and Jingru Zhang. Computing the Center of Uncertain Points on Tree Networks, *Proceedings of the 14th Algorithms and Data Structures Symposium* (WADS), pages 606–618, 2015.

Haitao Wang and Jingru Zhang. Line-Constrained k-Median, k-Means, and k-Center Problems in the Plane, *Proceedings of the 25th International Symposium on Algorithms and Computation*(ISAAC), pages 3–14, 2014.

Haitao Wang and Jingru Zhang. One-Dimensional k-Center on Uncertain Data, *Proceedings of the 20th Annual International Computing and Combinatorics Conference*(COCOON), pages 104–115, 2014.

Shuguang Li, Hongkai Yu, Jingru Zhang. A Video-based Traffic Data Collection System for Multiple Vehicle Types, *Compendium of Papers CD-ROM for 91st Transportation Research Board Annual Meeting*, Washington, D.C., USA, 2012.

Shuguang Li, Hongkai Yu, Jingru Zhang, Ke Yue. Multi-type Vehicles' Traffic Data Collection Using Video Processing. *Proceedings of The 2nd International Conference on Intelligent Control and Information Processing*(ICICIP), pages 271-276, 2011.

PATENTS

Dynamic and Semi-real Platform of Microscopic Traffic Simulation for Intersection. Granted Patent No: CN102280027A, China.

Video-based Traffic Data Collection System for Multiple Vehicle Types. Granted Patent No: CN102810250A, China.