INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

Bell & Howell Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600

Université d'Ottawa • University of Ottawa

Polygon Visibility Decompositions with Applications

Ralph Patrick Boland

THESIS SUBMITTED TO THE
SCHOOL OF GRADUATE STUDIES AND RESEARCH
IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY IN COMPUTER SCIENCE

UNDER THE AUSPICES OF THE
OTTAWA-CARLETON INSTITUTE FOR COMPUTER SCIENCE

UNIVERSITY OF OTTAWA
OTTAWA, ONTARIO, CANADA
FEBUARY, 2002

Abstract

Many problems in Computational Geometry involve a simple polygon P and a family of geometric objects, say ς, contained in P. For example, if ς is the family of chords of P then we may want to find the longest chord in P. Alternatively, given a chord of P, we may wish to determine the areas of the two subpolygons of P determined by the chord. Let Π be a polygonal decomposition of a polygon P. We call Π a visibility decomposition with respect to ς if, for any object $g \in \varsigma$, we can cover g with $o(|P|)$ of the subpolygons of Π. We investigate the application of visibility decompositions of polygons to solving problems of the forms described.

Any visibility decomposition Π of a polygon P that we construct will have the property that, for some class of polygons $\wp$ where the polygons in $\wp$ have useful properties, $\Pi \subseteq \wp$. Furthermore, the properties of $\wp$ will be key to solving any problems we solve on P using Π.

Some of the visibility decomposition classes we investigate are already known in the literature, for example weakly edge visible polygon decompositions. We make improvements relating to these decomposition classes and in some cases we also find new applications for them.

We also develop several new polygon visibility decomposition classes. We then use these decomposition classes to solve a number of problems on polygons including the circular ray shooting problem and the largest axis-aligned rectangle problem.

It is noteworthy that the solutions to problems that we provide are usually more efficient and always simpler than alternative solutions.

Acknowledgements

I would like to thank the many people who helped in completing this thesis of which only the most prominant are mentioned.

My supervisor, Jorge Urrutia, for having faith in me and for difference all this mathematical and writing skills made to the quality of this thesis and especially for all the patience required of him to do so.

Felipe Contreras, for keeping the computer hardware and software that I needed working both in Ottawa and Mexico City and for being the friend that he is.

Margaret Urrutia, Harvinder Singh, and Norman Taylor for proof reading my thesis. For this I am most grateful.

Nicola Bobic, for many things but most of all for recommending that I write my thesis in Lyx. I also thank the people who develop and maintain lyx for the wonderful job they are doing.

Laura Pastrana, Gina Herrera, and Francisco Herrera for their help and friendship during my stay in Mexico City.

Finally, I must thank Brenda Butler, for things too numberous to mention, but most of all for her love, patience, and encouragement. She kept me sane and alive during that final difficult year.

Contents

List of Figures viii

Chapter 1. Introduction 1

1.1. Why Study Visibility Decompositions? 4

1.2. Results Based on Visibility Decompositions 7

1.3. Other Results 8

1.4. Post Thesis Research Plans 9

Chapter 2. Polygon Visibility Decompositions 10

2.1. Introduction 10

2.2. Definitions 15

2.2.1. Visibility Decomposition Combinations 20

2.3. Previously Known Visibility Decompositions 23

2.4. Other Previous Results 26

2.5. Visibility Decompositions From Our Research 27

2.5.1. The 2*-Wind Subpolygon Partition Class 28

2.5.2. The Monotone Subpolygon Partition Class 30

2.5.2.1. A Problem Solving Strategy Involving Monotone Polygons 31

2.5.3. The SAM Polygon Rectangle Visibility Cover Class 31

Chapter 3. 2-Wind Subpolygon Partitions 34

3.1. Introduction 34

3.2. k^*-Wind Chains and Polygons 34

3.2.1. α-Cusps 38
3.2.2. k^*-Wind Chains 39
3.2.3. k^*-Wind Polygons 40
3.2.4. Y-Monotone Subpolygon Partitions of Polygons 45
3.3. Proper 2^*-Wind Chains and Polygons 50
3.4. Decomposing Polygons into Proper $\pm x$-2^*-Wind Subpolygons 55
3.5. Plane Sweep Algorithms 61
3.6. Applications 63

Chapter 4. Monotone Subpolygon Partition Classes 67
4.1. Introduction 67
4.2. A Monotone Subpolygon Visibility Partition Class 70
4.3. A Size Sensitive Monotone Visibility Partition Class 85

Chapter 5. Circular Ray Shooting 90
5.1. Introduction 90
5.2. A Reduction of the Circular Ray Shooting Problem 93
5.3. Intersection of an Arc with a y-Monotone Polygon 95
5.3.1. Detecting an Intersection 95
5.3.2. Circular Ray Shooting in Monotone Polygons 101
5.4. Constructing $\mathbf{V}_{\text{far}}(Q)$ in $O(m \log m)$ Time 105
5.5. Circular Ray Shooting in a Simple Polygon 108
5.5.1. Some Practical Considerations 109

Chapter 6. Rectangle Visibility Decompositions 113
6.1. Introduction 113
6.2. Histogram Subpolygon Rectilinear Visibility Partitions 116
6.3. Histogram Family Subpolygon Visibility Covers 121

6.4. SAM Subpolygon Rectangle Visibility Covers 122
6.4.1. The SAMPc of a Histogram Family Polygon 122
6.4.2. The SAMVC1 of a Rectilinear Polygon 124
6.5. The Largest Axis-Aligned Rectangle Problem 126
6.6. Generalization to Simple Polygons 126
6.6.1. The SAMPc of a Pseudo-Histogram Family Polygon 127
6.6.2. A SAM Subpolygon Rectangle Visibility Cover of a Pseudo-Histogram Polygon 128
6.6.3. A SAM Subpolygon Rectangle Visibility Cover of a Polygon 134
6.6.4. The Largest Axis-Aligned Rectangle Problem 135

Chapter 7. Conclusion and Future Research 136
7.1. Chapter Three 136
7.2. Chapter Four 140
7.3. Chapter Five 141
7.4. Chapter Six 146
7.5. Appendix A 147
7.6. Appendix B 150
7.7. Other Visibility Decompositions 151
7.8. Other applications 151

Appendix A. Finding a Range in Unsorted Arrays and Trees 152
A.1. Introduction 152
A.2. Range Finding in Arrays 156
A.2.1. Relaxing the Constraint on the Size of B 160
A.3. Searching Weighted Arrays 163
A.4. Range Finding in Unordered Trees 175
A.4.1. Handling Multi-way Trees 185

A.4.2. Partitioning Trees into Paths without Weights. 186

Appendix B. Polygon Area Problems 188

B.1. Introduction 188

B.2. Calculating the Area of a Polygon 189

B.3. Areas of Self-overlapping Polygons 194

Bibliography 196

List of Figures

1.0.1	A geodesic triangle (left) and a partition of a polygon into geodesic triangles (right)	2
1.0.2	Cutting a polygon by a chord	3
1.0.3	A 2*-wind polygon (left) and a partition of a polygon into 2*-wind subpolygons (some upside down)	4
2.1.1	A histogram cover of a rectilinear polygon	13
2.2.1	A tree partition of a polygon and the dual graph of the partition	17
2.2.2	The composition of visibility decomposition classes	21
3.2.1	Direction of turns on chains	35
3.2.2	A Proper k^*-wind chain	37
3.2.3	A proper 1-wind polygon	41
3.2.4	For any polar angle α there is a unique $\psi_\alpha(P)$	42
3.2.5	The neighbor objects of a cusp.	46
3.2.6	The proper $\pm x$-2*-wind subpolygon partitioning of P by $\mathfrak{L}_{\text{red}}(P)$	50
3.3.1	A proper x-2*-wind polygon	53
3.6.1	A maximum set of non-intersecting monotone column chains of P	65
4.2.1	Replacing a non column chord by a column chord	72
4.2.2	Efficient y-monotone partitions of a polygon	77

4.2.3	A partition Π of polygon P such that $\mathbf{C}_i(\Pi)$ is minimized	80
5.3.1	Chain Q_r^c of a y-monotone polygon Q	98
5.3.2	No vertex on Q_r^c is to the right of circle (ζ)	100
6.2.1	A histogram subpolygon partition using a large amount of memory	117
6.2.2	A pseudo-histogram	118
6.4.1	A SAM subpolygon cover of a rectilinear polygon	123
6.6.1	Covering a pseudo-histogram	130
6.6.2	A cover of a nose histogram	133
7.1.1	2^*-wind partition of a polygon with holes	138
7.3.1	An Asymmetric Voronoi diagram using $O(n^2)$ space	143
A.2.1	Labelling of an augmented complete binary tree	157
A.2.2	Range Finding	161
A.3.1	A weighted tree as constructed by Algorithm 16	166
A.3.2	An augmented weighted tree	169
A.4.1	Finding root subpaths on a tree	177
A.4.2	Two methods of partitioning a tree T into paths	179
B.1.1	A polygon P with a chord α and a pseudo chord γ ($P_{\overline{\gamma}}$ is simple)	190
B.2.1	An area problem on two polygons	192
B.2.2	A self overlapping polygon with two trapezoidizations	193

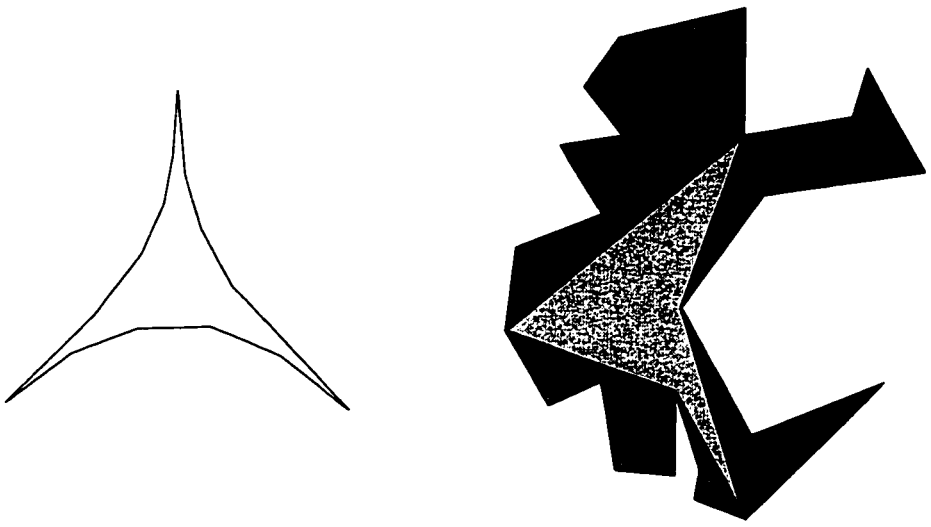
CHAPTER 1

Introduction

The input to many problems in Computational Geometry is a polygon. A common approach to tackling such problems is to first partition the input polygon into "simpler" pieces. The idea is to exploit the fact that the pieces are simpler than the original polygon to solve the problem on the pieces efficiently. The solutions for the pieces are then combined to form a solution for the original polygon. For example, polygons are commonly triangulated as a first step in solving many problems. Among the problems that can easily be solved on a triangulated polygon is that of computing its area; we simply sum the areas of the triangles in the triangulation.

However, for a variety of problems, some partitions are better than others. For example, consider that we have a polygon P and whenever we are given a ray whose origin is contained in P we want to quickly find the intersection point of the ray with the boundary of P nearest to the ray's origin. We call this problem the ray shooting problem. The ray shooting problem can be solved by first triangulating P and then following the set of triangles hit by a ray until it reaches the boundary of the polygon. Since in a bad case a ray could intersect most of the triangles in a triangulation of P, using a triangulation this way results in a relatively expensive ray shooting algorithm.

By using a different partition of P, however, far faster techniques have been developed to solve this problem. For example, for any geodesic triangle polygon (a polygon consistng of three concave chains; see Fig. 1.0.1) it is known that the ray shooting problem can be solved quickly. Furthermore, P can be partitioned into pieces in the shape of geodesic triangles such that a ray crosses a small number of them before



A polygon can be partitioned into geodesic triangles such that any chord of the polygon intersects at most a logarithmic number of them. Some geodesic triangles in the partition are degenerate.

FIGURE 1.0.1. A geodesic triangle (left) and a partition of a polygon into geodesic triangles (right)

hitting the boundary of P; see **[12]**. Thus, by partitioning P into geodesic triangles we can do ray shooting much more quickly than if a triangulation is used; see Fig. 1.0.1.

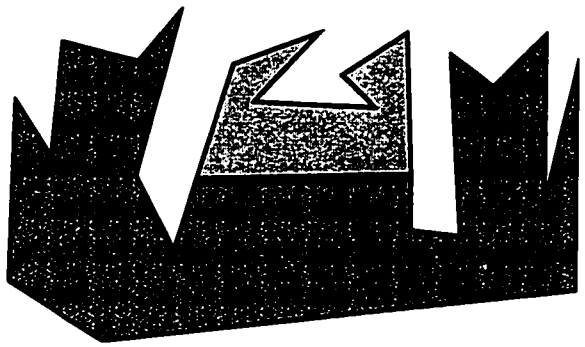
We study in this thesis partitions of polygons that will be called "visibility partitions". They have the following properties:

(1) The partition allows a problem to be solved by processing a small number of the pieces.

(2) Each piece of the partition has properties such that solving the problem on each piece is simpler than for general polygons.

For some problems we allow the use of overlapping pieces of the original polygon; to incorporate these cases as well we use the term "visibility decompositions".

In this thesis we will study a number of problems that we were able to solve by first developing visibility decompositions of polygons suitable for the problem at



A chord partitions a polygon into two pieces whose areas can be computed in constant time based on information about the polygon and the chord.

FIGURE 1.0.2. Cutting a polygon by a chord

hand. This thesis was motivated by the discovery of a new visibility partition which we designed to solve the following problem; see Fig 1.0.2.

PROBLEM 1.0.1. *Given a polygon P, calculate the areas of the pieces that result from cutting P along a chord.*

In [20] this problem was solved in logarithmic time using sophisticated means. We instead developed a new visibility partition with which we were able to solve this problem in constant time. We were able to solve this problem in constant time by introducing a visibility partition of polygons into what we call 2*-wind polygons. A 2*-wind polygon has a vertex or an edge from which every point of the polygon is reachable by a monotonically increasing curve contained in the polygon; see Fig. 1.0.3 (left).

As it later turned out we were able to obtain a constant time solution to Problem 1.0.1 that doesn't use decompositions at all; it turns out that the algorithm presented in [20], for convex polygons generalizes to non convex polygons as well! We present this solution in Appendix B.

By this time, however we had realized that our method of partitioning a polygon into 2*-wind polygon pieces could be used to solve a number of other problems. In particular we solved the following problem:

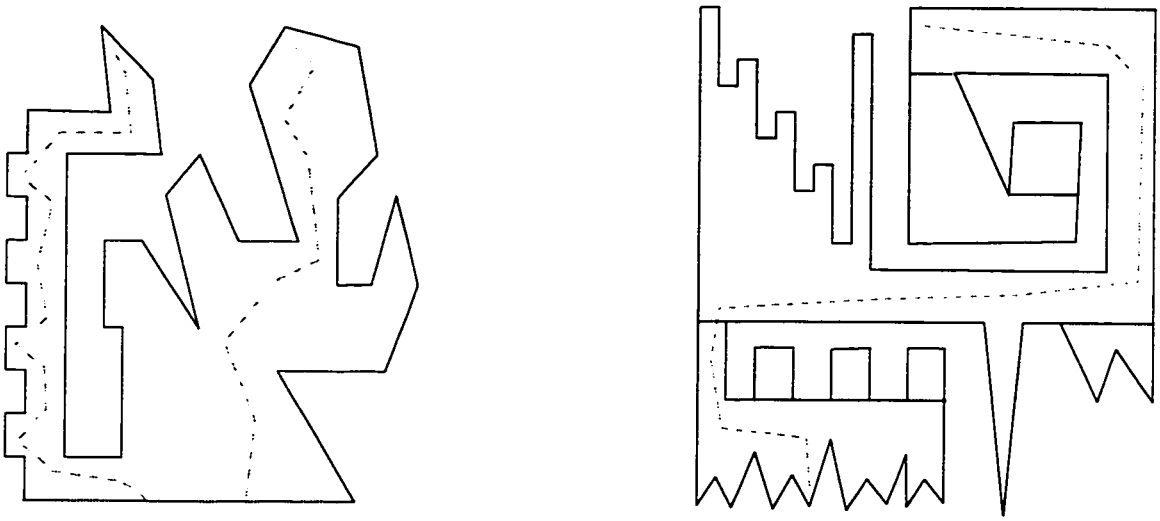


FIGURE 1.0.3. A 2*-wind polygon (left) and a partition of a polygon into 2*-wind subpolygons (some upside down)

PROBLEM 1.0.2. *Partition a polygon into the minimum number of y-monotone pieces efficiently.*

Our algorithm to solve this problem uses the property that any monotonically increasing curve contained in the polygon intersects at most two pieces of the partition; see Fig. 1.0.3 (right). This result is presented in Chapter 3.

Next, we became interested in discovering and applying new visibility decompositions. We eventually developed several new visibility decompositions of polygons.

Finally, we observed that the Computational Geometry literature contains a number of useful visibility decompositions which we believe can be used to solve additional problems. At this point, the study of visibility decompositions and their applications became the focus of this thesis.

1.1. Why Study Visibility Decompositions?

We have seen in the previous section that visibility decompositions can be used to solve some problems effectively. But are visibility decompositions worthy of the

intense study conducted here? This thesis answers this question clearly in the affirmative. It does so by establishing the usefulness of visibility decompositions in four ways.

Efficiency: Visibility decompositions have already been used effectively to solve numerous problems. A survey of these visibility decompositions and their applications is provided in Chapter 2. It is noteworthy that both optimal ray shooting algorithms use visibility decompositions **[12, 27]**, as do the most efficient circular ray shooting algorithms; see **[1]** and also Chapter 5. To this list we have added a number of new algorithms that solve problems efficiently, based upon visibility decompositions; these are discussed in more detail shortly.

Simplicity: While we have established that the asymptotic complexity of problems can sometimes be reduced by using visibility decompositions, an equally important gain is that of simplicity. Visibility decompositions allow us to reduce the complexity of the class of polygons for which we need to find a solution. It is often the case, in this thesis, that there are results in the literature with the same or similar asymptotic complexities as our results for the same problems. The authors of these results have solved the problems on general polygons, instead of the simpler ones that we use, by instead using complex algorithms and data structures. As a result, when we not have found an asymptotically more efficient algorithm, we have found a simpler one. Furthermore, the constant coefficients that go with the asymptotic complexities are generally lower for our algorithms.

The tendency, in this thesis at least, for solutions to problems using visibility decompositions, to be simpler than solutions that use other methods is a considerable advantage of this technique. It is noteworthy that all of the visibility decompositions presented in this thesis are relatively simple to construct (given a triangulated polygon).

Another way that visibility decompositions provide simplicity is through modular development. Once discovered, the same visibility decomposition is available for use in solving multiple problems. Thus if a problem has two solutions of equal difficulty and efficientcy, one of which uses a visibility decomposition that is also used elsewhere, then the solution using the visibility decomposition is generally the better solution both in terms of understanding and implementation.

Availability: Except for our results, it may be that the authors of the algorithms that use visibility decompositions simply developed the visibility decompositions they needed to solve the problem they were studying; so why single out this technique as something to be studied?

First of all, developing a new visibility decomposition is not necessarily simple and is a research task in and of its own accord. Thus it is useful to have a collection of visibility decompositions and the algorithms that construct them available for researchers to use and for researchers to be familiar with them. It is noteworthy that, for most of the problems solved in this thesis, using visibility decompositions, the visibility decomposition was developed before the problem was itself was considered; the one exception is given in Chapter 6.

Systematic Methods: To some degree the creation and use of visibility decompositions can be approached profitably in a systematic way. Consider, for example, the largest axis-aligned rectangle problem studied in Chapter 6. In [**21**] the authors were able to reduce the asymtotic complexity of solving the largest axis-aligned rectange problem for polygons with holes, in part, by using visibility decompositions. However, by using, what we consider to be a standard technique in applying visibility decompositions, we were able to reduce the preprocessing time and space used to construct the same kind of visibility decomposition for polygons (without holes) by a logarithmic factor. Furthermore, we found this result with minimal effort for

rectilinear polygons; some effort was needed to generalize the result to simple polygons. If the authors of the first result were familiar with the systematic application of visibility decompositions they might well have preceded us in finding this result.

1.2. Results Based on Visibility Decompositions

This section contains a short description of the results in this thesis that apply visibility decompositions. These results are reviewed in more detail in Chapter 2 and then presented in the subsequent four chapters.

In Chapter 2 we formalize the concept of visibility decomposition. We then provide a more through review of the literature on visibility decompositions and review the visibility decompositions developed in this thesis. We also study methods of creating visibility decompositions and applying them to problems is a systematic way. We believe that the process of developing visibility decompositions is interesting beyond it's immediate application to a problem to be solved.

In Chapter 3 we study k^*-wind polygons of which 2^*-wind polygons are a special case. We then focus on the properties of 2^*-wind polygons, especially those relating to visibility decompositions. Next we show how to construct a 2^*-wind subpolygon visibility partition of a polygon with the visibility properties claimed above. We then demonstrate the usefulness of this partitiion by tackling Problem 1.0.2.

In Chapter 4 we show how to further partition a 2^*-wind partition of a polygon into monotone pieces such that any monotonically increasing curve contained in the polygon intersects at most a logarithmic number of them. We then use this result to construct a partition of a polygon into monotone pieces such that the maximum number of the pieces needed to cover any y-monotone curve contained in the polygon is within two of the minimum number possible for any partition of the polygon into y-monotone pieces.

In Chapter 5 we study the application of the visibility partitions studied in Chapter 4 to the circular ray shooting problem. We develop an algorithm that is simpler than the best previous result for this problem. The algorithms also use less preprocessing time and space by constant factors. These results are also presented in [8].

In Chapter 6 we show how to construct a rectangle visibility decomposition of a polygon using SAM polygons (defined in the chapter). We then use our algorithms to reduce the time needed to solve the largest axis-aligned rectangle problem on simple polygons by a logarithmic factor. These results are also presented in [7].

1.3. Other Results

We have several results in this thesis that are neither visibility decompositions nor applications of them but were developed during the research process.

In Chapter 5 we provide an algorithm that reduces the preprocessing complexity of our circular ray shooting algorithm and the circular ray shooting algorithm presented in [16] by a logarithmic factor.

In Appendix A we present a data structure that is needed for Chapter 5. This data structure is the third of three data structures we study in Appendix A to solve three variations of a range query problem. In the first case, the data is stored in an array. We show that in this case the problem has a simple and efficient solution.

In the second case, the data is also stored in an array but the weights are assigned to the entries in the array and the solution has to be particularly efficient when the weights of the entries near the ends of the solution range are high, relative to the weights corresponding to the other entries in the array. We solve this problem by constructing a new kind of weighted binary tree data structure. The algorithm to construct the weighted binary tree is simple; its main operations involve the use of two stacks and basic operations on those stacks.

In the third case, we deal with finding subpaths of trees. Here we use the weighted array case above to find the solution in logarithmic time regardless of how unbalanced the tree is.

Finally, in Appendix B, we present the our solution to Problem 1.0.1. This problem was solved in [19] with a preprocessing time complexity of $O(n \log n)$, a space complexity of $O(n \log n)$, and a query complexity of $O(\log n)$. Our solution reduced the preprocessing time and space complexity to linear and the query complexity to $O(1)$; clearly this is optimal. Furthermore, our algorithm is simple; it does not even require that the polygon be partitioned. These results are also presented in [6].

1.4. Post Thesis Research Plans

In Chapter 7 we present our conclusions and discuss our research plans for visibility decompositions. These plans are twofold.

First, we plan to further exploit the visibility decompositions studied in this thesis to reduce the cost of solving a number of problems in Computational Geometry. To firmly establish that a visibility decomposition is a useful tool we would like to find several applications for each visibility decomposition class we have studied.

Second, we hope to discover new useful visibility decompositions.

The goal is to firmly establish the use of visibility decompositions of polygons as a method of solving problems in Computational Geometry and in related fields.

CHAPTER 2

Polygon Visibility Decompositions

2.1. Introduction

The input to many problems in Computational Geometry is a polygon. A common approach to tackling such problems is to first decompose the input polygon into simpler subpolygons; see, for example, [32]. The idea is to exploit the fact that the subpolygons are simpler than the input polygon to solve the problem on the subpolygons efficiently. The solutions for the subpolygons are then combined to form a solution for the original polygon. This approach relies upon the properties of the subpolygons and upon the properties of the decomposition itself.

We refer to the set of decompositions of polygons that all satisfy some set of properties as a *decomposition class*. There are many decomposition classes of polygons used to solve a large number of problems in Computational Geometry and in other related fields. The best known example of these decomposition classes is the decomposition of polygons into triangles.

This thesis has two main objectives:

(1) To develop decomposition classes of polygons with useful properties.

(2) To find applications for these decomposition classes.

We have defined four new polygon decomposition classes and we use them to develop four new algorithms for solving four different problems in the literature.

The polygon decomposition classes that we define in this thesis all have two properties in common:

(1) The subpolygons of a decomposition have useful properties not generally applicable to the original polygon.

(2) The decompositions fit into a class of decompositions we call "visibility decompositions".

We now pose the question: what is a visibility decomposition and why is it important?

Let Π be a decomposition of an n vertex polygon P into subpolygons and let G be a family of geometric objects contained in P where G is related to a problem to be solved on P. Then a useful property for Π to have is that any element $g \in G$ can be covered by a small number of subpolygons of Π. This property is important because the number of subpolygons required to cover any element of G often reflects the computational difficulty in solving the problem under consideration.

If Π has the property that any object $g \in G$ can be covered by at most $o(n)$ subpolygons of Π, then, for some adjective J describing G, we call Π a J *visibility decomposition* of P. For example if G is the set of chords contained in P and any chord of G can be covered by $o(n)$ subpolygons of Π then we call Π a stabbing visibility decomposition of P. We shall define more types of visibility shortly.

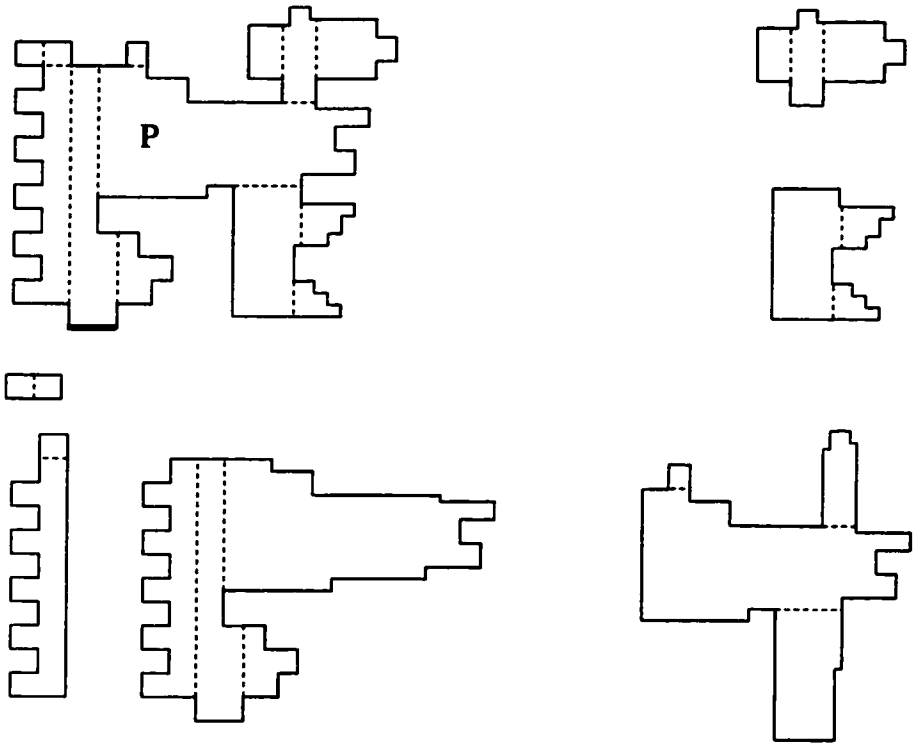
For most known polygon decomposition classes, including the best known, and for any type of visibility considered in this thesis, the decompositions are not generally visibility decompositions. For example, neither triangulations, trapezoidizations, nor convex polygon partitions are generally visibility decompositions for any type of visibility we consider. While these decomposition classes are quite useful, for many problems their use means that it is necessary, in the worst case, to process a large number of subpolygons in order to extract a necessary piece of information, or that one must develop complex algorithms and data structures in order to reduce the costs involved.

Thus there is considerable value in finding polygon decompositions in which any element $g \in G$ can be covered by $o(n)$ subpolygons. Furthermore, it is important to reduce to as small a number as reasonably possible the number of subpolygons needed to cover an object. For every visibility decomposition class studied in this thesis at most $O(\log n)$ subpolygons are needed to cover any element $g \in G$. For several of the decomposition classes only $O(1)$ subpolygons are required and for some cases one subpolygon is all that is needed, in the worst case, to cover any element $g \in G$. See Figure 2.1.1.

If the subpolygons of a polygon decomposition have disjoint interiors then the decomposition is called a *partition*; otherwise the decomposition is called a *cover*.

Let $\wp$ be a family of polygons such that $\Pi \subset \wp$. Then we say that Π is a $\wp$ *subpolygon decomposition* of P. Also, for some adjective K describing $\wp$ we say that Π is a K *subpolygon decomposition* of P. For example, in this thesis, we say that Π is a *monotone subpolygon decomposition* if every element of Π is a y-monotone polygon. We further say that $\wp$ is a *decomposition polygon class* of Π. In this thesis, for any given polygon decomposition, we will only be interested in one decomposition polygon class so we will normally say "$\wp$ is the decomposition polygon class of Π".

Let $\mathcal{A}_1$ be an algorithm whose input is a polygon and whose output is a partition of the polygon. Let $\mathbb{D}_1$ be the set of decompositions that can be constructed by Algorithm $\mathcal{A}_1$ and $\wp_1$ be the family of polygons such that every polygon $P_j \in \wp_1$ is a subpolygon of some decomposition $\Pi \in \mathbb{D}_1$. Then we refer to $\mathbb{D}_1$ as the *decomposition class generated by* $\mathcal{A}_1$ and $\wp_1$ as the *decomposition polygon class generated by* $\mathcal{A}_1$. We further say that $\mathcal{A}_1$ is a $\mathbb{D}_1$ *generating algorithm.* We refer to the set of valid input polygons of $\mathcal{A}_1$ as the *input polygon class* of $\mathcal{A}_1$. Let K be an adjective describing a kind of visibility. We refer to an algorithm that takes as input a polygon (in some



The histogram subpolygon partition of a rectilinear polygon P and the components of the histogram family cover of P. For the partition, any rectangle interior to P can be covered by at most three subpolygons. For the cover, any rectangle interior to P can be covered by one subpolygon. These decompositions are called rectangle visibility decompositions. See Chapter 6.

FIGURE 2.1.1. A histogram cover of a rectilinear polygon

cases polygons with holes) and outputs a K visibility decomposition of the polygon as a *K visibility decomposition generating algorithm.*

In this thesis we are interested in decompositions of polygons that belong to specific visibility decomposition polygon classes and in applying them to problems involving specific classes of geometric objects. Every problem solved in this thesis using visibility decompositions is an application of the following strategy:

Strategy 2.1.1

Input: A problem on polygons of class $\wp_i$ involving a class of geometric objects G.

Output: An algorithm that solves the problem efficiently.

1: Find or develop an algorithm $\mathcal{A}_1$ with input class $\wp_i$ such that each decomposition constructed by $\mathcal{A}_1$ is a J visibility decomposition where J is an adjective describing G. Let $\wp_1$ ($\mathbb{D}_1$) be the decomposition polygon class (decomposition class) generated by Algorithm $\mathcal{A}_1$.

2: Solve the problem for the special case that the input polygon is a member of $\wp_1$.

3: Design a method that, for a given partition $\Pi \in \mathbb{D}_1$ of an input polygon $P \in \wp_i$, can combine the solutions of the problem on some or all components of Π to form a solution of the problem on P.

4: Return the algorithm that is the combination of steps 1, 2, and 3.

In this thesis $\wp_1$, from Strategy 2.1.1, will at various times be the class of monotone polygons (Chapter 4), the class of 2^*-wind polygons (Chapter 3), and separated axis monotone polygons, (henceforth call SAM polygons; see Chapter 6).

In this thesis we will expand the usefulness of visibility decomposition generating algorithms that already exist in the literature and develop several new ones. For the algorithms that already exist in the literature we either simplify them (possibly changing slightly the decomposition class they generate) or develop new algorithms that use them. The visibility decomposition generating algorithms that we develop are all fairly simple and all run in linear time with a small constant coefficient. We solve a number of problems by using the decompositions generated by these algorithms. Each algorithm we develop to solve a problem is considerably simpler and usually more efficient than alternative algorithms in the literature.

We believe that many other problems on polygons, beyond those handled in this thesis, will eventually be solved using the visibility decomposition classes studied here. As a result we expect these visibility decomposition classes to become standard items in the Computational Geometer's tool bag.

2.2. Definitions

A polygon is defined by a finite set of closed straight line segments $\{e_0, \ldots, e_{n-1}\}$ such that e_i and e_{i+1} have a common end point shared by no other edges, addition taken mod n. A polygon is simple if no pair of non-consecutive edges intersect and consecutive edges intersect only at an endpoint. All polygons in this thesis will be assumed to be simple, and thus the term polygon will always refer to simple polygons. In this thesis the vertices or edges of a polygon will always be given in counterclockwise order.

In this thesis we assume that the edges of polygons are directed. The orientation of an edge of a polygon is the direction it is crossed during a counterclockwise traversal of the polygon.

We denote the first vertex of a polygon P by start(P).

Chords of polygons play a key role in this thesis. Let γ be a chord of P. We denote by $P_{\widehat{\gamma}}$ the subpolygon of P determined by γ that contains start(P) unless both of the subpolygons contain start(P) in which case $P_{\widehat{\gamma}}$ denotes the subpolygon that contains the last vertex of P. The remaining subpolygon of P determined by γ is denoted P_γ.

In this thesis the chords of polygons are also directed. Chord γ is given the direction it it receives by virtue of being an edge of $P_{\widehat{\gamma}}$. Note that this is opposite to the direction it would receive by virtue of being an edge of P_γ. The source vertex of γ is denoted $v^-(\gamma)$ and the destination vertex is denoted $v^+(\gamma)$. Unless otherwise stated we assume that start($P_{\widehat{\gamma}}$) = start(P) and start(P_γ) = $v^-(\gamma)$. A key property of this method of orienting chords of P is that any chord of P_γ or $P_{\widehat{\gamma}}$ is given the same orientation as it would receive by virtue of being a chord of P.

For any polygon P there are five classes of geometric objects for which we define visibility decompositions. We denote

(1) by $P_{\dagger}$ the set of chords of P,

(2) by $P_{\imath}$ the set of strictly y-monotone curves contained in P except possibly at their endpoints,

(3) by $P_{\circlearrowleft}$ the set of circular arcs interior to P except possibly at their endpoints,

(4) by $P_{\square}$ the set of axis-aligned rectangles contained in P, and

(5) by $P_{\diamond}$ the set of convex sets contained in P.

Let Π be a polygon decomposition of a polygon P. For any geometric object g contained in P we denote by $\mathsf{C}(\Pi, g)$ the minimum number of elements of Π needed to cover g. We call $\mathsf{C}(\Pi, g)$ the *cover number* of Π with respect to g. Let G be a family of geometric objects contained in P. We denote $\mathsf{C}(\Pi, G) = \max\{\mathsf{C}(\Pi, g) : g \in G\}$. We further denote $\mathsf{C}_{\dagger}(\Pi) = \mathsf{C}(\Pi, P_{\dagger})$, $\mathsf{C}_{\imath}(\Pi) = \mathsf{C}(\Pi, P_{\imath})$, $\mathsf{C}_{\circlearrowleft}(\Pi) = \mathsf{C}(\Pi, P_{\circlearrowleft})$, $\mathsf{C}_{\square}(\Pi) = \mathsf{C}(\Pi, P_{\square})$, and $\mathsf{C}_{\diamond}(\Pi) = \mathsf{C}(\Pi, P_{\diamond})$. We call

(1) $\mathsf{C}_{\dagger}(\Pi)$ the *stabbing cover number* of Π,

(2) $\mathsf{C}_{\imath}(\Pi)$ the *monotone cover number* of Π,

(3) $\mathsf{C}_{\circlearrowleft}(\Pi)$ *the circular cover number* of Π,

(4) $\mathsf{C}_{\square}(\Pi)$ the *axis-aligned rectangle cover number* of Π, and

(5) $\mathsf{C}_{\diamond}(\Pi)$ the *convex cover number* of Π.

When no adjective is used or implied the expression "cover number" implies "stabbing cover number". We will also shorten "axis-aligned rectangle cover number" to just "rectangle cover number" since, in this thesis, the only rectangle covers considered are axis-aligned.

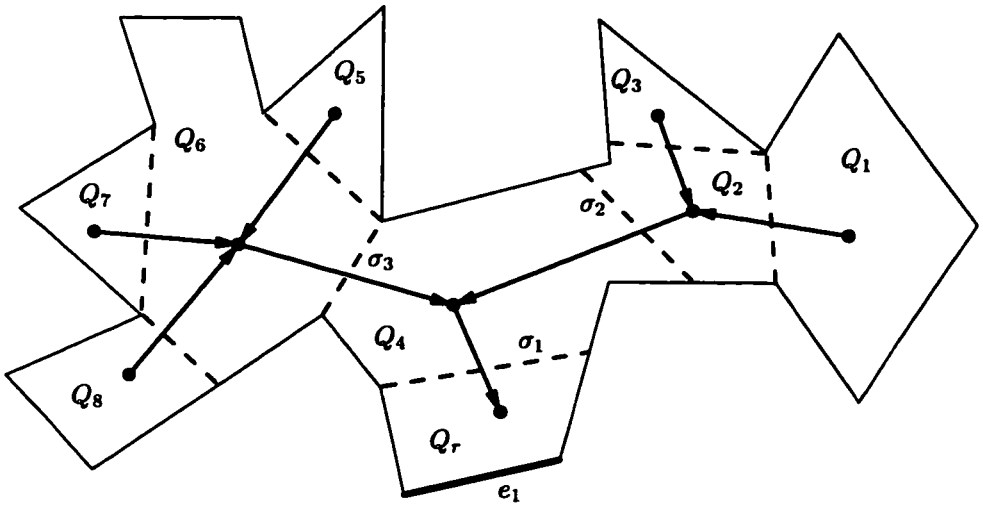
We say that Π is:

(1) *a stabbing visibility decomposition* of P if $\mathsf{C}(\Pi, P_{\dagger}) = o(|P|)$,

(2) a *monotone visibility decomposition* of P if $\mathsf{C}(\Pi, P_{\imath}) = o(|P|)$,

(3) a *circular visibility decomposition* P if $\mathsf{C}(\Pi, P_{\circlearrowleft}) = o(|P|)$,

(4) a *rectilinear visibility decomposition* of P if $\mathsf{C}(\Pi, P_{\square}) = o(|P|)$, or



$e_1 = door(Q_r)$. $\gamma_1 = door(Q_4)$. $\{\gamma_2, \gamma_3\} = Q_4$.
$succ(\gamma_1) = Q_r$. $pred(\gamma_1) = Q_4$.
Q_r is the parent of Q_4 and Q_4 is the only child of Q_r.

FIGURE 2.2.1. A tree partition of a polygon and the dual graph of the partition

(5) a *convex visibility decomposition* of P if $\mathbf{C}(\Pi, P_\diamond) = o(|P|)$.

When no adjective is used or implied the expression "visibility decomposition" implies "stabbing visibility decomposition".

Let $\Pi = \{Q_1, Q_2, ..., Q_m\}$ be a decomposition of a polygon P. We denote by Π^* the graph such that $V(\Pi^*) = \{Q_1^*, Q_2^*, ..., Q_m^*\}$ and $E(\Pi^*) = \{\overline{Q_i^* Q_j^*};\ 1 \leq i < j \leq m$: the intersection of the boundaries of Q_i and Q_j contain a line segment$\}$. Clearly if Π is a partition then Π^* is a planar graph.

If Π^* is a tree then we call Π a *tree partition.* Assume now that Π is a tree partition. One vertex of Π^* is then designated the root vertex. Let this vertex be Q_r^*. We call Q_r the *root polygon* of Π. For each polygon Q of Π such that Q^* is a leaf node of Π^* we call Q a *leaf polygon* of Π. We assume that the edges of Π^* have been oriented so that all paths of Π^* are directed towards Q_r. Let Q be any element of Π except Q_r and Q_p be the element of Π such that $\overrightarrow{Q^* Q_p^*} \in E(\Pi^*)$. Then we denote by door(Q) the chord of P belonging to Q that separates Q from Q_p. Unless otherwise

stated we assume that the endpoints of door(Q) are the first and last vertices of Q visited during a counterclockwise traversal of Q.

We denote by windows(Q) the remaining chords of P that belong to Q. We further denote by windows (Q_r) the chords of P that belong to Q_r. Let $\gamma = \text{door}(Q)$. Then Q is called the *successor* of γ and Q_p its *predecessor*. The predecessor and successor of γ will be denoted by pred(γ) and succ(γ) respectively. We further call Q_p the *parent polygon* of Q and Q a *child polygon* of Q_p. Finally, by door(Q_r) we denote an edge of Q_r that is not a chord of P; which edge it is, is dependent upon the specific definition of Π. We denote by $\widehat{\Gamma}(\Pi)$ the chords of P that determine Π. See Figure 2.2.1.

Most of the decomposition classes used in this thesis are partition classes and most of the partition classes are tree partition classes. Most of the tree partition classes in this thesis have a common form.

Let $\wp$ be a class of polygons such that one edge of each member $W \in \wp$ is called a base edge of W and is denoted base(W). (In the degenerate case base(W) may be a vertex.) Let T be a simple polygon and e be an edge of T or some portion of an edge of T. Let Q be a subpolygon of T such that $Q \in \wp$, base$(Q) = e$, and Q is not contained in any subpolygon $R \subseteq T$ where $R \in \wp$ and base$(R) = e$. Then we say that Q is a *maximal $\wp$ subpolygon of T with respect to e*. We further denote $Q = \hat{\mathcal{P}}(\wp, T, e)$. We point out that for some polygon partition classes there may be more than one maximal $\wp$ subpolygon of T with respect to e. In this case an additional rule is required to select an appropriate $\wp$ subpolygon of T. We define $\hat{\Pi}(\wp, W, e)$ as follows:

DEFINITION 2.2.1. [1] Let $\wp$ be a class of polygons such that each member $W \in \wp$ has a base edge denoted base(W). Let T be a polygon and e be an edge of T or portion of an edge of T.

[1]This definition is based on a similar definition in [42].

If $T \in \wp$ and $e = \text{base}(T)$ then

$$\hat{\Pi}(\wp, T, e) = \{T\}.$$

Otherwise

$$\hat{\Pi}(\wp, T, e) = \left\{\hat{\mathcal{P}}(\wp, T, e)\right\} \cup_{\gamma_i \in \text{windows}\left(\hat{\mathcal{P}}(\wp,T,e)\right)} \hat{\Pi}(\wp, P_{\gamma_i}, \gamma_i).$$

We call $\hat{\Pi}(\wp, T, e)$, the *maximal $\wp$ partition of T with respect to e*. If a partition Π is a maximal partition of a polygon P with respect to some edge or part of an edge e belonging to P then Π is called a *maximal $\wp$ partition of P*. We call the set of maximal $\wp$ partitions a *maximal $\wp$ partition class*.

All four of the tree visibility partition classes in this thesis are maximal partition classes:

(1) The 2*-wind subpolygon partition class; see Chapter 3.

(2) The monotone subpolygon partition class; see Chapter 4.

(3) The histogram subpolygon visibility partition class; see Chapter 6.

(4) The pseudo-histogram subpolygon visibility partition class; see Chapter 6.

See, for example, polygon P of Figure 2.1.1 which is partitioned into histogram polygons.

We point out that Definition 2.2.1 implies a strategy for constructing maximal partition classes.

In this thesis, the last three of the maximal partition classes listed above are actually generated using Strategy 2.2.1. The 2*-wind subpolygon partition class could also be constructed this way but a simpler and more flexible algorithm is used instead. However the version of the 2*-wind subpolygon class that deals with polygons with holes uses a version of Strategy 2.2.1; see Chapter 7.

Strategy 2.2.1 BuildMaximalPartition($\wp, T, e$)

Input: A polygon class $\wp$, a polygon T, and a line segment or point e belonging to T.

Output: The maximal $\wp$ partition Π of T with respect to e.

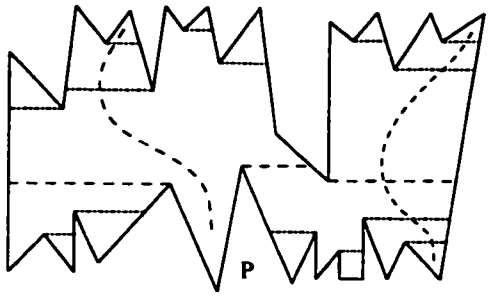
1: if $T \in \wp$ and $e = \text{base}(T)$ then Return $\{T\}$.
2: Let Q be the maximal $\wp$ polygon of T with respect to e. If there is more than one choice for Q then some rule is applied to select Q.
3: Let $W = \gamma_1, \gamma_2, ..., \gamma_k$ be the windows of Q.
4: Return $\{Q\} \cup_{i=1}^{k}$ BuildMaximalPartition$(\wp, P_{\gamma_i}, \gamma_i)$.

Many of the problems that we will study involve preprocessing a polygon to construct a data structure that allows us to perform some kind of query operation on the polygon quickly. Solutions to such problems have three complexities of interest to us. The first complexity is the amount of time spent preprocessing the polygon. We refer to this complexity as the *preprocessing complexity.* Often this complexity will be determined entirely by the time required to construct a decomposition of the polygon. We may then use the term *decomposition complexity* instead. We refer to the section of the algorithm that does the preprocessing as the *preprocessing algorithm* or, if appropriate, the *decomposition algorithm.*

The second complexity is the amount of space required to store the data structure constructed by the preprocessing algorithm. We refer to this complexity as the *space complexity.* Often this will be the amount of space required to store the decomposition constructed.

The third complexity is the amount of time required to perform a query. We refer to this complexity as the *query complexity.* We refer to the section of the algorithm that performs the query as the *query algorithm.*

2.2.1. Visibility Decomposition Combinations. Visibility decompositions can be combined to form new decompositions. We will use two methods to achieve this; "composition" and "fusing".



A partition Π of a polygon P consisting of a 2^*-wind subpolygon partition of P (thick dashed lines) followed by a partition of each 2^*-wind subpolygon into monotone subpolygons (thin dashed lines). A 2^*-wind subpolygon partition has a monotone cover number of at most two. A monotone subpolygon partition of a 2^*-wind polygon has a monotone cover number that is at most logarithmic in the size of the 2^*-wind polygon. Combining compositions we have $\mathbf{C}_i(\Pi) = O(\log |P|)$. This partition class is used for circular ray shooting.

FIGURE 2.2.2. The composition of visibility decomposition classes

Let $\wp_0$ be a polygon class, $\mathbb{D}_1$ be a polygon decomposition class, and $\mathcal{A}_1$ be an algorithm such that $\mathcal{A}_1$ is a $\mathbb{D}_1$ generating algorithm whose input polygon class is $\wp_0$. Let $\wp_1$ be the polygon class of $\mathbb{D}_1$. Let $\mathbb{D}_2$ be a polygon decomposition class and $\mathcal{A}_2$ be an algorithm such that $\mathcal{A}_2$ is a $\mathbb{D}_2$ generating algorithm whose input polygon class is $\wp_1$.

For any given polygon $P \in \wp_0$ we can construct a decomposition $\Pi_2 \in \mathbb{D}_2$ of P by applying algorithm $\mathcal{A}_1$ to construct a decomposition Π_1 of P and then applying algorithm $\mathcal{A}_2$ to each subpolygon of Π_1 of P. Let A_3 be the algorithm that combines algorithms $\mathcal{A}_1$ and $\mathcal{A}_2$ in this way. We say that algorithm $\mathcal{A}_3$ is the *composition* of algorithms $\mathcal{A}_1$ and $\mathcal{A}_2$. Let $\mathbb{D}_3$ be the decomposition class generated by algorithm $\mathcal{A}_3$. We further say that decomposition class $\mathbb{D}_3$ is the *composition* of decomposition classes $\mathbb{D}_1$ and $\mathbb{D}_2$. Note that $\wp_2$ is the partition polygon class of $\mathbb{D}_3$. See Figure 2.2.2.

By this process of polygon decomposition composition we can construct visibility decompositions with more restricted decomposition polygon classes. In combination with Strategy 2.1.1 this can lead to simpler or more efficient algorithms. In Chapter

5 we apply this approach to solving the circular ray shooting problem and in Chapter 6 we apply this approach to solving the largest axis-aligned rectangle problem.

Although most of the decompositions in this thesis are partitions, we will also use visibility covers. One way to construct a visibility cover is by using a visibility partition Π as a base and constructing each of the subpolygons of the cover by fusing together elements of Π. All the polygon visibility covers we will study are constructed by this general method at least in part.

Let Π be a visibility partition of a polygon P. Let Π^c be a cover of P such that, for any polygon $Q^c \in \Pi^c$, some subset of Π is a partition of Q^c. Clearly Π^c is a visibility cover of P. We contrast the visibility decompositions Π and Π^c as follows:

(1) For any family of geometric objects G of P we have that $\mathbf{C}(\Pi^c, G) \leq \mathbf{C}(\Pi, G)$. If $\mathbf{C}(\Pi^c, G) < \mathbf{C}(\Pi, G)$ then Step 3 of Strategy 2.1.1 should be easier to solve. It is for this reason that we might prefer Π^c to Π for solving a problem.

(2) In practice the polygon decomposition class of Π^c is at least as complex as the polygon decomposition class of Π. This should make Step 1 of Strategy 2.1.1 more difficult to solve.

Thus for partition Π^c to be a better choice to solve a problem than Π we must exploit advantage 1 while overcoming disadvantage 2.

Now assume Π is a visibility tree partition of P. Then one method of constructing Π^c is of particular interest. For each polygon $Q \in \Pi$ such that Q^* is not a leaf node of Π^* we denote by Q^f the subpolygon of P that is the union of Q and all child polygons Q in Π. Let $\{Q_1, Q_2, ..., Q_k\}$ be the non-leaf polygons of Π. Then we denote $\Pi^f = \left\{Q_1^f, Q_2^f, ..., Q_k^f\right\}$. If the elements of Π are called X polygons then we give the polygons of Π^f the name X *family polygons*. See Figure 2.1.1.

Let $\Pi = \{P_1, P_2, ..., P_k\}$ be a decomposition of a polygon P. We denote $\|\Pi\| = \sum_{i=1}^{k} |P_i|$. One of the reasons that family covers are useful is the following observation:

OBSERVATION 2.2.2. *Let* Π *be a tree partition of a polygon* P. *Then* $\|\Pi^f\| < 2\|\Pi\|$. *Furthermore* Π^f *can be constructed from* Π *in* $O(\|\Pi\|)$ *time and space.*

In the thesis we solve one problem using family polygon classes with Strategy 2.1.1; the largest axis-aligned rectangle problem; see Chapter 6. In this case the rectilinear cover number is reduced from three to one by replacing a partition by a cover. As a result, Strategy 2.1.1 is reduced to solving the problem for the class of family polygons involved (pseudo-histogram family polygons; in the case of rectilinear polygons histogram family polygons).

2.3. Previously Known Visibility Decompositions

We now describe the visibility decomposition classes in the literature that we are aware of. We use n to denote the size of an input polygon.

Although the term "visibility decomposition" is defined for the first time here, there are already at least seven visibility decompositions in the Computational Geometry literature.

(1) To the best of our knowledge the first polygon visibility decomposition class was presented in **[49]**, (see also **[50]**) where a polygon was partitioned into weakly edge visible subpolygons in such a way that a chord intersects at most three subpolygons. We call this class of partitions the *weakly edge visible subpolygon visibility partition class.* In **[49]**, this partition class was used to solve link distance problems. Subsequently it has been used to solve other problems **[3]**, **[36]**, **[30]**. Of these **[3]** is the only result that we are aware of that exploits the stabbing visibility properties of this partition class despite the fact that its usefulness in this regard was claimed in **[50]**. It is also straightforward to construct a shortest path tree of a polygon by first constructing this partition.

(2) In [12] the geodesic triangle subpolygon visibility partition class was developed and then used to solve the ray shooting problem. The cover number for a member of this visibility partition class is $O(\log n)$. The ray shooting algorithm developed in [12] requires linear time and space to construct the necessary data structures and $O(\log n)$ time to find the collision point of a query ray with the polygon. Thus this algorithm is optimal. One of the difficulties with these algorithms is that the geodesic triangle partition of a polygon is difficult to construct requiring a number of complex data structures.

(3) In [27] the polygon decomposition composition technique and a method to partition geodesic triangles into triangles was used to develop the triangle subpolygon visibility partition class. The cover number of a partition of this class is also $O(\log n)$. With this partition class the query part of the ray shooting problem can be solved easily.

(4) In [16] the hierarchical vertical cover class was developed. The visibility cover number of a cover of this class is at most two, regardless of the class of geometric object being covered, because two of the subpolygons cover the entire polygon. However the polygon decomposition class of the algorithm is the class of simple polygons. This visibility cover class gains its power not from the properties of the decomposition class but from the hierarchical structure of the cover.

In [16] an algorithm is then developed whose input is a hierarchical vertical cover of a polygon and whose output is a monotone subpolygon visibility cover of the polygon such that the cover number is $O(\log n)$. This cover requires $O(n \log n)$ time to construct and, in worst case, also requires $O(n \log n)$ space. The circular visibility cover number of a cover of a polygon

generated by this algorithm is also $O(\log n)$. This algorithm is used in [16] to solve a number of problems including the circular ray shooting problem.

(5) In [37] a partition class of rectilinear polygons into "normal-histogram" polygons was developed. This result was later generalized in [34] to a partition class of general polygons into pseudo-histogram polygons. These partitions have been used to solve the problem of finding the shortest rectilinear path in rectilinear polygon [42], constructing the medial axis of a polygon [34, 17], and finding the shortest diagonal in a polygon [53].

A partition of a polygon that is an element of either of these partition classes has the property that chords of the polygon, where the chords satisfy certain constraints, cannot intersect more than three of the subpolygons of the partition.

A partition contained in either of these classes also has a rectangle visibility cover number of at most three. We use this property of these partition classes in Chapter 6 to develop an improved solution to the problem of finding the largest axis-aligned rectangle inscribed by a polygon.

(6) In [21] the vertically separated horizontally convex subpolygon class was developed; see Chapter 6. The rectangle cover number of a cover of this class is one. This cover class is used to find the largest axis-aligned rectangle inscribed by a polygon with holes.

(7) In [40] a cover of a polygon into fan subpolygons was developed. A fan polygon consists of three edges connected by three concave chains. This cover requires $O(n^4)$ time and space to construct. This class is used to find the largest triangle contained in a polygon in $O(n^4)$ time. The space requirement is kept to linear by only constructing and using one fan at a time. A number of other containment problems are also solved.

(8) Finally, in [12], a polygon with holes is reduced to a polygon with a simple interior using a linear number of line segments joining the polygon and its holes. This construction is such that a line interior to the polygon with holes intersects at most $O(\sqrt{n})$ line segments where n is the total number of vertices in the polygon and its holes.

2.4. Other Previous Results

For a number of the problems we study in this thesis there are previous results in the literature that do not use visibility decompositions. Let P be a simple polygon.

In [19] (see also [20]) the problem of computing the areas of the subpolygons of P determined by any chord of P was studied. It was shown that this problem can be solved in $O(\log n)$ time for a query chord after a preprocessing stage requiring $O(n \log n)$ time and $O(n \log n)$ space. We improve upon this result in Appendix B.

In [38] the problem of partitioning a polygon into the minimum number of y-monotone subpolygons was studied. It was shown that, when the partition is restricted to being constructed by chords of the polygon, a partition can be constructed in $O(n \log n + nN + N^3)$ time where N is the number of reflex vertices of the polygon. This result was improved to $O(n^2)$ in [5]. In Chapter 3 we study the general version of this problem; that is where we are not restricted to partitioning the polygon using only its chords.

In [1] the problem of determining where a directed circular arc first intersects a simple polygon was studied. It was shown that a polygon with n vertices can be preprocessed in $O(n \log^3 n)$ time and $O(n \log^3 n)$ space so that a circular ray shooting query can be done in $O(\log^4 n)$ time. This result was recently improved in [16] and we improve upon this result further in Chapter 5. Both the result in [16] and the result in Chapter 5 use visibility partitions.

2.5. Visibility Decompositions From Our Research

To this list of visibility decomposition classes we add about four more. We use n to denote the size of an input polygon.

In Chapter 3 we define the polygon class called k-wind polygons where $k \geq 1$. We then show how to construct the 2^*-wind subpolygon visibility partition of a polygon. For any 2^*-wind visibility partition the cover number is at most two and the circular cover number is at most four.

In Chapter 4 we also show how to further partition a 2^*-wind partition into monotone subpolygons. The cover number and circular cover numbers of such a partition are both $O(\log n)$. We use this partition class to solve the circular ray shooting problem; see Chapter 5.

In Chapter 6 we show how to construct the SAM subpolygon rectangle visibility cover of a polygon. The rectangle cover number of this cover class is one. These covers are constructed from histogram family covers. We use them used to solve the largest axis-aligned rectangle problem.

All of the visibility decomposition generating algorithms that we develop are relatively simple assuming the input polygon is already triangulated and each algorithm requires linear time and space to construct its partition of a polygon. A polygon can be triangulated in linear time by a complex algorithm [13]. Alternatively, there is a choice of simpler $O(n \log n)$ time triangulation algorithms to choose from; see for example [15], [26], and [23]. We point out that these algorithms sometimes require $o(n \log n)$ time to construct a triangulation of a polygon.

We use our visibility decomposition classes to solve a number of problems and the algorithm we develop for each problem we solve is considerably simpler than its competitors.

In the following subsections we consider the decomposition classes mentioned here in more detail.

2.5.1. The 2*-Wind Subpolygon Partition Class. In Chapter 3 we study the 2*-wind subpolygon partition class. We prove several useful properties of 2*-wind polygons. We then present a simple algorithm (after triangulation) to construct a 2*-wind polygon visibility partition of a simple polygon.

The 2*-wind subpolygon partition class has a particularly strong connection to this thesis. It was the first visibility decomposition we developed. We used it originally to solve a problem that we call the area problem. This solution was the first optimal algorithm for this problem.[2] However we have since found a much simpler optimal algorithm for this problem which does not use visibility decompositions. This algorithm is presented in Appendix B.

In Chapter 3 we present a direct application of the 2*-wind subpolygon partition class. We study the problem of partitioning a polygon into the minimum number of y-monotone subpolygons. We develop an algorithm that determines this number in linear time. The algorithm also determines a maximum set of pairs of points that can be joined by a set of non-intersecting y-monotone chains so as to construct such a partition. Constructing such a partition can take $O(n^2)$ space. Although this problem has not been studied in the literature before, a related problem has been and a comparison is worthwhile. Consider that we wish to partition a polygon into the minimum number of y-monotone subpolygons such that the partition is constructed using only chords of the polygon. In [38] it was shown that this problem can be solved in $O(n \log n + nN + N^3)$ time where N is the number of reflex vertices. These results were improved to $O(n^2)$ in [11]. Thus although the problem we solve appears

[2]This is a simplification. Our first solution to the area problem used a 3*-wind subpolygon visibility cover class. This algorithm is even more complex than the algorithm based on the The 2*-wind subpolygon partition class.

harder it has a solution that is simple and more efficient that the algorithms in [38] and [11]. For more information see Section 3.6.

We are actively studying the 2*-wind subpolygon partition class with the intent of finding other direct applications. In the meantime, in Chapter 4, we use the 2*-wind subpolygon visibility partition class to construct another visibility decomposition class by the method of composition. We believe that using the 2*-wind subpolygon partition class to construct other visibility decomposition classes will prove to be more profitable than direct applications in the long run.

Since weakly edge visible polygons are special (and simpler) cases of 2*-wind polygons, one would expect the weakly edge visible subpolygon partition of a polygon to be more useful than the 2*-wind subpolygon partition of the polygon. However, the particular 2*-wind subpolygon partition class that we define has a number of advantages over the weakly edge visible subpolygon partition class. Let Π_w be the 2*-wind subpolygon partition of a polygon P and Π_v be the weakly edge visible subpolygon partition of P.

First, $\mathbf{C}_{\dagger}(\Pi_w) \leq 2$ while $\mathbf{C}_{\dagger}(\Pi_v) \leq 3$. The difference can have a considerable impact when designing an algorithm.

Second, $\mathbf{C}_{\imath}(\Pi_w) \leq 2$ and $\mathbf{C}_{\circlearrowleft}(\Pi_w) \leq 4$ whereas $\mathbf{C}_{\imath}(\Pi_v) = O(n)$ and $\mathbf{C}_{\circlearrowleft}(\Pi_v) = O(n)$. It follows that a 2*-wind visibility partition is also a monotone visibility partition and a circular visibility partition whereas a weakly edge visible partition generally is neither.

Third, there are more useful properties relating to neighbors of a given 2*-wind subpolygon in the 2*-wind subpolygon partition class than in the weakly edge visible subpolygon partition class.

Thus the 2*-wind partition class is sometimes more useful in solving a problem than the weakly edge visible partition class. In this thesis we present several applications of the 2*-wind partition class but no applications of the the weakly edge visible partition class. However, this should not be interpreted to suggest that the former class is more useful. We have under development several applications of the weakly edge visible partition class as well but this work is currently incomplete.

2.5.2. The Monotone Subpolygon Partition Class. In Chapter 4 we present the monotone subpolygon monotone visibility partition class. For a partition of this class the monotone cover number is $O(\log n)$. It follows that the circular cover number for a partition of this class is also $O(\log n)$. The monotone subpolygon partition class generating algorithm is simple. We use this partition class, in combination with the 2*-wind partition class and Strategy 2.5.1, to solve the circular ray shooting problem; preprocess a simple polygon so that, when given a query directed circular arc, one can quickly find the first point on the arc that intersects the boundary of the polygon. This algorithm is simple except for the need for Voronoi diagrams. It is based on the algorithm in [1] but is a considerable simplification and reduces the preprocessing time from $O(n \log^4 n)$ to $O(n \log^3 n)$, the space complexity from $O(n \log^3 n)$ to $O(n \log n)$ and the query time from $O(n \log^3 n)$ to $O(n \log n)$. With the additional use of weighted binary trees, developed in Appendix A, we reduce the query time of our algorithm to $O\left(\log^2 n\right)$.

Recently in [16] a new algorithm for this problem was presented, also based on the algorithm in [1]. This algorithm has the same query and space complexity values as our algorithm but the preprocessing time complexity is slower by a $O(\log n)$ factor. Our algorithm has a number of other advantages over this algorithm; see Chapter 4 for details.

In Chapter 4 we also present a second algorithm for partitioning a polygon into y-monotone subpolygons. This algorithm constructs a partition of a simple polygon such that the monotone visibility cover number of the partition is within two of the minimum for the polygon. We also show that if we are restricted to using horizontal chords then we can construct a partition whose monotone visibility cover is minimized. All algorithms in this chapter are simple and efficient.

2.5.2.1. *A Problem Solving Strategy Involving Monotone Polygons.* The 2^*-wind polygon class is a simple generalization of the class of y-monotone polygons and, in fact, a 2^*-wind polygon can be further partitioned into y-monotone polygons in such a way that any y-monotone curve can intersect at most $O(\log n)$ monotone subpolygons.

These connections between monotone polygons and 2^*-wind polygons lead us to a simple strategy for solving problems on polygons.

Strategy 2.5.1

Input: A problem on polygons.
Output: An algorithm that solves the problem efficiently.

1: Find or design an algorithm that solves the problem for the case of monotone polygons.
2: Generalize this algorithm to work for the case of 2^*-wind polygons. This may involve the partitioning of the polygon into y-monotone subpolygons.
3: Use the algorithm from Step 2 and the 2^*-wind polygon partition class generating algorithm to develop an algorithm that solves the problem.
4: Combine Steps 1, 2, and 3 and return the resulting algorithm.

The circular ray shooting problem (see Chapter 5) is solved using Strategy 2.5.1.

2.5.3. The SAM Polygon Rectangle Visibility Cover Class. A SAM (separated axis monotone) polygon is a polygon which is either a vertically separated horizontally convex polygon or a horizontally separated vertically convex polygon. In Chapter 6 we present a SAM subpolygon rectangle visibility cover class for rectilinear polygons and a slight variation of this cover class for general polygons. The rectangle

cover number for these covers is one. Both of these cover classes are constructed using the polygon decomposition composition technique.

We consider first a rectilinear polygon P. We start by constructing a histogram polygon rectangle visibility partition Π of P. Now $\mathsf{C}_{\square}(\Pi) \leq 3$. From this partition we construct the histogram family polygon rectangle visibility cover of P; namely Π^f. Now $\mathsf{C}_{\square}(\Pi^f) = 1$. With the addition of some additional steps to deal with pseudo-histogram polygons instead of histogram polygons, this algorithm also works for general polygons.

We next construct a SAM subpolygon rectangle visibility partition of each histogram family polygon in Π^f. (This statement is not rigorous; see Chapter 6 for details.) The composition of these decompositions is a SAM subpolygon rectangle visibility cover of P. Call this cover Π_s. Now $\mathsf{C}_{\square}(\Pi_s) = 1$. With the addition of a slight kludge this algorithm also works for general simple polygons.

Despite the number of steps we have described in the construction of the SAM subpolygon cover class the generating algorithm for this cover class is fairly simple. We believe that it is simpler than the algorithm it replaces [**21**] although only marginally so. More importantly the SAM subpolygon rectangle visibility cover class requires linear space whereas the one in [**21**] requires $O(n \log n)$ time and space to construct.

We use this cover class to find the largest axis-aligned rectangle inscribed in a rectilinear polygon. We solve this problem in $O(n \log n)$ time. This in an improvement over the algorithm in [**21**] which requires $O(n \log^2 n)$ time.

The additional steps needed for the general polygon case do not affect the time or space complexity. Thus, for the general case, we also reduce the time required to solve the largest inscribed rectangle problem from $O(n \log^2 n)$ to $O(n \log n)$.

We point out that in [21] this problem is solved for polygons with holes. We are investigating how to generalize our results to apply to polygons with holes as well.

CHAPTER 3

2-Wind Subpolygon Partitions

3.1. Introduction

In this chapter we present 2^*-wind subpolygon visibility partitions. The 2^*-wind subpolygon visibility partition class can also be further partitioned into other visibility partitions. They are the topics of the following three chapters where applications of these subpolygon visibility partition classes are presented.

In Section Two we define k^*-wind chains and k^*-wind polygons for any non-negative integer k. We also define what it means for a k^*-wind chain or k^*-wind polygon to be proper.

In Section Three we focus on proper 2^*-wind polygons and chains and establish a number of their useful properties.

In Section Four we present proper 2^*-wind subpolygon visibility partitions. We prove that a proper 2^*-wind partition of a polygon has a cover number of two, a circular cover number of four, and a convex cover number of two (Any convex closed curve can be covered by two subpolygons.).

3.2. k^*-Wind Chains and Polygons

In this section we define k^*-wind chains and k^*-wind polygons. We begin by providing some definitions and notation. A polygonal chain (henceforth called a chain) is a concatenation of line segments in the plane; the segments are called edges and their endpoints are called vertices. In this thesis a chain will always be directed and thus has a starting vertex (edge) and an ending vertex (edge). The edges of

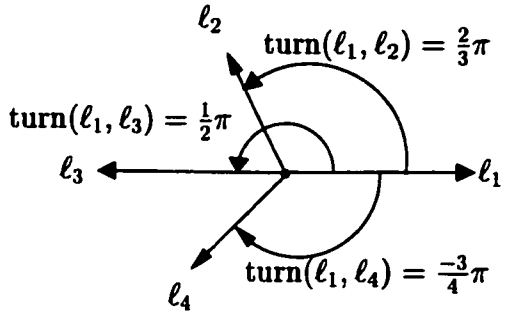


FIGURE 3.2.1. Direction of turns on chains

a chain are also directed and have the orientation implied by the chain. Unless otherwise stated a traversal of a chain is performed in the direction implied by the chain. A chain is written by listing its edges and vertices in the order in which they occur in the chain. Thus a chain C with $m+1$ vertices and m edges may be written $\{v_0, e_0, v_1, e_1, v_2, \ldots, e_{m-1}, v_m\}$. We say that a chain is closed if $v_0 = v_m$ and open otherwise; all other vertices and edges of a chain are assumed to be distinct. If C is an open chain then we call v_0 and v_m the *end vertices* of C and the remaining vertices the *inner vertices* of C. Similarly, if C is open, we call e_0 and e_m the end edges of C and the remaining edges the inner edges of C. If C is a closed then we call all of its vertices inner vertices and all of its edges inner edges.

If C is a closed chain then it is a polygon. When a polygon is specified the last vertex is normally not given. Thus, C can be specified as $\{v_0, e_0, v_1, e_1, v_2, \ldots, e_{m-1}\}$ if it is closed. If C is a polygon then v_j denotes $v_{(j \bmod m)}$ and e_j denotes $e_{(j \bmod m)}$. Although a polygon P has no end vertices it does have a start vertex which is the first vertex in the list of vertices of P. We denote this vertex by start(P). If P is a simple polygon it will always be oriented in the counterclockwise direction.

For convenience we will sometimes write the same chain without the edges as $\{v_0, v_1, v_2, \ldots, v_m\}$ or without the vertices as $\{e_0, e_1, e_2, \ldots, e_{m-1}\}$. Unless otherwise stated a chain is assumed to be defined by its vertices only.

To simplify matters we assume that no three consecutive vertices of a chain are collinear.

Let α be a polar angle. We use $\angle\alpha$ to denote the value ($\alpha \bmod 2\pi$). We further denote $\alpha^{\perp} = \angle(\alpha + \frac{\pi}{2})$; that is, the polar direction orthogonal to α. Let ℓ be a directed line, ray, or edge. We use $\angle\ell$ to denote the positive polar angle in radians between ℓ and the x-axis. We further denote $\angle\ell^{\perp} = \angle(\angle\ell + \frac{\pi}{2})$; that is the polar direction orthogonal to ℓ. We use $-\ell$ to denote the directed line coincident with ℓ but with opposite orientation.

For any three points p, q, r we denote $\angle(p, q, r) = (\angle(\overrightarrow{qr}) - \angle(\overrightarrow{qp})) \bmod 2\pi$. Let ℓ_1 and ℓ_2 be two polar angles, rays, directed lines, or directed edges. Let $\widehat{\ell_1}$ ($\widehat{\ell_2}$) be the unit ray whose origin is $(0,0)$, whose other endpoint is p_1 (p_2), and such that $\angle\widehat{\ell_1} = \angle\ell_1$ ($\angle\widehat{\ell_2} = \angle\ell_2$). Then we denote $\text{turn}(\ell_1, \ell_2) = (\angle(p_1, (0,0), p_2) + \pi) \bmod 2\pi) - \pi$; see Figure 3.2.1. Note that $\text{turn}(\ell_1, -\ell_1) = \pi$ for any ℓ_1.

For any vertex v and polar angle α we denote by $\alpha(v)$ the directed line ℓ through v such that $\angle\alpha = \angle\ell$.

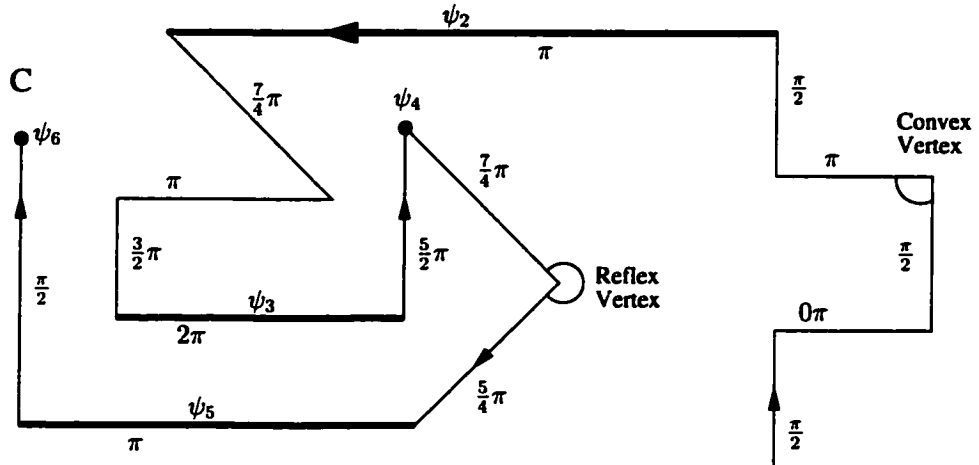
Let $C = \{v_0, e_0, v_1, e_1, v_2, \ldots, e_{m-1}, v_m\}$ be a chain and α be a polar angle such that $0 \leq \alpha < 2\pi$. We now define $\Delta_\alpha(C, i)$, a parameter to measure how C winds, as follows:

- $\Delta_\alpha(C, 0) = \text{turn}(\alpha, e_0)$.
- $\Delta_\alpha(C, i) = \Delta_\alpha(C, i-1) + \text{turn}(e_{i-1}, e_i)$, for $i \geq 1$.

If C is closed, we also define:

- $\Delta_\alpha(C, i) = \Delta_\alpha(C, i+1) - \text{turn}(e_i, e_{i+1})$, for $i < 0$.

When C is implied we denote $\Delta_\alpha(i) = \Delta_\alpha(C, i)$. Also $\Delta(C, i)$ denotes $\Delta_0(C, i)$ and $\Delta(i)$ denotes $\Delta_0(i)$. Note that unlike $\angle e_i$, $\Delta_\alpha(C, i)$ is not bounded by 0 or 2π. See Figure 3.2.2. If C is open we further denote $\Delta_\alpha(C) = \Delta_\alpha(m-1)$ and $\Delta(C) = \Delta_0(C)$. If C is closed (a polygon) we instead denote $\Delta(C)$ by $\Delta_{\angle e_{m-1}}(C, m-1)$.



The x-cusps of C are $\{\psi_1, \psi_2, \psi_3, \psi_4, \psi_5, \psi_6\}$

$\Delta\psi_2 = \pi$

$\Delta\psi_3 = 2\pi$

$\Delta\psi_4 = 2\pi$

$\Delta\psi_5 = \pi$

C is a proper x-3-wind and proper -y-3*-wind chain but is not a proper -x-k^*-wind or a proper y-k^*-wind chain for any k.*

FIGURE 3.2.2. A Proper k^*-wind chain

Then $\Delta_\alpha(C)$ is a multiple of 2π and the value is independent of the vertex used as start(C) and of the value of α; furthermore, if C is simple then $\Delta(C) = 2\pi$. These properties of polygons are established in **[10, 35]**. Note that if C is closed then $\Delta(i) = \Delta(i \bmod m) + 2k\pi$ for some integer k. Furthermore, if C is closed and simple then $\Delta_\alpha(i+m) = \Delta_\alpha(i) + 2\pi$.

For each inner vertex $v_i \in V(C)$ we denote turn$(v_i) = \Delta_\alpha(i) - \Delta_\alpha(i-1)$. Also, if C is open then turn(v_0) denotes $\Delta_\alpha(0)$ and $\Delta_\alpha(m)$ denotes turn(e_{m-1}, α). For each inner vertex $v_i \in V(C)$, if $\Delta(v_i) < 0$ we call v_i a *reflex vertex* and if $\Delta(v_i) > 0$ we call v_i a *convex vertex*. See Figure 3.2.2. The use of reflex and convex vertices in chains arises from the fact that many of the chains used in this thesis are constructed by deleting an edge from a simple polygon (which is always oriented counterclockwise).

3.2.1. α-Cusps. An important construct with respect to k-wind chains and polygons relates to cusps. Let $C = \{v_0, e_0, \ldots, e_{m-1}, v_m\}$ be a chain, and let α be a polar angle, $0 \leq \alpha < 2\pi$. An α-cusp ψ of C is either an end vertex of C, a vertex v_i such that v_{i-1}, and v_{i+1} lie on the same side of $\alpha(v_i)$, or an inner edge e_i with endpoints v_i, v_{i+1} where $\angle e_i = \pm\alpha$ and where v_{i-1} and v_{i+2} lie on the same side of $\alpha(v_i)$. If ψ consists of a single vertex, it will be called a *vertex α-cusp*; otherwise it will be called an *edge α-cusp.*

If ψ is an end vertex of C then ψ is called an *end α-cusp* of C; otherwise it is called an *inner α-cusp* of C. If C is closed then all of its α-cusps are inner.

Let ψ be an α-cusp of a chain $C = \{v_0, e_0, \ldots, e_{m-1}, v_m\}$. If C is closed, ψ is an edge α-cusp, and v_0, v_{m-1} are both vertices of ψ, then we denote $v^-(\psi) = v_{m-1}$ and $v^+(\psi) = v_0$. In all other cases we denote the first vertex of ψ in a traversal of C by $v^-(\psi)$ and the last vertex of ψ in the traversal by $v^+(\psi)$. Note that if ψ is a vertex cusp then $v^-(\psi) = v^+(\psi) = \psi$. We denote the edge incident to ψ and which precedes (follows) ψ in a traversal of C by $e^-(\psi)$ $(e^+(\psi))$. If t is either $v^-(\psi)$ or $v^+(\psi)$ then we use $\text{cusp}_v(t, \alpha)$ to denote ψ. For any $e_i \in E(C)$ we use $\text{cusps}_e(e_i, \alpha)$ to denote the set of α-cusps of P that have a vertex in common with e_i. Clearly $|\text{cusps}_e(e_i, \alpha)| \leq 2$. When $\alpha = 0$ we often abbreviate $\text{cusp}_v(t, \alpha)$ by $\text{cusp}_v(t)$ and $\text{cusps}_e(e_i, \alpha)$ by $\text{cusps}_e(e_i)$. Let j be such that either $\psi = e_j$ or $\psi = v_j$. Then we use $\iota^+(\psi)$ to denote $j+1$. If $\psi = e_j$ then we use $\iota^-(\psi)$ to denote $j-1$. If, instead, $\psi = v_j$ then we use $\iota^-(\psi)$ to denote j. Note that then $e_{\iota^-(\psi)} = e^-(\psi)$ and $e_{\iota^+(\psi)} = e^+(\psi)$.

Observe that if ψ is an inner α-cusp then there is a unique integer k such that either $\Delta_\alpha(i^-(\psi)) < \alpha + k\pi < \Delta_\alpha(i^+(\psi))$ or $\Delta_\alpha(i^+(\psi)) < \alpha + k\pi < \Delta_\alpha(i^-(\psi))$ depending upon whether ψ is convex or reflex respectively. Then we use $\Delta(\psi)$ to denote $\alpha + k\pi$. Note that if $\psi = e_i$ then $\Delta(\psi) = \Delta_\alpha(i)$. We further use $\angle\psi$ to denote $\angle(\Delta(\psi))$. See Figure 3.2.2.

Note that an α-cusp is also a $-\alpha$-cusp. In this thesis most α-cusps will be either 0-cusps or $\frac{\pi}{2}$-cusps. These will be denoted by x-cusps and y-cusps respectively.

If ψ is an inner or edge α-cusp of a chain C then we say that ψ is a *forward (backward) α-cusp* of C if $\angle\psi = \alpha$ ($\angle\psi = -\alpha$). Furthermore, we say that ψ is a *reflex (convex) α-cusp* if either it is a vertex α-cusp whose vertex is reflex (convex) or it is an edge α-cusp whose endpoints are both reflex (convex) vertices. Thus, for example, y-monotone polygons always have two convex cusps; one of which is forward and one of which is backward.

We denote by $\Psi_\alpha(C)$ the ordered set of α-cusps of C ordered as they are in C. If C is closed and start(C) is the endpoint of some α-cusp $\psi \in \Psi_\alpha(C)$ then ψ is the first element of $\Psi_\alpha(C)$. When $\alpha = 0$, $\pi/2$, π, or $3\pi/2$. we denote $\Psi_\alpha(C)$ by $\Psi_x(C)$, $\Psi_y(C)$, $\Psi_{-x}(C)$ or $\Psi_{-y}(C)$ respectively. We further abbreviate $\Psi_x(C)$ by $\Psi(C)$.

OBSERVATION 3.2.1. *Let ψ_1 and ψ_2 be consecutive inner cusps of a chain or polygon C. Then*

(1) $\Delta(\psi_2) = \Delta(\psi_1)$ if ψ_1 and ψ_2 are neither both convex nor both reflex.

(2) $\Delta(\psi_2) = \Delta(\psi_1) + \pi$ if ψ_1 and ψ_2 are both convex.

(3) $\Delta(\psi_2) = \Delta(\psi_1) - \pi$ if ψ_1 and ψ_2 are both reflex.

Let ψ be a cusp of a polygon P. We say that ψ is a *floor cusp* if for any point $p \in P$ there exists an $\varepsilon > 0$ and point q such that $q_y > p_y$ and every point on the open segment $\overline{pq}$ is interior to P. Otherwise we call ψ a *ceiling cusp*.

OBSERVATION 3.2.2. *Let P be a polygon. The forward cusps of P are floor cusps and the backward cusps of P are ceiling cusps.*

3.2.2. k^*-Wind Chains. Let $C = \{e_0, e_1, e_2, \ldots, e_{m-1}\}$ be an open chain and let α, $0 \leq \alpha < 2\pi$, be a polar angle. Let $\Delta_{\min}(C, \alpha) = \min\{\Delta_\alpha(i) : 0 \leq i < m\}$. We say that C is *α-proper* if $\Delta_{\min}(C, \alpha) \geq 0$.

We now categorize a chain according to the maximum amount that it winds. Let $\Delta_{\max}(C,\alpha) = \max\{\Delta_\alpha(i) : 0 \le i < m\}$. We say that C is an α-k-wind chain, $k \ge 0$, if:

$$(k-1)\pi < \Delta_{\max}(C,\alpha) \le k\pi. \tag{3.2.1}$$

See Figure 3.2.2.

If the final inequality is strict in equation 3.2.1 then we say that C is a strictly α-k-wind chain. We further say that C is a k-wind chain if there exists a polar angle α' such that C is an α'-k-wind chain. Clearly a proper α-1-wind chain is an $\alpha^\perp$-monotone chain and a proper 1-wind chain is a monotone chain. (In this thesis we do not require that a monotone chain be strictly monotone.)

If a chain C is α-j-wind for some j, $0 \le j \le k$, then we say that C is an α-k^*-wind chain. We further say that C is a k^*-wind chain if there exists a polar angle α' such that C is an α'-k^*-wind chain.

In this thesis we are interested in k-wind and k^*-wind chains that are α-proper. We show that a simple polygon can be decomposed into α-proper k^*-wind sub-chains in mathematically and algorithmically useful ways.

3.2.3. k^*-Wind Polygons. Let P be a polygon. We say that P is an α-k-wind (k-wind) polygon if there exists a convex edge e of P such that $P - e$ is an α-k-wind (k-wind) chain. Furthermore, we say that P is a proper α-k-wind (k-wind) polygon if there exists a convex edge e of P such that $P - e$ is a proper α-k-wind (k-wind) chain. If $P - e$ is strictly α-k-wind (k-wind) then we say that P is strictly α-k-wind (k-wind). Note that it is possible for P to be a strictly k-wind polygon even if $k = 1$. See Figure 3.2.3.

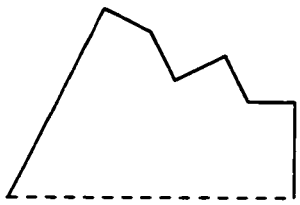


FIGURE 3.2.3. A proper 1-wind polygon

We denote by $P(i)$ the chain $P - e_{i-1}$. In order for $P(i)$ to be a proper α-k-wind chain it must necessarily be a α-proper chain. However, there is no guarantee that $P(i)$ is an α-proper chain for any polar angle α, $0 \leq \alpha < 2\pi$. Nevertheless, we have the following useful result:

LEMMA 3.2.3. *Let $P = \{v_0, e_0, v_1, e_1, v_2, \ldots, e_{n-1}\}$ be a polygon such that $\Delta(P) = 2\pi$. For any polar angle α, $0 \leq \alpha < 2\pi$, there exists a unique ι, $0 \leq \iota \leq n-1$, such that $P(\iota)$ is an α-proper chain and either $\psi = e_{(\iota-1)}$ or $\psi = v_\iota$ is a convex forward α-cusp of P.*

PROOF. Let $\{\ldots, h_k, w_k, h_{k+1}, w_{k+1}, h_{k+2}, w_{k+2}, \ldots\}$ be the edges of an infinite open rectilinear chain C such that the coordinates of the endpoints of (horizontal) edge h_k are $(k, \Delta_\alpha(C,k))$ and $(k+1, \Delta_\alpha(C,k))$. Recall that $\Delta(P) = 2\pi$ and thus $\Delta_\alpha(P, k+n) = \Delta_\alpha(P,k) + 2\pi$. Now $|E(P)|$ is finite, turn(v_k) is finite, and $\Delta_\alpha(C, k+n) = \Delta_\alpha(C,k) + 2\pi$ for $-\infty < k < \infty$, so for any horizontal line there must be a last horizontal edge of C that is below this line. See Figure 3.2.4. Now, for some integer p, h_p is the last edge of C below the line $y = \alpha$. We will show that the unique value for ι that satisfies this lemma is the one such that $e_p = e^-(\psi)$; note that p and α together uniquely determine ψ which in turn uniquely determines ι. Now, with this choice for ι, we know that $\Delta(P, \iota-1) \leq \alpha$ and $P(\iota)$ is α-proper. Furthermore, either $\Delta_\alpha(P, \iota-1) = \alpha$ and then $\psi = e_{\iota-1}$ or else $\Delta_\alpha(\iota-1) < \alpha$ and then $\psi = v_\iota$. It remains to be shown that ι satisfies the uniqueness requirement of this lemma.

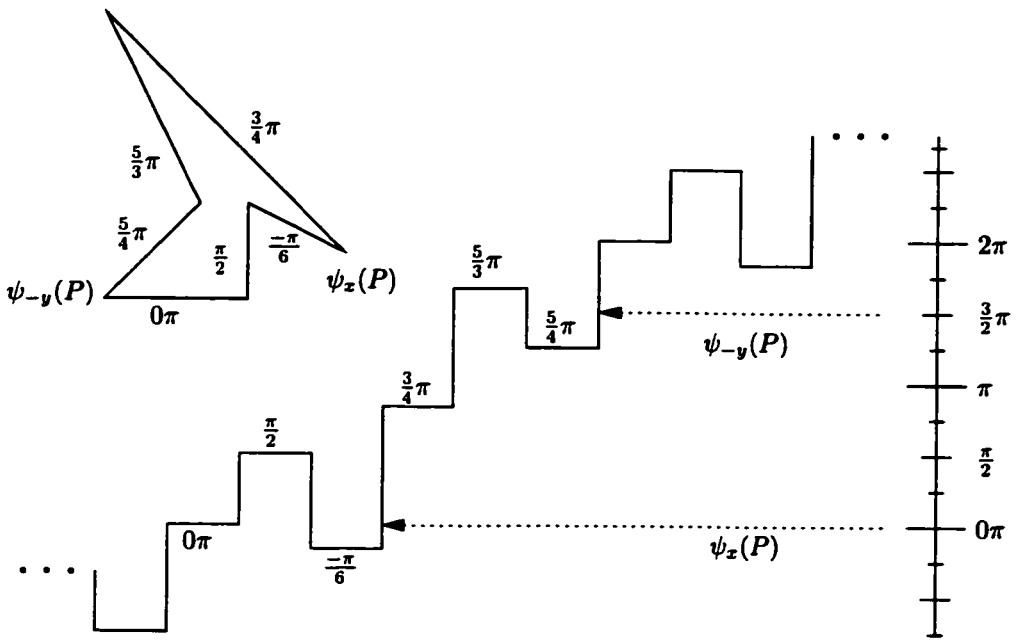


FIGURE 3.2.4. For any polar angle α there is a unique $\psi_\alpha(P)$

Assume otherwise that some j, where $\iota - n < j < \iota$, satisfies all the requirements of this lemma except uniqueness. Then either $\psi' = e_{j-1}$ or $\psi' = v_j$ must be a convex forward α-cusp of P. Let $e_l = e^-(\psi')$ and $e_r = e^+(\psi')$. Now $P(\iota - n)$ is also α-proper and P cannot contain consecutive α-cusps so $j - 1 > \iota - n$ and $l > \iota - n$. Let b and b' be such that $\Delta(\psi) = 2b\pi + \alpha$ and $\Delta(\psi') = 2b'\pi + \alpha$. Now, $P(j)$ is α-proper so $b > b'$ since $\Delta(P,p) < \Delta(\psi) = 2b\pi + \alpha$ and $j < p < j + n$. Also, $\Delta(P,l) < 2b'\pi + \alpha = \Delta(\psi') < \Delta(P,r)$. Therefore $\Delta(P, l+n) < 2(b'+1)\pi + \alpha \leq 2b\pi + \alpha = \Delta(\psi)$. But $i < l + n < \iota + n$ so this contradicts that $P(\iota)$ is α-proper. □

Note that if $\Delta(P) \geq 4\pi$ then the value ι (where ι is the value determined by α and defined in Lemma 3.2.3) still exists but may not be unique. If, instead, the restriction that ψ be convex forward is removed then again uniqueness is not guaranteed.

We denote by $P(\alpha)$ the chain $P(\iota)$ where ι is as defined in Lemma 3.2.3. We further denote $\psi_\alpha(P) = \text{cusp}_v(v_\iota)$. Note that $v_\iota = v^+(\psi_\alpha(P))$. When $\alpha = 0, \pi/2,$

π, or $3\pi/2$ we denote $\psi_\alpha(P)$ by $\psi_x(P)$, $\psi_y(P)$, $\psi_{-x}(P)$, or $\psi_{-y}(P)$ respectively. From Lemma 3.2.3 we get the following Corollary:

COROLLARY 3.2.4. *Let α, $0 < \alpha \leq 2\pi$, be a polar angle and let there be a simple polygon $P = \{e_0, e_1, e_2, \ldots, e_{n-1}\}$ such that $\psi_\alpha(P) = e_0$. Then $\pi < \Delta(\iota^-(\psi_\alpha(P))) < 2\pi$.*

If a polygon P is α-j-wind for some j, $1 \leq j \leq k$, then we may say that P is α-k^*-wind. When $\alpha = 0$, $\pi/2$, π, or $3\pi/2$ we denote α-k-wind by x-k-wind, y-k-wind, $-x$-k-wind, or $-y$-k-wind respectively. A corresponding notation applies to the term α-k^*-wind. The rectilinear polar angles are the only four directions we will use for α-k-wind chains. If a polygon or chain C is either α-k-wind or $-\alpha$-k-wind then we say that C is $\pm\alpha$-k-wind.

Henceforth, for any simple polygon P, we assume that start(P) is an endpoint of a convex forward α-cusp of P for some polar angle α, $0 \leq \alpha < 2\pi$. This restriction does not prevent the selection of the vertex of P with minimum y-coordinate for start(P) as is commonly done and which we will sometimes do. It also does not restrict us from selecting a vertex of $\psi_\alpha(P)$ as start(P) which we will usually do when dealing with proper α-k^*-wind polygons.

We make the following useful observation.

PROPOSITION 3.2.5. *If start(P) $\in V(\psi_\alpha(P))$ then P has the property that any subchain of P starting at start(P) has at least as many convex α-cusps as reflex α-cusps.*

PROOF. Assume the conditions of the observation and let ψ_1 and ψ_2 be two consecutive α-cusps of P and assume $\psi_1 \neq \psi_\alpha(P)$. By Observation 3.2.1, if ψ_2 is a convex α-cusp then $\Delta(\psi_2) = \Delta(\psi_1) + \pi$; otherwise if ψ_2 is a reflex α-cusp then $\Delta(\psi_2) = \Delta(\psi_1) - \pi$. Now the proposition follows since $\Delta(\psi_2) \geq 0$ by the definitions of $\Delta(\psi_2)$ and proper k^*-wind chain. □

OBSERVATION 3.2.6. *Any simple polygon has an even number of cusps and has two more convex cusps than reflex cusps.*

The following result will prove useful.

LEMMA 3.2.7. *In any simple polygon, the number of convex forward (convex backward) cusps exceeds the number of reflex forward (reflex backward) cusps by one.*

PROOF. To each cusp ψ of P we associate a pair (x, y) as follows: If ψ is a convex (reflex) cusp then $x = c$ ($x = r$). If ψ is a forward (backward) cusp then $y = f$ ($y = b$). The set of cusps of P determines a sequence of pairs according to their type. For example, a y-monotone polygon determines the sequence $(c, f)(c, b)$. Now, by Observation 3.2.1, if ψ_i and ψ_j are consecutive cusps of P then their pairs differ in exactly one of their coordinates; e.g. if ψ_i generates the pair (c, f) then ψ_j can only generate the pairs (c, b) and (r, f). We call a pair of the form (x, y) a symbol pair. We say that a symbol pair is convex, reflex, forward, or backward if it corresponds to a pair generated by a cusp that is convex, reflex, forward, or backward respectively.

We call any sequence of the symbol pairs (c, f), (c, b), (r, f), and (r, b) balanced if it has:

1) two more convex symbol pairs than reflex symbol pairs and

2) adjacent symbol pairs of the following eight forms only:

$$(c, f)(c, b),\ (c, f)(r, f),\ (c, b)(c, f),\ (c, b)(f, b),$$

$$(r, f)(r, b),\ (r, f)(c, f),\ (r, b)(r, f),\ (r, b)(r, b).$$

Note that these are exactly the adjacent symbol pairs that can be generated by adjacent cusps of a simple polygon. From Observations 3.2.1 and 3.2.6, any symbol pair sequence constructed from the cusps traversed in the traversal of a simple polygon is balanced. We now prove this lemma by proving the more general result that all

balanced sequences have one more (c, f) symbol pair than (r, f) symbol pair and also one more (c, b) symbol pair than (r, b) symbol pair.

Suppose that $S = S_1 r_f c_f S_2$ is a balanced pair symbol sequence for some subsequences S_1, S_2. Then it is easily checked that $S_1 S_2$ is also a balanced sequence. Similarly, it is easily checked that any two consecutive symbol pairs, exactly one of which is a reflex symbol pair, can be removed from a balanced sequence and the result is still a balanced sequence. Thus any balanced sequence can be reduced to a balanced sequence containing no adjacent convex/reflex symbol pairs. But any balanced sequence that contains a reflex symbol pair also contains consecutive convex/reflex symbol pairs since any balanced sequence has two more convex symbol pairs than reflex symbol pairs. Thus the only possible combinations of balanced sequences where all consecutive convex/reflex symbol pairs have been removed are $(c, f)(c, b)$ and $(c, b)(c, f)$. Clearly these sequences have one more (c, f) symbol pair than (r, f) symbol pair and also one more (c, b) symbol pair than (r, b) symbol pair. Thus, so do all balanced sequences since all the removed consecutive symbol pairs have equal numbers of convex and reflex symbol pairs. □

3.2.4. Y-Monotone Subpolygon Partitions of Polygons. The partition of a polygon into y-monotone subpolygons is the starting point of our algorithm to construct a proper $\pm x$-2*-wind subpolygon partitions of polygons. In this section we provide some necessary notation relating to y-monotone subpolygon partitions.

We remind the reader that, for any chord γ of a polygon P with endpoints p_1 and p_2 such that p_1 is reached before p_2 in a traversal of P, we assume that γ is directed from p_1 to p_2; see Section 2.2.

We say that γ is an α-chord of P if $\angle\gamma = \pm\alpha$. We say that γ is a forward α-chord if $\angle\gamma = \alpha$ and a backward α-chord if $\angle\gamma = -\alpha$. For any angle α we denote $\dot{-}\angle\alpha = (\angle\alpha + \pi) \bmod 2\pi$.

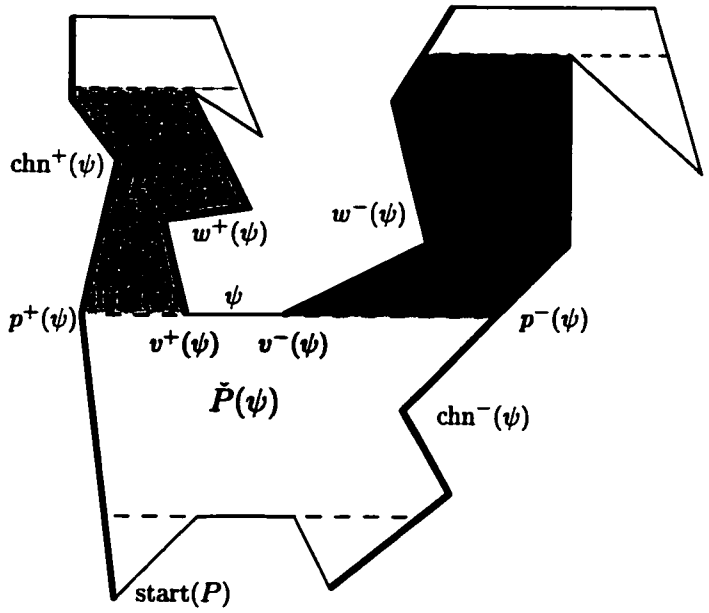


FIGURE 3.2.5. The neighbor objects of a cusp.

If ψ is a reflex α-cusp of P then we denote by $p^-(\psi)$ ($p^+(\psi)$) the first point of P struck by the ray with direction $\dot{-}\angle\psi$ ($\angle\psi$) and starting at $v^-(\psi)$ ($v^+(\psi)$). We denote by $\text{lid}^-(\psi)$ the α-chord of P with endpoints $v^-(\psi)$ and $p^-(\psi)$ and by $\text{lid}^+(\psi)$ the α-chord of P with endpoints $v^+(\psi)$ and $p^+(\psi)$. We further denote the set $\{\text{lid}^-(\psi), \text{lid}^+(\psi)\}$ by $\text{lids}(\psi)$. See Figure 3.2.5.

The set of α-cusps of P divides the edges of P not belonging to any α-cusp of P into a set of chains joining consecutive α-cusps. Let M be this set of chains. The elements of M alternate between being a proper α-1-wind chain and a proper $-\alpha$-1-wind chain as we traverse P.

We denote by $\text{chn}^-(\psi)$ the chain of M containing point $p^-(\psi)$ and by $\text{chn}^+(\psi)$ the chain of M containing $p^+(\psi)$. Let ψ' be the cusp following ψ in a traversal of P. Then we denote the element of M with starting point $v^+(\psi)$ and ending point $v^-(\psi')$ as $w^+(\psi)$ and also as $w^-(\psi')$ (w stands for wall). See Figure 3.2.5.

Assume now that start(P) is a vertex of a convex forward α-cusp of P. We use $\mathfrak{L}_\alpha(P)$ to denote:

$$\{\text{lid}^-(\psi), \text{lid}^+(\psi) : \psi \text{ is a reflex } \alpha\text{-cusp of } P\}.$$

When $\alpha = 0$ or $\pi/2$ we denote $\mathfrak{L}_\alpha(P)$ by $\mathfrak{L}_x(P)$ or $\mathfrak{L}_y(P)$ respectively. Furthermore, we abbreviate $\mathfrak{L}_x(P)$ to $\mathfrak{L}(P)$. We note that $\mathfrak{L}_\alpha(P) = \mathfrak{L}_{-\alpha}(P)$.

In Section 3.4 we will show that a subset of $\mathfrak{L}_\alpha(P)$ partitions P into proper $\pm\alpha$-2^*-wind subpolygons. We show now how to determine this subset.

Let $\pi_\alpha(P)$ denote the partitioning of P induced by $\mathfrak{L}_\alpha(P)$. When $\alpha = 0$ or $\pi/2$ we denote $\pi_\alpha(P)$ by $\pi_x(P)$ or $\pi_y(P)$ respectively. Furthermore, we abbreviate $\pi_x(P)$ to $\pi(P)$. We note that $\pi_\alpha(P) = \pi_{-\alpha}(P)$.

Clearly each element of $\pi_\alpha(P)$ is an $\alpha^\perp$-monotone polygon. For each polygon $Q \in \pi_\alpha(P)$ let Q_γ be the edges of Q that are chords of P. We make the following observations.

OBSERVATION 3.2.8. *For each polygon $Q \in \pi_\alpha(P)$ at most one element of Q_γ has a clockwise orientation. Furthermore, this element exists if and only if it is the door of Q.*

If Q contains start(P) we say that door(Q) is the α-cusp of P containing start(P). Doors have the following important properties.

OBSERVATION 3.2.9. *Let q be a point on Q and p be a shortest path contained in P from q to start(P). Then p intersects door(Q).*

OBSERVATION 3.2.10. *Let γ be any α-chord of P such that γ is also either an edge of Q or an α-chord of Q. Then γ has the same orientation in Q as it has in P if and only if $\gamma \neq$ door(Q).*

Let γ be a chord in $\mathfrak{L}_\alpha(P)$ and let P_1 and P_2 be the elements of $\pi_\alpha(P)$ containing γ. If $\gamma = \text{door}(P_2)$ then P_2 is called the successor of γ and P_1 its predecessor. We denote the predecessor and successor of γ by $\text{pred}(\gamma)$ and $\text{succ}(\gamma)$ respectively.

For any convex α-cusp $\psi \in \Psi_\alpha(P)$ we denote by $\check{P}(\psi)$ the polygon of $\pi_\alpha(P)$ that has ψ on its boundary. If, instead, ψ is reflex then we use $\check{P}(\psi)$ to denote the unique polygon of $\pi_\alpha(P)$ such that both $\text{lid}^-(\psi)$ and $\text{lid}^+(\psi)$ are on its boundary. If ψ is reflex then we also use $P^-(\psi)$ to denote the polygon of $\pi_\alpha(P)$ other than $\check{P}(\psi)$ that is incident to $\text{lid}^-(\psi)$ and use $P^+(\psi)$ to denote the polygon of $\pi_\alpha(P)$ other than $\check{P}(\psi)$ that is incident to $\text{lid}^+(\psi)$. See Figure 3.2.5.

The elements of $\mathfrak{L}(P)$ can be colored with three colors, red, blue, and green such that the red color class partitions P into proper $\pm x$-2^*-wind polygons; this will be shown in Section 3.4. This coloring of $\mathfrak{L}(P)$ is obtained as follows. For each chord $\gamma \in \mathfrak{L}(P)$ let $Q_\gamma = \text{pred}(\gamma)$ and $d_\gamma = \text{door}(Q)$:

RULE 3.2.11. *If d_γ and γ are collinear then color γ red. Also color d_γ green even if it was already colored blue. However, if it was already colored red then leave it unchanged.*

RULE 3.2.12. *If d and γ are not collinear and for each $\psi_i \in \text{cusps}(\gamma)$ we have that $\check{P}(\psi_i) = Q_\gamma$ then color γ blue unless γ was already colored green.*

It is simple and efficient to apply these rules to the chords in $\mathfrak{L}(P)$. The key is finding the doors of the elements of $\pi(P)$. Let Q be an element of $\pi(P)$ and let S be the subset of the chords of $\mathfrak{L}(P)$ that are edges of Q. Then $\text{door}(Q)$ is the one and only element of S whose orientation is clockwise on S. Alternatively, among the elements of S, $\text{door}(Q)$ is the one whose endpoints are the first and last ones crossed during a traversal of P. The first method has the advantage in that it works when we deal with polygons with holes, where a subpolygon can have multiple doors; see

Chapter 7. Either method can be applied to find the doors of the elements of $\pi(P)$ in linear time in the size of $|E(P)| + |\mathfrak{L}(P)| = O(|P|)$.

The remaining steps needed to apply Rules 3.2.11 and 3.2.12 are straightforward. Once the doors have been determined, these rules can be applied to a chord in $O(1)$ time per chord. Thus linear time is required in total to color the elements of $\mathfrak{L}(P)$.

Note that if P has no collinear edges (for example if we assume general position of the vertices of P) then $|\text{cusps}(\gamma)| = 1$ for all $\gamma \in \mathfrak{L}(P)$. Then we can use a simpler algorithm to apply the same color scheme to the elements of $\mathfrak{L}(P)$. Let ψ be any reflex x-cusp of P and $d = \text{door}(\check{P}(\psi))$. We first make the following observations:

OBSERVATION 3.2.13. *$d \in lids(\psi)$ if and only if $\angle lid^-(\psi) = \dot{-}\angle lid^+(\psi)$.*

OBSERVATION 3.2.14. *Let γ and γ' be the lids of ψ. Now if $d = \gamma$ then $\angle d = \dot{-}\angle\psi = \dot{-}\angle r$.*

Note that this observation applies even if P has collinear edges. We will use this fact later.

Now let ψ be any α-cusp of P. Then we color $\text{lid}^-(\psi)$ and $\text{lid}^+(\psi)$ according to the following rules:

RULE 3.2.15. *If $\angle lid^-(\psi) \neq \angle lid^+(\psi)$ then by Observation 3.2.13 one of them is the door of $\check{P}(\psi)$. We color this chord green (using Observation 3.2.14) and color the remaining chord red.*

RULE 3.2.16. *If $\angle lid^-(\psi) = \angle lid^+(\psi)$ then we color both of these chords blue.*

Red chords are the key we will use to partition a polygon into proper $\pm x$-2^*-wind subpolygons. See, for example, Figure 3.2.6. First, however, we must study proper 2^*-wind polygons themselves. This is the topic of the next section.

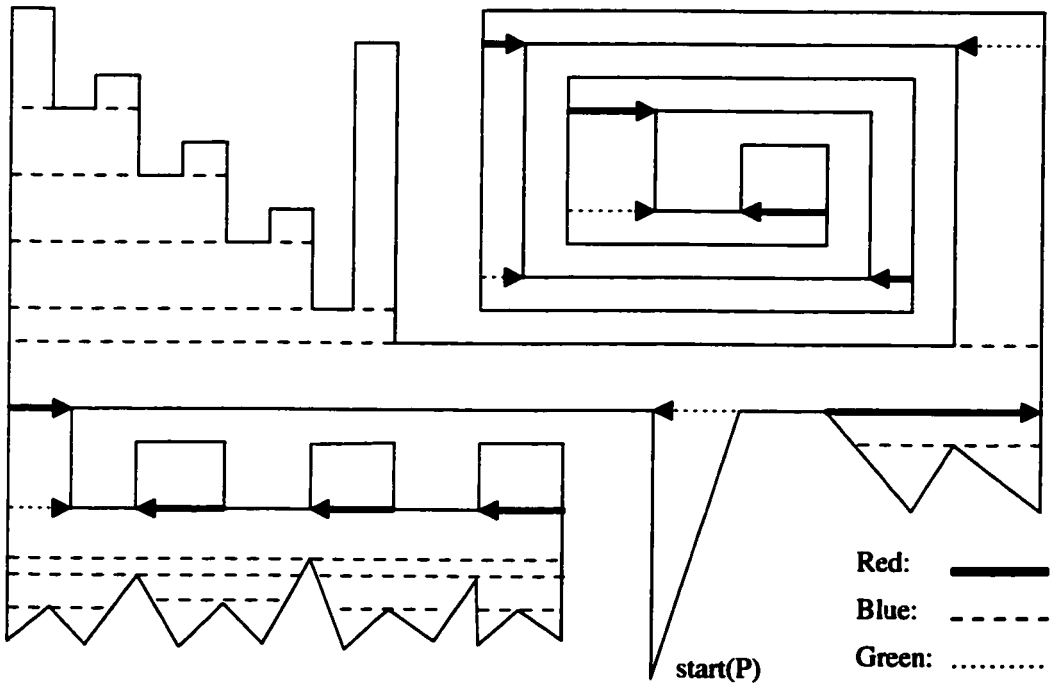


FIGURE 3.2.6. The proper $\pm x$-2*-wind subpolygon partitioning of P by $\mathfrak{L}_{\text{red}}(P)$

3.3. Proper 2*-Wind Chains and Polygons

In this section we focus on the properties of proper 2*-wind chains and polygons. These properties will help us in two ways. First, they will aid us in developing algorithms specifically for proper 2*-wind polygons. Second, they will aid us in the partitioning of polygons into proper 2*-wind subpolygons. The combination of these results will be used in Chapters Three and Four to solve a number of problems.

For this section we will deal, among other things, with α-cusps, proper α-2*-wind chains, proper α-2*-wind polygons, and $\alpha^{\perp}$-monotone chains in which α is always 0. The results that we obtain are easily seen to apply when α is any other direction.

LEMMA 3.3.1. *Let C be a chain such that $\Delta(0) > 0$. Then C is proper x-2*-wind if and only if it has no forward cusps.*

PROOF. $\Rightarrow$ First note that, since C is x-2*-wind, for every edge e_i of C we have that $0 \leq \Delta(i) \leq 2\pi$. Now assume that C has a forward cusp ψ. Then $\Delta(\psi) = 2j\pi$ for some integer j. If ψ is convex then $\Delta(\iota^-(\psi)) < 2j\pi < \iota^+(\psi)$ so either $\Delta(\iota^-(\psi)) < 0$ or $2\pi < \Delta(\iota^+(\psi))$. Either case contradicts that C is a proper x-2*-wind chain.

If, instead, ψ is reflex then $\Delta(\iota^-(\psi)) > 2j\pi > \iota^+(\psi)$ so either $\Delta(\iota^-(\psi)) > 2\pi$ or $0 > \Delta(\iota^+(\psi))$. Either case contradicts that C is a proper x-2*-wind chain.

$\Leftarrow$ Assume that C is a not a proper x-2*-wind chain. If C is proper then C has a first edge e_i during a forward traversal of C such that $\Delta(i) > 2\pi$. Then there exists a cusp ψ' of C such that $e^+(\psi') = e_i$. Furthermore $\Delta(\psi') = 2\pi$. But then ψ' is a forward cusp, contradicting that C has no forward cusps.

If C is not proper then C has a first edge e_i during a forward traversal of C such that $\Delta(i) < 0$. Then there exists a cusp $\overline{\psi}$ of C such that $e^+(\overline{\psi}) = e_i$. Furthermore $\Delta(\overline{\psi}) = 0$. But then $\overline{\psi}$ is a forward cusp contradicting that C has no forward cusps.

□

Embedded in the proof of Lemma 3.3.1 is the following result:

COROLLARY 3.3.2. *Let C be a chain that is not proper. Then C has a forward reflex cusp.*

LEMMA 3.3.3. *The inner cusps of a proper x-2*-wind chain are alternately convex backward cusps and reflex backward cusps with the first inner cusp being convex.*

PROOF. Let C be a proper x-2*-wind chain and let the inner cusps of C in order be $\{\psi_1, \psi_2, \ldots, \psi_{2k}\}$. Then ψ_1 must be convex backward since $w^-(\psi_1)$ is x-1-wind and since ψ_1 cannot be reflex forward by Lemma 3.3.1. Therefore $\Delta(\psi_1) = \pi$.

Now ψ_2 must be reflex backward since if it is convex forward then $\Delta(\psi_2) = 2\pi$ by Observation 3.2.1 and then $\Delta(\iota^+(\psi_2)) > 2\pi$ contradicting that C is proper x-2*-wind. Therefore $\Delta(\psi_2) = \pi$ by Observation 3.2.1.

Similarly we can prove that the remaining cusps alternate between convex and reflex backwards. □

OBSERVATION 3.3.4. *Let P be a polygon and let $\Psi(P) = \{\psi_1, \psi_2, \ldots, \psi_{2k}\}$. If P is proper x-2-wind then ψ_1 is a convex forward cusp and cusps ψ_2 and ψ_{2k} are convex backward cusps.*

LEMMA 3.3.5. *A simple polygon P is proper x-2*-wind if and only if it has exactly one forward cusp. Furthermore, this cusp is convex.*

PROOF. By Lemma 3.2.3, for some unique index i, $P(i)$ is 0π-proper and furthermore $\psi_{0\pi}(P)$ is a forward convex cusp of P. Now P has exactly one forward cusp $\iff$ $P(i)$ has no forward cusps $\iff$ $P(i)$ is a proper x-2*-wind chain (by Lemma 3.3.1) $\iff$ P is a proper x-2*-wind polygon (by the definition of α-k*-wind polygon and Lemma 3.2.3). □

LEMMA 3.3.6. *A polygon P is proper x-2*-wind if and only if it contains no forward reflex cusp.*

PROOF. By Lemma 3.2.3, every polygon has at least one forward convex cusp. Also, by Lemma 3.2.7, P has a second forward cusp if and only if it has a forward reflex cusp. Therefore, by Lemma 3.3.5, P is x-2*-wind if and only if it does not also have a forward reflex cusp. □

See Figure 3.3.1.

PROPOSITION 3.3.7. *A polygon P contains no forward reflex cusp if and only if, for any point q on P, there exists a proper x-1*-wind shortest path contained in P from $\psi(P)$ to q.*

PROOF. Relabel the vertices of P, if necessary, such that $\text{start}(P) = v^+(\psi(P))$.

Assume first that P has a forward reflex cusp ψ. Now either $P^+(\psi)$ or $P^-(\psi)$ cannot contain $\text{start}(P)$; assume wlog that $P^+(\psi)$ does not contain $\text{start}(P)$. Let q

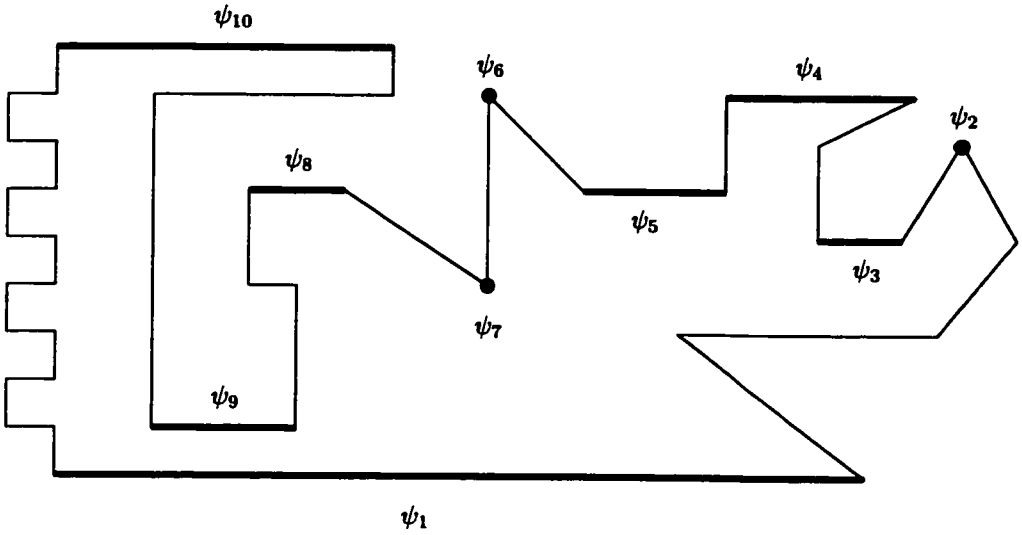


FIGURE 3.3.1. A proper x-2*-wind polygon

be a point interior to $P^+(\psi)$. Then any shortest path from start(P) to q must cross $\text{lid}^+(\psi)$ and at the point that it does so this path is directed downwards. Therefore this path is not x-1-wind.

Now assume that there exists a shortest path p contained in P from $\psi(P)$ to some point on P that is not proper x-1-wind. Let ψ_p be the highest cusp of p. Then ψ_p is coincident with a reflex cusp ψ_r of P. Now immediately before and after overlapping ψ_r it must be that p was in $P^+(\psi)$, and then $P^-(\psi)$ or conversely. Since, at those points, p was below ψ_p it must be that ψ_r is a floor reflex cusp. Now, by Observation 3.2.2, floor reflex cusps are also forward reflex cusps. □

From Proposition 3.3.7 and Lemma 3.3.6 the next result is immediate.

LEMMA 3.3.8. *A simple polygon P is proper x-2*-wind if and only if, for any point q on the boundary of P, there exists a proper x-1-wind shortest path contained in P from $\psi(P)$ to q.*

In general we say that an α-2*-wind polygon P is α-2*-wind from $\psi_\alpha(P)$ meaning that for any point q on the boundary of P there exists a proper α-1*-wind shortest path contained in P from $\psi_\alpha(P)$ to q.

LEMMA 3.3.9. *Let P be a simple polygon. If P has a forward reflex cusp then there exists a y-monotone subpolygon M of P which is the boundary of the union of the interiors of a subset of the elements of $\pi(P)$ and whose $-x$-cusp is a lid of a forward reflex cusp of P, and whose x-cusp is $\psi(P)$.*

PROOF. Let T be the dual tree of $\pi(P)$ such that the root node of T corresponds to the element of $\pi(P)$ that has $\psi(P)$ on its boundary. Now, since P has a forward reflex cusp then it is not x-2*-wind by Lemma 3.3.6. Therefore $\mathfrak{L}_{\text{green}}(P) \neq \emptyset$ by Lemma 3.4.2 and Rule 3.2.11. Let p be a shortest directed path interior to P, from a point of $\psi(P)$, that does not cross any element of $\mathfrak{L}_{\text{green}}(P)$, and ends at a point of some element g of $\mathfrak{L}_{\text{green}}(P)$. Let M be the smallest subpolygon of P that contains all the elements of $\pi(P)$ that are intersected by p. Now p cannot cross any elements of $\mathfrak{L}_{\text{red}}(P)$ by Rule 3.2.11. Therefore p, and thus M, is contained in some element of Q of $\pi_{\text{red}}(P)$. Furthermore, the elements of $\mathfrak{L}(P)$ intersected by p are all lids of reflex backward cusps of Q, and thus of P, by Lemma 3.3.6. Thus M must be y-monotone. Now the $-x$-cusp of M is g and thus $\angle g = \pi$. Let ψ be a cusp whose lid is g. Then $\angle\psi = \dot{-}\angle g$ by Observation 3.2.14 and Rule 3.2.11. Therefore ψ is a forward reflex cusp and so g is a lid of a forward reflex cusp of P. □

PROPOSITION 3.3.10. *Let P be a proper α-2*-wind polygon such that start$(P) = v^+(\psi_\alpha(P))$. Let γ be an α-chord of P. Then $\angle\gamma = -\alpha$.*

PROOF. Clearly γ is a proper α-convex cusp of $P_{\overline{\gamma}}$ and furthermore, by Lemma 3.3.8, $P_{\overline{\gamma}}$ is α-2*-wind. Thus, by Lemma 3.3.3, γ is an α-backward cusp of $P_{\overline{\gamma}} - \psi_\alpha(P)$ and it therefore has direction $\dot{-}\alpha$. □

We summarize the key results of this section with the following theorem.

THEOREM 3.3.11. *Let P be a proper α-2^*-wind polygon. Then P is proper α-2^*-wind from $\psi_\alpha(P)$. Furthermore, $\psi_\alpha(P)$ is the only α-forward cusp of P. Finally, any α-chord of P has direction $\dot{-}\angle\psi_\alpha(P)$.*

PROOF. Let $\psi = \psi_\alpha(P)$. Then P is proper α-2^*-wind from ψ by Lemma 3.3.8. Also, ψ has direction α but all other cusps of P have direction $-\alpha$ by Lemma 3.3.3. Finally, any α-chord γ of P has direction $-\alpha$ by Proposition 3.3.10. □

3.4. Decomposing Polygons into Proper $\pm x$-2^*-Wind Subpolygons

In this section we show how to decompose any simple polygon P into proper $\pm x$-2^*-wind subpolygons in linear time. Let $\mathfrak{L}_{\text{red}}(P)$ be the red chords of $\mathfrak{L}(P)$ and $\pi_{\text{red}}(P)$ be the partition of P into subpolygons induced by the chords $\mathfrak{L}_{\text{red}}(P)$. See Figure 3.2.6. In particular we show that $\pi_{\text{red}}(P)$ is such a decomposition into proper $\pm x$-2^*-wind subpolygons. We further show that any straight line interior to P intersects the interiors of at most two elements of $\pi_{\text{red}}(P)$; in fact any y-monotone chain interior to P will intersect the interiors of at most two elements of $\pi_{\text{red}}(P)$. It then follows that any curve decomposable into k y-monotone components and contained in the polygon can intersect the interiors of at most $k+1$ elements of $\pi_{\text{red}}(P)$. Thus, for example, a circular arc contained in P can intersect at most four proper $\pm x$-2^*-wind subpolygons of $\pi_{\text{red}}(P)$. Another useful property of this proper $\pm x$-2^*-wind decomposition is that any convex set contained in the polygon can intersect at most two of the subpolygons.

Our algorithm for constructing $\pi_{\text{red}}(P)$ also has the advantage of being very simple given a horizontal trapezoidization of the polygon.

Proving that $\pi_{\text{red}}(P)$ has these properties, however, requires some effort. We begin by noting that the set of chords $\mathfrak{L}(P)$, is partitioned into three groups according to their color; blue, red, or green. We now use this partitioning.

We use $\mathfrak{L}_{\text{blue}}(P)$ to denote the blue chords of $\mathfrak{L}(P)$ and $\mathfrak{L}_{\text{green}}(P)$ to denote the green chords of $\mathfrak{L}(P)$.

LEMMA 3.4.1. *Let P be a simple polygon such that $|\pi_{\text{red}}(P)| > 1$. Let Q be an element of $\pi_{\text{red}}(P)$ such that $E(Q) \cap \mathfrak{L}_{\text{red}}(P) - \{door(Q)\} \neq \emptyset$. Let r be an element of $E(Q) \cap \mathfrak{L}_{\text{red}}(P) - \{door(Q)\}$ and let g be the element of $\mathfrak{L}_{\text{green}}(P)$ that determined the color of r. Then g is a chord of Q.*

PROOF. First, $g \notin \mathfrak{L}_{\text{red}}(P)$ so g is a chord of some subpolygon $R \in \pi_{\text{red}}(P)$. Furthermore, by Rule 3.2.11, g and r are edges of the same y-monotone polygon of $\pi(P)$ and clearly this polygon is also an element of $\pi(R)$. Therefore g is a chord of R. Now if $Q \neq R$ then r is the door of Q since r is not the door of R by Rule 3.2.11. But this contradicts the definition of Q so $R = Q$. □

LEMMA 3.4.2. *Let P be a simple polygon such that $start(P) = v^+(\psi_\alpha(P))$ for some angle α. Then P is proper $\pm\alpha$-2^*-wind if and only if all the chords of $\mathfrak{L}_\alpha(P)$ are colored blue.*

PROOF. Assume P is proper $\pm\alpha$-2^*-wind but some chord $r \in \mathfrak{L}_\alpha(P)$ is not blue. By Rules 3.2.11 and 3.2.12, the existence of a green chord immediately implies the existence of a red chord as well. Let r be this red chord. Let $\psi \in \text{cusps}(r)$ and $g = \text{door}(\check{P}(\psi))$ be such that r and g are collinear; note that ψ and g exist by Rule 3.2.11. Let $Q \in \pi_\alpha(P)$ be such that $\text{door}(Q) = r$; let q be a point on $Q - r$; and p be the shortest path contained in P from q to $\psi_\alpha(P)$. Then p must intersect α-chords r and g, which are collinear, so p cannot be $\alpha^\perp$-monotone. But then P is not proper $\pm\alpha$-2^*-wind by Lemma 3.3.8. Therefore r cannot exist and all chords of $\mathfrak{L}_\alpha(P)$ are colored blue.

Now assume that P is not proper $\pm\alpha$-2^*-wind. Then, by Lemma 3.3.8, there exists a point u on P such that the shortest path p' contained in P from $\psi_\alpha(P)$ to u is not $\alpha^\perp$-monotone. Let ψ' be an α-cusp of p'. Then, for some polygon $Q' \in \pi_\alpha(P)$ we have

that $\psi' \in Q$ since otherwise p' is not a shortest path. Furthermore, by Observation 3.2.9, p' must intersect $d = \text{door}(Q')$. Now clearly ψ' must be collinear with and completely cover some cusp $\overline{\psi}$ of P such that $\check{P}(\overline{\psi}) = Q'$ and $d \in \text{lids}(\overline{\psi})$. Then, by Rule 3.2.11, g is green (and $\mathfrak{L}_{\text{red}}(P) \neq \emptyset$). □

LEMMA 3.4.3. *Let P be a simple polygon and Q be an element of $\pi_{\text{red}}(P)$. Let d be the edge of Q containing door(Q) and α be the direction of d on Q; that is $a = \dot{-}\angle d$ where d is a chord of P. Then*

(1) *$Q - d$ is an α-proper chain.*

(2) *Q is proper α-2*-wind from d.*

PROOF. Assume first, contrary to hypothesis, that $d \neq \psi_\alpha(Q)$. Then, since d is an α-forward convex cusp of Q and, by Lemma 3.2.3, it must be that $Q - d$ is not proper and thus $Q - d$ contains an α-forward reflex cusp by Corollary 3.3.2. This cusp is also an α-forward reflex cusp of Q. Thus, by Corollary 3.3.9, there exists a reflex α-forward cusp ψ of Q and a y-monotone subpolygon M of Q such that the y-cusps of M are $\psi_\alpha(Q)$ and a lid of ψ. Let γ_M be the lid of ψ that is a cusp of M and let γ_r be the remaining lid of ψ. We note that $\text{start}(Q) = v^+(\psi_\alpha(Q))$. Thus, by the construction of $\pi(Q)$ and Rules 3.2.11 and 3.2.12, we have that $\gamma_M \in \pi_{\text{green}}(Q)$ and thus $\gamma_r \in \pi_{\text{red}}(Q)$. But, by Observation 3.2.10, the chords of Q have the same orientation as they have by virtue of being chords of P and thus $\pi_{\text{green}}(Q) \subseteq \pi_{\text{green}}(P)$ by Rule 3.2.11. Therefore $\gamma_r \in \pi_{\text{red}}(P)$ which is impossible since γ_r is an edge of Q. Therefore $d = \psi_\alpha(P)$.

Now assume, contrary to hypothesis, that Q is not α-2*-wind even though $d = \psi_\alpha(Q)$. Then, by Corollary 3.3.9, there exists a reflex α-forward cusp ψ' of Q and a y-monotone subpolygon M' of Q such that the y-cusps of M' are $\psi_\alpha(Q)$ and a lid of ψ'. Now the same arguments as in the previous paragraph similarly lead to a contradiction. Therefore Q is α-2*-wind from d. □

LEMMA 3.4.4. *Let P be a simple polygon and let Q be an element of $\pi_{red}(P)$. Let d be the edge of Q containing door(Q) and let α be the direction of d on Q. Then Q is a maximal proper α-2^*-wind subpolygon of P from d.*

PROOF. First, Q is proper α-2^*-wind from d by Lemma 3.4.3. Assume that there exists a subpolygon R of P such that R is proper α-2^*-wind from d and $Q \subset R$. Then there exist chords r_P of P and r_R of R such that $r_P \in \mathfrak{L}_{\text{red}}(P)$ and $r_R \subseteq r_P \in E(Q)$. Let g be the chord of P from which it was determined that r_P is red (by Rule 3.2.11). Then, by Lemma 3.4.1, g is a chord of Q and thus g is also a chord of R. Also, for some subpolygon W of $\pi(P)$, we have that $g = \text{door}(P)$ and r_R is on the boundary of W by Rule 3.2.11. Clearly $W \in \pi(R)$ also. Therefore $g \in \mathfrak{L}_{\text{green}}(R)$ and thus $r_R \in \mathfrak{L}_{\text{red}}(R)$ by Rule 3.2.11. Therefore, by Lemma 3.4.2, R is not α-2^*-wind. □

For an x-2^*-wind polygon Q we refer to $\psi(Q)$ as the base of Q and denote it by base(Q). Then by the definition of maximal partition in Chapter 2, Section 2.2 and by Lemma 3.4.4, for any simple polygon P we have that $\pi_{\text{red}}(P)$ is a maximal $\pm x$-2^*-wind partition of P from $\psi(P)$.

From Lemma 3.4.4 the following result is clear:

THEOREM 3.4.5. *Let P be a simple polygon. Each element of $\pi_{\text{red}}(P)$ is a maximal proper $\pm x$-2^*-wind subpolygon of P.*

Let P be a simple polygon such that $|\pi_{\text{red}}(P)| > 1$ and let Q_1 and Q_2 be two elements of $\pi_{\text{red}}(P)$. By Theorem 3.4.5, Q_1 is proper α-2^*-wind from door(Q_1) for some angle $\alpha \in \{0, \pi\}$. We say that Q_1,Q_2 are *in opposition* if Q_2 is proper $-\alpha$-2^*-wind from door(Q_2).

LEMMA 3.4.6. *Let P be a simple polygon such that $|\pi_{\text{red}}(P)| > 1$ and let Q_1 and Q_2 be two adjacent elements of $\pi_{\text{red}}(P)$. Then Q_1 and Q_2 are in opposition.*

PROOF. Assume, contrary to hypothesis, that Q_1 and Q_2 are not in opposition. Let d_1 and d_2 be the doors of Q_1 and Q_2 respectively. Assume wlog that Q_1 is the parent of Q_2 and that Q_1 (Q_2) is proper x-2^*-wind from d_1 (d_2). Let Q_3 be the polygon such that $\text{int}(Q_3) = \text{int}(Q_1) \cup \text{int}(Q_2) \cup d_2$ and $\text{start}(Q_3) = \text{start}(Q_1)$. Now, by Lemmas 3.3.8 and 3.4.3, for any point on d_2 there exists a proper y-1-wind path from d_1 to that point. Also, by the same lemmas, for any point on Q_2 there exists a y-1-wind path from d_2 to that point. Therefore, by Lemma 3.3.8, polygon Q_3 is x-2^*-wind from d_1. But now Q_1 is not maximal from d_1, contradicting Lemma 3.4.4. □

THEOREM 3.4.7. *Let P be any simple polygon. Then any y-monotone curve C contained in P intersects the interior of at most two $\pm x$-2^*-wind polygons of $\pi_{\text{red}}(P)$.*

PROOF. Assume, by contradiction, that C intersects the interior of three different subpolygons Q,R,$T \in \pi_{\text{red}}(P)$ in succession. We further assume C is directed such that it originates in Q. Let C_Q, C_R, C_T be the portion of C interior to or on the doors of Q, R, T respectively. Assume wlog that R is proper x-2^*-wind from door(R). Then Q and T are proper $-x$-2^*-wind from their respective doors by Lemma 3.4.6. After symmetry there are two cases to consider:

Case 1: Both Q and T are children of R. Then C_Q must be proper y-1-wind since C exits Q at door(Q) and Q is proper $-x$-2^*-wind from its door. Similarly C_T must be proper $-y$-1-wind since C enters T at door(T) and T is proper $-x$-2^*-wind from its door.

Case 2: T is a child of R and R is a child of Q. Then C_R must be proper y-1-wind since C enters R at door(R) and R is proper x-2^*-wind from its door. Similarly, C_T must be proper $-y$-1-wind since C enters T at door(T) and T is proper $-x$-2^*-wind from its door.

Clearly either case contradicts that C is y-monotone. □

COROLLARY 3.4.8. *Assume that C is y-monotone and intersects subpolygon $W \in \pi_{\text{red}}(P)$. Then C can enter W or exit W but not both.*

From Theorem 3.4.7 and Corollary 3.4.8 the following result follows:

LEMMA 3.4.9. *Any curve C contained in a simple polygon P and decomposable into k y-monotone curves intersects the interior of at most $k+1$ of the polygons of $\pi_{\text{red}}(P)$.*

From the above result the following results are clear.

LEMMA 3.4.10. *Any curve contained in a simple polygon P and on the boundary of a convex set intersects the interior of at most four $\pm x$-2^*-wind polygons of $\pi_{\text{red}}(P)$.*

LEMMA 3.4.11. *Any convex set S contained in the interior of a polygon P intersects the interior of at most two $\pm x$-2^*-wind polygons of $\pi_{\text{red}}(P)$.*

PROOF. Let T be the subset of polygons of $\pi_{\text{red}}(P)$ that intersect S. Then S contains a directed y-monotone curve C with endpoints p_1 and p_2 such that p_1 has the maximum y-coordinate of any point on the boundary of S, p_2 has the minimum y-coordinate of any point on the boundary of S, and C intersects every polygon in T. Now, applying Theorem 3.4.7 to C, we get that $|T| \leq 2$. □

We have demonstrated that the proper $\pm x$-2^*-wind decomposition $\mathfrak{L}_{\text{red}}(P)$ of a polygon P has a number of useful properties. In particular we have shown that it is a visibility partition with a cover number of at most two, a circular cover number of at most four, and a convex cover number of at most two.

It is also easy to construct $\mathfrak{L}_{\text{red}}(P)$. Clearly $\mathfrak{L}(P)$ can be constructed in linear time from a horizontal trapezoidization of P which itself can be done in linear time [13]. By Rules 3.2.11 and 3.2.12 or by Rules 3.2.15 and 3.2.16 it is clear that $\mathfrak{L}_{\text{red}}(P)$ is easily constructed in linear time from $\mathfrak{L}(P)$. We can also easily construct $\pi_{\text{red}}(P)$ from $\mathfrak{L}_{\text{red}}(P)$ in linear time. We summarize these results in the following theorems.

THEOREM 3.4.12. *For any polygon P, $\pi_{red}(P)$ is a visibility proper $\pm x$-2^*-wind subpolygon partition of P with a cover number of two, a circular cover number of four and a convex cover number of two.*

THEOREM 3.4.13. *Given a polygon P, we can construct $\mathfrak{L}_{\text{red}}(P)$ and $\pi_{\text{red}}(P)$ in linear time.*

Although we do not investigate problems on polygons with holes in this thesis the simplicity of the proper $\pm x$-2^*-wind visibility partitioning algorithm suggests that it might be modified to work for polygons with holes. This is indeed the case albeit at the cost of some extra steps. See Chapter 7 for the algorithm and further details.

It is also noteworthy that Lemma 3.3.8 and many of the theoretical results of this section that follow from it can be generalized to higher dimensions. This is also discussed in Chapter 7.

3.5. Plane Sweep Algorithms

A common method of solving Computational Geometry problems is that of a plane sweep of the vertices. There are three variations of this approach that are quite useful in processing a proper $\pm\alpha$-2^*-wind decomposition of polygon P.

Monotone Sweep: This applies to monotone polygons only. The vertices of P are processed according to their y-coordinate distance from start(P). In case of ties on the same monotone chain, the order of the vertices in the chain is used. In the case of ties on different monotone chains, the ordering in a traversal of P is used to resolve the ties. This method is used, for example, in triangulating monotone polygons [**24**].

Nested Monotone Sweep: A monotone sweep is used on each of the polygons in $\pi(P)$. Each y-monotone polygon $Q_i \in \pi(P)$ can be processed once the subpolygon of $\pi(P)$ coincident to door(Q_i) has been processed.

Blocking Sweep: A monotone sweep is used on each of the y-monotone subpolygons in $\pi(P)$ but the vertices are processed in reverse order. Each y-monotone polygon $Q_i \in \pi(P)$ can be processed once the subpolygons of $\pi(P)$ coincident to windows (Q_i) have been processed.

For nested monotone sweep and blocking sweep Q_i may fall into one of five disjoint classifications that may be processed differently.

(1) windows$(Q_i) = \emptyset$. For a blocking sweep Q_i is not blocked and can be processed right away.

(2) windows$(Q_i) \cap \mathfrak{L}_{\text{green}}(P) \neq \emptyset$. In this case Q_i has only one window.

(3) windows$(Q_i) \cap \mathfrak{L}_{\text{blue}}(P) \neq \emptyset$ but windows$(Q_i) \cap \mathfrak{L}_{\text{red}}(P) = \emptyset$.

(4) windows$(Q_i) \cap \mathfrak{L}_{\text{red}}(P) \neq \emptyset$ but windows$(Q_i) \cap \mathfrak{L}_{\text{blue}}(P) = \emptyset$.

(5) windows$(Q_i) \cap (\mathfrak{L}_{\text{blue}}(P) \cup \mathfrak{L}_{\text{red}}(P)) \neq \emptyset$.

In the last case the required action may simply be the sum of the actions required in the third and fourth cases or may be a different action. Also, the action taken when door (Q_i) is an edge of P may be different than when door (Q_i) is a chord of P.

The event point for each of these sweeps is a vertex. However we will often use other objects for event points. The event points could be trapezoids or the chords of trapezoids. Except for monotone sweep the event points could also be the elements of $\pi(P)$ or $\Psi(P)$.

One approach to solving a problem on a simple polygon is to first solve the problem for monotone polygons using monotone sweep, then generalize the result to 2^*-wind polygons using nested monotone sweep or blocking sweep, and finally to generalize the algorithm for simple polygons using nested 2-wind sweep or blocking sweep.

3.6. Applications

The main purpose of our 2*-wind subpolygon partition generating algorithm is as a stepping stone to developing algorithms that generate visibility decompositions whose component polygons are even more constrained. However we can use a $\pm x$-2*-wind subpolygon partition of a simple polygon directly. In this section we present an application of a $\pm x$-2*-wind subpolygon of a simple polygon. We will study the partitioning of a polygon into uniformly monotone subpolygons; by uniformly monotone we mean that the subpolygon are all monotone in the same direction.

The partitioning of simple polygons into monotone subpolygons has been studied in the literature before. In [31] an algorithm to decompose a polygon into the minimum number of monotone polygons was presented where the subpolygons were not required to be uniformly monotone. This algorithm requires $O(n^4N)$ time where n is the number of vertices in the polygon and N is the number of reflex vertices in the polygon.

In the case that the subpolygons are required to be uniformly monotone and the partition is restricted to be constructed by chords of the polygon, an algorithm requiring $O(N^2 n \log n + nN^3 + N^5)$ time was presented in [38]. This paper also presents an algorithm that partitions a simple polygon into the minimum number of y-monotone subpolygons using only chords in $O(n \log n + nN + N^3)$ time. These results were improved to $O(m * n)$ in [5] where m is the number of pairs of reflex cusps of the polygon such that one is a floor cusp, the other is a ceiling cusps and there is a chord of the polygon joining the two cusps. Finally, in [11] an algorithm was presented that partitions a polygon into the minimum number of y-monotone subpolygons using only chords in $O(n^2)$ time. We do not attempt to solve this problem in this thesis. However, it is trivial to solve this problem in linear time for the special case that the

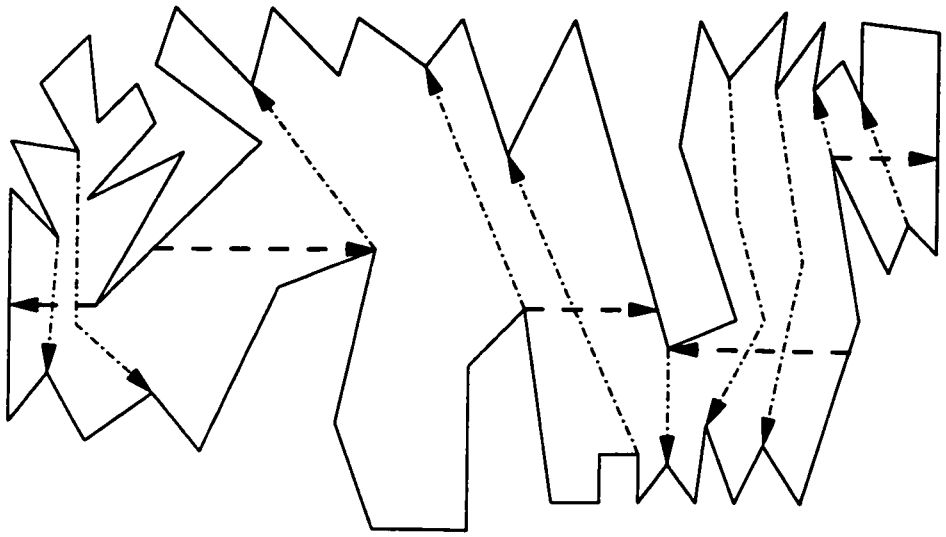
polygon is x-2^*-wind. Hopefully we will eventually be able to use this partial result to improve upon the results in [38] and [11].

Instead, we consider the case that we are not restricted to using partitions involving only chords of the polygon. Let P be a simple polygon with n vertices. We say that a y-monotone chain, say C, interior to P except at its endpoints, is a *column chain* of P if there exist two reflex cusps ψ_i and ψ_k of P such that one endpoint of C belongs to each of ψ_i and ψ_k and C intersects both $\check{P}(\psi_i)$ and $\check{P}(\psi_k)$; it may be that $\check{P}(\psi_i) = \check{P}(\psi_k)$. Note that one of ψ_i and ψ_k is a ceiling cusp and the other is a floor cusp. Let m be the maximum number of non-intersecting column chains that can be used to partition P. Let r be the number of reflex cusps of P. Then it is straightforward to show that the minimum number of y-monotone subpolygons into which P can be partitioned is $r - m + 1$; note that in [38] this is shown to be the case when the chains are restricted to be chords.

The following algorithm constructs a maximum set of non-intersecting column chains of a simple polygon P. With the output of this algorithm it is straightforward to further partition P until all the subpolygons are y-monotone. See Figure 3.6.1.

The time complexity of Algorithm 1 depends upon the cost of constructing the actual y-monotone column chains. If we restrict the analysis to the cost of determining the pairs of reflex cusps that each y-monotone column chains intersects then the cost can be shown to be $O(n)$. If we actually construct a maximal set on non-intersecting y-monotone column chains then as much as $O(n^2)$ space may be required. This situation can occur when there are $O(n)$ chains and each chain must pass through an element, say R, of $\pi(P)$ where $O(n)$ turns are required to pass through R.

However, it is not necessary to construct these chains in order for Algorithm 1 to be useful. Using Algorithm 1, we can determine, in linear time, the number of subpolygons in a minimum y-monotone subpolygon partition of P. We can use



These chains would be an example of the output of Algorithm 1 for a polygon P.

FIGURE 3.6.1. A maximum set of non-intersecting monotone column chains of P

Algorithm 1

Input: A polygon P.

Output: A maximum set y-monotone column chains of P such that no two intersect or intersect the same cusp.

1: Construct $\pi(P)$ and $\pi_{\text{red}}(P)$.

2: $C := \emptyset$. /* C is a set of non-intersecting y-monotone column chains of P.*/

3: Set each reflex cusp of $\mathfrak{L}(P)$ to unmarked.

4: **For** each $Q \in \pi(P)$ in blocking sweep order **do**

gWin := the subset of $\mathfrak{L}_{\text{green}}(P)$ that are chords of Q.

For each $T \in \Pi(\text{gWin})$ in blocking sweep order **do**

Let $g = \text{door}(T)$ and e_T be the edge of T containing g. Compute Ψ_r, the set of unmarked reflex cusps that are either reflex cusps of the tree y-monotone subpolygons whose bases are part of e_T or are reflex cusps that are part of e_T. Let Y_b be the set of unmarked reflex cusps such that there exists a generalized chord between each element of Ψ_r and each element of Ψ_b.

while there is an unmarked element in Ψ_r and in Ψ_b **do**

Select unmarked elements $\psi_r \in \Psi_r$ and $\psi_b \in \Psi_b$ that minimizes $\ell(\psi_r, \psi_b)$.

Let c be a column chain from ψ_r to ψ_b that intersects no element of C.

$C := C \cup \{c\}$. Mark ψ_r and ψ_b.

5: Return(C).

this value to determine how many subpolygons in excess of the minimum are the y-monotone subpolygon partitions of P found by applying the algorithms in [38] or [11].

CHAPTER 4

Monotone Subpolygon Partition Classes

4.1. Introduction

In this chapter we study partitions of polygons into uniformly monotone subpolygons. By uniformly monotone we mean that the subpolygons are all monotone in the same direction. Unless otherwise stated the subpolygons will always be y-monotone. The problem of partitioning a simple polygon into the minimum number of uniformly monotone subpolygons has been covered in Section 3.6.

In this chapter we are interested in decomposing a polygon into uniformly monotone subpolygons such that the monotone cover number is minimized or at least kept small. It was shown in Chapter 3 that, given a simple polygon P, $\pi_{\text{red}}(P)$ is a $\pm x$-2^*-wind partition of P such that $\mathsf{C}_{\uparrow}(\pi_{\text{red}}(P)) \leq 2$, $\mathsf{C}_{\downarrow}(\pi_{\text{red}}(P)) \leq 2$, $\mathsf{C}_{\circlearrowleft}(\pi_{\text{red}}(P)) \leq 4$, and $\mathsf{C}_{\diamond}(\pi_{\text{red}}(P)) \leq 2$. In this section we show how to construct a partition Π of an x-2^*-wind polygon Q with m vertices into y-monotone subpolygons such that $\mathsf{C}_{\uparrow}(\Pi)$, $\mathsf{C}_{\downarrow}(\Pi)$, $\mathsf{C}_{\circlearrowleft}(\Pi)$, and $\mathsf{C}_{\diamond}(\Pi)$, are all $O(\log m)$. Thus, by combining these partition algorithms, we can construct a partition Π' of a simple polygon P with n vertices into y-monotone subpolygons such that $\mathsf{C}_{\uparrow}(\Pi')$, $\mathsf{C}_{\downarrow}(\Pi')$, $\mathsf{C}_{\circlearrowleft}(\Pi')$, and $\mathsf{C}_{\diamond}(\Pi')$, are all $O(\log n)$. We call any partition of P for which the monotone cover number is $o(n)$ a *monotone subpolygon visibility partition* or *MSVP* of P. If the direction of monotonicity is any direction other than the y direction, we call the partition an α-MSVP where α is the direction of monotonicity. We provide two separate algorithms for constructing MSVPs of simple polygons, both of which produce MSVPs with a cover number

that is $O(\log n)$. One of these MSVP generating algorithms takes into consideration the size of the subpolygons constructed and the other ignores this factor.

Our results are not the first algorithms for constructing uniformly monotone subpolygon visibility decompositions. In [16] the authors present an algorithm that constructs a uniformly y-monotone subpolygon visibility cover of a polygon in $O(n \log n)$ time and space such that the y-monotone cover number is $O(\log n)$. They call this cover a Hierarchical Vertical Decomposition. This monotone visibility cover is noteworthy in that it allows a set of non-overlapping subpolygons that cover a y-monotone curve to be generated dynamically; as far as we know this is the only visibility decomposition that uses this idea. We believe the idea of dynamically generating the set of subpolygons that cover a curve to be very useful and may be used in future as a way to construct visibility decompositions.

Our MSVP generating algorithms generate partitions that also have a monotone visibility cover number of $O(\log n)$ but they can be constructed in linear time and space. Furthermore, our algorithms are simpler and have smaller complexity constant coefficients. We point out, however, that the y-monotone visibility cover in [16] was designed specifically to solve the circular ray shooting problem (see Chapter 5) and for this problem the query time and space complexity of the algorithm is the same as the result that we present for this problem in Section 5.5 using one of our MSVP generating algorithms. However, our algorithm for solving this problem is simpler.

In Section 4.2 we present our first MSVP class and a linear time algorithm that generates it. This algorithm constructs a MSVP of a polygon such that the monotone cover number is minimized for $\pm x$-2*-wind polygons and minimized to within two for a simple polygon. We then show that this partition is within one of the minimum if the chords inducing the partition are restricted to being horizontal. We also show how to modify any partition generated by the this algorithm so that the new partition has

a monotone visibility cover number that is minimum among partitions constructed from horizontal chords. The algorithm that makes this modification runs in linear time.

For problems in which the sizes of the subpolygons is a concern, it is sometimes better to partition the polygon differently than by the method used in Section 4.2. In Section 4.3 we develop an algorithm that constructs a MSVP that does consider the sizes of the subpolygons. In linear time and space, it constructs a MSVP, for an x-2*-wind polygon, such that the sequence of y-monotone subpolygons intersected by any x-1-wind chain interior to the polygon have sizes that reduce by at least half with each step along the sequence. This property allows weighted binary trees to be used when solving problems using the partitions generated by this algorithm. Let P be a simple polygon with n vertices. By using this MSVP generating algorithm and weighted binary trees, instead of the MSVP generating algorithm from Section 4.2, we are able to reduce the complexity of solving the query component of the circular ray shooting problem by a factor of $O(\log n)$. The circular ray shooting problem is: "given a simple polygon P, preprocess it so that when given a directed circular arc ζ, the first intersection of ζ with P can be found quickly.[1] The improvements in our solution to this problem over the result in **[16]** are largely due to the use of the MSVP generating algorithm developed in Section 4.3. The circular ray shooting problem problem is dealt with in Chapter 5.

We remind the reader that, for a simple polygon P, we use $\Psi(P)$ to denote the ordered set of cusps of P, $\mathfrak{L}(P)$ to denote the lids of the open cusps in $\Psi(P)$, and $\pi(P)$ to denote the set of (y-monotone) subpolygons into which P is partitioned by the elements of $\mathfrak{L}(P)$. These and several other terms that we use here have been defined in Chapters 2 and 3.

[1]For a precise formulation of this problem see Chapter 5.

4.2. A Monotone Subpolygon Visibility Partition Class

In this section we develop an algorithm for constructing a decomposition of an x-2^*-wind polygon into y-monotone subpolygons such that the monotone cover number of the partition is minimized. Using this algorithm we can decompose a simple polygon into y-monotone polygons such that the monotone cover number of the partition is within two of the minimum. We then show that the monotone cover number of this partition is within one of the minimum if the chords inducing the partition are constrained to being horizontal. The partitions generated using this algorithm are monotone subpolygon visibility partitions.

Suppose we wish to construct a partition of a simple polygon P, using only chords of P, into y-monotone subpolygons such that the monotone cover number is minimum. The following lemma provides a useful constraint on the problem. First we need a proposition. We remind the reader that for a partition Π of a simple polygon, $\widehat{\Gamma}(\Pi)$ denotes the set of chords that induces Π.

PROPOSITION 4.2.1. *Let Π be a partition of P such that $\mathbf{C}_i(\Pi)$ is minimized. Assume wlog that $\widehat{\Gamma}(\Pi)$ contains the minimum number of non-horizontal chords possible while still minimizing $\mathbf{C}_i(\Pi)$. Then, for each non-horizontal chord γ_i of $\widehat{\Gamma}(\Pi)$, there exists a reflex cusp ψ_i of $\Psi(P)$ such that γ_i intersects both ψ_i and $\check{P}(\psi_i)$.*

PROOF. Let $\Gamma = \widehat{\Gamma}(\Pi)$. Assume otherwise that there exists a non-horizontal chord γ of Γ such that for no reflex cusp ψ of P does γ intersect both ψ and $\check{P}(\psi)$. If

1) γ intersects no cusps of P or

2) γ intersects only one cusp, say ψ_i, but γ does not intersect $\check{P}(\psi_i)$ or

3) γ intersects a ceiling cusp, say ψ_j^c, and a floor cusp, say ψ_j^f, but does not intersect either $\check{P}(\psi_j^c)$ or $\check{P}(\psi_j^f)$

then $\Gamma - \{\gamma\}$ induces a partition, say Π_γ, of P into y-monotone subpolygons and

$\mathbf{C}_i(\Pi_\gamma) = \mathbf{C}_i(\Pi)$ but $\Gamma - \{\gamma\}$ has fewer non-horizontal chords than Γ which is impossible by the constraints on Γ. For the remaining cases we show that ψ must exist. If γ intersects two floor reflex cusps or two ceiling reflex cusps, say ψ_k and ψ_l, then, for exactly one of these cusps, say ψ_k, chord γ intersects $\check{P}(\psi_k)$. Now let $\psi = \psi_k$ and we are done. Finally, if γ intersects two reflex cusps such that one is a ceiling cusp, say ψ_c, and the other is a floor cusp, say ψ_f, such that case 3) above is not satisfied then γ intersects both $\check{P}(\psi_c)$ and $\check{P}(\psi_f)$. Now ψ may be either ψ_c or ψ_f. □

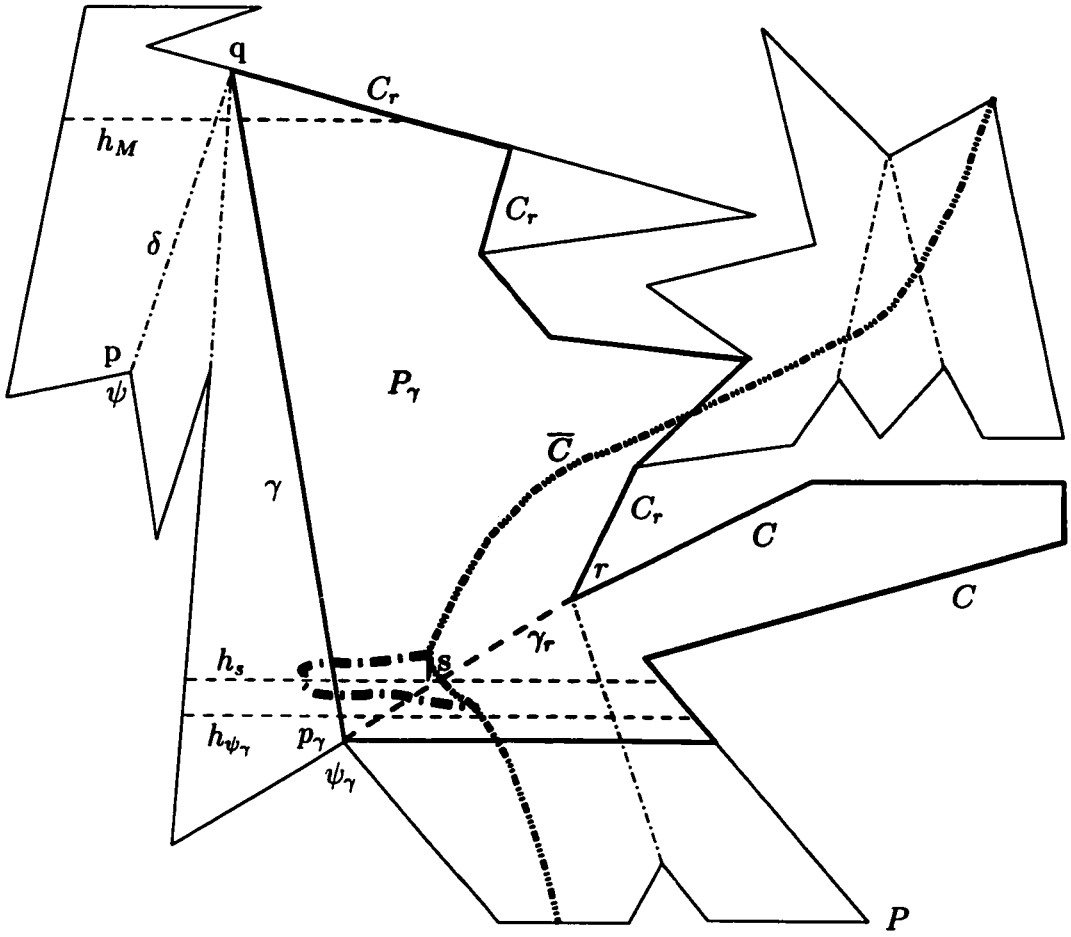
We say that a chord γ of a simple polygon P is a *column chord* if there exist two reflex cusps ψ_i and ψ_k of P such that one endpoint of γ belongs to ψ_i, the other endpoint of γ belongs to ψ_k, and γ intersects both $\check{P}(\psi_i)$ and $\check{P}(\psi_k)$; it may be that $\check{P}(\psi_i) = \check{P}(\psi_k)$. Note that one of ψ_i and ψ_k is a ceiling cusp and the other is a floor cusp.

Let C be a y-monotone curve interior to a simple polygon P and let Π be a partition of P into y-monotone subpolygons. Then we denote by $\mathbf{C}_i(\Pi, C)$ the number of subpolygons in Π that are intersected by C.

LEMMA 4.2.2. *Let a partition Π of a simple polygon P be such that $\mathbf{C}_i(\Pi)$ is minimized. Then $\widehat{\Gamma}(\Pi)$ need not contain any non-horizontal chords unless these chords are column chords.*

PROOF. Let $\Gamma = \widehat{\Gamma}(P)$ be a set of chords that induces Π. We assume wlog that Γ contains the fewest number of chords that are not horizontal yet are not column cords of any choice for Γ. We will show that this number is zero.

Assume, by way of contradiction, that Γ contains one or more non-horizontal chords that are not column chords. We claim that at least one of these chords, say γ, can be removed from Γ and replaced by another chord, say γ_{add} that is either a horizontal or column chord and such that $\Gamma \cup \{\gamma_{\text{add}}\} - \{\gamma\}$ induces a partition, say Π_{new}, of P and $\mathbf{C}_i(\Pi_{\text{new}})$ is minimized. The existence of such a chord would violate



In any partition Π of a polygon P such that $\mathsf{C}_i(\Pi)$ is minimized either $\widehat{\Gamma}(\Pi)$ contains no chords that are neither non-horizontal nor non column chords of P or else one these chords can be replaced by a column chord or horizontal chord to produce of partition $\overline{\Pi}$ of P such that $\mathsf{C}_i(\overline{\Pi})$ is minimized.

FIGURE 4.2.1. Replacing a non column chord by a column chord

the constraints on Γ, thus proving this lemma. The main difficulty in finding γ and γ_{add} is in ensuring that γ_{add} does not cross any element of Γ. We now find γ and γ_{add}.

Let δ be an element of Γ that is neither horizontal nor a column cord. First, by Proposition 4.2.1, for some reflex cusp, say ψ, of P we have that δ intersects both ψ and $\check{P}(\psi)$. Let p be the endpoint of δ belonging to ψ and let q be the other endpoint

of δ. We assume wlog that ψ is a floor cusp and that q has the minimum y-coordinate of the choices for δ in Γ that satisfy the constraints on δ. Let Γ_q be the subset of non-horizontal chords of Γ whose upper endpoint has the same height as q. If q is on a vertex of P or an edge of P which is a not a convex cusp of P then let Γ'_q be the subset of Γ_q whose upper endpoint is q. Otherwise let Γ'_q be the subset of Γ_q whose upper endpoint is on the convex cusp of P to which q belongs.

If δ is the only element of Γ'_q then we set $\gamma = \delta$. Otherwise we select γ as follows. Let M be any maximal y-monotone subchain of P to which q belongs. There will be two choices for M if q belongs to a convex cusp of P; one choice otherwise. Let h_M be a horizontal line below but arbitrarily close to q. We select for γ the element of Γ_q whose intersection with h_M is nearest to M. Let p_γ be the endpoint of γ that is not q and ψ_γ be the reflex cusp to which p_γ belongs.

Now we need to find γ_{add}. Consider the largest subpolygon of P containing the portion of the interior of P visible from p_γ and on or above the horizontal line through p_γ. Chord γ partitions this visibility polygon into two parts. Let P_γ be the part whose boundary includes a portion of M that is below q.

Now let C be the directed path on $P_\gamma - \gamma$ from q to p_γ. If C contains any points that are on reflex cusps of P and are visible from p_γ then let r be the first such point and let γ_r be the chord of P from p_γ to r. (We point out that it is possible for the endpoint of a reflex cusp of P to be on C yet not be visible from p_γ.) Otherwise let γ_r be the lid of ψ_γ on $P_\gamma - \gamma$ and let r be the endpoint of this lid that is not on ψ. Now let C_r be the subpath of C from q to r. We point out that C_r is a y-monotone chain since no point of C_r that is visible from p_γ is part of any reflex cusp of P. See Figure 4.2.1.

Claim 1: γ_r is a chord of P that does not intersect any element of Γ except at p_γ or possibly r.

Proof. Consider that γ_r intersects some chord, say γ_Γ, other than at p_γ where $\gamma_\Gamma \in \Gamma$. Then γ_Γ cannot cross γ because $\gamma \in \Gamma$ and Γ induces a partition of P. Furthermore γ_Γ cannot cross C_r (at a chord of P); otherwise γ_Γ would partition P_γ into two subpolygons, one of which would contain a convex cusp that is also a cusp of C_r thus violating that C_r is y-monotone. Therefore an endpoint of γ_Γ, say t, must belong to $C_r - r$. But since C_r is y-monotone we have that either $t_y < q_y$ or $t = q$ and h_M intersects γ_Γ at a point closer to M than the point where h_M intersects γ_r. Furthermore, by the construction of M and Proposition 4.2.1, the existence γ_Γ violates the rules by which γ was chosen (We should have chosen γ_Γ.).

Also, by construction, γ_r is either a horizontal chord or a column chord. We claim now that γ_r is the chord to use for γ_{add}. To see this, let $\overline{\Gamma} = \Gamma \cup \{\gamma_r\} - \{\gamma\}$. By Claim 1, $\overline{\Gamma}$ induces a partition, say $\overline{\Pi}$, of P. Furthermore, $\overline{\Gamma}$ has one fewer chord that is non-horizontal and not a column chord than does Γ. Therefore, if the monotone cover number of $\overline{\Pi}$ is minimum then the conditions by which Γ were selected are violated and this lemma is proved. We now show that this is the case.

Let $\overline{C}$ be a y-monotone curve interior to P except at its endpoints, such that $\mathsf{C}_i(\overline{\Pi}, \overline{C}) = \mathsf{C}_i(\overline{\Pi})$ and assume wlog that the endpoints of $\overline{C}$ belong to convex cusps of P. From $\overline{C}$ we will construct a y-monotone curve, $\widehat{\overline{C}}$, interior to P except at its endpoints, such that $\mathsf{C}_i(\overline{\Pi}, \overline{C}) \leq \mathsf{C}_i(\Pi, \widehat{\overline{C}})$. Now if $\overline{C}$ does not intersect γ_r then we are done, so assume $\overline{C}$ intersects γ_r at a point s. Let h_s be the horizontal chord of P that goes through s.

If γ_r is not horizontal then h_s also intersects γ. Thus we can remove from $\overline{C}$ a sufficiently small portion of $\overline{C}$ that includes s and replace that portion of $\overline{C}$ by another y-monotone curve with the same endpoints in such a way that the new y-monotone curve is interior to P and intersects γ. Let this curve be $\widehat{\overline{C}}$. Then $\mathsf{C}_i(\Pi) \geq \mathsf{C}_i(\Pi, \widehat{\overline{C}}) \geq \mathsf{C}_i(\overline{\Pi}, \widehat{\overline{C}}) = \mathsf{C}_i(\overline{\Pi})$.

If, instead, γ_r is horizontal then let h_{ψ_γ} be any horizontal chord of $\check{P}(\psi_\gamma)$ such that h_{ψ_γ} is below q. Now $\overline{C}$ necessarily intersects h_{ψ_γ} and h_{ψ_γ} necessarily intersects γ. Thus we can remove the section of $\overline{C}$ between γ_r and h_{ψ_γ} and replace it by another y-monotone curve with the same endpoints in such a way that the new y-monotone curve is interior to P and intersects γ. Let this curve be $\widehat{\overline{C}}$. Now again $\mathtt{C}_i(\Pi) \geq \mathtt{C}_i(\Pi, \widehat{\overline{C}}) \geq \mathtt{C}_i(\overline{\Pi}, \widehat{\overline{C}}) = \mathtt{C}_i(\overline{\Pi})$.

Thus we have violated the rules by which Γ was selected. Therefore no such chord γ exists. □

Based on Lemma 4.2.2 we also know the following:

COROLLARY 4.2.3. *Any $\pm x$-2*-wind polygon admits a partition using only lids of its reflex cusps such that the monotone cover number of the partition is minimized.*

Let P be a x-2*-wind polygon. Next we develop an algorithm that constructs a partition of P whose monotone cover number is minimum. Based on Corollary 4.2.3 we restrict ourselves to partitions of P that use only the lids of $\Psi(P)$. Clearly, for any reflex cusp $\psi_i \in \Psi(P)$, any such y-monotone subpolygon partition contains at least one chord $\gamma_i \in \text{lids}(\psi_i)$. We need to construct a y-monotone subpolygon partition of P such that every chord in the partition is necessary to ensure that the subpolygons are y-monotone. We call such a partition a *minimal monotone subpolygon partition* or *MMSP;* recall that we assume the chords of a MMSP are horizontal.

Let Γ be any horizontal set of chords of P that induces a partition, say Π, of P, whose monotone cover number is minimum; we know that Γ and Π exist by Corollary 4.2.3. Clearly Γ contains only chords of $\mathfrak{L}(P)$. Furthermore, unless $\mathfrak{L}(P) = \emptyset$ (i.e. unless P is y-monotone), Γ cannot contain every chord of $\mathfrak{L}(P)$. Let Q be any (y-monotone) subpolygon in $\pi(P)$ such that $\text{windows}(Q) \neq \emptyset$. We make the following observations:

OBSERVATION 4.2.4. *Exactly one element of windows(Q) is not contained in Γ.*

Let $\gamma(\Gamma, Q)$ denote the element of windows(Q) not contained Γ.

OBSERVATION 4.2.5. *Let γ be any element of windows$(Q) - \gamma(\Gamma, Q)$. Then $\Gamma + \{\gamma\} - \{\gamma(\Gamma, Q)\}$ induces a MMSP of P.*

Now, from Observation 4.2.4, we get the following result:

OBSERVATION 4.2.6. *Π is a MMSP of P.*

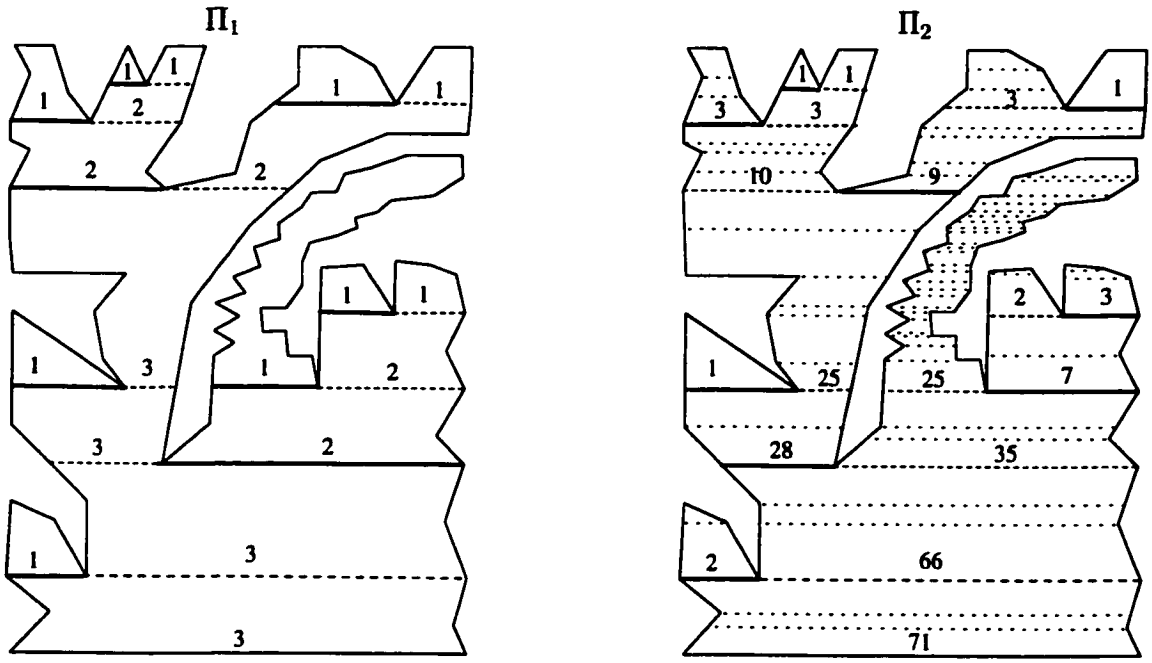
Based on Observation 4.2.6 we restrict ourselves to MMSPs of P. Using Observations 4.2.4 and 4.2.5 we claim that Algorithm 2 constructs a MMSP of an $\pm x$-2^*-wind polygon whose monotone cover number is minimum. We further show that Algorithm 2 generates a y-monotone subpolygon visibility partition class. For each $d_i = \text{door}(Q_i)$ where $Q_i \in \pi(P)$, we will assign a weight $w(d_i)$. An example of the partitioning of a polygon by the output of Algorithm 2 is given in Figure 4.2.2.

Algorithm 2

Input: A simple $\pm x$-2^*-wind polygon P.
Output: A MSVP Π of P such that $\mathfrak{C}_i(\Pi)$ is minimized.

1: $\Gamma := \emptyset$.
2: Construct $\pi(P)$.
3: Do a blocking monotone sweep of $\pi(P)$: for each $Q \in \pi(P)$ do
  if windows$(Q) = \emptyset$ **then** $w(\text{door}(Q)) := 1$
  else weight $:= \max\{w(\gamma_i) : \gamma_i \in \text{windows}(Q)\}$.
    $\gamma :=$ the rightmost element of windows(Q) such that $w(\gamma) =$ weight.
    $\Gamma := \Gamma \cup \text{windows}(Q) - \{\gamma\}$.
    $w(\text{door}(Q)) := w(\gamma)$.
    if $|\{\gamma_i : \gamma_i \in \text{windows}(Q) \text{ and } w(\gamma_i) = \text{weight}\}| > 1$ **then**
      $w(\text{door}(Q)) := w(\text{door}(Q)) + 1$.
4: Let Π be the partition of P induced by Γ.
5: Return(Π).

In order to prove that Algorithm 2 constructs a MSVP whose monotone cover number is minimized, we consider the dual of $\pi(P)$. Let T be the dual of $\pi(P)$ where the subpolygons of $\pi(P)$ correspond to the vertices of T and where there is an edge between two vertices of T if and only if the subpolygons corresponding to the vertices are coincident to a common element in $\mathfrak{L}(P)$.



Partitions Π_1 (Algorithm 2) and Π_2 (Algorithm 5) of an x-2-wind polygon P into y-monotone polygons such that $\mathbf{C}_t(\Pi_1) = O(\log|P|)$ and $\mathbf{C}_t(\Pi_2) = O(\log|P|)$

FIGURE 4.2.2. Efficient y-monotone partitions of a polygon

We claim that there is a one to one correspondence between the MMSP of P constructed by Algorithm 2 and the partition of T into root subpaths, denoted $\widehat{\pi}(T)$ in Subsection A.4.2. First, any y-monotone subpolygon in an MMSP corresponds to a root subpath of T. Thus any MMSP of P corresponds to a partition of T into maximal length root subpaths. Finally, the rules used in Algorithm 2 to determine which elements of $\mathfrak{L}(P)$ to put into Γ correspond to the rules used to determine which edges to remove from T in order to construct $\widehat{\pi}(T)$. The following three lemmas follow from the correspondence between Π and $\widehat{\pi}(T)$ and the lemmas proved about $\widehat{\pi}(T)$.

First, we observe that $|T| = |\pi(P)| \leq n - 1$. Therefore, as a direct consequence of Corollary A.4.9, we have the following result.

LEMMA 4.2.7. *Algorithm 2 constructs a MSVP Π such that $\mathbf{C}_t(\Pi) \leq \log(n)$.*

LEMMA 4.2.8. *Algorithm 2 builds a MSVP* Π *of* P *such that the monotone cover number of* Π *is minimum.*

PROOF. From Corollary 4.2.3 we know that there exists a partition of P with a minimized monotone cover number that uses only horizontal chords. From Observations 4.2.4, 4.2.5, and 4.2.6 we know that there exists a MMSP of P whose monotone cover number is minimized. Now, that Π is a MMSP of P whose monotone cover number is minimized follows from considering the dual of $\pi(P)$ and Lemma A.4.8. □

The following lemma will be used to prove Theorem 4.2.10 which deals with nearly minimized y-monotone subpolygon partitions of simple polygons.

LEMMA 4.2.9. *Let* Π *be the MSVP of* P *constructed by Algorithm 2. Let* Q *be a subpolygon of* P *consisting of portions of* P *including all of base*(P) *but no convex cusps of* P *and a subset of* $\widehat{\Gamma}(\Pi)$. *Let* $\{Q_1, Q_2, \ldots, Q_k\}$ *be the set of maximal subpolygons of* P *that intersect* Q *but not the interior of* Q. *For* $1 \leq i \leq k$, *let* Γ_i *be the subset of* $\widehat{\Gamma}(\Pi)$ *that are chords of* Q_i, *and let* Π_i *be the partition of* Q_i *induced by* Γ_i. *Assume wlog that* $\mathbf{C}_i(\Pi_1) \geq \mathbf{C}_i(\Pi_i)$, $2 \leq i \leq k$. *Assume that there exists a y-monotone path interior to* P, *say* C, *that intersects* $\mathbf{C}_i(\Pi)$ *elements of* Π *and intersects* Q_1. *Then* $\mathbf{C}_i(\Pi) \leq \mathbf{C}_i(\Pi_1) + \log k$.

PROOF. Note that $\mathbf{C}_i(\Pi_i)$ is minimized, $1 \leq i \leq k$. Now the statement of this Lemma corresponds directly to the statement of Lemma A.4.11 and thus a proof of this lemma corresponds directly to the proof of Lemma A.4.11. □

Of course we would like to generalize the above algorithm with a method of constructing a partition of a simple polygon whose monotone cover number is minimum. However, for simple polygons the problem is more difficult. Consider the case of a simple polygon Q with only two reflex cusps ψ_1 and ψ_2 and assume these reflex cusps

can be joined by a chord γ such that γ intersects $\check{P}(\psi_1)$ and $\check{P}(\psi_2)$. Then γ partitions the polygon into two y-monotone components and clearly the monotone cover number of this partition is two. But the monotone cover number of any partition using only the elements of $\mathfrak{L}(Q)$ has a cover number of at least three so a partition of a simple polygon with a minimum cover number may include non-horizontal chords. Nevertheless, for a simple polygon P, we can construct a partition of P using only the elements of $\mathfrak{L}(Q)$ such that the cover number of the partition is within two of the minimum. Algorithm 3 constructs such a partition.

Algorithm 3

Input: A simple polygon P.
Output: A MSVP Π of P such that $\mathsf{C}_i(\Pi)$ is within two of the minimum.

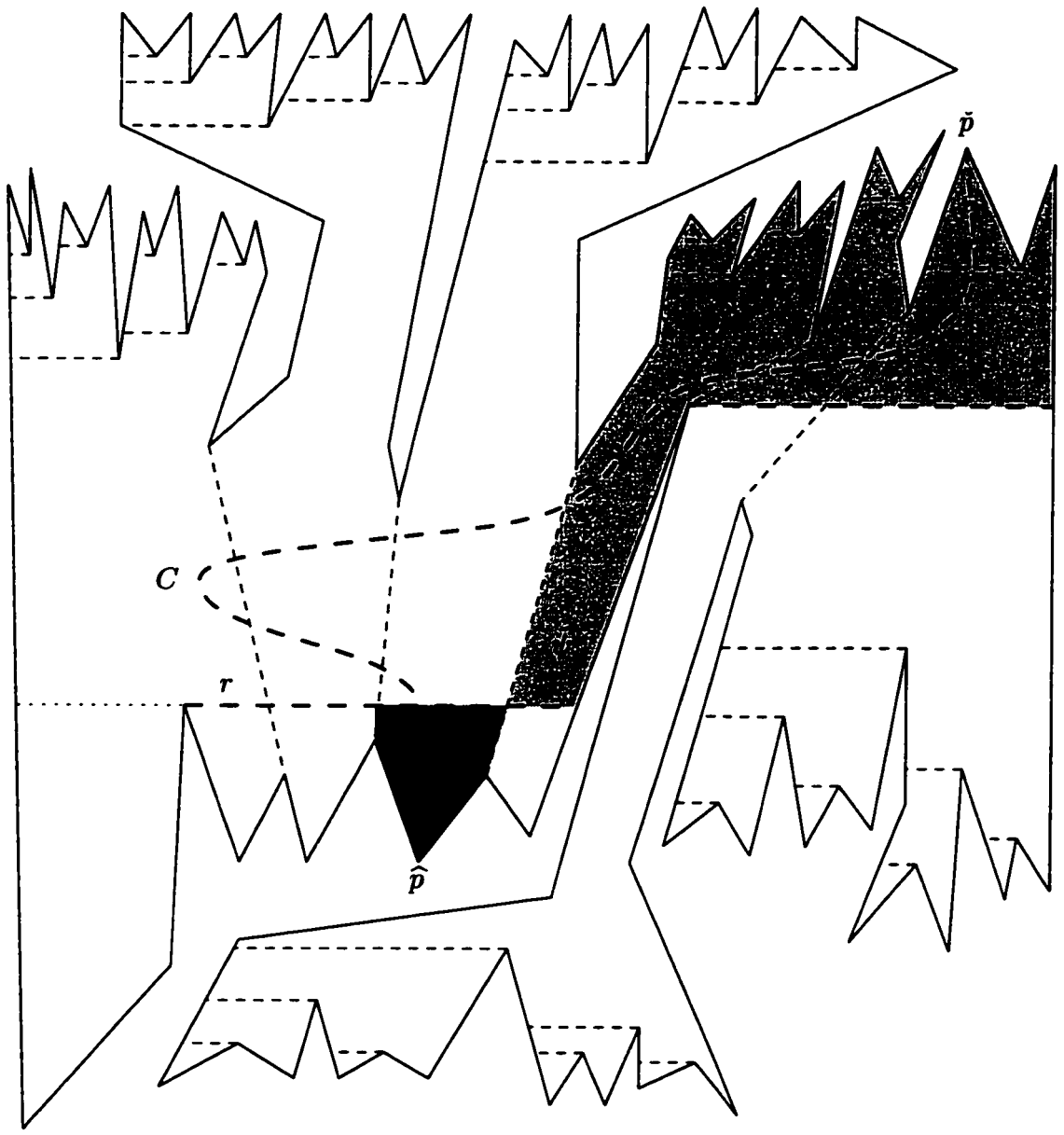
1: Construct $\pi(P)$ and $\pi_{\text{red}}(P)$.
2: Apply Algorithm 2 to each $\pm x$-2^*-wind polygon of $\pi_{\text{red}}(P)$.
3: Return the partition of P which is the union of the partitions returned by applications of Algorithm 2 to each $\pm x$-2^*-wind polygon of $\pi_{\text{red}}(P)$.

For any polygon R we denote $\mathsf{C}_i(R) = \min\{\mathsf{C}_i(\Pi) : \Pi \text{ is a partition of } R\}$. We refer to $\mathsf{C}_i(R)$ as the monotone cover number of R.

Let $\overline{\Pi}$ be the partition generated by applying Algorithm 3 to a simple polygon P.

THEOREM 4.2.10. *$\overline{\Pi}$ is a partition of P whose monotone cover number is within two of the minimum.*

PROOF. Let Γ be a set of chords that induces a partition, say Π, of P such that $\mathsf{C}_i(\Pi)$ is minimized and assume wlog that Γ has the minimum number of non-horizontal chords possible among the choices for Γ. Let $\widetilde{\Gamma}$ be the subset of the elements of Γ that are non-horizontal. Now, by Lemma 4.2.2, the elements of $\widetilde{\Gamma}$ are all column chords of P. Also, by the construction of $\widetilde{\Gamma}$ and by the properties of $\mathfrak{L}_{\text{red}}(P)$, every element of $\widetilde{\Gamma}$ intersects exactly one chord of $\mathfrak{L}_{\text{red}}(P)$.



For any simple polygon P there exists a curve C interior to P such that $\mathsf{C}_i(\Pi, C) \geq \mathsf{C}_i(\overline{\Pi}) - 2$ where $\mathsf{C}_i(\Pi)$ is minimized and $\overline{\Pi}$ is the partition of P produced by applying Algorithm 3 to P.
In this case $\mathsf{C}_i(\Pi, C) = \mathsf{C}_i(\overline{\Pi}) - 2 = 6$.

FIGURE 4.2.3. A partition Π of polygon P such that $\mathsf{C}_i(\Pi)$ is minimized

Let $C_{\min}$ be any y-monotone curve whose interior is contained in P such that $\mathsf{C}_i(\Pi, C_{\min}) = \mathsf{C}_i(\Pi)$. If $C_{\min}$ intersects no elements of $\overline{\Pi}$ then $C_{\min}$ is entirely contained in an element of $\pi_{\text{red}}(P)$. Then $\mathsf{C}_i(\Pi) = \mathsf{C}_i(\overline{\Pi}, C_{\min})$ by Lemma 4.2.8. Therefore we assume that $C_{\min}$ intersects some element of $\mathfrak{L}_{\text{red}}(P)$. Let r be any element of

$\pi_{\text{red}}(P)$. Consider the set of y-monotone curves whose interior is contained in P and that intersect r. Among the elements of this set let C be any element intersecting the maximum number of elements of Π and $\overline{C}$ be any element intersecting the maximum number of elements of $\overline{\Pi}$. We will show that $\mathbf{C}_i(\Pi, C) \geq \mathbf{C}_i(\overline{\Pi}, \overline{C}) - 2$. Then this theorem is proved since r was chosen arbitrarily.

Now r is the only element of $\mathfrak{L}_{\text{red}}(P)$ that $\overline{C}$ intersects by the properties of $\mathfrak{L}_{\text{red}}(P)$. Also, r is on the boundary of two $\pm x$-2^*-wind subpolygons of $\pi(P)$, such that one is x-2^*-wind and the other is $-x$-2^*-wind. Furthermore, one of these subpolygons has r as its base; call this polygon $\widehat{Q}$. We assume wlog that $\widehat{Q}$ is the $-x$-2^*-wind subpolygon. Let $\mathbb{Q}$ be the set of subpolygons determined by the intersection of the remaining (x-2^*-wind) subpolygon that is on or above the line through r and the half plane on or above the line through r. Let $\check{Q}$ be the element of $\mathbb{Q}$ with r on its boundary. Then $\check{Q}$ is also an x-2^*-wind polygon. Let Φ be the partition of $\widehat{Q}$ and $\check{Q}$ by the line through r and the elements of $\widetilde{\Gamma}$ that intersect r. Then each element of Φ is a $\pm x$-2^*-wind polygon and has a portion of the line through r as its base.

Claim 1: Let ϕ be any element of Φ. Then there exists a directed y-monotone curve C_ϕ, interior to ϕ except at its endpoints, whose source belongs to r, and that intersects at least $\mathbf{C}_i(\phi) - 1$ chords of Γ.

Proof: Let Γ_ϕ be the set of chords of ϕ that are either elements of Γ or portions thereof and Π_ϕ be the partition of ϕ induced by Γ_ϕ. We point out that portions of any element, say δ, of Γ may be a chord of ϕ if δ intersects a chord of $\mathfrak{L}_{\text{red}}(P) - \{r\}$ that is on the boundary of ϕ. Now $\mathbf{C}_i(\phi) \leq \mathbf{C}_i(\Pi_\phi)$, so C_ϕ must exist.

We assume wlog that the endpoints of C and $\overline{C}$ belong to convex cusps of P. Let $\check{p}$ be the endpoint of C above r and let $\widehat{p}$ be the endpoint of C below r. Let $\check{\phi}$ be the element of Φ containing $\check{p}$ and $\widehat{\phi}$ be the element of Φ containing $\widehat{p}$. Let ν be the number of elements of $\widetilde{\Gamma}$ that r intersects; by definition, C intersects each of these

elements. Let $\check{m} = \mathbf{C}_i(\check{\phi})$ and $\hat{m} = \mathbf{C}_i(\hat{\phi})$. Then, by the definition of C and Claim 1, C intersects at least $\check{m} + \hat{m} + \nu - 2$ elements of Γ; in fact this number is exact. Therefore $\mathbf{C}_i(\Pi, C) = \check{m} + \hat{m} + \nu - 1$. See Figure 4.2.3.

Claim 2: $\check{m}$ ($\hat{m}$) is at least as large as the monotone cover number of any element of Φ above (below) r.

Proof: Let ϕ' be an element of Φ above r whose monotone cover number exceeds $\check{m}$ by some positive integer k. Then there exists a y-monotone curve interior to P that intersects $\check{m} + k + \hat{m} + \nu - 2$ elements of Γ by Claim 1. But this number exceeds $\mathbf{C}_i(\Pi, C) - 1 \leq \check{m} + \hat{m} + \nu - 2$, which is impossible by the definition of Γ. A similar argument applies to $\hat{m}$.

Now, from Claim 2 and Lemma 4.2.9, we have that $\mathbf{C}_i(\hat{Q}) \leq \hat{m} + \lfloor \log(\nu + 1) \rfloor$ and similarly $\mathbf{C}_i(\check{Q}) \leq \check{m} + \lfloor \log(\nu + 1) \rfloor$. Therefore $\mathbf{C}_i(\overline{\Pi}, \overline{C}) \leq \mathbf{C}_i(\hat{Q}) + \mathbf{C}_i(\check{Q}) \leq \check{m} + \hat{m} + 2\lfloor \log(\nu + 1) \rfloor$. Let $t = \mathbf{C}_i(\overline{\Pi}, \overline{C}) - \mathbf{C}_i(\Pi, C)$. Then $t \leq 2\lfloor \log(\nu + 1) \rfloor - \nu + 1$ and thus ν is $t \leq 2$. □

Based upon the last sentence of Theorem 4.2.10 we can make a number of statements about minimum monotone visibility covers. First we need a lemma.

LEMMA 4.2.11. *Let e be any element of $\mathfrak{L}(P)$. Then for some vertex v_i of $V(P)$, if v_i is used as start(P) then $e \in \mathfrak{L}_{\text{red}}(P)$.*

PROOF. Let ψ be a cusp of P such that $e \in \text{lids}(\psi)$ and let e' be the lid of ψ that is not e. Then there exists a convex cusp, say ψ_c, of P and a directed y-monotone curve C contained in or on P such that C intersects in order ψ_c, ($P^+(\psi)$ or $P^-(\psi)$), e', ψ, and e. Note that it cannot be that C crosses e. Now let start$(P) = v^+(\psi_c)$ and then let Q be the $\pm x$-2^*-wind polygon of $\pi_{\text{red}}(P)$ whose base is ψ_c. Now e can be reached but not crossed by C or any other similarly constrained curve so, since Q is the maximal $\pm x$-2^*-wind subpolygon of P with base ψ_c by Lemma 3.4.4, e is an edge of Q and thus $e \in \mathfrak{L}_{\text{red}}(P)$. □

The next three lemmas all follow from the last sentence of Theorem 4.2.10 and Lemma 4.2.11.

LEMMA 4.2.12. *Let Π be a partition of a simple polygon P whose monotone cover number is minimum. Then no chord of $\mathfrak{L}(P)$ is intersected by exactly six chords of $\widehat{\Gamma}(\Pi)$ or by more than seven chords of $\widehat{\Gamma}(\Pi)$.*

LEMMA 4.2.13. *Let P be a simple polygon. Then there exists a partition Π of P such that the monotone cover number of Π is minimum and no chord of $\mathfrak{L}(P)$ is intersected by more than four chords of $\widehat{\Gamma}(\Pi)$.*

In some situations we may wish to restrict the chords used to partition a polygon to those that are horizontal. This is the case, for example, when we deal with the circular ray shooting problem in Chapter 5. Thus following lemma is of interest.

For any polygon P we denote

$$\mathfrak{C}_{\overline{i}}(P) = \min\{\mathfrak{C}_i(\Pi) : \Pi \text{ is a partition of } R \text{ induced by elements of } \mathfrak{L}(P)\}.$$

We refer to $\mathfrak{C}_{\overline{i}}(P)$ as the horizontal monotone cover number of P.

LEMMA 4.2.14. *Let P be a simple polygon. Then the horizontal monotone cover number of the partition constructed by applying Algorithm 3 to P exceeds $\mathfrak{C}_{\overline{i}}(P)$ by at most one.*

It is straightforward to show that the above three lemmas apply not only to the elements of $\mathfrak{L}(P)$ but to any horizontal chord of P.

If we restrict ourselves to horizontal chords then it is also possible to construct a partition of P whose monotone cover number is minimized. First, we apply Algorithm 3. Let Π be the partition returned by Algorithm 3 when applied to P. Now, for each chord of $\mathfrak{L}_{\text{red}}(P)$, we can compute the maximum number of elements of Π intersected by any y-monotone curve interior to P that intersects the chord. Assume that this has been computed for each for each chord of $\mathfrak{L}_{\text{red}}(P)$ and that the maximum number

Algorithm 4

Input: A simple polygon P.
Output: A MSVP Π of P such that $\mathbf{C}_{\overline{\iota}}(P)$ is minimized.

1: Compute $\pi(P)$.
2: Let Π be the partition that results from applying Algorithm 3 to P.
3: Compute oldMax, the maximum number of chords of $\widehat{\Gamma}(\Pi)$ intersected by any y-monotone curve interior to P. (A blocking sweep may be used).
4: $\Gamma := \emptyset$. Set the reflex cusps of P to unmarked.
5: Do a blocking sweep of $\pi_{\text{red}}(P)$: for each $R \in \pi_{\text{red}}(P)$ do
  Do a blocking sweep of $\pi(R)$: for each $Q \in \pi(R)$ do
    if Q has a window, say g, such that $g \in \mathfrak{L}_{\text{green}}(Q)$ **then**
      $w(\text{door}(Q)) := w(g)$.
    else
      bWin := windows $(Q) \cap \mathfrak{L}_{\text{blue}}(P)$.
      if bWin $= \emptyset$ **then** $w(\text{door}(Q)) := 1$.
      else
        weight := $\max\{w(\gamma_i) : \gamma_i \in \text{bWin}\}$.
        $\gamma_b :=$ the rightmost element of bWin such that $w(\gamma) =$ weight.
        $\Gamma_{\text{new}} := \Gamma_{\text{new}} \cup \text{bWin} - \{\gamma_b\}$.
        $w(\text{door}(Q)) := w(\gamma_b)$.
        if $|\{\gamma_i : \gamma_i \in \text{bWin and } w(\gamma_i) = \text{weight}\}| > 1$ **then**
          $w(\text{door}(Q)) := w(\text{door}(Q)) + 1$.
        if $w(\text{door}(Q)) \geq$ oldMax then return(Π).
      rWin := windows $(Q) \cap \mathfrak{L}_{\text{red}}(P)$.
      if rWin $\neq \emptyset$ **then**
        nowMax := $w(\text{door}(Q)) + 1 + \max\{w(\text{door}(r)) : r \in \text{rWin}\}$.
        if nowMax > oldMax then return (Π).
        Mark each element r_j of rWin for which $w(r_j) + w(\text{door}(Q)) =$ oldMax.
        saturated := $|\{\gamma_i : \gamma_i \in \text{rWin and } \gamma_i \text{ is marked}\}|$.
        if saturated > 1 **then** return(Π).
        if saturated $= 1$ **then**
          $w(\text{door}(Q)) := w(\text{door}(Q)) + 1$.
          $\gamma_r :=$ the only marked element of rWin.
          $\Gamma := \Gamma \cup \{\text{door}(Q)\} - \{\gamma_r\}$.
6: Let Π be the partition of P induced by Γ.
7: Return(Π).

intersected by any monotone curve interior to P is m. Then $m - 1 \leq \mathbf{C}_{\overline{\iota}}(P) \leq m$ by Lemma 4.2.14. If $\mathbf{C}_{\overline{\iota}}(P) = m - 1$ then for each element r_i of $\mathfrak{L}_{\text{red}}(P)$ for which a y-monotone curve interior to P can intersect m elements of Π we must replace

r_i by another horizontal chord and the only choice is the element of $\mathfrak{L}_{\text{green}}(P)$ that determined r_i. Let Γ_1 be the set of lids after making these replacements. Note that if in constructing Γ_1 the same green lid is used to replace more than one red lid then $\mathfrak{C}_{\overline{\mathfrak{L}}}(P) = m$. Once this step has been completed we can recalculate the maximum number of chords of Γ_1 intersected by any y-monotone curve interior to P. If this number exceeds m then $\mathfrak{C}_{\overline{\mathfrak{L}}}(P) = m$. If this number equals m then we replace each red chord r_j of $\mathfrak{L}_{\text{red}}(P) \cap \Gamma_1$ that can be intersected by some y-monotone curve, say C_j interior to P such that C_j intersects m elements of Γ_1 by the element of $\mathfrak{L}_{\text{green}}(P)$ that determined r_j. Let Γ_2 be set of chords after making these replacements. Again in making these replacements we cannot use the same green lid twice. This process can be repeated until either a set of lids is selected that induces a partition whose monotone cover number is $m-1$ or it is determined that no such partition exists using only elements of $\mathfrak{L}(P)$. This algorithm can be implemented to run in $O(n^2)$ time. However, by using a blocking sweep, the cost of constructing a partition of P whose horizontal monotone cover number is minimized can be reduced to linear. This is achieved with Algorithm 4.

4.3. A Size Sensitive Monotone Visibility Partition Class

Although Algorithm 2 builds a MSVP whose y-monotone cover number is minimum in the case of a $\pm x$-2*-wind polygon and this algorithm can be used to construct a MSVP of a simple polygon whose y-monotone cover number is within two of the minimum, it fails to consider the size of the subpolygons constructed. Thus, for problems in which the size of the subpolygons is a concern, it is sometimes better to partition the polygon differently.

Algorithm 5 constructs a MMSP that considers the size of the subpolygons. In linear time and space it constructs an MSVP, for an x-2*-wind polygon, such that the sequence of y-monotone subpolygons intersected by any x-1-wind chain interior

to the polygon have sizes that are reduced by at least half with each step along the sequence.

Algorithm 5

Input: A simple $\pm x$-2*-wind polygon P. Note that the endpoints of the chords of $\mathfrak{L}(P)$ are assumed to be vertices of P.

Output: A MSVP Π of P.

1: $\Gamma := \emptyset$.
2: Compute $\pi(P)$.
3: for each $Q \in \pi(P)$ such that windows$(Q) \neq \emptyset$ do
    weight := $\max\{\text{nTrap}(P_{\gamma_i}) : \gamma_i \in \text{windows}(Q)\}$.
    $\gamma :=$ the rightmost element of windows(Q) such that $w(\gamma) =$ weight.
    $\Gamma := \Gamma \cup \text{windows}(Q) - \{\gamma\}$.
4: Let Π be the partition of P induced by Γ.
5: Return(Π).

(We remind the reader that we denote by P_γ the subpolygon of P determined by chord γ that does not include $\psi_x(P)$; see Chapters 2 and 3.)

Consider, for example, a partition Π with this property for some x-2*-wind polygon P with n vertices. Let T be the dual of Π where each subpolygon of Π corresponds to a vertex of T and two vertices of T are joined by an edge if the corresponding subpolygons of P are coincident to an element of the set of lids that induces Π. As we shall show, if Π is the MSVP of P constructed by Algorithm 5 then T has a depth of at most $\log n$. Now, since a vertex of T can have as many as $O(n)$ neighbors, searching for a leaf node starting at the root of T can take $O(\log^2 n)$ time if binary search is applied to find a child at each internal node. However, by assigning weights to nodes according to the size of the subtree rooted at the node and constructing a weighted binary tree using these weights at each internal node, the cost of such a search can be reduced to $O(\log n)$ time. If, instead, we use Algorithm 2 to construct Π then T would still have a depth of at most $\log n$. However, constructing a weighted binary tree at each internal node would no longer guarantee the reduction of the cost

of the search to $O(\log n)$ time. It is to achieve this saving in applying a search that we need Algorithm 5.

Algorithm 5 tends to produce a partition with a small number of large (by vertex count) y-monotone polygons and a large number of small y-monotone polygons. It is most beneficial when an individual y-monotone polygon can be processed (after preprocessing) in sublinear time.

It is also possible to produce a partition in which the sizes of the y-monotone subpolygons tend to be more balanced. It may be quite useful to do so when processing an individual y-monotone subpolygon requires superlinear time. However the partition generated is not guaranteed to be a MSVP and thus an algorithm that constructs such a partition is not of major interest in this thesis.

Let P be a simple $\pm x$-2*-wind polygon with n vertices.

By using Algorithm 5 instead of Algorithm 2 in the circular ray shooting problem we are able reduce the cost of a query for that problem from $O(\log^3 n)$ to $O(\log^2 n)$. To achieve this we also need to use the algorithm for weighted binary trees given in Appendix A.

We also believe that, using Algorithm 5, we can develop an optimal algorithm for simple ray shooting; research on this problem is ongoing.

Algorithm 5 differs from Algorithm 2 only in the criteria used when selecting chords from those available. We use nTrap(P) to denote the number of trapezoids in a horizontal trapezoidization of P.

Let $T_{\text{trap}}(P)$ denote an embedding of the dual of the horizontal trapezoidization of P into into the plane; we embed $T_{\text{trap}}(P)$ into the plane such that $T_{\text{trap}}(P)$ is interior to P and each vertex of $T_{\text{trap}}(P)$ is interior to the trapezoid corresponding to it. One way of constructing the output of Algorithm 5 is as follows. First construct $T_{\text{trap}}(P)$ and then $\pi(T_{\text{trap}}(P))$ as described in Section A.4 with the modifications of Subsection

A.4.1 included. We now partition P into a set of y-monotone subpolygons Π such that each path in $\pi(T_{\text{trap}}(P))$ is contained in a unique subpolygon of Π.

This construction may be done without explicitly constructing $\pi(T_{\text{trap}}(P))$; it may instead be constructed directly from $\pi(P)$. In fact this is how Algorithm 5 actually works. An example output of Algorithm 5 is shown in Figure 4.2.2.

We denote by $\mathfrak{L}^w(P)$ the subset of $\mathfrak{L}(P)$ selected by applying Algorithm 5 to P. We further denote by $\pi^w(P)$ the set of y-monotone subpolygons of P determined by $\mathfrak{L}^w(P)$. The key advantage of Algorithm 5 over Algorithm 2 is made clear by the following observation which follows directly from the rules for constructing $\pi(T)$.

OBSERVATION 4.3.1. *Let $\gamma \in \mathfrak{L}^w(P)$ and $Q \in \pi^w(P)$ be such that $\gamma \in windows(Q)$. Then $nTrap(P_\gamma) < 2 * nTrap(P_{door(Q)})$.*

Observation 4.3.1 provides us with a useful property of $\pi^w(P)$. In particular, this property is important for the partition to have if weighted binary trees are to be used; and we will use Algorithm 5 in combination with the weighted binary trees developed in Subsection A.4.1 in our solution to the circular ray shooting problem.

However we still need to establish that $\pi^w(P)$ is a MSVP of P.

LEMMA 4.3.2. *Algorithm 5 constructs a MSVP Π such that $\mathbf{C}_\dagger(\Pi) \leq \lfloor \log n \rfloor$, $\mathbf{C}_\imath(\Pi) \leq \lfloor \log n \rfloor$, $\mathbf{C}_\circ(\Pi) \leq 2\lfloor \log n \rfloor - 2$, and $\mathbf{C}_\diamond(\Pi) \leq \lfloor \log n \rfloor$.*

PROOF. Let $T = T_{\text{trap}}(P)$. We observe that $|T| \leq n - 1$. Therefore any y-monotone curve interior to P can intersect at most $\lambda(T) \leq \lfloor \log(n) \rfloor$ elements of $\pi^w(P)$ by Lemma A.4.4 and the definition of $\pi^w(P)$. Therefore $\mathbf{C}_\imath(\Pi) \leq \lfloor \log n \rfloor$.

Now assume wlog that P is x-2*-wind.

Let C be any curve that is interior to P and partitionable into three y-monotone curves by removing a top point and a bottom point. Now the shorter of the two subcurves coincident to the bottom point can only intersect elements of $\pi^w(P)$ intersected by the longer subcurve and thus we do not count the chords of $\pi^w(P)$

intersected by the shorter subcurve. The remaining subcurves between them can intersect at most $\lfloor \log n \rfloor + \lfloor \log n \rfloor - 1 = 2 \lfloor \log n \rfloor - 1$ elements of $\pi^w(P)$ by Lemmas A.4.4 and the definition of $\pi^w(P)$. Furthermore, one of these elements of $\pi^w(P)$, the one containing the top point of C, must have been counted twice. Therefore at most $2 \lfloor \log n \rfloor - 2$ elements of $\pi^w(P)$ can be intersected by C. But C can be any circular arc so $\mathbf{C}_{\bigcirc}(\Pi) \leq 2 \lfloor \log n \rfloor - 2$. A similar argument establishes that $\mathbf{C}_{\diamond}(\Pi) \leq \lfloor \log n \rfloor$. □

Now let P be a simple polygon. We denote by $\mathfrak{L}^w(P)$ the union of the subsets of $\mathfrak{L}(P)$ selected by applying Algorithm 5 to each element of $\pi_{\text{red}}(P)$ together with $\mathfrak{L}_{\text{red}}(P)$. We further denote by $\pi^w(P)$ the set of y-monotone subpolygons of P determined by $\mathfrak{L}^w(P)$. From the previous Lemma and Lemmas 3.4.7, 3.4.10, and 3.4.11, the following result is easily shown:

LEMMA 4.3.3. *Let P be a simple polygon and let $\Pi = \pi^w(P)$. Then Π is a MSVP such that $\mathbf{C}_i(\Pi) \leq 2 \lfloor \log n \rfloor - 1$, $\mathbf{C}_{\bigcirc}(\Pi) \leq 4 \lfloor \log n \rfloor - 5$, and $\mathbf{C}_{\diamond}(\Pi) \leq 2 \lfloor \log n \rfloor - 1$.*

CHAPTER 5

Circular Ray Shooting

5.1. Introduction

In this chapter we will use monotone subpolygon visibility partitions or MSVPs from Chapter 4 to solve the circle shooting problem.

Given a directed circular arc ζ, we denote the starting point of ζ by ζ_o and the endpoint by ζ_e. We denote by $\hbar_\zeta(P)$ the first point on $\zeta - \zeta_o$ to intersect P in a traversal of ζ in the direction implied by ζ. If $(\zeta - \zeta_o) \cap P = \emptyset$ then $\hbar_\zeta(P)$ is $\emptyset$.

PROBLEM 5.1.1. *Preprocess a polygon P so that, for a query arc ζ, with ζ_o interior to P, one can quickly determine $\hbar_\zeta(P)$.*

The first solution for this problem was was given in [1] where it was shown that a polygon with n vertices can be preprocessed in $O(n \log^3 n)$ time and $O(n \log^3 n)$ space so that a circular ray shooting query can be done in $O(\log^4 n)$ time. This result was recently improved in [16] with an algorithm in which a polygon is preprocessed in $O(n \log^2 n)$ time and $O(n \log n)$ space so that a circular ray shooting query can be done in $O(\log^2 n)$ time.

In this chapter we present another solution to Problem 5.1.1. Our algorithm has the same space complexity and query complexity as the algorithm in [16]. The more basic form of our algorithm has the same preprocessing complexity as the algorithm in [16]. However, with some modifications, we are able to reduce the preprocessing time to $O(n \log n)$.

We also provide a number of other improvements over the algorithm in [16].

(1) The preprocessing and query components of our algorithms are simpler. This is the main advantage our algorithm.

(2) The constant coefficients of the preprocessing and space complexity of our algorithms are all smaller.[1]

(3) We have a more clear-cut distinction between the geometric and non-geometric components of our algorithm making both easier to apply to other problems.

The algorithm in [16] partitions an arc into components which can be rotated and flipped into a form we call semi-canonical arcs. This partition is crucial to their algorithm and also to ours.

Both the algorithm in [16] and our algorithm reduce the circular ray shooting problem to the special case that the arc is semi-canonical and the polygon is y-monotone. The major difference between the algorithms is in the method used to achieve this reduction. In [16] the authors develop a subpolygon cover class of simple polygons which they call a Hierarchical Vertical Decomposition. In the worst case, this decomposition requires $O(n \log n)$ time and space to construct. We instead use a monotone subpolygon visibility partition or MSVP of the polygon using the algorithms in Sections 4.2 and 4.3. These MSVPs are much simpler to construct and use than a Hierarchical Vertical Decomposition. They also use a factor of $O(\log n)$ less space than a Hierarchical Vertical Decomposition and require a factor of $O(\log n)$ less time to construct.

In order to perform a ray shooting query with a semi-canonical arc, the algorithm in [16] applies an algorithm that detects if the arc intersects a y-monotone polygon. This requires the construction of Voronoi diagrams for the points and edges of each y-monotone polygon in the Hierarchical Vertical Decomposition.[2] These Voronoi

[1]The constant coefficients for the query complexity are about the same. We believe implementations are needed to determine which is faster in practice.

[2]Actually, in [16] these polygons are not y-monotone but x-monotone. We have taken the liberty of applying a 90 degree rotation.

diagrams require an additional $O(n \log n)$ space and $O(n \log^2 n)$ time to construct. Once these diagrams have been constructed the Hierarchical Vertical Decomposition can be dispensed with.

Using these Voronoi diagrams, detecting if a semi-canonical arc intersects an m-vertex y-monotone polygon requires $O(\log m)$ time. In order to find the first intersection point of the arc with a simple polygon this algorithm must be applied $O(\log n)$ times. Thus, in total, $O(\log^2 n)$ time is required to find this intersection.

In order for our algorithm to perform a ray shooting query with a semi-canonical arc, it applies an algorithm that finds the first intersection of the arc with a y-monotone polygon. In order to find this intersection $O(\log m)$ applications of the algorithm in **[16]** (described above) are required to detect that the arc intersects a y-monotone polygon where the polygon has m vertices. This is done by recursively partitioning the y-monotone polygon into smaller y-monotone subpolygons and then recursively searching this structure during a query. Now, in total, $O(\log^2 m)$ time is required to find an intersection of a semi-canonical arc with the y-monotone polygon. In order to apply this detection algorithm we too must construct a set of Voronoi diagrams which, in our case, requires $O(m \log m)$ space and $O(m \log^2 m)$ time to construct. However, by modifying the algorithm in **[16]** that constructs these Voronoi diagrams, we are able to reduce the time needed to construct them to $O(m \log m)$.

In order to find the first intersection of a semi-canonical arc with a simple polygon, our algorithm finds the first intersection of the arc with $O(\log n)$ y-monotone subpolygons. Thus, to find the first intersection of the arc with a simple polygon requires $O(\log^3 n)$ time in total. However, by using the weighted binary tree structure defined in Appendix A.4, we reduce the cost of finding this intersection to $O(\log^2 n)$ time. Constructing the Voronoi diagrams for each y-monotone polygon in our $MSVP$ requires $O(n \log n)$ space and preprocessing time.

At this point the reader might be tempted to conclude that the algorithm in [16] is simpler than ours. But the opposite is true. The reason is in the relative difficulty in constructing and using the Hierarchical Vertical Decomposition (a cover) compared to the simplicity in constructing and using the MSVPs of Chapter 4 and the weighted binary tree in Appendix A.

In Section 5.2 we discuss the algorithm in [16] for decomposing an arc into semi-canonical components.

In Section 5.3 we develop an algorithm to solve the circular ray shooting problem efficiently for semi-canonical arcs on y-monotone polygons.

In Section 5.4 we modify one of the algorithms from Section 5.3 so that the cost of the preprocessing stage of the algorithm is reduced from $O(m \log^2 m)$ to $O(m \log m)$ for an m vertex y-monotone polygon.

In Section 5.5 we show how to generalize the results of the previous section to the case that the polygon is x-2*-wind. This is achieved while maintaining the same asymptotic complexities as for y-monotone polygons by using weighted binary trees. Then we present our results for simple polygons. Finally, we discuss some useful optimizations that can be applied to our algorithms.

5.2. A Reduction of the Circular Ray Shooting Problem

For any circular arc ζ we denote the circle whose boundary contains ζ by circle (ζ). We further denote by radius (ζ) the radius of circle (ζ) and by center (ζ) the center of circle (ζ).

In [16] the authors assume wlog that a query arc ζ is a subarc of the top quarter arc of circle(ζ) and is directed from left to right with center (ζ) below ζ. The authors base this assumption on the fact ζ can be cut into at most five components, each of which is either the top quarter circle, the bottom quarter circle, the left quarter circle, or the right quarter circle of circle (ζ) or a portion thereof. They then solve the circular

ray shooting problem for input ζ by solving the problem for each component of ζ in turn in the order implied by the direction of ζ. For each component, appropriate reflections of both the component and P about the x-axis, the y-axis, the line $y = x$ or combinations of these reflections are applied, if necessary, so that the component satisfies the stated assumption.[3] The key reason for making this decomposition and transformation is so that the range of the slopes of the tangents to the component arcs is between -1 and 1. This property is crucial to one of the subalgorithms in their paper.

We use the same methods to reduce the problem to a subarc of a quarter arc except that for consistency with the rest of this thesis we use the leftmost quarter arc of a circle instead of the topmost quarter arc.

We remind the reader that, for a y-monotone polygon Q, the cusps are $\psi_x(Q)$ and $\psi_{-x}(Q)$; see Chapter 3.

We say that an arc ζ is a canonical arc of a y-monotone polygon Q if ζ is a subarc of the left quarter of a circle, ζ is clockwise directed, ζ_o belongs to the interior of $\psi_x(Q)$ and ζ_e belongs to the interior of $\psi_{-x}(Q)$. We say, instead, that ζ is a semi-canonical arc of Q if ζ is a subarc of the left quarter of a circle, ζ is clockwise directed, and ζ_o is interior to Q.

In order to perform the necessary transformations on an arc to make it canonical or semi-canonical we must also apply these transformations to the input polygon during preprocessing. As a result, two copies of the main data structure are required, both for their algorithm and ours. In our case this means we need to construct a y-MSVP and an x-MSVP of the input polygon. Also, for each x-monotone or y-monotone subpolygon in the y-MSVP or x-MSVP respectively, we need to construct

[3] In fact the method used in [16] is slightly different than described here but the differences are immaterial.

four Voronoi diagrams, not the two implied by considering only the canonical arc case. These additional Voronoi diagrams are needed in [16] as well.

5.3. Intersection of an Arc with a y-Monotone Polygon

In this section we will develop an algorithm to solve the circular ray shooting problem in the special case of a canonical arc of a y-monotone polygon. Many of the results in this section are from [1] or [16]. We have, however, made modifications to these results where necessary and added some results of our own.

5.3.1. Detecting an Intersection. Consider the following problem:

PROBLEM 5.3.1. [1] *Preprocess a chain C so that, for a query circle ζ, one can quickly determine whether or not ζ intersects C.*

The following observation leads to an efficient algorithm for solving Problem 5.3.1:

OBSERVATION 5.3.2. [1] *Let ζ be a circle, C be a chain, p_{near} be a point on C with minimum distance to center(ζ), and p_{far} be a point on C with maximum distance to center(ζ). Then ζ intersects C if and only if $p_{near} \leq$ radius$(\zeta) \leq p_{far}$.*

This observation suggests the use of Voronoi diagrams to solve Problem 5.3.1. For any chain C we denote by $\mathrm{Vor}_{\mathrm{near}}(C)$ the nearest site Voronoi diagram of the edges of C and denote by $\mathrm{Vor}_{\mathrm{far}}(C)$ the farthest point Voronoi diagram of the vertices of C. For any Voronoi diagram $\mathcal{V}$ we denote by $\mathrm{dist}(\mathcal{V}, p)$ the distance from point p to the object (either a vertex or an edge) that determines the region of $\mathcal{V}$ containing p. Both $\mathrm{Vor}_{\mathrm{near}}(C)$ and $\mathrm{Vor}_{\mathrm{far}}(C)$ can be preprocessed for point location so that the region containing any point can be found in $O(\log n)$ time ([43], [52]). Thus, after preprocessing, for any point p, both $\mathrm{dist}(\mathrm{Vor}_{\mathrm{near}}(C), p)$ and $\mathrm{dist}(\mathrm{Vor}_{\mathrm{far}}(C), p)$ can be computed in $O(\log |C|)$ time. With these tools we now present Algorithm 6 which is our version of an algorithm in [1] that solves Problem 5.3.1.

Algorithm 6 (modified from [1])

Preprocessing:
Input: A chain C.
1: Compute $\text{Vor}_{\text{near}}(C)$ and $\text{Vor}_{\text{far}}(C)$.
2: Preprocess $\text{Vor}_{\text{near}}(C)$ and $\text{Vor}_{\text{far}}(C)$ for efficient point location.
Query:
Input: A circle ζ.
Output: 1 if ζ intersects C; 0 otherwise.
1: Compute $d_c := \text{dist}(\text{Vor}_{\text{near}}(C), \text{center}(\zeta))$.
2: Compute $d_f := \text{dist}(\text{Vor}_{\text{far}}(C), \text{center}(\zeta))$.
3: Return 1 if $d_c \leq \text{radius}(\zeta) \leq d_f$. Return 0 otherwise.

We need a more complex version of Algorithm 6 to solve a more complex version of Problem 5.3.1.

PROBLEM 5.3.3. *Preprocess a y-monotone polygon Q whose cusps are edges so that, for a query canonical arc ζ of Q, one can quickly determine whether or not ζ intersects Q other than at the interior of its cusps.*

This problem is solved in **[16]**. As in the solution to Problem 5.3.1, Voronoi diagrams are used. However there are complications in using the farthest point Voronoi diagram. These problems relate to the fact that we are dealing with a circular arc rather than a circle. For Problem 5.3.1 we knew that the circle intersected the chain if and only if part of the chain was not interior to the circle and part of the chain was not exterior to the circle. However, for a circular arc ζ, having circle(ζ) intersect a y-monotone polygon does not guarantee that ζ does.

Let ζ be a canonical arc of a y-monotone polygon Q whose cusps are edges. Now, we know that ζ_o belongs to the interior of $\psi_x(Q)$ and ζ_e belongs to the interior of $\psi_{-x}(Q)$. This helps considerably.

We remind the reader that $\text{chn}^-(\psi_x(Q))$ and $\text{chn}^+(\psi_x(Q))$ denote the left and right y-monotone chains of Q joining $\psi_x(Q)$ and $\psi_{-x}(Q)$ respectively; see Chapter 3. We use Q_l to denote $\text{chn}^-(\psi_x(Q))$ and use Q_r to denote $\text{chn}^+(\psi_x(Q))$.

Now, since ζ is canonical, at least part of Q_l is exterior to circle(ζ). Furthermore, since Q_l is y-monotone and bounded by the cusps of Q, we have that Q_l intersects circle(ζ) if and only if Q_l intersects ζ. So there are no additional difficulties dealing with Q_r and its edge Voronoi diagram.

Dealing with Q_r is more difficult. It may be that Q_r intersects circle(ζ) even though Q_r does not intersect ζ. This can occur if one or more of the vertices of Q_r are sufficiently far enough to the right of ζ to be exterior to circle(ζ). We deal with this difficulty by using an algorithm which is a variation on the algorithm in **[16]**. Our variation has two advantages:

(1) If $\psi_x(Q)$ and $\psi_{-x}(Q)$ are sufficiently large, it removes from consideration all vertices that cannot be a farthest point from some canonical arc of Q.

(2) It works well with the algorithm we develop in Section 5.4 to reduce the cost of constructing the farthest point Voronoi diagrams of Q from $O(m \log^2 m)$ to $O(m \log m)$ time where $|Q| = m$.

Let D be a y-monotone chain with a single highest vertex and a single lowest vertex. We use $\ell(D)$ to denote the leftmost vertex on D with ties broken by (arbitrarily) choosing the lowest vertex. Let H be the convex hull of D. Note that, since H is a polygon, it is directed counterclockwise and its edges are directed accordingly; see Section 3.2. We call the subchain of H whose edges have directions between π and 2π the left hull of D and denote it by $L(D)$. We denote by D^c the subchain of H whose edges have directions strictly between $5\pi/4$ and $7\pi/4$. If $D = Q_r$ then we denote $Q_r^c = D^c$. See Figure 5.3.1.

Now, if ζ is to the left of the vertical line through $\ell(Q_r)$ then clearly ζ does not intersect Q_r.

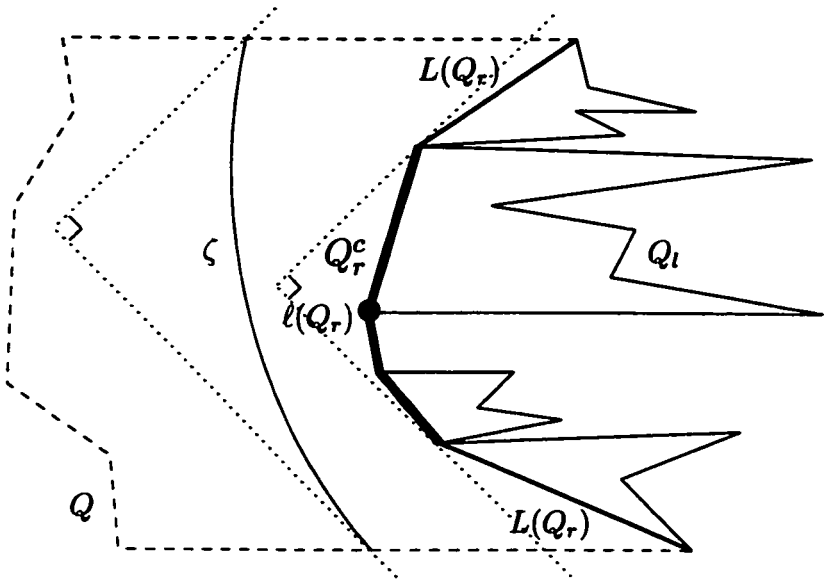


FIGURE 5.3.1. Chain Q_r^c of a y-monotone polygon Q

If, instead, any part of ζ is not to the left the vertical line through $\ell(Q_r)$ then we claim that ζ intersects Q_r if and only if:

$$\text{dist}(\text{Vor}_{\text{far}}(Q_r^c), \text{center}(\zeta)) \geq \text{radius}(\zeta).$$

If this claim were untrue it could fail for only two reasons:

(1) Some vertex v on $L(Q_r)$ is to the left of ζ yet $\text{dist}(\text{Vor}_{\text{far}}(Q_r^c), \text{center}(\zeta)) < \text{radius}(\zeta)$. Then v is on a subchain, say C, of $L(Q_r)$ such that exactly one vertex (an endpoint) of C, say p, is a vertex of Q_r^c and the other endpoint, say q, of C is an endpoint of $L(Q_r)$. For any point t on C let $\text{dist}(t)$ be the horizontal distance from t to a point on ζ where this distance is negative if t is to the left of ζ. Then $\text{dist}(q) > 0$ since ζ is a canonical arc of Q. Furthermore, by the construction of C and the fact that ζ is canonical, the slopes of the tangents of ζ and C are such that the function dist is monotonically increasing as we move along C from p to q. But this contradicts that $\text{dist}(v) \leq 0$. See Figure 5.3.1.

(2) Some vertex u on $L(Q_r)$ is to the right of circle(ζ) yet $\zeta \cap Q_r = \emptyset$. Assume wlog that u is above $\ell(Q_r)$. Let λ_h be the horizontal line through $\psi_{-x}(Q)$. Let p_ℓ be the intersection of the vertical line through $\ell(Q_r)$ and λ_h. Let λ_ℓ be the line through $\ell(Q_r)$ with a slope of one and let q_ℓ be the intersection point of λ_ℓ with λ_h. Let $\overline{\zeta}$ be the extension of ζ to a full left quarter circle. Let $p_{\overline{\zeta}}$ be the intersection point of the vertical line through the endpoints of $\overline{\zeta}$ with λ_h. Let p_ζ be the point where circle(ζ) intersects λ_h for the first time and let q_ζ be the point where circle(ζ) intersects λ_h for the second time where we traverse circle(ζ) in the clockwise direction starting from ζ_o. Let d_y be the vertical distance between $\psi_x(Q)$ and $\psi_{-x}(Q)$. Now, since ζ is a canonical arc of Q, the distance from $p_{\overline{\lambda}}$ to q_ζ is at least d_y. Also, by construction the distance from to p_ℓ to q_ℓ is less than d_y. But then, since p_ℓ is to the left of $p_{\overline{\lambda}}$, it must be that q_ℓ is to the left of q_ζ. But clearly u is to the left of λ_ℓ so u cannot be to the right of circle(ζ) if $u \in V(Q_r^c)$. Thus $u \in V(L(Q_r)) - V(Q_r^c)$. See Figure 5.3.2.

Note that if ζ were not constrained to be a subarc of the left quarter arc of a circle then this claim would not in general hold; thus the need for this constraint.

This solution to Problem 5.3.3 is given in point form in Algorithm 7.

Algorithm 7 (modified from [16]).

Preprocessing:
Input: A y-monotone polygon Q.
1: Compute $\text{Vor}_{\text{near}}(Q_l)$ and $\text{Vor}_{\text{far}}(Q_r^c)$.
2: Preprocess $\text{Vor}_{\text{near}}(Q_l)$ and $\text{Vor}_{\text{far}}(Q_r^c)$ for efficient point location.
Query:
Input: A canonical arc ζ of Q.
Output: 1 if $\zeta \cap (Q_l \cup Q_r) \neq \emptyset$; 0 otherwise.
1: Compute $d_c := \text{dist}(\text{Vor}_{\text{near}}(Q_l), \text{center}(\zeta))$ and
2: $d_f := \text{dist}(\text{Vor}_{\text{far}}(Q_r^c), \text{center}(\zeta))$.
3: Return 0 if $d_c \leq \text{radius}(\zeta) \leq d_f$; Return 1 otherwise.

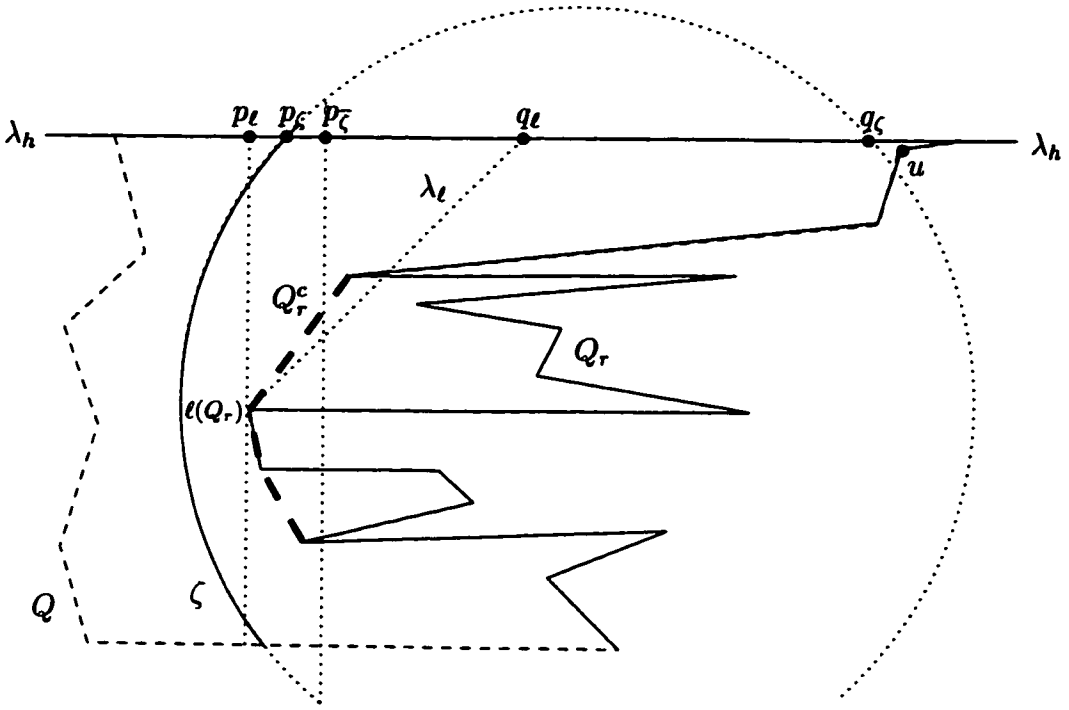


FIGURE 5.3.2. No vertex on Q_r^c is to the right of circle (ζ)

Our algorithm for determining which points of Q_r to include in the farthest point Voronoi diagram is optimal in the following sense.

LEMMA 5.3.4. *Let Q be a simple y-monotone polygon. If, for every edge e_i of Q_r^c, the line through e_i intersects both $\psi_x(Q)$ and $\psi_{-x}(Q)$ then for every site s_i of $Vor_{far}(Q_r^c)$ there exists a canonical arc ζ_i such that s_i is the only vertex of Q_r that is on or to the left of ζ_i.*

PROOF. Let v be any vertex of Q_r^c and let l be the line through the midpoints of the two edges of Q_r^c incident to v (The case that v has fewer than two incident vertices can be handled similarly.). Then l is an arc of infinite radius, every vertex of Q_r^c is to the right of l except v, and by construction l is a canonical arc. □

With some additional effort we can remove from consideration all vertices of Q_r^c that cannot be to the right of some canonical arc of Q_r. This can be done in $O(|Q|)$ time.

5.3.2. Circular Ray Shooting in Monotone Polygons. We now tackle the more general problem of determining where (if anywhere) a semi-canonical arc intersects a y-monotone polygon.

PROBLEM 5.3.5. *(modified from* **[16]***) Given a y-monotone polygon Q, preprocess it so that, for any query semi-canonical arc ζ of Q, one can quickly find $\hbar_\zeta(Q)$.*

We will solve this problem by transforming it into Problem A.1.1 and then solve it using the algorithm that solves Problem A.1.1, namely Algorithm 15.

We remind the reader that we denote the set of trapezoids of a horizontal trapezoidization of a polygon P by $\text{trap}(P)$ and the set of chords that induces partition $\text{trap}(P)$ of P by $\widehat{\Gamma}(\text{trap}(P))$. We refer the top and bottom sides of any element of trap (P) that is not an edge or vertex of P as *gates*. Note that a gate always contains a chord of $\widehat{\Gamma}(\text{trap}(P))$ but may contain many chords of $\widehat{\Gamma}(\text{trap}(P))$ and many edges or vertices of P. Note also that a strictly y-monotone curve can pass between two adjacent trapezoids of $\text{trap}(P)$ without intersecting P if and only if intersects the interiors both of the gates that separate the trapezoids. In the special case that P is y-monotone, a gate contains only one chord of $\widehat{\Gamma}(\text{trap}(P))$ but may contain as many as two edges of P if the endpoints of the cord are also the endpoints of horizontal edges of P.

Let Q be a y-monotone polygon with m vertices. Our first step in solving Problem 5.3.5 for Q is to construct $\text{trap}(Q)$. If R is a y-monotone polygon then we call the result of applying the preprocessing stage of Algorithm 7 to R a hatted polygon and denote it by $\widehat{R}$. Now, for each trapezoid t_i of $\text{trap}(Q)$, we compute $\widehat{t_i}$. Next we treat these hatted trapezoids as the elements of B in Problem A.1.1. We assume that the

elements of trap(Q) are ordered and indexed by height with t_0 on the bottom. This mapping is such that element b_i of B corresponds to $\widehat{t_i}$, $0 \leq i < |\text{trap}(Q)|$.

The next step is to determine the input value s of Problem A.1.1. First we determine the element of trap(Q) containing ζ_o. This can be done in $O(\log m)$ time after linear preprocessing using point location [33]. Let this trapezoid be t_i; if ζ_o intersects two gates then we choose the higher trapezoid for t_i and set $s = i$.

The case that ζ_o is interior to t_i gives us special problems so we shall eliminate this case. We say that an arc δ is nearly canonical with respect to a y-monotone polygon T if δ is canonical with respect to T except that the requirement that $\delta_e \in \psi_{-x}(T)$ is removed. Now if ζ intersects t_i other than at the interior of its gates or doesn't intersect t_i at all then we compute and return $\hbar_\zeta(t_i)$ and we are done. Otherwise we remove from ζ the portion that is interior to t_i. Finally, we set $s = i + 1$.

Henceforth we will only need to deal with the case that ζ is nearly canonical or canonical with respect to any hatted y-monotone polygon against which it will be tested.

Now we start our search from t_s.

Next we need to specify the version of the operator $\dot{<}$ that we will use in Algorithm 15. We use as the first argument to operator "$\dot{<}$" a hatted polygon, say $\widehat{W}$. For the second argument we use a semi-canonical arc of Q, say ζ, such that some subarc of ζ is a canonical or nearly canonical arc of W. In the case that $\widehat{W}$ is an element of B, that is a hatted trapezoid, then we can test in $O(1)$ time if ζ intersects W. In this case operator "$\dot{<}$" returns true if and only if ζ intersects W only at the interior of its gates. In the case that $\widehat{W}$ is not an element of B, then "$\dot{<}$" tests if ζ intersects the interior of $\psi_{-x}(W)$. If it doesn't then "$\dot{<}$" returns false. Otherwise "$\dot{<}$" invokes the query part of Algorithm 7 with these arguments and returns the result of the query.

To complete our transformation, we need to determine the replacement for the operator "max" for the preprocessing stage of Algorithm 15. We denote this replacement by $\widehat{\max}$. By our definition of operator "$\dot{<}$", the input to $\widehat{\max}$ must be two hatted polygons and the output must also be a hatted polygon. Let Q_1 and Q_2 be two y-monotone polygons such that either $\psi_{-x}(Q_1) \subseteq \psi_x(Q_2)$ or $\psi_x(Q_2) \subseteq \psi_{-x}(Q_1)$ and let Q_3 be the smallest polygon whose interior encloses the interiors of Q_1 and Q_2. We will show that $\widehat{Q_3}$ is the value that we need $\widehat{\max}$ to return for input $\{\widehat{Q_1}, \widehat{Q_2}\}$. For this to be the case we need, for any any nearly canonical arc ζ of Q_3, that $\widehat{Q_3} \dot{<} \zeta$ if and only if:

(1) ζ is a nearly canonical arc of Q_1 and Q_2 and
(2) $\widehat{Q_1} \dot{<} \zeta$ and $\widehat{Q_2} \dot{<} \zeta$.

There are two cases to consider.

The first case is that ζ is a canonical arc of all of Q_1, Q_2, and Q_3. Then ζ does not intersect Q_3 other than at the interior of its cusps if and only if intersects neither Q_1 nor Q_2 except at the interior of their cusps. Thus $\widehat{Q_3} \dot{<} \zeta$ if and only if $\widehat{Q_1} \dot{<} \zeta$ and $\widehat{Q_2} \dot{<} \zeta$ in this case.

The remaining case is that ζ does not intersect $\psi_{-x}(Q_3)$. In this case operator $\dot{<}$ will return false when applied to at least one of $\widehat{Q_1}$ and $\widehat{Q_2}$ and will also return false when applied to $\widehat{Q_3}$. Thus $\widehat{Q_3} \dot{<} \zeta$ if and only if $\widehat{Q_1} \dot{<} \zeta$ and $\widehat{Q_2} \dot{<} \zeta$ in this case also.

Algorithm 8

Input: Two hatted y-monotone polygons $\widehat{Q_1}$ and $\widehat{Q_2}$ such that either $\psi_{-x}(Q_1) \subseteq \psi_x(Q_2)$ or $\psi_x(Q_2) \subseteq \psi_{-x}(Q_1)$.

Output: A hatted polygon $\widehat{Q_3}$ such that Q_3 is the smallest polygon enclosing Q_1 and Q_2.

1: Compute $\text{Vor}_{\text{near}}((Q_3)_l)$ by merging $\text{Vor}_{\text{near}}((Q_1)_l)$ and $\text{Vor}_{\text{near}}((Q_2)_l)$.
2: Compute $\text{Vor}_{\text{far}}((Q_3)_r^c)$ from scratch.

The reader may wonder what happens when ζ is not a nearly canonical arc of Q_3. In fact this does not occur. The reason is that the first application of the operator $\dot{<}$ is applied to a hatted polygon such that ζ is nearly canonical, and after that the algorithm ensures that each hatted polygon tested is nearly canonical. In part this is because, in the discussion above, ζ is a nearly canonical arc of Q_1 if and only if ζ is a nearly canonical arc of Q_3. We summarize the above discussion with Algorithm 8 which is a more detailed specification of Step 1 of Algorithm 7.

Thus the version of Algorithm 15 specified here solves Problem 5.3.5; note that this algorithm returns one less than the index of the trapezoid containing $\hbar_\zeta(Q)$ so as a final step we must find $\hbar_\zeta(Q)$ in this trapezoid. We incorporate all these changes to Algorithm 15 into Algorithm 9.

Algorithm 9

Preprocessing:

Input: A y-monotone polygon Q.

1: Construct $\text{trap}(Q)$. For each $t_i \in \text{trap}(Q)$ construct $\widehat{t_i}$.
2: Set $B = \{\widehat{t_0}, \widehat{t_1}, \ldots, \widehat{t_{n-1}}\}$ where $|\text{trap}(Q)| = n$.
3: Now perform the preprocessing stage of Algorithm 15 after replacing the max operator by $\widehat{\max}$.

Query:

Input: A semi-canonical arc ζ of Q.

Output: $\hbar_\zeta(Q)$.

1: Compute the trapezoid t_i containing ζ_o.
2: If $\hbar_\zeta(Q) \in t_i$ then return $\hbar_\zeta(Q)$.
3: Compute t_j such that some subarc of ζ is a nearly canonical arc of t_j and $j \in \{i, i+1\}$. If t_j does not exist then $\hbar_\zeta(Q)$ is on the boundary of t_i.
4: Apply the query part of Algorithm 15 with input j and ζ using the version of "$\dot{<}$" defined in this section. Let k be the value returned.
5: Determine whether $\hbar_\zeta(Q) \in t_k$ or $\hbar_\zeta(Q) \in t_{k+1}$.
6: return $\hbar_\zeta(Q)$.

The cost of applying the preprocessing stage of Algorithm 15 in this case is best seen if we consider the balanced tree version of Algorithm 15. This cost is dominated by the cost of constructing the Voronoi diagrams. It costs linear time to construct all

the edge Voronoi diagrams at a given level of the tree from the edge Voronoi diagrams of the level below it. Since the bottom level can be constructed in linear time and there are $\lceil \log |\text{trap}(Q)| \rceil + 1$ levels, the total cost is $O(m \log m)$ where $|Q| = m$. Constructing all the farthest point Voronoi diagrams at a given level from scratch requires $O(m \log m)$ time in the worst case. We denote by $\mathbf{V}_{\text{far}}(Q)$ the set of farthest point Voronoi diagrams constructed during the preprocessing stage of Algorithm 9 for input polygon Q. Thus $O(m \log^2 m)$ time is needed to construct $\mathbf{V}_{\text{far}}(Q)$.

By being smarter about the way $\text{Vor}_{\text{far}}((Q_3)_r^c)$ is constructed in Algorithm 8 we can reduce the time required to construct $\mathbf{V}_{\text{far}}(Q)$ to $O(m \log m)$. We will show how this is done in Section 5.4.

Next we consider the cost of a circular ray shooting query. Finding the trapezoid containing ζ_o, if not already known, costs $O(\log m)$ time using point location. Now each application of operator $\dot{<}$ requires $O(\log h)$ time where h is the size of the polygon corresponding to the first parameter. Therefore h has size at most $s + 1 - k$ where k is the trapezoid containing $\hbar_\zeta(Q)$. Thus, from the analysis of Algorithm 15, a query requires $O\left(\log^2 h\right)$ time.

We summarize the results of this section with the following result.

LEMMA 5.3.6. *We can preprocess an m vertex y-monotone polygon Q in $O(m \log^2 m)$ time and $O(m \log m)$ space so that, for a query semi-canonical arc ζ of Q, we can determine $\hbar_\zeta(Q)$ in $O(\log^2 m)$ time.*

5.4. Constructing $\mathbf{V}_{\text{far}}(Q)$ in $O(m \log m)$ Time

Let Q be a y-monotone polygon with m vertices. In the previous section, we required $O(m \log^2 m)$ time to construct $\mathbf{V}_{\text{far}}(Q)$. We required $O(m \log^2 m)$ rather than the $O(m \log m)$ time needed to construct the edge Voronoi diagrams of the polygon because we could not merge the farthest point Voronoi diagrams directly. In this section we will show how to do this and thus reduce this cost to $O(m \log m)$.

We exploit the fact that the sites in a farthest point Voronoi diagram of a point set are exactly the points on the corners of its convex hull. We point out that the farthest point Voronoi diagram of the vertices of a convex polygon can be constructed in linear time. This is so because of two facts. First, the Voronoi diagram of the vertices of a convex polygon can be constructed in linear time [2] and second, the problem of constructing the farthest point Voronoi diagram of a point set can be transformed into the problem of constructing a Voronoi diagram of the same set in linear time [41].

Let h be the total number of sites in of all the farthest point Voronoi diagrams in $\mathbf{V}_{\text{far}}(Q)$ where duplicates are counted each time. Note that h is $O(m \log m)$. Then we can construct $\mathbf{V}_{\text{far}}(Q)$ in $O(m+h)$ time if we can find the convex hull of the sites of each element of $\mathbf{V}_{\text{far}}(Q)$ in the same time complexity. We now show how this can be done.

We denote by $\mathbb{L}_{\text{far}}(Q)$ the set of left hulls of the sites of each element in $\mathbf{V}_{\text{far}}(Q)$. We observe that a triangulation of the region between Q_r and $L(Q_r)$ requires $O(m)$ edges and that the edges of the chains in $\mathbb{L}_{\text{far}}(Q)$ form a partial triangulation of this region. Our algorithm will triangulate this region using these edges and adding additional edges as necessary[4]. The total cost of this construction will be $O(m)$ since it will cost $O(1)$ per edge of the triangulation. The rest of our algorithm will deal with constructing the elements of $\mathbb{L}_{\text{far}}(Q)$. The total cost of this construction will be $O(h)$, since for each element of $\mathbb{L}_{\text{far}}(Q)$ the cost will be proportional to its length.

We will modify the operator $\widehat{\max}$ so that it does the necessary calculations during the preprocessing stage of Algorithm 9. Let $\widehat{W}$ be either an input parameter to $\widehat{\max}$ or it's output. Then we require that $\widehat{W}$ have the following additional information:

(1) A triangulation of the regions between W_r and $L(W_r)$.

[4]The edges need not actually be added. We add them because it simplifies the discussion.

(2) Chain $L(W_r)$ represented as a doubly linked list.

(3) $\ell(W_r)$.

Also, $\widehat{\max}$ needs to be able to compute W_r^c from $L(W_r)$ in $O(|W_r^c|)$ time. It can then compute $\mathrm{Vor}_{\mathrm{far}}(W_r^c)$ using chain W_r^c in $O(|W_r^c|)$ time as required. With these modifications Algorithm 9 runs $O(m \log m)$ time. It remains to be shown how $\widehat{\max}$ computes the additional information listed above.

Let R, S, and T be three subchains of Q_r such that $T = RS$ and R is below S. Let v be the vertex shared by R and S. Assume that $L(R)$ and $L(S)$ are known and represented by doubly linked lists, that $\ell(R)$ and $\ell(S)$ are known, and that the regions between chains R and $L(R)$ and between chains S and $L(S)$ have been triangulated. Thus R and S are valid inputs to $\widehat{\max}$.

Observe that if v is not a vertex of $L(T)$ then there is a polygon, say U, bounded by $L(R)$, $L(S)$, and an edge $\overline{pq}$ of $L(T)$ such that $p \in V(R)$ and $q \in V(S)$. We find $\overline{pq}$ by triangulating U which can be done in $|U|$ time without prior knowledge of $\overline{pq}$. Then by altering the pointers at p and q we construct $L(T)$ in $O(1)$ time. We charge these computations to the cost of constructing the triangulation of the region between T and $L(T)$. Note that now the regions between T and $L(T)$ are triangulated.

If v is a vertex of $L(T)$ then we can construct $L(T)$ represented as a doubly linked list in $O(1)$ time. Furthermore, the region between T and $L(T)$ is already triangulated.

Next we compute $\ell(T)$ from $\ell(R)$ and $\ell(S)$ in $O(1)$ time.

Now T^c is a subchain of $L(T)$ and $\ell(T) \in V(T^c)$. Therefore the endpoints of T^c can be found by following the links between nodes in $L(T)$, starting from $\ell(T)$, in $O(|T^c|)$ time.

We now charge all computations not already charged to constructing the triangulation of the region between T and $L(T)$ to the cost of constructing T^c. This is $O(T^c)$ time in total.

Now $L(T)$, represented by a doubly linked list, has been constructed, a triangulation of the region between T and $L(T)$ has been constructed, T^c has been constructed, and $\ell(T)$ has been computed. Thus T is a valid output result for operator $\widehat{\max}$. Finally, $\mathrm{Vor}_{\mathrm{far}}(W_r^c)$ can now be computed in $O(|W_r^c|)$ time.

We summarize the results of this section with the following result.

THEOREM 5.4.1. *Let Q be an m vertex an m vertex y-monotone polygon. We can preprocess Q in $O(m \log m)$ time and $O(m \log m)$ space so that, for a query semi-canonical arc ζ of Q we can determine $\hbar_\zeta(Q)$ in $O(\log^2 m)$ time.*

5.5. Circular Ray Shooting in a Simple Polygon

Next we need to generalize Algorithm 9 to work with an x-2*-wind polygon P with n vertices. One method of doing this is to partition P into y-monotone polygons using Algorithm 2. Then, for a semi-canonical arc ζ of P we simply apply Algorithm 9 to each subpolygon intersected by ζ in turn starting with the subpolygon containing ζ_o. This gives us an algorithm with a query time complexity of $O(\log^3 n)$ since the query part of Algorithm 9 may need to be applied $O(\log n)$ times.

We remind the reader that, for a simple polygon P we use $\Psi(P)$ to denote the ordered set of cusps of P, $\mathfrak{L}(P)$ to denote the lids of the open cusps in $\Psi(P)$, and $\pi(P)$ to denote the set of (y-monotone) subpolygons into which P is partitioned by $\mathfrak{L}(P)$. These and several other terms that we use here are defined in Chapters 2 and 3.

To avoid the extra $O(\log n)$ factor we instead partition P into the set of y-monotone polygons $\pi^w(P)$ (see Section 4.3) using Algorithm 5 and the version of weighted binary trees developed in A.4. If we modify Algorithm 9 in this way, the

cost of the preprocessing and space increase by a constant factor. This is because, for the same input data, the weighted binary trees developed in A.4 have a depth of at most three times the depth of the (implicit binary) tree constructed using Algorithm 15 and the cost of constructing each level is linear in time and space.

The cost of a circular ray shooting query is now $O(\log t)$ applications of operator $\dot{<}$ by Theorem A.4.7 where $t = |\text{trap}(P)| < n$. Since an application of the operator $\dot{<}$ requires $O(\log n)$ time, in the worst case, a query requires $O(\log^2 n)$ time.

We now discuss how to perform circular ray shooting in a simple polygon. Let P be a simple polygon with n vertices. The first step is to construct $\pi_{\text{red}}(P)$. Let $k = \max\{|\text{trap}(Q)| : Q \in \pi_{\text{red}}(P)\}$. We then preprocess each element of $\pi_{\text{red}}(P)$ for circular ray shooting as described for $\pm x$-2^*-wind polygons in this section. This costs $O(n \log k)$ time and space. Now, by Lemma 3.4.12, a circular arc can intersect at most four elements of $\pi_{\text{red}}(P)$ without hitting P. Thus the cost of a circular ray shooting query is at most the cost of doing a circular ray shooting query in four of the elements of $\pi_{\text{red}}(P)$. Thus the cost is $O(\log^2 k)$ time.

We summarize the result of this section with the following theorem.

THEOREM 5.5.1. *For a polygon P with n vertices the circular ray shooting problem can be solved, using $O(n \log n)$ preprocessing time and space, so that, for any query arc ζ, we can find $\hbar_\zeta(P)$ in $O(\log^2 n)$ time.*

5.5.1. Some Practical Considerations. We solved the circular ray shooting problem using a trapezoidization of the polygon. This may not be the most efficient approach. This approach causes three difficulties:

(1) In the worst case each, trapezoid (except one) introduces a steiner point of the polygon that then becomes part of the Voronoi diagrams that are to be constructed. Since the memory requirements are dominated by the Voronoi diagrams this is expensive.

(2) Because we partition a query arc into quarter arcs we need to use not only a horizontal trapezoidization of the input polygon but a vertical trapezoidization as well, thus doubling the preprocessing and space costs of the algorithm and increasing the worst case query costs by two thirds. If we partitioned the arc instead into parts of a left and right semi-circle then we would not incur these extra costs. Unfortunately we do not know how to modify the part of Algorithm 9 that uses farthest point Voronoi diagrams to deal with semi-circles. However, the algorithm that deals with edge Voronoi diagrams has no such difficulties. Furthermore, edge Voronoi diagrams are far more expensive to construct and use than the farthest point Voronoi diagrams so dealing with semi-circular arcs instead of quarter-circular arcs would be very beneficial.

(3) Our algorithm is heavily dependent upon the use of weighted binary trees. The algorithm that constructs these trees attempts to organize the data to balance the weights corresponding to the data points efficiently. However it can do nothing about the fact that the two sides of a monotone polygon may have a very different number of vertices.

We propose breaking Problem 5.1.1 into two subproblems. Let P be a simple polygon. For any point p on P let $\sigma(p)$ be a point on P preceding p in a (counterclockwise) traversal of P by an arbitrarily small amount.

PROBLEM 5.5.2. *Preprocess P so that, for a query arc ζ, with ζ_o interior to P one can quickly determine the first point p (if any) along ζ that intersects P such that $\sigma(p)$ is exterior to circle(ζ).*

PROBLEM 5.5.3. *Preprocess P so that, for a query arc ζ, with ζ_o interior to P one can quickly determine the first point p (if any) along ζ that intersects P such that $\sigma(p)$ is interior to circle(ζ).*

We solve Problem 5.5.2 by first constructing $\pi_{\text{red}}(P)$. Let Q be a element of $\pi_{\text{red}}(P)$. We deal only with how to solve this problem on Q. The generalization to solving the problem on P is essentially the same as covered earlier in this section. We start by constructing $\pi^w(Q)$. Now we can solve Problem 5.5.2 using a weighted binary tree for each of the monotone chains of each element of $\pi^w(Q)$ where the nodes of the tree contain edge Voronoi diagrams of y-monotone chains. It is necessary to cut ζ into at most three components each of which is a left or right semi-circle of circle (ζ). Note that each edge of Q, and thus P, is used in only one weighted binary tree as opposed to two trees if trapezoids are used. We can now detect whether ζ intersects the y-monotone chain corresponding to a node in the weighted binary tree using the edge Voronoi diagram stored at the node. The remainder of the algorithm works the same as discussed in the previous sections.

To solve Problem 5.5.3 we still need to cut ζ into five components. Thus we still need to use a vertical trapezoidization as well as a horizontal decomposition. For simplicity we assume ζ is a left quarter arc. How to deal with the case that it is not has already been discussed. Thus we can assume wlog that ζ is contained in some subpolygon $Q \in \pi_{\text{red}}(P)$. We start by constructing $\pi^w(Q)$. Next we construct a weighted binary tree for each of the monotone chains of each element of $\pi^w(Q)$ where the nodes of the tree contain farthest point Voronoi diagrams built from y-monotone chains as described in Sections 5.3 and 5.4. Now we can detect whether ζ intersects the y-monotone chain corresponding to a node in the weighted binary tree using the farthest point Voronoi diagram stored at the node. The remainder of the algorithm works the same as discussed in the previous sections.

We can now solve a query for Problem 5.1.1 by solving Problems 5.5.2 and 5.5.3 and then choosing the intersection closest to ζ_o along ζ. There is a difficulty with this solution however. It may be that the solution to one of Problems 5.5.2 and 5.5.3

is very close to ζ_o (measured along ζ) and found quickly yet the other solution is relatively far from ζ_o and thus much more costly to find. We resolve this difficulty as follows.

Call the algorithm for solving Problem 5.5.2 Algorithm Ext and the algorithm for solving Problem 5.5.3 Algorithm Int. We modify Algorithm Ext so that each time it applies an intersection test that reports no intersection it returns with a point p_i on ζ such that the algorithm has determined that the section of ζ from ζ_o to p_i is clear of any intersections of P that can be found by Algorithm Ext. Of course, if the algorithm finds an intersection with ζ it also returns. Algorithm Ext is also modified so that the next time it is invoked it continues the search from the point that it stopped earlier. We make similar modifications to Algorithm Int. Now we solve a query for Problem 5.1.1 by calling Algorithms Int and Ext repeatedly, each time calling the algorithm that has searched the least distance from ζ. We stop searching when either both of the algorithms have found ζ_e or one of the algorithms has found $\hbar_\zeta(P)$ and the other has found a point along ζ beyond $\hbar_\zeta(P)$ without finding an intersection point between ζ_o and $\hbar_\zeta(P)$.

Using this approach we are assured that no Voronoi diagram point location test is performed unnecessarily.

CHAPTER 6

Rectangle Visibility Decompositions

6.1. Introduction

In this chapter we study rectangle visibility decompositions of simple polygons. Using these rectangle visibility decompositions, we solve the problem of finding the largest axis-aligned rectangle inscribed in a polygon which we, like [21], refer to as the *LR Problem.* In [21], the authors showed that this problem can be solved, in the case of a rectilinear polygon with n vertices, in $O(n \log^2 n)$ time. They then showed that the LR problem can also be solved in the same time complexity for the general polygon case.

They solved both problems by using a rectangle visibility decomposition of a simple polygon[1] into subpolygons we call SAM polygons. They construct a SAM rectangle visibility cover, say Π, of an n vertex simple polygon such that $\mathbb{C}_{\square}(\Pi) = 1$. In the worst case $\|\Pi\| = O(n \log n)$. They construct Π in $O(n \log n)$ time, avoiding a space complexity of $O(n \log n)$, when solving the LR problem, by never storing more than one element of Π at a time. They then show that the LR problem can be solved for a SAM polygon with m vertices in $O(m \log m)$ time. Thus, they solve the LR problem in $O(n \log^2 n)$ time.

The contribution we make in this chapter is a new method of decomposing a polygon into SAM subpolygons. We also construct a rectangle visibility cover, say Π', of an n vertex simple polygon P such that $\mathbb{C}_{\square}(\Pi') = 1$; however, for our algorithm, $\|\Pi'\| = O(n)$, an improvement by a factor of $O(\log n)$. We are able to construct

[1] In fact their algorithm works also with polygons with holes; an advantage of their algorithm over ours.

Π' in linear time and space. We then solve the LR problem using the methods developed in [21] for solving the LR problem on SAM polygons. If P is rectilinear then the subpolygons of Π' are all SAM polygons; otherwise they are a combination of SAM polygons and quadrilaterals; we point out that the LR problem is easily solved on quadrilaterals. Now, since the LR problem can be solved for an m vertex SAM polygon in $O(m \log m)$ time, we are able to solve the LR problem for a simple polygon in $O(n \log n)$ time. Thus, we have achieved a lower time complexity for the LR problem by developing a better rectangle visibility cover class.

We say that a rectangle is axis-aligned if its sides are parallel to either the x-axis or the y-axis. Given a simple polygon P, we say that two points $p, q \in P$ are *rectangularly visible* if there exists an axis-aligned rectangle inscribed in P that contains them. Assume that p and q are rectangularly visible. We denote the smallest rectangle in P containing both p and q by rect (p, q). Let Π be a decomposition of P. We denote the minimum number of subpolygons of Π needed to cover rect (p, q) by $\mathsf{C}_{\square}(\Pi, p, q)$. We further denote

$$\mathsf{C}_{\square}(\Pi) = \max\left\{\mathsf{C}_{\square}(\Pi, p, q) \,:\, p, q \text{ are rectangularly visible points of } P\right\}.$$

If $\mathsf{C}_{\square}(\Pi) = o(|P|)$ then we say that Π is a *rectangle visibility decomposition* of P.

There are three main types of rectangle visibility decompositions that we will study but each will be split into two sub-types; one for rectilinear polygons and one for general polygons. The algorithms that construct decompositions of rectilinear polygons are simpler and will be dealt with first. The algorithms for the general cases require all the tools we develop for the rectilinear polygon cases but also require some additional steps. We deal with the general polygon cases in Section 6.6.

The three decomposition classes that we study here form a hierarchy with successive decomposition classes being the composition of the previous ones. We use the last of these decompositions to solve the LR problem.

In Section 6.2 we study the first of the three decomposition classes, the histogram subpolygon visibility partition class. The rectangle visibility cover number for elements of this partition class is at most three.

In Section 6.3 we study the second class of decompositions, the histogram family subpolygon visibility cover class. The rectangle cover number for elements of this class is one. The main reason we study this cover class here is because it is needed by the algorithm that generates the polygon cover class of Section 6.4.

In Section 6.4 we define Separable Axis Monotone polygons or SAM polygons. We then present the third class of visibility decompositions, a SAM subpolygon rectangle visibility cover class. The rectangle cover number for elements of this class is one. The advantage of the SAM subpolygon cover class over the histogram family subpolygon cover class is that SAM polygons are simpler than histogram family polygons. An element of the SAM subpolygon cover class can be constructed in linear time and space in the size of the polygon.

In Section 6.5 we study the largest inscribed axis-aligned rectangle problem or LR problem for rectilinear polygons. We show that the results of Section 6.4 imply that this problem is reducible to the same problem on SAM polygons. Then the fact that the rectangle cover number of a SAM rectangle visibility cover of a polygon is one implies that the LR problem can be solved in $O(n \log n)$ time where n is the size of the polygon.

In Section 6.6 we generalize the results of the previous sections. We show how to construct a cover of simple polygon using SAM subpolygons and quadrilaterals such that the rectilinear visibility cover number of the cover is one. The algorithm that

generates this cover class uses linear time and space in the size of the input polygon. We then use this cover class to solve the LR problem in the same complexity as for rectilinear polygons.

All rectangles used in the remainder of this chapter will be assumed to be axis-aligned.

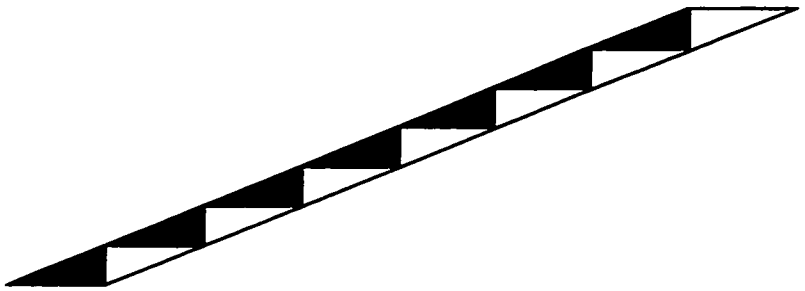
6.2. Histogram Subpolygon Rectilinear Visibility Partitions

We say that a polygon P with a base edge e is an *α-histogram polygon* if $\dot{-}\angle e = \alpha$ and for any point p interior to P there exists a point q belonging to e such that $\overline{pq}$ is perpendicular to e and interior to P. We remind the reader that the edges and chords of a polygon are directed and that $\dot{-}\angle e$ denotes the direction opposite to that of edge e on P. The reason we reverse the direction of e is that in almost all cases where we deal with histogram polygons they will be subpolygons of another polygon, say Q, such that e is also a chord of Q and the chord has an orientation of $\dot{-}\angle e$ assigned to it by virtue of being a chord of Q; see Chapter 3. For the remainder of this chapter we will only be concerned with α-histogram polygons where $\alpha \in \{0, \pi/2, \pi, 3\pi/2\}$.

We point out that in [34] an α-histogram polygon is called a normal histogram polygon. We also point out that, unlike standard histograms, these polygons need not be rectilinear. In fact an α-histogram polygon is nothing more than a $\alpha^{\perp}$-1-wind polygon in which the last edge has an orientation of $-\alpha^{\perp}$; see Chapter 3. We will use the term histogram polygon since it more closely matches the terminology in [34].

If P is a rectilinear polygon then we denote the histogram subpolygon partition of P constructed, using Strategy 2.2.1 and using $\psi(P)$ as the initial base edge, by $\pi_{\text{hist}}(P)$. An algorithm that generates this partition class in linear time is given in [37]; see also [42].[2] An example of a partition constructed using this algorithm is

[2] In [42] these partitions are called histogram partitions.



A thin parallelogram may require a partition into histograms polygons using an arbitrarily large number of Steiner points. The histograms in this partition alternate in color between black and white.

FIGURE 6.2.1. A histogram subpolygon partition using a large amount of memory

given in Figure 2.1.1. In **[42]** the partition class generated by this algorithm is used to solve the shortest rectilinear path problem for a rectilinear polygon.

However, partitioning general polygons into histogram subpolygons can cause problems. A histogram subpolygon partition of a simple polygon may require an arbitrarily large amount of space. To see this, consider a parallelogram with two non horizontal parallel sides that are very close together. The partition of this parallelogram into histogram polygons can be arbitrarily large depending on how thin we make the parallelogram; see Figure 6.2.1.

This problem is dealt with using pseudo-histogram polygons.

Let H be a simple polygon, l be the edge immediately preceding base(H), and r be the edge immediately following base(H). We say that H is a *pseudo-histogram polygon* or just *pseudo-histogram* if the end-points of base(H) are convex and, for any point p interior to H, there is point q on either base(H), l, or r such that directed line segment $\overrightarrow{pq}$ is perpendicular to base(H), interior to H, and directed towards the line through base(H). Let C be the subchain of the chain $l, \text{base}(H), r$ that is reachable from some interior point s of H by a directed line segment that is perpendicular to

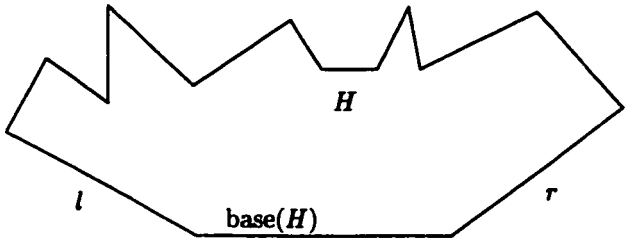


FIGURE 6.2.2. A pseudo-histogram

base(H), interior to H, and directed towards the line through base(H). We call C the *base chain* of H and denote it by baseChain (H). Note that, in degenerate cases, a pseudo-histogram polygon may have a base chain of length one or two but it always has a base edge. See Figure 6.2.2. Since the meaning is clear, we will henceforth abbreviate pseudo-histogram polygon to just *pseudo-histogram.*

Let P be a polygon. A variation of the method used to partition polygons into histogram polygons can be used to partition polygons into pseudo-histograms. Let l be the edge of P preceding base(P) and r be the edge of P following base(P). Now let ζ be the chain of P that consists of:

- l if the shared vertex of base(P) and l is convex and its interior angle is obtuse,
- base(P), and
- r if the shared vertex of base(P) and r is convex and its interior angle is obtuse.

We denote by phist(P) the subpolygon Q of P such that, for any point p interior to Q, there is a point $q \in \zeta$ where line segment $\overline{pq}$ is perpendicular to base(P) and interior to P. We denote the pseudo-histogram subpolygon partition of P, constructed using the definition of phist(P) to construct subpolygons and Strategy 2.2.1, with $\psi(P)$

as the initial base edge, by $\pi_{\text{phist}}(P)$. Each element of $\pi_{\text{phist}}(P)$ is either an 0π-pseudo-histogram, a $\pi/2$-pseudo-histogram, a π-pseudo-histogram, or a $3\pi/2$-pseudo-histogram. We can construct $\pi_{\text{phist}}(P)$ in $|P|$ time using an algorithm in **[34]**. We call $\pi_{\text{phist}}(P)$ the *pseudo-histogram subpolygon visibility partition* or *PHistVP* of P. PHistVPs have been used to solve a number of problems including constructing the medial axis of a polygon **[34, 17]** and finding the shortest diagonal in a polygon **[53]**. For the remainder of this chapter we will only be concerned with α-pseudo-histograms where $\alpha \in \{0, \pi/2, \pi, 3\pi/2\}$.

We will solve the LR problem for rectilinear polygons first. However, we will use terminology relating to pseudo-histograms so as to avoid needing two sets of terms, one for rectilinear polygons and one for general polygons. Of importance here is that if a polygon is rectilinear then the polygon is an α-histogram polygon if and only if it is an α-pseudo-histogram assuming $\alpha \in \left\{0, \frac{\pi}{2}, \pi, \frac{3\pi}{2}\right\}$. Also, for a rectilinear polygon P, we have that $\pi_{\text{hist}}(P) = \pi_{\text{phist}}(P)$. Again, to avoid duplication of terminology, we will also use $\pi_{\text{phist}}(P)$ instead of $\pi_{\text{hist}}(P)$ for any rectilinear polygon P.

Let $\Pi = \pi_{\text{phist}}(P)$ where P is a simple polygon. We remind the reader that $\widehat{\Gamma}(\Pi)$ denotes the set of chords of P that determine Π and that, for each chord $\gamma_i \in \widehat{\Gamma}(\Pi)$, we denote by P_{γ_i} the subpolygon of P determined by γ_i that does not include $\psi(P)$; see Chapters 2 and 3. By construction, partition Π has the property that each subpolygon $Q_i \in \Pi$, is the maximal α-pseudo-histogram subpolygon of $P_{\text{door}(Q_i)}$. If P is rectilinear then Q_i is an α-histogram polygon.

The following observation will prove useful when dealing with rectilinear polygons.

OBSERVATION 6.2.1. *Let P be a rectilinear polygon with n reflex vertices. Then P has $n+4$ convex vertices and $2n+4$ vertices altogether.*

The reason we call a PHistVP of a rectilinear polygon a rectangle visibility partition is the following result:

LEMMA 6.2.2. *Let P be a simple rectilinear polygon. Then* $\mathbf{C}_{\square}(\pi_{phist}(P)) \leq 3$.

PROOF. Let R be a rectangle interior to P and Q be the element of $\pi_{\text{phist}}(P)$ that is intersected by R and closest to base(P) by a shortest path interior to P. Then R can intersect a child polygon of Q, say Q_c, if and only if two edges of R, both parallel to door(Q) intersect door(Q_c). Then an edge of R must be contained in Q_c and this edge must be parallel to door(Q_c). Now R has only two such edges so R can intersect at most two child polygons of Q. Also, clearly, no other subpolygons of $\pi_{\text{phist}}(P)$ can be intersected by R. □

Let Π be a decomposition of a polygon P. Let v_i be a vertex of P and let s_i be a Steiner point of P. If $\overline{v_i s_i}$ is a chord of P and an edge or part of an edge of any element, say Q, of Π and s_i is a vertex of Q then we say that v_i is the cause for the use of s_i in Q; note that in this chapter it will never be that s_i was caused to be used by two vertices of P. However, v_i may cause s_i to be used several times, once for each subpolygon of Π that has s_i as a vertex.

LEMMA 6.2.3. *Let P be any rectilinear polygon. Then* $\|\pi_{phist}(P)\| \leq 2|P| - 4$.

PROOF. Each vertex of P is a vertex of exactly one element of $\pi_{\text{phist}}(P)$. Also, a reflex vertex may cause the use of a Steiner point and this Steiner point may used in as many as two elements of $\pi_{\text{phist}}(P)$. Now assume that P has n reflex vertices. Then, by Observation 6.2.1, $\|\pi_{\text{phist}}(P)\| \leq n + 4 + 3n = 2|P| - 4$. □

We will use the pseudo-histogram partition class of rectilinear polygons and the algorithm that generates it in a novel way. Instead of solving a problem directly with it we will use it as a stepping stone to constructing a simpler rectilinear visibility cover class which we will then use to solve the LR problem. The next step in this progression is covered in the following section.

6.3. Histogram Family Subpolygon Visibility Covers

We remind the reader that, for any tree partition Π of a polygon P, we can construct the cover Π^f of P in linear time and space; see Subsection 2.2.1.

Let P be a simple polygon. If $\Pi = \pi_{\text{phist}}(P)$ then we call Π^f a *pseudo-histogram family rectangle visibility cover or PHistFVC* of P and we call the subpolygons of the cover *pseudo-histogram family polygons.* We further denote $\pi^f_{\text{phist}}(P) = \Pi^f$. Let Q be an element Π^f and Q_B be the element of $\pi_{\text{phist}}(Q)$ such that $\text{door}(Q_B) = \text{door}(Q)$. Let α be the angle such that Q_B is an α-pseudo-histogram. Then we say that Q is an *α-pseudo-histogram family polygon.* Note that $\alpha \in \{0, \frac{\pi}{2}, \pi, \frac{3\pi}{2}\}$. The reason we call the PHistFVC of a rectilinear polygon a rectangle visibility cover is the following result:

LEMMA 6.3.1. *Let P be any rectilinear polygon. Then $\mathbf{C}_{\square}(\pi^f_{phist}(P)) = 1$.*

PROOF. Follows immediately from the proof of Lemma 6.2.2. □

LEMMA 6.3.2. *Let P be any rectilinear polygon. Then $\|\pi^f_{phist}(P)\| \leq 3|P| - 4$.*

PROOF. Each vertex of P is a vertex of at most two elements of $\pi^f_{\text{phist}}(P)$. Also, a reflex vertex may cause the use of a Steiner point and this Steiner point may be a vertex in as many as two elements of $\pi^f_{\text{phist}}(P)$. Now assume that P has n reflex vertices. Then, by Observation 6.2.1, $\|\pi^f_{\text{phist}}(P)\| \leq 2n + 8 + 4n = 3|P| - 4$. □

An example of a pseudo-histogram and its partition into histogram polygons is given in Figure 2.1.1.

Our main interest in PHistFVCs of rectilinear polygons is as a stepping stone in the construction of a rectilinear polygon cover of a polygon composed of even simpler subpolygons. This is the subject of the next section.

6.4. SAM Subpolygon Rectangle Visibility Covers

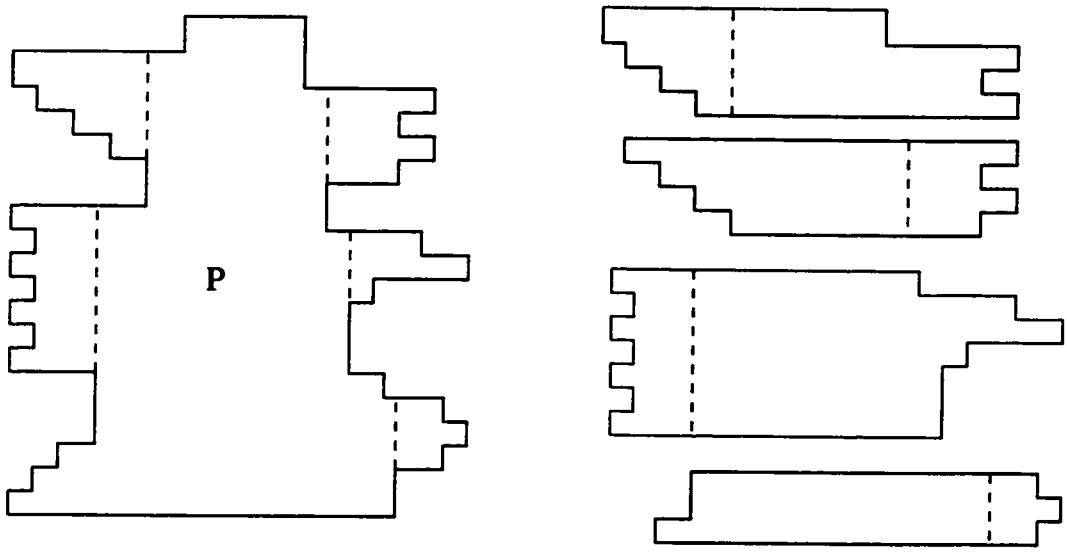
We say that a y-monotone polygon is *vertically separated* if there is a vertical chord of the polygon joining its cusps. We say that an x-monotone polygon is *horizontally separated* if there is a horizontal chord of the polygon joining its y-cusps. We call a polygon a *separated axis monotone polygon* or *SAM polygon* if it is either a vertically separated y-monotone polygon or a horizontally separated x-monotone polygon.[3] We consider any histogram polygon to be a degenerate SAM polygon.

In this section we develop an algorithm that constructs a SAM subpolygon rectangle visibility cover with a cover number of one for any rectilinear histogram family polygon. The composition of this algorithm and the algorithm that generates the histogram family subpolygon cover class for rectilinear polygons is an algorithm that constructs a SAM subpolygon rectangle visibility cover of any rectilinear polygon. We call a cover constructed by this algorithm, for a rectilinear polygon, a *SAM rectangle visibility cover* or *SAMVC1*. If the polygon is not rectilinear then the algorithm generally only returns a partial cover.

Like PHistFVCs, SAMVC1s have a rectangle visibility cover number of one. The advantage of using SAMVC1s instead of a PHistFVCs is that a SAM polygon is simpler than a pseudo-histogram family polygon, rectilinear or not, and we can exploit this fact to solve problems on polygons. In particular, we will use it to solve the LR problem.

6.4.1. The SAMPc of a Histogram Family Polygon. Let γ be a chord or edge of a simple polygon P. We denote by $\text{vis}_{\perp}(\gamma)$ the subpolygon Q of P such that, for any point p interior to Q, there is point $q \in \gamma$ where line segment $\overline{pq}$ is perpendicular to γ and interior to P.

[3] In [21] these polygon classes are called vertically separated horizontally convex polygons and horizontally separated vertically convex polygons respectively.



The subpolygons SAMPc(P) of a rectilinear histogram family polygon P. In this case each subpolygon is separated y-monotone.

FIGURE 6.4.1. A SAM subpolygon cover of a rectilinear polygon

Let Q be an α-pseudo-histogram family polygon for some angle $\alpha \in \{0, \frac{\pi}{2}, \pi, \frac{3\pi}{2}\}$. Let U be the α-pseudo-histogram of Q such that base (U) = base (Q). Let $l_1, l_2, \ldots, l_j$ be the parts of the boundary of U that are chords of Q and have orientation $\angle\alpha^{\perp}$ on Q and let $r_1, r_2, \ldots, r_k$ be the parts of the boundary of U that are chords of Q and have orientation $\dot{-}\angle\alpha^{\perp}$ on Q. For any two distinct chords $l_h, l_i \in \{l_1, l_2, \ldots, l_j\}$, we observe that $\text{vis}_{\perp}(l_h) \cap \text{vis}_{\perp}(l_i) = \emptyset$. Therefore $\Sigma_{i=1}^{j} |\text{vis}_{\perp}(l_i)| = O(|Q|)$. Similarly, $\Sigma_{i=1}^{k} |\text{vis}_{\perp}(r_i)| = O(|Q|)$.

Now let $S = \{\text{vis}_{\perp}(l_h),\ 1 \leq h \leq j;\ \text{vis}_{\perp}(r_i),\ 1 \leq i \leq k\}$. Let $S' \subset S$ contain the elements of S that are interior to other elements of S; in the case that S contains two copies of the same subpolygon only one is put into S'. We denote SAMPc $(Q) = S - S'$; see Figure 6.4.1.

Note that, generally, SAMPc(Q) is not a cover of Q but SAMPc$(Q) \cup \{U\}$ is a cover of Q if Q is rectilinear. The key properties of SAMPc(Q) are captured in the following lemmas:

LEMMA 6.4.1. *If Q is rectilinear then $SAMPc(Q) \cup \{U\}$ is a cover of Q and $\mathsf{C}_{\square}(SAMPc(Q) \cup \{U\}) = 1$.*

PROOF. This lemma can be proven using arguments similar to those used in the proof of Lemma 6.2.2. □

LEMMA 6.4.2. *If Q is rectilinear then $\|SAMPc(Q)\| \leq 2|Q|$.*

PROOF. Each vertex of Q is a vertex of at most two elements of SAMPc(Q). Also, a reflex vertex may cause the use of a Steiner point in a element of SAMPc(Q). However a reflex vertex can cause a Steiner point to be used only if it is a vertex of at most one element of SAMPc(Q). Therefore $\|\text{SAMPc}(Q)\| \leq 2|Q|$. □

LEMMA 6.4.3. *$SAMPc(Q)$ can be constructed in $O(|Q|)$ time.*

PROOF. First we construct a trapezoidization T of Q such that the chords of the trapezoidization are parallel to base(Q). Now, for each chord $\gamma_i \in \widehat{\Gamma}(\pi_{\text{phist}}\{Q\})$, we construct the element, say σ_i, of SAMPc(Q) corresponding to γ_i using the trapezoids of T contained in σ_i in time proportional to $|\sigma_i|$. Thus, constructing SAMPc(Q) requires $O(\|\text{SAMPc}(Q)\| + |T|)$ time. But if Q is rectilinear then $\|\text{SAMPc}(Q)\| \leq 2|Q|$ by Lemma 6.4.2, otherwise $\|\text{SAMPc}(Q)\| \leq 3.5|Q|$ by Lemma 6.6.5. □

6.4.2. The SAMVC1 of a Rectilinear Polygon. The above three lemmas lead directly to Algorithm 10 which constructs a SAM subpolygon rectangle visibility cover of a rectilinear polygon. For any simple polygon P, we denote the SAM subpolygon partial cover of P, constructed by applying Algorithm 10 to P, by $\pi^c_{\text{saml}}(P)$.

Some important properties of $\pi^c_{\text{saml}}(P)$ are captured in the following lemmas:

LEMMA 6.4.4. *If P is rectilinear then $\mathsf{C}_{\square}(\pi^c_{saml}(P)) = 1$.*

PROOF. Let R be any rectangle contained in P. We need to show that R is contained in some element of $\pi^c_{\text{saml}}(P)$. First, R is contained in an element, say

Algorithm 10

Input: A simple polygon P.

Output: $\pi^c_{\text{saml}}(P)$. /* A SAM subpolygon partial cover of P. If P is rectilinear this result is a SAM subpolygon rectilinear visibility cover of P. */

1: $\Pi := \{\text{vis}_\perp(\psi(P))\}$.

2: **for** each pseudo-histogram family polygon $Q \in \pi^f_{\text{phist}}(P)$ **do**
$\Pi := \Pi \cup \text{SAMPc}(Q)$.

3: Return Π.

Q, of $\pi^f_{\text{phist}}(P)$ since $\mathbf{C}_\square(\pi^f_{\text{phist}}(P)) = 1$ by Lemma 6.3.1. Let B be the element of $\pi_{\text{phist}}(Q)$ such that $\text{door}(Q) = \text{door}(B)$. Let T be the element of π^f_{phist} such that $\text{base}(T) = \psi(P)$. If $\text{base}(T) \neq \text{base}(Q)$ then let U the of element of π^f_{phist} such that $\text{base}(Q)$ is a chord of U. Now, by Lemma 6.4.1, either R is contained in some element $\text{SAMPc}(Q)$ or else R is contained in B.

In the first case R is contained in an element of $\pi^c_{\text{saml}}(P)$ by Step 2 of Algorithm 10.

In the second case either $Q = T$ and then R is contained in an element of $\pi^c_{\text{saml}}(P)$ by Step 1 of Algorithm 10 or $Q \neq T$ and then R is contained in an element of $\text{SAMPc}(U)$. But $\text{SAMPc}(U) \subset \pi^c_{\text{saml}}(P)$ by Step 2 of Algorithm 10. □

LEMMA 6.4.5. *If P is rectilinear then $\|\pi^c_{saml}(P)\| < 3|P| - 4$.*

PROOF. Each convex vertex of P is a vertex of at most two elements of $\pi^c_{\text{saml}}(P)$ and each reflex vertex is a vertex of at most three elements of $\pi^c_{\text{saml}}(P)$. Also, a reflex vertex may cause the need for two Steiner points and one of these Steiner points may be a vertex in two elements of $\pi^c_{\text{saml}}(P)$ while the other may be a vertex in only one element of $\pi^c_{\text{saml}}(P)$. However, if a reflex vertex is used two or three times it causes at most one Steiner point to be used. Now assume that P has n reflex vertices. Then, by Observation 6.2.1, $\|\pi^c_{\text{saml}}(P)\| \leq 6n + 8 = 3|P| - 4$. □

LEMMA 6.4.6. *$\pi^c_{saml}(P)$ can be constructed in $O|P|$ time.*

PROOF. It was shown in [34] that $\pi_{\text{phist}}(P)$ can be constructed in $O|P|$ time. Also, it is straightforward to construct $\pi^f_{\text{phist}}(P)$ from $\pi_{\text{phist}}(P)$ in linear time. Now, $\pi^c_{\text{saml}}(P)$ can be constructed from $\pi^f_{\text{phist}}(P)$ in $O(|P|)$ time by Lemmas 6.4.3 and 6.3.2. □

6.5. The Largest Axis-Aligned Rectangle Problem

The LR problem can be solved for a rectilinear SAM polygon with m vertices in $O(m \log m)$ time [21]. From this fact and Lemmas 6.4.4 and 6.6.7 the following result follows:

THEOREM 6.5.1. *Let P be a simple rectilinear polygon with n vertices and let k be the size of the largest element of $\pi^c_{saml}(P)$ by vertex count. Then the LR problem can be solved on P in $O(n \log k)$ time.*

This is an improvement by a factor of $O(\log n)$ over the algorithm in [21] for an n vertex rectilinear polygon.

6.6. Generalization to Simple Polygons

We now generalize the results of the previous sections to simple polygons. The methods we use will largely parallel those used in the rectilinear polygon case.

LEMMA 6.6.1. *Let P be a simple polygon. Then $\mathbf{C}_{\square}(\pi_{phist}(P)) \leq 3$.*

PROOF. The proof is analogous to the proof of Lemma 6.2.2. □

The reason we call the PHistFVC of a simple polygon a rectangle visibility cover is the following result:

LEMMA 6.6.2. *Let P be any simple polygon. Then $\mathbf{C}_{\square}(\pi^f_{phist}(P)) = 1$.*

PROOF. This result follows immediately from the proof of Lemma 6.6.1. □

LEMMA 6.6.3. *Let Q be any pseudo-histogram family polygon. Then $\|\pi_{phist}(Q)\| \leq 2|Q|$.*

PROOF. Each vertex of Q is used in at most twice in $\pi_{\text{phist}}(Q)$. Also, a reflex vertex may cause the use of a Steiner point. However, for each such Steiner point there is a convex vertex that can be uniquely mapped to this Steiner point. Furthermore, if a convex vertex is used twice then it prevents a Steiner point from being used, so, for accounting purposes we need count the convex vertex only once. Thus, we can redistribute the counting of vertices such that no vertex is counted more than twice. Therefore $\|\pi_{\text{phist}}(Q)\| \leq 2|Q|$. □

LEMMA 6.6.4. *Let P be any simple polygon. Then $\|\pi^f_{\text{phist}}(P)\| < 7|P|$.*

PROOF. Each convex vertex of P is used in at most twice in $\pi_{\text{phist}}(P)$ and each reflex vertex is used in at most three times. Also, a reflex vertex may cause the need for as many as two Steiner points and each Steiner point may be a vertex of as many as two elements of $\pi^f_{\text{phist}}(P)$. Thus, since P has at least one convex vertex, $\|\pi^f_{\text{phist}}(P)\| < 7|P|$. □

Finally, since $\pi_{\text{phist}}(P)$ is a tree partition and by Observation 2.2.2, we know that $\pi^f_{\text{phist}}(P)$ can be constructed from phist(P) in linear time. We believe that there are direct applications for the PHistFVC rectilinear visibility cover class in Computational Geometry. For example it is easily shown that the problem of finding the shortest diagonal in a simple polygon is reducible to the same problem on pseudo-histogram family polygons by means of these covers; see **[28]**. Our main interest in PHistFVCs here, however, is as a stepping stone in the construction of rectilinear visibility polygon covers composed of even simpler subpolygons.

6.6.1. The SAMPc of a Pseudo-Histogram Family Polygon. Consider the construction of SAMPc(Q) where Q is a pseudo-histogram family polygon. All the results on SAMPcs are given in Subsection 6.4.2 except for the following result.

LEMMA 6.6.5. $\|SAMPc(Q)\| \leq 3.5|Q|$.

PROOF. Each convex vertex of Q is a vertex of at most three elements of SAMPc (Q) but in the case that this value is three, the vertex prevents the use of two Steiner points. Thus, for accounting purposes, we assume that no convex vertex is used more than twice. Also, each reflex vertex of Q is a vertex of at most three elements of SAMPc (Q); note that a reflex vertex cannot be used on four elements of SAMPc (Q) because SAMPc (Q) contains no duplicates and no polygons that are contained in other elements of SAMPc (Q). Also, a reflex vertex may cause the use of as many as two Steiner points in total. However, for each reflex vertex that is used in more than two elements of SAMPc(Q) or that causes the use of more than one Steiner point, there is a convex vertex that can be mapped uniquely to this reflex vertex. This convex vertex is contained in the element, say H, of $\pi_{\text{phist}}(Q)$ for which the reflex vertex is the destination vertex of base (H). Thus the vertices of Q can be partitioned into pairs such that no pair can account for more than seven vertices of the elements of SAMPc (Q). Therefore $\|\text{SAMPc}(Q)\| \leq 3.5\,|Q|$. □

Of particular interest with respect to pseudo-histogram family covers, is that Lemma 6.4.1 applies only to rectilinear polygons; for non rectilinear polygons this is not generally true. However, any rectangle, say R, not covered by $\pi^{c}_{\text{saml}}(P)$ must be contained in some element $Q \in \pi_{\text{phist}}(P)$ and furthermore, any such rectangle is not entirely contained in $\text{vis}_{\perp}(\text{base}(Q))$. Thus, for each $Q_i \in \pi_{\text{phist}}(P)$ we need to construct a rectangle visibility cover of Q_i including $\text{vis}_{\perp}(\text{base}(Q_i))$.

6.6.2. A SAM Subpolygon Rectangle Visibility Cover of a Pseudo-Histogram Polygon. Generally, not all the elements of this cover will be SAM polygons; in some cases some of them will be quadrilaterals. However, finding the largest rectangle inscribed by a quadrilateral can be done in $O(1)$ time in a relatively

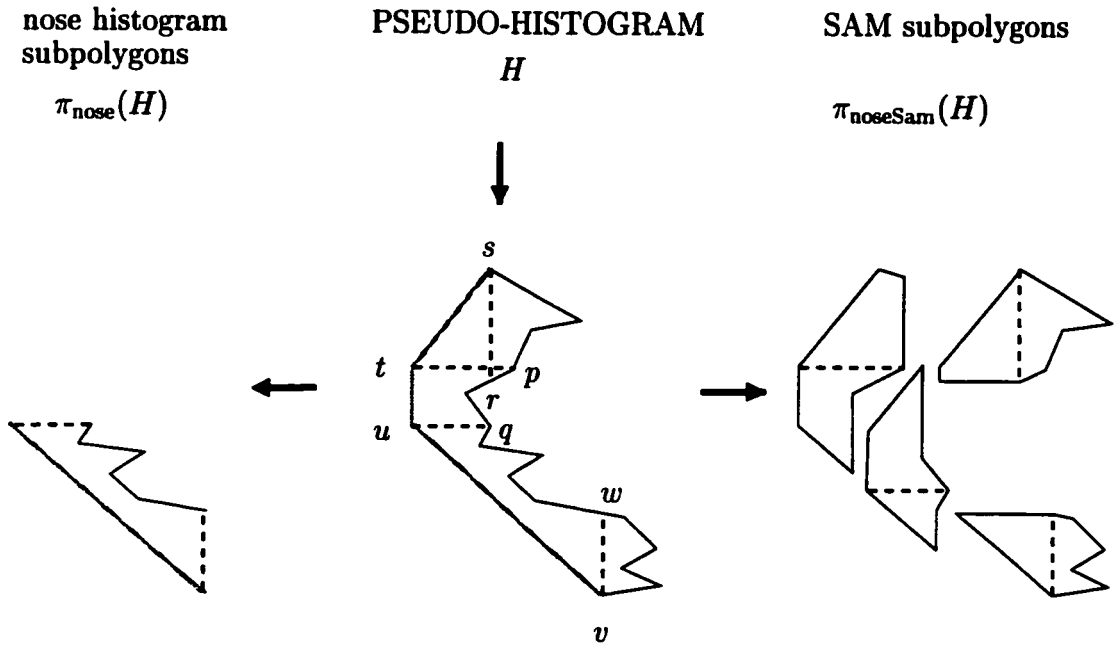
straightforward manner[4] so the complexity of our algorithm will still be determined by the cost of finding a largest rectangle inscribed in the SAM subpolygons.

Let $H = \{v_0, \ldots, v_{m-1}\}$ be a pseudo-histogram polygon and assume without loss of generality that base(H) is vertical and directed downward. We also assume that $|\text{baseChain}(H)| = 3$; the case that $|\text{baseChain}(S)| < 3$ is handled in a similar but simpler manner and is omitted. Let $B = \text{vis}_\perp(\text{base}(H))$. We now construct a rectangular visibility cover of H including B.

Let $U = \{u_0, \ldots, u_{k-1}\}$ be an k vertex polygon. We say that U is a *nose polygon* if edges $\overline{u_1u_2}$ and $\overline{u_{k-1}u_0}$ are both axis-aligned and perpendicular and if, for some edge $e \in \{\overline{u_1u_2}, \overline{u_{k-1}u_0}\}$, we have that, for every point p interior to U, there is a point, say q, on edge $\overline{u_0u_1}$ such that the line segment $\overline{pq}$ is interior to U except at q and is parallel to e. We call $\overline{u_0u_1}$ the *base edge* of U and chain $u_{k-1}u_0u_1u_2$ the *base chain* of U. Note that base(U) is not axis-aligned. We construct a cover of H by nose histograms and SAM polygons as follows.

Let $\text{baseChain}(H) = stuv$ where $v_0 = t$. Let $\overline{rs}$ and $\overline{vw}$ be vertical chords of H; the degenerate and simpler cases that $\overline{rs}$ or $\overline{vw}$ or both don't exist are omitted. Let $\overline{tp}$ and $\overline{uq}$ be horizontal chords of S. If $\overline{rs} \cap \overline{tp} - \{r\} \neq \emptyset$ then let S be the set containing the largest subpolygon of H containing both $\overline{rs}$ and $\overline{tp}$ as edges; otherwise let $S = \emptyset$. If $\overline{vw} \cap \overline{uq} - \{w\} \neq \emptyset$ then let T be the set containing the largest subpolygon of H containing both $\overline{vw}$ and $\overline{uq}$ as edges; otherwise let $T = \emptyset$. We denote $\pi_{\text{noseSam}}(H) = \{\text{vis}_\perp(\overline{rs}), \text{vis}_\perp(\overline{tp}), \text{vis}_\perp(\overline{vw}), \text{vis}_\perp(\overline{uq})\}$ and $\pi_{\text{nose}}(H) = S \cup T$. Let $\Pi = \pi_{\text{noseSam}}(H) \cup \pi_{\text{nose}}(H)$. Then it is easily established that $\mathbb{C}_\square(\Pi \cup \{B\}) = 1$ and $\|\Pi\| = O(m)$. Furthermore, Π can be constructed $\Omega(m)$ time. Now, since $\pi_{\text{nose}}(H)$ is

[4]In fact matters are easier than for arbitrary quadrilaterals. These quadrilaterals have a pair of non adjacent edges such that one is horizontal and one is vertical. As a result one of these axis aligned edges (and it is easy to determine which one) contains an edge of the largest rectangle in the quadrilateral and furthermore, one of the endpoints of this edge is an endpoint of the rectangle. The set of rectangles satisfying these constraints is describable by a function in one variable. The largest rectangle is then found by simple calculus.



A pseudo-histogram can be covered by at most four SAM polygons and two nose histogram polygons such that the rectilinear visibility cover number of the cover is one

FIGURE 6.6.1. Covering a pseudo-histogram

empty or contains nose histograms and $\pi_{\text{noseSam}}(H)$ contains SAM polygons, we have reduced the LR problem on pseudo-histograms to the LR problem on nose histograms or SAM polygons and furthermore, this reduction requires $\Omega(m)$ time and space; see Figure 6.6.1.

We next reduce the LR problem on nose histograms to the LR problem on SAM polygons.[5] Let N be a nose histogram with base chain $tuvw$. For simplicity we assume that the edges of N other than $\overline{tu}$ and $\overline{vw}$ are not axis-aligned; it is not difficult to deal with additional axis-aligned edges and we omit the details. Assume without loss of generality that the start vertex is u and u is the top leftmost vertex of N. We will

[5]In fact this is is a poor idea in the sense that (we claim) the LR problem on a nose polygon can be solved in linear time but a better algorithm would not help as long as the cost solving the LR problem on an m vertex SAM polygon is $O(m \log m)$.

decompose N into a cover Π such that $\mathsf{C}_{\square}(\Pi) = 1$ and Π contains only SAM polygons and quadrilaterals. Initially $\Pi = \emptyset$.

Let s be the point on edge $\overline{uv}$ such that $\overline{ws}$ is a horizontal chord of N. If w is a reflex vertex then let w' be the point on N such that $\overline{ww'}$ is a vertical chord of N; otherwise let $w' = w$. Let r be the destination vertex of the edge containing w'; if w' is a vertex of N then we choose the edge for which w' is the source vertex. See Figure 6.6.2. For any point p we denote the x-coordinate of p by $x(p)$. There are two cases to consider:

(1) $x(r) \leq x(s)$. Let r_h and r_v be the points on edge $\overline{uv}$ such that $\overline{rr_h}$ is a horizontal chord of N and $\overline{rr_v}$ is a vertical chord of N. Then we add quadrilateral $vw'rr_h$ to Π and, if $w' \neq w$, we add polygon $\text{vis}_{\perp}(\overline{ww'})$ to Π as well. Let N_1 be the subpolygon of N determined by chord $\overline{rr_v}$ and containing vertex u. Now, any rectangle, say R, in N must be contained in either quadrilateral $vw'rr_h$, polygon $\text{vis}_{\perp}(\overline{ww'})$, or N_1. Otherwise R would need to intersect the area of overlap of $vw'rr_h$ and N_1 and then R would need to intersect both $\overline{rr_h}$ and $\overline{rr_v}$ which is impossible.

(2) $x(r) > x(s)$. Let s_v be the point on the subchain of H from w' to t such that $\overline{ss_v}$ is a vertical chord of N. We compute the SAM subpolygons $S_h = \text{vis}_{\perp}(\overline{ws})$ and $S_v = \text{vis}_{\perp}(\overline{ss_v})$ of N, and add them to Π. Let N_1 be the subpolygon of N determined by chord $\overline{ss_v}$ and containing vertex u. Now, any rectangle, say R', in N must be contained in one of S_h, S_v, or N_1. Otherwise R' would need to intersect the area of overlap of S_h and S_v or the overlap of S_v and H_1. But, in the first case, R' would then need to intersect both $\overline{ss_v}$ and $\overline{ws}$, which is impossible. In the second case, R' would need to intersect both $\overline{ss_v}$ and the horizontal chord of H with an endpoint at s_v,

which is also impossible. Note that if $w' \neq w$ then $\text{vis}_{\perp}(\overline{ww'})$ is contained in S_ν and thus is not put into Π.

Assume that N is represented by a doubly linked list of its vertices. Then, for either of the above cases, we can construct the subpolygons to put into Π and the subpolygon N_1 as a doubly linked list in $\Omega(|N| - |N_1|)$ time and space. Thus, we can repeat the above procedure until a SAM polygon is added to Π that has u has a vertex in $\Omega(|N|)$ time and space. We denote Π by $\pi^{c}_{\text{samQuad}}(N)$.

Now, by an inductive argument, it is clear that any rectangle in N is contained in an element of Π. We denote

$$\pi^{\text{pc}}_{\text{sam}}(H) = \pi_{\text{noseSam}}(H) \cup \Big(\bigcup_{Q_i \in \pi_{\text{nose}}(H)} \pi^{c}_{\text{samQuad}}(Q_i) \Big).$$

We summarize the above discussion with the following lemmas:

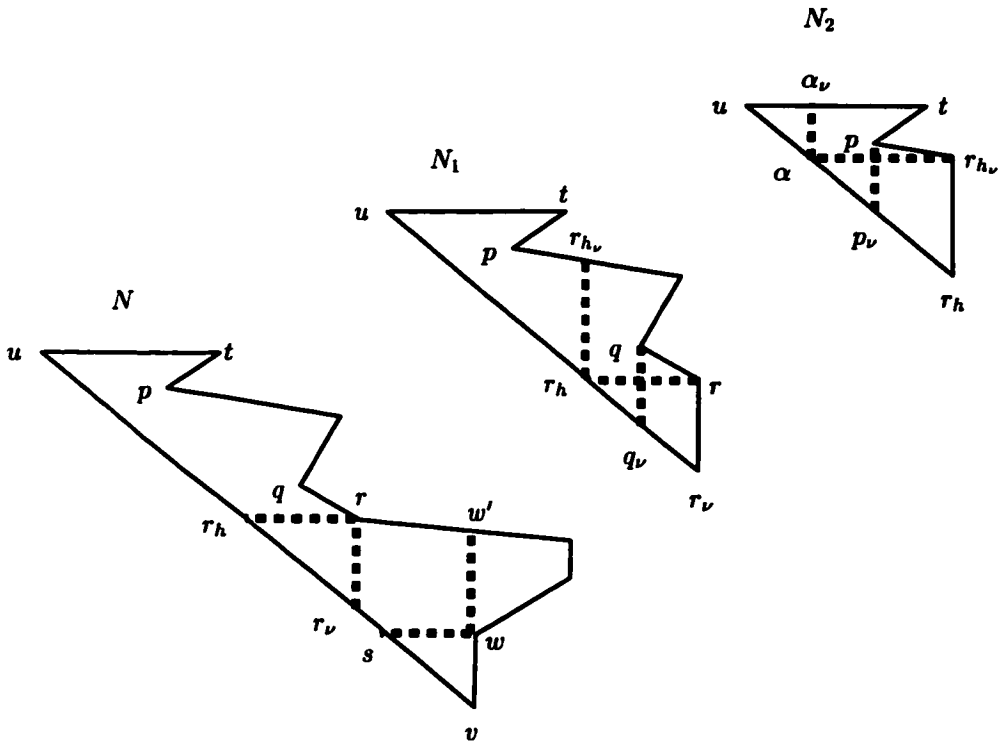
LEMMA 6.6.6. *$\pi^{\text{pc}}_{\text{sam}}(H) \cup B$ is a cover of H and $\mathsf{C}_{\square}(\pi^{\text{pc}}_{\text{sam}}(H) \cup B) = 1$.*

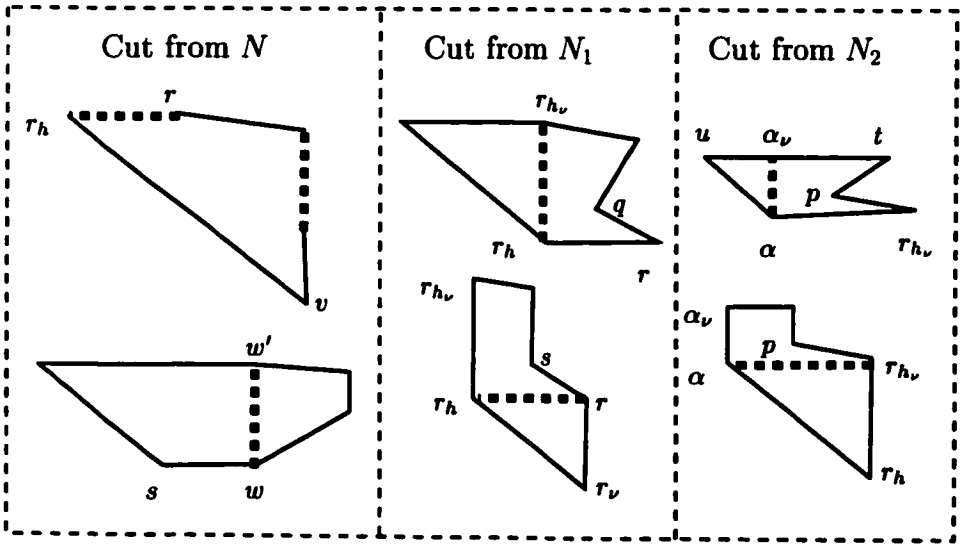
LEMMA 6.6.7. *$\pi^{\text{pc}}_{\text{sam}}(H)$ can be built in $O(|H|)$ time.*

LEMMA 6.6.8. $\|\pi^{\text{pc}}_{\text{sam}}(H)\| \leq 9n$.

PROOF. We consider only the case that H is a nose histogram. The more general case can be shown not to result is a larger cover. It is easily shown that the worst case scenario during the construction of $\pi^{\text{pc}}_{\text{sam}}(H)$ is that we add two SAM subpolygons to $\pi^{\text{pc}}_{\text{sam}}(H)$ but the size of the next nose histogram constructed has only one vertex less than the previous one. The subpolygons added to $\pi^{\text{pc}}_{\text{sam}}(H)$ in this situation can have as many as nine vertices in total. □

We now have the tools we need to modify Algorithm 10 to construct a SAM subpolygon rectangle visibility cover of a simple polygon. This is the subject of the next section.





A cover of a nose histogram N by SAM polygons and quadrilaterals with a rectilinear cover number of one

FIGURE 6.6.2. A cover of a nose histogram

6.6.3. A SAM Subpolygon Rectangle Visibility Cover of a Polygon. In this subsection we present Algorithm 11 which constructs a SAM subpolygon rectangle visibility cover with a rectangle visibility cover number of one for any simple polygon.

For any simple polygon P, we denote the cover of P constructed by applying Algorithm 11 to P by $\pi^c_{\text{sam2}}(P)$. The key properties of $\pi^c_{\text{sam2}}(P)$ are captured in the following lemmas:

Algorithm 11

Input: A simple polygon P.
Output: $\pi^c_{\text{sam2}}(P)$.
1: $\Pi := \{\text{vis}_\perp(\psi(P))\}$.
2: **for** each pseudo-histogram family polygon $Q \in \pi^f_{\text{phist}}(P)$ **do**
$\Pi := \Pi \cup \text{SAMPc}(Q)$.
3: **for** each pseudo-histogram polygon $U \in \pi_{\text{phist}}(P)$ **do**
$\Pi := \Pi \cup \pi^{\text{pc}}_{\text{sam}}(U)$.
4: Return Π.

LEMMA 6.6.9. *$\pi^c_{sam2}(P)$ is a cover of P and $\mathbb{C}_\square(\pi^c_{sam2}(P)) = 1$.*

PROOF. Let R be any rectangle contained in P. We need to show that R is contained in some element of $\pi^c_{\text{sam2}}(P)$. First, R is contained in an element, say Q, of $\pi^f_{\text{phist}}(P)$ since $\mathbb{C}_\square(\pi^f_{\text{phist}}(P)) = 1$ by Lemma 6.6.2. Let W be the element of $\pi_{\text{phist}}(Q)$ such that $\text{door}(Q) = \text{door}(W)$. Let $B = \text{vis}_\perp(\text{base}(W))$ as computed for W and T be the element of $\pi^f_{\text{phist}}(P)$ such that $\text{base}(T) = \psi(P)$. If $\text{base}(T) \neq \text{base}(Q)$ then let U be the of element of $\pi^f_{\text{phist}}(P)$ such that $\text{base}(Q)$ is a chord of U. Now, using arguments similar to those used in Lemma 6.4.1, it can be shown that R is contained in either B, some element of $\text{SAMPc}(Q)$, or some element of $\pi^{\text{pc}}_{\text{sam}}(W)$.

In the first case, R is contained in an element of $\pi^c_{\text{sam2}}(P)$ by Step 2 of Algorithm 11.

In the second case, R is contained in an element of $\pi^c_{\text{sam2}}(P)$ by Step 3 of Algorithm 11.

In the third case, either $Q = T$ and then R is contained in $\text{vis}_\perp(\psi(P))$ which is an element of $\pi^c_{\text{sam2}}(P)$ by Step 1 of Algorithm 11 or $Q \neq T$ and then R is contained in an element of $\text{SAMPc2}(U) \subset \pi^c_{\text{sam2}}(P)$ by Step 2 of Algorithm 11. □

LEMMA 6.6.10. $\|\pi^c_{sam2}(P)\| < 9|P|$.

PROOF. It is clear that the worst case scenario is that P is a nose histogram polygon. The result now follows from Lemma 6.6.8. □

LEMMA 6.6.11. $\pi^c_{sam2}(P)$ *can be constructed in* $O|P|$ *time.*

PROOF. It was shown in [34] that $\pi_{\text{phist}}(P)$ can be constructed in $O|P|$ time. Also, it is straightforward to construct $\pi^f_{\text{phist}}(P)$ from $\pi_{\text{phist}}(P)$ in linear time. Finally, $\pi^c_{\text{sam2}}(P)$ can be constructed from $\pi^f_{\text{phist}}(P)$ in $O(|P|)$ time by Lemmas 6.4.3, 6.6.7, and 6.6.4. □

6.6.4. The Largest Axis-Aligned Rectangle Problem. The LR problem can be solved for a SAM polygon with m vertices in $O(m \log m)$ time [21]. From this fact and Lemmas 6.6.9 and 6.6.11 the following result follows:

THEOREM 6.6.12. *Let* P *be a simple polygon with* n *vertices and let* k *be the size of the largest element of* $\pi^c_{sam2}(P)$ *by vertex count. Then the LR problem can be solved on* P *in* $O(n \log k)$ *time.*

This is an improvement by a factor of $O(\log n)$ over the algorithm in [21]. We made this improvement by replacing one rectangle visibility cover by another more efficient one in that the new cover can be constructed in $O(\log n)$ less time and uses $O(\log n)$ less space. However, since it is not necessary to store all of the cover at once in order to solve the LR problem and the algorithm in [21] avoids doing so, there is no corresponding improvement in space efficiency.

CHAPTER 7

Conclusion and Future Research

Algorithms that construct visibility decompositions of polygons have been used before but they have not been well recognized as a tool for solving geometric problems. In this thesis we have unified, clarified and refined the concept of visibility decompositions. We have shown that the systematic study of visibility decompositions of polygons can lead to useful tools and applications in the development of geometric algorithms. We plan to continue research into this area over the next several years and expect to further demonstrate its value by solving more problems.

In the meantime the problems we have studied have resulted in many new problems and ideas to study beyond what is covered in this thesis. We now present some of these new questions and ideas.

7.1. Chapter Three

In Chapter 3 we showed that a simple polygon can be partitioned into 2^*-wind polygons by a simple algorithm (after triangulation). We then studied the problem of partitioning a polygon into the minimum number of y-monotone subpolygons. We developed an algorithm that determines what this number is in linear time. The algorithm also determines a maximum set of pairs of points that can be joined by a set of non-intersecting y-monotone chains so as to construct such a partition. Constructing such a partition can take as much as $O(n^2)$ space. However, we have not given an algorithm to construct the actual partition.

It would be interesting to put constraints on the size of individual y-monotone chains used for this partition or on the total size of these chains. The case that these

chains must all have length one has been studied before; see Section 3.6. However, perhaps with the new tools from this chapter more efficient algorithms can be found in this case.

In this chapter we defined k^*-wind polygons. However we have not presented any applications for k^*-wind polygon when k has any value other than 2. We have however studied this problem. We believe that we can solve problems when k has the value 1.5. More difficult is the case when $k > 2$. While we currently have no applications of 3^* -wind polygons partitions it is noteworthy that our first optimal solution to the area problem of Appendix B used a cover by 3^* -wind polygons. It was quite complicated compared to the solution in Appendix B. Nevertheless, it demonstrates that such decompositions can be used to solve problems.

Although we do not investigate problems on polygons with holes in this thesis the simplicity of the 2-wind visibility partitioning algorithm suggests that it might also be made to work for polygons with holes. This is indeed the case. It is possible to partition a simple polygon with holes into subpolygons with holes such that the subpolygons are 2^*-wind and the holes are also (a variation of) 2^*-wind polygons. This can be done by an algorithm conceptually similar to breadth first search. In Algorithm 12 we provide a basic version of an algorithm to solve this problem.

We can refine Algorithm 12 such that the resulting algorithm runs in linear time if the initial polygon with holes has already been partitioned into trapezoids. Let Π be the partition of P returned by Algorithm 12. It is also easily shown that any y-monotone chain interior to P intersects at most two elements of Π. With this partition we can now deal with problems on polygons with holes that are easier to solve if the polygon is 2^*-wind and its holes are 2^*-wind like. An investigation of such problems is in progress.

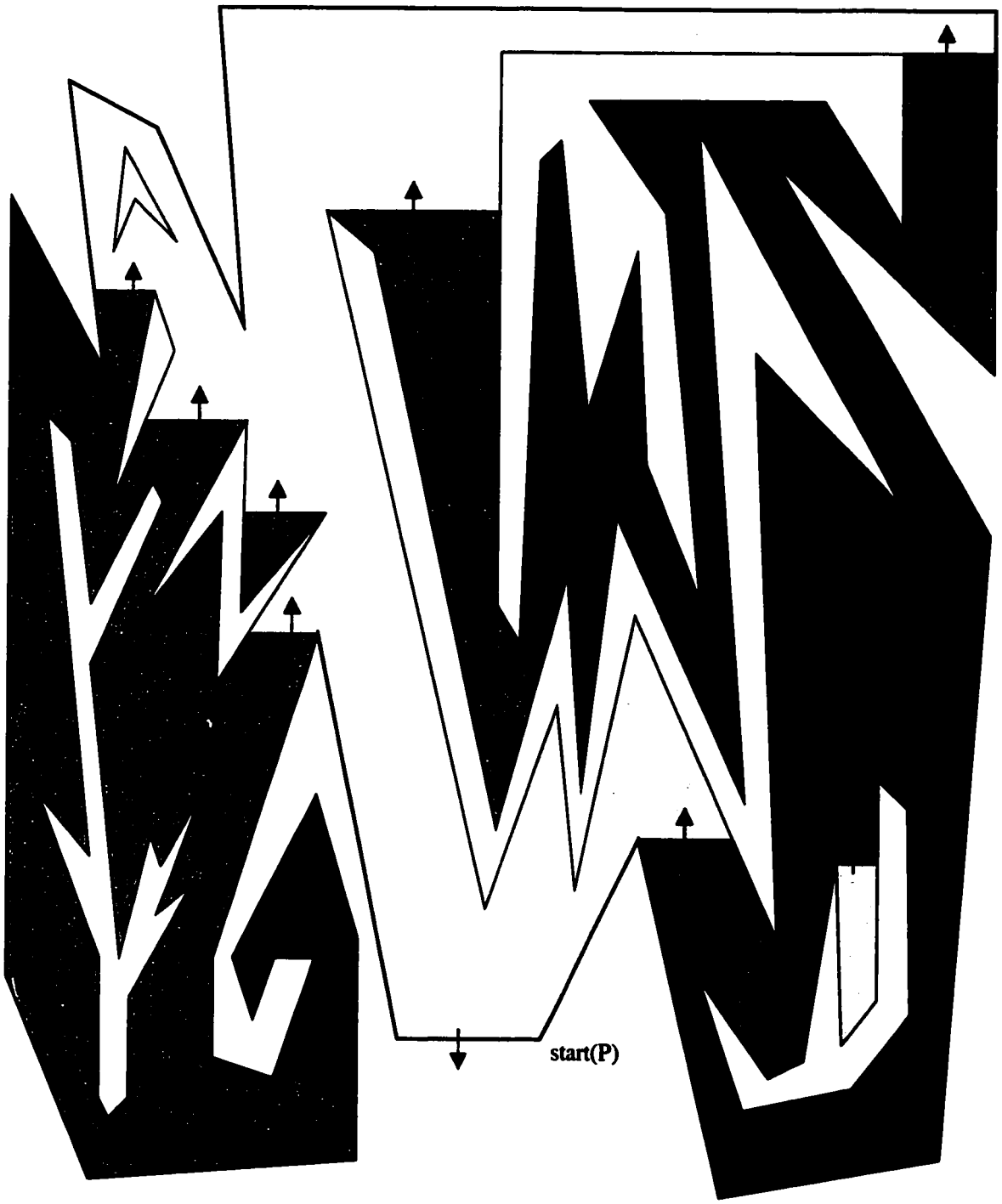


FIGURE 7.1.1. 2*-wind partition of a polygon with holes

Algorithm 12

Input: P A polygon with holes.
Output: A visibility partitioning of P into 2^*-wind subpolygons and 2^*-wind like holes.

1: $\alpha := 0\pi$.
2: doors := $\{\psi(P)\}$.
3: $\Pi := \emptyset$.
4: **repeat**
5: Construct the set of maximum subpolygons with holes, say Π', such that for each subpolygon with holes $Q_i \in \Pi'$, any point in the interior of Q_i is reachable from some element $d_j \in$ doors by an α-1-wind chain starting from a point belonging to d_j. Note that Q_i may have multiple doors.
6: $\Pi := \Pi \cup \Pi'$.
7: windowsAndDoors := the set of chords of P that are on the boundary of some element of Π'.
8: doors := windowsAndDoors − doors.
9: $\alpha := -\alpha$.
10: **until** doors $= \emptyset$.
11: return Π.

Another approach to generalizing the results of this chapter is to work in three dimensions (or higher). Consider that we have a polyhedra P in three dimensions. Let p be a lowest point on P. Let Q be the maximal sub-polyhedra of P such that any point interior to Q is reachable by any y-monotone chain starting at p and interior to P. (We mean that a chain C is y-monotone if any horizontal plane intersects C at most along one subchain of C.) Now let $\Pi = \{Q\}$. Let the parts of the boundary of Q that are not on the boundary of P be the windows of Q. Now construct the set of maximal sub-polyhedron of P whose interiors are reachable from a window of Q by a y-monotone chain interior to P without entering Q and add these polyhedron to Π. Note that a sub-polyhedron may have multiple windows of Q on its boundary. Now repeat this process until P has been completely partitioned. We call the sub-polyhedron constructed by this process 2^*-wind polyhedron. This partition has the property that any y-monotone chain interior to P intersects at most of the 2^*-wind polyhedrons in Π. We point out that this partition of P may require $O(n^2)$ space

where n is the number of vertices of P. It would also be interesting to determine classes of polyhedron for which this partition requires $o(n^2)$ space. We are not sure of the time complexity of this algorithm.

7.2. Chapter Four

In Chapter 4 we studied the partitioning of polygons into y-monotone subpolygons. We developed two algorithms that construct y-monotone visibility partitions of simple polygons. For each algorithm the monotone cover number of the partition constructed for a polygon P is $O(\log n)$.

The first of these algorithms constructs a partition that is within two of the minimum monotone cover number of any partition of P by chords. We then showed that if the partition is restricted to being constructed using horizontal chords then this partition has a monotone cover number that is within one of the minimum. Finally we showed how to modify this partition such that in the restricted case the modified partition is optimal with respect to horizontal monotone cover number.

An interesting question is then "Is the modified partition within one of the optimal for the unrestricted case?". We believe this to be the case.

CONJECTURE 7.2.1. *For an input simple polygon P Algorithm 4 constructs a partition of P such that the monotone cover number of the partition is within one of the minimum monotone cover number of any partition of P.*

Assuming that this conjecture is true the obvious next question is "Can we construct a partition of any polygon such that the monotone cover number of the partition is minimum by modifying the partition returned from Algorithm 4?". We believe that this can be done. We make this into a conjecture.

CONJECTURE 7.2.2. *A partition of a simple polygon into subpolygons by chords such that the monotone cover number of the partition is minimum can be constructed in time polynomial in the size of the polygon.*

We used the second of these algorithms in solving the circular ray shooting problem. We believe that there are other applications of both of these algorithms. Our research so far looks promising.

7.3. Chapter Five

In Chapter 5 we studied the circular ray shooting problem. We showed that we could solve this problem using a factor of $O(\log n)$ less preprocessing time than the algorithm in [16]. The space requirement is better in our algorithm than the results in [16] by a constant factor. More importantly our algorithm is simpler and, we believe, more easily applied to other problems.

In [16] it was shown that if the query arc was restricted to having a fixed radius (say one) then preprocessing the circular ray shooting problem can be solved using $O(n \log n)$ preprocessing time, $O(n)$ space, and $O(\log n)$ query time. We believe that by replacing their hierarchical decomposition of a polygon by our monotone subpolygon visibility partition of the polygon we can reduce the preprocessing time to linear. Furthermore we expect that our algorithm will be simpler. This is under investigation.

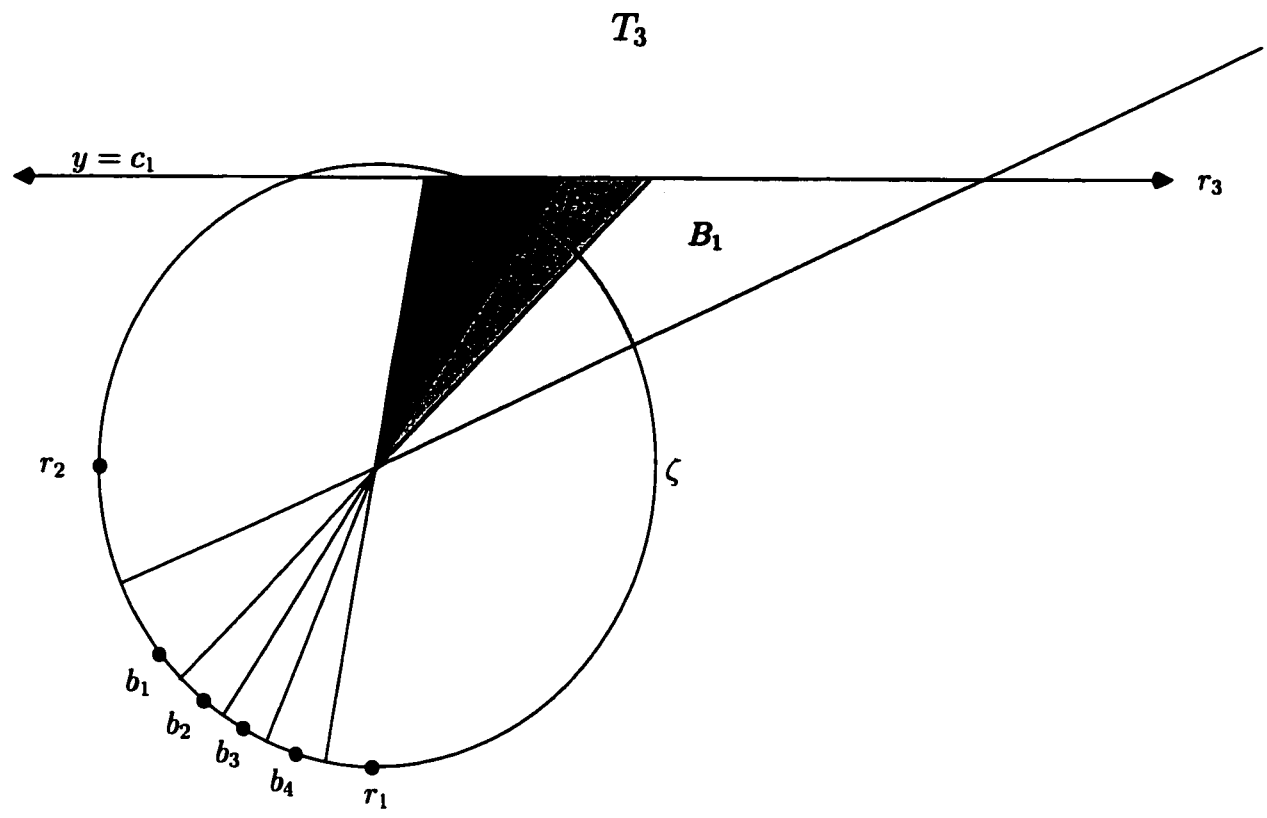
There is also some potential for improving the algorithm we presented for solving the circular ray shooting problem. Let Q be a y-monotone polygon and assume that we have constructed the an edge Voronoi diagram of Q_l. Now our algorithm used every edge in Q_l in this Voronoi diagram. However it may be that many of these edges cannot be the first edge intersected by canonical circular arc of Q. Thus it would be profitable in space and query time to determine these edges and discard

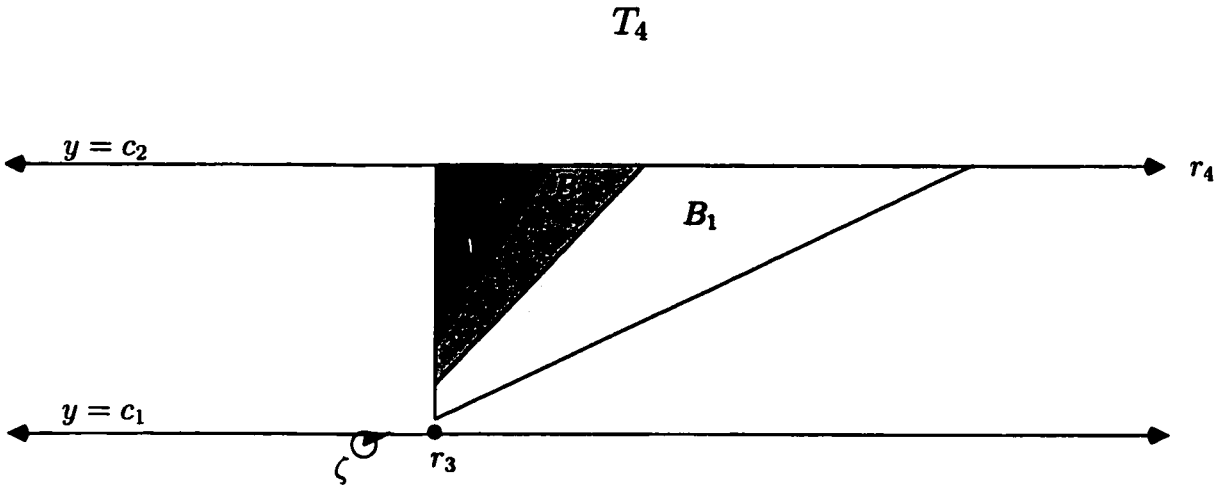
them before constructing the Voronoi diagram. A similar argument can be made for the farthest point Voronoi diagram of Q_r.

It may also be possible to make some saving in terms of how we test if the arc intersects Q_r (or Q_l). The algorithm tests the center of the circle against $O(\log n)$ Voronoi diagrams. Each of these tests are done without using any information about the location of the center of the circle in the previous Voronoi diagram. If information about the previous test were used in the subsequent test perhaps it could reduce the cost of performing the later test. We expect this modification to make a significant improvement on the expected query time and might improve the worst case query time.

One difficulty we had with the circular ray shooting problem was the need to partition the arc into as many as five parts. This requires that we needed two copies of our data structure thus doubling the preprocessing time and space required. In the worst case the query time also doubles. If, instead, we could work with semi-circular arcs instead of quarter circle arcs then this difficulty would go away. For the edge Voronoi diagrams we can easily use semi-circular arcs instead of quarter circle arcs; see Subsection 5.5.1. The problem is more difficult for the farthest point Voronoi diagrams. We have investigated this problem without success. One idea we came up with is to use an asymmetric Voronoi diagram.

For two arbitrary points a and b on the plane we denote the distance measure $\text{dist}_y(a,b) = \text{dist}(a,b)$ if $a_y \leq b_y$ and $\text{dist}_y(a,b) = 0$ otherwise. Note that this distance measure is not a metric since $\text{dist}(a,b) = \text{dist}(b,a)$ only if $a_y = b_y$. We call the farthest point Voronoi diagram based on this distance measure the *asymmetric farthest point Voronoi diagram* which we shorten to the *AFP Voronoi diagram.* For a query point q we refer to the site in the AFP Voronoi diagram associated with the region containing q as the asymmetric farthest site from q.





The diagrams T_3 and T_4 when $S_b = \{b_1, b_2, b_3, b_4\}$ and $S_r = \{r_1, r_2, r_3, r_4\}$. The regions associated with b_i are marked B_i, $1 \leq i \leq 4$.

FIGURE 7.3.1. An Asymmetric Voronoi diagram using $O(n^2)$ space

Unfortunately the cost of constructing the AFP Voronoi diagram can be much higher than for constructing the standard farthest point Voronoi diagram. In fact the AFP Voronoi diagram may require $O(m^2)$ edges and faces for a diagram consisting of m sites. Let S be a set of points. The reason that this can happen is that a point p_i of S may have $O(|S|)$ regions for which it is the furthest point of S by the distance measure dist_y and furthermore there can be $O(|S|)$ points of S for which this is true.

We show this by constructing a point set S for which the AFP Voronoi diagram has $O(|S|^2)$ edges and faces. We denote the AFP Voronoi diagram of S by $\mathbf{V}_A(S)$. To begin, assume S consists of two equal size subsets $S_b = \{b_1, b_2, \ldots, b_m\}$ and $S_r = \{r_1, r_2, \ldots, r_m\}$. We will select these points such that for each pair of points (p, q) with $p \in S_b$ and $q \in S_r - \{r_1, r_2\}$ there is a region for which p is the farthest point of any point in S by distance measure dist_y. To simplify matters we will construct a simplified diagram by merging all the regions of $\mathbf{V}_A(S)$ for which the farthest point is an element of S_r. We refer to this region by T_r. To simplify matters we will also add to T_r the region for which there is no farthest point in S. Let T be this simplified diagram. We show that T has $O(|S|^2)$ edges and $O(|S|^2)$ faces. For each region t of T we either associate t with a point $s \in S_b$ or we associate t with the set of points T_r. We further denote the set of regions associated with a point $s \in S_b$ by T_s. Our construction has the property that T_s contains $m - 1$ regions of T for each point $s \in S_b$. Furthermore, each region t_i in T_s is contained in a region t_j of $\mathbf{V}_A(S)$ whose asymmetric farthest site is s and whose boundary contains at least as many edges as t_j. Thus T has $O(m^2)$ edges and so does $\mathbf{V}_A(S)$. We now show how to construct T.

We construct T by constructing a sequence of diagrams $T_1, T_2, \ldots, T_m = T$ where T_1 is the diagram constructed based on the point in S_b and T_i, $2 \leq i \leq m$, is the diagram constructed based on the points $S_b \cup \{r_1, r_2, \ldots, r_i\}$.

We construct T_1 as follows. Let ζ be a circle and let the points of S_b be distributed in reverse order strictly between the six o'clock and nine o'clock position on ζ. Now T_1 is just the farthest point Voronoi diagram of S_b.

We construct T_2 from T_1 as follows. We place r_1 at the bottom of ζ and r_2 at the left most point on ζ. We now add to T_r all territory on or below the line $y = (r_1)_y$. We also add to T_r all territory above this line and further from r_1 or r_2 than any point in S_b. The resulting diagram is T_2.

We construct T_3 from T_2 as follows. Let $y = c_1$ be a line above ζ_c and let r_3 be a point on the line $y = c_1$ and sufficiently far to the right that any point q_1 on the line is farther from one r_2 or r_3 than q_1 is from any point in S_b. We now add to T_r all territory that is both on or above the line $y = c_1$ and on or to the left of the vertical line containing r_3. This construction partitions each region associated with a point in S_b into two regions, one between the lines $y = \zeta$ and $y = c_1$ and one above the line $y = c_1$ and to the right of r_3. The latter region is unbounded. The resulting diagram is T_3. See Figure 7.3.1.

We construct T_4 from T_3 as follows. Let $y = c_2$ be a line sufficiently above the line $y = c_1$ that, for each point $b_i \in S_b$, the highest region associated with point b_i is intersected by the line $y = c_2$. Let r_4 be a point on the line $y = c_2$ and sufficiently far to the right that any point q_2 on the line is further from one of r_2 or r_4 than q_2 is from any point in S_b. We now add to T_r all territory that is both on or above the line $y = c_2$ and on or to the left the vertical line containing r_4. For each point $b_i \in S_b$, this construction partitions the highest region associated with b_i into two regions. One of the two regions is below the line $y = c_2$ and one is above and to the right of r_4. The latter region is unbounded. The resulting diagram is T_4.

We repeat the pattern of the above constructions until all points in S_r have been added and $T_m = T$ has been constructed. Now each point in S_b has $m - 1$ regions

associated with it so there are $m * (m-1) + 1$ regions in total in T. Furthermore T satisfies the requirements stated for it and so $\mathbf{V}_A(S)$ must contain $O(m^2)$ regions as well.

Although this result points out that it may be expensive to construct T, matters may not be as bad as they appear. The distribution of points we used to construct our example is contrived so for reasonably distributed sets of points it may be that the AFP Voronoi diagram of a set of points can be constructed at a reasonable cost.

In the case of the circular ray shooting problem we also have the advantage that the number of vertices in the AFP Voronoi diagrams may be much smaller than the number of vertices in the polygons from which they are constructed. This is in part because for the farthest point Voronoi diagrams only the points on the convex hull are sites of the diagram and a similar (but different) situation occurs for the AFP Voronoi diagram.

Perhaps there is another way to avoid cutting circular arcs into quarter circles and the associated costs. But we have abandoned the search, at least for now.

7.4. Chapter Six

In Chapter 6 we studied the problem of finding the largest axis-aligned rectangle inscribed by a simple polygon. In [21] the authors solved this problem by reducing the problem to that of finding the largest axis-aligned rectangle inscribed by a SAM polygon. This reduction was achieved by constructing a rectangle visibility cover of a simple polygon into SAM polygons such that the rectangle visibility cover number is one. This cover requires $O(n \log n)$ time and space to construct for an n vertex polygon. The authors then showed that this problem could be solved on a SAM polygon in $O(m \log m)$ time for a m vertex SAM polygon. Thus they solved this problem for an n vertex simple polygon in $O(n \log^2 n)$ time. We were able to reduce

this cost to $O(n \log n)$ by constructing a cover of a simple polygon by SAM polygons that required only linear time and space.

In this chapter the largest rectangle was defined according to its area. We could also define the largest rectangle according to the length of its perimeter or diagonal. Our reduction of the problem to the SAM polygon case works for these cases also. However a solution for this problem for these cases needs to be found.

We also expect that this cover class and variations of it can also be used to solve other problems.

The rectangle visibility cover class presented in Chapter 6, in contrast to the decompositions studied in this thesis, is generated by a divide and conquer algorithm. As a result the amount of space required to store this cover is $O(n \log n)$.[1] There are probably many other algorithms in the Computational Geometry literature that construct visibility covers using divide and conquer and may suffer from requiring super-linear time and space to construct. It may be a profitable task to find them and see which ones can be replaced by algorithms that are more efficient.

7.5. Appendix A

In Appendix A we studied three versions of a range query problem. For the first version of the problem we provided a very simple solution using an array. This solution is based upon the idea of a heap and is closely connected to a solution to the same problem using a balanced binary tree. An interesting point about this solution is that we do not need to assume that the tree is binary. We can also implement in an array a solution corresponding to a balanced w-way tree. Basing our solution on a w-way tree may have applications in databases where w would correspond to the

[1]The algorithm for finding the largest inscribed rectilinear rectangle in [21] does not store the entire rectangle visibility cover at once so it only requires linear space.

number of records that we could store in a block of data that could be fetched in one disk access. Algorithm 13 is a variation of Algorithm 15 but based on w-way trees.

We need the following modifications:

- n needs to be a multiple of w.
- H now requires $nw/(w-1)$ entries. Note that larger values of w imply less space is needed for H.
- During the up phase we need to test all the siblings to the right of the current node that have the same parent to see if any of them cause the down phase to be initiated.
- During the down phase we need to check the child nodes in left to right order and choose the first one satisfying the dot operator test to continue the search.

With these changes the Algorithm 13 is the result. Note that the algorithm uses sequential search within each block. An obvious improvement is to use Algorithm 15 for searching within a block.

Our second algorithm relates to range queries in arrays where each data element has a weight associated with it which is used to indicate to what degree each element is to be favored when searching for a range. We solved this problem by creating a new kind of weighted binary tree. There are a number of open problems related to the algorithm that builds the weighted binary tree.

First of all the weighted binary tree favors searches to the right. The cost of searches to the right are proportional to the log of the sum of the weights of the elements in the range found. But we have only shown that searches to the left can cost $O(\log n)$ time where n is the size of the array preceding the start point even if the total of the weights of the elements of the range found is $O(1)$. In fact we have now shown that the cost of searches to the left are also proportional to the log of the

Algorithm 13

Constants: $m = nw/(w-1) - n$; $m^+ = m + n$.
Key Variables: An array $H[1, \ldots, m^+]$ of integers.
Notation:
$\text{up}(a) = a * w^{\lceil \log_w a \rceil} / w^{\lfloor \log_w a \rfloor}$
$\text{down}(a) = a * w^{\lfloor \log_w a \rfloor} / w^{\lceil \log_w a \rceil}$
$f(c,d) = (n-1)/(w-1) + c - ((\text{down}(c*(w-1)) - 1)/(w-1)) * \text{up}(d)/w^{\lfloor \log_w c \rfloor}$
$\text{maxRep}(e) = \max\{b_s, \ldots, b^{f(e,n)-n})\}$.

Preprocessing:
Input: A list $B = \{b_0, b_1, \ldots, b_{n-1}\}$ of integers. n is a multiple of w.
1: **for** $i := 0$ To $n-1$ **do** $H[m+i] := b_i$.
2: **for** $i := m-1$ **downto** 1 **do**
$H[i] := \max\{H[wi], H[wi+1], \ldots, H[(w+1)i-1]\}$.

Query:
Input: A start index s, $0 \le s < n$, and a test value $t \in \mathbb{T}$.
Output: The largest index k, $s-1 \le k < n$, such that $b_j \dot{<} t$ for $i \le j \le k$.
1: $i := s + m$. levelLast $:= m^+ - 1$.
2: /* search initial block */
while $i \bmod w \neq w-1$ **do** /* searching a block */
if $H[i] \dot{<} t$ **then** $i := i+1$
else return $i - 1 - m$.
3: **if** not$((H[i] \dot{<} t))$ **then** return $i - 1 - m$.
4: **while** $i <$ levelLast and $H[i] \dot{<} t$ **do**
/* maxRep(i) $\dot{<} t$ */
$i := \lfloor (i + w - 1)/w \rfloor$. /* go to parent of right sibling; */
levelLast $:= \lfloor$levelLast$/w \rfloor$.
5: **if** $H[i] \dot{<} t$ **then** return $n-1$. /* $b_j < t$, $s < j < n$ */
6: **while** $i < n$ **do** /* w-way searching */
$i := i * w$.
for $j := 2$ **to** w **do** /* searching a block */
if $H[i] \dot{<} t$ **then** $i := i+1$. /* maxRep(i) $\dot{<} t$ and not maxRep(i) $\dot{<} t$ */
7: return $i - n - 1$.

sum of the elements of the weights found. However, this result has not been included here since it established after this thesis was submitted for evaluation.

Our algorithm builds a weighted binary tree that has at most (approximately) three times the number of levels as is the number of levels it would have if it were perfectly balanced. It is an interesting problem to see if this difference can be reduced.

The current algorithm makes its decisions by looking at only three elements of the data at a time. If this were increased to say five then perhaps a more balanced tree would result.

Finally, consider that we don't necessarily want a balanced tree because we have to favor more the elements with higher weights. Taking the weights into account it is interesting to ask what is an optimal tree and how far is our algorithm from the optimal. We expect that an optimal tree can be constructed using dynamic programming in polynomial but not linear time.

Our third algorithm involves range queries on binary trees. We think that this algorithm may be used in some cases where the a centroid decomposition of a tree [14]; see also [9, 25] has been used. This algorithm has many applications; see, for example, [13]. The centroid decomposition of a tree is a data structure that, like the algorithm in Appendix A, allows the searching of a binary tree in logarithmic time in the size of the tree. However it is difficult to construct in linear time while our path partition of a tree is quite simple to construct.

7.6. Appendix B

In Appendix B we solved a problem relating to the area of polygons. This result has also been published in [6]. It appeared to us that this problem could not be generalized to higher dimensions. However, recently, a three dimensional version of this problem has been solved [29].

We are also interested in seeing this result applied to solve other problems. Given the simplicity and efficientcy of our algorithm we expect applications to be found.

7.7. Other Visibility Decompositions

In the Chapter two of this thesis we discussed many visibility decompositions that already existed in the literature. However we have not used any of these decompositions in this thesis. This does not in any way reflect negatively on these visibility decompositions as useful tools. In fact we have studied these decompositions and have a number of exciting partial results. We expect that in the future we will demonstrate the usefulness of several of these visibility decompositions.

Also of considerable interest to us is the discovery of new visibility decompositions. We have a number of ideas here as well but these efforts are still in a preliminary stage. However we do expect the number of useful visibility decompositions known and their applications will grow considerably.

7.8. Other applications

Although our efforts have focused as much upon the creation of visibility decompositions as on their application, we believe that many applications are to be found. In particular, visibility decompositions have the potential to be useful anywhere that visibility is an issue. Thus, since visibility problems constiture a large part of computational geometry, we can expect many such applications.

All our efforts have been in the area of computational geometry. However, we also expect to see applications of our techniques in other areas such as computer graphics, computer vision, robotics and image processing. We have established the usefulness of visibility decompositions to Computational Geometry. We expect that the usefulness of visibility decompositions to these other areas of computer science will also be shown.

APPENDIX A

Finding a Range in Unsorted Arrays and Trees

A.1. Introduction

In this appendix we develop three data structures that we need in Chapter 5 to solve the circular ray shooting problem. These data structures are not geometric in nature and may have applications in other areas so we present them separately. In particular, for many problems, we believe that the third data structure can replace the centroid decomposition of a tree; see [9], [14], [25]. The third data structure has two advantages over a centroid decomposition. First, it is easily constructed in linear time while the linear time algorithms for constructing the centroid decomposition of a tree are all complex. Second, for problems of the type described in Section A.4, a centroid decomposition of a tree cannot be searched towards the tree's root efficiently while our data structure allows this. None of these data structures require that the data be sorted.

Given a starting point in the data, these data structures allow us to determine how many elements we can step through, such that each element processed satisfies a certain property. The first two data structures are applied where the original data are stored in arrays. In these cases we find a maximal subarray of the original array, all of whose elements satisfy a certain property. The second data structure is a new weighted binary tree structure which works with unsorted data. It is constructed using a simple bottom up method that only requires two stacks and basic stack operations. It is used to construct the third data strucure. The third data structure is applied to a tree. In this case we find a maximal length path on the original tree such that

the data stored on each data entry on the path satisfies a certain property. For each data structure we refer to the process of finding the list of data as range finding and the input to this process as a range query or just query.

Let $B = \{b_0, b_1, \ldots, b_{n-1}\}$ be an ordered set of n values, $n > 1$, with repetition allowed. For now we assume that the elements of B are positive integers. We will need to test the values in B against a test value t; not necessarily of the same type as the elements of B. We denote this test operator by the symbol " $\dot{<}$ " and refer to it as the *dot operator.* We refer to an application of the dot operator as a *dot operation.* We will express the complexities of a query in our data structures in terms of the number of dot operations applied instead of the time used. The reason is that the time complexity of applying the $\dot{<}$ operator depends on its specific definition and use in an application. For now, for the sake of simplicity, the test value will be an integer and the dot operator will represent the operator "less than".

For our first data structure the cost of finding the list of elements that satisfies the given property is logarithmic in the length of the list found. For the second data structure the cost is never more than a small constant times the *log* of the sum of the weights of the elements in the list found. In the final data structure the cost is never more than a small constant times the *log* of the size of the original data. For the final two cases we achieve our complexity bounds by developing a new weighted binary tree structure. We also show that our data structures behave particularly well in the case where the distance between the point where the search is started to the point finally found is small.

When the elements of B are stored in an array, the problem we solve has the following form:

PROBLEM A.1.1. *Preprocess B so that, for any given start index s and test value t, one can find the largest subarray $B_{s,t}$ of B starting at $B[s]$ such that $B[i] \dot{<} t$ for every element $B[i]$ of $B_{s,t}$. If $B[s] \dot{<} t$ is false then $B_{s,t} = \emptyset$.*

The remainder of this appendix is organized as follows:

In Section A.2 we will study Problem A.1.1. The data structure we construct to solve this problem is simple in this case and can be implemented in an array at most 3 times the size of B. A variation of the data structure is at most twice the size of B. It can be constructed in linear time and with it we can answer a query, as specified in Problem A.1.1, in at most $2\lceil \log |B_{s,t}| \rceil + 1$ dot operations.

In Section A.3 we will study the version of Problem A.1.1 in which the elements of B have weights associated with them. Let $k = \min\{n-1, s+|B_{s,t}|\}$. We will show that we can solve this problem using a number of dot operations that, in the worst case, is proportional to the log of the sum of the weights of the elements from b_s to b_k. This is generally slower than the solution in Section A.2. However, it provides improved performance when the weights associated with b_s and b_k are high, relative to other values in the list. This characteristic of the query time complexity is key to achieving the query time complexity we achieve for trees in Section A.4.

In Section A.4 we will deal with the case that the elements of B are stored at the vertices of a tree T. In this section, to simplify matters, we assume the elements of B are real intervals, t is a real number, and "$\dot{<}$" is the operator such that $I \dot{<} t$ returns true if interval I contains t. We show that, for any vertex v of T and real number t, we can find the largest path, say P, of T, containing v, such that every interval on the vertices of P contains t. This query costs $O(\log |T'|)$ dot operations where T' is the largest subtree of T rooted at a vertex of P.

We point out that in Chapter 5 we transformed the circular ray shooting problem to the problems solved in Section A.4. For this case, the data elements of B are more

complex objects, composed of Voronoi diagrams. The test value t in this case is from the set of circular arcs. Except for these changes, the problem is mapped directly to the problems solved in Section A.4 and is solved using the data structures and algorithms developed there.

Our algorithm for this problem has the same complexity as the algorithm in [16][1] in the worst case but our algorithm is simpler and has smaller constant coefficients. Much of this improvement in efficiency and simplicity is a result of using the data structure from Section A.4. For more details see Chapter 5.

Binary trees are a key element of our data structures and we will need some basic terminology on them. Henceforth, unless otherwise stated, all trees used in this appendix are assumed to be binary. If v_i is a node in a tree, say T, then we denote the left (right) child of v_i by lChild(v_i) (rChild(v_i)). Also, lChild(v_i) is visited before (rChild(v_i) during an in-order traversal of T. We say that v_i is to the left of another node v_j of T if v_i is visited before v_j during an in-order traversal of T. The parent node of v_i is denoted parent(v_i). The absence of a child, parent, or any other node related to v_i is indicated by the value "nil". Thus parent(v_i) = nil if v_i has no parent. If v_i has child nodes it is called an internal node; otherwise it is called a leaf node. We denote by val(v_i) the value stored at node v_i. We denote the subtree rooted at v_i by $\Lambda(v_i)$, the set of leaf nodes of $\Lambda(v_i)$ by $\pounds(v_i)$, and the leftmost element of $\pounds(v_i)$ by $\ell(v_i)$. We use $|T|$ to denote the number of vertices of T.

A key idea that we need is that of a sibling. Let v_i be any node of T other than its root or the rightmost descendants of its root and v_j be the leftmost leaf node of T of those to the right of every element of $\pounds(v_i)$. Let v_p be parent(v_i) if parent(v_i) is an ancestor of v_j; otherwise let v_p be the right child of the lowest common ancestor

[1]Actually the preprocessing time for our algorithm is faster by a $O(\log n)$ factor but this improvement is due to an algorithm unrelated to this appendix.

of v_i and v_j. Then we call the path from v_j to v_p the *right wall* of v_i. We will exploit the following property of walls:

OBSERVATION A.1.2. *Let v_k be any node on the right wall of v_i. Then $v_k \in \{v_j, \ell(v_i)\}$.*

We will need a link from v_i to an element on its right wall; we call the element linked to the *right neighbor* of v_i and denote it by $\mathrm{rSib}(v_i)$. The exact value of $\mathrm{rSib}(v_i)$ will be specified later and will vary, depening on the algorithm in which it is used.

A.2. Range Finding in Arrays

In this section we solve Problem A.1.1 in its standard form; that is B is a simple list of positive integers, t is a positive integer, and $\dot{<}$ is the binary operator "less than".

Let $k = \min\{n-1, s + |B_{s,t}|\}$. We denote $B^+_{s,t} = B[s..k]$ and $b_{s,t} = b_k$. We will show that, for Problem A.1.1, after linear preprocessing, $B_{s,t}$ can be found in at most $2\lceil \log |B^+_{s,t}| \rceil + 1$ dot operations. We assume for the present that n is a power of two. After presenting our result for this case we will show how to remove this restriction.

We will solve this problem by implicitly embedding a tree like structure into an array. However, it is useful to first consider that we are using the implicit structure directly. This structure is a complete binary tree augmented with additional links. We refer to this structure as an *augmented tree.* The additional links are bidirectional and connect elements at the same level of the tree such that a bidirectional path is formed from the leftmost to rightmost element. If v is a node in one of these bidirectional paths then we set $\mathrm{rSib}\,(v)$ to its right neighbor on this path. We label the vertices of an augmented tree with m leaves (for now m is a power of two) as $\{v_1, v_2, \ldots, v_{2m-1}\}$ according to the order in which they are visited during a breadth first search of the

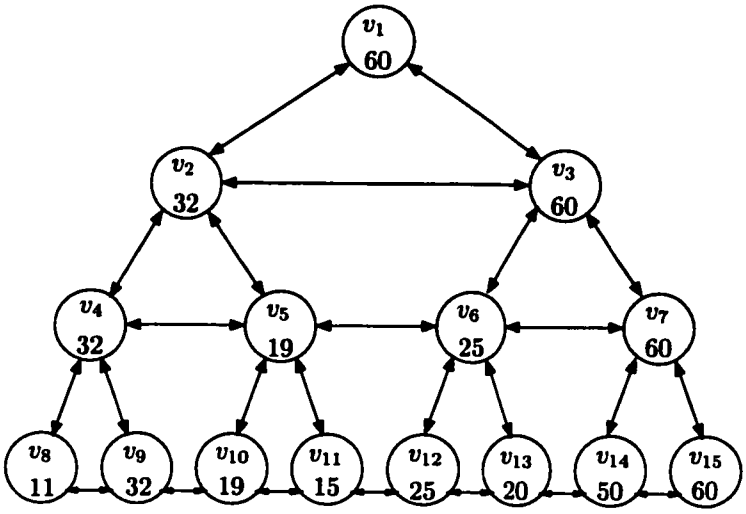


FIGURE A.2.1. Labelling of an augmented complete binary tree

tree starting at the root and visiting the children of a node in left to right order; see Figure A.2.1.

We denote by B^T the augmented tree whose leaves contain in left to right order the elements b_i, $0 \leq i < n$. Note that $|B^T| = 2|B| - 1$. Let v_i be a node of B^T. Note that if $i \geq n$ then $\text{val}(v_i) = b_{i-n}$. If v_i is an internal node then, for any test value t, we need to be able to test in one dot operation whether $v_j \dot{<} t$ for all $v_j \in \mathcal{L}(v_i)$. We achieve this by setting $\text{val}(v_i) = \max\{\text{val}(v_j) : v_j \in \mathcal{L}(v_i)\}$. Later on, when we change the operator $\dot{<}$ we will also change function max to a suitable function.

Assume v_i has a right sibling and v_h is the parent of the right sibling of v_i. Then the following observations express the properties of B^T that are key to the algorithms we develop in this section.

OBSERVATION A.2.1. $h = \lfloor (i+1)/2 \rfloor$.

OBSERVATION A.2.2. $|\mathcal{L}(v_h)| = 2|\mathcal{L}(v_i)|$

For example, in Figure A.2.1, $\mathcal{L}(v_2) = \{v_8, \ldots, v_{11}\}$ and $\mathcal{L}(\text{parent}(\text{rSib}(v_2))) = \mathcal{L}(v_1) = \{v_8, \ldots, v_{15}\}$. Also, $\mathcal{L}(v_5) = \{v_{10}, v_{11}\}$ and $\mathcal{L}(\text{parent}(\text{rSib}(v_5))) = \mathcal{L}(v_3) = \{v_{12}, \ldots, v_{15}\}$. Note also that $\ell(\text{parent}(\text{rSib}(v_2))) = v_8$ and $\ell(\text{parent}(\text{rSib}(v_5))) = v_{12}$.

We solve Problem A.1.1 by first finding the element following $B_{s,t}$ in B or, if there is no such element, by detecting that the last element of $B_{s,t}$ is b_{n-1}; that is we find element $b_{s,t}$. In terms of B^T this means finding node v_k where v_k is the node of B^T containing $b_{s,t}$. We denote this node by $v_{s,t}$. Observe that we can test if $v_{s,t} \in \mathcal{L}(v_i)$ with one dot operation, assuming we know that $v_{s,t}$ does not precede $\ell(v_i)$. If this dot operation returns true then Observation A.1.2 tells us that by searching in $\Lambda(v_h)$ we will not search among any elements that precede v_i and will not fail to search the elements that are between the element following the rightmost elements of $\mathcal{L}(v_i)$ and $\ell(v_h)$ (since there are none). Thus, by starting with v_{n+s} and applying this switch repeatedly, along with a dot operation for each subtree, we are able to find a subtree containing v_k.

Observation A.2.2 tells us that with each switch the size of the subtree under consideration doubles so at most $\log n - 1$ switches are needed to find a subtree containing $v_{s,t}$. Once this subtree is found we can find $v_{s,t}$ by recursively selecting the child node of the subtree that contains v_k until the leaf node containing v_k is found. Based upon these ideas we present a solution to Problem A.1.1 in Algorithm 14. In Figure A.2.2 we show two examples of solving a query for Problem A.1.1 in an augmented tree using Algorithm 14. Note that the tree in this figure is not complete since n is not a power of two. We show how to deal with this case in Subsection A.2.1.

THEOREM A.2.3. *Algorithm 14 finds $B_{s,t}$ using at most $2\lceil \log |B^+_{s,t}| \rceil + 1$ dot operations.*

PROOF. Assume that the loop at Step 3 iterates j times. The case that $j = 0$ is straightforward so we assume that $j > 0$. After $j - 1$ iterations of the loop, val(w) is the maximum of a contiguous list of 2^{j-1} elements of B, all of which are smaller than t and furthermore, any elements of B including or following b_s but preceding this list

Algorithm 14

Input: B^T, leaf vertex v_s of T and test value t.
Output: $B_{s,t}$.

1: **if** (not $(\text{val}(v_s) \dot{<} t)$) **then** return $\emptyset$. /* return $B_{s,t}$ */
2: $w := v_s$.
3: **do** /* $\max\{\mathcal{L}(w)\} \dot{<} t$ */
  if $\text{rSib}(w) = \text{nil}$ **then** return $B[s..n-1]$. /* return $B_{s,t}$ */
  $w := \text{parent}(\text{rSib}(w))$.
  while $(\text{val}(w) \dot{<} t)$.
4: **do**
  $w := \text{lChild}(w)$.
  if $\text{val}(w) \dot{<} t$ **then** $w := \text{rSib}(w)$.
  while $(\text{val}(w) \dot{<} t)$.
5: return $B[s..\text{index}(w)-1]$ /* return $B_{s,t}$ */

are also smaller than t. This follows from the application of Observations A.1.2 and A.2.2 once for each iteration. Therefore $j \leq \lceil \log |B^+_{s,t}| \rceil$. Now, at the start of Step 4, $|\mathcal{L}(w)| = 2^j$ so the loop at Step 4 iterates j times. Now, counting the dot operations, we find that at most $2\lceil \log |B^+_{s,t}| \rceil + 1$ dot operations are applied in total. □

Let v be a node of B^T. Now Observation A.2.1 and similar observations that can be made about the indices of neighbor nodes of augmented trees allow us to determine the indices of the neighbors of v by simple arithmetic. Thus, instead of constructing an actual tree to represent B^T, we can be more efficient by embedding B^T into an array and computing the indices of the neighbors of any node of B^T from the index of the node instead of storing pointers. Furthermore, the arithmetic is quite simple.

One difficulty we have though, is determining if a node v of B^T has a right wall. Let $\text{levelLast}(v)$ denote the rightmost node of B^T on the same level as v. We solve this problem by keeping track of $\text{levelLast}(w)$. We note that w has a right wall if and only if $w \neq \text{lastLevel}(w)$.

Now we consider how to embed B^T into an array. Let H be an array of $2n-1$ elements where the index of the first element is one. We store b_i, $0 \leq i < n$, in $H[n+i]$. Next, we assign values to the remaining elements of H such that every

index i, $1 \leq i < n$, satisfies the constraint $H[i] = \max\{H[2i], H[2i+1]\}$. Observe that H is a heap, as in heap-sort, except that instead of exchanging values to build the heap in place, we have allocated $n-1$ additional entries in which to place the results of the max operations. This is necessary since we need the original data items to maintain their position within H.

For $1 \leq i < 2n$, the locations of the neighbors of $H[i]$ in this structure are computed as follows:

(1) For $2 \leq i < 2n$, $H[\lfloor i/2 \rfloor]$ contains the parent of $H[i]$.

(2) For $1 \leq i \leq n$, $H[2i]$ contains the left child of $H[i]$.

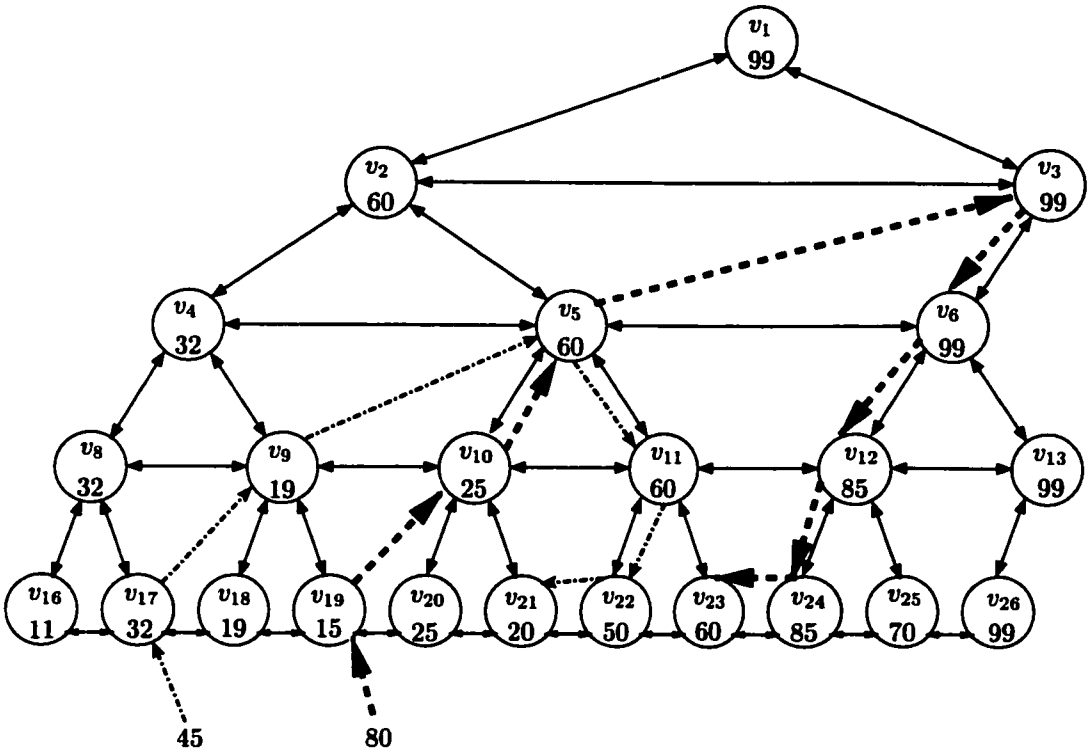
(3) For $1 \leq i < n$, $H[2i+1]$ contains the right child of $H[i]$.

(4) For $3 \leq i < 2n$, if $H[i]$ is not a power of two then $H[i-1]$ contains the left sibling of $H[i]$.

(5) For $2 \leq i < 2n$, if $i+1$ is not a power of two then $H[i+1]$ contains the right sibling of $H[i]$.

Now, Algorithm 15 is a version of Algorithm 14 in which B^T is stored in an array and no explicit pointers are used. See Figure A.2.2.

A.2.1. Relaxing the Constraint on the Size of B. We now show how to remove the constraint that n be a power of two. A simple solution is to pad B with a enough trailing zeros so that $|B|$ is a power of 2. Now, the trailing zeros mark unused elements as do the additional zeros generated during the preprocessing stage of Algorithm 15. The modifications to the algorithms needed to test for these unused elements are not difficult and are omitted. The difficulty with these modifications is that much unused space may result. We can save some space by simply removing the trailing zeros from H and modifying the code accordingly. One of these modifications is to zero all elements of H initially, so that any entries that are not overwritten have the appropriate "unused" markers. Note that the loop at Step 2 of the preprocessing



Finding $B_{1,45}$ and $B_{3,80}$ where B is a list (of 11 numbers).
Note: We treat the case where $|B|$ is not a power of two in Subsection A.2.1

FIGURE A.2.2. Range Finding

stage of Algorithm 15 still requires that n be a multiple of 2; this restriction is easily removed by doing the first iteration of the loop separately. While the savings with these modifications can be significant, there can still be as many as $n - 2$ unused entries in H. Asymptotically, this can be as much as a third of the entries in H.

To see how to eliminate these unused entries, consider the corresponding complete binary tree. Note that on Figure A.2.2 the missing elements from the corresponding complete binary tree are all recursively from the right side of the tree. The missing part of the tree corresponds to the elements of H storing value zero.

We can also organize the data in H and its corresponding augmented tree slightly differently; where the missing elements from the corresponding complete binary tree

Algorithm 15

Key Variables: An array $H[1..2n-1]$ of non negative integers.
Notation: $\text{up}(a) = a * 2^{\lceil \log a \rceil}/2^{\lfloor \log a \rfloor}$
$f(c,d) = ((c+1) * \text{up}(d))/2^{\lfloor \log c \rfloor} - 1.$
$\text{maxRep}(e) = \max\{b_s, \ldots, b^{f(e,n)-n})\}.$

Preprocessing:
Input: A list $B = \{b_0, b_1, \ldots, b_{n-1}\}$ of positive integers where n is even.
1: **for** $i := 0$ **to** $n-1$ **do** $H[n+i] := b_i$.
2: **for** $i := n-1$ **downto** 1 **do** $H[i] := \max\{H[2i], H[2i+1]\}$.

Query:
Input: A start index s, $0 \leq s < n$, and a positive integer test value t.
Output: $B_{s,t}$.
1: $i := s + n$.
2: levelLast := $2n-1$. /* The index of the last node of the current level */
3: **if not** $(H[i] \dot{<} t)$ **then return** $\emptyset$. /* return $B_{s,t}$ */
4: /* find an ancestor of $b_{s,t}$ */
do /* $\text{maxRep}(i-1) \dot{<} t$ */
if $i =$ levelLast **then return** $B[s..n-1]$. /* return $B_{s,t}$*/
$i := \lfloor (i+1)/2 \rfloor$. /* go to the parent of the right sibling */
levelLast := $\lfloor$levelLast$/2 \rfloor$. /* index of last node of previous level */
while $H[i] \dot{<} t$.
5: /* find leaf node containing $b_{s,t}$ */
do /* $\text{maxRep}(i-1) \dot{<} t$ and not $(\text{maxRep}(i) \dot{<} t)$ */
$i := i * 2$.
if $H[i] \dot{<} t$ **then** $i := i+1$. /* it must be that (not $(H[i] \dot{<} t)$) */
while $i < n$.
6: return $B[s..i-n-1]$. /* return $B_{s,t}$ */

are all from the bottom level of the tree. With this organization of the data we can eliminate the need for any unused elements. Consider for example that $n = 2^k + 1$ where k is a positive integer. Let $\widehat{b_0} = \max\{b_0, b_1\}$ and $\widehat{B} = \{\widehat{b_0}, b_2, b_3, \ldots, b_{n-1}\}$. Now we can apply Algorithm 15 to $\widehat{B}$ since $|\widehat{B}|$ is a power of two. Furthermore, by extending the size of H by two elements and placing b_0 and b_1 in these entries and modifying Algorithm 15 accordingly, Problem A.1.1 can be solved or B where $|B| = n = 2^k + 1$. Clearly now, this mofification can be generalized to deal with any positive value of n. With these changes, all entries of H are used and $|H| = 2n$. The

disadvantage with making this change is that it now takes $O(n)$ time to update H if a single element is added (or removed) at the end of B, whereas without the change this can be done in $O(\log n)$ time unless n is a power of two in which case linear time is still required. Note that these claims are based on the assumption that the size of H can be changed by two elements in $O(1)$ time.

A.3. Searching Weighted Arrays

We now consider the case that, for each element $b_i \in B$, there is an additional value w_i which we call the *weight* of b_i. We wish to be able to solve Problem A.1.1 such that the algorithm is more efficient if the weights of b_s and $b_{s,t}$ are high relative to the weights of the other elements of B. At this point we allow the values of B to be arbitrary integers.

We denote $\overline{w}(i,j) = \sum_{h=i}^{\min\{j,n-1\}} w_h$. For any subarray $K = B[k..l]$ of B we denote $w(K) = \overline{w}(k,l)$. We show that, after linear preprocessing, $B_{s,t}$ can be found in $3\left\lfloor \log w(B_{s,t}) - \log w(b_s) + \log w(B^+_{s,t}) - \log w(b_{s,t}) \right\rfloor + 11$ dot operations where s and t are as defined in Problem A.1.1. This version of Problem A.1.1 occurs in Section A.4 where we deal with searching data stored in trees. The negative terms in the dot operation complexity expression are quite important; they allow us to reduce the complexity of a search in the tree case from $O(\log^2 n)$ dot operations to $O(\log n)$ dot operations. This reduction is achieved, in part, using standard properties of weighted trees.

First, we must build a weighted tree. Weighted trees normally require that the data be in sorted order; see the section on "Nearly Optimal Binary Search Trees" as described in [**39**, pg 177-187]. However, we need a weighted tree that does not require sorted data so we create our own weighted tree data structure. We could modify the weighted tree construction algorithm in [**39**] and it may be that this works well for

the problem solved in this section. However, as we shall see, there are difficulties using it with our weighted tree structure in Section A.4.

In this section, we denote the weight associated with any node v_i of a tree by $w(v_i)$. Furthermore, if v_i is an internal node then $w(v_i)$ will denote the sum of the weights of its leaf nodes.

Algorithm 16

Key Variables: stacks S_1 and S_2. The elements of these stacks are nodes with the necessary pointers to be nodes of trees. Each node v has associated with it a weight $w(v)$.
Method: merge(l, r). Create a node p with left child l and right child r such that $w(p) = w(l) + w(r)$ and val$(p) = \max\{\text{val}(l), \text{val}(r)\}$. Return p.
Method: second(S). Returns the element of stack S second from the top.
Input: A list $B = \{b_0, b_1, \ldots, b_{n-1}\}$ of integers with associated weights w_i, $0 \leq i < n$.
Output: A weighted tree whose leaves contain the elements of B in order.

1: Put a sentinel node in each of S_1 and S_2 with weight $\overline{w}(0, n-1)+1$.
2: Put the elements of B into nodes and place the nodes in S_2 in reverse order (The node containing b_0 is the top element of S_2.). Also assign the weight of each node in S_2 to be the weight of the element of B that the node contains.
3: **while** $|S_1| + |S_2| > 3$ **do** /*or while fewer than $n-1$ merges done */
  while $w(\text{top}(S_1)) > w(\text{top}(S_2))$ **do**
    push$(S_1, \text{pop}(S_2))$.
  if $w(\text{second}(S_1)) < w(\text{top}(S_2))$ **then**
    tmp := pop(S_1). /* preparation for a ST merge */
  else
    tmp := pop(S_2). /* preparation for a TT merge */
  push$(S_2, \text{merge}(\text{pop}(S_1), \text{tmp}))$. /* An ST or TT merge */
4: return top(S_2). /* return the root node of the weighted tree */

We now present Algorithm 16 which builds a weighted tree whose leaves contain the elements of B such that the distance from any internal node v_i of the tree to any node v_l in $\Lambda(v_i)$ is at most $3\lfloor \log w(v_i) - \log(v_l) \rfloor + 2$. An example of the output of this algorithm is given in Figure A.3.1. The main step of Algorithm 16 is to merge two subtrees to form a new subtree such that the child nodes of the root

of the new subtree are the initial two subtrees. This merge can be done in two ways. The first way is to merge the top element from each of the two stacks; we call this a *TT merge.* The second way is to merge the top two elements of S_1; we call this an *ST merge.* These merges are applied until only a single tree remains; this tree being a weighted tree. Henceforth, the term "weighted tree" refers to a tree as constructed by Algorithm 16. Algorithm 16 also sets val(v) for each internal node v of the weighted tree such that val(v) is an element of B if v is a leaf node, otherwise $\text{val}(v) = \max\{\text{val}(v_i) : v_i \in \mathcal{L}(v)\}$.

OBSERVATION A.3.1. *The internal nodes of a weighted tree all have two child nodes.*

OBSERVATION A.3.2. *Throughout Algorithm 16 the weights of the elements of S_1 form a strictly monotonically increasing sequence starting from the top element of S_1.*

LEMMA A.3.3. *Algorithm 16 returns a tree whose leaves contain the elements of B in order according to an in-order traversal of the tree.*

PROOF. Note first that with each iteration of the outer loop the total size of the stacks decreases by one so an infinite loop is impossible. Furthermore, the result must be a tree since the input nodes to a merge operation cannot be accessed again.

Initially, other than the sentinels, the nodes in the stacks are ordered according to the ordering of the elements of B. This ordering remains invariant in the sense that, if after any iteration of either loop the subtrees are replaced by an in-order listing of their leaves and the values from S_1 precede those from S_2 then we get the initial ordering again; in particular the merge operations always merge adjacent elements such that this ordering is preserved. Thus, the leaf nodes of the tree returned are in the correct order.

The last merge performed by the algorithm results in only 3 nodes in the stacks and the sentinels are at the bottom of the stacks so the last merge constructs a tree

B	11	32	19	15	25	20	50	60
weights	12	17	4	11	5	3	4	9

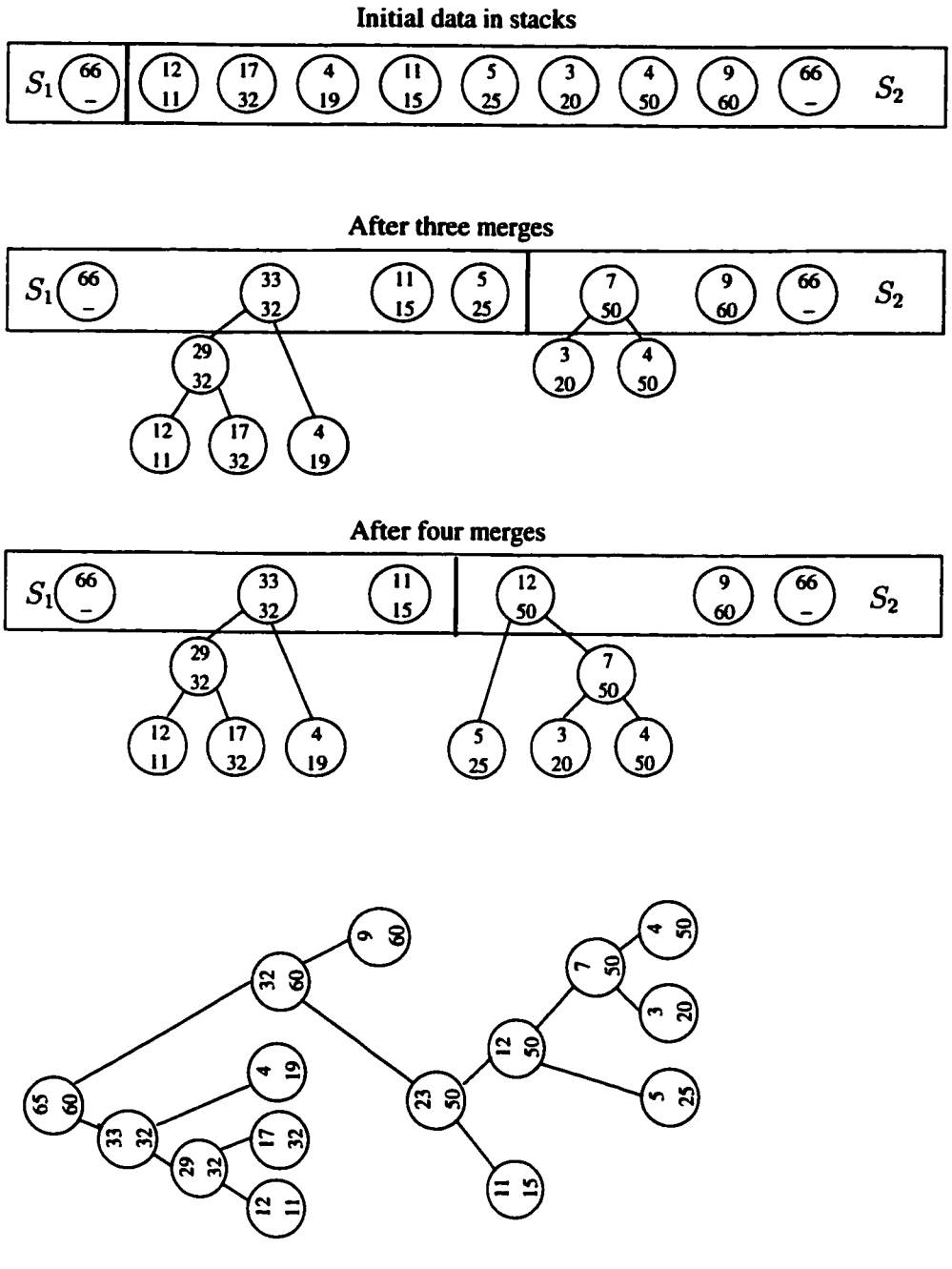


FIGURE A.3.1. A weighted tree as constructed by Algorithm 16

whose leaves contain all the elements of B in the required ordering. Furthermore, this is the tree returned by the algorithm. □

THEOREM A.3.4. *Algorithm 16 runs in $O(|B|)$ time.*

PROOF. The stacks contain $n+2$ nodes initially and 3 nodes at the end so $n-1$ merges occur. Furthermore, the nodes returned from merges are pushed onto S_2 so the number of elements pushed onto S_2 is $n+1+(n-1)=2n$. The number of elements pushed onto S_1 is at most one more than the number of elements pushed onto S_2 so the total number of push operations is at most $4n+1$. There are 3 fewer pop operations than push operations so in total there are at most $8n-1$ push/pop operations. Now, at least one push or pop operation occurs for each iteration of each loop so there are $O(n)$ iterations of loops in total. Finally, each iteration of any loop costs $O(1)$ time so the algorithm runs in linear time. □

LEMMA A.3.5. *Let T be any subtree of a weighted tree such that $|\pounds(T)|=4$ and there exists a path s_1, s_2, s_3, s_4 of T where s_1 is a leaf node of T and s_4 is the root of T. Then, $w(s_4) > 2w(s_1)$.*

PROOF. Let c_2, c_3, c_4 be the leaf nodes of T such that c_i is the child of s_i, $2 \leq i \leq 4$. Assume that $w(c_2) < w(s_1)$ and $w(c_3) < w(s_1)$. We will show that $w(c_4) \geq w(s_1)$ from which this lemma follows. First, $s_1 \neq \text{top}(S_1)$ when s_1 and c_2 were merged since $w(c_2) < w(s_1)$. Let d_1 be the element on the top of S_1 after s_2 was placed on S_2. There are two ways that c_2 and s_1 could have been merged:

Case 1: When s_1 was on top of S_2 (a TT merge). Then $w(d_1) > w(s_1)$ since otherwise d_1 and c_2 (which was on top of S_1) would have been merged instead.

Case 2: When s_1 was the second element of S_1 (an ST merge). Then $w(d_1) > w(s_1)$ since d_1 was below s_1 on S_1 and by Observation A.3.2.

Now $s_2 = \text{lChild}(s_3)$ since otherwise either $d_1 = c_3$ or d_1 is a descendant of c_3 and then, either way, $w(c_3) \geq w(d_1) > w(s_1)$; a contradiction. Therefore, since $w(c_3) < w(s_2)$, we have that s_2 was the second element of S_1 and c_3 was the top element of S_1 when s_2 and c_3 were (ST) merged.

Let d_2 be the element on the top of S_1 after s_3 was placed on S_2 and d_3 be the node on top of S_2 before s_3 was put there. Then $w(d_3) \geq w(s_2)$ since otherwise d_3 and c_3 would have been TT merged. Also, $w(d_2) > w(s_2)$ since d_2 was below s_2 when s_2 was on S_1 and by Observation A.3.2. But now, either $c_4 = d_2$ or c_4 is an ancestor of d_2 or $c_4 = d_3$ or c_4 is an ancestor of d_3. Therefore $w(c_4) \geq \min\{w(d_2), w(d_3)\} > w(s_1)$. Now, $w(s_4) = w(c_4) + w(s_3) > 2w(s_1)$. □

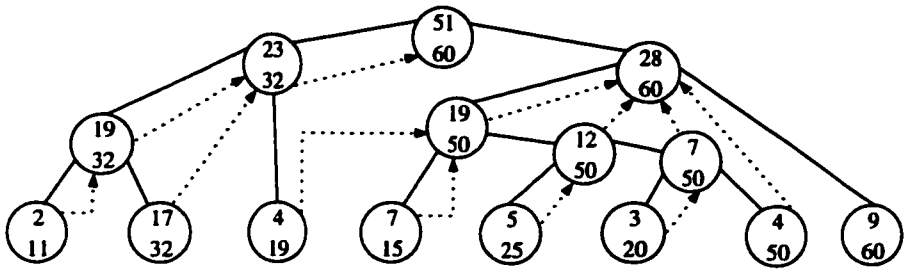
We denote the length of a path p by $\text{len}(p)$.

THEOREM A.3.6. *Let T be a weighted tree, v be any node of T, u be a node in $\Lambda(v)$, and p be the path from u to v. Then $\text{len}(p) \leq 3\lfloor \log w(v) - \log w(u) \rfloor + 2$.*

PROOF. We assume $\text{len}(p) > 2$. Let s_1, s_2, s_3, s_4 be any 4 consecutive nodes on p. It is sufficient to show that $w(s_4) > 2w(s_1)$. But this follows from Lemma A.3.5. □

For any leaf node v of a weighted tree whose leaves contain the elements of B and where $\text{val}(v) = b_i$ we denote $\text{index}(v) = i$ and $\text{leafNode}(b_i) = v$.

Consider a weighted tree T whose leaves contain the elements of B. We will perform the query part of Problem A.1.1 on T in two phases. In the up phase we find a subtree T' of T such that $\text{leafNode}(b_{s,t}) \in \mathcal{L}(T)$. In the down phase we find $\text{leafNode}(b_{s,t})$ itself. We will achieve the desired efficiency for the up phase using right siblings. We need to ensure that the weights of the nodes on any path, constructed by following right sibling links, grow at an approximately geometric rate. If the weights grow too slowly then too many dot operations may used in the up phase; if they grow too quickly then too many dot operations may be used in the down phase. For any node v_i in T with a right wall we set $\text{rSib}(v_i)$ to the lowest node on the right wall



$B = \{11, 32, 19, 15, 25, 20, 50, 60\}$. *Right neighbors are indicated by dotted lines.*

FIGURE A.3.2. An augmented weighted tree

of v_i such that $w(\text{rSib}(v_i)) > 2w(v_i) \geq w(\text{lChild}(\text{rSib}(v_i))$ if this node exists and to the highest node on the wall otherwise. We call a weighted tree with these additional links an *augmented tree* and denote the augmented tree cooresponding to T by T_A. Given T, Algorithm 17 constructs T_A.

Algorithm 17

Input: A weighted tree T without sibling links.

Output: An augmented tree T_A with right sibling links such that right siblings have approximately the same weights where possible.

```
1: for each internal node u_i of T do
       ls := lChild(u_i).
       rSib(ls) := u_i.
       ls := rChild(ls).
       rs := rChild(u_i).
       while ls ≠ nil do
          while rs is not a leaf node and w(rs) ≤ 2w(ls) do
             rs := lChild(rs).
          rSib(ls) := parent(rs).
          ls := rChild(ls).
2: return the modified tree.
```

It is easily shown that Algorithm 17 can be implemented to run in linear time. A tree returned by Algorithm 17 has the following useful properties. Let T_A be the augmented tree returned by Algorithm 17.

OBSERVATION A.3.7. *Let u_i be an internal node of T_A. Then rSib(u_i) $\neq$ nil if and only if rChild(v_i) $\neq$ nil.*

LEMMA A.3.8. *Let $p = v_1, v_2, \ldots, v_m$ be any path on T_A such that $rSib(v_i) = v_{i+1}$, $1 \leq i < m$. Then $len(p) \leq 3\lfloor \log w(v_m) - \log w(v_1) \rfloor + 2$.*

PROOF. Let T be the input to Algorithm 17 for which it returned T_A. Assume that $\text{len}(p) > 2$. To prove this lemma it is only necessary to show that, for any subpath $q = v_i, v_{i+1}, v_{i+2}$ of p of length 3, we have that $w(v_{i+2}) > 2w(v_i)$. Now if there is a path of length at least 3 on T whose endpoints are vertices of q then $w(v_{i+2}) > 2w(v_i)$ by Lemma A.3.6. Otherwise, for some vertex, say v_j, of q, we have that $\text{rSib}(v_j)$ is not an ancestor of v_j on T. But then $w(v_{j+1}) > 2(v_j)$ by the construction of right siblings in Algorithm 17. Also, for the same reason, the weights of the nodes of q form a strictly increasing sequence so $w(v_{i+2}) > 2w(v_i)$. □

Now, using the augmented weighted tree returned by Algorithm 17, Algorithm 18 solves the query part of Problem A.1.1.

Algorithm 18

Preprocessing:

Input: A weighted list $B = b_0, b_1, \ldots, b_{n-1}$ of positive integers.

1: Build a weighted tree with input B using Algorithm 16 and then add right sibling links using Algorithm 17.

Query:

Input: Start index s, $0 \leq s < n$, and test integer value t.

Output: $B_{s,t}$.

1: **if** not $(b_s \dot{<} t)$ **then** return $\emptyset$. /* return $B_{s,t}$ */
2: $\kappa :=$ leafNode(b_s).
3: /* up phase: find subtree containing $b_{s,t}$ */
do
if $\text{rSib}(\kappa) = \text{nil}$ **then** return $B[s..n-1]$. /* return $B_{s,t}$ */
$\kappa := \text{rSib}(\kappa)$.
while $\text{val}(\kappa) \dot{<} t$.
4: /* down phase: find leaf node containing $b_{s,t}$ */
do
if not $(\text{val}(\text{lChild}(\kappa)) \dot{<} t)$ **then** $\kappa := \text{lChild}(\kappa)$.
else $\kappa := \text{rChild}(\kappa)$. /* it must be that (not $(\text{val}(\text{rChild}(\kappa)) \dot{<} t)$) */
while κ is not a leaf node.
5: return $B[s..\text{index}(\kappa) - 1]$. /* return $B_{s,t}$ */

LEMMA A.3.9. *If the up phase of the query part of Algorithm 18 is reached then it invokes at most* $3\lfloor \log w(B_{s,t}) - \log w(b_s) \rfloor + 3$ *dot operations where* s *and* t *are as defined in Problem A.1.1.*

PROOF. Assume that the loop at Step 3 iterates j times. This theorem follows immediately if $j = 0$ so assume $j > 0$. Let κ_j, $0 \leq \kappa \leq j$ be the value of κ after j iterations of the loop at Step 3. Let $v_{s,t}$ be the node of T containing $b_{s,t}$. Now, κ_{j-1} is the root of a subtree containing neither $v_{s,t}$ nor v_{s-1} (assuming $s > 0$) so this subtree has at most $|B_{s,t}|$ leaves. Thus, by Lemma A.3.8, $j - 1 \leq 3\lfloor \log w(B_{s,t}) - \log w(b_s) \rfloor + 2$. □

LEMMA A.3.10. *The down phase of the query part of Algorithm 18 invokes at most* $3\lfloor \log w(B^+_{s,t}) - \log b_{s,t} \rfloor + 7$ *dot operations where* s, t *are as defined in Problem A.1.1.*

PROOF. Clearly if the down phase begins at node $b^+_{s,t}$ then this Lemma follows so we assume that this is not the case. Let h be the number of iterations of the loop at Step 4 needed to find $B^+_{s,t}$. We assume that $h > 3$. Let κ_j, $0 \leq j \leq h$, be the value of κ after iteration j of the loop at Step 4. Observe that if $w(\kappa_2) < 2w(B^+_{s,t})$ then, by Theorem A.3.6,

$$h \leq 2 + 3\lfloor \log(2w(B^+_{s,t})) - \log(w(b_{s,t})) \rfloor + 2 \leq 3\lfloor \log w(B^+_{s,t}) - \log w(b_{s,t}) \rfloor + 7.$$

It remains to be shown that $w(\kappa_2) < 2w(B^+_{s,t})$. Let λ be the second last node reached during the up phase; that is $\mathrm{rSib}(\lambda) = \kappa_0$. Then $w(\lambda) \leq w(B_{s,t})$.

Let $v_{s,t}$ be the element of T containing $b_{s,t}$, $l = \mathrm{lChild}(\kappa_0)$, and $r = \mathrm{rChild}(\kappa_0)$. Then $\kappa_1 \in \{l, r\}$.

If $\kappa_1 = l$ then $v_k \notin \mathcal{L}(\text{parent}(\lambda))$. Therefore, $w(l) < 2w(\lambda)$ since otherwise $\text{rSib}(\lambda) \neq \kappa_0$ by the rules for setting right siblings. Now

$$w(\kappa_2) < w(\kappa_1) = w(l) < 2w(\lambda) \leq 2w(B_{s,t}) < 2w(B^+_{s,t}).$$

So assume $\kappa_1 = r$. Then either $\lambda = l$ or $w(l) < 2w(\lambda)$ since otherwise $\text{rSib}(\lambda) \neq \kappa_0$. Either way, $w(l) < 2w(\lambda)$. Also, $w(l) \leq w(B_{s,t})$ since $v_{s,t} \notin \mathcal{L}(l)$. Let $r_l = \text{lChild}(r)$ and $r_r = \text{rChild}(r)$. Now, r_l was on stack S_1 when r_l and r_r were merged. Let u be the element immediately below r_l on S_1 when this merge occurred. Note that either $u = l$ or u is a rightmost descendant of l so $w(u) \leq w(l)$. Now $w(r_l) < w(u)$ by Observation A.3.2. If r_l and r_r were TT merged then $w(r_r) \leq w(u)$ since otherwise r_l and u would have been ST merged. If, instead, r_l and r_r were ST merged then $w(r_r) < w(u)$. Either way $w(r_r) \leq w(u)$. Now $\kappa_2 \in \{r_l, r_r\}$ so $w(\kappa_2) \leq w(u) \leq w(l) \leq w(B_{s,t}) < 2w(B^+_{s,t})$. □

We summarize with the following result.

THEOREM A.3.11. *Problem A.1.1 can be solved with an algorithm that has a preprocessing stage requiring linear time and space and a query part that requires at most* $3\lfloor \log w(B_{s,t}) - \log w(b_s) + \log w(B^+_{s,t}) - \log w(b_{s,t}) \rfloor + 11$ *dot operations.*

One of the difficulties with Algorithm 18 is that if $|B_{s,t}| \ll \log w(B_{s,t}) - \log w_s + \log w(B^+_{s,t}) - \log b_{s,t}$ then the algorithm may require considerably more tests than would be required by a simple sequential search. This can occur in the algorithm when a long down phase occurs after a short up phase ($|B^+_{s,t}|$ is guaranteed to be at least as large as the number of dot operations performed in the up phase). We can make modifications to Algorithm 18 to prevent such a difficulty from occurring. Let λ be the last node found during the up phase for which the dot operator returns true. Let v_λ be the leaf node that is immediately to the right of the rightmost leaf node of $\Lambda(\lambda)$. One method of ensuring that the number of dot operations applied is at most

$O(w(B^+_{s,t}))$ is to perform a sequential search in lock step with the search in Algorithm 18. This secondary search need not start until the beginning of the down phase and can start from leaf node v_λ.

Another alternative is to create a second right sibling. We set this sibling to be the ancestor of v_λ whose distance to v_λ equals the distance from λ to its rightmost child (or rSib(λ) if it is closer). Let v_k be the node of B^T containing $b_{s,t}$. The chain joining the two right siblings of v_λ can be searched from both ends in lock step until either the search from above finds a node whose right child's subtree contains v_k or the search from below finds a node whose subtree contains v_k. From there, the down phase can continue as usual. A hybrid of these two methods may also be used in which v_λ is used instead of a second right sibling in the second method.

In Section A.4 we also need to solve the reverse of Problem A.1.1.

PROBLEM A.3.12. *Preprocess B so that, for any given end index s and test value t, one can find the largest subarray $B_{s:t}$ of B ending at $B[s]$ such that $B[i] \dot{<} t$ for every element $B[i]$ of $B_{s:t}$. If $B[s] \dot{<} t$ is false then $B_{s:t} = \emptyset$.*

In the non-weighted case, this problem can be solved by means totally analogous to the methods we developed for Problem A.1.1. In fact, the same data structure may be used to solve both problems. For the weighted case, we can of course just reverse the data in the original array or modify Algorithms 16, 17, and 18 accordingly to have the same effect. However, this would require two data structures and twice as much memory if we wanted to be able to solve both problems. A simple alternative is simply to create left siblings[2] for each node using an analogous version of Algorithm 17 and then create an analogous version of Algorithm 18 that uses left siblings instead of right siblings in order to solve Problem A.3.12.

[2]Note that the left sibling of the right sibling may not be the original node. This assymetric property of sibling nodes is of no consequence for our applications.

This is quite simple to do but there is a wrinkle; Algorithm 16 uses two stacks, S_1 and S_2, and they are used in an asymmetric manner. This asymmetry forces us to write a second set of the proofs to establish the complexity of the version of Algorithm 18 used for the case that we are solving in Problem A.3.12. Now the proofs that this algorithm solves Problem A.3.12 in the stated number of dot operations are analogous to the proofs for establishing the time complexity of Algorithm 18 except for the proof of Lemma A.3.10. This proof depends on internal use of the stacks in Algorithm 16 and thus there are problems relating to the asymmetric use of the stacks. Thus, we need to write a different proof for the version of this lemma we need in order to solve Problem A.3.12. We now have of proof of this result but it was not part of the version of this thesis submitted to my committe for evaluation and has thus been left out. It is, however, part of a paper on the results of this chapter that is in preparation.

Nevertheless, because Lemma A.3.6 still applies and Lemma A.3.9 can be modified to apply to the version of Algorithm 18 that solves Problem A.3.12, we can solve this problem in a logarithmic number of dot operations. We summarize with the following theorem:

THEOREM A.3.13. *The query part of Algorithm 18 can find subarray $B_{s:t}$ in at most* $3\lfloor \log w(B_{s:t}) - \log b_s \rfloor + \lfloor 3 \log n \rfloor + 6$ *dot operations when modified to solve Problem A.3.12.*

Let T_A be an augmented tree. For Section A.4 we need the following results. For any node v_h of T we define *left wall, left neighbor*, and $\mathrm{lSib}(v_h)$ analogously to the definitions of right wall, right neighbor and $\mathrm{rSib}(v_h)$.

PROPOSITION A.3.14. *Let p be an internal node of T_A and $t = lSib(lChild(p))$. If $t \neq nil$ then $w(t) \geq w(v)$.*

PROOF. Let $l = \mathrm{lChild}(p)$ and $r = \mathrm{rChild}(p)$. Assume $\mathrm{lSib}(l) \neq p$ since this proposition follows immediately otherwise. Clearly, l was on stack S_1 when l and r

were merged. Let u be the element directly below l on S_1 when l and r were merged. Let d be any rightmost descendant of u. Then $w(d) < w(l)$ since otherwise, once formed, d would not merge with any other node until l was formed and then d and l would be merged; a contradiction since l and r were merged. Therefore, either $t = u$ or t is an ancestor of u; either way $w(t) \geq w(u)$. Also $w(u) \geq w(r)$ or else l and u would have merged instead of l and r. Therefore $w(t) \geq w(r)$. □

Let u and v be nodes of an augmented tree such that either $u = \text{lSib}(v)$ and $\text{lSib}(v) \neq \text{parent}(v)$ or $u = \text{lChild}(\text{parent}(v))$ and $\text{lSib}(v) = \text{parent}(v)$. We denote $\overline{\text{lSib}}(v) = u$.

OBSERVATION A.3.15. *Let v_i, be a leaf node of T_A with a left sibling. Then the leftmost element of $\pounds(\overline{lSib}(v_i))$ is v_{i-1}.*

LEMMA A.3.16. *Let v be a a node of T_A and $q = u_1, u_2, \ldots, u_m$ be a path on T_A such that $lSib(u_i) = u_{i+1}$, $1 \leq i < m$, and $u_1 = \overline{rSib}(v)$. Now $len(q) \leq 3\lfloor \log w(parent(u_m)) - \log w(v) \rfloor + 3$.*

PROOF. If $\text{lSib}(v) = \text{parent}(v)$ then $w(\text{lSib}(u)) \geq w(v)$ by Proposition A.3.14 and then this lemma follows from Lemma A.3.8. So assume that $u = \text{lSib}(v)$. Then this lemma follows directly from Lemma A.3.8. □

We point out that if we had instead defined weighted trees based on a variation of the weighted tree structure in **[39]**, then Lemma A.3.16 would not generally hold.

A.4. Range Finding in Unordered Trees

For simplicity, in this section, we assume that the elements of B are real intervals, t is a real number, $\dot{<}$ is the binary operator such that $I \dot{<} t$ if t is contained in interval I, and the operator max, when used in the preprocessing stage of Algorithm 16, is replaced by the (interval) intersection operator.

We now consider the case when the elements of B are stored on the vertices of a rooted tree. Let T be a tree with n nodes. Let the vertices of T be v_i, $0 \leq i < n$, where vertex v_{i-1} is the ith vertex traversed in an in-order traversal of T. We store b_i at vertex v_i of T. To simplify matters, we will treat the vertices of T as the elements of B so that the operations normally applied to the elements of B may be applied directly to the vertices of T. Thus each element of T is a real interval. For example, we use $v_i \dot{<} t$ to denote $b_i \dot{<} t$.

We also add a constraint on the values of B stored in T.

CONSTRAINT A.4.1. Let v_i be any internal node of tree T with two child nodes. Then $\text{lChild}(v_i) \cap \text{rChild}(v_i) = \emptyset$.

The consequence of Constraint A.4.1 is that, for any real number x, it is impossible that both $\text{lChild}(v_i) \dot{<} x$ and $\text{rChild}(v_i) \dot{<} x$.

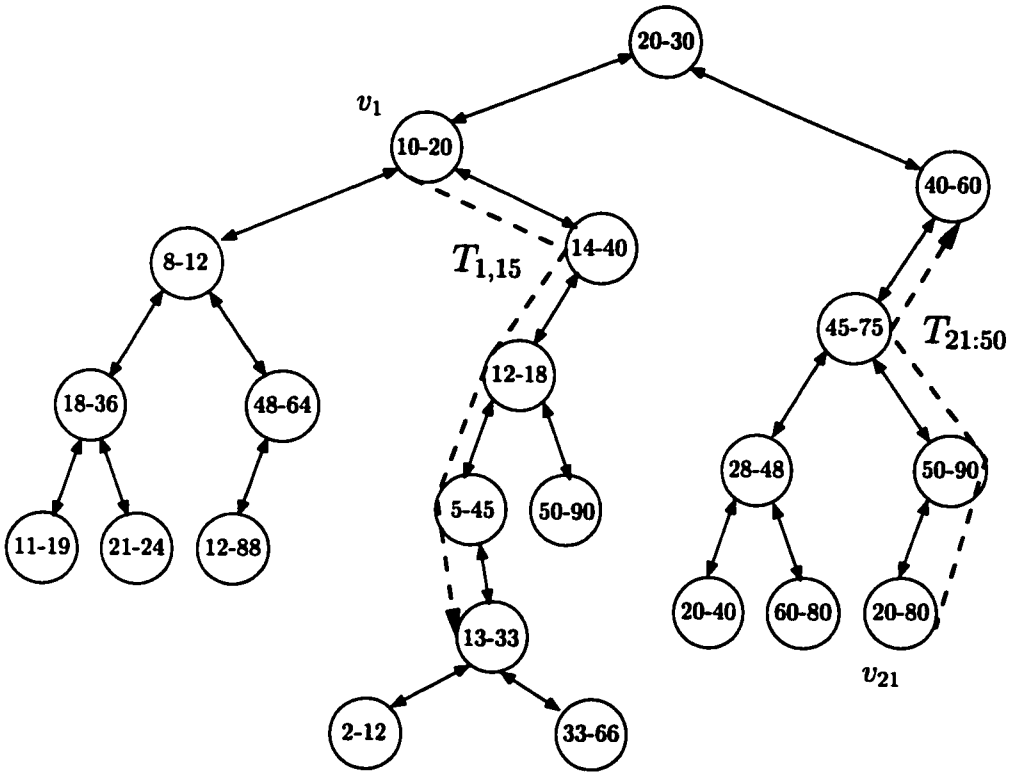
We call any path in a tree, with the root and a leaf as its endpoints, a *root path* and any subpath of a root path a *root subpath*. There are two main problems we are interested in solving.

PROBLEM A.4.2. *Preprocess T so that, for any vertex v_s of T and real number t, one can quickly find the longest path, denoted $T_{s,t}$, with top vertex v_s, such that t is contained in every interval on $T_{s,t}$.*

By Constraint A.4.1, for any real number x and vertex v_i of T, it is impossible that both $\text{lChild}(v_i) \dot{<} x$ and $\text{rChild}(v_i) \dot{<} x$. As a consequence, at most one of $\text{lChild}(v_i)$ and $\text{rChild}(v_i)$ can be on the solution path to Problem A.4.2.

PROBLEM A.4.3. *Preprocess T so that, for any given vertex v_s and real number t, one can quickly find the longest path, denoted $T_{s:t}$, with bottom vertex v_s, such that t is contained in every interval on $T_{s:t}$.*

Let p be any root subpath of T. We denote the endpoint of p closest to the root by $\text{top}(p)$ and the other endpoint of p by $\text{bot}(p)$. We will show that, after



$T_{1,15} = (\{10, 20\}, \{14, 40\}, \{12, 18\}, \{5, 45\}, \{13, 33\})$.
$T_{21:50} = (\{20, 80\}, \{50, 90\}, \{45, 75\}, \{40, 60\})$.

FIGURE A.4.1. Finding root subpaths on a tree

linear time preprocessing of T, we can answer a query for Problem A.4.2 or A.4.3 in $O(\log |\Lambda(\text{top}(q))|)$ dot operations where q is the root subpath found. We achieve this query complexity, despite the fact that we may have $O(\log n)$ paths to query, by using the algorithms in the previous section and, in particular, by exploiting the negative terms in the complexity of the query phase of Algorithm 18. We will solve both Problems A.4.3 and A.4.2 using the following strategy:

Searching a single element of π in Strategy A.4.1 may require $O(\log n)$ dot operations, and $O(\log n)$ paths may need to be searched. This suggests that $O(\log^2 n)$ dot operations may be required. However, by using the algorithms in the previous

Strategy A.4.1

Preprocessing:

Input: A tree T.

1: Partition T into a set of root subpaths π such that any root path of T intersects at most $O(\log|T|)$ paths of π.

2: Construct an augmented weighted tree (with both left and right siblings) on each path in π where the values of the weights depend on the sizes of subtrees of T attached to the path only at a vertex.

Query:

Input: A vertex v_s of T and a test interval t.

Output: $T_{s,t}$ if we are solving Problem A.4.2 or $T_{s:t}$ if we are solving Problem A.4.3.

1: Find the longest (up or down) subpath, say P_s, on the path of π starting from v_s such that the interval on every node on P_s contains t. Do this using the augmented weighted tree built for the path.

2: **while** there is a path P_π of π through which the path P_s can be continued **do**
Extend P_s as far as possible along P_π using the augmented weighted tree built for P_π.

section and by exploiting the negative terms in the complexity of the query phase of Algorithm 18, we will show that only $O(\log n)$ dot operations are required in total.

Our first step, in refining Strategy A.4.1 is to partition T into root subpaths such that any root path of T intersects at most $O(\log n)$ paths of the partition. We achieve this by using a centroid path decomposition of T **[18]** which we denote by $\pi(T)$. We construct $\pi(T)$ as follows.

For each internal node we select the edge to the child node that is the root of the largest subtree. We resolve ties by choosing (arbitrarily) the rightmost edge. Now $\pi(T)$ is the smallest subgraph of T containing the selected edges.

We call any path of $\pi(T)$ an φ-path. We denote by $\lambda(T)$ the maximum number of φ-paths intersected by any root path of T. See Figure A.4.2. Let φ be any φ-path of $\pi(T)$. If φ does not contain the root of T then we say that φ is *blocked* both by parent (top(φ)) and by the path that contains parent (top(φ)). We denote the φ-path of $\pi(T)$ that blocked φ by φ^ρ. If φ contains the root of T then $\varphi^\rho = \text{nil}$. If $\varphi^\rho \neq \text{nil}$

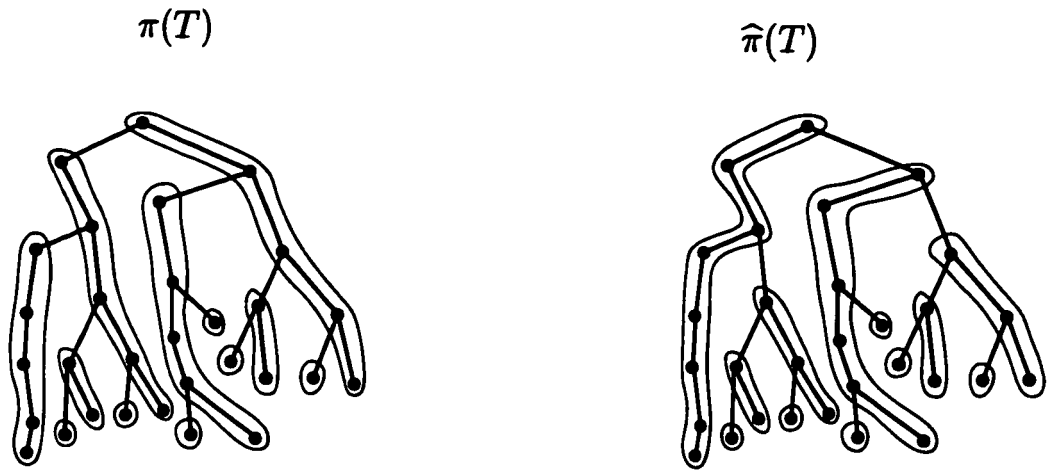


FIGURE A.4.2. Two methods of partitioning a tree T into paths

then we denote the child vertex of parent (top(φ)) that is not top(φ) by φ^{blk}. We denote $\Lambda(\varphi) = \Lambda(\text{top}(\varphi))$.

LEMMA A.4.4. $\lambda(T) \leq \lfloor \log(n+1) \rfloor$.

PROOF. Let q be a root path of T that intersects $\lambda(T)$ elements of $\pi(T)$. Let $\varphi_1, \varphi_2, \ldots, \varphi_{\lambda(T)}$ be the φ-paths of $\pi(T)$ intersected by q given in the order that they are first encountered during a traversal of q starting from the root of T. Then $|\Lambda(\varphi_1)| = n$ and $|\Lambda(\text{parent}(\text{top}(\varphi_i)))| \geq 2|\Lambda(\varphi_i)| + 1$ for $1 \leq i < \lambda(T)$ by the rules for constructing $\pi(T)$. Thus $n \leq 2^{\lambda(T)} - 1$. This result now follows. □

In implementing Strategy A.4.1 we need to search the paths in $\pi(T)$ efficiently. We will use the methods developed in Section A.3 to do a range query in weighted arrays. The next step in the refinement of our algorithm is to assign weights to the vertices of T. For each vertex v_i of T we denote by $w(v_i)$ the weight of v_i. For each path φ of $\pi(T)$ such that $\varphi^{\rho} \neq \text{nil}$ we set $w(\varphi^{\text{blk}})$ to $|\Lambda(\varphi)|$. For all remaining vertices of T we assign a weight of one. Next, for each path φ of $\pi(T)$, we construct a weighted tree using Algorithm 16 and then Algorithm 17 twice to add both left and right siblings. We use the vertices of φ as the elements of B in this construction. The

weight we assigned to vertex v_i is also the weight we associate with b_i, $0 \leq i < n$, for use in Algorithms 16 and 18. We denote the augmented weighted tree constructed for φ-path φ by $W(\varphi)$. For any vertex v_i of φ we call the node of $W(\varphi)$ containing v_i a *wnode* and denote it by wNode(v_i). We set the left siblings and right siblings of the nodes in $W(\varphi)$ to be directed towards top(φ) and bot(φ) respectively.

At this point, we make a change that simplifies our queries when searching upwards for up root subpaths. If $\varphi^\rho \neq$ nil then for each node of $W(\varphi)$ whose left sibling is nil we change the left sibling to $\overline{\text{lSib}}(\text{node}(\varphi^{\text{blk}}))$. The effect of this change should be noted. In Algorithm 18 (the version that searches to the left), if the up phase of the search is completed solely because a node with no left sibling is reached then $B[0, s]$ is returned implying that every element of the array from $B[0]$ to $B[s]$ passes the test being applied. In the current situation, this means that all of the portion of the current φ-path from the start of the search to the top vertex is part of our solution. Let φ be the current φ-path and η be the last wnode of $W(\varphi)$ checked by Algorithm 18. Then we must continue the query upward starting at wNode(parent(top(φ))). Now, by Observation A.3.15, $\ell(\overline{\text{lSib}}(\text{node}(\varphi^{\text{blk}})))$ = wNode(parent(top(φ))) so it is valid for the range query to continue from this point.

The benefit of this trick is that we can now apply Algorithm 18 (almost) directly to solve Problem A.4.3; there is a caveat however. At Step 5 of Algorithm 18 one is subtracted from index(κ) since index(κ) is one beyond the value we want. In Problem A.4.3, if we are at a node of T with only one child node at this point then we do the same thing. However, if we are at a node of T with two child nodes we must determine which of the child nodes is the root node of the previous φ-path searched. This is easily done in $O(1)$ time and requires no dot operations.[3]

[3]If multi-way trees are used (see Section A.4.1) then a weighted binary tree structure must be searched to find the correct child at this point. This cost is logarithmic in the number of children the node has.

There is also a cost of this trick. First, Algorithm 18 is analyzed using Theorem A.3.8 which says that it is possible to traverse as many as two right (now left) sibling links without the weight of the nodes traversed increasing by a factor of two. This trick allows two additional links to be traversed without the weight of the final node being double that of the first. This analysis is not tight.

Second, that $w(\overline{\text{lSib}}(\text{node}(\varphi^{\text{blk}}))) \geq w(\text{wNode}(\text{parent}(\text{top}(\varphi))))$ cannot be guaranteed. However, Lemma A.3.16 ensures us that the additional costs due to this fact is at most one additional dot operation. This is needed to ensure that the worst case cost of our solution to Problem A.4.3 is $\Omega(\log n)$ rather than $\Omega(\log^2 n)$.

Thus, in total, this trick can cost us up to three dot operations each time it is applied. Thus, if the up root subpath we find crosses j of the φ-paths then we incur an additional cost of up to $3j$ dot operations. Other than the additional costs for this trick, the analysis for Algorithm 18 still applies.

Of course the analysis of Algorithm 18 is given in terms of weights on the elements of an array; we need an analysis in terms of T. So let us transform the analysis of Algorithm 18. Let φ be the φ-path containing $\text{top}(T_{s:t})$. Then, from the above analysis and Theorem A.3.13, finding $T_{s:t}$ requires at most $3\lfloor \log |T_{s:t}| - \log w(\text{node}(\text{bot}(v_s))) \rfloor + 2j + 3\lfloor \log |\varphi| \rfloor + 7)$ dot operations. We summarize these results in the following theorem.

THEOREM A.4.5. *A query for Problem A.4.3 can be solved in at most*

$$3\lfloor \log |T_{s,t}| - \log w(v_s) \rfloor + 2j + 3\lfloor \log |\varphi| \rfloor + 6) < 8 \log n + 7$$

dot operations where j is the number of φ-paths crossed by $T_{s:t}$, and φ is the φ-path containing top $(T_{s:t})$.

Performing a range query for Problem A.4.2 is no more difficult. The case that $T_{s,t}$ is contained in a single φ-path is solved by a single application of Algorithm 18 to $W(\varphi)$ where φ is the φ-path containing $T_{s,t}$. We consider now the case that $m \geq 2$ φ-paths are crossed by $T_{s,t}$. Let $\varphi_{s,t}$ denote the φ-path containing the last vertex of $T_{s,t}$. If the leaf vertex of $\varphi_{s,t}$ is contained in $T_{s,t}$ then let $v_{s,t} = \text{bot}(T_{s,t})$; otherwise let $v_{s,t}$ denote the vertex of $\varphi_{s,t}$ immediately following $\text{bot}(T_{s,t})$ on $\varphi_{s,t}$. Let $T^+_{s,t}$ denote the path from v_s to $v_{s,t}$. Let φ_j, $1 \leq j \leq m$, be the set of φ-paths crossed by $T_{s,t}$, given in the order in which they are crossed. To avoid an using $O(\log^2 n)$ dot operations we query the first φ-path crossed using Algorithm 18 and query the remaining φ-paths by searching their cooresponding augmented weighted trees downward starting from their root nodes. We denote by $\overline{w}(v_i, v_j)$ the total weight of the vertices on the path of T whose end vertices are v_i and v_j. Let n_s be the size of the subtree of T rooted at v_s. The total number of dot operations applied is at most:

$$
\begin{aligned}
& 3\left\lfloor \log \overline{w}(v_s, \text{parent}(\varphi_2^{\text{blk}})) - \log w(v_s) \right\rfloor + 3\left\lfloor \log \overline{w}(v_s, \varphi_2^{\text{blk}}) - \log w(\varphi_2^{\text{blk}}) \right\rfloor + 11 \\
+ \;& \Sigma_{j=2}^{m-1}(3 \log w(\varphi_j) - \log w(\varphi_{j+1}) + 3) + 3\left\lfloor \log w(\varphi_m) - \log w(v_{s,t}) \right\rfloor + 3 \\
< \;& 6 \log \overline{w}(v_s, \text{node}(\varphi_2^{\text{blk}}) - 3 \log w(v_s) - 3 \log w(v_{s,t}) + 3m + 8 \\
< \;& 9 \log n_s + 8 .
\end{aligned}
$$

We summarize these results with the following theorem.

THEOREM A.4.6. *A query for Problem A.4.2 can be solved in at most* $9 \log n_s + 8$ *dot operations.*

The only difficulty with this result is that we have lost the sensitivity to the length of $T_{s,t}$. This is because we have stopped the up phases of our searches of the φ-paths after processing the first φ-path; the phase that ensures sensitivity to the length $T_{s,t}$. We can correct this by storing some additional information in our data structure. We need to be able to continue the up phase of the search of the next φ-path, if needed,

when we switch φ-paths. In detail we do this as follows: Assume that we have just completed a range query along an φ-path and are about to start a new range query along the next φ-path, say $\overline{\varphi}$. Let τ be the number of right links taken during the previous up phases in the previous range queries along φ-paths. Then we start an up phase search in the next φ-path starting at the subtree $\tau+1$ levels above the level of the node containing the top vertex of the φ-path. This change requires that an array large enough to store all nodes on the path of $W(\overline{\varphi})$ from the leftmost leaf node to the root be stored at φ-path $\overline{\varphi}$. In the worst case, adding this array, for each φ-path, requires $\Omega(n)$ additional space. This change, in combination with the optimizations suggested for Algorithm 18 in Section A.3, ensures that at least as many up-links are taken as down-links. Thus, the total number of dot operations applied is at most proportional to the length $T_{s,t}$.

Using the results of this section we can also solve the problem of finding the longest path on T containing an input vertex v_s such that a test input real number t is on every interval on the path. We state this as a theorem.

THEOREM A.4.7. *We can preprocess T in linear time so that, for any given vertex v_s and real number t, we can find the longest path on T containing v_s such that t is contained in every interval on the path in $O(\log n)$ dot operations.*

It may appear that we can also use the weighted binary tree structure in algorithm in [**39**, pg 177-187] (with some modifications) and achieve the same results in this section as we achieved with using the weighted tree structure described in Section A.3. The basic idea in constructing the weighted binary tree in this case is to recursively partition the list of values into two parts that have as close to the same total weight as possible. We presume that the same method of constructing left siblings and right siblings would be used and that the same computation of $\overline{\text{lSib}(v)}$ for leaf node v would be used as presented here. We also assume that results similar to the results proven

for our augmented weighted binary tree structure can be proven for this augmented weighted tree structure except for Lemma A.3.16. Now, Lemma A.3.16 does not apply to this structure and this can cause a query for Problem A.4.3 to take $\Omega(\log^2 n)$ time in the worst case. To see this consider a tree T containing $2^m + m - 4$ edges such that $\pi(T) = \{P_1, P_2, ..., P_m\}$ where

- $|P_1| = 1$,
- $|P_2| = 2$,
- $|P_3| = 6$,
- $|P_k| = 2 * |P_{k-1}| - 1$, $4 \leq k \leq m$,
- $\text{parent}(\text{top}(P_j)) = \text{parent}(\text{bot}(P_{j+1}))$, $1 \leq j < m$, and
- the intervals on the nodes of T are assigned values such that, for some query value t, $\text{top}(T_{\text{bot}(P_1):t}) = \text{top}(P_m)$.

Now consider a query for $T_{\text{bot}(P_1):t}$. Observe that, for this query. we must perform a query on each φ-path of $\pi(T)$ and, for each φ-path except the first, we start the search at the second vertex from the bottom, which has weight one. Furthermore, for each φ-path except the first, all the nodes of the portion of the φ-path on which a range query is performed have weight one. It is thus easily shown that this query requires $\Omega(\log^2 |T|)$ time.

We could fix this difficulty by storing with each node of each φ-path the intersection of the intervals of the φ-path from the node to the top node of the φ-path. Then before beginning a search on an φ-path we can first test if the search is necessary in one dot operation. This works for Problem A.4.3.

However, we cannot apply this fix to solve the circular ray shooting problem in Chapter 5 without causing the memory needed to increase from $O(n \log n)$ to $O(n^2)$. This is because for the circular ray shooting problem we must store Voronoi diagrams instead of intervals and a single diagram may require $O(n)$ space.

A.4.1. Handling Multi-way Trees. It is worth noting that the results of this section are easily modified to deal with trees that are not binary. We simply have to add a weighted binary tree at the internal nodes that have degree greater than three.[4] The leaves of this weighted binary tree in left to right order are the weights of the subtrees at this node in left to right order with the exception of the subtree containing part of the current φ-path. This adds no additional costs for solving Problem A.4.3 because the trick we used in solving this problem bypasses these weighted binary trees. But these weighted binary trees do add an additional cost to Problem A.4.2. For this problem, when switching between φ-paths, we must search the weighted binary tree placed at the corresponding internal node. This does not affect the complexity of the algorithm for solving Problem A.4.2 in the worst case. However, we can no longer guarantee that the number of dot operations required to solve the problem is at most proportional to the length of the path found. In fact, this is clearly impossible to do since searching the weighted binary tree at an internal node to determine along which child node to proceed can require $O(\log n)$ dot operations. Yet, having done so, we can extend the path by at most one vertex.

Another option is to construct a binary tree based on T by replacing each node of degree three by a binary tree whose leaves are the subtrees of the internal node and whose root is the internal node. If this is done it is important that this binary tree be constructed based upon the weighted binary tree that would otherwise be stored at the internal node. Otherwise, the claims on the complexity of solving Problem A.4.2 are no longer valid.

Also, if this approach is used then the algorithm to solve Problem A.4.3 will have to step through the nodes of this binary tree. This does not increase the worst case

[4]We may at this point construct a weighted binary tree for the node using the methods of Section A.3. However, a standard weighted binary tree also works here since, by Constraint A.4.1, the subtrees may be assumed to be in sorted order. For this problem, we believe, a standard weighted binary tree is more efficient by a small constant factor.

complexity of this problem but does mean that a potential saving is lost since the weighted binary tree corresponding to the internal node must be traversed.

A.4.2. Partitioning Trees into Paths without Weights. We have seen how to partition T into φ-paths so that a root path can intersect at most $\log n$ of the φ-paths. We now address the problem of partitioning T into root subpaths such that the number of them intersected by any root subpath of T is minimized. We use this result in Chapter 4 to partition a polygon into y-monotone subpolygons such that the monotone cover number of the partition is within two of the minimum monotone cover number of any partition of the polygon.

Again, we partition T into paths directed towards the root. We use the same method used to construct $\pi(T)$ except that we change the rule used to determine which path is blocked at an internal node of degree two. We apply the rule that the longest path is the one to reach the internal node with ties resolved by choosing (arbitrarily) the rightmost path. We denote this partition of T into paths by $\widehat{\pi}(T)$. We call any element of $\widehat{\pi}(T)$ a $\widehat{\varphi}$-path.

Let φ be any $\widehat{\varphi}$-path of $\widehat{\varphi}(T)$. We denote the number of times that φ was allowed to continue during the construction of $\widehat{\varphi}(T)$ in a tie situation by $\widehat{\lambda}(\varphi)$. We call $\widehat{\lambda}(\varphi)$ the $\widehat{\lambda}$-number of φ. We further denote by $\widehat{\lambda}(T)$ the largest $\widehat{\lambda}$-number of any element of $\widehat{\pi}(T)$; note that $\widehat{\lambda}(T)$ is determined by the element of $\widehat{\lambda}(T)$ that contains the root of T. We refer to $\widehat{\lambda}(T)$ as the $\widehat{\lambda}$-number of T.

The partition $\widehat{\pi}(T)$ is optimal in the following sense:

LEMMA A.4.8. *Let Π be any partition of T into root subpaths and m_Π be the maximum number of paths of Π intersected by any root path of T. Then $\widehat{\lambda}(T) \leq m_\Pi$.*

PROOF. For each element φ of $\widehat{\pi}(T)$ that was blocked in a non tie situation, remove from T all of $\Lambda(\varphi)$. Then contract the remaining tree by contracting to an edge each maximal path on the remaining tree whose internal vertices have degree

two while maintaining the labels of the start and end vertices of each path. The result is a complete binary tree T' with k nodes where $k \leq n$. Now, $\widehat{\lambda}(T)$ is the number of levels in T'. Therefore $\widehat{\lambda}(T) = \log(k+1) \leq \lfloor \log(n+1) \rfloor$.

Note that each vertex of T' is also a vertex of T. Furthermore, any root subpath p on T, covering at least one vertex of T that is also a vertex of T', may be transformed into a root subpath p' on T' such that the vertices covered by p' on T' are also covered by p on T. Let $\overline{\Pi}$ be the set of subpaths that result if this transformation is applied to every applicable element of Π. Now, $\overline{\Pi}$ is a partition of T' into root subpaths. Let $m_{\overline{\Pi}}$ be the maximum number of paths of $\overline{\Pi}$ intersected by any root path of T'. Clearly $m_{\overline{\Pi}} \leq m_{\Pi}$. Furthermore, we can always find a path from the root of T' to a leaf that crosses a new path of $\overline{\Pi}$ for each non-leaf vertex on the path. Therefore $m_{\overline{\Pi}} \geq \log(k+1)$. But $\widehat{\lambda}(T) = \log(k+1)$. Therefore $\widehat{\lambda}(T) \leq m_{\overline{\Pi}} \leq m_{\Pi}$. □

COROLLARY A.4.9. $\widehat{\lambda}(T) \leq \lfloor \log(n+1) \rfloor$.

COROLLARY A.4.10. $|\mathcal{L}(\Lambda(T))| \geq 2^{\widehat{\lambda}(T)-1}$.

The following lemma is needed in Chapter 4.

LEMMA A.4.11. *Let t_r be a subtree of T containing the root of T but not of it leaves and $\{t_1, t_2, \ldots, t_k\}$ be the subtrees in the graph $T - t_r$. Assume wlog that $\widehat{\lambda}(t_1) \geq \widehat{\lambda}(t_i)$, $2 \leq i \leq k$. Assume that there exists a root path of T that intersects the maximum number of elements of $\widehat{\pi}(T)$ and that intersects t_1. Then $\widehat{\lambda}(T) \leq \widehat{\lambda}(t_1) + \lfloor \log k \rfloor$.*

PROOF. Let $l = \widehat{\lambda}(T) + 1 - \widehat{\lambda}(t_1)$ and T_l be the maximum subtree of T rooted at the root of T such that the subtrees of T rooted at the leaves of T_l all have a $\widehat{\lambda}$-number of $\widehat{\lambda}(t_1)$. Then $|\mathcal{L}(T_l)| \geq 2^l$ by Corollary A.4.10. Now, since $\widehat{\lambda}(t_1) \geq \widehat{\lambda}(t_i)$, $2 \leq i \leq k$, and the elements of $\{t_1, t_2, \ldots, t_k\}$ contain every leaf of T, we have that every element of $\mathcal{L}(T_l)$ is the root of a subtree of T that contains at least one element of $\{t_1, t_2, \ldots, t_k\}$. Therefore $k \geq 2^l$ from which this lemma follows. □

APPENDIX B

Polygon Area Problems

B.1. Introduction

One of the basic problems of geometry is the computation of the area or volume of geometric objects. A number of interesting computational problems on the area of polygons have been studied; see for example [22, 44, 46, 48]. In [19] a number of problems and algorithms are described relating to the computation of the area or volume of geometric objects. In particular the following problem is solved:

PROBLEM B.1.1. *Let $P = \{v_0, \ldots, v_{n-1}\}$ be a simple polygon. Preprocess P so that for a query chord γ of P we can quickly determine the area of $P_{\overline{\gamma}}$.*

The solution to this problem in [19] requires $O(n \log n)$ preprocessing, $O(n \log n)$ space and $O(\log n)$ query time. See also [20]. In this appendix we present a solution to this problem that requires linear preprocessing and space and $O(1)$ query time. We also show that our algorithm can be used to solve more general problems with the same preprocessing, space, and query complexities.

We originally solved Problem B.1.1 using the 2^*-wind visibility partition of a simple polygon and Strategy 2.5.1. In fact, we initially created the 2^*-wind visibility partition class specifically to solve Problem B.1.1. The algorithm that we developed requires linear preprocessing and space and $O(1)$ query time and is much simpler than the algorithm in [19]. We then became interested in the fact that this algorithm sometimes solves a more difficult problem.

Let P be a simple polygon. We call a line segment whose endpoints are on the boundary P a *pseudo chord* of P. We note that any pseudo chord of P partitions P

into two subpolygons which are coincident along the pseudo chord. Assume v_0 is the first vertex of P. For any pseudo chord γ of P we denote by P_γ the subpolygon of P determined by γ such that if v_0 is an endpoint of γ then P_γ contains the vertices of P with smaller indices and otherwise P_γ does not include vertex v_0. Note that this definition does not require that P be simple nor does it imply that P_γ is simple. The remaining subpolygon of P determined by γ we denote by $P_{\overline{\gamma}}$. Note that if γ is a chord then this definition of P_γ and $P_{\overline{\gamma}}$ corresponds to the definitions of P_γ and $P_{\overline{\gamma}}$ when γ is a chord given in Chapter 2.

PROBLEM B.1.2. *Let $P = \{v_0, \ldots, v_{n-1}\}$ be a simple polygon. Preprocess P so that for a query pseudo chord γ of P, such that $P_{\overline{\gamma}}$ is a simple polygon, we can quickly determine the area of $P_{\overline{\gamma}}$.*

This problem includes Problem B.1.1 as a special case. However, the algorithm in [19] for solving Problem B.1.1 cannot be used to solve Problem B.1.2. In fact, at first glance, this problem appears to require linear time since γ may intersect $O(n)$ edges of P; see Figure B.1.1. We show however, that like Problem B.1.1, this problem can be solved with linear preprocessing and space and $O(1)$ query time. Furthermore the algorithm we present to solve this problem is very simple; it does not even require the triangulation of P. We also show that the algorithm can be generalized to work on self-overlapping polygons (see [45]) once the area for such polygons is defined.

B.2. Calculating the Area of a Polygon

There are numerous methods of calculating the area of a simple polygon; for a survey see [19]. Of particular interest to us is the method known as the polar formula. For any line segment $\gamma = \overline{(x_a, y_a)(x_b, y_b)}$ we denote $\overline{\mathbf{A}}(\gamma) = x_a y_b - x_b y_a$. We point out that $\overline{\mathbf{A}}(\gamma)$ equals twice the area of the triangle whose vertices are the origin and the endpoints of γ if $a \geq b$ and the negative of this value if $a < b$.

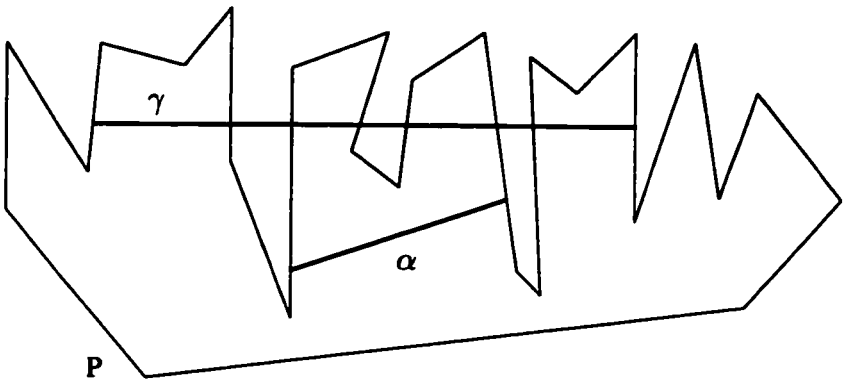


FIGURE B.1.1. A polygon P with a chord α and a pseudo chord γ ($P_{\overline{\gamma}}$ is simple)

We denote the area of a polygon P by $\mathbf{A}(P)$. Let $P = \{v_0, e_0, v_1, e_1, \ldots, e_{n-1}\}$ be a simple polygon. Then the polar formula for the area of P can be written:

$$\mathbf{A}(P) = \frac{1}{2}\sum_{i=0}^{n-1} \overline{\mathbf{A}}(e_i) \tag{B.2.1}$$

Note that the right hand side of equation B.2.1 can be applied to any polygon; that is the simplicity of P is not assumed. We now turn equation B.2.1 around and say that equation B.2.1 defines the area of P. Thus the area of every polygon is defined and if the polygon is simple then our definition of area corresponds to the normal definition.

Now to solve Problem B.1.2 and thus also Problem B.1.1, we use the idea of partial sums. For each edge $e_k \in V(P)$ we denote:

$$s_k = \sum_{i=0}^{k} \overline{\mathbf{A}}(e_i) \tag{B.2.2}$$

We say that a pseudo chord γ of P is a pseudo diagonal of P if the endpoints of γ are vertices of P. Let γ be a pseudo diagonal of P with endpoints v_j, v_k where $j < k-1$. Then by equations B.2.1 and B.2.2 we have:

$$\begin{aligned} \mathbf{A}(P_{\overline{\gamma}}) &= \frac{1}{2}\left(\sum_{i=0}^{j-1}\overline{\mathbf{A}}(e_i) + \overline{\mathbf{A}}(\gamma) + \sum_{i=k}^{n-1}\overline{\mathbf{A}}(e_i)\right) \\ &= \mathbf{A}\{P\} + \frac{-s_{k-1} + s_{j-1} + \overline{\mathbf{A}}(\gamma)}{2} \\ &= \frac{s_{n-1} - s_{k-1} + s_{j-1} + \overline{\mathbf{A}}(\gamma)}{2} \end{aligned} \tag{B.2.3}$$

Consider now a chain C with the same endpoints as γ. Let

$$C = \{v_j = w_0, h_0, w_1, h_1, \ldots, h_{m-1}, w_m = v_k\}$$

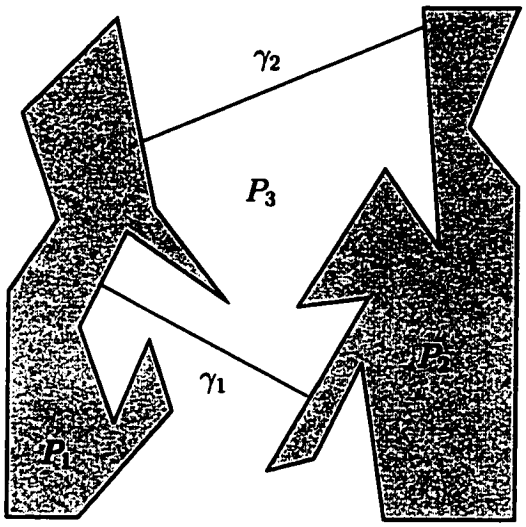
where the vertices of C are $\{w_i : 0 \leq i \leq m\}$ C and the edges of C are $\{h_i : 0 \leq i < m\}$. Consider the polygon

$$\begin{aligned} P_C = \{&v_0, e_0, v_1, \ldots, v_j, h_0, w_1, h_1, w_2, \ldots, \\ &w_{m-1}, h_{m-1}, v_k e_k, v_{k+1}, \ldots, v_{n-1}, e_{n-1}\}. \end{aligned}$$

By equations B.2.1 and B.2.3 we have:

$$\mathbf{A}(P_C) = \frac{s_{n-1} - s_{k-1} - s_{j-1} + \sum_{i=0}^{m-1}\overline{\mathbf{A}}(h_i)}{2} \tag{B.2.4}$$

Consider now that γ is a pseudo chord of P with endpoints p_j, p_k such that $p_j \in \overline{v_j v_{j+1}}$ and $p_k \in \overline{v_{k-1} v_k}$. Then we can compute $\mathbf{A}(P_{\overline{\gamma}})$ by computing $\mathbf{A}(P_C)$ where the vertices of C in order are v_j, p_j, p_k, v_k.



Using methods similar to those described here the polygons P_1 and P_2 can be preprocessed in linear time so that the area of a query polygon P_3 determined by line segments γ_1 and γ_2 can be computed in constant time.

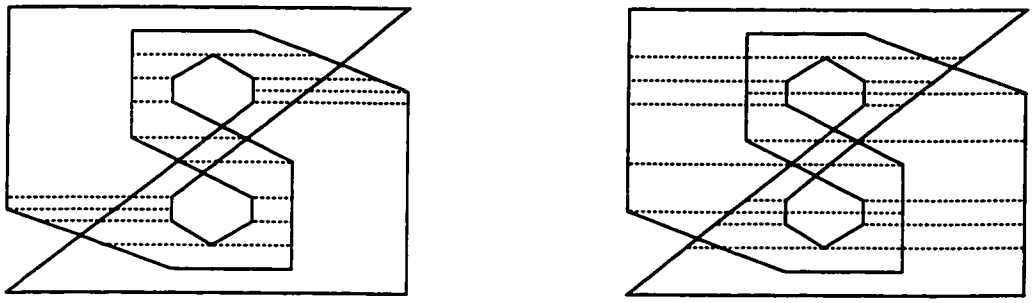
FIGURE B.2.1. An area problem on two polygons

We now have a method to compute $\mathbf{A}(P_{\overline{\gamma}})$; use C in equation B.2.4. What will the cost of computing $\mathbf{A}(P_{\overline{\gamma}})$ be? Note first that, for $0 \leq i \leq n-2$, clearly $s_{i+1} = s_i + \overline{\mathbf{A}}(e_i)$. Therefore the values s_i, for $0 \leq i < n$, can easily be computed in linear time in total. From these facts and Equation B.2.4 we obtain the following result:

THEOREM B.2.1. *Problem B.1.2 (and thus also Problem B.1.1) can be solved with linear preprocessing, linear space, and a query complexity of $O(1)$.*

We can apply the results of this section to situations involving more than one polygon. Consider for example the following problem.

PROBLEM B.2.2. *Let P_1, P_2 be two disjoint simple polygons. Preprocess P_1 and P_2 so that for a query simple polygon P_3 whose boundary is constructed from a subchain C_1 of P_1, a subchain C_2 of P_2, and pair of line segments γ_1, γ_2 joining C_1 and C_2, the area of P_3 can be computed quickly.*



A self-overlapping polygon may have multiple distinct trapezoidizations. However the total area of all trapezoids in a trapezoidization is independent of the trapezoidization chosen.

FIGURE B.2.2. A self overlapping polygon with two trapezoidizations

It can be shown by methods similar to the above that Problem B.2.2 can be solved with linear preprocessing and space and a query complexity of $O(1)$. See Figure B.2.1.

We can also generalize the results of this section to the situation where a polygon is partitioned into two components by a curve. Let P be a simple polygon and let C be a curve interior to P except at its endpoints. Let the endpoints of C be p and q such that p is reached before q in a traversal of P and neither endpoint is start(P). Let γ be the pseudo chord of P with endpoints p and q. Assume that the concatenation of C and γ forms a simple curve, say $\overline{C}$, and that the interior of C is contained in P_γ. Let C_A be the area of $\overline{C}$. Then the areas of the two components of P determined by C are $\mathbf{A}(P_{\overline{\gamma}}) + C_A$ and $\mathbf{A}(P_\gamma) - C_A$. Furthermore, the cost of computing these areas, after performing the linear preprocessing described in this section, is $O(1)$ plus the cost of computing C_A.

If C is a circular arc then C_A can be computed in $O(1)$ time. In this case $\mathbf{A}(P_{\overline{\gamma}}) + C_A$ and $\mathbf{A}(P_\gamma) - C_A$ can be computed in $O(1)$ time. Now consider that we are given a set of circular arcs of a circle that are the maximal arcs of the circle contained in a polygon P with n vertices and that we are given the intersection points of these arcs with P. If there are k such arcs then we can compute the area of the intersection of

P with the circle in $O(k)$ time after linear preprocessing of P. If we are only given the circle then we can find these k arcs in at most $O(k \log^2 n)$ time after $O(n \log n)$ preprocessing and using $O(n \log n)$ space by using the algorithm in Chapter 5. We state this as a theorem.

THEOREM B.2.3. *Given a simple polygon P with n vertices we can preprocess P in $O(n \log n)$ time and using $O(n \log n)$ space so that when given a query circle C we can compute the area of intersection of P and C in $O(k \log^2 n)$ time where k is the number of arcs into which C is partitioned by P.*

B.3. Areas of Self-overlapping Polygons

Self-overlapping polygons [45] have the interesting property that, even though they are not generally simple polygons, they can, by the definition of triangulation given in [45], be triangulated. Equivalently we can construct a horizontal trapezoidization of a self-overlapping polygon. However, in contrast to simple polygons, a self-overlapping polygon may have multiple distinct horizontal trapezoidizations; see Figure B.2.2 and also [45]. This fact makes the question of the area of a self-overlapping polygon an interesting one.

We note first that the following observation follows directly from equation B.2.1.

Let P be a (not necessarily simple) polygon and let γ be a pseudo chord of P. Let P_1 and P_2 be the two subpolygons of P determined by γ. Then $\mathbf{A}(P) = \mathbf{A}(P_1) + \mathbf{A}(P_2)$.

It is easily shown using induction and Observation B.3 that the sum of the areas of the trapezoids of a horizontal trapezoidization of a self-overlapping polygon is equal to the area computed for the polygon by Equation B.2.1. Thus no matter which horizontal trapezoidization is used (assuming that there is more than one) the sum of the areas of the trapezoids will be the same. We summarize with the following result:

LEMMA B.3.1. *Let P be a self-overlapping polygon with more than one distinct horizontal trapezoidization and let T_1, T_2 be two distinct horizontal trapezoidizations of P. Let $\mathbf{A}_1$ and $\mathbf{A}_2$ be the sum of the areas of the trapezoids of trapezoidization T_1 and T_2 respectively. Then $\mathbf{A}_1 = \mathbf{A}_2$.*

In a straightforward manner we can generalize Lemma B.3.1 to apply to self-overlapping curves [**45, 51**].

A comment about computing volumes of polyhedra in three or higher dimensions is worthwhile. We point out that Equation B.2.1 and Observation B.3 can be generalized to apply to (not necessarily simple) polyhedra of any fixed dimension; see [**19, 47, 4**]. Furthermore the concept of self-overlapping polygons can also be generalized to self-overlapping polyhedra of any fixed dimension. It must therefore be the case that Lemma B.3.1 also generalizes to self-overlapping polyhedra of any fixed dimension.

A final nice point about Equations B.2.1 through B.2.4 is that the only operations used are multiplication, addition, subtraction, and at most one division (by two). Thus if the vertices of P and the endpoints of γ have only integer coordinates and all the x-coordinates, say, are a multiple of two then no floating point operations are required to compute the area of $P_{\overline{\gamma}}$. Under these conditions $\mathbf{A}(P_{\overline{\gamma}})$ is computed exactly using only basic integer arithmetic.

Bibliography

[1] Pankaj K. Agarwal and Micha Sharir. Circle shooting in a simple polygon. *J. Algorithms*, 14:69–87, 1993.

[2] A. Aggarwal, Leonidas J. Guibas, J. Saxe, and P. W. Shor. A linear-time algorithm for computing the Voronoi diagram of a convex polygon. *Discrete Comput. Geom.*, 4(6):591–604, 1989.

[3] A. Aggarwal, S. Moran, P. W. Shor, and Subhash Suri. Computing the minimum visible vertex distance between two polyg ons. In *Proc. 1st Workshop Algorithms Data Struct.*, volume 382 of *Lecture Notes Comput. Sci.*, pages 115–134. Springer-Verlag, 1989.

[4] Eugene L. Allgower. Computing volumes of polyhedra. *Math. Comput.*, 46:171–174, 1986.

[5] T. Asano, H. Imai, and A. Mukaiyama. A faster algorithm for finding a maximum weight independent set of a circle graph. In *Inform. Process. Soc. Japan SIGAL Report ALS-18*, pages 133–138, 1989.

[6] R. P. Boland and J. Urrutia. Polygon area problems. In *Proc. 12th Canad. Conf. Comput. Geom.*, pages 159–162, 2000.

[7] R. P. Boland and J. Urrutia. Finding the largest axis-aligned rectangle in a polygon in O(n log n) time. In *Proc. 13th Canad. Conf. Comput. Geom. Electronic version 8 pages*, pages 41–44 (Electronic version 8 pages), 2001.

[8] R. P. Boland and J. Urrutia. A simpler circular ray shooting algorithm. In *Proc. 13th Canad. Conf. Comput. Geom.*, pages 37–40 (Electronic version 7 pages), 2001.

[9] H. Booth. *Some fast Algorithms on Graphs and Trees.* Ph.D. thesis, technical report cs-tr-296-90, Dept. Comput. Sci., Prinston University., Prinston, NJ, 1990.

[10] M.P. Carmo. *Differential Geometry of Curves and Surfaces.* Prentice-Hall, 1996.

[11] R. C. Chang and H. S. Lee. Finding a maximum set of independent chords in a circle. *Inform. Process. Lett.*, 41(2):99–102, 1992.

[12] B. Chazelle, H. Edelsbrunner, M. Grigni, and et al. Ray shooting in polygons using geodesic triangulations. *Algorithmica*, 12(1):54–68, 1994.

[13] Bernard Chazelle. Triangulating a simple polygon in linear time. *Discrete Comput. Geom.*, 6(5):485–524, 1991.

[14] Bernard Chazelle and Leonidas J. Guibas. Visibility and intersection problems in plane geometry. In *Proc. 1st Annu. ACM Sympos. Comput. Geom.*, pages 135–146, 1985.

[15] Bernard Chazelle and Janet Incerpi. Triangulation and shape-complexity. *ACM Trans. Graph.*, 3(2):135–152, 1984.

[16] Siu-Wing Cheng, Otfried Cheong, Hazel Everett, and René van Oostrum. Hierarchical vertical decompositions, ray shooting, and circular arc queries in simple polygons. In *Proc. 15th Annu. ACM Sympos. Comput. Geom.*, pages 227–236, June 1999.

[17] F. Chin, J. Snoeyink, and C. A. Wang. Finding the medial axis of a simple polygon in linear time. *Discrete Comput. Geom.*, 21(3):405–420, 1999.

[18] Richard Cole, Martin Farach-Colton, Ramesh Hariharan, Teresa Przytycka, and Mikkel Thorup. An $O(n \log n)$ algorithm for the maximum agreement subtree problem for binary trees. *SIAM J. Comput.*, 30(5):1385–1404 (electronic), 2000.

[19] Felipe Contreras-Alcalá. Cutting polygons and a problem on illumination of stages. Masters thesis, Dept. Comput. Sci. University of Ottawa., Ottawa Ont, Canada, 1998.

[20] Jurek Czyzowicz, Felipe Contreras-Alcalá, and Jorge Urrutia. On measuring areas of polygons. In *Proceedings of the 10th Canadian Conference on Computational Geometry*, pages 56–57, Montréal, QC, Canada, August 1998. McGill University. Extended Abstract.

[21] Karen Daniels, Victor Milenkovic, and Dan Roth. Finding the largest area axis-parallel rectangle in a polygon. *Comput. Geom.*, 7(1-2):125–148, 1997. Fifth Canadian Conference on Computational Geometry (Waterloo, ON, 1993).

[22] M. Díaz and J. O'Rourke. Ham-sadwich sectioning of polygons. In *Proceedings of the 2th Canadian Conference on Computational Geometry*, pages 282–286, Ottawa, ON, Canada, August 1990. University of Ottawa.

[23] A. Fournier and D. Y. Montuno. Triangulating simple polygons and equivalent problems. *ACM Trans. Graph.*, 3(2):153–174, 1984.

[24] M. R. Garey, D. S. Johnson, F. P. Preparata, and Robert. E. Tarjan. Triangulating a simple polygon. *Inform. Process. Lett.*, 7(4):175–179, 1978.

[25] Leonidas Guibas, John Hershberger, Daniel Leven, Micha Sharir, and Robert E. Tarjan. Linear-time algorithms for visibility and shortest path problems inside triangulated simple polygons. *Algorithmica*, 2(2):209–233, 1987.

[26] Martin Held. Efficient and reliable triangulation of polygons. In *Proc. Comput. Graphics Internat. 1998*, pages 633–643, June 1998.

[27] John Hershberger and Subhash Suri. A pedestrian approach to ray shooting: shoot a ray, take a walk. *J. Algorithms*, 18(3):403–431, 1995. Fourth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA) (Austin, TX, 1993).

[28] John Hershberger and Subhash Suri. Finding a shortest diagonal of a simple polygon in linear time. *Comput. Geom. Theory Appl.*, 7:149–160, 1997.

[29] J. Iacono and S. Langerman. Volume queries in polyhedra. In *Japan Conference on Descrete and Computational Geometry 2000.*, pages 87–88, 2000.

[30] Y. Ke. An efficient algorithm for link-distance problems. In *Proc. 5th Annu. ACM Sympos. Comput. Geom.*, pages 69–78, 1989.

[31] J. M. Keil. *Decomposing Polygons into Simpler Components.* PhD thesis, Dept. Comput. Sci., Univ. Toronto, Toronto, ON, 1983.

[32] J. Mark Keil. Decomposing a polygon into simpler components. *SIAM J. Comput.*, 14(4):799–817, 1985.

[33] David. G. Kirkpatrick. Optimal search in planar subdivisions. *SIAM J. Comput.*, 12(1):28–35, 1983.

[34] Rolf Klein and Andrzej Lingas. A linear-time randomized algorithm for the bounded Voronoi diagram of a simple polygon. In *Proc. 9th Annu. ACM Sympos. Comput. Geom.*, pages 124–132, 1993.

[35] W. Klingenberg. *A Course in Differential Geometry*. Springer Verlag, 1996.

[36] W. Lenhart, R. Pollack, Jörg-Rüdiger Sack, R. Seidel, Micha Sharir, Subhash Suri, G. T. Toussaint, S. Whitesides, and C. K. Yap. Computing the link center of a simple polygon. In *Proc. 3rd Annu. ACM Sympos. Comput. Geom.*, pages 1–10, 1987.

[37] C. Levcopoulos. *Heuristics for Minimum Decompositions of Polygons*. Ph.D. thesis, Linköping Studies in Science and Technology, Linköping, Sweden, 1987. No. 155.

[38] Robin Liu and Simeon Ntafos. On decomposing polygons into uniformly monotone parts. *Inform. Process. Lett.*, 27(2):85–89, February 1988.

[39] Kurt Mehlhorn. *Data Structures and Algorithms 1: Sorting and Searching*, volume 1 of *EATCS Monographs on Theoretical Computer Science*. Springer-Verlag, Berlin/Heidelberg, Germany, 1981.

[40] E. A. Melissaratos and D. L. Souvaine. On solving geometric optimization problems using shortest paths. In *Proc. 6th Annu. ACM Sympos. Comput. Geom.*, pages 350–359, 1990.

[41] F. P. Preparata and M. I. Shamos. *Computational Geometry: An Introduction*. Springer-Verlag, 3rd edition, October 1990.

[42] S. Schuierer. An optimal data structure for shortest rectilinear path queries in a simple rectilinear polygon. *Internat. J. Comput. Geom. Appl.*, 6:205–226, 1996.

[43] M. I. Shamos. *Computational Geometry*. Ph.D. thesis, Dept. Comput. Sci., Yale Univ., New Haven, CT, 1978.

[44] T. C. Shermer. A linear time algorithm for bisecting a polygon. *Inform. Process. Lett.*, 41(1):135–140, 1992.

[45] P. W. Shor and C. J. Van Wyk. Detecting and decomposing self-overlapping curves. *Comput. Geom. Theory Appl.*, 2(1):31–50, 1992.

[46] Steven S. Skiena. Probing convex polygons with half-planes. *J. Algorithms*, 12(3):359–374, 1991.

[47] T. Speevak. An efficient algorithm for obtaining the volume of a special kind of pyramid and application to convex polyhedra. *Math. Comput.*, 46:531–536, 1986.

[48] Ivan Stojmenović. Bisections and ham-sandwich cuts of convex polygons and polyhedra. *Inform. Process. Lett.*, 38(1):15–21, 1991.

[49] Subhash Suri. *Minimum link paths in polygons and related problems*. Ph.D. thesis, Dept. Comput. Sci., Johns Hopkins Univ., Baltimore, MD, 1987.

[50] Subhash Suri. On some link distance problems in a simple polygon. *IEEE Trans. Robot. Autom.*, 6:108–113, 1990.

[51] H. Whitney. On regular closed curves in the plane. *Compositio Math.*, 4:276–284, 1937.

[52] C. K. Yap. An $O(n \log n)$ algorithm for the Voronoi diagram of a set of simple curve segments. *Discrete Comput. Geom.*, 2:365–393, 1987.

[53] Binhai Zhu. Computing the shortest diagonal of a monotone polygon in linear time. *Inform. Process. Lett.*, 42(6):303–307, 1992.