1-1-2011

Minimax and Maximin Fitting of Geometric Objects to Sets of Points

Yan B. Mayster
University of Denver

Minimax and Maximin Fitting of Geometric Objects to Sets of Points

A Dissertation

Presented to

the Faculty of Engineering and Computer Science

University of Denver

In Partial Fulfillment

of the Requirements for the Degree

Doctor of Philosophy

by

Yan Mayster

June 2011

Advisor:
Dr. Mario A. Lopez

Author: Yan Mayster
Title: Minimax and Maximin Fitting of Geometric Objects to Sets of Points
Advisor: Dr. Mario A. Lopez
Degree Date: June 2011

Abstract

This thesis addresses several problems in the facility location sub-area of computational geometry. Let S be a set of n points in the plane. We derive algorithms for approximating S by a step function curve of size $k < n$, i.e., by an x-monotone orthogonal polyline $\mathcal{R}$ with $k < n$ horizontal segments. We use the vertical distance to measure the quality of the approximation, i.e., the maximum distance from a point in S to the horizontal segment directly above or below it. We consider two types of problems: $\min$-ε, where the goal is to minimize the error for a given number of horizontal segments k and $\min$-$\#$, where the goal is to minimize the number of segments for a given allowed error ε. After $O(n)$ preprocessing time, we solve instances of the latter in $O(\min\{k \log n, n\})$ time per instance. We can then solve the former problem in $O(\min\{n^2, nk \log n\})$ time. Both algorithms require $O(n)$ space. The second contribution is a heuristic for the $\min$-ε problem that computes a solution within a factor of 3 of the optimal error for k segments, or with at most the same error as the k-optimal but using $2k-1$ segments. Furthermore, experiments on real data show even better results than what is guaranteed by the theoretical bounds. Both approximations run in $O(n \log n)$ time and $O(n)$ space. Then, we present an exact algorithm for the weighted version of this problem that runs in $O(n^2)$ time and generalize the heuristic to handle weights at the expense of an additional $\log n$ factor. At this point, a randomized algorithm that runs in $O(n \log^2 n)$ expected time for the unweighted version is presented. It easily generalizes to the weighted case, though at the expense of an additional $\log n$ factor. Finally, we treat the maximin problem and present an $O(n^3 \log n)$ solution to the problem of finding the furthest separating line through a set of weighted points. We conclude with solutions to the "obnoxious" wedge problem: an $O(n^2 \log n)$ algorithm for the general case of a wedge with its apex on the boundary of the convex hull of S and an $O(n^2)$ algorithm for the case of the apex of a wedge coming from the input set S.

Acknowledgements

I thank my advisor Mario Lopez for his many years of close friendship and mentoring. His patience, advice, and consistent encouragement were absolutely invaluable to me in completing this dissertation and degree. I also thank my committee members, Alvaro Arias, Chris GauthierDickey, Scott Leutenegger, and Ronald DeLyser, for their time, advice, and comments. In addition, I would like to thank Drs. GauthierDickey and DeLyser for the many grammatical and presentation suggestions to my thesis.

I thank again my advisor Mario Lopez for inviting me to continue my graduate school experience with doctoral work. I also want to thank the faculty at the departments of Computer Science and Mathematics whose courses I have taken at the University of Denver throughout all my degrees. I am grateful to all of you.

Last, but not least, I wholeheartedly thank my graduate student colleagues and co-authors, Mohammed Al-Bow and Riquelmi Cardona, for their friendship, collaboration, conversations, and all the time we have spent together. I also thank Jeff Edgington for leading the way and showing me the path to graduation and freedom. All of you friends and colleagues, whose names are too many to mention here, have imparted your knowledge and wisdom to me, and I thank you all.

Most importantly, I thank my parents, Boris and Rita Mayster,
and my brother, Dmitriy Mayster, for their love and attention.

Contents

List of Tables

List of Figures

Chapter 1

Introduction and Definitions

1.1 Introduction

Facility location and routing are important areas of computational geometry, a branch of computer science devoted to finding solutions for problems that can be formulated in terms of relationships between spatial objects. Generally, the solution takes the form of an *algorithm*, a finite sequence of steps that can be performed on any valid input, perhaps with some stated limitations (this is called an *instance* of the problem), to produce a correct answer using finite resources. In the case of computer execution, the most crucial resources are time (number of small elementary steps) and memory.

The need for the correct and most efficient algorithms has led to the creation of a field immensely rich in both theory and applications. An integral part of this study of algorithms is the subject of data structures - different ways of organizing and storing data to facilitate and reduce the cost of its processing. These fundamental subjects of computer science are elucidated in many textbooks, including a true classic by Cormen et al. [13] and a detailed compendium of data structures in [40]. Computational geometry is naturally a part of this vast area of study and is treated in depth in [14, 42].

Measuring the resource costs of an algorithm requires consistent and mathematically grounded methods and notation. Fortunately, a general method with more or less uniform terminology and notation has been developed to address this need - the asymptotic complexity analysis. Employing the so-called asymptotic notation, commonly used $O(\cdot)$ (big-Oh) and $\Theta(\cdot)$, it is used to find out and express the time and memory costs of an algorithm as non-decreasing functions of the size of input and, sometimes, also output. As the term "asymptotic" suggests, it is generally oblivious to any constant factors and all but the dominating term. The results expressed in this notation are also extremely useful for comparing the performance of different algorithms as well as, for any specific problem, demonstrating the lower theoretical bounds for the amount of resources required to solve it. That is, asymptotic complexity analysis can be used to investigate and express the least cost of an *optimal* algorithm, even if as yet undiscovered. In our area of study, the input is generally a set of points, which may originate from the vertices of a polygon or a collection of polygons. Hence, resource costs are expressed in terms of their number as in almost all cases the amount of memory and the number of elementary steps taken by an algorithm are proportional to that quantity.

Facility location and routing studies the placement of geometric objects, called "facilities", among another set of objects of the same or different kind under certain criteria of cost. A real-world example of such a problem would be that of finding the best path for a pipeline which has to connect to a set of wells. Due to the mathematical nature of the problems studied by this area of computational geometry as well as the diversity and complexity of potential applications, there are a great many problems that have received considerable attention from researchers but still hold many unanswered questions and more than a few that are open and awaiting any reasonable solution. Every type of geometric object may create a different subproblem and require a different approach to processing the input data and computing the final solution. Further, the very notion of "optimal" placement of an object among other objects of the same or different type opens up a wealth of variants for the researcher. Any one of the following, and many other, criteria generally yields its own setup and may vastly change the nature of the problem – the main strategy of placement (e.g., staying as close to or as far away as possible to the given objects), the definition of the approximation cost used, the complexity of the approximating object (e.g., number of segments in a polyline, or facets on a polytope, etc.), the restrictions on angles inside an object or the overall orientation of an object. The dimension and the underlying geometry of the space, including the definition of distance, produce yet more problems. Still of a more theoretical nature are the questions of computability and lower bounds on performance. We shall give a more formal treatment of many of these aspects when defining the problem and surveying the previous work as well as during the presentation of the contributions of this thesis.

1.2 Minimax Approximation

Minimax approximation derives the name from its goal of MINImizing the distance to the furthest removed, i.e., MAXimally distant, object. This mainly agrees with the straightforward notion of approximation, yet is quite different from the mathematical least-squares regression approach in that the points are ultimately not being approximated as a set in the aggregate sense. Specifically, the outliers are being targeted the most. It does not matter how close the other points are to the approximating object - the sole representation of cost is the distance to the furthest one. From another point of view, there is a certain degree of equality and balance accorded to all points in the sense that clusters play no significant role here. Therefore, the example with the pipeline is perhaps not the one illustrating this particular type of approximation. There, the *real* cost is the total pipe length, which is the main influence on the maintenance and operation costs of the pipeline not to mention other yet more complicated cost ingredients that may or may not be present, such as the difficulties presented by the terrain, etc. Thus, a good application for this strategy would be when the subject points indeed present a case that merits a more equitable distribution of facilities, such as, perhaps, the placement of hospitals, post offices, and cell phone towers, or the routing of state highways. In the purely abstract setting, it is of course easy to claim that the best solution is to place the approximating object on or through every input point, thus achieving the total error of 0 no matter what the metric is. This may in fact be feasible but additional restrictions on the complexity (or number) of the approximating object(s) most often prevent such situations. In the presence of a limit on the size of the approximating object, say, number of segments in a polyline, it is the trade-offs due to the interplay of this limit with the distribution of the approximated points as well as the notion of “high” cost in the application setting that ultimately determine the quality of the fit.

1.3 Other Approximation Methods

Having reviewed the motivation and application of the minimax approximation, it is then natural to ask how to formulate an alternative method for the placement and routing of facilities that are, conversely, less than desirable for the surrounding environment. Such facilities are often termed “obnoxious” in the literature. Thus, the goal of the approximation becomes to MAXimize the distance to the closest, i.e. MINimally distant, input point. Deciding exactly which of the two sets of objects - the placed ones or the input points - are “obnoxious” is, of course, a matter of perspective. Alternatively, the input points could be the undesirable locations, e.g., toxic dumps, and the placed route could be a highway used by people who do not want to be exposed to the health hazards. Such situations give rise to the *maximin* method, full opposite of the minimax, used to address these “obnoxious” facility location problems. Still other approximation methods exist and are quite well covered in the literature. One of them is the *minsum* approximation, where the objective is to minimize the total of all distances from the input objects to the placed facility. For an in-depth treatment of such problems the reader is referred to the work by Aronov et al. [3], as well as the papers referenced therein. We note here that this method fits the pipeline example mentioned in the previous section. Finally, we note that our thesis is restricted to the minimax and maximin methods.

1.4 The Facility and The Input Objects

In the most general sense, facility location refers to the fitting of objects of one kind to a set of objects of another (possibly, same) kind. In this work, and in the previous work surveyed in this thesis, the facility is usually a straight line or polyline, a curve made up of multiple straight-line segments. Sometimes, we may represent the facility with a more complex object, such as a wedge, the 2D equivalent of the familiar cone shape - the interior of an angle formed by two half-rays with a common endpoint. A look at Figure 6.1 would help visualize this type of facility location problem. For the work on one less common facility type, the reader may consult the paper [17] by Díaz-Báñezet al. that studies the largest empty annulus problem.

The input objects, though possibly representing vertices of polygons or other geometric entities, are generally treated as a set of n two dimensional points without concern as to their origin. Thus, the problem of facility location and/or routing in this context becomes abstract "curve fitting." While there is a lot of attention in the literature devoted to the fitting of general polylines, the primary focus of this thesis is on the so-called *x-monotone orthogonal curves* or *step functions*. A curve $\mathcal{R}$ is orthogonal (or *rectilinear* as referred to in some publications) if it consists only of alternating horizontal and vertical segments. This, of course, implies a fixed frame of reference although not necessarily the canonical coordinate axes since a linear transformation suffices to convert between any pair of bases. $\mathcal{R}$ is said to be x-monotone if the x-domains of any two consecutive horizontal segments meet in a single value, namely the x-coordinate of the vertical segment joining them. Figure 1.1 provides an illustration of this type of facility.

It is important to note here that there is a general lack of agreement among the researchers as to the preferred name for such polylines. Some, as do the authors of [21] and [49], whose work was extended and improved upon by the author of the present thesis, chose the above-mentioned term of x-monotone rectilinear curve. However, even some of

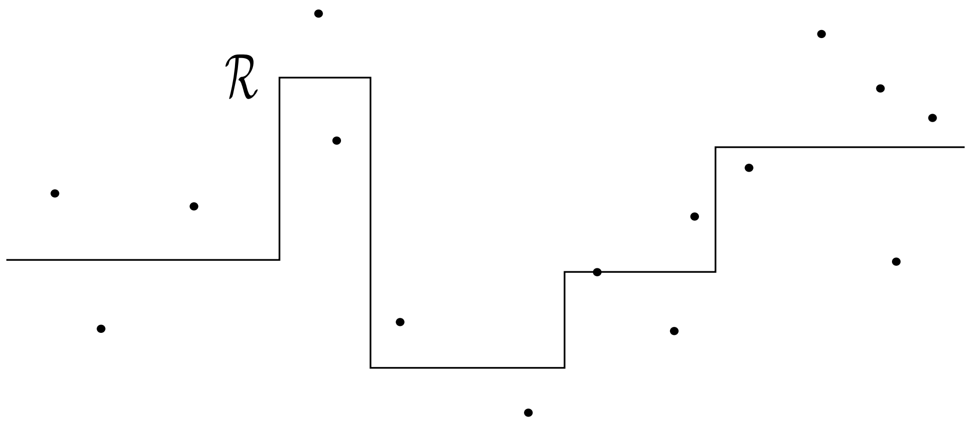

Figure 1.1: A step function curve approximating a set of points.

these same authors in another paper may replace the term *rectilinear* with *orthogonal* and *curve* with *chain* (e.g., in [18]), and at times add the word *polygonal* to the description. Still other authors may refer to these curves (polylines/chains) as *step functions* [26]. This term, if applied mathematically to the same entity, clearly contradicts the existence of the vertical segments as well as the overlap in the x-domains of the horizontal segments that touch at their endpoints. While this strict mathematical view may be at odds with the picture of the polyline in Figure 1.1 and, hence, be construed as a slight abuse of the concept, it only serves to underscore the abstract nature of this type of approximation. The nature or even presence of the vertical segments is overall immaterial to the problem as they are added in simply to produce a connected path representing the route or facility. Similarly, the locations of where one horizontal segment ends and another begins are not fixed but are free to be designated anywhere between a pair of input points adjacent along the x-axis. These locations may be fixed to be half-way between a pair of such points defining the beginning of a new horizontal segment or "step", but only as a matter of convention. The curve could be specified simply as a collection of k constituent horizontal segments, since vertical ones are implied, $s_1, ..., s_k$, with their respective x-ranges. However, due to its evolving nature during the execution of the algorithm it is almost always more convenient to think of each segment in terms of the input objects it approximates and thus of a curve as a partition of the input set into a collection of subsets with contiguous and non-overlapping x-domains (we shall formalize this later in the definition of the *allocation* set). Finally, going back to the subject of the confusion over the use of various terms for the facility in Figure 1.1, we note that in this thesis we shall attempt to use one term - step function curve/polyline. Notwithstanding the mentioned contradictions to the mathematical definition of a step function, this term introduces the least ambiguity as to the shape and orientation of the approximating entity, while adding in the word "curve" or "polyline" clarifies the continuity of the entity and alludes to the presence of vertical (or some other non-approximating and non-essential) links. Any occurrence of another term mentioned in this paragraph is to be interpreted synonymously.

1.5 Min-$\#$ vs. Min-ε

In the case of facilities represented by polylines or any type of geometric objects with a quantifiable complexity, such as number of segments, sides, etc., two natural optimization criteria are used. The first is to minimize the complexity of the polyline while aiming to achieve the target error, which is measured according to any of the above mentioned approximation methods. This is the $\min$-$\#$ optimization. In some cases, when the solution can be constructed in an incremental or online "greedy" mode, it is the easier variant. The $\min$-ε optimization requires the minimization of the error while keeping the complexity of the facility below a certain pre-specified threshold. More specifically, in the case of step function curves, $\min$-$\#$ calls for a solution with the least number of horizontal line segments given a target error ε. Similarly, $\min$-ε specifies a number k and asks for a curve with no more than k segments that achieves least possible error bounded by ε. It is worthwhile to note that if the space of possible errors is discrete and can be searched efficiently, the solution to the $\min$-ε problem may be obtained by repeated applications of the solution to the $\min$-$\#$ problem in the same setting. This will be shown in an example in a later section describing an algorithm by Díaz-Báñez et al.

1.6 Distance Metrics

We have already discussed several common ways to measure the error of the approximation: minimax, maximin, and minsum. Each of these approximation methods depends on the definition of the distance function, which determines the structure of the underlying space. It needs to be noted that words "distance" and "cost" are used interchangeably here and no reference to a specific or "natural" distance is made. In fact, we shall mention these methods later in a pure optimization context, replacing the word "distance" with "cost" and calling "furthest" objects "most expensive", etc. However, in an abstract sense almost all such "cost" metrics are proper distance metrics and so such use of terms is justified. Many facility location problems have been studied in the context of what may appear as unusual yet well motivated metrics, such as the Fréchet distance (for a look at one such work, refer to [1] by P.K. Agarwal). Throughout most of this thesis, unless otherwise noted, the distance metric used, i.e., the approximation cost from a single object to the facility, is the Euclidean distance with two important reservations.

In the case of step function curves, the distance from a point to the function $\mathcal{R}$ is always measured to the closest horizontal segment - the *approximating* segment. This is the reason it is often referred to as the *vertical* distance. We now define this *vertical distance* formally since this is the error function used throughout most of our results. For any two points $p = (x, y)$ and $p' = (x', y')$, the vertical distance between p and p' is

$$d_v(p, p') = \begin{cases} |y - y'| & \text{if } x = x', \\ \infty & \text{otherwise.} \end{cases}$$

Then, the vertical distance between a point p and a curve $\mathcal{R}$ is defined as

$$d_v(p, \mathcal{R}) = \min_{q \in \mathcal{R}} d_v(p, q).$$

Thus, using the minimax method, and following the notation of [21], the overall error, called *eccentricity*, of $\mathcal{R}$ with respect to a set of points S is the maximum vertical error between the points of S and $\mathcal{R}$,

$$e(S, \mathcal{R}) = \max_{1 \leq i \leq n} d_v(p_i, \mathcal{R}).$$

This is illustrated in the Figure 1.2 below. Here, the step function curve $\mathcal{R}$ approximates the set of points $S = p_1, \ldots, p_{15}$. $\mathcal{R}$ has complexity $k = 5$. Each segment of $\mathcal{R}$ is subscripted both in order of increasing x domain as well as with the indices of points it approximates (recall the discussion of Section 1.4 on the nature of the curve and its segments). The

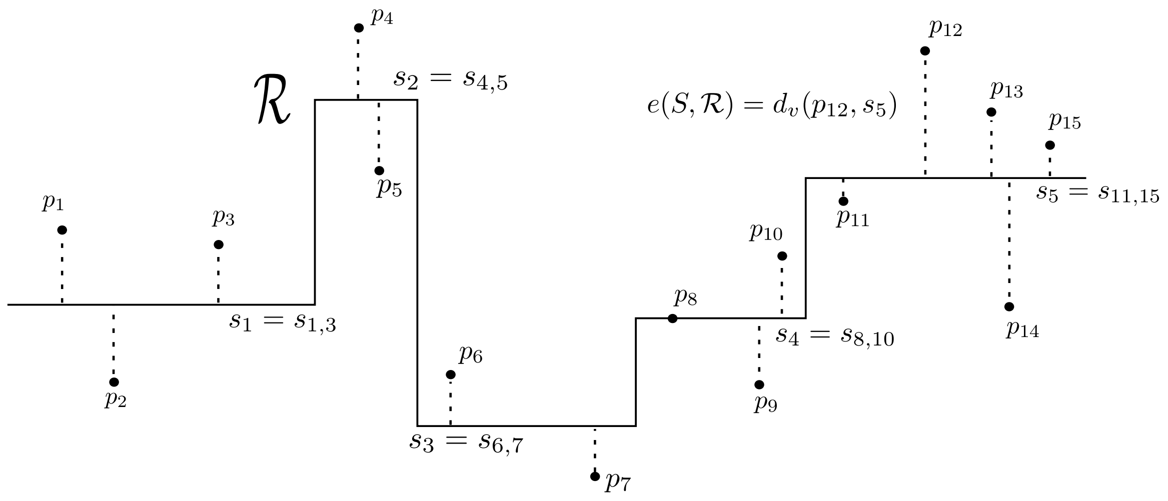

Figure 1.2: Illustration of vertical distance, segments of $\mathcal{R}$/subsets of S, and minimax eccentricity of a curve.

error induced by s_1 is $d_v(p_1, \mathcal{R}) = 5$ (counting the dashes from p_1 to s_1), that of s_2 is $d_v(p_4, \mathcal{R}) = 5$ and s_3 is $d_v(p_6, \mathcal{R}) = 3.5$. Looking ahead to the discussion in Chapters 2 and 3, note that each segment has at least two points at maximum distance to it. This is due to the fact that a segment must be centered in order to minimize its error - a requirement that in theory has to be enforced only for the segment contributing the maximum error, the eccentricity $e(S, \mathcal{R})$. In this case, it is s_5, which has distance 9 to both p_{12} and p_{13}. Also note that some points can be arbitrarily close to $\mathcal{R}$, such as $d_v(p_8, \mathcal{R}) = 0$, but this has no effect on the overall error. The second reservation mentioned previously concerns the use of weighted distance metrics.

1.6.1 Weighted vs. Unweighted

Input objects may be endowed with additional characteristics that influence their relative priority or “desirability.” Such characteristics influence and augment the approximation costs to the objects, while not completely ignoring the costs stemming from the distances in the underlying space. These are often specified in the form of distance-modifying multiplicative weights, which may come from any nonnegative subset of real numbers. Each point is assigned a single weight value which does not have to be unique. We now re-define the vertical distance above in this new weighted setting. If a horizontal segment s has y-coordinate y_s and x-range $[x_s, x'_s]$, then the weighted vertical distance from s to a point p_i with associated weight w_i is

$$d_v^W(p_i, s) = \begin{cases} w_i|y_i - y_s| & \text{if } x_i \in [x_s, x'_s], \\ \infty & \text{otherwise.} \end{cases}$$

The weighted vertical distance between a point p_i and a curve $\mathcal{R}$ is, therefore, defined as

$$d_v^W(p_i, \mathcal{R}) = \min_{s \in \mathcal{R}} d_v^W(p_i, s),$$

and similarly d^W takes the place of d_v in the definition of the eccentricity. Let's re-consider the example of Figure 1.2 after having assigned to some points of S weights that are different from 1. The weights are shown in Figure 1.3 along with the entailed changes to $\mathcal{R}$. The points that do not have a new weight value next to them in the figure kept the old implied weight equal to 1. Now, we have s_1 with error $d_v^W(p_1, \mathcal{R}) = d_v^W(p_2, \mathcal{R}) = 10$, s_2 unchanged, s_3 having merged with s_4 due to the much greater contributions of p_9 and p_{10} and relative insignificance of p_6 and p_7, and a different position for s_5 which has been re-labeled to s_4. The curve still passes through p_8, which is all the more important now that its weight is 10 times greater than it used to be. Further, both the new s_3 and the new

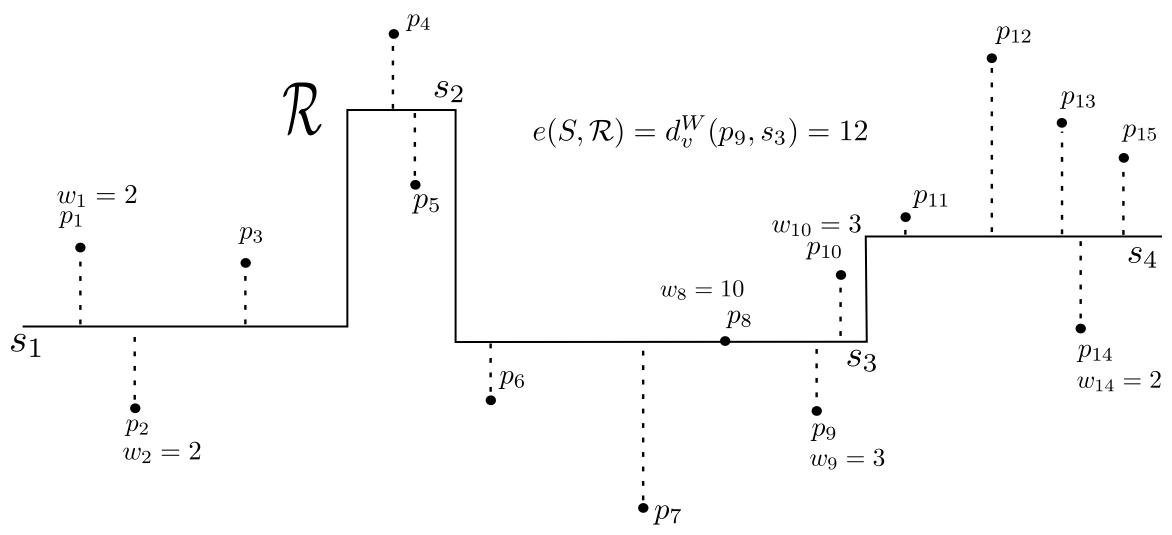

Figure 1.3: The curve $\mathcal{R}$ from Figure 1.2 with distances altered by weights.

s_4 give the eccentricity of the curve, error that is equal to 12. To see this, count the dashes emanating from either p_9 or p_{10} and multiply by the weights of those points (3 in each case, yielding total error 12) and similarly check p_{12} and p_{14}, remembering that the weight for p_{12} is 1 while that of p_{14} is 2 (counting the dashes from p_{12} may be a visual challenge but the author asserts that they number 12). Note that each segment is still centered with respect to the points it approximates, except now centering takes the weights into account.

1.7 Other Variants and Generalizations

The framework of the facility location problem described above already admits a large body of variants. Changing the approximation method, the distance metric, and the facility and input object types according to the possibilities mentioned or those beyond yields many interesting problems. Many different restrictions and generalizations are possible and oftentimes require unique and clever approaches. Some are directions for future work to extend the results in this thesis. For example, in each approximation method described there is an implicit notion of a “corridor,” although in the presence of weights not necessarily conforming to our intuition. This corridor is centered around the facility and in the maximin case is empty with its “width” maximized while in the minimax case it encloses all points while its width is minimized. We may relax this requirement on emptiness or complete containment and admit up to k points whose distance to the facility has no effect on the overall approximation cost. This is called the k-dense variant and it may be a trivial extension of an existing 0-dense result or may require a unique approach for a more efficient answer. As a matter of clarification, we note that the use of k here is in no way related to the complexity of the approximating curve, also denoted k, and is preserved here to conform to the widespread notation in the literature.

Similarly, a further generalization for any approximation problem that restricts angles between constituent segments of the curve is to allow for its arbitrary orientation relative to the axes. Likewise, one could allow for arbitrary but fixed angles between the segments of the curve. For example, one could extend the step function curve approximation problem to include the 45° angle or any finite (usually, rather small) subset of angles. This appears to make the problem considerably harder and an efficient solution is known in the presence of an additional restriction. The minimax $\min\text{-}\varepsilon$ and maximin problems have been studied in the presence of a fixed and known limit on the number of segments in the curve, say 1, 2, or 3. Several interesting results specific to these restrictions are in [15, 20, 12]. A result that combines this restriction (to 2 segments) with the generalization to arbitrary orientation has been recently published [18].

Finally, many of these problems carry over to higher dimensions albeit usually requiring some redefinitions due to greater degrees of freedom available for angles and orientation. These 3- and higher dimensional cases have been studied in [47, 18].

1.8 Applications

Just like the example with pipeline placement, many similar but perhaps more likely scenarios can be conceived in more or less familiar settings, including VLSI design and query optimization in DBMS based on predictions with histograms. Step function polylines are just one kind of histogram and have been applied in this field. However, the solutions to these problems are important from the point of view of the theoretical completeness of the field of computational geometry and have been studied and presented in various papers and in this thesis primarily for this reason.

1.9 Contributions and Structure of this Thesis

This thesis continues in Section 2 with a brief survey of the previous work in the facility location domain with particular focus on the pre-existing solutions to the specific problems solved in this thesis. The contributions presented in this text are new solutions to several problems in the facility location domain. The primary focus is on the minimax step function polyline approximation of point sets in the plane. First, in Section 3 we solve the unweighted version exactly in $O(\min\{n^2, nk \log n\})$ time. Then, we present a heuristic that yields solutions within a factor of 2 from the exact one but performs significantly faster - the time bound achieved is $O(n \log n)$. The focus subsequently shifts in Section

4 to the weighted approximation in the otherwise similar setting and adapted versions of these algorithms are presented. The running times have additional logarithmic factors but the memory requirements remain linear. At this point, we also present a randomized algorithm that takes $O(n \log^2 n)$ expected time for the unweighted case and $O(n \log^3 n)$ for the weighted one. Further, in Section 5 a treatment of the weighted maximin line problem follows with the description of a new algorithm and the issues encountered in a pre-existing one. Finally, Section 6 is on the maximin approximation with a wedge and concluding remarks in Section 7 complete the thesis.

Chapter 2

Background and Previous Work

2.1 History of Facility Location/Routing Problems

The problem of approximating a piecewise-linear curve $\mathcal{C}$ by another piecewise-linear curve $\mathcal{R}$ of smaller complexity whose vertices are a subset of the vertices of $\mathcal{C}$ was among the first to be studied in the field of facility location. Using the usual Euclidean measure of error, Imai and Iri [34] produced a polynomial time algorithm for the $\min$-$\#$ variant of this problem. This result was improved by Melkman and O'Rourke [41], Hakimi and Schmeichel [29], and Varadarajan [47]. In [36] Imai and Iri solved the $\min$-ε variant (faster algorithms were later developed in [11, 25, 29]) and in [35] they described an optimal algorithm to solve the $\min$-$\#$ variant of the same problem under the vertical distance metric but without requiring that the vertices of $\mathcal{R}$ also be vertices of $\mathcal{C}$. The $\min$-$\#$ and $\min$-ε variants of this unrestricted version of the problem under the vertical distance metric were tackled in [28, 29, 48]. A special case of the unrestricted $\min$-ε variant, where the approximating curve has exactly 2 segments (called 1-corner chains), is treated by Díaz-Báñez et al in [15]. Eu and Toussaint [25] considered other possible metrics in the context of the restricted version, including L_1 and the so-called "parallel-strip" error criterion, where the distance is measured from the approximated vertices to the *line* collinear with the approximating line segment. They also devised algorithms to approximate polygonal curves in 3D under L_1 and L_∞ metrics, with the restriction that the curves must be monotone with respect to one of the three axes. This restriction is removed by Barequet et al [4], who also consider both $\min\text{-}\varepsilon$ and $\min\text{-}\#$ optimization variants in dimensions higher than 3. Finally, some polyline approximation problems have been studied in the general weighted case in [31, 51]. The weighted case for a single approximating line under the minsum method has received particular attention as the problem of the location of the median line (see [50]).

The problem of step function approximation has been previously studied in [21, 49]. Díaz-Báñez and Mesa derive a simple $O(n)$ time algorithm to solve the step function $\min\text{-}\#$ problem and use it as a subroutine to solve the $\min\text{-}\varepsilon$ problem in $O(n^2 \log n)$ time. This result was improved by Wang, who proposed an algorithm that runs in $O(n^2)$ time. His algorithm also makes use of the $\min\text{-}\#$ algorithm of [21] but achieves better performance by reducing the number of subroutine calls.

The obnoxious location problem has been tackled in several papers [38, 9], with variants involving various error metrics [6] and constraints on the location of the obnoxious facility ([33, 5]), such as restricting it to lie in a given polygonal region [7]. In [10], Cappanera et al. tackle the problem of simultaneous facility location and routing. Different types of obnoxious facilities such as planes [19] or annuli [17] have also been considered. The first paper to solve the widest empty corridor problem (when the obnoxious facility is a line) was [32]. Using duality, Houle and Maciel solved the problem in $O(n^2)$ time. Subsequent papers addressed other variants, such as allowing the corridor to contain up to some fixed number of input points [37, 46], weighted distances [23, 22], L-shaped [12] and 1-corner corridors [16, 20].

To the best of our knowledge, step function curve approximation in the weighted minimax setting had not been previously studied, and hence, our solution was the first in that area. However, subsequent to the publication of our results, new work has appeared ([26]) that has tackled both the unweighted case (in $O(n \log n)$ time) and the weighted one (in $O(n \log^4 n)$) using the sorted matrix searching technique ([27]). The $O(n \log n)$ running time for the unweighted case on unsorted input is clearly an optimal result.

2.2 Previous Work in Minimax Step Function Curve Approximation

The $\min$-# step function curve approximation problem can be solved in $O(n)$ time [21] by sweeping the points from left to right and extending the current segment (by including the new point and re-centering) provided that its range is no more than twice the allowed eccentricity. When that amount is exceeded, a new segment is started. This approach also carries over to the weighted case as will be explained in Section 4. We now need the following definition.

Definition 2.1. Let S be an input set of points indexed by x-coordinate. For any $1 \leq i \leq j \leq n$, let $s_{i,j}$ be the optimal minimax segment approximating the set of points $\{p_k | i \leq k \leq j\}$ (as in Figure 1.2). We define ϵ_{ij} as the approximation error introduced by $s_{i,j}$.

The authors of [21] noticed that the eccentricity of any optimal minimax step function approximation curve generated for S must be some ϵ_{ij}. Then, the $\min$-ε problem can be reduced to a binary search on the sorted list of candidate eccentricities $\mathcal{E} = \{epsilon_{ij} | 1 \leq i \leq j \leq n\}$, comparing target complexity k with the complexity of the current candidate curve as a guide. What allows this technique to be applied is the simple observation that curves with higher complexity yield lower eccentricity, and vice versa. Using this fact and their linear time $\min$-$\#$ algorithm just explained, the authors develop an $O(n^2 \log n)$ algorithm for the $\min$-ε problem.

Wang [49] keeps the $\min$-$\#$ algorithm intact but improves the running time for $\min$-ε by avoiding the generation and sorting of all possible eccentricities. Instead he uses a double sweepline scan to determine which instances of the $\min$-$\#$ problem to solve. Initially, the left and right sweeplines start at p_1 and p_2, and ϵ_{12} as well as an approximation curve $\mathcal{R}'$ with this eccentricity are computed. Then, every subsequent step of the algorithm depends on how the number of segments k' in the curve computed in the previous step compares to k. The author makes the following important observation.

Observation 2.2. For $1 \leq i < j \leq n-1$, $\epsilon_{ij} \leq \epsilon_{i(j+1)}$ and for $2 \leq i < j \leq n$ it holds that $\epsilon_{(i-1)j} \geq \epsilon_{ij}$, since $S_{ij} \subseteq S_{i(j+1)}$ and $S_{(i-1)j} \supseteq S_{ij}$.

Then, if $k' > k$, the candidate curve had eccentricity smaller than that of the k-optimal curve and the right sweepline is advanced to the next point (to increase eccentricity, by the above observation). Similarly, if $k' \leq k$, the candidate curve is pushed into an answer array and the left sweep line is moved one point to the right to look for $\mathcal{R}'$ with a potentially greater number of links and eccentricity no bigger than that at the last step. As a result, [49] tries no more than $2n$ values of ϵ and, thus, solves the $\min$-ε problem in $O(n^2)$ time. See [49] for details.

2.3 Previous Work in Maximin Approximation with a Line

The problem of finding a farthest separating line, also known as an obnoxious straight line route, through a set $S = \{p_1, \ldots, p_n\}$ of points in the plane has resulted in algorithms for both the unweighted [32] and the weighted cases [22, 23]. Given S and a set of associated positive weights w_i, we search for a line ℓ, with at least one point from S on each side, that maximizes the distance to S, $\min_{p_i \in S} w_i d(p_i, \ell)$. In this subsection we shall refer to the weighted distance simply as distance and the optimal ℓ as the obnoxious line.

Houle and Maciel [32] describe a beautiful solution to the unweighted variant by taking the problem into the dual space. There, this problem, which can be thought of as that of searching for the widest empty corridor, can be solved by sweeping an arrangement of lines that came from the input points. Two types of candidate best corridors are identified - those bordered by lines that rest each on a single point and are perpendicular to the segment joining these two points and those where one of the lines rests on at least two points. In the dual arrangement, the corridor in the second case becomes a segment joining a vertex of a face with an opposing edge while the first case results in a segment abutted by two edges of the same face. What is important here is that in each case only segments within faces of the arrangement need be inspected and there is only $O(n^2)$ of them. Each candidate combination can be evaluated by going back into the primal space and performing the necessary calculations as per the case that this combination corresponds to. This takes constant processing per combination of input points. Thus, a topological search ([24]) is employed and yields a solution that runs in $O(n^2)$ time and $O(n)$ space.

Other work on this subject includes a paper by Janardan and Preparata ([37]) that introduces algorithms for k-dense widest corridors for any k, including the case of $n-1$, and an algorithm to maintain the widest empty corridor under online insertions and deletions as well as an algorithm that generalizes the input objects to polygonal obstacles. A recent paper [22] also considers this more general version of the problem (separating polygonal regions) and the algorithms proposed work for the case of points, too. The ideas in the paper also use duality and require $O(n^2)$ space. Furthermore, in order to complete the algorithm for the weighted case parametric search (refer to [44]) is utilized. Meanwhile, in an earlier paper by Drezner and Wesolowsky [23], the authors provide an $O(n^3)$-time and $O(n)$-space algorithm for computing the obnoxious line through a set of weighted points. However, upon careful examination of the proposed algorithm, we found a subtle error that renders the algorithm incorrect. Additionally, the problem cannot be easily corrected, without radically changing the algorithm.

In [23] the authors introduce formulas to compute various quantities, such as the distance to the weighted midpoint of the segment joining two points ($f_{ij}(\Theta)$) and the distance from a point to the line R_L equidistant either from one point on each side or from two points on one side and one on the other ($f_k(\Theta)$). For the actual formulas as well as the specifics of their derivations, the reader is referred to the original paper. Here, $0 \leq \Theta < \pi$ is the angle made by T, the vector perpendicular to R_L, with the positive x-axis. Other interpretations of Θ lead to inconsistencies in the distance formulas given in the paper.

The authors outline an algorithm consisting of two parts, *Procedure 1* and *Procedure 2*. The first computes for any two points i, j the perpendicular route ℓ through the weighted midpoint and checks that no other point is closer to ℓ. It remembers the pair i, j that yields the best such "empty corridor". *Procedure 2* then computes the distance $f_k(\Theta)$ from any other point k to the line R_L equidistant from i, j on one side of it and k on the other. For every pair i, j, it then classifies all such points k as "below" or "above" (our terminology) depending on whether the quantity $S_k(\Theta)$, the signed Euclidean distance from R_L to k, is positive or negative, respectively, and picks the one closest to its corresponding line for each set (see [23] for details). For each i, j, the algorithm then picks as its candidate for *Procedure 2* the point k which yields the larger of these two minimum distances. The intent is to choose the wider of the two weighted "corridors" on each side of R_L. The algorithm terminates by returning the overall maximum among all candidates picked by both procedures. In doing so, it appears that the authors are using, for the weighted case, properties that only hold for the unweighted case. Their assumption seems to be that in *Procedure 2* they do not need to check for "emptiness" of the candidate regions, as they are supposedly ensuring that by picking the k that gives the smaller of the weighted distances on each side. While perfectly reasonable for unweighted corridors, this fails in the weighted case. Here, for the same pair i, j, we have routes that have different slopes for different points k (unlike the unweighted case!). This means that for different k the weighted corridors may intersect even if they belong to two different sets (i.e., one "above" and one "below") and, as a result, a point k_1 may be inside of another point k_2's "corridor".

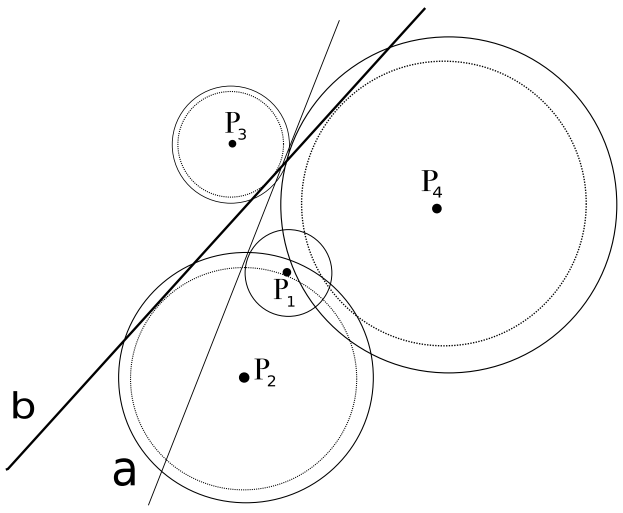

Figure 2.1: Counterexample to algorithm in [23].

This situation is illustrated in Figure 2.1 and Table 2.2. We start with points $p_1 = (5.92, 4.92)$, $p_2 = (5.92, -0.44)$, $p_3 = (2.25, 11.03)$, $p_4 = (12.64, 7.96)$, and weights 0.598802, 0.241546, 0.3709, 0.176678, respectively. *Procedure 1* investigates 6 pairs and finds that none of their weighted perpendicular bisectors yield proper corridors. *Procedure 2* then considers each of the 6 pairs in turn on the same side of ℓ and each of the remaining two points on the other side. This gives a total of 12 candidate separating lines, which are then placed for each pair into either the "above" or the "below" group. These results are tabulated in Table 2.2 which exposes the flaw in the algorithm: the last column shows the weighted distance f_h from a point h to the line determined by the other three. One can observe that sometimes p_h gets closer to the candidate separator than any of the other three points. This violates the "emptiness" of the candidate, yet if f_k is large enough, it may be picked by the algorithm while the true best separator is skipped as inferior. In our example,

Point	Location	Weight
p_1	(5.92,4.92)	0.598802
p_2	(5.92, -0.44	0.241546
p_3	(2.25, 11.03)	0.3709
p_4	(12.64, 7.96)	0.176678

Table 2.1: Points $p_1, \ldots, p_4$ in our counterexample to [23].

the line determined by p_1, p_4 and p_3 is reported as the final answer, yet p_2 is closer to it than the other three points. At the same time, p_2, p_4 and p_3 determine the line, whose f_k is only slightly smaller, but turns out to actually be empty (since p_1 is further away from it than any of these three points) and thus is the true best separator.

Pair	k	sign of $S_k(\theta)$	$f_k(\theta)$	$f_h(\theta)$
$(1,2)$	3	negative	1.59954	0.878394
	4	negative	1.00462	0.779681
$(1,3)$	2	positive	0.534473	1.43659
	4	negative	1.00579	0.904625
$(1,4)$	2	positive	0.918379	2.69556
	3	negative	*1.29187* ‡	0.991894
$(2,3)$	1	negative	0.314024	1.39185
	4	negative	0.979299	1.09746
$(2,4)$	1	negative	0.521644	2.72721
	3	negative	*1.26881* †	1.62224
$(3,4)$	1	positive	0.809943	1.45618
	2	positive	1.09385	0.0468722

Table 2.2: Optimal (†) and Procedure 2 (‡) solutions.

As already mentioned in the introduction, we present a correct algorithm for this problem in Section 5 that avoids the pitfall of [23] albeit at a cost of a somewhat higher time cost - $O(n^3 \log n)$.

Chapter 3

Unweighted Minimax Approximation with Step Function Curves

3.1 Background and Definitions

In this chapter, we first develop a new $\min$-$\#$ algorithm for minimax step function curve approximation which, after a preprocessing cost of $O(n)$, can solve multiple instances of the problem (with different target errors) at a cost of $O(\min\{k \log n, n\})$ time per instance. When coupled with the approach proposed by Wang, this results in an algorithm for the $\min$-ε problem than runs in $O(\min\{n^2, nk \log n\})$ time. This is asymptotically faster than the result in [49] when $k = o(n/\log n)$. We then propose an algorithm that has $O(n \log n)$ running time and yields an approximation curve with error within a factor of 3 of the optimal and with the same number of segments. Furthermore, a curve of size $2k - 1$

generated by our algorithm achieves an error which does not exceed that of an optimal curve of size k.

In the rest of this section, we define the problem formally as well as introduce notation that will be used throughout the paper. Let $S = \{p_i = (x_i, y_i), i = 1, \ldots, n\}, x_1 < x_2 < \cdots < x_n$, be a set of n points in the plane. For $1 \leq i \leq j \leq n$, define $S_{ij} := \{p_i, p_{i+1}, \ldots, p_j\}$. In the remainder of this and the next chapter, all approximation curves are step function and x-monotone as defined in Section 1.4.

Because of our definition of distance, it suffices to restrict our attention to those curves whose domain in x includes the interval $[x_1, x_n]$ and all of whose vertical segments fall strictly within it. We also assume that the first and the last segments of an approximation curve are horizontal and, if needed, can be extended arbitrarily far in the negative or positive x-directions. Given a horizontal segment s of a curve $\mathcal{R}$ with x-domain $[a, b]$, we say that a point p_i of S is "covered" by s if $x_i \in [a, b]$. Obviously, every point of S falls in the x-domain of some horizontal segment of $\mathcal{R}$ and the set of all points covered by s is some S_{ij} (which, as in [21], we call the *allocation set* of s). Furthermore, since we are not interested in increasing the complexity of $\mathcal{R}$ without changing its eccentricity, all allocation sets are nonempty, i.e. there is no horizontal segment that does not cover at least one point of S. We note that the quality of the approximation is not affected by moving the vertical segment in the space between the allocation sets of two consecutive horizontal segments of $\mathcal{R}$.

Let us call the y-span of points in the allocation set of s, the *range* of s. Then, it is clear (and shown in [21]) that in the search for an optimal solution to the $\min$-ε problem we may restrict our attention to curves whose horizontal segments are centered with respect to their range. Therefore, we may think of an approximation curve $\mathcal{R}$ with k segments as a partition of S into k subsets such that if $i < j$ and both p_i and p_j belong to the same subset s, then $\{p_{i+1}, \ldots, p_{j-1}\} \subset s$. Hence, $\mathcal{R}$ is completely specified by $k-1$ integers in $[2, n]$, which are the indices of the leftmost points of $k-1$ subsets (the first subset always starts at p_1). Moreover, denoting $\epsilon_{ij} = \frac{1}{2} \max_{p_r, p_s \in S_{ij}} |y_r - y_s|$, i.e., the error of a segment covering S_{ij}, and $\mathcal{E} = \{\epsilon_{12}, \epsilon_{13}, \ldots, \epsilon_{1n}, \epsilon_{23}, \ldots, \epsilon_{n-1n}\}$, the errors of all segments in a candidate curve $\mathcal{R}$ are members of $\mathcal{E}$ and, hence, so is the eccentricity of $\mathcal{R}$.

It is important to point out that the solutions to the problem of fitting a step function curve to a set of points are also applicable to the problem of fitting a curve to a set of vertical segments in the plane, which may come from measurements with associated error bounds. Each vertical segment is an interval along the y axis and has three values associated with it, x, y_{low}, and y_{high}. It is trivial to see that the only definition given earlier that needs to be changed under this new setting is that of the approximation error of a segment. We may assume that when the segment goes through a vertical data interval in its allocation set no error is introduced. If a horizontal segment s does not pass through a data interval p in its allocation set, the error with respect to p is the vertical distance from $y_{low}[p]$ to s or the vertical distance from $y_{high}[p]$ to s, whichever is smaller. Then, the best approximation range for s is obtained by subtracting the lowest y_{high} value from the highest y_{low} value of the data intervals in its allocation set if that quantity is positive, or otherwise is defined to be 0 (in which case, s can pierce all the intervals it approximates). The approximation error of s is $\frac{1}{2}$ of this range as it still makes sense for s to be centered with respect to it (obviously when range is 0 it is irrelevant where s is placed so long as it goes through all the data intervals it approximates). With this new definition of error, all step function approximation algorithms can also solve this more general problem.

3.2 An Optimal $\min$-ε Algorithm Based on Binary Search

In our solution to the $\min$-ε problem, we use Wang's sweep line algorithm to generate instances of the $\min$-$\#$ problem but solve each of these instances using a different approach. Let $k = \lceil \log_2 n \rceil$, where $n = |S|$, and let A be an array of size $2^{k+1} - 1$. The last 2^k elements of A contain the points of S sorted by x-coordinate, padded on the right with $2^k - n$ copies of (x_n, y_n) whenever $n < 2^k$. The entries of A can be interpreted as a full binary tree whose leaves are the (padded) elements of S and whose internal nodes occupy locations $A[1..2^k - 1]$. We adopt the convention that the parent of $A[i]$ is stored at location $\lfloor i/2 \rfloor$, as it is normally done for binary heaps (see [13], for example). Each of the first $2^k - 1$ elements of A stores the range of the y values for all points in its subtree. This information can be generated in $O(1)$ time per node by proceeding bottom-up, one level at a time. Thus, the tree can be built in linear time.

The algorithm creates an optimal k-curve with eccentricity $\leq \epsilon$ one segment at a time but does not necessarily investigate each point individually. Rather, it tries to build a segment by expanding it to cover progressively larger groups of points that are adjacent in the sorted array followed by a phase of contraction when (and if) the error of the segment exceeds ϵ. The segment is contracted by eliminating fewer and fewer points from it until the final correct breaking point is found. The expansion and contraction phases correspond to following upward and downward node sequences through the tree, which we shall now describe. The construction of the first segment starts at the first leaf node (i.e., $A[1] = p_1$) and proceeds upward as long as the union of y-ranges of the nodes examined (i.e., the error of the segment being built) is no more than ϵ. Each new node in the upward sequence is the parent of the previous node's right neighbor, i.e. the parent of the node whose array index in A is one greater than that of the previous node. Note that this relationship means that the new node may not necessarily be linked to the previous one by an edge, i.e. generally we do not traverse a connected path. If the previous node had no right neighbor, then the upward sequence must terminate since it means that the last leaf node is already covered and so the segment extends to include the nth point.

Consider the figure below that shows the structure built on the points in Table 3.1 and suppose that $\epsilon = 2$. In our example, nodes in the upward and downward sequences are designated with letters U and D, respectively, which are subscripted to indicate the segment being built. We start with p_1 and set the range of the first segment to p_1's y-coordinate, $[low_1 = 5, high_1 = 5]$. From the first node p_1 we move up to q_8, the ancestor of p_2 (which also happens to be the ancestor of p_1 as well). The y-range at q_8 is $[3, 5]$ and, therefore, its union with the current range $[5, 5]$ is less than the total y-range allowed ($2\epsilon = 4$). Therefore, it becomes the new range of the segment under construction and we move to the ancestor of q_9, which is q_4. The range at q_4 is $[1, 6]$ and since its union with $[3, 5]$ exceeds 4 units, the upward sequence terminates and we need to descend to the leftmost child of q_4 whose range unioned with $[3, 5]$ exceeds 2ϵ. In our example, the left child of q_4 is q_8 which had already been examined (but this need not always be the case as we shall see later). Therefore, we proceed downwards to q_9. Following the same logic as before, we first investigate the left child of q_9, p_3, whose y-coordinate is 6 and, therefore, the point can be added to our segment whose y-range now becomes $[3, 6]$. The right child's y-coordinate is 1 and so we cannot include it in this segment. Since we have reached the leaf level no further descent is possible and the first segment is finished.

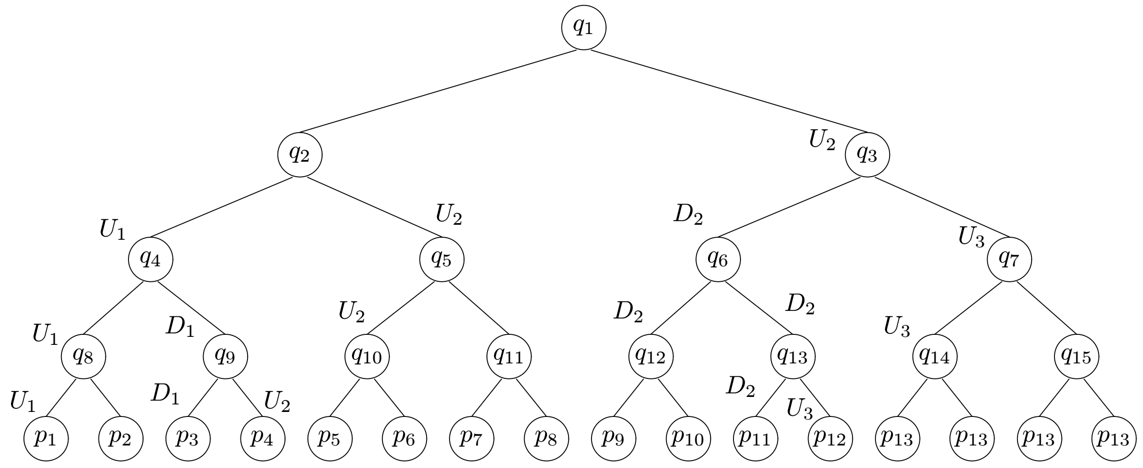

Figure 3.1: A tree structure constructed by the algorithm with 16 leaf nodes (storing 13 points) and 15 non-leaf "range" nodes. Upward and downward sequences are designated with letters U and D, respectively, and subscripted to indicate the number of the segment built.

The second segment is constructed starting at the leftmost point not included in the first segment, p_4. We then ascend to q_{10} (the ancestor of p_5) and q_5, increasing the range first to $[1, 3]$ and then to $[1, 4]$. At q_3 (whose range is $[2, 13]$) we descend to q_6. Then, we find that the left subtree of q_6 can be included in this segment since the range of q_{12} unioned with $[1, 4]$ is $[1, 5]$. This subtree does not need to be examined further and we descend to the right child of q_{13}, where we find that we may include p_{11} into this segment but not p_{12}. The third segment is constructed in a single upward sequence since at q_7 we find that there are no more nodes on that level.

Note that in each upward sequence no more than 1 node and in each downward sequence no more than 2 nodes from the same tree level are examined. Therefore, the total cost of creating a segment is proportional to $3 \log n$. Since in order to advance either of the sweeplines in Wang's $\min$-ε algorithm all we need to know is how the number of segments

Point id	x	y
p_1	1	5
p_2	2	3
p_3	3	6
p_4	5	1
p_5	6	2
p_6	9	3
p_7	11	3
p_8	12	4
p_9	13	5
p_{10}	14	2
p_{11}	15	4
p_{12}	19	9
p_{13}	22	13

Node id	low	high
q_1	1	13
q_2	1	6
q_3	2	13
q_4	1	6
q_5	2	4
q_6	2	9
q_7	13	13
q_8	3	5
q_9	1	6
q_{10}	2	3
q_{11}	3	4
q_{12}	2	5
q_{13}	4	9
q_{14}	13	13
q_{15}	13	13

Table 3.1: Point set used to build the tree in Figure 3.1 (left) and y-range information stored in its "range" nodes, q_1 - q_{15} (right).

k' in the candidate curve compares to the target number k, our $\min$-# algorithm may finish and simply return that $k' > k$ whenever any points are not covered after k segments have been finalized. This ensures that we spend no more than $O(k \log n)$ time building a single candidate curve.

The other upper bound for this $\min$-# algorithm can be shown by observing that for each segment the length of upward and downward node sequences depends on the number of points it would be extended to cover. Letting $n_i, 1 \leq i \leq k,$ be the number of points in the final allocation set of the ith segment, we know that no more than $3 \log 4n_i$ nodes were in both sequences for segment i since the highest node in the upward sequence could have at most $4n_i$ nodes in its subtree (since the subtree sizes in the upward sequence double every time we move up a level and the last node in this sequence may contribute no points to the allocation set, it may have $2n_i$ leaves in its subtree and therefore have height $\log 4n_i$). Hence, the total number of nodes visited by the algorithm is $\sum_{i=1}^{k} 3\log 4n_i \leq 3(2k+n) = O(n)$ since $n_1 + \cdots + n_k \leq n$. This implies that no matter what k is, our $\min$-$\#$ algorithm takes no more than linear time and so our solution to the $\min$-ε problem, which uses Wang's method of solving $2n$ $\min$-$\#$ problems, takes total time $O(\min\{n^2, nk\log n\})$ and is asymptotically faster than Wang's algorithm when $k < n/\log n$.

We also note that the above algorithm can be implemented without padding the leaf point data to bring its size to a power of 2. Then, the number of the non-leaf elements preceding S in A is $n-1$ if n is even and n otherwise. Now, the point with the leftmost x-coordinate may not necessarily be the leftmost leaf due to the fact that leaves may reside at two different levels of the tree. Therefore, we may have to allow some upward sequence that reached the rightmost node of some level to cross over to the leftmost node on that same level. This situation may occur exactly once during the execution of the algorithm and to detect it it is enough to keep track of the largest leaf index reachable from the currently inspected node.

3.3 An $O(n\log n)$ Approximation Algorithm

Let $k > 0$ and consider a set S of n points in the plane. The $\min$-ε problem calls for a curve that consists of no more than k horizontal segments (we shall call this a k-curve) and achieves the minimum possible vertical error (eccentricity). Let us denote an optimal k-curve for S by C^* and its eccentricity by ϵ^*. The algorithm in [49] finds C^* and ϵ^* in $\Theta(n^2)$ time. We sought a faster approximation algorithm to generate curves with eccentricity no more than ϵ^* and at most αk horizontal segments, where $\alpha > 1$ is a small constant. In this section we first describe our simple greedy algorithm and then in Subsection 3.3.1 show that $\alpha = 2$ always suffices. Furthermore, in Subsection 3.3.2 we prove that the eccentricity of a k-curve generated by our algorithm is within a factor of 3 from optimal.

We start with a set of n points stored in an array A. The *Greedy Combine Segment Approximation* (GCSA) Algorithm, described in Figure 3.2, is called on A with a single additional parameter m that specifies the number of segments in the output curve C. The first step of the GCSA algorithm is to sort the points in A by x-coordinate. Then, GCSA creates a doubly-linked list L of n segments (line 2), each going through a different point and linked in the order of appearance in A of the corresponding points. Thus, with a call to BUILDSEGMENTLIST, we begin with an n-curve of eccentricity 0 that consists of singleton segments ordered from left to right. In order to produce an m-curve, adjacent allocation sets will be merged and new longer segments created. GCSA follows a greedy approach minimizing the eccentricity of the resulting curve at every merge step.
The following list describes the fields found in each segment.

start	Index in A of the first point covered by the segment
low	Lowest y-coordinate of the points covered by the segment
high	Highest y-coordinate of the points covered by the segment
cost	Error of the segment merged with its left neighbor, ∞ for leftmost segment
left	Pointer to the left neighbor of the segment
right	Pointer to the right neighbor of the segment

```
GCSA($A, m$)
Sort $A$ in ascending order by $x$-coordinate
$L \leftarrow$ BuildSegmentList($A$)
if $size[A] \leq m$
  then return $L$
$H \leftarrow$ BuildCostHeap($L$)
while $size[H] > m$
do $s \leftarrow$ ExtractMin($H$)
   $low[left[s]] \leftarrow$ Min($low[left[s]], low[s]$)
   $high[left[s]] \leftarrow$ Max($high[left[s]], high[s]$)
   $right[left[s]] \leftarrow right[s]$
   if $cost[left[s]] \neq \infty$
     then $h \leftarrow$ Max($high[left[s]], high[left[left[s]]]$)
          $l \leftarrow$ Min($low[left[s]], low[left[left[s]]]$)
          $cost[left[s]] \leftarrow h - l$
          Heapify($H, left[s]$)
   if $right[s] \neq$ NIL
     then $h \leftarrow$ Max($high[left[s]], high[right[s]]$)
          $l \leftarrow$ Min($low[left[s]], low[right[s]]$)
          $cost[right[s]] \leftarrow h - l$
          $left[right[s]] \leftarrow left[s]$
          Heapify($H, right[s]$)
   delete $s$
return $L$
```

Figure 3.2: Pseudocode for the GCSA Algorithm.

Initialization of these fields is done inside of the function BUILDSEGMENTLIST. The fields *low* and *high* are set to the y-coordinate of the only point covered by the segment. The only field whose initialization requires a computation is *cost*, which is set to the length of the y-range of the points covered by the segment and its left neighbor. Since the leftmost segment has no left neighbor, its cost is ∞. Note that if $n \leq m$, the algorithm simply returns the curve obtained in this first step.

The next step is to prepare the list of segments for processing (merging) by prioritizing them according to cost, which we define as the eccentricity of the curve resulting from merging a segment with its left neighbor. This step may be done as in line 5 by building a min-heap on the segments using *cost* as key.

GCSA repeatedly extracts the segment with the minimum cost from the heap (line 7) and merges it with its left neighbor. Merging is carried out by updating the neighbor's fields (lines 8-14), re-heapifying (line 15), and then updating the fields of and re-heapifying on its right neighbor (lines 16-20). As part of this operation, the algorithm also maintains the adjacency pointers between segments (lines 10, 20). At every point in the execution of this loop, the eccentricity of the curve is dominated by the cost of the segment at the top of the heap and equals the error of the last generated segment. Another important observation is that the costs of the neighbors of an extracted segment can only increase, which justifies the calls to HEAPIFY in lines 15 and 21. Note that the leftmost segment is never extracted from the heap and as a result the splicing of nodes never changes the head node of L. The GCSA algorithm stops after $n - m$ iterations and L stores the segments of the curve sorted by x-range.

The following theorem proves the efficiency of GCSA.

Theorem 3.1. *The GCSA Algorithm produces the curve C in $O(n \log n)$ time.*

Proof. The sorting of the points in A takes $O(n \log n)$ time. The construction of the segment list takes linear time, since it involves creating a linked list of n segments. Building the cost heap H also takes $O(n)$ time. Finally, each iteration of GCSA takes $O(\log n)$ time, since in addition to the constant time spent on merging segments and updating adjacent costs, one Extract-Min and two Heapify operations are called on H. Because the algorithm terminates after $n - m$ steps, the total cost is therefore $O(n \log n + n + (n - m) \log n) = O(n \log n)$. □

Let us now introduce some terminology which will facilitate our discussion of the properties of curves constructed with GCSA. First, we shall define the cardinality of a step function curve.

Definition 3.2. The cardinality of a step function curve C, denoted $|C|$, is taken to be the number of horizontal segments contained in C (including the possibly semi-infinite beginning and ending segments).

We shall call a curve constructed by the GCSA algorithm (after any number of iterations) a *GCSA curve* and classify its segments through their relationship with the segments of some (fixed) optimal k-curve.

Definition 3.3. Let S be a set of points and C^* an optimal k-curve for S. A horizontal segment s of a GCSA curve C is called an *inside* segment *with respect to* C^* if its allocation set is a contiguous subset of the allocation set of a horizontal segment s^* of C^*. In that case, we say that s^* contains s and denote it $s \subseteq s^*$. Also, if $s \subseteq s^*$ and s^* covers a point not covered by s, we say that s^* properly contains s. Any segment of C which is not an inside segment with respect to C^* is a *straddling* segment (or a *straddler*) *with respect to* C^*.

Since we shall always keep the choice of C^* fixed throughout every argument, we shall drop the reference to C^* when qualifying the segments of C and simply speak of them as *inside* or *straddling* segments. When the allocation sets of a segment s of C and a segment s^* of C^* intersect, we shall simply say that s intersects s^*. Clearly, any segment of C can only intersect a subset of the segments of C^* that are adjacent. We shall also expand the last definition and differentiate between the straddlers that intersect exactly two segments of C^* and call these *simple* straddlers and all other straddling segments and name those *compound* straddlers. Obviously, no GCSA curve C may contain more than $k-1$ straddlers with respect to an optimal k-curve C^* (and only have that many when they are all simple) because there are exactly $k-1$ pairs of adjacent segments in C^*.

3.3.1 Worst-case Bound on the Cardinality of a GCSA Curve with Eccentricity ϵ^*

We now prove one of the two main properties of the GCSA algorithm.

Theorem 3.4. *If $n \geq 2k$, the GCSA Algorithm with $m = 2k-1$ produces a curve C with eccentricity $\epsilon \leq \epsilon^*$.*

Proof. Let S be a set of n points and $k > 0$. Let C^* be an optimal k-curve with eccentricity ϵ^*. Suppose to the contrary that the curve C returned by GCSA and consisting of $2k-1$ segments has eccentricity $\epsilon > \epsilon^*$. Then, C contains at least one segment s, whose error is equal to ϵ. Consider the situation just before s was created (i.e., before its cost is extracted from the heap) and call the curve constructed up to that point C'. Then, clearly, $|C'| > 2k - 1$, since s resulted from merging two segments. Now, let s^* be a segment of C^* and consider how many inside segments of C' may be contained by s^*. Assume there are two or more such segments of C'. Then, there must be at least one adjacent pair of these inside segments and the cost of merging any such pair is $\leq error(s^*) \leq \epsilon^*$ and so is $< \epsilon$ by hypothesis. Therefore, merging these pairs of segments has smaller cost than creating s contradicting the fact that the cost of s is at the top of the heap. Thus, we infer that all such pairs have been merged before and there can be at most one segment of C' completely inside of s^*. Since s^* was arbitrary, it follows each segment of C^* may contain at most one segment of C'. Then, since there can be no more than $k - 1$ straddling segments, C' has no more than $k + k - 1 = 2k - 1$ segments. However, earlier we established that $|C'| > 2k - 1$, and so we have arrived at a contradiction. We conclude that the curve C consisting of $2k - 1$ segments has eccentricity $\epsilon \leq \epsilon^*$. Since GCSA starts with n segments and with each iteration decreases their number by one, C is obtained after $n - 2k + 1$ iterations. □

Construction 3.5. We now show that the bound on m in Theorem 3.4 is tight. To this end, we construct an arbitrarily large set S of points whose GCSA $(2k - 2)$-curve has eccentricity bigger than ϵ^*. For $k = 2$, Figure 3.3a shows a set of $n = 4$ points along with an optimal 2-curve (dotted lines) and the GCSA curve with $2k - 2 = 2$ segments (solid lines). The middle pair of points in this figure is separated by vertical distance $1 - \delta$, while the points in the two other adjacent pairs are exactly 1 vertical unit apart from each other. If δ is very small, this shows that the GCSA 2-curve is almost twice as bad as an optimal one. Using two copies of this point set (with point D shared) in a V-shaped arrangement, we can

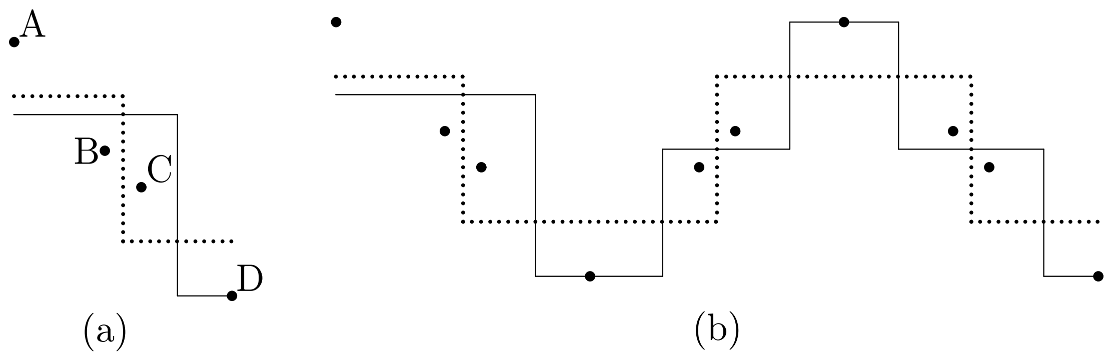

Figure 3.3: (a) A set of 4 points with the optimal 2-curve (dotted) and the GCSA curve approximating it (solid). The distance between the points A and B as well as between C and D is 1 unit, while the distance between B and C is $1 - \delta$, (b) 3 copies of the set in (a) adjoined in a V-shaped arrangement with the optimal 4-curve (dotted) and the GCSA 6-curve (solid) whose eccentricity is almost twice as bad.

show that an optimal 3-curve has eccentricity up to nearly 2 times better than the 4-curve obtained with GCSA. Figure 3.3b shows how we can build a bigger example with three copies of the same point set and this construction carries over to arbitrarily large numbers of points and segments. In general, an optimal k-curve may have eccentricity almost twice as small as that of the GCSA $(2k-2)$-curve. To see this, let $k = h+1$ and so $2k-2 = 2h$. With h copies of the point set in Figure 3.3a we have $3h + 1$ points and after merging in each copy the middle pair of points we obtain h 2-point segments and $h + 1$ singleton segments. This necessitates the creation of one more segment with error strictly bigger than the eccentricity of an optimal $(h + 1)$-curve.

3.3.2 Worst-case Bound on the Eccentricity of a GCSA Curve with Cardinality k

Before we proceed with our next result, we need to make one more definition.

Definition 3.6. For any $0 < i < n$ and the same set of points S, the GCSA curves produced after $0, \ldots, i-1$ merge steps are called the *ancestor curves* of the GCSA curve C obtained after the i^{th} step. Furthermore, any segment in an ancestor curve of C contained by a segment s of C is an *ancestor segment* of s (i.e., the ancestor segments of s are all those segments whose merging eventually produced s).

In the next two theorems, we shall make use of the following easy to prove lemma.

Lemma 3.7. *Let s and t be neighboring segments in a GCSA curve C. Suppose that with respect to some optimal curve C^*, s is an inside segment and t is a straddler with error $error(t)$. Then, $error(s \cup t) \leq \epsilon^* + error(t)$.*

In the next theorem, we shall prove an intermediate result that determines the minimum number of segments in a GCSA curve C for it to have eccentricity within a factor of 2 of optimal.

Theorem 3.8. *The GCSA curve C with at least $k + \lfloor \frac{k}{3} \rfloor$ segments has eccentricity at most $2\epsilon^*$.*

Proof. Consider the curve C generated by GCSA just prior to creating a segment with error $\epsilon > 2\epsilon^*$. Clearly, $|C| \leq 2k - 1$ (Theorem 3.4). First, suppose that C has no segment properly contained by a segment of C^*. Then, the conclusion of the theorem follows since in that case C is made up only of straddling segments or segments that also belong to C^*, and thus $|C| \leq k$.

Therefore, we assume that C has at least one segment properly contained by a segment of C^*. Then, let s be the leftmost such segment of C, inside of some segment s^* of C^*, that does not extend all the way to the right end of s^*. We observe that, if s exists, the number of segments in C to the left of s is no more than the number of segments in C^* to the left of s^*. If s does not exist, then $|C| \leq |C^*| = k$ and the theorem follows. We shall compute the greatest possible number of segments in C between s and the next inside segment r to the right of s and compare it to the number of segments in C^* between s^* and the segment containing r (note that r may or may not extend to the right end of that segment). Let C' be the last ancestor curve of C with eccentricity $\epsilon' \leq \epsilon^*$ (soon we shall see that because of the existence of s, C' must be different from C). Then, since $error(s) \leq \epsilon^*$, $s \in C'$ and from the proof of Theorem 3.4 we know that s has no neighbors in C' and, consequently, in C that are properly inside of s^*. Therefore, the right neighbor of s in C, call it t, "straddles" s^* and its right neighbor t^*.

Now, can $t \in C'$? Clearly not, as otherwise $error(t) \leq \epsilon^*$, and, therefore, merging s with t results in a segment whose error is less than ϵ (by Lemma 3.7, $error(s \cup t) \leq error(t) + \epsilon^* \leq 2\epsilon^*$), but by hypothesis ϵ resides at the top of the heap after C was created. Therefore, $t \notin C'$ and so t has an ancestor segment $t' \in C'$ that also straddles s^* and t^* (since t covers points of s^* that s does not, C' cannot contain an ancestor of t that covers only these points, otherwise merging s with that ancestor creates a segment of eccentricity $\leq \epsilon^*$). Now, t' covers all points of s^* that are covered by t and $error(t') \leq \epsilon^*$. Since both t and t' cover points of t^* and $t' \subseteq t$, it follows that t was obtained from t' by merging it on the right with another segment of C' and so C cannot have any segment contained in t^* (since at most one existed in C').

If the rightmost point covered by t is the same as the rightmost point covered by t^*, then C has no more segments up to that point than C^* (at most the same number of segments prior to s^* and then $\{s, t\}$ in C and $\{s^*, t^*\}$ in C^*). Now, suppose that t does not cover at least one point of t^*. Then, the right neighbor of t^*, u^*, cannot contain a segment of C. Assume otherwise and consider the segment u of C that straddles t^* and u^*. It must exist since we have already shown that t^* contains no segments of C and, since the one segment of C' that could be contained by t^* had to be used up to produce t, u must have an ancestor in C' that also straddles t^* and u^* and whose left endpoint is the same as u's. Moreover, u must be identical to its ancestor in C', which could not be merged on the right side either since, by assumption, the inside neighbor on the right survives. Therefore, we have a contradiction - u and the segment inside of u^* have to be merged before we get to C (again by Lemma 3.7)! Thus, there are at most two segments of C that cover points of u^*, u and another segment (call it v) that straddles u^* and its right neighbor r^*. Therefore, the next inside segment r of C to the right of s must be beyond u^* (the first segment of C^* that can contain it is r^*). Consequently, since every segment of C from s to r is straddling except s itself, there is at most one more segment in C up to r than there are segments in C^* up to the segment containing r. Since there are at least 3 such segments in C^* (namely, $\{s^*, t^*, u^*\}$ versus $\{s, t, u, v\}$ in C), it follows that C has at most $\lfloor \frac{k}{3} \rfloor$ more segments than C^*. It remains to consider the case when t ends in u^* or beyond. However, then t is a compound straddler, and so in that case C has no more segments spanning the range from

s to the left endpoint of r than C^* (again, since all segments other than s are straddling and t is a compound straddler). Thus, it is always true that $|C| \leq k + \lfloor \frac{k}{3} \rfloor$. $\square$

We are now ready to state and prove that not only does GCSA produce a good approximation curve with $2k - 1$ segments but that a curve with as few as k segments has eccentricity that is at most 3 times greater than optimal.

Theorem 3.9. *A GCSA curve C with k segments achieves eccentricity at most $3\epsilon^*$.*

Proof. Let C be the GCSA curve prior to creating a segment with error $> 3\epsilon^*$. Previously, we have shown that no two segments of C can be inside of the same segment of C^* (Theorem 3.4). Then, we have shown in the proof of Theorem 3.8 that no two inside segments of C can be adjacent to the same simple straddler. Now, we show that in our case no simple straddler can be a neighbor of an inside segment in C. This is easy. Suppose that s is such a simple straddler next to an inside segment t of C. Let C' be the last ancestor curve of C whose eccentricity is no more than ϵ^*. Since $t \in C'$, the proof of Theorem 3.4 implies that there is an ancestor of s in C', call it s', which is also a straddler. Moreover, s' is a neighbor of t in C' and, therefore, s was obtained from s' by merging on (at most) one side. Therefore, $error(s) \leq 2\epsilon^*$. It follows by Lemma 3.7 that $error(s \cup t) \leq error(s) + \epsilon^* = 3\epsilon^*$. Hence, s and t cannot both be present in C and we have a contradiction. Thus, we have shown that for every inside segment C must contain a compound straddler. Otherwise, if C has no inside segments, it consists only of straddlers and segments that are also in C^*. Therefore, we are done and $|C| \leq k$. $\square$

Construction 3.10. We shall illustrate with this example that situations when the GCSA algorithm produces k-curves with eccentricity arbitrarily close to $3\epsilon^*$ do arise. Figure 3.4

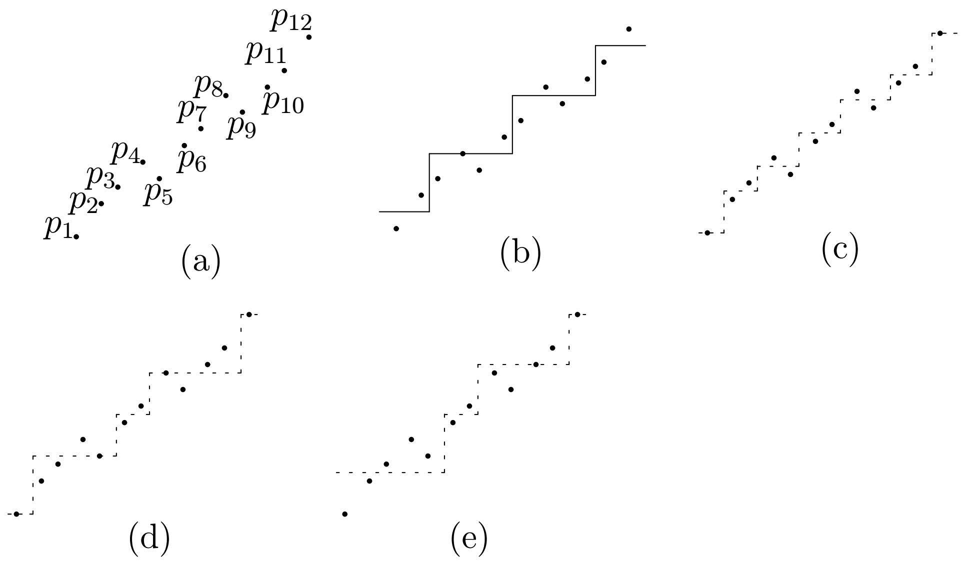

Figure 3.4: (a) A set of 12 points, (b) An optimal approximation, (c) Stage 1 of GCSA ($\epsilon = 1$), (d) Stage 2 of GCSA ($\epsilon = 2 + \delta$), (e) Stage 3 of GCSA ($\epsilon = 3 + 3\delta$).

shows one such situation with 12 points, numbered in the order of increasing x-coordinate, and Table 3.2 displays the y-coordinates as well as the vertical distance between consecutive points (δ is taken to be an arbitrarily small positive number). We let $k = 4$ and display the optimal 4-curve in Figure 3.4b. A quick look at the figure and Table 3.2 suffices to verify that the eccentricity ϵ^* is $1 + 3\delta$. In Figures 3.4c, d, e we show the construction of an approximating 4-curve with our algorithm, divided into 3 stages. The first stage (3.4c) results in a curve of eccentricity 1 and consists of merging 5 pairs of singleton segments (note that $12 - 5 = 7 = 2 \cdot 4 - 1$ and $1 < 1 + 3\delta$, so Theorem 3.4 holds). In the second stage, whose result is shown in Figure 3.4d, we merge two pairs of neighboring segments created in the first stage and separated by $1 + \delta$ and leave the middle segment intact as it is separated from each of its neighbors in 3.4c by $1 + 2\delta$. The eccentricity of the new curve

Point	y_i	$\|y_i - y_{i-1}\|$	Point	y_i	$\|y_i - y_{i-1}\|$
p_1	0	$--$	p_7	$4+5\delta$	1
p_2	$1+2\delta$	$1+2\delta$	p_8	$5+7\delta$	$1+2\delta$
p_3	$2+2\delta$	1	p_9	$4+7\delta$	1
p_4	$3+3\delta$	$1+\delta$	p_{10}	$5+8\delta$	$1+\delta$
p_5	$2+3\delta$	1	p_{11}	$6+8\delta$	1
p_6	$3+5\delta$	$1+2\delta$	p_{12}	$7+10\delta$	$1+2\delta$

Table 3.2: Vertical distances between neighboring points in Figure 3.4

is $1 + 1 + \delta = 2 + \delta$. Finally, we need to make the final merge and as we see from 3.4d there is no choice but to increase the eccentricity by $1 + 2\delta$. The final curve, therefore, has eccentricity $3 + 3\delta$ and one possibility for it is Figure 3.4e.

This example carries over to arbitrarily large n and k. Imagine putting together h copies of the 12 points in Figure 3.4a, with p_1 of the $(i+1)$st copy having x-coordinate greater and y-coordinate 1 less than those of p_{12} of the ith copy. Then, the optimal curve that extends the one in Figure 3.4b will have $4h-(h-1) = 3h+1$ segments (since the first and last two points of each copy of 3.4a will form one segment instead of two as in 3.4b; therefore, these $h-1$ segments at the junctures will have error $1 + 4\delta$). The GCSA curve with eccentricity $2+\delta$ (similar to the one in 3.4d) will have $5h - (h - 1) = 4h + 1$ segments, and the reader is invited to verify. Finally, as is clear from Figure 3.4d, with target eccentricity $3 + 3\delta$ we can make at least one merge in each of the h copies of the original point set, thus producing a curve with $3h + 1$ segments. Therefore, for arbitrarily large n and k, we have the ratio

$$\frac{\epsilon}{\epsilon^*} = \frac{3 + 3\delta}{1 + 4\delta} \to 3,$$

as $\delta \to 0$.

3.3.3 Experimental Results

We would like to emphasize that the curves shown in the worst-case examples of Subsections 3.3.1 and 3.3.2 are artificial constructions and are unlikely to occur in practice. The GCSA algorithm performs very well on both synthetic (correlated and uncorrelated) and real data and achieves approximation factors well below the upper bounds proven in Theorems 3.4 and 3.9. In fact, the experiments we have conducted on real datasets taken from websites such as the Time Series Data Library[1] and The Financial Data Finder[2] have produced near-optimal or optimal k-curves. The specific results are displayed in Table 3.3 for four data files and for several representative values of k.

The contents of the files used in our experiments are described below.

`arosa.dat`[1]	Ozone concentration measurements in Arosa, Switzerland, 1932-1972
`daily.asc`[2]	Daily Dow Jones Industrial Average stock price index data, 1915-1989
`pphil.dat`[1]	Monthly precipitation in mm in Philadelphia, 1820-1950
`tpmon.dat`[1]	Monthly temperature in England (F), 1723-1970

Inspecting the results in Table 3.3 reveals that GCSA produced highly accurate curves with k segments and always significantly outperformed k-optimal curves with $2k - 1$ segments. We note that for meteorological data, which has a limited range and is highly cyclic, GCSA produced optimal curves for most reasonable values of k. It is not surprising that for monthly weather data, such as temperature in England, the difference in eccentricity between a GCSA curve and an optimal one emerges at $k = n/5$ as this reflects how an op-

[1] *http://www-personal.buseco.monash.edu.au/~hyndman/TSDL/*
[2] *http://fisher.osu.edu/fin/osudown.htm*

File	n	$k = \log n$	$k = \sqrt{n}$	$k = n/20$	$k = n/10$	$k = n/5$
Ozone Data	481	1.00 (1.07)	1.00 (1.16)	1.00 (1.16)	1.00 (1.67)	1.10 (1.94)
Precipitation	1572	1.00 (1.15)	1.00 (1.18)	1.00 (1.24)	1.00 (1.33)	1.00 (1.62)
Temperature	2976	1.00 (1.07)	1.00 (1.09)	1.00 (1.14)	1.00 (1.69)	1.14 (1.89)
Dow Jones	18840	1.13 (1.38)	1.11 (1.39)	1.07 (1.82)	1.05 (1.96)	1.04 (2.56)

Table 3.3: Results of GCSA test runs on financial and meteorological data shown as ratios of GCSA k-curve eccentricities ϵ to those (ϵ^*) of optimal k-curves and ratios of optimal k-curve eccentricities ϵ^* to those (ϵ') of GCSA $(2k-1)$-curves (in parentheses).

timal curve takes somewhat better advantage of global correlation in the data caused in this case by distinct seasons. This situation is to be contrasted with the experiments on the Dow Jones historic index data, which exhibits a steady upward trend except for the period of decline and stagnation in 1929-1945. The GCSA approximation of this data becomes progressively better with larger k and is within 11% of optimal with just $\sqrt{n} = 137$ segments. Furthermore, we note that on this dataset GCSA computed its approximation $\sqrt{n}$-curve 1000 times faster than Wang's algorithm took to find an optimal curve of the same complexity. Overall, we conclude that with real data the GCSA algorithm quickly produces remarkably good k-curve approximations with eccentricity within 15% of optimal while the eccentricity of $(2k-1)$-curves always remains significantly below k-optimal.

Chapter 4

Weighted Minimax Approximation with Step Function Curves

4.1 Preliminaries

Recall from Section 1.6.1 that weighted metrics introduce for each point an additional coefficient that is used to multiply the distance from that point to the facility. To begin our discussion of the weighted case, we consider the optimal placement of a horizontal segment with respect to its (fixed) allocation set. In the unweighted case the error is minimized when the segment is centered with respect to the y-range of the points. Thus, the optimal location of the horizontal segment is unique and can be determined from two points in the allocation set. This is still true in the weighted scenario, but the optimal location may not correspond to the midpoint of the y-range. However, it must still be equidistant (under weighted distance) from the furthest points above and below it, as otherwise a small shift in its position would decrease the error.

There cannot be two different locations for the optimal segment because of the semi-monotonicity of the distance function. If two distinct segments s and s' were both optimal, then the distance from s' to one of the two points that define s would be greater than the distance from s to that point, in violation of the optimality of s'. If the two points that define the y-coordinate y_s of the best approximating segment s have coordinates $(x_i, y_i), (x_j, y_j)$ and corresponding weights w_i, w_j, then y_s is given by

$$(y_i - y_s)w_i = (y_s - y_j)w_j \Rightarrow y_s = \frac{y_i w_i + y_j w_j}{w_i + w_j}.$$

Therefore, the solution to the problem is the intersection of two lines $c = -w_i y + y_i w_i$ and $c = w_j y - y_j w_j$, where c stands for the cost of approximating the point by a segment located at y. This leads us to consider a "cost-location" space composed of such lines, each point in S giving rise to one upward and one downward sloping line with the absolute values of the slopes equal to the weight of the point. Let us suppose that all points in S are located in the first quadrant, i.e. $x_i, y_i > 0 \ \forall 1 \leq i \leq n$. We map each point p_i with the corresponding weight w_i to the pair of lines in the "cost-location" plane $\ell_{i0} = w_i y_i - w_i y$ and $\ell_{i1} = -w_i y_i + w_i y$ and restrict their domain to the first quadrant. Thus, for each point we have a linear transformation ℓ_i of the absolute value metric function restricted to the nonnegative domain. Each such wedge shaped function ℓ_i computes the distance from p_i to the approximating segment as we hypothetically sweep it upward starting from $y = 0$ and consists of a finite down-sloping segment (recording the cost for $y < y_i$) and an infinite up-sloping ray (for the cost when $y > y_i$). Which portion of this arrangement of $2n$

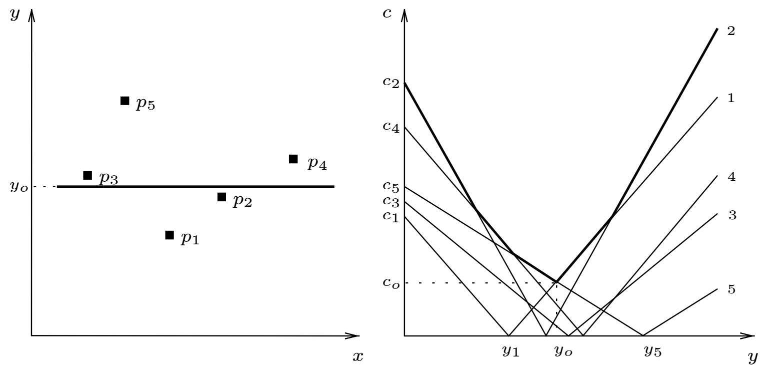

Figure 4.1: (a) A set of points $p_i = (x_i, y_i)$, numbered by increasing y-coordinate, having respective weights w_i such that $w_2 > w_4 > w_1 > w_3 > w_2$, and the best fit segment. (b) The corresponding lines in the cost-location plane with the slopes w_i and the vertical axis intercepts c_i. The lowest point of the envelope is identified with the cost and y-coordinate of the best fit segment.

cost lines keeps track of the greatest distance (i.e., the furthest point) to the approximating segment for any segment position y? The answer is quite obvious - the upper envelope of the arrangement is made up of the segments of the cost lines of those points that at some y are furthest from the approximating segment. The optimal location is given by the lowest point, which is also the lowest vertex, of the upper envelope.

We observe that in the case when S is known and fixed the above-mentioned problem of finding the lowest vertex of the upper envelope has been tackled successfully before, as it is nothing other than finding the optimal solution to a linear program in 2D. The best known deterministic algorithm for this has been developed by [39] and runs in $O(n)$ time. In addition, a very simple randomized algorithm [45] exists that has $O(n)$ expected running time. In our optimal algorithm we shall need to solve this problem repeatedly for each new subset of S, which differs in a single point from the previous subset, in order to compute the eccentricities of candidate curves and, therefore, using the $O(n)$ algorithm as a subroutine is an overkill.

However, there are more efficient algorithms to dynamically maintain common intersections of half-planes. In particular, a clever dynamization technique by Overmars and van Leeuwen [43] can be exploited to maintain the upper envelope in $O(\log^2 n)$ time per update (insertion or deletion of a line) and enables us to query for the lowest point on the boundary in just $O(\log n)$ time. The essence of their approach is to store the "left" half-planes (i.e., those that contain the left ray of any horizontal line) and the "right" half-planes in two separate augmented binary search trees. The lines bounding the half-planes are stored at the leaves and ordered by slope. In our case, since each point contributes an entire wedge with both bounding lines having the same (in absolute value) slope, it makes sense to have just one tree and store the points themselves at the leaves sorted by weight. Then, the bounding lines of the left half-planes are sorted in descending order and the bounding lines of the right half-planes are sorted in ascending order (without explicitly storing these lines). As per [43], each internal node is augmented with a pointer to the parent and the largest slope value (largest point weight) of the lines in its left subtree (needed for concatenation). Most importantly, the portion of the upper envelope of the left half-plane lines in its subtree that does not contribute to the upper envelope of the left half-planes of its parent is stored in a concatenable queue along with the number of lines on its envelope that belong to the upper "left" envelope of the parent. The "right" upper envelope is handled similarly, so each internal node has two concatenated queues associated with it.

Then, the overall "left" upper envelope is stored at the root of the "left" tree (and, similarly, the "right" upper envelope is stored at the root of the "right" tree). Using the procedures DOWN and UP described in [43] one can insert and delete lines and maintain the queue structures as well as the balance of the tree. Then, the intersection of the left and right envelopes can be found efficiently in $O(\log n)$ time as is also proven in the original paper.

Finally, we note that there are other dynamic half-plane intersection algorithms that outperform the above-mentioned algorithm by Overmars and Leeuwen and run in $O(n \log n)$ amortized time, such as [30] and [8].

4.2 An Exact Algorithm

As observed in the previous section, the error of each approximating segment in its best position is determined by two points and, therefore, so is the eccentricity of the curve. It is still valid to use Wang's choice of candidate eccentricities and then it remains to describe how to compute these and the candidate curves that they give rise to. In Wang's algorithm, when one of the two pointers (called sweep lines in the original paper) is advanced, the error of the best approximating segment for the set of points between the two pointers is computed. This error computation in the weighted distance case corresponds to finding the lowest point on the upper envelope of the wedge lines in the cost-location plane as these lines are added or deleted one at a time. As mentioned in the previous section, computing the candidate eccentricities can be done using the $O(\log^2 n)$ dynamic half-plane intersection algorithm of [43].

We now turn to the question of how to compute a candidate curve itself once the target eccentricity ε has been found. This can be done with a slightly modified $\min$-$\#$ algorithm of [21]. In this new version, each point (x_i, y_i) with the weight w_i is represented by a vertical line segment $v_i = (x_i, y_i - \frac{\varepsilon}{w_i})(x_i, y_i + \frac{\varepsilon}{w_i})$. Then, the algorithm proceeds in essentially the same way as described in [21]. We build horizontal segments of the curve by piercing consecutive vertical segments v_i. At first, we initialize the allocation set of the first horizontal segment to the single point (x_1, y_1) and define its corridor to be $(y_{min} = y_1 - \frac{\varepsilon}{w_1}, y_{max} = y_1 + \frac{\varepsilon}{w_1})$. Then, adding each additional point p_i to the allocation set causes the segment's corridor to be updated to $y'_{min} = \max\{y_{min}, y_i - \frac{\varepsilon}{w_i}\}, y'_{max} = \min\{y_{max}, y_i + \frac{\varepsilon}{w_i}\}$. We keep extending the current horizontal segment of the curve for as long as adding new points does not cause the corridor to become empty, i.e. until further expansion of the allocation set would make $y'_{min} > y'_{max}$. Therefore, computing both the candidate eccentricity and curve takes $O(n)$ time leading to the following result.

Theorem 4.1. *The weighted step function approximation problem can be solved in $O(n^2)$ time.*

This time bound becomes considerably reduced if the number of distance weights associated with the points of S is equal to a constant. In this case, the line wedges in the cost-location plane only have a constant number of distinct slopes. It is easy to see that for any given slope only one line wedge with that slope may contribute to the downward (and, similarly, upward) portion of the upper envelope. Furthermore, in our case, it is obvious that only the line wedge that contributes the first segment to the downward portion may also contribute a segment to the upward portion (due to the fact that all other line wedges that are part of the downward portion have smaller slope and a further x-intercept than the first one). All other line wedges may contribute only to one of the two portions. This means that the upper envelope consists of no more than $n + 1$ segments. In the case of a constant number of slopes c, we have no more than $c + 1$ segments on the envelope and, therefore, the above algorithm runs in linear time. This is summarized in the next theorem.

Theorem 4.2. *The weighted step function approximation problem with a constant number c of distance-modifying weights can be solved in $O(cn)$ time.*

4.3 A Heuristic with Provable Bounds

In the previous chapter, the author has described the GCSA approximation algorithm for the problem of step function curve fitting. Recall that the algorithm begins by building a curve consisting of n singleton segments and computes the costs that would result from merging the allocation sets of each adjacent pair of such segments. These costs are prioritized by storing them in a min-heap and, subsequently, at each iteration the minimum cost is extracted and the pair of associated segments is merged. The algorithm then updates the structure and the costs that involve the newly created enlarged segment and its neighbors.

We now modify this algorithm to be able to solve the weighted version of the same problem. While the overall structure of the algorithm shall remain unchanged, we have to supply new details for the merge step and analyze how these affect the overall running time. Merging two allocation sets can no longer be accomplished in constant time since the points responsible for the error of the new larger segment are not necessarily a subset of the points defining the placement of the old segments. Recall that the y-coordinate of the new longer segment s is determined by a pair of points whose so-called cost lines in the cost-location plane define the lowermost point of the upper envelope of all such cost lines coming from the points in the allocation set of s. Clearly, the cost lines that define this point come from the upper envelopes of the old segments' cost lines. Hence, the placement of the new longer segment can be determined by any two points whose cost lines were on the upper envelopes of their respective segments. We, therefore, have to keep track of the points defining these upper envelopes for each allocation set (upper envelope points).

Each of these points contributes at most two edges to the upper envelope and no two edges on the same envelope have overlapping x-ranges except at the boundaries. We can, therefore, store these in a binary tree ordered by x-range with pointers going to the original points. We also note that ordering the edges by x-range also has the effect of sorting them by slope as well as inducing a semi-sorted order on their y-ranges, since these decrease until the lowest point on the envelope and then monotonically increase. Furthermore, all upper envelopes are necessarily concave down, an important property that will be of use later.

When "merging" the allocation sets of two curve segments, their upper envelopes S (for *small*) and B (for *big*) need to be "merged" to produce the upper envelope of the new segment. Suppose that $|B| = n, |S| = m$ and $n > m$ (where the cardinality of an envelope is equal to the number of lines contributing segments to it). When we merge S and B, we always traverse S sequentially and B sometimes sequentially (when B is below S) and sometimes logarithmically (when B is above S). Clearly, the segments that survive (either partially or in their entirety) are on the upper envelope of $S \cup B$. Therefore, we need to find all points of intersection between S and B (for this is where they switch roles, one going below the other) and stitch together those portions that contribute to the overall upper envelope. Hence, when B is below S, we remove segments from B one by one (in $O(\log n)$ time per segment) and replace them by segments from S. Once removed, these

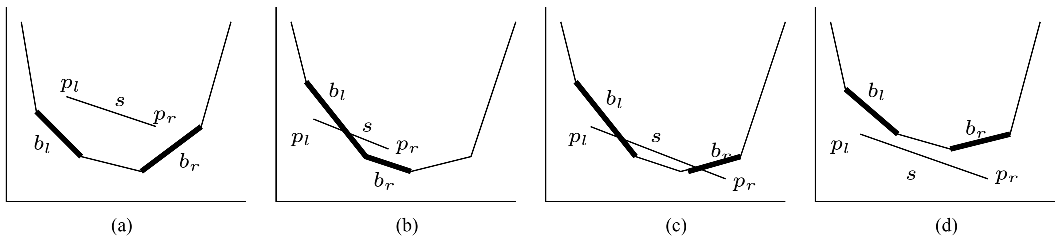

Figure 4.2: (a) Case I: both endpoints of s are above the bigger envelope B. (b) Case II: p_l is below a segment b_l of B while p_r is above B (same as p_l above B and p_r below). (c) Case III: both endpoints of s are below B and an intersection exists. (d) Case IV: endpoints of s are as in Case III but there is no intersection.

lines (i.e., points in the allocation set) will never contribute to the upper envelope. When B is above S, that portion of B needs to be preserved and traversing it sequentially in order to find the next intersection between S and B would lead to a linear amortized time per merge and, thus, to the total quadratic time for the entire algorithm (consisting of $O(n)$ merges).

We begin with the leftmost segments of S and B. As we move along S, for each of its segments s with endpoints p_l, p_r we locate (via a binary search) the segments b_l, b_r (potentially, $b_l = b_r$) in B whose x-ranges contain the x-coordinates x_l, x_r of those endpoints. In the case of ties, when the endpoints of two segments of B have x-coordinate x_l or x_r, we always pick the segment of B that begins at x_l and ends at x_r. We then test if p_l is above or below b_l and, similarly, whether p_r is above or below b_r. If p_l or p_r coincide with the endpoints of b_l or b_r, we test whether s itself is below or above b_l or b_r. If both p_l and p_r (or s itself in the case of coinciding endpoints) are above the segments of B, then because of the concavity of upper envelopes we know that s is completely above B (Figure 4.2a) and, therefore, it must be added to the upper envelope of $S \cup B$ and all segments of B from b_l to b_r (except, perhaps, b_r itself if its x-range is not completely covered by the x-range of

s) can be removed from consideration. As a way to simplify and speed up the process, we create the upper envelope of $S \cup B$ completely inside of the data structure for B. Therefore, all deletions of segments of B and insertions of the segments of S are carried out straight on the binary tree containing B with the result that after the merge is complete B contains the final "merged" envelope.

If one of the endpoints of s (again, in the case of endpoints coinciding, s itself) is above B and the other is below B, then an intersection exists (Figure 4.2b) and can be found in logarithmic time by simply doing a binary search on the segments of B and testing them as being above or below s, or simply walking along B starting from the segment which is below s and deleting segments from B until we arrive at the intersection at which point we link up with s. Thus, all segments of B below s are removed (again, except perhaps for b_r even if it is below s) and a portion of s is added to B (starting or ending at the intersection point, depending on which part of s is above B).

Finally, we come to the case when both endpoints of s are below B, which leads to the two possibilities illustrated in Figures 4.2c and 4.2d. In this case, there may or may not be an intersection and some extra work needs to be done to determine this. Namely, we certainly do not have an intersection when s belongs to the downsloping part of S and p_l is below the upsloping part of B or vice versa, when s has an upward slope and p_r is below the downsloping part of B. However, this is not sufficient to decide whether there is an intersection between s and B. These cases, then, are subsumed by the following simple check. First, we determine whether the slope of s is between the slope of b_l and that of b_r (remember, that slopes uniformly increase from b_l to b_r). Only if it is, there may be an intersection. We then find, via a binary search on the slopes of lines between b_l and b_r, the line b of B that has slope closest to that of s. If this line is not above s (Figure 4.2c), then we have two intersections which can be found by walking from b in opposite directions, while deleting segments from B. Otherwise, there is still no intersection (Figure 4.2d). To see that this is indeed a correct strategy, we remember that if s were to pierce B it must either intersect or "obscure" the line with the closest slope since in the resulting envelope lines must appear in the order from smallest to largest slope.

To complete the description of the modified GCSA heuristic, we need to address one more problem and that is the computation of the merge cost, i.e. the eccentricity of the resulting curve if the two allocation sets were merged. This, however, can be achieved with the same algorithm as above except that no changes should be made to B (i.e., we "simulate" a merge) and we can stop once the lowest point on the envelope has been found (note that this technique cannot be used for the exact algorithm in the previous section for it only handles envelopes obtained by merges and does not handle those obtained by deleting lines).

Let's analyze now the running time of this new GCSA algorithm. We first look at the operation of a single merge step involving the smaller allocation set S with $|S| = n_S$ and the bigger allocation set B with $|B| = n_B$. How many times can a segment of S's envelope intersect B's envelope? The answer is at most twice, since envelopes have parabolic shape. Therefore, only one part of a segment of S or that segment in its entirety can be inserted into B's envelope and since the number of segments in the envelope is at most one more than the size of the allocation set, no more than $n_S + 1$ insertions take place. Therefore, the total cost of insertions per merge step is $O(n_S \log n_B)$. It remains to sum the cardinalities of all such smaller allocation sets S participating in merge steps. This question can be approached from the point of view of how many times, at the most, the same point can belong to the smaller set over the course of all merge steps. This is very similar to the analysis of the disjoint data set union operation and we know that the same point can be merged from a smaller set at most $\log n$ times, for the sizes of the smaller sets it is part of will in the worst case increase as the sequence $1, 2, 4, 8, \ldots$ So, the number of insertions over all merge steps is at most $O(n \log n)$ and with each insertion taking $O(\log n)$ time, the total time is $O(n \log^2 n)$. We still need to remember to account for the deletions taking place during merging, but this is easy for once a line has been removed from an envelope, the point responsible for it will no longer be considered. Hence, the total cost of deletions is only $O(n \log n)$. Also, "simulating" a merge to compute the prospective eccentricity has the same cost as an ordinary merge and we know that only at most two such simulations are needed for every real merge step. Thus, we can perform all $O(n)$ merges in $O(n \log^2 n)$ time. This gives us the following result.

Theorem 4.3. *The modified GCSA algorithm runs in $O(n \log^2 n)$ time and guarantees the error bounds proven for the original GCSA. Namely that for $n \geq 2k$, the GCSA algorithm with $m = 2k - 1$ produces a curve C with eccentricity $\epsilon \leq \epsilon^*$ and with $m = k$ segments achieves eccentricity at most $3\epsilon^*$.*

The above claims regarding the error bounds follow directly from the proofs given in Section 3, as they carry over verbatim to this modified version of GCSA.

4.4 A Randomized Algorithm

In this chapter, we discuss a new algorithm with good expected performance for step function approximation in both unweighted and weighted settings. The main idea of this algorithm is to perform an efficient search on the set of $O(n^2)$ possible eccentricities but, unlike [21], the entire set of eccentricities is not generated explicitly. Instead, only those for which a candidate curve is constructed are computed. This results in $O(\log n)$ candidates on average and $O(n \log^2 n)$ expected running time. We begin by describing the unweighted version of the algorithm and then show how to extend it to handle weights.

The algorithm starts by picking a random pair of points p_i and p_j and computing the eccentricity of the allocation set S_{ij}. This can be done in $O(n)$ time (e.g., using the linear programming algorithm in [39]). Then, using the $\min$-$\#$ algorithm of [21], the first candidate curve R_{ij} of size k_{ij} is constructed and compared against the target k. The result of this comparison is to be used to decide about the bounds on the achievable eccentricity. The algorithm, therefore, keeps track of the feasible eccentricity window $\mathcal{E}_f = [\varepsilon_{min}, \varepsilon_{max}]$, which is updated after investigating each candidate curve. This window is initialized to $[0, \infty)$. Now, if $k_{ij} \leq k$, we update the window to $[0, \varepsilon_{ij}]$. While, in the opposite case of $k_{ij} > k$, we know that the eccentricity has to be increased, and so the feasible window becomes $[\varepsilon_{ij}, \infty)$.

Now, to discard all allocation sets whose errors are outside of the feasible eccentricity window, we create a data structure that records for each point p_i of S the number of allocation sets that start at p_i and end at some p_j with errors still in the current feasible window as well as the smallest index l_i and the largest index r_i such that $i \leq l_i \leq j \leq r_i$. For each p_i and a given feasible eccentricity window $\mathcal{E}_f$, we thus have the set $S_i^{\mathcal{E}_f}$ of possible values of j. In this set, $j = l_i$ specifies the index of the closest (in x-direction) point to p_i such that the error of the allocation set $\{p_i, \dots, p_{l_i}\}$ is at least ε_{min} and, similarly, $j = r_i$ gives the furthest point from p_i with the error of the allocation set $\{p_i, \dots, p_{r_i}\}$ at most ε_{max}. To see that p_j runs across a contiguous subset of S, we note that the eccentricity of an allocation set S_{ij} is monotonically non-decreasing as i is kept fixed and j is advanced.

Let us now explain how to compute for each p_i the cardinality and bounds l_i, r_i of $S_i^{\mathcal{E}_f}$ as well as how to maintain this information as $\mathcal{E}_f$ changes. After the first candidate curve is generated and $\mathcal{E}_f$ is initialized, we compute $|S_1^{\mathcal{E}_f}|$ and l_1, r_1 by scanning S. We then note that $l_i \leq l_k, r_i \leq r_k$ whenever $i \leq k$. This holds because the y-range of the set $\{p_k, \dots, p_{l_i}\}$ (possibly empty if $k > l_i$) is subsumed by the y-range of the set $\{p_i, \dots, p_{l_i}\}$ forcing l_k to be no less than l_i and, similarly, for r_k and r_i. Therefore, it seems that l_2 and r_2 can be found by simply moving ahead the pointers from l_1 and r_1, respectively, if needed. Unfortunately, in order to know when to stop for l_2 and r_2 we need to know the error of the allocation sets that begin at p_2 rather than p_1 for it could be that p_1, which is now removed from consideration, was one of the two points determining the error of $L_1 = \{p_1, \dots, p_{l_1}\}$ or the two points determining the error of $R_1 = \{p_1, \dots, p_{r_1}\}$. This necessitates the creation of two priority queues, such as min-max heaps, to keep track of the lowest and highest points in the two allocation sets L_i, R_i as they are being determined for each p_i. Then, in the case of p_2, we set $L_2 = L_1, R_2 = R_1$ and then remove p_1 from the heaps for each of the sets. Then, we start adding points to L_2 beginning with p_{l_1+1} and stop after having added the first point that had caused the error of L_2 to exceed or become equal to ε_{min}. Similarly, we add points to R_2 until adding the next point would make the error of R_2 become greater than ε_{max}. We remember the index of the last point added to L_2 as l_2 and that of the last point added to R_2 as r_2. All subsequent bounds for $S_i^{\mathcal{E}_f}, 2 \leq i \leq n$, can be found by advancing these two pointers, each making at most one full pass through S. Finally, computing the size of $S_i^{\mathcal{E}_f}$ is trivial as it is just $|r_i - l_i + 1|$. We note that every point is added to each of the two queues exactly once and is removed at most once as we compute all values of i_l, i_r for a given eccentricity window $\mathcal{E}_f$.

Thus, every time $\mathcal{E}_f$ changes, recomputing the bounds and cardinality information takes only $O(n \log n)$ time since it only involves $O(n)$ heap operations. Hence, the key to good performance becomes reducing the (expected) number of changes to the feasible eccentricity window that are necessary to process before the optimal eccentricity is found. This goal we achieve through randomization as we shall describe next.

After the initial step has determined $\mathcal{E}_f$ and the bounds and cardinalities of each of the sets $S_i^{\mathcal{E}_f}$ have been computed, we pick a pair of points that enclose an allocation set with error in the feasible eccentricity window at random from the set of all such possible pairs. In order to do this, and have a uniform distribution of probabilities, for each $2 \leq i \leq n$ we sum up the cardinalities of the sets $S_k^{\mathcal{E}_f}$ for all $k \leq i$ and store this number for p_i, i.e. we have

$$K_i = \sum_{k=1}^{i} \left| S_k^{\mathcal{E}_f} \right| .$$

Clearly, these can be computed in one scan of the array since K_i is just the sum of K_{i-1} and the cardinality of $S_i^{\mathcal{E}_f}$. Then, we can generate a pseudo-random number x between 1 and K_n and identify the unique pair (p_i, p_j) corresponding to this index (we just search for x in the array of K_i's, find the smallest i_0 such that $K_{i_0} \geq x$, and then find the unique j from $S_{i_0}^{\mathcal{E}_f}$.

Using this method, we ensure that each pair is selected with the same probability, but only from the set of those pairs that already fulfill the criteria for the error of its allocation set. Thus, we can expect that on average picking a new pair will reduce the number of pairs in the feasible eccentricity window roughly in half and so, our search has an expected logarithmic number of steps in the size of the set of possible eccentricities, that is, $O(\log(n^2)) = O(\log n)$. Since after each pair is picked, an $O(n \log n)$ time is spent updating the auxiliary arrays described above and constructing a candidate curve, the total expected running time of this randomized algorithm is $O(n \log^2 n)$.

Notice that even though the discussion has so far focused on the unweighted case only, our algorithm can be easily adapted to the weighted case. First, we observe that it is still true in the presence of weights that $l_i \leq l_k, r_i \leq r_k$ for any two points p_i, p_k such that $i \leq k$. Clearly, not just the y-range, but the weighted error range of the set $\{p_k, \ldots, p_{l_i}\}$ is subsumed by the weighted error range of $\{p_i, \ldots, p_{l_i}\}$ because the lowest point on the upper envelope of a set of cost lines can be no lower than the lowest point on the upper envelope of its subset. Consequently, instead of min-max heaps to keep track of the errors of L_i and R_i we would have to maintain upper envelopes and we can do so again using the algorithm from [43]. Each individual update of that structure takes $O(\log^2 n)$ and so one full pass through the array to update the pointers l_i, r_i for all i takes $O(n \log^2 n)$. Hence, the total time is $O(n \log^3 n)$ as there are still $O(\log n)$ candidate curves to construct.

Chapter 5

Weighted Maximin Approximation with a Single Line

5.1 An $O(n^3 \log n)$ General Algorithm

In this section we describe an algorithm for locating the “obnoxious” line. Recall from our introductory discussion that a facility is obnoxious if the goal is to place it as far as possible from the closest input point. Given the nature of the problem, it is simple to show that a maximal separating line (in the sense that any sufficiently small perturbation, rotation or translation, leads to worse solution) must be equidistant from at least two points, one in each half-plane. We call these the “witness” points of the separating line. It is perhaps less obvious that the best line, regardless of its orientation, must pass through the weighted midpoint of its witnesses. This property, described in [23], is exploited differently here to synthesize a different algorithm.

For each pair of candidate witnesses p_i, p_j, we find the maximal separating line ℓ_{ij} as the line that passes through the weighted midpoint of p_i and p_j making the least angle with the weighted perpendicular bisector of the segment $\overline{p_i p_j}$ (which, without loss of generality, we assume to coincide with the x-axis) and such that there is no other point p_k closer to ℓ_{ij} than p_i or p_j. Effectively, this combines the work done by the two procedures of [23] into a single algorithm, except that every candidate line is tested to be a proper separator (i.e., the condition of emptiness is ensured). Therefore, we look for a solution to the problem below.

Problem 5.1. *MSTWP: Maximal Separator of Two Weighted Points.* Given a set of weighted points $P = \{p_1 = (x_1, y_1, w_1), \ldots, p_n = (x_n, y_n, w_n)\}$ and two witness points p_i, p_j aligned on a vertical line, find the separator line ℓ_{ij} that maximizes the least distance to P and passes through the weighted midpoint of p_i and p_j.

Let ϕ be the counterclockwise angle measured from the positive x-axis to a given candidate orientation $\ell_{ij}(\phi)$ of ℓ_{ij}. Clearly, unless emptiness is violated, we would like to keep ℓ_{ij} as the weighted perpendicular bisector itself. Otherwise, we have to exclude all orientations from $[0, \frac{\pi}{2}) \cup (\frac{\pi}{2}, \pi)$ that would result in non-empty "corridors" (i.e., some point of P being closer to $\ell_{ij}(\phi)$ than the witnesses). We then have to find the smallest angle $0 < \phi_1 < \frac{\pi}{2}$ and the largest angle $\frac{\pi}{2} < \phi_2 < \pi$ from those orientations that remained available. The best separator line has the orientation ϕ_1 if $\phi_1 < \pi - \phi_2$ and ϕ_2 otherwise.

We solve this problem by converting each point $p_k \in P, k \neq i, j$ to at most two intervals of angles ϕ_k that result in p_k being closer to $\ell_{ij}(\phi_k)$ than the two witnesses. In many cases, there is only one interval (ϕ_k^s, ϕ_k^e) of values of ϕ that correspond to such "bad" separators $\ell_{ij}(\phi)$. However, in the case when $\ell_{ij}(0)$ is closer to p_k than to the witness points, we get two intervals $[0, \phi_k^e), (\phi_k^s, \pi)$ to be excluded from further consideration. In either case, and for a fixed k, all of these angle intervals lie entirely either in $[0, \frac{\pi}{2})$ or in $(\frac{\pi}{2}, \pi)$, since $\ell_{ij}(\frac{\pi}{2})$ passes through both p_i and p_j and is not a separator line at all. The figure above illustrates the process of investigating various candidate orientations ϕ.

Naturally, we have reduced MSTWP to the problem below (since it is the same as finding the best $\phi = \min\{\phi_1, \pi - \phi_2\}$).

Problem 5.2. *LRE: Least Right Endpoint.* Given intervals $(a_1, b_1), \ldots, (a_n, b_n)$ in $[0, M)$, find the leftmost right endpoint b_i such that $b_i \notin (a_j, b_j) \forall j$.

We now argue that MSWTP can be solved optimally in $O(n \log n)$ time. (This does not necessarily mean that the original problem has a lower bound of $\Omega(n^3 \log n)$ even though we can solve it with $O(n^2)$ instances of MSWTP.)

Claim 5.3. MSWTP can be solved in $\Theta(n \log n)$ time. This is optimal.

Proof. (By reduction from Connected Union (CU) [2]). Let $(c_1, \ldots, c_n, \epsilon)$ be an instance of CU. We assume that all c_i's and ϵ have already been shifted and scaled to fit inside $[0, \frac{\pi}{2})$, i.e. that $\min_i(c_i - \frac{\epsilon}{2} = 0)$ and $\max_i(c_i + \frac{\epsilon}{2}) < \frac{\pi}{2}$. Let $p_1 = (0, 1)$ and $p_2 = (0, -1)$ be the two "witness" points for the separator line. We set the weights of p_1 and p_2 to be 1 (we shall see that an unweighted version of this problem has the same complexity). Then, any separator line ℓ_{12} passes through the origin. Now, construct the points $p_3, \ldots, p_{n+2}$ one by one based on $c_1, \ldots, c_n$ and ϵ as follows. Let $\phi^s_{i+2} = c_i - \frac{\epsilon}{2}$ and $\phi^e_{i+2} = c_i + \frac{\epsilon}{2}$ be two angles in $[0, \frac{\pi}{2})$. Then, since both $\phi^s_{i+2} < \frac{\pi}{2}$ and $\phi^e_{i+2} < \frac{\pi}{2}$, p_{i+2} is in the first quadrant of Figure 5.1 and above the separator line with slope $\tan(\phi^s_{i+2})$ and below the separator line with slope $\tan(\phi^e_{i+2})$. So, we can compute the coordinates x_{i+2}, y_{i+2} of p_{i+2} by solving the

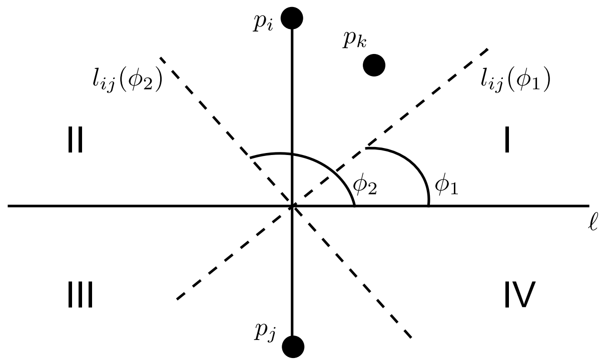

Figure 5.1: Finding the orientation of the maximal separator. For a pair of witness points p_i, p_j, the angle range $[0, \frac{\pi}{2}) \cup (\frac{\pi}{2}, \pi)$ of possible orientations ϕ is considered and all angles resulting in $\ell_{ij}(\phi)$ closer to some p_k are excluded. Each p_k results in at most two subintervals of angle values removed from further inspection. For p_k in quadrants I and III, we have $0 < \phi_k^s < \phi_k^e < \frac{\pi}{2}$ or $0 < \phi_k^e < \frac{\pi}{2} < \phi_k^s < \pi$, and for p_k in quadrants II and IV, we have $\frac{\pi}{2} < \phi_k^s < \phi_k^e < \pi$ or $0 < \phi_k^e < \frac{\pi}{2} < \phi_k^s < \pi$.

two equations given by the latter equalities in each of the following

$$\tan(\phi_{i+2}^s) = \frac{w_{i+2}y_{i+2} - w_1 y_1}{w_{i+2}x_{i+2} - w_1 x_1} = \frac{y_{i+2} - 1}{x_{i+2}},$$

$$\tan(\phi_{i+2}^e) = \frac{w_1 y_1 + w_{i+2}y_{i+2}}{w_{i+2}x_{i+2} + w_1 x_1} = \frac{y_{i+2} + 1}{x_{i+2}}.$$

Since the choice of w_i's is immaterial, we fix $w_i = 1$ for all $1 \leq i \leq n+2$. Then, we end up with two equations in two unknowns (x_{i+2}, y_{i+2}) and can find the coordinates of each $p_i, 3 \leq i \leq n+2$. We can then find the best separator line ℓ_{12} and if one exists with non-horizontal orientation we know that its slope is not properly inside of the slope ranges for any of the points p_i and, therefore, the union of the intervals from which these points came must be disconnected. Otherwise, we conclude that all angle ranges overlap and, therefore, the union of the open intervals is connected. Hence, since the transformation of the intervals into points takes linear time, we have shown that the lower bound of $\Omega(n \log n)$ carries over from CU to MSWTP.

MSWTP can be solved in $O(n \log n)$ time using a simple counting algorithm. First, sort the set of all angle range endpoints $\{\phi_k^s, \phi_k^e | 1 \leq k \leq n\}$. If $\phi = 0$ is a proper solution (i.e., if no interval has the form $[0, \phi_k^e))$ this is the answer. Otherwise, we process the endpoints, one at a time. Whenever we process a starting (resp. ending) endpoint, we increment (resp. decrement) a counter. Note that because of the way we have created our intervals, we never encounter an ending endpoint before the corresponding starting endpoint. Now, the first time after processing some $\phi_k^e < \frac{\pi}{2}$ that the counter became 0 we remember that angle as ϕ_1 and the last time that the counter became 0 after processing some $\frac{\pi}{2} < \phi_k^e < \pi$, we remember the solution as ϕ_2. Afterwards, we pick the smaller of ϕ_1 and $\pi - \phi_2$ as the overall best solution. □

5.2 An $O(kn \log n)$ Restricted Orientation Algorithm

We now turn to the problem of finding the furthest separating line with a prespecified orientation, which we assume to be horizontal without any loss of generality. We begin by looking at where in the vertical range of the point set S the best separating line should be placed. Just as in the general case, the goal of maximizing the width of the "corridor" dictates that the line must be centered with respect to the closest points on each side. This means that only two input points $(x_i, y_i), (x_j, y_j)$ with weights w_i, w_j (the witnesses of the previous section) are responsible for determining the location $y = y_s$ of the best separating line s, which must satisfy

$$(y_i - y_s)w_i = (y_s - y_j)w_j \Rightarrow y_s = \frac{y_i w_i + y_j w_j}{w_i + w_j}.$$

Therefore, the solution to the problem is the intersection of the two lines $d^w = -w_i y_s + y_i w_i$ and $d^w = w_j y_s - y_j w_j$, where d^w stands for the weighted distance to the horizontal line at y_s. This leads us to consider a "distance-location" space composed of such lines (refer to Figure 5.2), each point in S giving rise to one upward and one downward sloping line with the absolute values of the slopes equal to the weight of the point.

Let us suppose that all points in S are located in the first quadrant, i.e., $x_i, y_i > 0, \forall 1 \leq i \leq n$ (we can, in linear time, find the right translation to move the origin of the coordinate system). We map each point p_i with the weight w_i to the pair of lines in the "distance-location" plane $\ell_{i0} = w_i y_i - w_i y$ and $\ell_{i1} = -w_i y_i + w_i y$ and restrict the domain to the first quadrant. Thus, for each point we have a linear transformation ℓ_i of the absolute value metric function on the nonnegative domain. Each such wedge ℓ_i computes the distance from p_i to the separating line as we hypothetically sweep it upward from $y = 0$, and consists of a finite down-sloping segment (recording the distance for $y < y_i$) and an infinite up-sloping ray (for $y > y_i$). In this arrangement of $2n$ lines we are interested in the greatest achievable minimum distance, i.e., in the point with the highest vertical coordinate on the lower envelope of the wedge lines in the distance-location plane. This is illustrated in Figure 5.2 on the right. The lower boundary keeps track of the point or points (at a vertex) that are closest to the separating line at any position y_s. Since our goal is to find y_s that maximizes the distance from the line to the closest point, the optimal location is given by

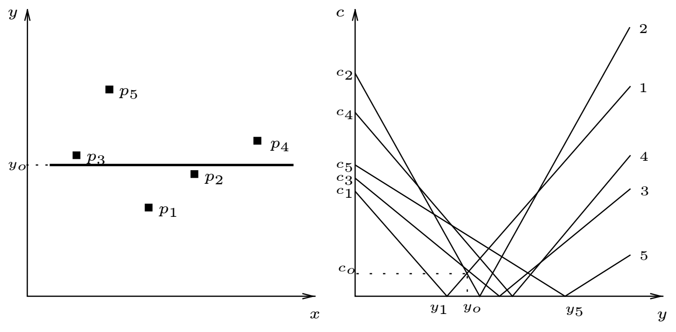

Figure 5.2: (a) A set of points $p_i = (x_i, y_i)$, sorted by y-coordinate, having weights w_i such that $w_2 > w_4 > w_1 > w_3 > w_5$, and the best fit segment. (b) The corresponding lines in the distance-location plane with the slopes w_i and the vertical axis intercepts c_i. The highest point on the lower boundary (envelope) gives the y-coordinate of the furthest separating line.

the highest vertex of the lower boundary. Note that the optimal vertex cannot be found by solving a linear program. Next, we need the following result.

Claim 5.4. The lower envelope of a set of n wedges in the distance-location plane has complexity $O(n)$.

Proof. If we relabel the wedges and consider them sorted by slope, $w_1, \ldots, w_n$, then the wedge with slope w_i, when added to the set of wedges with slopes $w_1, \ldots, w_{i-1}$, only modifies the envelope locally, between the two immediate neighbors of its vertex on the x-axis. Thus, its branches contribute two new edges and three new vertices to the envelope. When the newly added wedge intersects a single edge of the old envelope, it may split the old edge in two, creating one more edge. Therefore, there can be at most $3n - 1$ edges (the first wedge adds only two) and $3n - 2$ vertices on the lower envelope of n wedges. This is somewhat more tight than what could be obtained from the theory of Davenport-Schinzel sequences.

Another way to think of this problem is to classify the vertices on the lower boundary according to the slopes of the lines responsible for them. There are two basic types of vertices that we distinguish. The first type is when the slopes of the two lines are either both positive or both negative. There can be no more than $n-1$ vertices in each of these two cases, since for example in the case of both lines having positive slopes, the line with the bigger slope will no longer contribute any vertices to the boundary to the right of the intersection. How many vertices can there be on the lower boundary resulting from a line of positive and a line of negative slope? For each such vertex, there must either be an intersection of the first type, since each line enters the "inside" of the other's wedge and must either exit it in order to intersect again or that line is "consumed" and reaches the end of the domain while inside that wedge. Hence, there can be no more than $2n$ such vertices. This gives a more loose bound of $5n-2$ vertices (including n wedge vertices on the horizontal axis). $\square$

Finding the optimal vertex requires $\Omega(n \log n)$ time. This is true because any solution to the problem of finding the best horizontal separating line takes $\Omega(n \log n)$ time, as can be easily seen by reduction from Max-Gap (see [2]). The argument, of course, carries over to arbitrary fixed orientations of the separating line.

In order to find the highest vertex on the lower envelope of the wedges we employ a simple divide-and-conquer "skyline" merge algorithm. Alternately, we can build the lower boundary incrementally by adding wedges, one by one, in the order of lightest to steepest slope. Then, each new wedge would intersect the lower boundary in just two places which can be found by binary search. The wedges already added and with apices to the left of the apex of the new wedge have their left segments entirely below the left segment of the new wedge (similarly, for the right rays of the wedges to the right of the new wedge). The wedges already added and with apices to the right of the apex of the new wedge have their right rays dominated by the right ray of the new wedge. Therefore, after a binary search on the apices of the wedges we just have to intersect the new wedge with the region of the lower boundary between its two immediate neighbors resulting in exactly two new intersections (regardless of whether the downward line of the new wedge intersects a downward or an upward line of some old wedge, etc.). The edges that need to be removed from the boundary form a contiguous set and number $O(n)$ altogether over the entire execution of the algorithm. Both the divide-and-conquer and the incremental algorithms are optimal as they take $O(n \log n)$ time. Consequently, given k possible orientations $\alpha_1, \ldots, \alpha_k$ of the separating line, we can find the best separator in $O(kn \log n)$ time.

Chapter 6

Maximin Cone Facility Location

6.1 Background

We take up the problem of finding the widest (obnoxious) empty cone (wedge) through a set S of points. Let S be a set of points and *CH*(S) its convex hull. We shall describe an algorithm to compute the widest empty wedge anchored anywhere on the boundary of or outside *CH*(S) and analyze its running time. The wedge (or cone) $\mathcal{W}$ is a convex open polygon, formed by the intersection of two-halfplanes. Therefore, the boundary of the wedge is formed by two rays that meet in a point q, the apex of $\mathcal{W}$. In our problem, it always makes sense to widen the wedge until each of its boundary rays comes to rest on one or more input points. Hence, we shall refer to the boundary rays of $\mathcal{W}$ by the identities of the "supporting" points, the "supports" of the wedge. Further, since the apex of $\mathcal{W}$, for the reasons explained below, is to be fixed on the boundary of *CH*(S), we shall distinguish between the two supports of $\mathcal{W}$ based on the order in which the two supported edges ap-

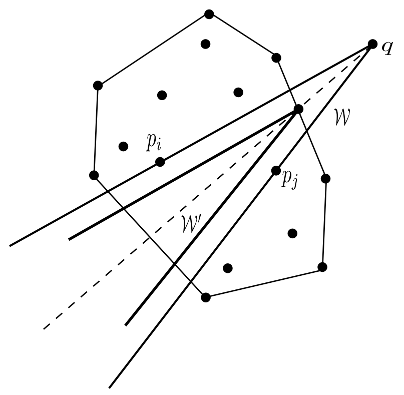

Figure 6.1: An illustration of the widest empty cone problem.

pear when the wedge is encountered in the counterclockwise traversal of the boundary of *CH*(S), ∂*CH*(S).

6.2 Finding the Widest Wedge with an Apex on the Boundary or Outside of *CH*(S)

In this chapter we describe the algorithm to solve the general problem with the location for the apex unrestricted. Before we proceed, observe that it suffices to consider cones with apices on ∂*CH*(S). If one assumes that the best wedge $\mathcal{W}$ rests on an apex outside *CH*(S), then a quick argument shows that there is a wedge $\mathcal{W}'$ that has a superior width and an apex on ∂*CH*(S), namely rests on the point q where an edge e of ∂*CH*(S) intersects the bisector of $\mathcal{W}$. This is shown in Figure 6.1. Even though attempting to fix the supports of $\mathcal{W}'$ on the same pair of points that support $\mathcal{W}$ may not produce an empty wedge, any wedge with an apex at q that is maximally empty is wider than $\mathcal{W}$ (we can always find support points radially closest from q to the supports for $\mathcal{W}$). Alternatively, keeping the supports of $\mathcal{W}'$ parallel to the supports of $\mathcal{W}$ preserves both the angular width and the emptiness of the wedge. Finally, the case for the apex of $\mathcal{W}$ situated inside of $CH(S)$ does not merit consideration either, because it makes the problem ill-defined. $\mathcal{W}$ can be made arbitrarily wide by moving the apex closer and closer to an edge of $CH(S)$. Hence, we can restrict our attention exclusively to the wedges that have apices on $\partial CH(S)$.

First, we tackle the purely geometric problem of finding the widest wedge with supports on any two points p_i, p_j of S with the anchor q allowed to slide anywhere on a line ℓ. This has the flavor of a classic problem and indeed it can be solved through recourse to an argument by an ancient Greek mathematician Apollonius. We divide the analysis into two cases.

Case 1: The line ℓ and the line that passes through the segment $\overline{p_i p_j}$ are parallel. In this case we claim that the widest wedge is achieved at the intersection W_{ij} of the line ℓ and the perpendicular bisector of the segment $\overline{p_i p_j}$. To show that this point produces the widest wedge, let's consider the circle Γ that passes through p_i, p_j, W_{ij}. The line ℓ is tangent to Γ because the perpendicular bisector of $\overline{p_i p_j}$ passes through its center and is perpendicular to ℓ. Now consider any point q on ℓ different from W_{ij}. Let q' be the intersection of the segment $\overline{q p_j}$ and the circle Γ. Then we have that $\angle p_i q p_j + \angle q p_i q' = \angle p_i q' p_j = \angle p_i W_{ij} p_j$. The last equation and the fact that $\angle q p_i q' \geq 0$ imply that $\angle p_i W_{ij} p_j \geq \angle p_i q p_j$ for any q on ℓ.

Case 2: The line that passes through p_i and p_j intersects ℓ in the point m_{ij}. We claim that the points on ℓ such that the circle defined by p_i, p_j and one of these points is tangent to ℓ produce the widest possible wedges. The problem of finding these points is known as one of the special cases of the problem of Apollonius. The solution to this problem (refer to

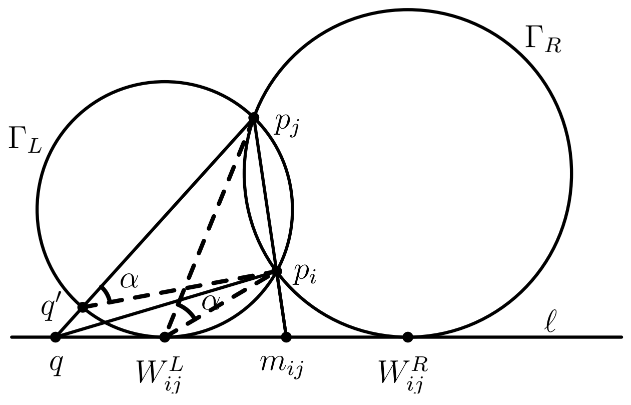

Figure 6.2: An illustration of the problem of Apollonius.

Figure 6.2) is well known and shows that there exist two points that fulfill the conditions. One of the points will be lexicographically to the left of m_{ij} and we will refer to it as $W_{ij}^L(\ell)$, while the point lying to the right of m_{ij} we shall call $W_{ij}^R(\ell)$ (we shall often drop ℓ from these names for brevity when it is implied or irrelevant). Now we prove that for every q on ℓ that is to the left of m_{ij}, $W_{ij}^L(\ell)$ produces the widest wedge.

Consider q to the left of m_{ij}. Let Γ_L be the circle that passes through p_i, p_j and W_{ij}^L. Let q' be the intersection of the segment $\overline{qp_j}$ and the circle Γ_L. Then, we have that $\angle p_i q p_j + \angle q p_i q' = \angle p_i q' p_j = \angle p_i W_{ij}^L p_j$. Again the result is that $\angle p_i W_{ij}^L p_j \geq \angle p_i q p_j$ for any q on ℓ to the left of m_{ij}. The argument for W_{ij}^R and q being to the right of m_{ij} is similar. Therefore, we conclude that for points to the right of m_{ij}, W_{ij}^R produces the widest wedge.

To determine the position of points W_{ij}^L and W_{ij}^R we use the notion of the power of a point. Let's consider Γ_L as defined previously. The power of m_{ij} with respect to Γ_L can be calculated using points p_i, p_j or using the point of tangency W_{ij}^L. In this case the relation obtained is $m_{ij}p_i \cdot m_{ij}p_j = (m_{ij}W_{ij}^L)^2$ which results in $m_{ij}W_{ij}^L = \sqrt{m_{ij}p_i \cdot m_{ij}p_j}$. Since the coordinates of m_{ij} and the product $m_{ij}p_i \cdot m_{ij}p_j$ are easy to calculate, so is the distance between m_{ij} and W_{ij}^L. The distance from m_{ij} to W_{ij}^R has the same magnitude but opposite sign. Therefore the value $\sqrt{m_{ij}p_i \cdot m_{ij}p_j}$ needs to be computed only once.

Now, let's define the function $W_{ij}(q) = \angle p_i q p_j, \forall q \in \ell$. We will consider the point q as it is moved from left to right. For the interval $q \in [-\infty, W_{ij}^L]$ the function is increasing. Consider the points q_1, q_2 both of them to the left of W_{ij}^L with q_2 being the closer to it. Let Γ_{L1} be the circle that passes through p_i, p_j and q_2. Γ_{L1} intersects the line ℓ in two points: q_2 and another point to the right of W_{ij}^L. So, the point q_1 is outside of Γ_{L1}. Repeating this geometrical argument, we conclude that $W_{ij}(q_1) = \angle p_i q_1 p_j \leq \angle p_i q_2 p_j = W_{ij}(q_2)$. Therefore, the function increases as we approach W_{ij}^L from the left. Similar analysis show that $W_{ij}(q)$ decreases on $[W_{ij}^L, m_{ij}] \cup [W_{ij}^R, \infty]$ and increases on $[-\infty, W_{ij}^L] \cup [m_{ij}, W_{ij}^R]$, with W_{ij}^R and W_{ij}^L being the local maxima and m_{ij} being the absolute minimum.

Now, that we have seen how the problem of finding the widest wedge can be solved for a single pair of points and a single line, we are ready to put forth a description of an algorithm for S and restrict the anchors to lie anywhere on $\partial \mathit{CH}(S)$. First, we can see that since $\partial \mathit{CH}(S)$ can have $O(n)$ edges in the worst case, the number of points $W_{ij}^L(\ell), W_{ij}^R(\ell)$ can be $O(n^3)$ (or more precisely, $O(kn^2)$, if $\partial \mathit{CH}(S)$ has k edges). In this setting, we now define $W_{ij}^L(\ell)$ as the local maximum of $W_{ij}(q), q \in \partial CH(S)$ that is encountered first along the supporting line ℓ of an edge e of $\partial \mathit{CH}(S)$ in the counterclockwise traversal of $\partial CH(S)$ (thus, $W_{ij}^R(\ell)$ is the relative maximum that occurs later in such a traversal, and if any of the two points W_{ij}^L, W_{ij}^R are outside of e, it is not considered, yet the naming is still consistent since the direction of traversal along ℓ is well defined).

We shall now see that we do not need to analyze more than $O(n^2)$ of these points. We are interested only in those wedges that are empty and thus it makes sense to start with a valid configuration of $O(n)$ empty wedges that can be produced with a radial sort on S at some vertex v_1 of $\partial \mathit{CH}(S)$. For p_i, p_j non-parallel to ℓ, we then compute the points $W_{ij}^M(\ell) = W_{ij}^L(\ell)$ (or $W_{ij}^M(\ell) = W_{ij}^R(\ell)$ when $W_{ij}^L(\ell)$ is outside of e, or null if both values are outside) for all pairs of points p_i, p_j adjacent in the radial sort, which provide supports to this initial set of wedges, for the line ℓ through the edge $e = (v_1, v_2)$. Hence, we retain only those $W_{ij}^M(\ell)$'s that lie on the edge e itself, giving initial priority to W_{ij}^L since it is the first to be encountered. For p_i, p_j that are parallel to ℓ, we simply set $W_{ij}^M(\ell)$ to be the point where the perpendicular bisector of $p_i p_j$ intersects e or if it falls on either side of e, the vertex of e closest to that point. We further compute the points $m_{ij}(\ell)$ on e where the changes to the radial order for currently adjacent pairs p_i, p_j take place (for all p_i, p_j such that $p_i p_j$ is not parallel to ℓ). Below we shall describe where this information will be stored for every adjacent pair p_i, p_j.

We observe that the radial order changes when the moving apex q becomes collinear with $K \geq 2$ input points, thereby changing the identities of the supports of the neighboring two wedges (the wedge supported by p_i, p_j does not really change except that the order of the supports is flipped). We make another observation that for each pair p_i, p_j, their relative order in the radial sort as computed from points q along $\partial \mathit{CH}(S)$ can change exactly twice - when q becomes collinear with p_i, p_j at $q = m_{ij}(\ell)$, which happens for exactly two edges e_r, e_s of $\partial \mathit{CH}(S)$, with their supporting lines ℓ_r, ℓ_s (q may become collinear with $K > 2$ points simultaneously with the outcome being the reversal of the order in which the wedges appeared prior to this event, which can be viewed as a sequence of pairwise wedge order reversals). Hence, in total, there are no more than $O(n^2)$ such points $m_{ij}(\ell_r), m_{ij}(\ell_s)$. We, therefore, for each adjacent pair of points in the radial order compute when, if at all, they are "scheduled" to flip along e and put that flip priority (we can represent $m_{ij}(\ell)$ as a single parameter value along (v_1, v_2) if it falls inside, ∞ for those wedges not scheduled to flip along e) in a heap organized in counterclockwise order from v_1. The locations and values for W_{ij}^M can be stored in these nodes as additional data.

The crucial observation here is that the only points where this structure needs to be updated are the points where the radial order changes and the only information that needs to be updated at those points is the information for the flipped wedge and the two neighboring wedges. Since their identities have changed, we need to recompute their W^M's, as well as determine $m_{ij}(\ell)$ for these newly adjacent pairs (or put ∞ as their flip priority if it is to the left of q) and re-heapify on each of these two nodes (we also need to set the flip priority of the wedge causing the change to ∞ and compute $W_{ij}^M = W_{ij}^R$, if W_{ij}^R is inside of e). Finally, the way we keep track of the maximum width $\mathcal{W}$ is by investigating W_{ij}^M (which we precomputed and stored with each wedge as it comes into existence) only those wedges that have been involved in an update at a particular $m_{i'j'}(\ell)$ where the radial order has changed. Since the wedge width function $\mathcal{W}_{ij}(q)$ is semi-monotone between any pair of points on the same side of $m_{ij}(\ell)$ (it contains the single extremum W^L or W^R on such an interval), the wedges that did not participate in the update need not be accounted for at the point of update. Their widths are getting either uniformly wider or narrower anywhere in the vicinity of that point, or they could have achieved a single relative maximum, which has been recorded for these wedges and will be investigated either at some point of update when such an edge is affected or at v_2 if it survives till the end of e. Hence, at the point of update we need to consider W_{ij}^M's for the three wedges in question, if these have been recorded for them, or evaluate them at the point of update itself and maintain the best width so far and the identity of that wedge. We then proceed to move along the edge until v_2, at which point we recompute $W_{ij}^M(\ell')$ for each wedge with respect to the new line ℓ' through the edge (v_2, v_3) and produce a new structure. The cost of this algorithm is $O(kn \log n)$ for the initial constructions at each of the k vertices of $\partial CH(S)$ and $O(n^2 \log n)$ amortized time for the $O(n^2)$ points of update that occur along the entire boundary.

This algorithm always finds the optimal wedge. This is a straightforward consequence of the fact that every wedge that can exist with an apex on ∂*CH*(S) (and we have already shown that only these wedges need be examined) is, in fact, processed when it comes into existence and no wedge is ever destroyed without the algorithm performing the correct procedure for determining the absolute maximum of $W_{ij}(q)$ by investigating its local maximum W_{ij}^M and evaluting $W_{ij}(q)$ at the endpoints of the interval of existence (which are exactly the points of update to the structure). Hence, the optimal wedge cannot evade detection.

6.3 Finding the Widest Wedge with an Apex at an Extreme Point of S

We now consider the case where the apex of the widest empty cone is constrained to coincide with a vertex of *CH*(S). For clarity, we assume that S is in general position (no two input points share the same x coordinate). We solve this case through recourse to duality, where a point (a, b) becomes the line $ax - b$. Of particular importance is the fact that duality preserves incidence and topological relationships: point P is below line ℓ in primal space iff the dual of ℓ, the point $D(\ell)$, is below the line $D(P)$ in dual space.

We therefore convert S to its dual representation and look at the resulting arrangement of lines $A(S)$. The key to the solution is to describe what an arbitrary empty cone in the primal space that is anchored at a vertex v of *CH*(S) and with supports at p_i and p_j looks like in the dual. The apex of the cone, v, as well as p_i and p_j, become lines $D(v)$, $D(p_i)$, and $D(p_j)$, respectively. The two boundary rays (or really lines through them) become two points on $D(v)$, call them $D(\ell_i)$ and $D(\ell_j)$. Hence, since the cone contains all rays through v with slopes between those of the rays through p_i and p_j, it becomes a segment on $D(v)$ in the dual, except in the case when it contains a vertical ray (in that case, actually, it's the complement of the interval of slopes between the supports). In fact, since we assumed general position for S, the dual of every point of S intersects $D(v)$. Therefore, what does it mean for the cone at v to be empty? If there is a point p inside of the cone apexed at v, then in the case of that cone not containing a vertical ray (we shall look at that case later), p ends up being above the ray through p_i and below the one through p_j, or vice versa. That means that in the dual space the points $D(\ell_i)$ and $D(\ell_j)$ lie on opposite sides of the line $D(p)$. Therefore, $D(p)$ must intersect the segment connecting $D(\ell_i)$ with $D(\ell_j)$, which we know lies on $D(v)$. Hence, an empty cone in primal space becomes an edge of $A(S)$ in the dual.

For the remaining case, when the empty cone contains a vertical ray, a point inside of the cone is either below or above *both* rays. Furthermore, because the apex is on *CH*(S) there can only be either points that are below both rays, or above both rays, but never points of each kind simultaneously. Otherwise, it is a trivial contradiction to the fact that v is on the boundary of a convex shape (in fact, angle at v must be less than $180°$) and p_i and p_j are on the boundary of or inside that shape, since it would imply that there is an internal angle at v greater than $180°$. Therefore, if such a cone is empty, then all other points of S not lying on the rays of that cone must be above $D(p_i)$ and below $D(p_j)$ or vice versa. This means that the supports of this cone correspond in the dual to the lines that produce the intersections on $D(v)$ that are furthest apart, i.e. the "extreme pair" of points on $D(v)$.

In order to compute all such empty cones, we build $A(S)$ in $O(n^2)$ time with a topological sweep (in order to use $O(n)$ memory). For every edge encountered, we can go back to the primal space and in constant time compare its angular width with the best found so far. The time to examine the candidate cones is $O(n^2)$, i.e., proportional to the size of the arrangement. Finding the extreme pairs also takes at most $O(n^2)$, since we can in linear time find the extreme pair for each candidate apex. As a concluding remark, we note here that this approach actually allows us to compute the widest empty cone with an apex in S, which may or may not be extreme.

Chapter 7

Conclusion

The work presented in this thesis can be continued with many interesting and as yet unaddressed variants. The lower bounds for the problems investigated in Sections 5 and 6 have not yet been resolved, thus leaving open the possibility of finding more efficient algorithms. The “obnoxious” wedge problem can be extended to three dimensions and to handle weights. Finally, all of the problems mentioned in the previous sections can be treated under the sum definition of error (i.e., minimizing or maximing the sum total of error contributions from all points). Many of the problems the author has worked on remain open in this context (e.g., the case of a fixed and known number k of segments, for $k > 3$, of the approximating step function curve).

The further appeal of these subjects stems from their applicability in spatial and temporal databases, GIS, and computer graphics. In particular, the author is interested in pursuing further the following subjects: map simplification, mesh refinement, and surface reconstruction. Pattern recognition has many applications when it comes to analyzing maps that are at the elementary level just collections of segments or polygonal subdivisions of the plane. Similar problems can be explored in three-dimensional space, where potential applications deal with the analysis and generation and simplification/refinement of polygonal meshes used to represent objects to be rendered in the graphics pipeline. A recent trend in the field is to explore various techniques for implementing spatial algorithms using parallel processing architectures, such as CUDA. This area is extremely relevant in today's computing and the author would like to investigate the usefulness of such methods for the topics presented in this work.

Bibliography

[1] P. K. Agarwal. Near-linear time approximation algorithms for curve simplification. In *Proceedings of the 10th European Symposium on Algorithms*, pages 29–41, 2002.

[2] E. Arkin, F. Hurtado, J.S.B. Mitchell, C. Seara, and S. Skiena. Some lower bounds on geometric separability problems. *International Journal of Computational Geometry and Applications*, 16(1):1–26, 2006.

[3] B. Aronov, T. Asano, N. Katoh, K. Mehlhorn, and T. Tokuyama. Polyline fitting of planar points under min-sum criteria. *International Journal of Computational Geometry and Applications*, 16(1):97–116, 2006.

[4] G. Barequet, D. Z. Chen, O. Daescu, M. T. Goodrich, and J. Snoeyink. Efficiently approximating polygonal paths in three and higher dimensions. *Algorithmica*, 33(2):150–167, 2002.

[5] B. Ben-Moshe, M. Katz, and M. Segal. Obnoxious facility location: complete service with minimal harm. *International Journal of Computational Geometry and Applications*, 10(6):581–592, 1999.

[6] S. Bespamyatnikh, K. Kedem, and M. Segal. Optimal facility location under various distance functions. In *Proceedings of the 6th International Workshop on Algorithms and Data Structures (WADS'99)*, pages 318–329, 1999.

[7] P. Bose and Q. Wang. Facility location constrained to a polygonal domain. In *Proceedings of the 5th Latin American Symposium on Theoretical Informatics. Lecture Notes in Computer Science*, volume 2286, pages 153–164, 2002.

[8] G. Brodal and R. Jacob. Dynamic planar convex hull. In *Proceedings of the 43rd IEEE Symposium on Foundations of Computer Science*, pages 617–626, 2002.

[9] P. Cappanera. A survey on obnoxious facility location problems. *Technical Report. University of Pisa*, 1999.

[10] P. Cappanera, G. Gallo, and F. Maffioli. Discrete facility location and routing of obnoxious activities. *Discrete Applied Mathematics*, 133(1-3):3–28, 2003.

[11] S. Chan and F. Chin. Approximation of polygonal curves with minimum number of line segments or minimum error. *International Journal of Computational Geometry and Applications*, 6:59–77, 1996.

[12] S.-W. Cheng. Widest empty l-shaped corridor. *Information Processing Letters*, 58(6):277–283, 1996.

[13] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, Cambridge, MA, 2001.

[14] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry - Algorithms and Applications*. Springer, 2000.

[15] J.M. Díaz-Bánez, F. Gomez, and F. Hurtado. Approximation of point sets by 1-corner polygonal chains. *INFORMS Journal on Computing*, 12:317–323, 2000.

[16] J.M. Díaz-Bánez and F. Hurtado. Computing obnoxious 1-corner polygonal chains. *Computers and Operations Research*, 33(4):1117–1128, 2006.

[17] J.M. Díaz-Bánez, F. Hurtado, H. Meijer, D. Rappaport, and J. Sellares. The largest empty annulus problem. In *Proceedings of the International Conference on Computational Science, Part III. Lecture Notes in Computer Science*, volume 2331, pages 46–54, 2002.

[18] J.M. Díaz-Bánez, M.A. Lopez, M. Mora, C. Seara, and I. Ventura. Fitting a two-joint orthogonal chain to a point set. *Computational Geometry*, 44(3):135–147, 2011.

[19] J.M. Díaz-Bánez, M.A. Lopez, and J. Sellares. Locating an obnoxious plane. *European Journal of Operational Research*, 173(2):556–564, 2006.

[20] J.M. Díaz-Bánez, M.A. Lopez, and J. Sellares. On finding a widest empty 1-corner corridor. *Information Processing Letters*, 98(5):199–205, 2006.

[21] J.M. Díaz-Bánez and J.A. Mesa. Fitting rectilinear polygonal curves to a set of points in the plane. *European Journal of Operations Research*, 130:214–222, 2001.

[22] J.M. Díaz-Bánez, P.A. Ramos, and P. Sabariego. The maximin line problem with regional demand. *European Journal of Operations Research*, 181:20–29, 2007.

[23] Z. Drezner and G.O. Wesolowsky. Location of an obnoxious route. *Journal of the Operational Research Society*, 40(11):1011–1018, 1989.

[24] H. Edelsbrunner and L.J. Guibas. Topologically sweeping an arrangement. In *Proceedings of the 18th annual ACM symposium on theory of computing*, pages 389–403, 1986.

[25] D. Eu and G. T. Toussaint. On approximating polygonal curves in two and three dimensions. *CVGIP: Graphical Models and Image Processing*, 56(3):231–246, 1994.

[26] H. Fournier and A. Vigneron. Fitting a step function to a point set. *Algorithmica*, pages 1–15, 2009.

[27] G.N. Frederickson and D.B. Johnson. Generalized selection and ranking: Sorted matrices. *SIAM Journal on Computing*, 13(1):14–30, 1984.

[28] M.T. Goodrich. Efficient piecewise-linear function approximation using the uniform metric. *Discrete and Computational Geometry*, 14:445–462, 1995.

[29] S. L. Hakimi and E. F. Schmeichel. Fitting polygonal functions to a set of points in the plane. *CVGIP: Graphical Models and Image Processing*, 53(2):132–136, 1991.

[30] J. Hershberger and S. Suri. Off-line maintenance of planar configurations. *Journal of Algorithms*, 21:453–475, 1996.

[31] M. Houle, H. Imai, K. Imai, J.-M. Robert, and P. Yamamoto. Orthogonal weighted linear l_1 and l_∞ approximation and applications. *Discrete Applied Mathematics*, 43(3):217–232, 1993.

[32] M. Houle and A. Maciel. Finding the widest empty corridor through a set of points. *Snapshots of Computational and Discrete Geometry, G. T. Toussaint, ed.*, pages 201–213, 1988.

[33] F. Hurtado, V. Sacristan, and G. Toussaint. Some constrained minimax and maximin location problems. *Studies in Locational Analysis: Special Issue on Computational Geometry in Locational Analysis*, pages 17–35, 2000.

[34] H. Imai and M. Iri. Computational-geometric methods for polygonal approximations of a curve. *Computer Vision, Graphics and Image Processing*, 36(1):31–41, 1986.

[35] H. Imai and M. Iri. An optimal algorithm for approximating a piecewise linear function. *Journal of Information Processing*, 9(3):159–162, 1986.

[36] H. Imai and M. Iri. Polygonal approximations of a curve - formulations and algorithms. *Computational Morphology, G. T. Toussaint, ed.*, pages 71–86, 1988.

[37] R. Janardan and F. Preparata. Widest corridor problems. *Nordic Journal of Computing*, pages 231–245, 1994.

[38] M. Katz, K. Kedem, and M. Segal. Improved algorithms for placing undesirable facilities. *Computers and Operations Research*, 29(13):1859–1872, 2002.

[39] N. Megiddo. Linear programming in linear time when the dimension is fixed. *Journal of ACM*, 31(1):114–127, 1984.

[40] D. P. Mehta and ed. S. Sahni. *Handbook of data structures and applications*. Chapman and Hall/CRC, 2004.

[41] A. Melkman and J. O'Rourke. On polygonal chain approximation. *Computational Morphology, G. T. Toussaint, ed.*, pages 87–95, 1988.

[42] J. O'Rourke. *Computational Geometry in C*. Cambridge University Press, 2000.

[43] M.H. Overmars and J. van Leeuwen. Maintenance of configurations in the plane. *Journal of Computer and System Sciences*, 23(2):166–204, 1981.

[44] J.S. Salowe. Chapter 43: Parametric search. *Handbook of data structures and applications, D. P. Mehta and S. Sahni, ed.*, pages 683–695, 2004.

[45] R. Seidel. Linear programming and convex hulls made easy. In *SCG '90: Proceedings of the Sixth Annual Symposium on Computational Geometry*, pages 211–215, 1990.

[46] C.S. Shin, S.Y. Shin, and K.-Y. Chwa. The widest k-dense corridor problems. *Information Processing Letters*, 68(1):25–31, 1998.

[47] K. R. Varadarajan. Approximating monotone polygonal curves using the uniform metric. In *SCG '94: Proceedings of the 12th annual symposium on Computational geometry*, pages 311–318, 1996.

[48] D. P. Wang, N. F. Huang, H. S. Chao, and R. C. T. Lee. Plane sweep algorithms for the polygonal approximation problems with applications. In *ISAAC '93: Proceedings of the 4th International Symposium on Algorithms and Computation*, pages 515–522. Springer-Verlag, 1993.

[49] D.P. Wang. A new algorithm for fitting a rectilinear x-monotone curve to a set of points in the plane. *Pattern Recognition Letters*, 23:329–334, 2002.

[50] G.O. Wesolowsky. Location of the median line for weighted points. *Environment and Planning A*, 7(2):163–170, 1975.

[51] P. Yamamoto, K. Kato, K. Imai, and H. Imai. Algorithms for vertical and orthogonal l_1 linear approximation of points. In *Proceedings of the 4th Annual Symposium on Computational Geometry*, pages 352–361, 1988.