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.

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

UMI[®]

Geometric Placement Problems

By

Jason Boyd Morrison

A thesis submitted to
the Faculty of Graduate Studies and Research
in partial fulfilment of
the requirements for the degree of
Doctor of Philosophy

Ottawa-Carleton Institute for Computer Science
School of Computer Science
Carleton University
Ottawa, Ontario

May 2002

© Copyright

2002. Jason Boyd Morrison

**National Library
of Canada**

**Acquisitions and
Bibliographic Services**

**395 Wellington Street
Ottawa ON K1A 0N4
Canada**

**Bibliothèque nationale
du Canada**

**Acquisitions et
services bibliographiques**

**395, rue Wellington
Ottawa ON K1A 0N4
Canada**

Your file Votre référence

Our file Notre référence

The author has granted a non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of this thesis in microform, paper or electronic formats.

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de cette thèse sous la forme de microfiche/film, de reproduction sur papier ou sur format électronique.

The author retains ownership of the copyright in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

L'auteur conserve la propriété du droit d'auteur qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

0-612-71943-X

Canada

The undersigned hereby recommend to
the Faculty of Graduate Studies and Research
acceptance of the thesis,

Geometric Placement Problems

submitted by

Jason Boyd Morrison

F. Dehne
(Director, School of Computer Science)

J.-R. Sack
(Thesis Co-Supervisor)

P. Bose
(Thesis Co-Supervisor)

M. Dickerson
(External Examiner)

Carleton University

March 2002

Abstract

This thesis studies a variety of geometric placement problems. The main focus is to study the optimal placement of an object with respect to a fixed geometric structure. The objects studied in this thesis are grids, stars and polygons. The fixed geometric structures are point sets. The results in this thesis use an optimization criterion of occupancy. The allowable motions fall into two main categories: translations and rigid motions. We design and analyze algorithms for:

1. The translational placement of stars and grids over a fixed point set. In addition to the occupancy criterion we also show how to translate the star or grid such that the number of unoccupied cells is minimized.
2. The rigid motion placement of stars, grids and polygons over a fixed point set.

We study several variants of occupancy, such as k -occupancy, in order to find a placement of the object such that the number of cells containing at most k points is maximized. Finally, we present new solutions to the problem of determining the intersection of simple polygons. We provide two algorithmic solutions and show that the problem is **THREE-SUM HARD**.

Acknowledgments

No one finishes a dissertation alone. In my case I have been very fortunate to have a large number of people support me. It seems most appropriate to begin by thanking my parents who got me interested in doing graduate work and a PhD. They were always supportive and my mother has continued to be a role model even after my father's passing. The debt I owe them cannot be repaid, but I hope my respect and admiration can foot some of that bill.

Jörg Sack receives my thanks for sparking my interest in computational geometry, providing financial support when no one else would, and providing a place for me to work. Jörg has been a very supportive co-supervisor, allowing me to wander from my charted course and research that which became interesting. It is clear that without Jörg, I would never have done a PhD.

I want to thank Jit Bose for his energetic approach and positive outlook on my work. Without his reassurances and confidence in my work, I doubt I would have completed my degree as quickly as I did. As a co-supervisor, he played a crucial role by introducing new ideas and constantly challenging me with open problems.

From the faculty at Carleton I would like to thank Anil Maheshwari and Lyudmil Aleksandrov. I cannot overstate the help that Anil's open door policy, constant presence and desire to listen has been to me. His always friendly attitude made asking questions and discussing research a pleasure. Lyudmil I thank for keeping life in perspective, providing mathematical knowledge on demand and renewing my interest in chess.

Of the administration at Carleton, I only have kind words of thanks. Truly you ladies have managed, in spite of students and professors, to run an excellent department. Specifically I want to thank: Barbara, Joan, Judith, Linda, Marlene, Maureen, Nicki, Rosemary and Sandy for keeping my life in order and my mental health balanced during my time here.

Support in academia does not come solely from faculty or staff. Fellow students, support one another and I count myself blessed to have had the support of many a friend and fellow student. I have learned as much collectively from students as I have from faculty and I want to thank all those with whom I have interacted. Of all my friends, academic or otherwise I thank you for listening to me, putting up with my irritating opinions and over zealous defense of said opinions. I thank you for being there when I needed you and offering your opinions in light of my disposition to pick at them. Special mention goes to friends Alette, Anthony, Chris, Chris, Cindy, Darcy, Darrell, Kirsten, Jillian, Mike, Mike, Norbert, Pat, Paula and Rob who form a special part of my life and have helped me through this most difficult period.

To my sister Amy, I send my thanks and prayers. She has been a good friend and long-time confidante. I wish her the very best and thank her for always believing in me. To my in-laws, I send a special thanks as they are a comfort and wonderful addition to my family.

Of everyone who has helped me through the tough years of my PhD, I am most grateful to my wife, Michelle, who not only gives constant reassurance, but also proved that that obtaining a PhD is possible. She is my inspiration, passion and love. She is unfailing in helping me survive cancer, provides me with endless joy and is my greatest blessing in life.

Finally, I want to acknowledge the financial support of GEOIDE, MITACS, OGS and NSERC. Without their support, this thesis would have been strictly impossible.

Contents

Front Matter	i
Acknowledgments	i
Table of Contents	iii
List of Tables	vii
List of Figures	viii
List of Symbols	x
 1 Introduction	 1
1.1 Motivation	1
1.2 Research Topics	2
1.3 Thesis Summary	3
1.3.1 Chapter 2	4
1.3.2 Chapter 3	4
1.3.3 Chapter 4	5
1.3.4 Chapter 5	6
 2 Background and Related Research	 8
2.1 Geometric Properties & Techniques	9
2.1.1 Fundamentals	10
2.1.2 Dominance & Maximal Elements	11
2.1.3 Boundaries and Shape in Point Sets	13
2.1.4 Subdivisions, Partitions & Point Location	17

2.1.5	Arrangements	21
2.2	Single Object Placement Problems	26
2.2.1	Translation	26
2.2.2	Homothetic Copies	27
2.2.3	Rigid Motion	30
2.3	Multiple Object Placement Problems	32
2.3.1	Rectilinear Centre	33
2.3.2	Euclidean Centre	37
2.3.3	Line-Centre	38
2.3.4	An Alternative Classification to Centre Problems	39
2.4	Point Set Partitionings via Object Placement	40
2.4.1	Placement Evaluation	41
2.4.2	Dynamic Point Set Partition Maintenance	42
3	Star Placement by Translation	45
3.1	Introduction	45
3.2	k^{th} Maximal Regions	47
3.2.1	Properties	47
3.2.2	Construction	52
3.2.3	Representation and $\overline{M_k(P, \vec{C})}$	56
3.3	Star Translation Placement	58
3.3.1	A Maximal Region Approach	58
3.3.2	Solutions and Analyses	60
3.4	Common Intersection of Simple Polygons	62
3.4.1	Related Work	63
3.4.2	The Arrangement Algorithm	66
3.4.3	The Hidden Surface Algorithm	68
3.4.4	Three-Sum Hardness	71
3.5	Conclusions	73

4	Grid Placement by Translation	74
4.1	Introduction	74
4.2	Interval Data Structure	78
4.2.1	Query Operations	80
4.2.2	Update Operations	83
4.2.3	Re-balancing	88
4.3	Grid Translation Placement Solutions	89
4.3.1	Infinite Grids	89
4.3.2	Placing a Finite Grid	95
4.4	Conclusions	96
5	Rigid Motion Placement Problems	98
5.1	Introduction	99
5.1.1	Rotation Diagrams	100
5.1.2	Approach	103
5.2	Star Placement	105
5.2.1	Rigid Motion Placements	106
5.2.2	Rotation Diagrams for Stars	107
5.2.3	Algorithm and Diagram Complexity	114
5.3	Simple Polygon Placement	115
5.3.1	Introduction	115
5.3.2	Rotation Diagrams for Simple Polygons	116
5.3.3	Polygon Placement Algorithms	119
5.4	Grid Placement	121
5.4.1	Enumeration Algorithm	123
5.4.2	Rotation Diagrams for Grids	124
6	Summary & Conclusions	127
6.1	Summary	127
6.2	Future Work	131

Back Matter	133
Bibliography	133
Index	146

List of Tables

1.1	Summary of translational star placement results	5
1.2	Summary of translational grid placement results	6
2.1	Current best known results for the k -enclosing circle problem.	30
2.2	Summary of Nussbaum's g -piercing results	36
3.1	Summary of translational star placement results	73
4.1	Summary of grid-placement results	78
5.1	Summary of star rotation diagram equations	113
6.1	Summary of translational star placement results	128
6.2	Summary of translational grid placement results	129
6.3	Summary of deterministic results for rigid motion placement	130
6.4	Summary of expected results for rigid motion placement	130

List of Figures

2.1	Four maximal layers and the maximal outer hull	14
2.2	A convex hull of a point set	15
2.3	Quadtree example	19
2.4	k-d tree example	20
2.5	The upper envelope of an arrangement of lines in $\mathbb{R}^2$	22
2.6	The 3-level of an arrangement of lines in $\mathbb{R}^2$	23
2.7	The ≤ 3 -level of set of lines in $\mathbb{R}^2$	24
2.8	The vertical decomposition of a set of line segments	25
3.1	An example of a star	46
3.2	Two cones and a path within their union	49
3.3	Possible segments of $\partial M_k(P, \tilde{C})$ and $\tilde{C}$	51
3.4	Illustration of maximal boundary construction	53
3.5	A set of cones representing $\overline{M_k(P, \tilde{C})}$	56
3.6	A set of cones representing $M_k(P, \tilde{C})$	57
4.1	A canonical representation of intervals	76
4.2	$\text{Num}(v)$ and $\text{Int}(v)$ values for the example given in Figure 4.1.	77
4.3	An example of our data structure maintaining a function	80
4.4	The two cases in the proof of Claim 4.1.	86
4.5	A left rotation of a tree structure	88
4.6	Standard position of a grid and regions locating a point after translation	90

4.7	An example of occupation rectangles and a k -occupied set	93
5.1	Convex object and point set example for rigid motion placement . . .	101
5.2	Rotation diagram example	101
5.3	A star with 5 rays	106
5.4	Characterization of rigid motion star placement	108
5.5	Illustration of cases for rigid motion star placement: $\Psi_i = 0$	109
5.6	Illustration of cases for rigid motion star placement: $\Psi_i = \pi$	109
5.7	Illustration of cases for rigid motion star placement: $0 < \Psi_i < \pi$. . .	110
5.8	Illustration of cases for rigid motion star placement: $\pi < \Psi_i < 2\pi$. .	112
5.9	Star rotation diagram example	113
5.10	Splitting Δ with s_j	117
5.11	Splitting s_j with Δ	118
5.12	CH , Δ , $\mathcal{U}$ and the decomposition of $\mathcal{U}$	119
5.13	Computing the containing region for p_j and C	125

List of Symbols

The following section provides a list of the symbols used throughout this text. This reference tool is necessary as the notation used here is unavoidably different from the standard due to the breadth of topics considered and a desired consistency within the text.

Symbol	Meaning
α	The size of a generalized disc or an arbitrary angle.
β_j	The c.w. angle between positive vertical and the vector $p_j\tilde{p}_0$
δ	The difference between best possible output and actual output
Δ	A triangle
ϵ	An arbitrarily small constant <i>s.t.</i> $\epsilon > 0$
γ	A single geometric object
$\partial\gamma$	The boundary set of γ
Γ	A set of geometric objects
λ	The incremental size change of a dynamic object γ (e.g. a convex hull)
Ψ_i	The c.c.w. angle between r_1 and r_i
$\tau(\Gamma)$	The translation image of the objects of Γ
Θ	A numerical constant angle with radian units
$\varrho(\Gamma)$	The image of the homothetic copy of Γ
$\mathcal{A}(\Gamma)$	The arrangement of a set of geometric objects Γ
A	An arbitrary algorithm

Symbol	Meaning
B	A bounding box of a set of objects
c	An arbitrary constant scalar
c'	An arbitrary constant vector
c.w.	Clockwise direction
c.c.w.	Counter clockwise direction
C	An arbitrary cell or cone
$\vec{C}$	A cell or cone with fixed orientation Θ
$\vec{C}'$	A cell or cone with fixed orientation $\pi + \Theta$
$\partial\vec{C}$	The boundary of an oriented cone $\vec{C}$
d	The dimension of the space being considered
$\deg(u)$	The degree of the vertex u in a graph
e	An arbitrary edge of a polygon, arrangement or graph
$f(x)$	An arbitrary piecewise constant function mapping $\mathbb{R} \leftarrow \mathbb{R}$
g	The number of objects in Γ (i.e., $g = \Gamma $)
h	The height of a tree
H_k	The complexity of the k^{th} maximal layer
k	The size of the output or desired output
$K(v)$	$\max\{f(x) : x \in \text{INT}(v)\}$
l	The length of the bounding square of a planar pointset
L_k	The k^{th} Minkowsky distance $L_k(p, q) = (\sum_{1 \leq i \leq d} p_i - q_i ^p)^{1/p}$
$\mathcal{L}(\Gamma)$	The lower envelope of Γ
L	The positional location of p_0 in a rotation diagram
m	The size of γ (e.g. the number of vertices in a polygon)
$M_k(P, \vec{C})$	The k^{th} maximal region of P given $\vec{C}$
$\mathcal{M}$	A visibility map of a set of polygons in $\mathbb{R}^3$
$\mathcal{M}_i$	A visibility map of i polygons in $\mathbb{R}^3$

Symbol	Meaning
$\partial\mathcal{M}$	The boundary of the visibility map $\mathcal{M}$
$\overline{M_k(P, \vec{C})}$	$\mathbb{R}^2 - M_k(P, \vec{C})$
$\partial M_k(P, \vec{C})$	The k^{th} maximal layer of P given $\vec{C}$
n	The size of the set P (i.e., $n = P $)
$N_k(n)$	The number of k sets from a set of size n
p or q	A point in $\mathbb{R}^d$ (typically planar in this thesis)
p_i	The i^{th} point of P
p_{ij}	The j^{th} co-ordinate of p_i
$p \succ q$	Denotes p dominates q
P	A set of points $p \in P \{p \in \mathbb{R}^d\}$
P_i	The i^{th} sub-set of P
P_Θ	The unoriented Θ -maxima of the set P
PR	A problem
Q	A priority queue
$\mathcal{U}(\Gamma)$	The upper envelope of Γ
r_i	The i^{th} ray of the Star R or Cone C
R	A Star
$\mathcal{R}_i^j$	The region containing j occupation rectangles from P_i
root	The root of a tree
s_i	The i^{th} segment in an arrangement of segments
S_i	The unit square with lower right corner at p_i
T	A binary tree
u or v	Are arbitrary vertices of a graph
w	The width of an object
W	The perimeter of an object

Chapter 1

Introduction

This thesis focuses on the application of computational geometry, and geometric arrangements to the solution of a variety of geometric placement problems. These problems can be generalized as optimization problems of the form:

Object Placement Problem *Given a set of one or more movable geometric objects, place them optimally with respect to a fixed point set.*

This definition does not limit the movable sets of objects to a specific type or dimension. Neither does this general problem limit how the movable objects might be placed. Lastly the given generalization does not mention what is being optimized. Consequently, this generalization captures the spirit of an enormous set of problems. In the remainder of this chapter we present: the motivation for studying problems fitting this description; an overview of the specific problems studied and a summary of the contributions of this thesis.

1.1 Motivation

The subjects of combinatorial and geometric optimization intersect with a wide variety of topics. These topics include but are not limited to graph theory, calculus, abstract algebra, geometry, combinatorics and computational geometry. The focus

of this thesis is on stating and solving optimization problems involving placement of geometric objects. The motivations for studying these optimization problems, from the perspective of computational geometry, are easily broken into two groups: theoretical and practical. From a theoretical standpoint, the exploration of computational geometry methodologies and their relation to “classic” function, graph or geometric optimization problems gives new perspectives on both the problems and methods. These new perspectives drive scientific and mathematical discovery. Placed in the context of computer science we hope to gain a better understanding of the processes and limitations of geometric computation.

From a practical standpoint, a motivation for studying function, graph and geometric optimization problems is found in the first rule of business: “Location, Location, Location”. It is critical to analyze the placement of a new store with respect to its customers and competitors. In a similar vein it is crucial to analyze the location of city districts before placing a new retail area or a toxic waste dump. Placement of roads and airports to minimize noise near houses or travel time between local businesses can also be modelled as embedded graph, function or geometric optimization problems. Many of these problems can be modelled by geometric embeddings and useful approximate solutions can be achieved through the techniques of computational geometry.

Finally, these problems are studied using computational geometry because of its solid grounding in computer science. The field of computational geometry is relatively new [98] and its development is guided with algorithmics in mind. Thus its methods are well-suited for provably efficient problem solving.

1.2 Research Topics

As previously mentioned the Object Placement Problem encompasses a broad spectrum of problems. The research of this thesis focuses on solutions of several related problems within this spectrum. Three characteristics can be varied in the Object

Placement Problem to obtain different problems:

1. The number and kind of movable objects:
2. the kind of placements allowed and
3. the measure used to evaluate a given placement.

We focused the research of this thesis by choosing a few possibilities for each of these characteristics and looking at resulting combinations of problems. Thus, each of the problems involve the placement of exactly one movable object: either a star: rectangular grid or simple polygon. Given a fixed size movable object, only two types of placements are studied: translation or rigid motion. Lastly, each placement is evaluated by counting the size of all point-clusters formed by the placement of the movable object (e.g. two clusters are formed given a specific polygon placement: the points exterior to the polygon and the remaining points). Examples of the placement evaluations are maximizing or minimizing the number of clusters, minimizing the largest cluster, and maximizing or minimizing the number of clusters with at least k points.

Throughout this thesis we define placement evaluations with respect to the cells containing the partitions. If a cell contains a cluster of size k then we state that the cell is k occupied. We define occupancy of an object placement to be the number of clusters created. We further define the k -occupancy to be the number of clusters created that are at least of size k . Due to their number and complexity, we defer further discussion of different placement evaluations until Section 2.4. The summaries presented in the following section present results according to the type of occupancy information optimized.

1.3 Thesis Summary

This section is a summary of the content and contributions of this thesis. It is provided as a guide to the research contained within. Where applicable these summaries also

give bibliographic notes to associated publications and co-author's contributions. An effort has been made to keep the summaries self-contained. All necessary definitions are located within the text of the thesis and in the index at the end of the thesis.

1.3.1 Chapter 2

This chapter outlines the background for this research. The sections of the chapter are divided thematically into three categories. Section 2.1 presents an introduction to geometry, combinatorics and algorithmics sufficient for the comprehension of the thesis. Sections 2.2–2.3 contain the review of previous research on related problems. Lastly, Section 2.4 introduces point set partitioning, placement evaluation and dynamic maintenance of a partitioned point set. The occupancy terminology introduced in this section defines the majority of the problems solved in this thesis. This last section also represents the first substantive contribution of the thesis and is a crucial data structure for many of the solutions presented in this thesis. The occupancy terminology offered in this thesis is also presented in [27].

1.3.2 Chapter 3

This chapter studies the Object Placement Problem when the movable object is a star. A star is defined in this thesis to be a set of m cones each of size $\frac{2\pi}{m}$ whose union covers the plane and which share a single point. Specifically the star is of size m , the fixed point set contains n points and the placement is only done by translation. This implies that the star is given an orientation and that only placements of that orientation are considered. We initiate the study of these problems and are not aware of any literature directly related to the problems. We present solutions for a variety of problems all of which optimize the partitioning of the point set P relative to the placement of the star. The solutions summarized in Table 1.1 are an improvement of any trivial algorithm that must consider all $\Omega(m^2n^2)$ placements and their respective partitionings.

Objective	Time	Space
Maximize Occupancy	$O(nm \log nm + nm^3 \log m)$	$O(nm^3 \log m)$
Minimize/Maximize k -Occupancy	$O(nm \log nm + nm^3 \log m)$	$O(nm^3 \log m)$
Minimize Maximum Occupied Cone	$O((nm \log nm + nm^3 \log m) \log n)$	$O(nm^3 \log m)$
Maximize Minimum Occupance of c Cones	$O((nm \log nm + nm^3 \log m) \log n)$	$O(nm^3 \log m)$

Table 1.1: Summary of translational star placement results

Also included in the chapter are original definitions and results on k maximal regions and the arrangement of such regions (see Section 3.2). These results are necessary to the algorithms discussed above. This work also motivated the efficient construction of the common intersection of a set of regions and/or polygons. The results proved to be original and are discussed in Section 3.4. Specifically, we prove that the common intersection problem can be solved in time and space sensitive to the size of the arrangement of polygons. Smid [106] noted the relationship between common intersection and union and further hypothesized that a decision problem involving general common intersection is THREE-SUM HARD. A proof of this conjecture is provided in Section 3.4.4.

1.3.3 Chapter 4

This chapter presents research on the translational placement of regular rectangular grids. We initiate the study of these problems and are not aware of any literature directly related to the problems. As with the star translation problems of the previous chapter, only a specific orientation of the grid is allowed and each placement is evaluated based on the partitioning of the point set P by the grid G . The results of the different problems considered are given in Table 1.2. The brute force approach requires that $\Omega(n^2)$ different placements and their partitionings be evaluated for all

Objective	Time	Space
Maximize Occupancy	$O(n \log n)$	$O(n)$
Minimize/Maximize k -Occupancy	$O(kn \log n)$	$O(kn)$
Maximize k -Occupancy Finite Grids $ G = m$	$O(mkn \log n)$	$O(m_c kn)$
Minimize Maximum Occupied Cells	$O(kn \log n \log k)$	$O(kn)$
Maximize Minimum Occupance of c Cells	$O(kn \log n \log k)$	$O(kn)$

Table 1.2: Summary of translational grid placement results

of these problems. Our result for maximizing/minimizing the occupancy of the grid (*i.e.*, the number of cells containing points) is always faster than a brute force technique. The other results presented are sensitive to the largest number of points k contained in any cell.

In the development of this algorithm we also present a dynamic structure capable of maintaining the maximum or minimum stabbing depth of a set of intervals on the real line (see Section 4.2). The structure requires $O(n)$ space, and each update is completed in $O(\log n)$ time where n is the number of differing interval endpoints being stored. None of the endpoints or their number need to be known in advance. This interpretation of the analysis is more precise than the original interpretation presented in Bose *et al.* [25].

1.3.4 Chapter 5

The remaining chapter has the largest number of results of all chapters in the thesis. The chapter covers solutions to the rigid motion placement of stars, simple polygons and grids. The algorithms provided for these problems are all similar as each uses the concept of rotation diagrams originally introduced by Dickerson and Scharstein

[45]. Therefore it is appropriate that the chapter begins in Section 5.1.1 with a description of the generalities of our rotation diagrams in comparison with the original rotation diagram. Section 5.1.2 then presents a general method to solve rigid motion placement problems using rotation diagrams and the partition maintenance structure of Section 2.4.2.

The remainder of the chapter is split into three sections. Sections 5.2–5.4 each detail solutions for the rigid motion placement of stars, simple polygons and grids respectively. For stars and grids the same problems are explored as in the translation problems and as before we present results to new problems. For simple polygons we find the placement of maximal containment which improves on a previous result by Chazelle [31].

We begin by presenting an algorithm that evaluates each of the $\Omega(n^3m^2)$ possible star placements in $O(n^3m^2)$ expected time or $O(n^3m^2 \log nm)$ deterministic time and $O(n^2m^2)$ space. For simple polygons we provide two solutions both of which improve on the previous $O(n^4m^3 \log m)$ result of Chazelle [31]. The first result evaluates all possible $O(n^3m^3)$ placements in $O(n^3m^3)$ expected time or $O(n^3m^3 \log nm)$ deterministic time using $O(n^2m^2)$ space. The second result on polygons reduces the space requirement to $O(nm)$ and still solves the problem in $O(n^3m^3 \log nm)$ deterministic time. Finally we present two solutions for grids both of which evaluate all of the $\Omega(n^2 \sum_i \sum_{i>j} [d_{ij}])$ possible placements, where d_{ij} is the distance between points p_i and p_j and the grid is assumed to be comprised of unit squares. The first algorithm runs in $O(n^2 \log n \sum_i \sum_{i>j} [d_{ij}])$ time and $O(n)$ space. The second algorithm runs in $O((n^2 \sum_i \sum_{i>j} [d_{ij}]))$ expected time and $O(n^2 \sum_i \sum_{i>j} [d_{ij}] \log(n \sum_{i=\max} [d_{ij}]))$ deterministic time with $O(n \sum_{i=\max} [d_{ij}])$ space.

Chapter 2

Background and Related Research

This chapter introduces the terminology, conventions and research related to this thesis. Section 2.1 deals with the main geometric properties and techniques relevant to the thesis and presents most of the terminology and ideas used throughout. It also presents a small, but crucial overview of relevant background material and can be considered a reference for the remainder of the thesis.

Sections 2.2–2.3 deal with previous work on two subgroups of the Object Placement Problem. All of the problems reviewed in these sections and the problems studied in this thesis share the fact that the fixed objects form a planar point set. Section 2.2 contains a review of problems in which a single movable object is to be placed. The section further divides these problems according to the kind of placements possible.

Section 2.3 contains a review of problems in which more than one movable object is to be placed. The material is divided according to the type of objects to be placed.

The chapter culminates in Section 2.4 with a discussion of how various object placements are evaluated using point set partitionings. Section 2.4.2 also includes the description of a structure which maintains evaluation information about a dynamically changing partition in $O(1)$ time per partition change.

2.1 Geometric Properties & Techniques

Although geometry has been a basic subject of mathematics for centuries the field of computational geometry is recent and not as easily introduced. From a layman's perspective, geometry is the study of shapes. That is, the study of points, lines, curves, surfaces and solids. Often a layman will identify geometry with their adolescent use of compasses, protractors and rulers to solve high school math problems. This natural association of geometry with the use of compass and straight-edge is a good simile for the algorithms of computational geometry.

The use of compass and straight edge, like an algorithm in computational geometry, is often a set of well specified sub-tasks that are done in a specific order. Each sub-task is a geometric operation that is completed in a known amount of time (e.g. mark a circle with centre at point p). This set of sub-tasks form an algorithm that achieves its desired goal.

The main difference between a layman's view of geometry and a computational geometer's view lies within the geometer's knowledge of tools and techniques. Computational geometry provides a geometer with mathematical and algorithmic abstract representations useful in expressing a variety of geometric ideas. As a field we define computational geometry as the study of algorithms and data structures for combinatorial, topological, and metric problems concerning sets of geometric objects, typically in Euclidean space (this definition is expanded from "*The Free On-line Dictionary of Computing*").

This section begins with an introduction to the fundamental ideas, techniques and notation of computational geometry that are required for this thesis. The remainder of the section introduces mathematical and geometric ideas specific to the research being presented. First we introduce the concept of dominance and the idea of maximal elements. After presenting maximal layers we explore the concepts of boundary layers. This leads to a discussion about locating points, basic subdivisions and the partitioning of point sets. The section is concluded with a discussion of properties

and construction of arrangements of geometric objects.

2.1.1 Fundamentals

This section defines two notions necessary to the understanding of this thesis. The first is the notion of a geometric object. Throughout this thesis we are solving problems by finding an optimal placement of an object. The second notion is of a sweep which is crucial to several of our algorithms. Further definitions and concepts will be introduced throughout the text as needed.

A variety of objects are fundamental to computational geometry. In this thesis we refer to specific configurations of these fundamental objects as *geometric objects* or simply *objects* γ . Polygons, point sets and polygonal chains are among the most common objects in $\mathbb{R}^2$. Other examples include such objects as lines, line segments, rays, circles, arcs, hyper-planes and hyper-spheres. Hereafter such objects γ are referred to as if each is a set of points $P = \{p_1, p_2, \dots, p_n\}$. Although the set may be infinite, this is only mentioned if it has an affect on the topic being discussed. Should a specific definition be missing, details are available in one of the surveys on computational geometry [98, 44, 99].

A common technique in computational geometry is the *sweep*. Suppose a particular problem has many geometric objects as input and a possibly different set of objects to output. Each of these objects can in turn represent an infinite number of points. If the problem involves calculations with each of the objects then it is necessary, for computational tractability, to consider a finite subset of the input and output objects. Often this is done by “sweeping” through the input set in an ordered fashion, producing the output as the sweep progresses.

In $\mathbb{R}^2$ the sweep is known as a *plane sweep* and in $\mathbb{R}^3$ it is a *space sweep*. Given the wide application of this technique we present a brief description of a plane sweep and reference [98, p.10-11] for details. Suppose the problem is to calculate the area of a polygon γ with vertices $\{v_1, \dots, v_n\}$. Then during a plane sweep the vertices are

processed in the order of their x -coordinates. This is equivalent to moving a vertical “sweep line” from left to right stopping at each new vertex to calculate the latest amount of “swept” area. Each vertex is called an *event point* and the order of the event points is called an *event point schedule*. Innovation in many solutions lays in how the plane sweep is performed or how the problem is modified to make it feasible by a plane sweep.

2.1.2 Dominance & Maximal Elements

All of the placement problems dealt with in this thesis involve the placement of an object relative to a point set. Since the optimality of the placement is determined by the relative positions of points and objects it is necessary to exploit properties and structures of the point sets. This section covers the notions of dominance and maximal elements which are important structural characteristics and are used in the formulation of solutions to translation problems reported in this thesis.

When a set of points is given, the points’ relative positions define its structure. Consider a set of points P in $\mathbb{R}^d$. If $d = 1$ then there exists a natural definition of one point being greater than another (*i.e.*, $p_i > p_j$). If $p_i > p_j$ then we state that p_i *dominates* p_j . A natural extension of this one-dimensional definition is to apply the definition to each component of a multi-dimensional point. Thus if $p_i = (p_{i1}, \dots, p_{id})$ and $p_j = (p_{j1}, \dots, p_{jd})$ then p_i *dominates* p_j if and only if $p_{il} > p_{jl}$ for all l with $1 \leq l \leq d$. For the sake of brevity, from here on the phrase “ p dominates q ” is denoted as $p \succ q$. This definition of dominance leads to a variety of properties and problem transformations which are surveyed below.

This definition leads to such questions as “how many pairs of points (p, q) exist in P such that p dominates q ?” and “how quickly can we report all such dominance pairs?”. Before these questions are dealt with we make special note of the transitivity of the dominance relation: formally $\forall \{p, q, r\} \subseteq P, (p \succ q) \wedge (q \succ r) \rightarrow (p \succ r)$. Thus in any point set P there may exist dominance pairs such that $p \succ q$ with no point

$r \in P$ such that $p \succ r \succ q$. We refer to this kind of dominance as *direct dominance*, denote it as $p \succ q$ and call (p, q) a *direct dominance pair*. All other dominance pairs are termed *redundant dominance pairs* and together with direct dominance pairs they complete the transitive closure of direct dominance.

Point(s) that are not dominated by any other points are called *maximal elements* or *maxima* for short. In one dimension such a point is called the maximum point. Note that even in 1-D there may be many maximal elements. For example, the set $\{p_1 = (1), p_2 = (2), p_3 = (2), p_4 = (2)\}$ contains three maximal elements, each with the value (2).

Kung *et al.* [80], prove that the maximal elements of a d dimensional vector space can be calculated in $\Theta(n \log n)$ time for $d = \{2, 3\}$ and $O(n \log^{d-2} n)$ for $d \geq 4$. The approach taken to achieve these time bounds is through a divide and conquer algorithm, which is presented in [98, pp.161-165].

Several facts regarding the number of dominance pairs are evident once the definitions of dominance are scrutinized. First, there exists a placement of points [68] that gives $\Omega(n^2)$ direct dominance pairs. Bentley *et al.* [19] also prove that if all of the co-ordinates are chosen from an identical continuous distribution, then the expected number of maximal elements is $O(\log^{d-1} n)$. From this result Güting *et al.* [68] mention that one can expect $O(n \log^{d-1} n / (d-1)!)$ direct dominance pairs in the same situation.

In general it is possible to count or enumerate the number of dominance pairs or direct dominance pairs of a given point set. As the following survey shows, new solutions to these problems have been motivated by solving other related geometric problems. The first of these is the 2-D rectangle enclosure problem. In this problem the goal is to enumerate all of the rectangles that are strictly contained within another rectangle. Vaishnavi and Wood present a solution to this problem in [110] that uses range trees and segment trees. Their algorithm reports the k pairs of nested rectangles in $O(n \log^2 n + k)$ time and $O(n \log^2 n)$ space. Lee and Preparata [82] achieve the same time complexity as Vaishnavi and Wood and improve the space complexity of

this result to an optimal $O(n)$. This is achieved by transforming the rectangles into points in $\mathbb{R}^1$ and reporting all dominance pairs. The most recent work on this problem is by Gupta *et al.* [67] who give an $O(n \log n \log \log n + k \log \log n)$ time algorithm.

Güting *et al.* [68] look at the problem of reporting direct dominance pairs in 2 and 3 dimensions. They also give a full description of how to use direct dominance reporting to solve direct enclosure problems for intervals, homothetic triangles, and other convex polygons. Their results on direct dominance reporting give the k direct dominance pairs in $O(n \log n + k)$ time and $O(n)$ space for 2-D and $O((n + k) \log^2 n)$ time and $O((n + k) \log n)$ space for 3-D.

The question of dominance counting and enumerating appears more recently in the parallel programming literature. Atallah *et al.* [14] study the problem of counting the number of dominance pairs by 2-D points in set P_1 on points in set P_2 . If $n = \max(|P_1|, |P_2|)$ then their algorithm uses a CREW PRAM with $O(n)$ processors and $O(\log n)$ time. Goodrich studies the enumeration of these k dominance pairs in [61] using $O(\log n)$ time and $O(n/\log n + k)$ processors.

Datta *et al.* examine the direct dominance counting and enumeration problem in [41] on the CREW PRAM. Their algorithms count the direct dominance pairs using $O(\log n)$ time, $O(n)$ processors and $O(n \log n)$ space. The enumeration algorithm uses $O(\log n)$ time, $O(n + k/\log n)$ processors and $O(n \log n)$ space. The last work in this field is by Bauernöppel *et al.* [18] where the machine model is the EREW PRAM. That paper also provides a new sequential algorithm which is used as a basis for the parallel algorithm.

2.1.3 Boundaries and Shape in Point Sets

When the concept of maximal elements is examined carefully a distinct impression is left that no elements exist “beyond” the maximal elements. This is reinforced in 1-D where no points are greater than the maxima. This implies a boundary defined by the location of the points. In this section we explore this and other notions of a boundary.

Figure 2.1: Four maximal layers and the maximal outer hull

In doing so we also describe various definitions of where a point set “stops”, giving it an intrinsic shape.

Given a point set $P \in \mathbb{R}^d$, the maximal elements of P define a boundary of the set. Known as the *maximal layer*, the boundary is the sub-set of points in $\mathbb{R}^d$ that are not dominated by any points of P and do not dominate any maximal elements of P . Thus, the maximal layer separates the points in $\mathbb{R}^d$ that dominate maximal elements of P from those that are dominated by the maximal elements of P . Unfortunately the regions described are both infinite for any finite point set.

The basic properties of the problem are exhibited in the one dimensional version. Although a one-dimensional point set has a maximum and a minimum, the previous definition of dominance only allows for maxima. Since maxima vs. minima is an arbitrary decision we define the boundary layer called the *maximal outer hull* to be a combination of all possible maximal layers. This means that we define dominance with all combinations of $>$ and $<$ instead of just $>$. The result is 2^d maximal layers each definable by a specific dominance “direction” (see Figure 2.1). The *maximal outer hull* is defined by the boundary of the intersection of the complement of the maximal layers. The algorithm by Kung *et al.* [80], that calculates the maximal elements, can be used to create this outer layer in $O(n \log n)$ for 2-dimensions. We note here that each of the maximal regions is *orthogonally convex*. This means that the intersection of any vertical or horizontal line with the maximal region is a single line, ray or line segment.

Perhaps the best known structure is the *convex hull* of a point set. This boundary

Figure 2.2: The convex hull of P

layer of a planar point set P is the smallest convex polygon that encloses all points. As a structure it is easily explained in two dimensions using the following popular analogy: for each point imagine a nail is sticking out of the plane: now, take a rubber band and snap it around the outer-most points and the shape of the rubber band is the convex hull (see Figure 2.2). The points of P that are also on the convex hull are called *extreme points*. We note the convex hull defined here only includes the boundary but the convex hull is most often used to describe the boundary and its interior.

The convex hull is well studied and a complete listing here is inappropriate. Details on convex hulls can be found in [98, chap. 3-4]. The analysis of planar convex hull algorithms shows that its computation can be done in $\Theta(n \log n)$ time and linear space. The lower bound for this algorithm is proven by a linear time transformation to the one dimensional sorting lower bound.

The points interior to the *convex hull* of P are alternatively defined as all points q such that any line passing through q defines two closed half planes, both containing at least one point of P . Cole *et al.* [37] take this definition and expand it to the notion of k -hulls. A k -hull of a planar point set is the set of points q such that every line through q defines two closed half planes, both containing at least k points.

As Cole *et al.* point out: k -hulls are related to k -sets. A k -set is defined to be a subset of size k derived by the intersection of some half plane and the set P . The maximum complexity of a k -hull is understandably derived from the maximum number of such k -sets, hereafter denoted as $N_k(n)$. The paper also details an algorithm

to find the planar k -hull in $O(N_k(n) \log^2 k + n \log n)$ time and the three dimensional k -hull in $O(n^3 \log n)$ time.

A limitation of the three boundaries just described is their inherent convexity. The convex and k -hulls are convex and the maximal outer hull is the intersection of 2^d orthogonally convex regions. Since many point shapes are not perceived as convex by humans, the desire for shape recognition led to the definition of the α -hull. The original paper by Edelsbrunner *et al.* [51] conveys the definition of α -hulls, an algorithm for its calculation and describes a wide variety of commonalities to the Delaunay triangulation. For the sake of brevity we only comment on the basics of α -hulls as details are available in [51].

The definition of α -hulls is similar to k -hulls in that a single parameter (α and k resp.) determines the “shape” of the outer boundary. For α -hulls the adjustable parameter is $\alpha \in \mathbb{R}$ and can have the values $(-\infty, \infty)$. The definition of an α -hull of a point set P is the intersection of all closed discs with radius $1/\alpha$ that contain all the points of P . This definition of an α -hull is easily understood when $\alpha > 0$ as the usual definition of disc applies. The remaining values of α are only understood using a *generalized disc* of radius $1/\alpha$. If $\alpha = 0$ then the generalized disc is a half plane. And, if $\alpha < 0$ then the generalized disc is the complement of the disc with radius $1/\alpha$.

The α -hull has various properties depending on the value of α . First, if $\alpha = 0$ then we obtain the convex hull. If $\alpha > 0$ then the hull obtained is convex. As implied in the introduction, for negative values the hull obtained is not convex. Instead the boundary of the obtained region may have holes, be disconnected or both. This variety of properties make the boundary obtained by this algorithm an enduring part of shape description techniques.

A recent work in boundary and shape descriptors is the definition and calculation of the set of un-oriented Θ -maxima by Avis *et al.* [15]. The un-oriented Θ maxima $P_\Theta \subset P$ such that $q \in P_\Theta$ if and only if there exists any cone based at q with angle at least Θ , which contains no elements of P . When $\Theta = \pi$ the points returned are

the extreme elements. This implies a lower bound of $\Omega(n \log n)$ on the calculation of these points. For $\pi/2 < \Theta \leq \pi$ they give an algorithm with optimal time and space complexities. A variety of detailed techniques are also used to prove that an expected time bound of $O(n)$ using linear space is achievable when $\Theta = \pi/2$

Nevertheless, [15] only indicates how to find boundary points that are contained in the point set. It does not deal with how a closed boundary for the point set could be derived from this technique. Although several possibilities seem apparent it appears to be an open problem. Such a boundary could, like the α -hull, be non-convex and possibly even disconnected.

2.1.4 Subdivisions, Partitions & Point Location

The concept of a point set's boundary gives rise to the question: "What, if anything, lies within the boundary?". This leads to even more questions about their inherent structural properties and how to make queries on specific point sets.

One technique of creating a layered structure from a point set is often called "onion peeling" and has a long history. The idea begins by using any algorithm A that calculates $\partial P \subseteq P$ to be a boundary of the point set P (i.e., let $\partial P = A(P)$). The *onion peeling* algorithm produces a set of layers ∂P_i by defining $P_0 = P$, $\partial P_i = A(P_i)$ and $P_i = P_{i-1} - \partial P_{i-1}$. Intuitively this is identical to peeling an onion as each boundary layer is calculated using A that layer is removed and the process repeated until no points remain.

The first use of onion peeling is in the calculation of convex layers of a planar point set. One method of construction is to use any optimal convex hull algorithm to repeatedly find the i^{th} layer and then remove it this requires $O(n^2 \log n)$ time. Chazelle cites that there were at least two $O(n^2)$ results [63, 101] before the quadratic barrier was surmounted. The first sub-quadratic algorithm is given by Overmars and van Leeuwen [96] who use their dynamic maintenance of a convex hull to achieve $O(n \log^2 n)$ time. Chazelle improves this to an optimal $\Theta(n \log n)$ time in his paper

[32] by using a structure similar to Overmars and van Leeuwen's structure and by deleting all of the points from the outside-in. Chazelle's major contribution is in proving that the amortized cost is $O(\log n)$ per deletion resulting in the desired $\Theta(n \log n)$ algorithm.

Although Chazelle's algorithm is optimal in the input size it is not an output sensitive algorithm. Thus the algorithm is asymptotically optimal but performance on many point sets can be better. The best result of computing the first k layers is given by Nielson in [92] as $O(n \log H_k)$ where H_k is the complexity of the first k layers. Nielson not only presents a detailed history of the output-sensitive results for the convex layers, he also explains how to produce the first k maximal layers in $O(\log H_k)$.

The use of repeated boundaries to define the structure of a point set is not the only technique possible. Indeed a variety of concepts use partitioning of a point set and its space to determine the structural properties of the point set. Although partitioning the plane into convex layers or maximal layers is useful in some ways [32, 92], partitioning a space in other ways may be beneficial.

A classic example of a space partitioning structure is the Voronoi diagram. Given a point set P , the space is partitioned such that each partition is the sub-space closest to an individual point. The resulting partitioning is a nearest neighbour Voronoi diagram. Other variations include k -nearest neighbour and farthest neighbour. The variety of spaces and metrics studied are numerous and details are found in [95].

The quadtree is another structural partitioning of a point set. In the standard planar quadtree the point set is assumed to be bounded by a square with side length l . The partition is constructed by dividing the square into 4 equal size squares (see Figure 2.3). This is repeated until each square contains less than a pre-specified constant number of points. Points that fall on the vertical splitting line are assumed to be in one of the left quadrants and those on the horizontal line to the bottommost quadrants. This partitioning is generally stored as a tree. Each square is represented by a node in the tree. The nodes that are split have four children (one for each

Figure 2.3: a) Quadtree subdivision b) Resulting quadtree

quadrant). Nodes that are not split are leaf nodes and contain some point(s) in their square. Note that the size and location of each square can be calculated in an online fashion whenever the size and location of the original square is known. The simple structure and low overhead makes the quad tree and its higher dimensional variants very popular in practice.

Quadtrees have a long history and a wide variety of variants [44, sect. 14.4]. Each does allow easy neighbour finding and yields a structural representation of the point set. However, as pointed out by deBerg *et al.* [44, sect. 14.2], the depth of a quadtree can be very unbalanced. This is because a quadrant and one of its children might be split again and again until the last two points are finally separated. In fact the maximum tree height h for the definition presented above is $O(\log(\frac{l}{\min d_{ij}}) + \frac{3}{2})$ where $\min d_{ij}$ is the distance between the closest pair of points. Unfortunately, the theoretical construction time of the quadtree is also a problem as it is dependent on the depth of the tree.

The k-d tree is a partitioning structure that attempts to address the shortcomings of the quadtree. When the point set is planar, then it is initially divided in half by a vertical line passing through one of the points. The two sub-sets are then each partitioned in half again by two horizontal rays (see Figure 2.4). The subdivision continues to alternate direction until only a pre-specified constant number of points

Figure 2.4: a) k-d subdivision b) Resulting k-d tree

is in each region. Points on subdividing lines/rays are considered to be in the appropriate left or bottom region. The k-d tree is created by making a node for each partitioning ray. Each internal node contains the co-ordinate of its subdivision and leaf nodes only contain their internal point(s).

Since the tree is a balanced binary tree with n leaf nodes the overall size of a k-d tree is linear and the depth is $O(\log n)$. Furthermore each partitioning of the points is essentially a median operation requiring $O(n)$ time and the total construction time is bounded by $O(n \log n)$. This structure has two main criticisms when it is compared with the quadtree. First, the k-d tree requires an extra co-ordinate at each internal node of the tree. During a search of the tree this information must be retrieved and then the comparison can be made. This may take substantially longer than the calculation made by the quadtree and could require more space. The second criticism involves the nature of the obtained partitioning. Each subdivision in a quad tree is a square and in a k-d tree they have an arbitrary aspect ratio. As pointed out by Duncan *et al.* [49], this results in $\Omega(n)$ worst case time for search problems including nearest neighbour and approximate nearest neighbours. This worst case is derived from the fact that if the nearest neighbour is $d_{\min}$ away from the query point then all cells closer than $d_{\min}$ from the query point must be searched.

Several different space partitions try to address the problems of quadtrees and k-d trees; keeping their more desirable attributes [11, 49]. Many of these data structures,

along with k-d trees can be classified as binary space partition (BSP) trees. Following the definition of [49], each node in a BSP tree represents a convex region of space and all the points lying within it. Each leaf represents a constant number of points. Interior nodes have an associated hyperplane that partitions the region and the points into its two children. The balanced aspect ratio (BAR) tree achieves both a balanced aspect ratio of subdivisions and is a balanced tree. A formalized general framework for the structure and a detailed list of achieved goals is presented by Duncan *et al.* in [49].

2.1.5 Arrangements

The previous sections have been structured to lead into this discussion on arrangements. The computation and analysis of arrangements are crucial to most of the work found in this thesis. In short, we use the efficient computation and scanning of arrangements to decide which placements are distinct and which of those are optimal.

The partitioning of space is possible using the techniques described above: nevertheless, every placement of a set of objects within a space automatically defines a partitioning of that space. The partitioning of a space by the specific placement of a finite set of g objects $\Gamma = \{\gamma_1, \dots, \gamma_g\}$ is called an *arrangement* $\mathcal{A}(\Gamma)$. The topic of arrangements is well studied [8, 60, 65, 103]. Agarwal and Sharir's recent survey [8] contains over 360 references, has thirteen sections of definitions, results and an applications section covering: range searching, terrain visualization, transversals, robotics, molecular modelling and optimization. This section focuses on definitions, concepts and previous results required for this thesis.

Although the previous definition states that the arrangement $\mathcal{A}(\Gamma)$ is the partitioning of a space, we follow Agarwal and Sharir [8] in restricting our discussion to real *affine spaces* $\mathbb{R}^d$. An *affine space* $\mathbb{R}^d$ is a space such that a point and co-ordinate axis can be embedded such that every point in the space can be represented as a d tuple of its co-ordinates. As defined in [8] the arrangement, $\mathcal{A}(\Gamma)$, is the set of

Figure 2.5: The upper envelope of an arrangement of lines in $\mathbb{R}^2$

relatively open connected *faces* (or *cells*) of dimensions $0, \dots, d$ induced by Γ , where each face is a maximal connected set of points lying in the intersection of a fixed subset of Γ . We call a face with k -dimensions a k -*face*. Finally, low dimensional faces are alternatively known as vertices (0-faces), edges (1-faces) and cells (2-faces).

It is common to study the complexity of an entire arrangement or a k -face in an arrangement. Such a study is useful as it provides a lower bound for computing an arrangement or a face of an arrangement. The *complexity* of a k -face or k dimensional arrangement is the number of $(k - 1)$ -faces containing the k -face or all of the faces in the arrangement, respectively.

The upper and lower envelopes are sub-objects in an arrangement whose complexity is of significance. The *upper envelope* $\mathcal{U}(\Gamma) \subseteq \mathcal{A}(\Gamma)$, is the set of all points $p = (p_1, \dots, p_d) \in \Gamma$ such that there exists no point $q = (q_1, \dots, q_d) \in \Gamma$ with $q_d > p_d$. The *lower envelope* $\mathcal{L}(\Gamma)$ is defined similarly using $<$ instead of $>$. Note that both $\mathcal{U}(\Gamma)$ and $\mathcal{L}(\Gamma)$ are boundaries to Γ defined by a specific direction $(0, 0, \dots, 0, 1)$. An example of an upper envelope is given in Figure 2.5. Upper and lower envelopes respectively represent the maximum and minimum of a set of objects in $\mathbb{R}^d$. This is used in application of the parallel visibility model where a light source is placed at positive infinity of the d^{th} dimension. When this is done the upper envelope of the objects is the illuminated (visible) surface. Thus the complexity of these envelopes is therefore a lower bound on calculating such a surface.

A natural extension of the upper envelope is the k -level. Whereas the $\mathcal{U}(\Gamma)$ defines

Figure 2.6: The 3-level of an arrangement of lines in $\mathbb{R}^2$. Solid vertices are contained in the 3-level while hollow vertices are in the 2-level.

the surface with no points in $\mathcal{A}(\Gamma)$ above it. the k -level of $\mathcal{A}(\Gamma)$ defines the subset of Γ with exactly k points of Γ above it. Thus the upper envelope of an arrangement is the 0-level of the arrangement. Please note that papers differ on the definition of the k -level as some use below instead of above. For clarity an example is provided in Figure 2.6. Just as the envelopes define the boundaries, the k -levels define the structure of an arrangement. By definition, the k -level of an arrangement can be computed by onion peeling of upper envelopes. As Figure 2.6 shows, some care must be taken at the intersection of objects.

Although the structure of an arrangement $\mathcal{A}(\Gamma)$ is described by its k -levels, each k -level is only a surface of $\mathbb{R}^d$. Another extension of the upper envelope is the $\leq k$ -level of an arrangement $\mathcal{A}(\Gamma)$. Given any point $q \in \mathbb{R}^d$, let $P(q, \Gamma)$ be the set of all points $p \in \Gamma$ such that $q_d > p_d$. Then the $\leq k$ -level of an arrangement $\mathcal{A}(\Gamma)$ is the set of all points $q = (q_1, \dots, q_d) \in \mathbb{R}^d$ with $|P(q, \Gamma)| \leq k$. Informally the $\leq k$ -level is the subset of $\mathbb{R}^d$ with at most k points of Γ above it. Note that this definition varies in other texts, alternately using above or below. Figure 2.7 is given to eliminate possible confusion due to disparate definitions.

Although the internal structures of arrangements provide an important resource in geometry, the construction and representation of arrangements are crucial to the exploitation of this resource in computational geometry. In the remainder of this section we present the fundamental results known in arrangement construction. This

Figure 2.7: The ≤ 3 -level of set of lines in $\mathbb{R}^2$.

is important to the thesis as most of our solutions rely on the construction of arrangements of line segments or curve segments.

In the construction of an arrangement the first consideration must be to the representation of the arrangement. For example, suppose that in the construction of an arrangement of line segments we only store the pairwise intersections of line segments. Then to retrieve all of the 1-faces of the arrangements we have to split each segment into its corresponding 1-faces. Similarly to determine the number of 0-faces all pairwise intersections have to be compared to find identical vertices (*i.e.*, find vertices where more than two segments intersect). Such operations are costly and we have not even considered finding the 2-faces or cells of the arrangement.

In fact storing the intersections of the objects is often considered an insufficient representation of an arrangement. A more comprehensive representation is the *vertical decomposition* of an arrangement. In the vertical decomposition of a planar arrangement a vertical ray is extended from each vertex of the arrangement (see Fig. 2.8). These rays stop wherever they intersect some object γ thus creating a new vertex and subdividing both a 1-face and a 2-face. The resulting set of vertices and edges of the vertical decomposition are stored in a general graph data structure. Since the resulting cells of the arrangements are bounded by at most four edges each of these are also stored. More detail on such representations is available in Nievergelt and Widmayer [93].

As mentioned the calculation of the intersections between 1-faces of Γ does not

Figure 2.8: The vertical decomposition of a set of line segments

construct $\mathcal{A}(\Gamma)$. Nevertheless if Γ consists of line segments and there exist k such intersections then the construction of $\mathcal{A}(\Gamma)$ requires at least $\Omega(k)$ time. The first algorithm to construct $\mathcal{A}(\Gamma)$ in an output sensitive manner is presented by Bentley and Ottmann in [20]. Their suboptimal algorithm runs in $O(n \log n + k \log n)$ time and $O(n + k)$ space. The optimal deterministic result of Chazelle and Edelsbrunner [34] determines the vertical decomposition in $O(n \log n + k)$ time and $O(n + k)$ space. Two randomized results also exist that have optimal expected running times: Clarkson and Shor [36] and Mulmuley [89].

Fortunately the constructions of Bentley and Ottmann and Mulmuley also extend to arrangements of arbitrary curve segments [10, 100]. The model of computation must be sufficient to calculate the intersection of two curve segments in $O(1)$ time. Unfortunately, this does not lead to an optimal deterministic construction as Mulmuley's algorithm is randomized and the Bentley and Ottmann algorithm is sub-optimal.

This greatly affects the results of this thesis. Many of the solutions given in this thesis involve the construction of an arrangement of trigonometric and linear curve segments. Thus, a single solution may have different running times (expected and deterministic) depending on which of the two techniques is used to construct the arrangement.

The results of this thesis would benefit from an optimal deterministic result in the construction of trigonometric and linear curve segments. Unfortunately, the only deterministically optimal result on curve segments is limited to algebraic curves of bounded degree [10]. As this extended abstract by Amato *et al.* is limited to two pages it is not clear if the result can be extended to include trigonometric curves. Thus we assume that it is not possible until such a proof becomes available.

2.2 Single Object Placement Problems

The single object placement problems reviewed in this section take as input a specific “movable” geometric object γ and a point set P . The problem is to find one or more transformations of γ which satisfies or optimizes some criteria involving P . In this thesis we also refer to finding these transformations of γ as finding (possibly restricted) placements of γ .

Generally speaking, object placement problems may involve translation, scaling and or rotation of γ to find the optimal placement. To emphasize the kind of transformations possible we divide the research according to the transformations that are allowed in the given problem.

2.2.1 Translation

The following problems all have the property that an object and a specific orientation is included in the problem definition. The solution to the problem involves the placement of that object using only planar translation. A *planar translation* τ is an affine transformation of the plane that maps each point p in the plane to a point $\tau(p) = p + v$ for some fixed vector v . Given a translation τ and an object γ , $\tau(\gamma)$ is the object formed by the translation $\tau(p)$ of all points $p \in \gamma$ [17]. Note that rotation of the object is strictly not allowed in any of these problems.

An instance of these types of problems is the rectangle cover problem. In this

problem a rectangle's size and the location of a planar point set P is specified. The goal is to place the rectangle using translation, such that the number of points within the rectangle is maximized. It is usually assumed, without loss of generality, that the rectangle is axis parallel. Lee solves the axis parallel problem in [81] by transforming the problem into overlapping n rectangles and solving the maximum clique in the resulting overlaps. Lee's solution requires $O(n \log n + k)$ time where k is the maximum number of points included in the rectangle and $O(n \log n)$ space.

The convex polygon placement problems are similar to the rectangle cover problems. In fact the only difference is that instead of a rectangle the problem takes any convex polygon as an input. The first work on translation of a convex polygon is presented by Efrat *et al.* in [53]. In this work Efrat *et al.* use a plane sweep technique to achieve a time bound of $O(nk \log n \log m + m)$ time where m is the number of vertices of the convex polygon, k is the maximal number of points contained and n is the number of points in P . Nevertheless, Efrat *et al.* concentrate on another problem and do not give a detailed explanation of their result.

In [17] Barequet *et al.* improve on the original result twice. Their first algorithm uses a similar technique but improves the running time to $O(nk \log(nm) + m)$ with $O(n+m)$ space. The second algorithm reduces the number of candidate positions that have to be considered. This yields a more output sensitive run-time of $O(nk \log(mk) + m)$.

Another variation on the translation problem is the α -hull. In this problem the goal is to determine the region for which a generalized disc of radius $1/\alpha$ contains all of the points (see Section 2.1.3). Although the other problems are solved by a single placement, this problem requires the region of feasible placements.

2.2.2 Homothetic Copies

The following problems are given an object as part of the input and the solution is a homothetic copy of that object. A *homothetic copy* ρ is an affine transformation of

the plane that maps each point p in the plane to a point $g(p) = cp + v$ for some fixed scalar c and fixed vector v . Thus given the transform g and an object γ , $g(\gamma)$ is the homothetic copy of γ formed by the scaling and translation $g(p)$ of all points $p \in \gamma$. This type of solution does not allow for the rotation of the object.

The easiest of all the object placement problems are the two variations of the bounding box problem. In the less popular variation, the problem specifies a hyper-rectangle and the goal is to find the homothetic copy with the smallest c that encloses the points. This problem is identical to specifying an aspect ratio of the hyper-rectangle. An example of such a problem is the smallest enclosing square.

The more popular bounding box problem is to find the smallest hyper-rectangle that encloses the point set or object. This problem modifies the definition of homothetic copy from scalar multiplication cp to the dot product p with a fixed vector c' . It is easily proven that in this problem, "smallest" may be any monotonic measure of rectangles (e.g. smallest area or perimeter). This problem is used in Computer Graphics and Robotics applications to isolate pairs of objects that do not intersect. This is done by calculating which pairs of bounding boxes do not intersect. It is the speed of this calculation that makes it popular in practical application.

Another variation on bounding box problems is to find homothetic copies that enclose exactly k points. When this is combined with the fixed aspect ratio version of the problem the idea of smallest is again irrelevant. However, when this is combined with the mixed dimensional scaling the optimal result is different if perimeter or volume are minimized.

Perhaps the most famous of all object placement problems is the smallest enclosing circle problem. The history of this problem in facility location and computational geometry is quite long. The search for an efficient method of finding the solution to this problem is often credited to Sylvester in [107]. A full survey of the computational geometry related material is available in [98, p.255-259] and [73]. Note that the best algorithm is by Megiddo [86] who solves the problem in $O(n)$ time using linear programming. This is a landmark achievement as it had previously been conjectured

that the problem had a lower bound of $\Omega(n \log n)$.

Hurtado *et al.* [73] maintain that the first constrained smallest enclosing circle problem is introduced by Megiddo. Megiddo's solution to the unconstrained smallest enclosing circle [86] solves a series of sub-problems where the centre of the circle is required to be on a given line. Furthermore, Hurtado *et al.* [73] study the constraint, where the object must be located within m linear constraints (*i.e.*, within a convex polygon of size m). They provide an $O(n + m)$ time algorithm for this problem that follows the basic prune and search strategies of the linear time algorithms for the unconstrained problems.

Bose and Toussaint [26] study the constraint when the centre must be placed inside or on the boundary of an m -vertex polygon. They present an algorithm that runs in $O((n + m) \log(n + m))$ time for convex polygons and $O((n + m) \log(n + m) + k)$ time for non-convex polygons time, where k is the number of intersections between the polygon and the furthest point Voronoi diagram of the points. This technique was extended by Bose and Wang [28] to run in $O((n + m) \log m + m \log n)$ time. This last improvement is achieved by showing that not all k intersections need to be evaluated. Bose and Wang prove that there are $O(m)$ intersections that need to be calculated and that they can be evaluated in the desired time.

The smallest enclosing circle problem also has a variation which finds the smallest k enclosing circle. The current best solutions for this problem provide examples of the discrepancies that often exist between deterministic and randomized algorithmic complexities. For this particular problem there exist three simultaneous independent deterministic results [53, 57, 40] and one superior randomized result [85] (see Table 2.1). More details on randomized geometric optimization are available in Agarwal and Sen's survey [4].

The final homothetic problem involves any oriented convex polygon with m vertices. In this problem the goal is to find the smallest k -enclosing homothetic copy of the polygon. This problem is dealt with by Efrat *et al.* [53] as an extension to their algorithm for the k -enclosing circle. Efrat *et al.* solve this problem assuming that

Author	Ref	Time	Space
Efrat <i>et al.</i>	[53]	$O(nk \log^2 n)$	$O(n \log n)$
Eppstein & Erikson	[57]	$O(n \log n + kn \log k)$	$O(n \log n + kn + k^2 \log k)$
Datta <i>et al.</i>	[40]	$O(n \log n + kn \log k)$	$O(n + k^2 \log k)$
Matoušek	[85]	$O(n \log n + nk)$	$O(nk)$

Table 2.1: Current best known results for the k -enclosing circle problem.

$m = O(1)$ in $O(nk \log^2 n)$ time and $O(nk)$ space or in $O(nk \log^2 n \log \frac{n}{k})$ time and $O(n \log n)$ space. The same problem is solved by Barequet and Dickerson [16] using Translation Diagrams in a total of $O(n(n+m) \log(nm))$ time and $O(n(n+m))$ space.

2.2.3 Rigid Motion

Under the operations of rigid motion an object may rotate, translate or both but not scale. More formally, a planar *rigid motion* ρ is an affine transformation of the plane that preserves distance (therefore angles and area as well) [45]. A wide variety of problems fall into this category and address the question: “Can a specific object be moved using rigid motion such that it does not intersect any other object?”. These types of questions largely appear in motion planning [44, chap. 13]. For the sake of brevity and clarity only problems that are directly related to the Object Placement Problem and the problems studied in this thesis are discussed.

A natural extension of the convex polygon translation problem is to add rotation. Given a convex polygon with m vertices and a point set P with n points, the problem is to find a rigid motion that maximizes the number of points contained in the polygon. The first work on this problem is given by Chazelle [31]. Chazelle analyzes the number of combinatorially distinct placements (*cf.* stable placements) of two convex polygons such that one is interior to the other. Using these results Chazelle provides an algorithm which in $O(mn^2)$ time decides if a rigid motion placement of the convex polygon of size m exists that contains the entire point set. Chazelle notes that all

stable placements of an m vertex polygon on three points can be calculated in $O(m^3)$ time. This observation leads to a brute force algorithm for the rigid motion placement of a simple polygon on a point set requiring $O(n^4 m^3 \log m)$ time. This algorithm is dealt with in more detail in Section 5.3.

Dickerson and Scharstein [45] deal with the convex polygon case and comment on an improvement to the brute force technique which yields an $O(n^2 k^3 m^2 \log k)$ algorithm where k is the maximum number of points contained by the polygon. Their best result is the full solution given in [45] and requires $O(nk^2 m^2 \log(km))$ time and $O(m + n)$ space. This solution to the problem is achieved by the definition and use of the rotation diagram. Given a point $p \in P$ and a convex polygon γ , the *rotation diagram* $R_{\gamma,p}$ has a containing region for each $p_i \neq p \in P$. Each *containing region* is defined to be the set of rigid motions for which p is on $\partial\gamma$ and p_i is contained within γ . The definition of rotation diagrams and containing regions are mentioned in more detail in Section 5.1.1 as each proves useful in efficiently solving a wide variety of rigid motion placement problems.

The *width* of a planar point set P is the minimum distance between any two parallel lines γ_1 and γ_2 that contain P between them. The width of a point set can also be expressed as a rigid motion placement problem. This alternate formulation states that the goal of the *width* problem is to place a line γ such that the distance from the furthest point to the line is minimized. The solution to this formulation is possible using the solution to the first formulation. Essentially, γ is placed parallel to and equidistant from γ_1 and γ_2 . Using the rotating calipers method, γ_1 and γ_2 can be determined in $O(n \log n)$ time and linear space and by the problem transformation the same bounds apply to finding γ (see [109] for details on the rotating caliper method).

Offline dynamic maintenance of the width of a planar set of points is presented by Agarwal and Sharir in [5]. The solution they present runs in $O(n)$ space and $O(n \log^3 n)$ time to decide if after any operation the width of the set is less than a given value w .

The fully dynamic maintenance of the width of a planar set is achieved in a

deterministic result by Eppstein. In [56] Eppstein gives an algorithm which uses $O(\lambda n^\epsilon)$ time per update where λ is the amount of change in the convex hull during that update and ϵ is an arbitrarily small positive constant.

Eppstein's result is improved on by Chan [30] to $O(\sqrt{n} \log^3 n)$ amortized time per update and $O(\sqrt{n} \log^3 n)$ worst-case time per query. Chan does this by keeping a data structure allowing only deletions (using Eppstein's structure) and another structure allowing both insertions and deletions. When the second structure becomes too large (*i.e.*, updates are too expensive) the structures are merged together into a deletion only structure. The cost of this rebuilding is amortized over the number of insertions that occur to give the stated result. The significance of Chan's result is that it is the first "sublinear time bound for the general dynamic width problem in the plane" [30].

Another problem in rigid motion object placement is the Θ -maxima. In this problem the object is a cone with angle Θ . As we described in Section 2.1.3, the goal is to find all points $p \in P$ such that some Θ cone placed on p is empty. The resulting sub-set of P is also mentioned in Section 2.1.3.

2.3 Multiple Object Placement Problems

The following section details some previous results where g objects are optimally placed with respect to a single point set P . The goal is not to provide a comprehensive survey of these problems but rather to give the background to those problems that are most related to this thesis. Thus we focus on the problems known as the Centre Problems.

Formally, the goal of these Centre Problems is to place g objects amongst n points in $\mathbb{R}^d$ such that all points are enclosed by at least one object. In all of these Centre Problems the goal is to minimize the largest of these g objects. Primary categorization of the Centre Problems is by object type and then by the variety of measures used to determine each objects' size. Examples of such measures are diameter, area and perimeter of the object.

A *cluster* of points is defined to be all of the points enclosed by the same object. Objects used in centre problems have a well defined “Centre”. If a point is located in more than one object then it is defined to be in the cluster defined by the object whose centre it is closest to. Thus the clusters created by placing the objects form a partitioning of the point set. The measures used for optimality give precise definition to these vague concepts of “largest object” and “closest to centre”.

Further categorization of the Centre Problems is achieved by noting three constraint variations. The most common, the *unconstrained* problems, have no restrictions on the placement of the g objects. Less common and hence less studied are the *discrete* problems. In these problems the centres of all g objects must be located at g distinct points in P . A recent variation on discrete centre problems allows the centres of the g objects to be placed on g distinct points in a set of $m > n$ fixed points. Finally, the least studied of the Centre problems are the *constrained* problems. In these problems the centres are constrained to lie within a given non-discrete sub-space of $\mathbb{R}^d$. For example, the centre may have to be placed on a point within a collection of line-segments, polygons or polytopes. In the remainder of this thesis, any particular problem that does not mention a constraint is by default an unconstrained problem.

2.3.1 Rectilinear Centre

The Rectilinear Centre problems are all based on square or rectangle placements. The measure of a given square or rectangle can be: the longest edge L_∞ or the sum of the edge lengths L_1 . Thus the goal is to place homothetic copies of g hyper-rectangles to cover the point set. Variations on the placements are to allow the homothetic copies to rotate either independently for each shape or parallel to each other. In most cases the homothetic copies have different scaling factors (*i.e.*, rectangles are usually allowed to be of arbitrary aspect ratio).

Rectilinear 2-Centre

Given 2 rectangles that are planar, axis parallel with an L_1 or L_∞ metric then Hersberger and Suri's partitioning algorithm [71] solves the decision problem in $O(n \log n)$ time. That is, they can "decide" in $O(n \log n)$ time if the optimal partition is smaller than a given factor L , where smaller is measured by the metric of the problem. Decision problems are often used in multiple object placement problems as a sub-problem to the overall optimization.

Hershberger and Suri's result on the decision problem is improved to linear time by Drezner [47], Guha [64] and Rote and Woeginger. In the journal version of their paper [71] Hershberger and Suri note that Rote and Woeginger solved this problem in linear time but did not publish the result.

The weighted variation of the rectilinear problem assumes that the cost of an edge in one dimension is not equal to the cost in another. Thus each dimension has a weight associated with it. For the d -dimensional weighted case under L_∞ , Ko and Ching [79] prove a bound of $O(d^2n + d \log^* d)$ using prune and search. This result is improved upon by Bespamyatnikh and Kirkpatrick [22] who give a solution to the unweighted problem in $O(nd \log d)$.

For the constraints that the rectangles be planar, axis parallel and the rectangular metric is arbitrary, Glozman *et al.* use sorted matrices in their solution to obtain a $O(n \log n)$ time algorithm. These results use the sorted matrix search technique of Frederickson and Johnson and are expounded on in [59].

Jaromczyk and Kowaluk [75] give a solution to the problem where the rectangles are limited to being square but are allowed to be at any orientation if they are parallel. This solution takes $O(n^2)$ time.

Given a d -dimensional space $d \geq 2$ and axis parallel boxes Bespamyatnikh and Segal [24] give a solution with runtime $O(dn \log n + \binom{2d}{d} n^{d-1})$ that optimizes any monotone function on a box.

Discrete Rectilinear 2-Centre

Katz *et al.* [76] deal with three variations of the discrete rectilinear 2-centre problem. Common to each of their variations is the idea of using squares instead of rectangles and that the point set lies in the plane. The first variation forces the squares to be axis parallel and yields an $O(n \log^2 n)$ time algorithm requiring $O(n \log n)$ space. The second variation states that the 2 squares be parallel but may be of any orientation. They provide an algorithm that runs in $O(n^2 \log^4 n)$ time and requires $O(n^2)$ space. The final variation they consider allows the squares to rotate independently and runs in $O(n^3 \log^2 n)$ time and $O(n^2)$ space.

Bespamtyanikh and Segal [23] consider a generalization of the discrete 2-rectangle-centre problem. Here the goal is to cover a set P_1 while the centre of the rectangles must be points in P_2 . They proceed to show that optimizing the area of the maximum of the two axis parallel squares in the plane is possible in $O((n_1 + n_2) \log(n_1 + n_2))$ time. They also consider the case of optimizing the perimeter of the maximum of two axis parallel rectangles in the plane. For this problem they show a running time of $O(n_1 n_2 \log n_1 \log n_2)$.

Bespamyatnikh and Kirkpatrick [22] derive an $O(n \log^{d-2} n \log \log n + n \log n)$ time algorithm where $n = \max(n_1, n_2)$ and they assume the same constraint model as Bespamyatnikh and Segal.

Constrained Rectilinear 2-Centre

Bespamtyanikh and Kirkpatrick also consider other constrained rectilinear 2-centre problems in d dimensions. In the first variation the centres of the rectangles must reside on a set of m axis-parallel line segments. They show that the problem is solved in $O(N \log^{d-2} N \log \log N + N \log N)$ time where $N = \max(n, m)$. They also note that the result for axis-parallel line segments is extendible to polytopes with axis-parallel faces. In this last variation the centres are constrained to a set Γ of polytopes with axis-parallel faces where a total number of faces in all dimensions is m . This problem

Problem	Algorithmic Result
$\mathbb{R}^2$ 4-piercing	$O(n \log n)$
$\mathbb{R}^2$ 5-piercing	$O(n \log n)$
$\mathbb{R}^2$ g -piercing $g \geq 5$	$O(n^{g-4} + n \log n)$
$\mathbb{R}^2$ 4-centre	$O(n \log n)$
$\mathbb{R}^2$ 5-centre	$O(n \log n)$
$\mathbb{R}^2$ g -centre $g \geq 5$	$O(n^{g-4} + n \log n)$
$\mathbb{R}^3$ 2-piercing	$O(n)$
$\mathbb{R}^3$ 3-piercing	$O(n)$
$\mathbb{R}^3$ 4-piercing	$O(n^2)$
$\mathbb{R}^3$ 5-piercing	$O(n^3)$

Table 2.2: Summary of Nussbaum's g -piercing results

is also solved in $O(N \log^{d-2} N \log \log N + N \log N)$ time where $N = \max(n, m)$.

Rectilinear Piercing and Duality

Any discussion of the rectilinear piercing problems is incomplete without discussing their dual problems: the rectilinear centre problems. The rectilinear centre problems focus on placing g rectangles to cover a set of n points. Conversely the piercing problems place g points such that n rectangles each contain at least one point. The analogous decision problems decide, given a specific integer constant c and a set of n rectangles, if a set of $g = c$ points exist such that each rectangle contains a point without necessarily determining such a set of points.

The algorithms obtained to solve rectilinear piercing problems are important to the rectilinear centre problems because there is a reduction of axis-parallel rectilinear centre problems into axis parallel rectilinear piercing problems [94]. That same paper uses an in-depth case analysis by types of rectangle overlap to derive an algorithm (in $\mathbb{R}^2$ and $\mathbb{R}^3$) for the g axis parallel rectilinear piercing problems. A summary of results obtained is given in Table 2.2.

2.3.2 Euclidean Centre

Unlike the Rectilinear Centre problems, the Euclidean Centre problems have one metric (L_2). This means that the Euclidean Centre problem in $\mathbb{R}^d$ space is equivalent to placing g d -dimensional hyperspheres and subsequently minimizing the maximum sphere radius. The Euclidean 1-Centre problem is equivalent to the Smallest Enclosing Circle Problem and is discussed in Section 2.2.

Euclidean 2-Centre

Hershberger and Suri [71] consider the decision problem for planar Euclidean 2-centre problem and present an $O(n^2 \log n)$ time and $O(n)$ space algorithm. Agarwal and Sharir in [7] present a parametric search solution, that uses the previous result of Hershberger and Suri, and runs in $O(n^2 \log^3 n)$ time. In [77] Katz and Sharir also tackle the optimization problem but use expander graphs instead of parametric search. Each algorithm yields a time complexity of $O(n^2 \log^3 n)$ identical to Agarwal and Sharir's previous result. In a paper on dynamic low dimensional linear programming [54], Eppstein solves the planar Euclidean 2-centre problem using randomization. The result is an algorithm with an expected run time of $O(n^2 \log^2 n \log^2 \log n)$.

Sharir gives the first sub-quadratic algorithm for the planar 2-centre problem in [102]. His result of $O(n \log^9 n)$ time is achieved by using parametric search. Eppstein improves this result in [55] by using randomization to achieve $O(n \log^2 n)$ expected time. Chan in [29], provides a deterministic algorithm running in $O(n \log^2 n \log^2 \log n)$ time. This result is achieved by using the previous results of Sharir and Eppstein and using a better technique for when the two disks are overlapping. Chan notes that to obtain an $o(n \log^2 n)$ result for this problem a better algorithm must be developed for the case where two disks do not overlap.

Discrete Euclidean 2-Centre

The discrete version of this problem has considerably less completed research than the unconstrained version. Hershberger and Suri note that their algorithm in [71] works in $O(n^2)$ time and linear space for the decision problem.

The first sub-quadratic algorithm for this problem is presented in [9] by Agarwal, Sharir and Welzl. Their algorithm runs in $O(n^{\frac{4}{3}} \log^5 n)$ time and stands as the best known algorithm. This is in contrast to the unconstrained variation which has a time bound of $O(n \log^2 n)$.

c -Oriented Convex Polygons 2-Centre

One generalized variation on the 2-centre problem involves the covering of n points with two convex polygons whose boundaries are parallel to a set of fixed orientations of size c . For this problem Sharir and Welzl [105] give an $O(c^5 n \log n)$ time algorithm. They also comment that the 3-oriented triangles and 4-oriented trapezoid case can be solved in linear time.

2.3.3 Line-Centre

The difference between the Line-Centre problems and the Centre problem is that the objects are lines instead of circles. Thus the goal of the Line-Centre problems is to arrange g lines such the furthest distance from any customer to its nearest line is minimized. In the planar case the problem is thought of as the placement of g infinite strips such that the customers are covered and the maximal strip width is minimized. The Euclidean Line-Centre problem with 1 line is often called the width problem and is described in Section 2.2.3. What seems to have been overlooked by previous work in this area is the similarity of this problem and orientation independent rectangle centre problems.

Euclidean 2-Line-Centre

Agarwal and Sharir in [7] present a solution that runs in $O(n^2 \log^5 n)$ time and uses parametric search. This result is improved upon by Katz and Sharir in [77] by using expander graphs to achieve $O(n^2 \log^4 n)$. The polylog factor in the time complexity was lowered to the current best known by Jaromczyk and Kowaluk in [74]. This result is based on a new data structure called anchored upper and lower chains and results in $O(n^2 \log^2 n)$ time and $O(n^2)$ space. In [59] Glozman *et al.* also use the decision problem algorithm from [74]. By using a different approach to the search than Jaromczyk and Kowaluk, they are able to achieve the same bounds of $O(n^2 \log^2 n)$.

2.3.4 An Alternative Classification to Centre Problems

Throughout Sections 2.2-2.3 the centre problems considered are classified by the placement of one or more objects and the object and placement type. This classification is not the only classification possible but we feel that it best matches the description of the research found in the remaining chapters. Had we chosen another classification scheme then other centre problems would have appeared in the background survey.

One such alternative system of classification reformulates the problem's to focus on the metric used to measure the optimality of an object's placement. In this system the centre problems place the centre of the movable objects (usually one or more points) and then measure the fixed objects using the problems metric. The Euclidean centre problem (*i.e.*, minimum enclosing circle problem) finds a point placement such that the sum of distances in the L_2 metric is minimized. The rectilinear centre problem differs from the Euclidean by its use of the L_1 or L_∞ metrics.

Using this alternative system and placing a point relative to polygon there are two metrics and hence two centre problems which have been widely researched. The first metric is *link distance* which is the number of straight line segments lying within an object γ which required to connect two points p and q . The second metric is the

geodesic distance which is the shortest path between p and q which lies within an object γ .

The *link centre problem* finds the set of all points p within a given polygon γ such that the maximum link distance from p to all points $q \in \gamma$ is minimized. If γ contains n vertices and no holes then Djidjev *et al.*'s algorithm [46] finds the link centre in $O(n \log n)$ time. These results and others involving link distance are summarized in [83].

The *geodesic centre problem* finds the placement of point p within a given polygon γ such that the maximum geodesic distance from P to a point $q \in \gamma$ is minimized. It is known that such a placement is unique [13] and that it can be calculated in $O(n \log n)$ time [97]. More information on results involving geodesic distance are available in Mitchell [87, pp. 639–648].

2.4 Point Set Partitionings via Object Placement

Sections 2.2–2.3 have demonstrated a variety of related problems including different optimization criteria and movable objects. All of the problems studied in this thesis measure optimality of the placement of an object γ according to the partitioning of a given planar point set P . It is the goal of this section to introduce the terminology of partitioning a point set via object placement, describe various evaluations of such a placement and lastly describe a methodology of maintaining optimization information in a dynamic point set partitioning.

The problems in this thesis deal strictly with placing single objects γ to avoid large complexities of placing g objects. We have chosen to study stars and cones because although each star has a finite size descriptor it has many cells (cones) with which a point set is partitioned. For the remainder of this thesis we assume that a given placement partitions a point set into sub-sets P_i such that every point $p \in P_i$ is contained within the same cell of γ .

All of our measurements of a partitioning of P are based on the concept of *occupancy*. Given a placement of object γ we say that cell C_i of γ is *k-occupied* if it contains at least k points of P . A placement of object γ is said to have an *occupancy* of k if k of its cells contain one or more points. Lastly, a placement of object γ has a *k-occupancy* of c if at least c of its cells contain at least k points.

2.4.1 Placement Evaluation

A large number of different evaluations are possible using occupancy information. This section is a list of some such possibilities and the intuitive meanings of these evaluations.

The first set of problems is to find placements that minimize or maximize the occupancy of γ . This solution has as few or as many of the cells 1-occupied as possible. From a partitioning point of view this means minimizing or maximizing the number of sub-sets P_i in the partitioning of P .

Similarly a set of problems exist which minimize or maximize the *k-occupancy* of γ . A solution to such a problem has as few or as many *k-occupied* cells as possible. This is the same as minimizing or maximizing the number of sub-sets P_i such that $|P_i| \geq k$.

More complicated is the set of problems in which the goal is to find the smallest integer k such that there exists a placement γ with *k-occupancy* 0. Intuitively this means that we are finding the placement that minimizes the maximum number of points in any cell. We assume that this problem also requires a “witness” placement to be a part of the solution.

Lastly we comment on a problem which finds the largest integer k such that there exists a placement of γ with *k-occupancy* at least c where c is a given fixed integer. If this problem returns a placement satisfying this requirement then we obtain an optimal clustering of P . Essentially such a placement produces c clusters each of size at least k where k is maximized.

2.4.2 Dynamic Point Set Partition Maintenance

As we mention throughout this chapter, the problems studied in this thesis involve finding an optimal partitioning of a planar point set P by a placed object γ . Several solutions that we present involve examining all of the placements with different partitionings of P . The key to these solutions is to examine the placements/partitionings in such a way that there is a minimal difference between subsequent partitionings. The following section explains the data structure necessary to maintain and query this series of point set partitionings.

Consider the example problem of finding the placement of a grid γ such that the cell containing the most number of points, contains as few points as possible. We will assume that placement, in this context, is defined as all translations and rotation with different partitionings. The minimum difference between subsequent partitionings is when two partitionings of the same set differ by a single point being in a different cell.

The difference between two subsequent partitionings examined in our example is a single point p changing from cell i to cell j . Looking at the partitioning from the standpoint of sub-sets means that p will be removed from P_i and added to P_j . In order to maintain the partitioning our structure must support adding and removing points to/from point sets. We assume that the algorithm using the structure provides pointers to any point and sub-set involved in such an update.

The other functionality required for this structure is to aid in establishing which of the partitionings is optimal. In the example problem this means we must know the cell containing the most number of points and returning that number of points. From a sub-set perspective this is equivalent to finding $|P_{\max}|$ given that $P_{\max}$ is the largest sub-set in the current partitioning. This structure also provides such functionality as: number of sub-sets, minimum sub-set size and/or the number of sub-sets larger than a pre-specified constant c . From here on we only discuss the maintenance of the size of the maximum sub-set. The ideas for other measurements are not too different and

would only unnecessarily clutter the presentation.

With the desired functionality of the structure specified we turn our attention to the details of the structure itself. The structure Q is a priority queue and maintains every sub-set P_i in-order of their size $|P_i|$. As indicated, modifications to Q are calls to add or remove points from specified sub-sets. Specifically, two distinct operations exist on a sub-set P_i : 1. $\text{REMOVE}(P_i, p)$ and 2. $\text{ADD}(P_i, p)$. Since the addition or removal of p may cause a sub-set to be created or destroyed there are four specific cases that have to be dealt with:

1. $\text{REMOVE}(P_i, p)$

$P_i - p = \emptyset$ $\text{DELETE}(Q, P_i)$ This is the deletion of the sub-set P_i .

$P_i - p \neq \emptyset$ $\text{DECREMENT}(Q, P_i)$ This is the deletion of one element from the sub-set P_i . the partition still remains after the deletion.

2. $\text{ADD}(P_i, p)$

$P_i - p = \emptyset$ $\text{INSERT}(Q, P_i, |P_i| = 1)$ This is the creation of the new sub-set P_i . that is not in Q .

$P_i - p \neq \emptyset$ $\text{INCREMENT}(Q, P_i)$ This is the addition of one element to the sub-set P_i already existing in Q .

For now we are not concerned with the mechanisms that determine if either of the functions $\text{ADD}(P_i, p)$ or $\text{REMOVE}(P_i, p)$ need to be performed. Nor are we concerned with the mechanisms which actually maintain the sub-sets of the partitioning. The details of such maintenance is left to problem specific discussions. Instead we are interested in implementing the four operations described above and the $\text{GET-MAX}(Q)$ function.

The structure of Q is of the form of a sorted, doubly linked list with pointers to the head and tail of the list. Each element of this list represents a group of sub-sets

each of the same cardinality. Contained in each of these elements is the size of all of these sub-sets and a pointer to a doubly linked list of the sub-sets themselves.

Upon insertion of a new sub-set $\text{INSERT}(Q, P_i)$ the algorithm looks up the tail element of Q and determines if it contains sub-sets of size one. If not then the element is created with a pointer to an empty list of sub-sets and the algorithm continues as if it had found such an element. The sub-set P_i has its group pointer set to the element for sub-sets of size one. Then the sub-set p_i is finally added to the list of sets of size one.

The deletion of a sub-set $\text{DELETE}(Q, P_i)$ is handled in a reverse manner. The sub-set is removed from the list of sets of size one. This is easily accomplished in $O(1)$ time as the list is doubly linked. If the list is now empty then the element of Q for sub-sets of size one must be removed. This is also done in $O(1)$ time as the list of elements is also doubly linked.

The changing of a key is also easily done in constant time regardless of the call being $\text{INCREMENT}(Q, P_i)$ or $\text{DECREMENT}(Q, P_i)$. The only difference between these two cases is the direction in which P_i travels within Q . We describe the $\text{INCREMENT}(Q, P_i)$ operation here ($\text{DECREMENT}(Q, P_i)$ is similar and is not described here). First, the algorithm determines the element to the left of P_i 's element (using its group pointer). Then it determines if this element is exactly one greater in size than P_i 's element. If it isn't then a new element is created, of the appropriate size and inserted into its proper position. This is done in constant time and the algorithm continues as though it had found the correct element. The sub-set is inserted into its new group, deleted from its old group and has its group pointer reassigned to its new group. The old element's list is then checked to see if it is now empty and if so the element is removed from Q .

Since the total number of operations is constant for each of the operations then clearly a constant size change between two partitionings results in at most constant time to update Q .

Chapter 3

Star Placement by Translation

This chapter deals with the translational placement of a *star* object as defined in the following section. Most of this chapter deals with fundamental geometric and combinatoric theory on stars. We begin in Section 3.1 with a brief motivation for and a formal statement of the problems involving star translation. The study of arrangements of iso-oriented cones and maximal regions is especially relevant to these star problems. Section 3.2 reviews and defines these concepts, providing analyses of their properties and an algorithm for their calculation. Solutions are provided in Section 3.3 for the problems described in Section 3.1. Finally, Section 3.4 describes an offshoot result of this work involving simple polygon intersection.

3.1 Introduction

An *oriented star* R is defined as a set of m rays $\{r_1, \dots, r_m\}$ in $\mathbb{R}^2$, passing through a single point p such that ray r_1 makes a clockwise(c.w.) angle Θ with the vertical. We assume that the rays are regularly spaced around p . An example of an oriented star is given in Figure 3.1. It is crucial to the understanding of translational star placement problems and the solutions presented here to alternately define an oriented star as a family of m cones sharing the point p each with the angle $\frac{2\pi}{m}$ with appropriate

Figure 3.1: A star with $m = 6$ placed at p with orientation $\Theta = \frac{\pi}{4}$

orientations.

The goal in all of the Star Placement problems of this section and in Section 5.2 is to partition the point set. Here the partitioning is done by grouping all points that are contained in the same cone. Of course a little care must be taken to ensure that points on boundaries are consistently placed in only one such grouping. The placement of this structure allows the partition of the point set into several groups. This is rare in placement problems as most placement problems require the placement of multiple objects to obtain such a partitioning. As discussed in Section 2.3 most multiple object placement problems are difficult for even a constant number of objects and often NP-Hard for an arbitrary number.

The focus in this chapter is on solving five occupancy-optimized, partition-based placement problems. A placement of a star R is said to have an *occupancy* of k if at least k of its m cones contain a point of P . The placement of a star R has a *k-occupancy* of c if c cones of R contain at least k points. Finally we call a cone of R *k-occupied* if it contains at least k points of P .

Problem 3.1. *Given a planar point set P and a star R find a translation of R that maximizes the occupancy of R .*

Problem 3.2. *Given a planar point set P , a star R and a fixed integer k , find a translation of R that minimizes/maximizes the k -occupancy of R .*

Problem 3.3. *Given a planar point set P , a star R and a fixed integer k find a translation of R (if one exists) such that no cone of R is k -occupied.*

Problem 3.4. *Given a planar point set P and a star R , find the smallest integer k such that there exists a translation of R with k -occupancy 0.*

Problem 3.5. *Given a planar point set P , a star R and a fixed integer c , find the largest integer k such that there exists a translation of R with k -occupancy at least c .*

3.2 k^{th} Maximal Regions

Translational star placement problems impose a specific orientation Θ to the star and only allows the placement of R by choosing a point p to place its origin. Thus the star R is an oriented star and our analysis begins by considering the placement of R as a placement of a set of oriented cones.

Given a cone $\vec{C}$ with a specific orientation, define the *maximal region* to be all translational placements of $\vec{C}$ such that the cone is empty. Thus, given a cone with angle $\frac{\pi}{2}$, the maximal layer mentioned in Section 2.1.3 is the boundary for the maximal region. We extend this definition to the more general concept of cones with an internal angle less than π radians and that contain at most k points. The k^{th} *maximal region* $M_k(P, \vec{C})$ is the region of space where the cone $\vec{C}$ contains at most k points of P . We also define the k^{th} *maximal layer* to be the boundary of the k^{th} maximal region. A standard definition of the k^{th} maximal layer exists which comes from the idea of onion peeling. From here on, any maximal layer mentioned is the one defined by an arbitrarily oriented cone. For now, we assume that the k^{th} maximal region can be efficiently calculated. The details of this calculation follow.

3.2.1 Properties

The topic of this section is the properties of the k^{th} maximal region of a point set. These properties help establish characteristics and provide a construction of the k^{th}

maximal layer in the next section. Since the regions are dependent on an oriented cone $\vec{C}$ and the points in P , we assume that $\vec{C}$ has one ray oriented in the positive y axis and the second ray α radians c.w. from the first. Throughout the remainder of this section any reference to the k^{th} maximal region refers directly to the k^{th} maximal region with respect to the point set P , a given orientation Θ and an arbitrary value of k .

Property 3.1. *For a given point set P and an arbitrary integer $k < |P|$ the boundary of the k^{th} maximal region is not contained in the k^{th} maximal region.*

Proof. This property relies on the fact that points on the boundary of the cone are contained in the cone. The proof is by contradiction. Suppose that there exists a point p on the boundary of the k^{th} maximal region that is contained in the region. Then by definition, of the k^{th} maximal region, if the base of the cone $\vec{C}$ is placed at p then at most k elements are contained in the region. Consider an element q , which is outside the cone, but which is the first to be contained in the cone under any specified rigid translation. Since q is outside the cone it is some finite distance $2\epsilon > 0$ from the cone.

Consider a translation of the cone from p any distance $\delta \leq \epsilon$ in any direction. By definition of the point q the translated cone must either contain the same set of points or a sub-set of those points. This means that all of the new possible positions of the cone are within the k^{th} maximal region. This implies that p cannot be on the boundary of the k^{th} maximal region. This gives the necessary contradiction and proves the property. $\square$

Corollary 3.1. *For a given point set P , an arbitrary integer $k < |P|$ and any $\vec{C}$ placed on the k^{th} maximal layer, there exists at least one point q on $\partial\vec{C}$.*

Proof. This corollary follows directly from the proof of Property 3.1 $\square$

Property 3.2. *For a given point set P and an arbitrary integer $k < |P|$ the k^{th} maximal region of P is a connected region.*

Figure 3.2: Two cones and a path within their union

Proof. Again the proof is by contradiction. Suppose the k^{th} maximal region contains more than one connected component. Then there exist at least two distinct points q_1 and q_2 that are in different components of the k^{th} maximal region. Consider the two cones $\tilde{C}_1$ and $\tilde{C}_2$ that are based at q_1 and q_2 respectively. Each of these two cones contains at most k points and lies entirely within the k^{th} maximal region because any cone $\tilde{C}$ based within either $\tilde{C}_1$ and or $\tilde{C}_2$ contains at most k points. Without loss of generality we assume that the x co-ordinate of q_1 is smaller than the x co-ordinate of q_2 . Thus the y -axis parallel ray from $\tilde{C}_1$ either intersects the opposite ray from $\tilde{C}_2$ at the point q or $\tilde{C}_1 \subseteq \tilde{C}_2$ (see Figure 3.2).

Thus a continuous path exists from q_1 to q to q_2 , which stays within $\tilde{C}_1$, $\tilde{C}_2$ or both. This further implies that the k^{th} maximal region contains a continuous connected region, that includes q_1 and q_2 . This is a contradiction to the fact that q_1 and q_2 are in different components. $\square$

Property 3.3. *For a given point set P and an arbitrary integer $k < |P|$ the k^{th} maximal region of P has no holes (i.e., the complement of the region is connected).*

Proof. By Property 3.2 the k^{th} maximal layer must be defined by either a single continuous boundary or the k^{th} maximal region has holes. We offer a simple proof by contradiction that the k^{th} maximal region has no holes. If a hole did exist then a path would also exist which surrounds the hole and is entirely contained within the k^{th} maximal region. Within that hole the oriented dominance of every point must be greater than k . This infers that some point p_{path} is dominated by a point p_{hole} which

is dominated by more than k points. By transitivity of dominance, the path cannot be entirely within the k^{th} maximal region and must at some point be within the $k+1^{st}$ maximal region. This gives the contradiction required and proves Property 3.3. $\square$

Property 3.4. *For a given point set P and an arbitrary integer $k < |P|$, the boundary of the k^{th} maximal layer of P can be represented as a polygonal chain with segments parallel to the boundary of $\tilde{C}$.*

Proof. This property is proven directly by showing that any point q of the boundary can be enclosed by a circle of radius ϵ_q in which the boundary is comprised of two segments parallel to the boundary of $\tilde{C}$. Property 3.2 and 3.3 imply that $\partial M_k(P, \tilde{C})$ is a single connected boundary. Furthermore Corollary 3.1 states that if $\tilde{C}$ is placed anywhere on $\partial M_k(P, \tilde{C})$ then there exists at least one point $p \in P$ also on $\partial \tilde{C}$.

Let q be any point on $\partial M_k(P, \tilde{C})$. The placement of a cone $\tilde{C}$ at point q partitions the points of P into five distinct groups: $P_{int}(q)$ (the points strictly interior to the cone), $P_{r_1}(q)$ (the points lying on r_1 but not at q), $P_{r_2}(q)$ (the points lying on r_2 but not at q), $P_q(q)$ (any point lying at q) and $P_{ext}(q)$ (all remaining points). Given that q is on $\partial M_k(P, \tilde{C})$, Property 3.2 and Corollary 3.1 imply the following facts about the point sets:

$$|P_{int}(q)| \leq k \quad (3.1)$$

$$|P_{r_1}(q)| + |P_{r_2}(q)| + |P_q(q)| > 0 \quad (3.2)$$

We consider the $\partial M_k(P, \tilde{C})$ within the circle of radius ϵ_q which is centered at q . To calculate the value of ϵ_q a number of distances must be calculated: 1) for all points in $P_{int}(q) \cup P_{ext}(q)$ calculate the distance to their nearest ray r_1 or r_2 ; 2) for all points in $P_{r_1}(q)$ find the distance to r_2 ; and 3) for all points in $P_{r_2}(q)$ find the distance to r_3 . The value ϵ_q is equal to one half of the smallest of all of these distances.

The value of ϵ_q is chosen such that any translation of $\tilde{C}$ within the circle being considered results in minimal differences of the sub-sets defined by the cone. Essentially the points on either ray may stay on the same ray, be strictly within the interior

Figure 3.3: Possible segments of $\partial M_k(P, \vec{C})$ and $\vec{C}$

of the cone or be in the exterior of the cone. We use this restriction in conjunction with (3.1-3.2) to prove the result.

Suppose that q' is a point on $\partial M_k(P, \vec{C})$ and within ϵ_q of q . As with q this implies that:

$$|P_{int}(q')| \leq k \quad (3.3)$$

$$|P_{r_1}(q')| + |P_{r_2}(q')| + |P'_q(q')| > 0 \quad (3.4)$$

For (3.4) to be true the translation from q to q' must be parallel to either r_1 or r_2 . Thus within the circle the boundary must be comprised of straight line segments parallel to $\vec{C}$ with at least one endpoint on q (see Fig. 3.3).

Since (3.3-3.4) must be valid we can state the conditions under which each of the four segments are part of $\partial M_k(P, \vec{C})$:

$$s_1 \text{ if } (|P_{r_1}(q)| > 0) \wedge (|P_{r_1}(q)| + |P_{int}(q)| > k)$$

$$s_2 \text{ if } (|P_{r_2}(q)| > 0) \wedge (|P_{r_2}(q)| + |P_{int}(q)| > k)$$

$$s_3 \text{ if } (|P_{r_1}(q)| + |P_q(q)| > 0) \wedge (|P_{r_2}(q)| + |P_{int}(q)| \leq k)$$

$$s_4 \text{ if } (|P_{r_2}(q)| + |P_q(q)| > 0) \wedge (|P_{r_1}(q)| + |P_{int}(q)| \leq k)$$

We note that s_1 and s_4 are mutually exclusive as are s_2 and s_3 . This means that for any point q there are at most two segments within an ϵ_q radius. Thus the boundary is always parallel to $\partial\tilde{C}$ and this single boundary must be comprised of line segments (or rays) parallel to $\partial\tilde{C}$ and this proves Property 3.4. $\square$

Although we have proven that $\partial M_k(P, \tilde{C})$ is a polygonal chain, we have not noted that the segments for this chain can only come from the lines passing through the points of P parallel to $\partial\tilde{C}$. This fact follows immediately from Corollary 3.1 and Property 3.4 and is important to the construction of $\partial M_k(P, \tilde{C})$.

3.2.2 Construction

Properties 3.1–3.4 provide the basis for the main theorem on k maximal layers. Theorem 3.1. Before presenting the Theorem we present an algorithm for calculating the boundary of the k^{th} maximal region (*i.e.*, the k^{th} maximal layer hereafter denoted $\partial M_k(P, \tilde{C})$). The main idea of the algorithm is to construct the boundary of the region from left to right across the plane.

The preprocessing for this algorithm requires that the points be sorted twice, once in the direction perpendicular to r_1 and again perpendicular to r_2 . We subsequently call these the r_1 and r_2 orderings respectively and assume that ties in one ordering are broken using the other ordering. Without loss of generality we assume that the r_1 direction is vertical and henceforth use left and right to refer to lower and higher in the r_1 ordering.

The points are processed in a plane sweep using the r_2 ordering from maximum to minimum. After each event point p_i , the boundary of the k^{th} maximal region, $\partial M_k(n)$, is updated farther along the r_1 or r_2 ordering. A queue is also maintained that holds $k + 1$ of the points contained in a cone $\tilde{C}$ placed at the rightmost event point of $\partial M_k(n)$ (the event point with least r_2 ordering breaks ties). In the degenerate cases where more than $k + 1$ elements are contained in $\tilde{C}$ this queue contains all points interior to the cone and those points, on the rays and closest to the vertex of $\tilde{C}$. This

Figure 3.4: a) initializing $\partial M_k(P, \vec{C})$ b) 1st update of $\partial M_k(P, \vec{C})$ c) post-processing of $\partial M_k(P, \vec{C})$ Maximal region boundary is shown for $k = 2$ and oriented cone as displayed. updates to $\partial M_k(P, \vec{C})$ are shown with thick dotted segments.

implies the algorithm's invariant: that there are more than k points within any cone placed on the boundary constructed and at most k of those points are strictly interior.

The algorithm begins by initializing the queue with the first $k + 1$ event points. The boundary $\partial M_k(P, \vec{C})$ is updated by adding the ray of the line passing through the $k + 1^{st}$ event point and parallel to r_2 from $-\infty$ in the x -direction to the lowest r_1 ordering in the queue (See Fig. 3.4a). The point q is set to the end point of added ray.

When a new event point p_{bottom} is swept over it is inserted into the priority queue and one of the following cases is observed:

1. If the r_1 ordering of the inserted point is strictly less than all of the other points. then it is removed from the queue and $\partial M_k(P, \vec{C})$ remains the same.
2. Otherwise the point p_{left} in the queue with minimum r_1 ordering should be deleted from the queue (in the case of ties choose the maximum r_2 ordering). The boundary $\partial M_k(P, \vec{C})$ is subsequently updated as follows:

- (a) If the r_2 ordering of q and p_{left} are different then add the vertical segment

from q to the line parallel to r_2 passing through p_{left} . Set q to this new endpoint.

- (b) If the r_1 ordering of q and p_{left} are different then add a segment parallel to r_2 from q to the line parallel to r_1 passing through p_{left} . Set q to this new endpoint.

When the sweep is completed $\partial M_k(P, \vec{C})$ has to be updated with the vertical ray from q to $-\infty$ in the y direction. The correctness of the segments and rays added is only valid if the interior angle of $\vec{C}$ is less than π . This ensures that any cone placed at an unconsidered event point lies entirely to the right of the incomplete $\partial M_k(P, \vec{C})$.

Theorem 3.1. *For a given point set P an arbitrary integer $k < |P|$, and an oriented cone $\vec{C}$ the k^{th} maximal region of P with respect to $\vec{C}$ is computable in $O(n \log n)$ time and $O(n)$ space.*

Proof. The proof of Theorem 3.1 is in two stages: first by showing that the algorithm maintains the invariant and second by proving that by maintaining the invariant the boundary produced by the algorithm is the boundary of the desired region.

The invariant states that there are more than k points within a cone placed anywhere along the constructed boundary and at most k of those points are strictly interior. We prove this invariant by induction on the number of event points generating segments in the boundary.

As a base case we examine the insertion of the initial ray. By definition of this ray a cone $\vec{C}$ placed on the ray will contain $k + 1$ points at least one of which is on r_2 .

Assume that the invariant is maintained after the i^{th} event point generating a segment in the boundary. Now consider the $(i+1)^{st}$ such event point. By the assumption the boundary constructed by the first i “generating” event points is still valid. This is proven because all event points added after the i^{th} “generating” event point have a lesser r_2 ordering than the boundary constructed. Thus no cone on the boundary can

contain any more or less points than when the segments were added. Thus it only remains to prove that the segments added by the $i + 1^{st}$ “generating” event point are also valid for the assumption.

If a vertical segment is added to the boundary (as described in 2a) then any cone on this segment contains at least all of the points in the queue with the possible exception of the point with lowest r_2 ordering. In addition such a cone would contain the point p_{left} . Since p_{left} is on the boundary and the remaining i points of the queue can be in the interior the invariant must be true.

If the horizontal segment is added to the boundary (as described in 2b) then any cone on this segment contains at least all of the points in the queue and possibly p_{left} . The point with the lowest r_2 ordering in the queue is on the boundary of any cone on the segment. Since the cone always has at least a single point on the boundary and the remaining k points can be in the interior the invariant must be true. This proves the inductive case and thereby proves that the invariant is kept by the algorithm.

All that remains is to show that by maintaining the invariant the algorithm constructs the correct boundary. This proof is direct as the constructed boundary has properties identical to the desired boundary as proven by the invariant. Since Properties 3.2–3.3 state that there is only one, connected boundary then the constructed boundary is correct.

Since the algorithm described produces the correct region we can establish maximum bounds required to construct the region by analyzing the algorithm complexity. The two sorting phases require $O(n \log n)$ time. The $O(n)$ insertions and deletions in the priority queue take a total of $O(n \log k)$ time. The size of $\partial M_k(P, \vec{C})$ is $O(n)$ because at most two segments are added for each deletion and at most $O(n)$ deletions are performed.

□

We note here that this bound is tight for $k = 1$, as the maximal layer for a cone with angle $\frac{\pi}{2}$ requires $\Omega(n \log n)$ time (see Section 2.1.3).

Figure 3.5: A set of cones representing $\overline{M_k(P, \vec{C})}$

3.2.3 Representation and $\overline{M_k(P, \vec{C})}$

This last section on k^{th} maximal regions presents another representation of $M_k(P, \vec{C})$ and describes properties of $\overline{M_k(P, \vec{C})}$. The k^{th} maximal region divides the plane into two regions: the region $(M_k(P, \vec{C}))$ and its complement $(\overline{M_k(P, \vec{C})} = \mathbb{R}^2 - M_k(P, \vec{C}))$. The definition of $M_k(P, \vec{C})$ is the set of placements of $\vec{C}$ such that it contains at most k points of P . Thus any other placement of $\vec{C}$ must contain at least $k + 1$ points of P . Both of these regions are used in our solutions to various translational star placement problems (see Section 3.3).

Although $\overline{M_k(P, \vec{C})}$ can be defined by a polygonal chain another representation is used throughout Section 3.3. Briefly, $\overline{M_k(P, \vec{C})}$ is represented as the union of a set of $O(n)$ cones. The proof of correctness of this representation follows directly from Property 3.4 and Theorem 3.1. Formally, $\overline{M_k(P, \vec{C})} = \{\vec{C}'_1 \cup \dots \cup \vec{C}'_{O(n)}\}$ where $\vec{C}'_i$ is the cone $\vec{C}$ rotated by π radians. In fact using the construction of $\partial M_k(P, \vec{C})$ we note that the i^{th} addition to $\partial M_k(P, \vec{C})$ occurs at the origin of $\vec{C}'_i$ (see Fig. 3.5). Since $\partial M_k(P, \vec{C})$ can be calculated in $O(n \log n)$ time and $O(n)$ space, the cone representation of $\overline{M_k(P, \vec{C})}$ can be computed in the same time.

We now claim that $M_k(P, \vec{C})$ can also be represented by $O(n)$ cones. To prove this we provide a construction using the cone representation of $\overline{M_k(P, \vec{C})}$ and then prove that this construction is correct.

Figure 3.6: A set of cones representing $M_k(P, \vec{C})$

As before, represent $\overline{M_k(P, \vec{C})}$ as the union of $O(n)$ cones but this time the boundary of the union is not included (see Property 3.1). For each pair of cones $(\vec{C}_i', \vec{C}_{i+1}')$ we create a single cone $\vec{C}_i$ based at the intersection q_i of $\partial\vec{C}_i'$ and $\partial\vec{C}_{i+1}'$. Finally, we create two sufficiently large cones and place their origins on the origins of $\vec{C}_1'$ and $\vec{C}_{\max}'$ respectively. The orientation of the last two cones aligns their interiors inside $M_k(P, \vec{C})$ and one ray on $\partial M_k(P, \vec{C})$ (see Fig. 3.6).

It remains to be proven that this construction covers $M_k(P, \vec{C})$ and only $M_k(P, \vec{C})$ and that a construction of the final two cones exists. By its construction, the union of the cones has the same boundary as $M_k(P, \vec{C})$ and does not intersect $\overline{M_k(P, \vec{C})}$ except at the boundary. Thus it remains to be shown that all points of $M_k(P, \vec{C})$ are interior to at least one cone:

Proof. Let $p = (p_x, p_y)$ be interior to $M_k(P, \vec{C})$. Then we consider three cases: i) p_x is between the x co-ordinates of the base of two cones $\vec{C}_i'$ and $\vec{C}_{i+1}'$; ii) p_x is less than the x co-ordinate of the base of $\vec{C}_1'$; or iii) p_x is greater than the x co-ordinate of the base of $\vec{C}_{\max}'$.

In case one the point is included in the cone $\vec{C}_i$ by the definition of $\vec{C}_i$. The proof is concluded by noting that the remaining two cases are contained by halfplanes (*i.e.*, cones with angle π) with boundaries on the first and last rays of $\partial M_k(P, \vec{C})$. $\square$

3.3 Star Translation Placement

The first observation about Problems 3.1–3.5 is that there exists a simple brute force approach to each of the problems. Define the plane $\mathbb{R}^2$ to be all translational placements of an oriented star R . Divide the plane into the sets of placements that have the same partitioning of P . This is achieved by placing a star R on each point in P and viewing the resulting arrangement of rays. The correctness of this technique is guaranteed by Corollary 3.1. The resulting arrangement has a complexity of $O(n^2m^2)$ and alternatively represents the placement of any two points on up to $O(m^2)$ pairs of rays.

It is possible [31] to examine each of these placements and determine the measure of “fitness” of each placement. Such a brute force technique suffers from the necessity of examining $\Omega(n^2m^2)$ different placements. In the following section we achieve a better result by using geometric properties of k maximal regions. We begin by presenting a general technique for solving translational placement problems of stars (Section 3.3.1). In Section 3.3.2 we apply this technique to Problems 3.1–3.5 thus providing algorithmic solutions for each problem.

3.3.1 A Maximal Region Approach

The star translation problems of this thesis are occupancy based problems. Our approach to these problems is to construct the complement of the k^{th} maximal region for each of the m cones in R . Once the regions are calculated they are all overlayed and the resulting arrangement is examined. From this arrangement it is possible to tell exactly which cones contain strictly more than k points. Depending on the problem either one or more values of k may have to be examined. The remainder of this section explains the details of the approach outlined above for an arbitrary value k . Problems requiring the evaluation of one or more values of k are dealt with in Section 3.3.2.

First, each of the m regions $\overline{M_k(P, \vec{C}_i)}$ are calculated using the algorithm described

in Section 3.2.2. This requires $O(mn \log n)$ time and $O(nm)$ space. Next we must construct the arrangement of these regions. By Property 3.4 and Theorem 3.1 this arrangement may be constructed from $O(nm)$ line segments.

The representation of the regions as cones as described in Section 3.2.3 leads to a bound on the size complexity of this arrangement. This representation of each region $\overline{M_k(P, \vec{C}_i)}$ as a set of $O(n)$ cones gives us a representation of the arrangement of m regions as an arrangement of $O(nm)$ cones. The cones in question are all identical in size having a base angle of $\frac{2\pi}{m}$ radians. A set of cones with a minimum base angle of δ is defined to be δ fat [52]. Efrat *et al.* [52] prove that the complexity of the union of $O(nm)$ δ -fat cones is loosely bounded by $O(\frac{nm}{\delta^2} \log(\frac{1}{\delta}))$. The same paper also provides a construction of the union using Chazelle and Edelsbrunner's optimal line segment algorithm [34]. The arrangement of m maximal layers can be calculated in an identical fashion in $O(nm \log nm + nm^3 \log m)$ time and $O(nm^3 \log m)$ space.

Once the arrangement has been calculated, each edge of the arrangement can be visited in depth first search order. Changing from one edge to the next is equivalent to maintaining the dynamic partitioning of the point set. We prove that this can be done in $O(1)$ time per change and with at most $O(nm^3 \log m)$ such changes (see Section 2.4.2). The only requirement to maintaining such a partitioning is that we have a pointer to each sub-set and point that are involved. This is easily maintained as each segment of the original layer $\partial M_k(P, \vec{C}_i)$ is generated by a point p_j changing from one cone $\vec{C}_i$ to another $\vec{C}_{i+1}$.

Theorem 3.2. *Given a planar point set P , an oriented star R comprised of m oriented cones $\{\vec{C}_1, \dots, \vec{C}_m\}$ and an integer k , the arrangement of all regions $\overline{M_k(P, \vec{C}_i)}$ can be examined in $O(nm \log nm + nm^3 \log m)$ time and $O(nm^3 \log m)$ space.*

Proof. Immediate from above argument. □

Corollary 3.2. *Given a planar point set P , an oriented star R comprised of a constant number of oriented cones $\{\vec{C}_1, \dots, \vec{C}_{O(1)}\}$ and an integer k , the arrangement of all regions $\overline{M_k(P, \vec{C}_i)}$ can be examined in $O(n \log n)$ time and $O(n)$ space.*

3.3.2 Solutions and Analyses

This section details the application of the maximal region approach to star translation problems. Five occupancy based problems are studied and full analyses of the solutions are given.

Problem 3.1. *Given a planar point set P and a star R find a translation of R that maximizes the occupancy of R .*

In short the goal of this problem is to find a translation of R for which the most number of cones contain at least one point. This is the same as finding a point at which the most number of $\overline{M_0(P, \vec{C}_i)}$'s overlap. To solve this problem we simply examine the arrangement of $\overline{M_0(P, \vec{C}_i)}$'s as described in the previous section and determine the maximum overlap. Since only $k = 0$ must be examined the total complexity of this algorithm is $O(nm \log nm + nm^3 \log m)$ time and $O(nm^3 \log m)$ space. The minimization version of this problem is omitted as it is always possible to achieve an occupancy of 1 in any desired cone.

Problem 3.2. *Given a planar point set P , a star R and a fixed integer k , find a translation of R that minimizes/maximizes the k -occupancy of R .*

The second problem is simply a generalization of the first. In this case the goal is to find a translation of R that minimizes or maximizes the number of cones containing at least k points. The only difference between the solution for this problem and the solution for the previous one is the necessity to examine the arrangement of $\overline{M_{k-1}(P, \vec{C}_i)}$'s. While examining the arrangement we simply maintain where the (least/most) number of overlapping regions occurs. This is accomplished in $O(nm \log nm + nm^3 \log m)$ time and $O(nm^3 \log m)$ space.

Problem 3.3. *Given a planar point set P , a star R and a fixed integer k find a translation of R (if one exists) such that no cone of R is k -occupied.*

This problem is presented because our solution for the next problem uses this as a sub-problem. The problem is to decide if a translation of R exists such that no cone

contains at least k points. If such a placement does exist then it should be reported. This is the same finding the placement with minimum k -occupancy and determining if the k -occupancy is zero. Since that adds only $O(1)$ time to Problem 3.2 this problem can be solved in $O(nm \log nm + nm^3 \log m)$ time and $O(nm^3 \log m)$ space.

Problem 3.4. *Given a planar point set P and a star R . find the smallest integer k such that there exists a translation of R with k -occupancy 0.*

An alternative expression of this problem is to find the translation of R which minimizes that maximum number of points in any cone. Essentially the goal is to search through possible values of k and find the minimum value for which there exists a translation with k -occupancy 0.

We claim that the optimal value of k must lie in the interval $[\lceil \frac{n}{m} \rceil, \lceil \frac{n}{2} \rceil]$. The proof of this claim lies in the fact that each point must lie in at least one cone and at worst all of the points can be split into at least two cones using an arbitrary r_i ordering and placing the star on the median of that ordering.

We further claim that the search can be performed using binary search on the bounded interval $[\lceil \frac{n}{m} \rceil, \lceil \frac{n}{2} \rceil]$. This claim is supported by considering the k -occupancy of any placement that minimizes the maximum number of points in any cone. Thus as k increases the k -occupancy of such a point decreases monotonically.

Since the search is binary there are at most $O(\log n)$ values of k to examine. This leads to a $O((nm \log nm + nm^3 \log m) \log n)$ time and $O(nm^3 \log m)$ space algorithm. With a more careful binary search this is reducible from an $O(\log n)$ factor to $O(\log \delta)$ where the optimum value of $k = \lceil \frac{n}{m} \rceil + \delta$.

Problem 3.5. *Given a planar point set P , a star R and a fixed integer c . find the largest integer k such that there exists a translation of R with k -occupancy at least c .*

In this final star translation problem the goal is to find the largest k such that a placement exist where at least c cones contain k points. In some sense this is a clustering algorithm which attempts to create c equally large clusters using the

translational star placement as the clustering tool. This problem is solved much like the previous problem. Since n points must be placed among at least c cones then $k \leq \lfloor \frac{n}{c} \rfloor$.

We search the problem by choosing a value k and determining if the maximum k -occupancy is at least c (Problem 3.2). Binary search also applies in this case as the maximum k -occupancy is monotonic with k . The monotonicity is proven by the statement that the placement with maximum k -occupancy has at least the same $k-1$ occupancy. Thus the problem is solvable in $O((nm \log nm + nm^3 \log m) \log n)$ time and $O(nm^3 \log m)$ space.

3.4 Common Intersection of Simple Polygons

In Section 3.3.1 we presented the idea of finding an arrangement of maximal regions. Originally it was our idea to solve several star translation placement problems using the common intersection of a series of maximal regions. This original technique was conceived as a method to use previously studied algorithms for common polygon intersection which we assumed was a well studied field. We were surprised to find a lack of previous work on the problem. Although our solution to the star problems no longer requires finding the common intersection of simple polygons we feel that the topic is sufficiently connected to the thesis to present the following results.

Problem 3.6. *Given m simple-polygons, $\{\gamma_1, \gamma_2, \dots, \gamma_m\}$, each in the plane and consisting of a total of n line segments, find the common intersection, if any, of the polygons.*

Suppose that we want to find the common intersection of m convex polygons. This is a rather simple exercise as a convex polygon of size n_i is the intersection of n_i half planes. The intersection of all m convex polygons is, by commutativity, the intersection of n half planes. This kind of linear programming exercise can determine in linear time if the common intersection is empty given in dimensions of constant

size [86]. The less studied case is the intersection of arbitrary simple polygons.

3.4.1 Related Work

We do not want to give the impression that the subject of polygon intersection has not been researched. On the contrary a variety of related problems are well studied. For example, suppose we are given two polygons and wish to discover if they have a common intersection. If no preprocessing is performed on the polygons then this query can be answered in linear time [88]. For two identical convex polygons with $O(n)$ preprocessing, the query can be answered in $O(\log n)$ time, using Kirkpatrick and Snoeyink's tentative prune and search [78]. For two simple polygons with $O(n \log n)$ preprocessing the query can be answered in $O(l \log^2 n)$ time using Mount's scaffolding technique [88] where l is the size of the minimal link separator for the two polygons. The goal of the preprocessing in these techniques is to process each of the polygons regardless of their relative positions. This allows each polygon to be preprocessed independently and intersection queries between any pair of polygons can be answered in the time stated.

The preprocessing of Mount's technique is to produce triangulations for two simple polygons. These triangulations have the *scaffold* property that each region of the planar subdivision contains a vertex of their respective polygons. The two scaffold triangulations are overlayed and triangles exterior to one polygon and interior to another are incrementally removed. Since two polygons may require up to l line segments to separate them, there can be $O(\log n)$ triangles crossing each segment and there can be up to $O(\log n)$ query time to find one [72]. This results in a total query complexity of $O(l \log^2 n)$ time to determine if two simple polygons intersect.

Two other problems have been studied involving the intersection of polygons. The first is reporting or counting which pairs of polygons actually intersect. A variety of results can be obtained by finding the intersections of the n line segments which describe the boundaries of the polygons. However, this approach suffers as the number

of intersecting line segments can be $\Omega(n^2)$ and the number of intersecting polygons is $O(m^2)$.

Gupta *et al.* [66] provide the first substantial breakthrough with an algorithm which counts or reports the number of intersecting pairs of convex polygons. The requirement of convex polygons is used to establish a unique pair of *conflicting trapezoids* for each pair of intersecting polygons. Their algorithm creates a data structure that performs range queries on the trapezoidal decompositions of all m polygons. These queries take a query trapezoid γ and its left neighbour γ' of the same polygon and report/count any *conflicting trapezoids* from other polygons. Since there is exactly one pair of conflicting trapezoids for each pair of intersecting polygons it is necessary to use each trapezoid of the decompositions as a query trapezoid. The data structure is constructed using the *simplex composition* technique of van Kreveld [111] in $O(n^{\frac{4}{3}+\epsilon})$ time and space. The algorithm counts/reports pairwise intersecting polygons in $O(n^{\frac{4}{3}+\epsilon} + I)$ (resp. $O(n^{\frac{4}{3}+\epsilon} + I)$) time where ϵ is an arbitrarily small constant and I is the number of pairwise intersecting polygons.

Gupta *et al.* [66] also provide a more efficient technique for the reporting problem by exploiting segment intersection techniques. This alternative algorithm marks the segments of the lower hulls of each polygon as red and upper hulls as blue. They subsequently determine all intersections between red and blue segments using the Agarwal *et al.* result in $O(n^{\frac{4}{3}} \log^{O(1)}(n) + I)$ time [1]. The key component of this technique is their proof that the number of intersecting segments is only a constant factor (less than 8) from the number of pairwise intersecting convex polygons.

Alternative approaches to the problem dealt with by Gupta *et al.* are given by Agarwal *et al.* [3]. Agarwal *et al.*'s first approach is a randomized approach which first sub-divides the plane into regions and deals with each region successively. The sub-division is achieved by creating the *spines* of each polygons, which are the chords from the leftmost to rightmost points of each polygon. This collection of spines (*i.e.*, line segments) is stored in a hereditary segment tree [35] using the x projection for ordering. Each vertex of this tree represents a specific vertical strip and the spines

contained within it. Agarwal *et al.* define a property, dependent on the convexity of the polygons, which is true for exactly one node of the tree for every pair of intersecting polygons. Their algorithm then checks intersections of lower and upper hulls for each vertex through a randomized algorithm on planar arrangements by Har-Peled and Sharir [69]. This result reports all pairwise intersections in $O((n \log m + I)\alpha(n) \log n)$ expected time which is significantly faster than the previous result when $I = o(n^{\frac{4}{3}})$.

The second approach of Agarwal *et al.* [3] reports the relevant pairs of polygons in two distinct phases. During the first phase all pairs of polygons with one polygon's rightmost point contained within the other polygon are reported. This is accomplished by constructing a range query structure (see Matoušek [84]) on the set of all polygons' rightmost points and using all polygons as queries to the structure. The second phase reports rightmost intersections of upper and lower hulls of the polygons using segment intersection between two sets [1]. The combined time for these operations is $O(n^{\frac{4}{3}} \log n + I)$ time.

The other problem related to the intersection of m polygons is determining the maximum point of overlap in m translations of a fixed convex polygon. This problem was originally dealt with by Efrat *et al.* [53] who gave a solution for $m = O(n)$ that finds a point with maximum depth k in $O(nk \log n)$ time. Barequet *et al.* [17] describe an extension of this algorithm for general m producing an algorithm with a time complexity of $O(mk \log n / m \log m + n/m)$. Barequet *et al.* also present an algorithm which runs in $O(mk \log(kn/m) + n/m)$ time. Unfortunately it is not clear if this result can be extended to non-convex polygons or cases where the polygons are not identical.

In this thesis we present two algorithms to solve Problem 3.6. The first algorithm is presented in Section 3.4.2. It constructs and traverses the arrangement of polygons to determine the common intersection. The second algorithm is presented in Section 3.4.3 and begins by placing the converse of each polygon on a different plane in 3-space. The algorithm then computes the visibility map and hence the union of the converses. Calculating the converse of this union yields the desired intersection.

Lastly, Section 3.4.4 presents a proof that Problem 3.6 is THREE-SUM HARD and discusses the difficulties in discovering a more efficient algorithm than those presented.

3.4.2 The Arrangement Algorithm

Unless otherwise stated we assume that the polygons are planar, simple and do not contain holes. These assumptions allow the definition of a simple polygon to be a circular list of oriented line segments. Two successive segments in the list share an endpoint and the interior of the polygon is assumed to be left of each oriented segment.

The first observation about arrangements of polygons is that each is essentially an arrangement of line segments. Additionally information is required about the location of the interior of the polygons relative to the line segments. Fortunately this is a relatively minor addition as each segment only requires a constant amount of additional information [62]. Thus we consider arrangements of polygons to be modified versions of arrangements of lines and leave the implications of this reduction until later.

The remainder of this section briefly outlines a three phase algorithm that computes the common intersection of m polygons (see Problem 3.6). The first phase is to calculate the arrangement of the polygons. The second phase is to perform a “walk” from the exterior of the arrangement visiting the entire boundary for each polygon. That is, each segment and intersection point in the arrangement is examined. A list of polygons containing the segment or point being examined is maintained throughout the walk. A segment is marked whenever the list contains all m polygons. During the final stage the arrangement is searched again this time every marked segment of the arrangement is returned as a boundary to the common intersection.

Three facts must be proven and described to ensure the feasibility and correctness of this high-level description:

1. That the construction of the polygon arrangement is possible using the algorithms for line segment arrangements mentioned previously.
2. That the walk can be performed as mentioned.
3. That the segments reported form the entire boundary and only the boundary for the common intersection.

The first two facts are easily shown as each has already been proven by Goodrich [62]. In that paper Goodrich constructs the arrangement of polygons from the line segment arrangement and polygon interior information. He then performs a depth first search of the arrangement and maintains the desired list of polygons. Goodrich uses the information in the list to determine which edges are hidden from sight. This allows an $O(1)$ check at each intersection point to determine how many polygons are in the list and whether the list changes in traveling to the next intersection point. Thus the complexity of the above algorithm is identical to that of Goodrich's: $O(n \log n + k_1 + k_2)$ time where k_2 is the number of pairwise intersecting polygons and k_1 is the number of pairwise intersecting line segments.

It remains to be proven that the returned boundary is the boundary desired. The proof requires that any segment returned by the algorithm be on the border of the common intersection. This is true because of the correctness of Goodrich's algorithm, because the entire segment is a witness to the intersection of m intersecting polygons and additionally the segment is on the boundary of at least one polygon.

The main problem with this algorithm is that the result is entirely sensitive to the number of intersections of line segments and polygons and not sensitive to the output of the algorithm. Although it is possible to show the optimality of this algorithm for the common intersection of two polygons it is not possible to do so even for three. With two polygons, each of the intersections computed is necessary for the output of the common intersection: but that is not true for $m \geq 3$. In fact for $m \geq 3$ it is possible that the complexity of the algorithm is $\Omega(n^2)$ while the complexity of the

output is $O(1)$. Despite the non-optimality of the arrangement algorithm we focus on it to highlight the size of the arrangement versus the size of the desired output.

3.4.3 The Hidden Surface Algorithm

Before reviewing the definitions, concepts and results on Hidden Surface Removal (HSR) and visibility maps we want to indicate how this topic is useful to our solution. In Problem 3.6 the main issue is the intersection of polygons. Meanwhile the main focus of HSR and visibility maps is the union of polygonal sets. This apparent discrepancy is reversed by using DeMorgan's law to transform our problem into a union problem (*i.e.*, $\gamma_1 \cap \gamma_2 \cdots \cap \gamma_m = \overline{(\overline{\gamma_1} \cup \overline{\gamma_2} \cdots \cup \overline{\gamma_m})}$). We delay the exact nature of this transformation until the discussion of the algorithm. The main motivation for the use of visibility map algorithms is the possibility of not constructing the entire polygonal arrangement. This is key to deflating some of the complexity arrived at the algorithm presented in Section 3.4.2.

The fundamental concern of HSR is the question: Given a set of planar polygons in 3-space what portions of what polygons are “visible” in a specific projection of the space? This is the same as deciding a direction of projection and extending rays from all points in the set of polygons in that direction. The “visible” points are those points whose rays do not intersect any polygons.

The *visibility map* of the set of polygons in 3-space for a specific projection is defined to be the planar projection of all visible points in the polygons. The simplest method of determining the visibility map is to construct the projection of all polygons and then determine what portions of each polygon are visible (*i.e.*, remove the hidden surfaces). The fast computation of the visibility map in computer graphics leads to the quick rendering of three dimensional scenes. The main problem is that many scenes can contain huge numbers of polygons many of which are hidden [91]. When the simple approach is used the complexity of polygon arrangement is too insensitive to the output size of the visibility map [42, p. 149]. For this reason a variety of

approaches have been developed which are output sensitive to the visibility map size.

Since we are using HSR for a specific task a full review of material is unnecessary. Instead we focus the review on two algorithms: Sharir and Overmars [104]; and Agarwal and Sharir [6]. Both of these algorithms deal primarily with the calculation of a visibility map for a set of non-intersecting triangles in 3-space. It is also assumed that the triangles are well-behaved (*i.e.*, have a known depth ordering).

A *depth ordering* of a set of polygons is an ordering of the polygons $(\gamma_1, \gamma_2, \dots, \gamma_m)$ such that for any pair (γ_i, γ_j) where $i < j$ either γ_i hides some portion of γ_j or their projections are disjoint. Many algorithms rely on a pre-computed depth ordering of the polygons but such an ordering does not always exist.

The other commonality between [104] and [6] is the nature of the visibility map's construction. Both algorithms perform an incremental construction of the map $\mathcal{M}$. The construction proceeds in rounds of insertions such that $\mathcal{M}_i$ is a visibility map of all triangles inserted in the first i rounds. Both algorithms also maintain the boundary of $\mathcal{M}_i$ in $\partial\mathcal{M}_i$. This *contour* of $\mathcal{M}_i$ is the union of the triangles inserted in the first i rounds. In round $i + 1$ the contour $\partial\mathcal{M}_i$ determines what portion of the triangles for that round are in the new map. This is done efficiently because the triangles are inserted according to their depth ordering. The differences between the algorithms are their approach to rounds, how the new triangle information is calculated and ultimately in their performance.

In [104], Sharir and Overmars treat each round as the insertion of multiple triangles. The number of triangles to be inserted in each round is determined by the complexity of the current contour. The visibility map of the triangles to be inserted is calculated and the intersection of this new map with $\partial\mathcal{M}_i$ is calculated using arrangements of polygons. The visible portion of the union to be added is included thus forming $\mathcal{M}_{i+1}$. Finally the contour of the new map is constructed to prepare for the next round. The final complexity of the algorithm is $O(nc \log n + d)$ where $d = |\mathcal{M}|$ and $c = \max_i |\partial\mathcal{M}_i|$.

In Agarwal and Sharir's algorithm [6] each round is the insertion of a single triangle. Thus each round consists of: determining if the latest triangle is contained within the contour $\partial\mathcal{M}_i$; making necessary updates to $\mathcal{M}_i$ and making necessary updates to $\partial\mathcal{M}_i$. These determinations are made through ray shooting queries and the maintenance of three data structures on the contours. The complexity of this computation is $O(n^{2/3+\epsilon'} d^{1/3} c^{1/3} + dc^\epsilon + n^{1+\epsilon'})$ where ϵ' and ϵ are arbitrarily small positive constants and c, d are defined as before.

The main idea behind our algorithm is to use algorithms that construct the union of a set of polygons to aid in solving Problem 3.6. The goal of the remainder of this section is to describe: the transformation of the input polygons from Problem 3.6 into suitable input for the union of their complements; how to transform this union into the solution to Problem 3.6 and a brief analysis of the complexity of the entire solution.

The input to Problem 3.6 is a set of polygons. The goal of the initial transformation is to break the input into a representation of the complement of each of these polygons. The first step is to calculate a bounding box B that properly contains all of the polygons. We represent the complement of each polygon γ_i as the bounded region $\bar{\gamma}_i = B - \gamma_i$. Each of these regions $\bar{\gamma}_i$ is then broken into two polygons using vertical rays from the highest and lowest vertices of γ_i . The resulting polygons are then triangulated in $O(n)$ time to provide a set of $O(n)$ triangles representing the complements of all polygons [33].

The goal is to calculate the union of these triangles using [6]. As described above this algorithm requires that the triangles be embedded in 3-space and have an associated depth ordering. The embedding determines the depth ordering, the complexity of the visibility map and thereby the complexity of the algorithm. For now we assume that all triangles composing the regions $\bar{\gamma}_i$ are to be placed in the plane defined by $z = i$. The depth ordering of the triangles is constructed such that all triangles in $\bar{\gamma}_1$ come first, then triangles of $\bar{\gamma}_2$, etc. This is equivalent to the viewing direction of $(0, 0, 1)$ from the point $(0, 0, -\infty)$.

Next the contour of all the triangles is calculated and the boundary of B must be removed. By DeMorgan's law this leaves the complement of the solution to Problem 3.6. To perform the complement all boundaries simply change the side on which the "interior" is located.

Overall the first transformation of the algorithm requires linear time. The calculation of the union requires $O(n^{2/3+\epsilon'} d^{1/3} c^{1/3} + dc^\epsilon + n^{1+\epsilon'})$ where ϵ' and ϵ are arbitrarily small positive constants and c, d are defined as before. The final transformation requires at most $O(c)$ time as the size of the final contour is at most the largest contour achieved during the computation. Thus the total time is dominated by the calculation of the union.

The criticism of this algorithm is twofold. First, if the complexity of the visibility map of the polygons and the final contour is $d = c = O(n^2)$ then the algorithm is non-optimal by a factor of n^ϵ . Second, the algorithm is completely dependent on the depth ordering of the complements of the polygons. If at any point the contour is very large then the visibility map will be large and could be insensitive to the output.

3.4.4 Three-Sum Hardness

This chapter presents two solutions to the problem of finding the common intersection of an arbitrary number of polygons. Although the first algorithm is optimal in the worst case, only the second algorithm is even close to being output sensitive if $d = O(n^{2-\epsilon})$. These algorithms leave a critical question unanswered:

Problem 3.7. *Given m simple polygons consisting of a total of n line segments what is the complexity of determining if the common intersection is empty or not?*

?

In the remainder of this section we prove that Problem 3.7 belongs to the class of problems known as THREE-SUM HARD. These problems are studied because of the current conjecture that not one of them admits a solution in sub-quadratic time [58].

To compare the running times of groups of problems, Gajentaan and Overmars define the ideas of $f(n)$ solvable and $f(n)$ equivalent.

Definition 3.1. Given two problems PR1 and PR2 we say that PR1 is $f(n)$ solvable using PR2 if and only if every instance of PR1 of size n can be solved using a constant number of instances of PR2 (of at most linear size) and $O(f(n))$ additional time. This is denoted as

$$\text{PR1} \lll_{f(n)} \text{PR2}$$

This definition implies directly that when $\text{PR1} \lll_{f(n)} \text{PR2}$ that PR1 is easier than PR2. To quote the more formal lemma of [58]:

Lemma 3.1. *Let $\text{PR1} \lll_{f(n)} \text{PR2}$. Let $f(n)$ and $g(n)$ be polynomials. If PR2 can be solved in $O(g(n))$ time and $f(n) = O(g(n))$ then PR1 can be solved in $O(g(n))$ time. Reversely, if $\Omega(g(n))$ is a lower bound for PR1 and $f(n) = o(g(n))$ then $\Omega(g(n))$ is also a lower bound for PR2.*

To give an example, in the last section we proved that any instance of Problem 3.6 is $O(n)$ solvable using the union of triangles problem. We use this reduction to prove that Problem 3.7 is THREE-SUM HARD. Consider the same transformation of Problem 3.7 as was done to Problem 3.6. Then the problem has been transformed from determining if a common intersection exists to determining if the union of the complements contains any holes. Thus Problem 3.7 is $O(n)$ solvable by HOLE-IN-UNION.

The problem of HOLE-IN-UNION is known to be THREE-SUM HARD [58]. To prove that Problem 3.7 is THREE-SUM HARD it remains to be shown that HOLE-IN-UNION is $o(n^2)$ solvable by Problem 3.7.

Let Γ be the set of triangles whose union has a hole or doesn't. Find a bounding box B of the triangles, find the outer contour $\partial\mathcal{M}$ of the union of γ_i 's and triangulate the space between B and $\partial\mathcal{M}$. We call the set of all triangles Γ' and note that its construction is possible in $O(n\alpha(n)\log^2 n)$ time and its size is $O(n\alpha(n))$ [50]. Thus the boundary of Γ' is the bounding box B and any hole boundaries.

Objective	Time	Space
Maximize Occupancy	$O(nm \log nm + nm^3 \log m)$	$O(nm^3 \log m)$
Minimize/Maximize k -Occupancy	$O(nm \log nm + nm^3 \log m)$	$O(nm^3 \log m)$
Minimize Maximum Occupied Cone	$O((nm \log nm + nm^3 \log m) \log n)$	$O(nm^3 \log m)$
Maximize Minimum Occupancy of c Cones	$O((nm \log nm + nm^3 \log m) \log n)$	$O(nm^3 \log m)$

Table 3.1: Summary of translational star placement results

To determine if any holes exist in the union of Γ we simply determine if the complements of each triangle of Γ' have a common intersection interior to B . If a given triangle γ has an edge on B then we simply define the complement to be the remaining space of B . Otherwise we shoot two rays from the highest point in γ and define the complement to be the polygon defined by B, γ and the two segments connecting them. By determining if the resulting polygons have a common intersection we have solved HOLE-IN-UNION and proven that Problem 3.7 is THREE-SUM HARD.

3.5 Conclusions

This work on star placement derives a new structure called a *star* and analyzes some of its basic properties. This is achieved through the introduction of k maximal regions. Specifically we show that for any given integer k and point set P the k maximal region can be constructed in $O(n \log n)$ time and linear space.

Using these properties we have proven that several translation star placement problems can be solved in sub-quadratic time. A summary of the results is compiled in Table 3.1. We have derived this star structure and solved the introduced problems in the hopes of applying the structure to practical partitioning of point sets. Such partitionings could aid in the storage and query of large point sets and possibly in the creation of new data structures based on recursive partitioning.

Chapter 4

Grid Placement by Translation

The work of the previous chapter enables the placement of a star of size m to partition a point set. This is of interest because of the difficulties in partitioning a point set into multiple sub-sets by placing multiple objects (see Section 2.3). This chapter deals with these same issues by placing a regular rectangular grid using only translation to solve the occupancy problems as discussed in Section 2.4.

4.1 Introduction

We consider a set of problems involving the translation of a (finite or infinite) regular grid G of square cells relative to a set P of n points in the plane such that some objective function is optimized. For a grid cell C of G we say that C is *k-occupied* if C contains at least k points of P . The *k-occupancy* of G is the number of cells of G that are *k-occupied*. A cell is simply *occupied* if it is 1-occupied and the *occupancy* denotes the 1-occupancy.

Given a planar point set P , an infinite grid G and a fixed integer k , we consider the following problems where G is free to be translated by any vector:

Problem 4.1. *Given a planar point set P , an infinite grid G and a fixed integer k , find a translation of G that minimizes/maximizes the occupancy of G .*

Problem 4.2. *Given a planar point set P , an infinite grid G and a fixed integer k , find a translation of G that minimizes/maximizes the k -occupancy of G .*

Problem 4.3. *Given a planar point set P , an infinite grid G and a fixed integer k , find a translation of G (if one exists) such that no cell of G is k -occupied.*

We also consider the following problems of optimizing k :

Problem 4.4. *Given a planar point set P and an infinite grid G , find the smallest integer k such that there exists a translation of G with k -occupancy 0.*

Problem 4.5. *Given a planar point set P , an infinite grid G and a fixed integer r , find the largest integer k such that there exists a translation of G with k -occupancy at least r .*

For a finite grid G and a fixed integer k we also consider the following problem:

Problem 4.6. *Given a planar point set P , an finite grid G and a fixed integer k , find a translation of G that maximizes k -occupancy.*

This problem arises in several applications. In the problems of gridding and interpolation where we want to derive a regular grid from irregular data. To minimize the interpolation error we want to ensure that every grid cell contains at least 1 point to interpolate from. Finally, in the design of hash functions [12, 38], it is desirable to minimize the maximum number of points contained in any bucket.

The main tools we use to solve these problems are plane sweep [20] and our new dynamic data structure for maintaining a piecewise constant function on the Real line under insertions and deletions across intervals. Our approach is to reduce the grid translation problem to that of finding a maximally (or minimally) covered point in an arrangement of rectangles and then applying plane sweep and our data structure to find this point.

The previous work most closely related to our results is the work of Lee [81] on finding the maximum clique in a rectangle intersection graph. A rectangle intersection

Figure 4.1: The canonical representation of intervals $a = [5, 7]$, $b = [5, 8]$ and $c = [4, 9]$.

graph $G = (V, E)$ has vertices V , $|V| = n$, such that each vertex represents exactly one of the n iso-oriented rectangles. Every edge in E corresponds to a pair of rectangles that intersect and all such intersections are represented in E . Lee also uses plane sweep along with a data structure for maintaining the size of a maximum clique in an interval graph.

The off-line data structure that Lee develops is based on segment trees [21] and uses $O(\log n)$ time for insertions, deletions and queries. All of the intervals for a given problem are known in advance because the tree maintains the maximum clique in an interval graph in an off-line fashion. All of the endpoints of intervals are sorted as preprocessing, thus determining the primary intervals for the problem. Lee assumes, without changing time or space complexities, that the total number of primary intervals n is a power of two. A perfectly balanced tree is constructed in which each leaf corresponds to a primary interval. Each internal node then represents the union of primary intervals in its sub-tree. With this “segment tree” any interval over the set of primary intervals can be stored in a *canonical representation* of at most $O(\log n)$ tree vertices (see Figure 4.1).

Lee’s structure does not need to store the interval labels as shown in Figure 4.1.

Figure 4.2: $\text{Num}(v)$ and $\text{Int}(v)$ values for the example given in Figure 4.1.

For each node v in the tree the structure maintains two integers $\text{Num}(v)$ and $\text{Int}(v)$. The integer $\text{Num}(v)$ is the number of intervals whose canonical representation includes the node v . The integer $\text{Int}(v)$ is the maximum number of intervals that have a vertex of their canonical representation strictly in the subtree of v . Thus the maximum number of overlapping intervals in the interval for v is given by $\text{Clq} = \text{Num}(v) + \text{Int}(v)$. The example given in Figure 4.2 represents the $\text{Num}(v)$ and $\text{Int}(v)$ for the trees and interval from Figure 4.1. Lee proves that these values yield the correct result and can be maintained with insertion and deletions in $O(\log n)$ time per operation [81].

We present a data structure based on prefix-sum trees that also has $O(\log n)$ running time for all operations. This data structure is more flexible than the segment tree data structure as it allows intervals to be added in an online fashion. That is, it is fully dynamic, not requiring that the endpoints of the segments be known in advance. We also provide the functionality of reporting a witness value that is contained in the largest number of intervals without requiring additional space of $O(n \log n)$ as Lee's structure does. The usefulness of this structure has already been demonstrated in a paper by deBerg *et al.* [43].

The remainder of this chapter is organized as follows: In Section 4.2 we describe

Problem	Running Time	Memory
Problem 4.1	$O(n \log n)$	$O(n)$
Problem 4.2	$O(kn \log n)$	$O(kn)$
Problem 4.3	$O(kn \log n)$	$O(kn)$
Problem 4.4	$O(kn \log n \log k)$	$O(kn)$
Problem 4.5	$O(kn \log n \log k)$	$O(kn)$
Problem 4.6	$O(kmn \log mn)$	$O(m_c kn)$

Table 4.1: Summary of grid-placement results

our new data structure for maintaining the piecewise constant function: in Section 4.3 we show how to use this data structure to solve Problems 4.1–4.6; and in Section 4.4 we summarize and conclude with open problems.

4.2 Interval Data Structure

In this section we describe a data structure T that maintains a piecewise constant function $f : \mathbb{R} \rightarrow \mathbb{R}$ under the following query and update operations where $[a, b]$ is a continuous interval in $\mathbb{R}$.

1. INSERT(T, a, b, δ): Increase the value of $f(x)$ by δ for all $x \in [a, b]$.
2. DELETE(T, a, b, δ): Decrease the value of $f(x)$ by δ for all $x \in [a, b]$.
3. MAX(): Return $\max\{f(x) : x \in \mathbb{R}\}$.
4. MAX-WITNESS(): Return a value x^* such that $f(x^*) = \max\{f(x) : x \in \mathbb{R}\}$.
5. QUERY-AT(a): Return the node of T that contains a .
6. QUERY-LEFT-OF(v): Return the node v' that is the leaf predecessor of v .
7. QUERY-RIGHT-OF(v): Return the node v' that is the leaf successor of v .

Furthermore, a simple modification of this data structure answers the minimization queries $\text{MIN}()$ and $\text{MIN-WITNESS}()$ without any difficulties.

The goal of this entire section is to prove that these operations can be supported in a tree based structure using $O(n)$ space and $O(\log n)$ time where n is the number of distinct changes in the piecewise constant function. The sole exception to this is the $\text{MIN}()$ operation which returns the minimum of the function in $O(1)$ time. In the following paragraphs we highlight the basic data structure and an invariant that we maintain after each operation. Following that, we address the query operations (Section 4.2.1) and the update operations (Section 4.2.2) separately.

The data structure we use is a binary search tree T whose root is denoted by root . Each node v of T stores an interval. We denote by $\text{INT}(v)$ the interval stored at v , and by $\text{START}(v)$, respectively $\text{STOP}(v)$, the left, respectively right, endpoint of $\text{INT}(v)$. For an internal node v , the left, respectively right, child of v is denoted by $\text{LEFT}(v)$, respectively $\text{RIGHT}(v)$. The interval of an internal node v is always $\text{INT}(v) = \text{INT}(\text{LEFT}(v)) \cup \text{INT}(\text{RIGHT}(v))$. Initially, T contains one node root with $\text{INT}(\text{root}) = [-\infty, +\infty)$.

In addition to this information, each node v of T also maintains two values, $\text{STAB}(v)$ and $\Delta(v)$. Let $K(v) = \max\{f(x) : x \in \text{INT}(v)\}$ so that $K(\text{root}) = \text{MAX}()$. Intuitively, we want the value of $\text{STAB}(v)$ to be $K(v)$. Maintaining this property as an invariant would take linear time for each INSERT and DELETE operation. Instead, we maintain the following less strict invariant.

Invariant 4.1. *For every node v of T , $K(v) = \text{STAB}(v) + \sum_{u \in \text{Path}(v)} \Delta(u)$, where $\text{Path}(v)$ is the set of nodes (including v) on the path from v to the root of T .*

An implication of Invariant 4.1 is that a MAX query can be answered in constant time since $K(\text{root}) = \text{STAB}(\text{root}) + \Delta(\text{root})$. Furthermore, for a node v , if $\Delta(u) = 0$ for all nodes u on the path from v to root then $\text{STAB}(v) = K(v)$.

Our strategy for maintaining the STAB information is lazy because we postpone the update of any node values whenever possible. van Kreveld and Overmars [112]

Figure 4.3: An example of the data structure

use a similar mechanism in the design of concatenable segment trees. When updating becomes necessary, the procedure that we use to update values at internal nodes of T is $\text{PUSH-DELTA}(v)$, defined as follows.

$\text{PUSH-DELTA}(v)$

- 1: $\text{STAB}(v) \leftarrow \text{STAB}(v) + \Delta(v)$
- 2: $\Delta(\text{LEFT}(v)) \leftarrow \Delta(\text{LEFT}(v)) + \Delta(v)$
- 3: $\Delta(\text{RIGHT}(v)) \leftarrow \Delta(\text{RIGHT}(v)) + \Delta(v)$
- 4: $\Delta(v) \leftarrow 0$

Note that, if Invariant 4.1 holds for node v and its two children before calling $\text{PUSH-DELTA}(v)$, then it continues to hold after calling $\text{PUSH-DELTA}(v)$. Fig. 4.3 shows an example of the data structure.

4.2.1 Query Operations

This section introduces, details, proves and analyzes all of the query operations on the data structure. In an overview the goal is to show that each of the operations can be performed in time proportionate to the height of the tree. We prove in Section 4.2.2 that the tree can be maintained in such a way as to stay balanced (*i.e.*, have logarithmic height).

Of the five query operations, the easiest to implement is $\text{MAX}()$ which returns the maximum $f(x)$ in T . This is implemented by Invariant 4.1, in constant time since

$\text{MAX}() = K(\text{root}) = \text{STAB}(\text{root}) + \Delta(\text{root})$.

The procedure $\text{MAX-WITNESS}()$ requires a little more care, as it must return a witness to the maximum.

$\text{MAX-WITNESS}()$

```

1:  $v \leftarrow r$ 
2: while  $v$  is not a leaf do
3:    $\text{PUSH-DELTA}(v)$ 
4:   if  $\text{STAB}(\text{LEFT}(v)) + \Delta(\text{LEFT}(v)) > \text{STAB}(\text{RIGHT}(v)) + \Delta(\text{RIGHT}(v))$  then
5:      $v \leftarrow \text{LEFT}(v)$ 
6:   else
7:      $v \leftarrow \text{RIGHT}(v)$ 
8:   end if
9: end while
10: report any  $x^* \in \text{INT}(v)$ 

```

A proof of correctness for procedure $\text{MAX-WITNESS}()$ begins by noting that, before line 4 is executed, $\Delta(u) = 0$ for all $u \in \text{Path}(v)$. Therefore, by Invariant 4.1, $K(u) = \text{STAB}(u) + \Delta(u)$ for each child u of v . During each iteration, v is reset to the child u of v that maximizes $K(u)$. It follows that the algorithm reports a point in $\text{INT}(v)$ where v is a leaf such that all values in $\text{INT}(v)$ maximize f . Since modifications to T are only made by calls to PUSH-DELTA , which maintains Invariant 4.1, Invariant 4.1 is still true for all nodes after calling $\text{MAX-WITNESS}()$.

$\text{QUERY-AT}(a)$ is implemented in a similar fashion to $\text{MAX-WITNESS}()$ with the exception that the tree is traversed by searching using key a on the $\text{START}()$ values of each node.

QUERY-AT(a)

```

1:  $v \leftarrow \text{root}$ 
2: while  $v$  is not a leaf do
3:   PUSH-DELTA( $v$ )
4:   if  $a > \text{START}(\text{RIGHT}(v))$  then
5:      $v \leftarrow \text{LEFT}(v)$ 
6:   else
7:      $v \leftarrow \text{RIGHT}(v)$ 
8:   end if
9: end while
10: report  $v$ 

```

A proof that QUERY-AT(a) is correct begins by noting that, before line 4 is executed, $\Delta(u) = 0$ for all $u \in \text{Path}(v)$. Therefore, by Invariant 4.1, $K(u) = \text{STAB}(u) + \Delta(u)$ for each child u of v . So, during each iteration, v is reset to the child u of v such that $a \in \text{INT}(u)$. It follows that the algorithm reports the leaf node v such that $a \in \text{INT}(v)$. Since the only modifications to T are made by calls to PUSH-DELTA, which maintains Invariant 4.1, Invariant 4.1 is still true for all nodes after calling QUERY-AT(a). We further note that upon exiting QUERY-AT(a) the value $\text{STAB}(v)$ contains the value $f(a)$ as $\Delta(u) = 0$ for all $u \in \text{Path}(v)$.

The remaining two queries are used to check if after an update the node v is still required. Since the tree T does not store the intervals explicitly we need to check that the leaves of the tree are still required whenever we perform an update. This is discussed in more detail in the following section.

We begin with the QUERY-LEFT-OF(v) procedure. The goal of this procedure is to find the leaf v' such that $\text{STOP}(v') = \text{START}(v)$. We perform the search in a similar manner to the QUERY-AT(a) procedure.

QUERY-LEFT-OF(v)

```

1:  $a \leftarrow \text{START}(v)$ 
2:  $v' \leftarrow \text{root}$ 
3: while  $v'$  is not a leaf do
4:   PUSH-DELTA( $v'$ )
5:   if  $a \geq \text{START}(\text{RIGHT}(v'))$  then
6:      $v' \leftarrow \text{LEFT}(v')$ 
7:   else
8:      $v' \leftarrow \text{RIGHT}(v')$ 
9:   end if
10: end while
11: report  $v'$ 

```

The correctness of this algorithm is identical to that of QUERY-AT(v) with the exception that the search ensures that $\text{INT}(v')$ always properly contains a or $a = \text{STOP}(v')$.

The goal of the QUERY-RIGHT-OF(v) procedure is to find the leaf v' such that $\text{START}(v') = \text{STOP}(v)$. We perform the search using the QUERY-AT(a) as a subroutine.

QUERY-RIGHT-OF(v)

```

1:  $a \leftarrow \text{STOP}(v)$ 
2:  $v' \leftarrow \text{QUERY-AT}(a)$ 
3: report  $v'$ 

```

The proof of correctness for this last query is similar to the proof of correctness given for QUERY-AT(a).

4.2.2 Update Operations

This section is the culmination of data structure. Here we show how to dynamically update the tree while maintaining the information necessary for the query operations

of the previous section (Section 4.2.1). Finally we prove how to perform rotations on this tree without affecting the information necessary for queries. This permits the data structure to be any balanced tree that uses rotations (e.g. (a, b) trees, red-black trees, ...). We conclude with the statement of the Theorem 4.1 which analyzes the time for the operations discussed in this and the previous chapter.

Insertion and deletion operations are implemented using the single operation: $\text{INSERT-INFINITE-INTERVAL}(a, \delta)$, where a is a real number and δ is any real number. The $\text{INSERT-INFINITE-INTERVAL}(a, \delta)$ operation has the effect of increasing $f(x)$ by δ for all $x \in [a, +\infty)$. $\text{INSERT}(a, b, \delta)$ and $\text{DELETE}(a, b, \delta)$ are implemented with two calls to $\text{INSERT-INFINITE-INTERVAL}$ using values of $\pm\delta$.

We implement $\text{INSERT-INFINITE-INTERVAL}$ by first creating a new leaf w with $\text{START}(w) = a$ if T does not already contain such a leaf. We then traverse $P(w)$ from root to w calling PUSH-DELTA for each node we encounter. At the same time, we update the Δ values for nodes that lie to the right of $P(w)$. After doing this, Invariant 4.1 is maintained for all nodes except those on $P(w)$. A final traversal from w back up to root corrects the STAB and Δ values for all nodes of $P(w)$ thereby restoring Invariant 4.1. Finally the leaf w is checked to see if it needs to be present and if not then it is deleted. These ideas are implemented by the following pseudo-code:

INSERT-INFINITE-INTERVAL(a, δ)

```

1: BASIC-INSERT( $a$ )
2:  $v \leftarrow r$ 
3: while  $v$  is not a leaf do
4:   PUSH-DELTA( $v$ )
5:   if  $a \geq \text{START}(\text{RIGHT}(v))$  then
6:      $v \leftarrow \text{RIGHT}(v)$ 
7:   else
8:      $\Delta(\text{RIGHT}(v)) \leftarrow \Delta(\text{RIGHT}(v)) + \delta$ 
9:      $v \leftarrow \text{LEFT}(v)$ 
10:  end if
11: end while
12:  $\text{STAB}(v) \leftarrow \text{STAB}(v) + \delta$ 
13: repeat
14:    $v \leftarrow \text{PARENT}(v)$ 
15:    $\text{STAB}(v) \leftarrow \max\{\text{STAB}(\text{LEFT}(v)) + \Delta(\text{LEFT}(v)), \text{STAB}(\text{RIGHT}(v)) + \Delta(\text{RIGHT}(v))\}$ 
16: until  $v = r$ 
17: CLEAN-UP( $v$ )

```

We are now considering two different versions of the function f . To avoid confusion, we let f_i denote the function f before the operation and $f_{(i+1)}$ denote the function f after the operation. Similarly, we refer to Invariant 4.1 _{i} instead of Invariant 4.1 when $f = f_i$ and to Invariant 4.1 _{$(i+1)$} when $f = f_{(i+1)}$. Finally, $K_i(v)$ and $K_{(i+1)}(v)$ refer to the function $K(v)$ using $f = f_i$ and $f = f_{(i+1)}$, respectively.

The function BASIC-INSERT(a) creates a node w with $\text{START}(w) = a$ if such a node does not already exist and re-balances T . How this is accomplished is the topic of the next section. For now, we assume that after calling BASIC-INSERT(a), there exists a node v with $\text{START}(v) = a$ and Invariant 4.1 _{i} holds for every node in T .

Figure 4.4: The two cases in the proof of Claim 4.1.

Claim 4.1. *Let w denote the value of v in Line 12. Then, by the time Line 12 is executed, Invariant 4.1_(i+1) holds for all nodes not on $P(w)$.*

Proof. Suppose x is a node which has an ancestor u that is a node not on $P(w)$ and is a left child of a node on $P(w)$ (Fig. 4.4.a). Then Invariant 4.1 _{i} and Invariant 4.1_(i+1) are the same for x . The only changes to nodes in $P(x)$ are caused by calls to PUSH-DELTA, which maintains Invariant 4.1 _{i} . Therefore Invariant 4.1_(i+1) holds for x .

Suppose therefore that x has an ancestor u that is a node not on $P(w)$ but is the right child of a node on $P(w)$ (Fig. 4.4.b). Then $K_{(i+1)}(x) = K_i(x) + \delta$. If the Line 8 is omitted from the pseudo-code, then Invariant 4.1 _{i} holds. Since Line 8 is included then Invariant 4.1_(i+1) holds. $\square$

Claim 4.2. *After Line 16, Invariant 4.1_(i+1) holds for all nodes in T .*

Proof. After Line 11, we only update STAB values of nodes in $P(w)$, thus Invariant 4.1_(i+1) remains valid for any node not on $P(w)$. Therefore, we only need to show that Invariant 4.1_(i+1) is maintained for nodes of $P(w)$.

To show that Invariant 4.1_(i+1) is maintained for a node $v \in P(w)$, we use induction on the distance between v and w to show that, after time Line 15 is executed for v , Invariant 4.1_(i+1) holds for v . When $v = w$ and just before Line 12 is executed the value of $\text{STAB}(v) + \Delta(v) = K_i(v)$ due to the calls to PUSH-DELTA in Line 4l. But $K_{(i+1)}(v) = K_i(v) + \delta$ and is therefore correctly updated in Line 12.

Next we consider a node v at distance greater than 0 from w . Note that every ancestor u of v (including v itself) has $\Delta(u) = 0$ (from Line 4), and Invariant 4.1_(i+1) holds for both children of v (by the inductive hypothesis and Claim 4.1). This implies that for each child x of v , $K_{(i+1)}(x) = \text{STAB}(x) + \Delta(x)$. Since, by definition, $K_{(i+1)}(v)$ is the maximum of $K_{(i+1)}(\text{LEFT}(x))$ and $K_{(i+1)}(\text{RIGHT}(x))$, Invariant 4.1_(i+1) holds after the execution of Line 15. $\square$

CLEAN-UP(v)

```

1:  $f \leftarrow \text{STAB}(v)$ 
2:  $v_{\text{left}} \leftarrow \text{QUERY-LEFT-OF}(v)$ 
3:  $f_{\text{left}} \leftarrow \text{STAB}(v_{\text{left}})$ 
4: if  $f_{\text{left}} = f$  then
5:    $\text{STOP}(v_{\text{left}}) \leftarrow \text{STOP}(v)$ 
6:    $\text{BASIC-DELETE}(v)$ 
7: end if

```

Claim 4.3. *The procedure CLEAN-UP(v) can be implemented such that given a node v of T with $\Delta(u) = 0$ for all $u \in \text{Path}(v)$ it is possible to delete v from T if it does not represent any information and Invariant 4.1_(i+1) will still hold.*

Proof. After an interval (a, ∞) has been inserted with value δ it is key to note that the leaf interval v with $\text{START}(v) = a$ may have the same value $f(a)$ as its predecessor leaf. If that is true then the two intervals can be combined into one single leaf by expanding the predecessor's interval and deleting the leaf. We prove in the next section that $\text{DELETE}(v)$ does not affect Invariant 4.1. Since none of the other operations affect Invariant 4.1 the proof is complete. $\square$

Therefore, INSERT-INFINITE-INTERVAL correctly updates the tree T to maintain Invariant 4.1. A simple examination of the code for INSERT-INFINITE-INTERVAL and MAX-WITNESS shows that their running time is $O(h)$ where h is the height of the tree. Next we show that this height can be kept to $O(\log n)$.

Figure 4.5: A left rotation of a tree structure

4.2.3 Re-balancing

The details missing from the description of `INSERT-INFINITE-INTERVAL` are the implementations of `BASIC-INSERT( $v$ )` and `BASIC-DELETE( $v$ )`. Implementations of these functions are described in many algorithms texts (see Cormen *et al.* [39]) and thus details are not presented here.

Several methods exist for re-balancing binary search trees such that their height remains $O(\log n)$. Usually, these methods use *rotations* to re-balance the tree. For any of these re-balancing methods to be valid, it is necessary to show that the values `STAB`, Δ , and `INT` can be maintained during a rotation.

A left rotation is illustrated in Fig. 4.5. The following wrapper, placed around a left rotation, correctly updates the information stored at x and y . Right rotation is implemented symmetrically.

`LEFT-ROTATE-WITH-UPDATE( $x, y$ )`

- 1: `PUSH-DELTA( $x$ )`
- 2: `PUSH-DELTA( $y$ )`
- 3: `LEFT-ROTATE( $x, y$ )`
- 4: $\text{INT}(y) \leftarrow \text{INT}(x)$
- 5: $\text{INT}(x) \leftarrow \text{INT}(a) \cup \text{INT}(b)$
- 6: $\text{STAB}(x) \leftarrow \max\{\text{STAB}(a) + \Delta(a), \text{STAB}(b) + \Delta(b)\}$
- 7: $\text{STAB}(y) \leftarrow \max\{\text{STAB}(x), \text{STAB}(c) + \Delta(c)\}$

The following result is proven because the rotation can be implemented without affecting Invariant 4.1.

Theorem 4.1. *A set of n intervals can be stored in a data structure requiring $O(n)$ space and supporting the operations. INSERT. DELETE. MAX. MAX-WITNESS QUERY-AT. QUERY-LEFT-OF and QUERY-RIGHT-OF in $O(\log n)$ time per operation.*

4.3 Grid Translation Placement Solutions

In this section we show how the data structure from the previous section can be applied to solve several problems related to placing a regular square grid G over a set $P = \{p_1, \dots, p_n\}$ of n points in the plane.

4.3.1 Infinite Grids

Throughout this section we assume that G is an infinite grid consisting of unit squares and that one of G 's vertices is located at the origin. Since the grid is regular it is sufficient to consider translations whose x and y components are in the range $[0, 1)$. Thus, we represent translations as points in the square $[0, 1) \times [0, 1)$. This concept is explained in further detail in the following section.

Optimizing 1-Occupancy

In this section, we consider the problem of optimizing the 1-occupancy (*i.e.*, optimizing the number of grid cells occupied by at least 1 point of P). We focus on the maximization problem since the modifications required to solve the minimization problem are obvious.

Owing to the regularity of the grid, we consider only translation of at most 1 unit in the x and y directions. Therefore each point $p_i \in P$ must occupy one of nine cells of G after translation. Furthermore, of the nine cells of G within 1 unit of each point,

Figure 4.6: The left figure displays p_i in standard position and the cells that it might be in after the grid is translated. The right figure displays all locations that p_i might be in after translating the grid. This is broken into four regions by grid cell.

it is sufficient to consider translations into four such cells. This is because most of the translations are homomorphic to each other and can be ignored. Thus, we consider only translations that are up and to the right, which corresponds to p_i being in one of the cells: C_i^1 , C_i^2 , C_i^3 , and C_i^4 . This partitions the set of possible translations of p_i into four *occupation rectangles* that contain the top-left, top-right, bottom-left, and bottom-right corners of $[0, 1) \times [0, 1)$ (see Fig. 4.6).

For a cell C , let the k -occupied set of C be the set of translations of G for which C is k -occupied. It follows that the k -occupied set of C is the set of points contained in k or more occupation rectangles of C .

Lemma 4.1. *The 1-occupied set of a cell C can be computed in $O(g \log g)$ time, where g is the number of occupation rectangles of C .*

Proof. The above discussion implies that the 1-occupied set of C is the union of the occupation rectangles of C . In general the union of g rectangles has complexity $\Theta(g^2)$. By contrast, occupation rectangles have the property that each is contained in the unit square and each contains at least 1 corner of the unit square.

Let Γ_1 be the set of occupation rectangles that contain the top-left or top-right

corner of the unit square and let Γ_2 be the set of occupation rectangles that contain the bottom-left or bottom-right corner of the unit square. Then the boundary of Γ_1 is given by the lower envelope $\mathcal{L}(\Gamma_1)$ of the bottom edges of rectangles in Γ_1 and the boundary of Γ_2 is given by the upper envelope $\mathcal{U}(\Gamma_2)$ of the top edges of rectangles in Γ_2 . Since both of these are envelopes of at most g parallel line segments they each have at most $2g$ vertices and can be computed in $O(g \log g)$ time [70].

$\mathcal{L}(\Gamma_1)$ and $\mathcal{U}(\Gamma_2)$ are both x -monotone and the space above $\mathcal{L}(\Gamma_1)$ is Γ_1 and the space below $\mathcal{U}(\Gamma_2)$ is Γ_2 . Any vertex of $(\Gamma_1 \cup \Gamma_2)$ is either a vertex of Γ_1 or Γ_2 or an intersection point of $\mathcal{L}(\Gamma_1)$ and $\mathcal{U}(\Gamma_2)$. Since two iso-oriented monotone polygonal chains of length $O(g)$ intersect at most $O(g)$ times, the boundary of the 1-occupied set of C is of size $O(g)$ and can be obtained in an additional $O(g)$ time by merging $\mathcal{L}(\Gamma_1)$ and $\mathcal{U}(\Gamma_2)$. $\square$

To determine a translation that maximizes 1-occupancy we proceed as follows: For each point $p_i \in P$ we compute the four occupation rectangles generated by p_i in $O(1)$ time using the floor function. Using an $O(n \log n)$ time sorting algorithm, we then gather the occupation rectangles of each cell C and compute the 1-occupied set for C in $O(g \log g)$ time, where g is the number of occupation rectangles of C . Since each point in P generates at most 4 occupation rectangles, this step takes $O(n \log n)$ time for all cells.

Finally, we partition the 1-occupied regions into rectangles. The problem of computing the translation that maximizes the number of occupied cells then becomes the problem of finding the point contained in the largest number of rectangles (*i.e.*, finding the maximum clique in a rectangle intersection graph). This problem can be solved in $O(n \log n)$ time using plane sweep along with the data structure of Section 4.2. Refer to Lee [81] for details of the plane sweep.

Theorem 4.2. *Given n points $P = \{p_1, \dots, p_n\}$ and an infinite grid G it is possible to report a translation of G that minimizes or maximizes the occupancy of G in $O(n \log n)$ time and $O(n)$ space.*

Optimizing k -Occupancy

Next we consider the problem of optimizing the k -occupancy. As before, the algorithm for maximizing k -occupancy is easily converted into an algorithm for minimizing k -occupancy. Therefore, we focus on the maximization problem.

We use the same general approach of computing, for each cell C , the k -occupied set of C . By partitioning these sets into rectangles and then applying the algorithm for maximum clique in a rectangle intersection graph we obtain our result. The difficulty with the computation of the k -occupied set of a cell stems from its richer combinatorial structure than the 1-occupied set.

To compute the k -occupied set of a cell C having g occupation rectangles, we partition the occupation rectangles of C into four sets P_1, P_2, P_3 and P_4 , depending on which of the four corners of the unit square they contain (see Fig. 4.7). Rectangles that contain more than one corner of the unit square are assigned arbitrarily to one of their possible sets.

For each set P_i we then compute an arrangement $\mathcal{A}_i$ that partitions the unit square into $k + 1$ regions $\mathcal{R}_i^0, \dots, \mathcal{R}_i^{k-1}, \mathcal{R}_i^{\geq k}$ where all points in $\mathcal{R}_i^x$ are contained in x occupation rectangles (see the top-right of Fig. 4.7). This is done by a vertical line sweep over the rectangles in P_i starting at the corner of the unit square contained in all elements of P_i . The sweep line status is maintained by a balanced-binary tree and we construct the arrangement by keeping track of the k -smallest elements in the tree. Events only occur when rectangles terminate, at which point an element is removed from the tree, causing at most k of the $k + 1$ -smallest elements to change rank. Therefore each event can be handled in $O(k + \log g)$ time and the resulting arrangement has complexity $O(kg)$.

Each $\mathcal{A}_i$ consists of k x - y monotone chains (sometimes calls staircases), where the directions of monotonicity are different for each. Because the chains are monotone, the intersection of any horizontal or vertical line segment with all chains consists of at most $4k$ points and line segments. Since the chains consists of a total of $O(g)$

Figure 4.7: The original set of occupation rectangles for C , the k -occupied set of C and the four arrangements $\mathcal{A}_1, \dots, \mathcal{A}_4$ for $k = 3$.

line segments, this implies that the total number of intersections among all chains is $O(gk)$, and these intersections can be computed in $O(gk \log g)$ time using plane sweep [20]. This results in an arrangement $\mathcal{A}$ of orthogonal line segments that has $O(gk)$ faces.

For each face F of $\mathcal{A}$, we can examine the faces of $\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3$ and $\mathcal{A}_i$ that contain F . Since each face F' of $\mathcal{A}_i$ is labeled as being contained in $0, \dots, k-1$ or $\geq k$ occupation rectangles of P_i , we can determine whether the points in F are contained in k or more occupation rectangles of C . Thus we can determine whether for each face F of $\mathcal{A}$ if F is part of the k -occupied set. This step can be performed in $O(gk \log g)$ time using point location or, if we are a bit more careful, in $O(gk)$ time by doing a depth-first traversal of the faces of $\mathcal{A}$.

Thus, for each cell C , we can find the k -occupied set of C in $O(gk)$ time, where g is the number of occupation rectangles of C . As in the previous section, once the k -occupied sets have been computed for each cell, a plane sweep over all sets yields an optimal translation of G in an additional $O(kn \log n)$ time.

Theorem 4.3. *Given n points $P = \{p_1, \dots, p_n\}$, an integer $1 \leq k \leq n$, and an infinite grid G it is possible to report a translation of G that minimizes or maximizes the k -occupancy of G in $O(kn \log n)$ time and $O(kn)$ space.*

Finding a Translation with k -Occupancy 0

We note that the problem of finding a translation with k -occupancy 0 is solvable using the algorithm of the previous section. The translation with the minimum k -occupancy is found and if that k -occupancy is 0 then the solution is found otherwise none exists.

Theorem 4.4. *Given n points $P = \{p_1, \dots, p_n\}$, an integer $1 \leq k \leq n$, and an infinite grid G it is possible to decide if there exists a translation of G such that no cell of G is k -occupied and report such a translation if one exists in $O(kn \log n)$ time and $O(kn)$ space.*

Optimizing k

To solve Problems 4.4 and 4.5, we can perform a “doubling search” on the value of k . Initially, we guess k to be 1 and repeatedly double the value of k . For each value of k we use Theorem 4.3 or Theorem 4.4 to test if the value of k has become too large. If it has, we stop and perform binary search on the range $(k/2, k)$ to find the optimal value of k . The additional cost of this binary search is a $\log k$ factor in the running time.

Theorem 4.5. *Given n points $P = \{p_1, \dots, p_n\}$ and an infinite grid G it is possible to report the smallest integer k for which there exists a translation with k -occupancy equal to 0 in $O(kn \log n \log k)$ time and $O(kn)$ space. Additionally, given an arbitrary non-negative integer r , it is possible to report the largest integer k for which there exists a translation with k -occupancy at least r in the same time and space.*

4.3.2 Placing a Finite Grid

Next we turn to the problem of placing an $m_r \times m_c$ grid G over the point set P so that the k -occupancy of G is optimized. This appears to be a different problem, since now all translations are not equivalent up to integer additions to the x and y components. It should come as no surprise then that the grid size $m_r \times m_c = m \geq 1$ becomes a factor in the running time.

For each point $p_i \in P$, we transform p_i into a unit square s_i whose bottom right corner is p_i . Thus, if we place a grid square with its top-left corner inside of s_i then that square is occupied. Next, we compute the region $\mathcal{R}$ contained in k or more s_i 's. This is achieved, as in Section 4.3.1, by overlaying an infinite grid and calculating the k -occupancy region for each occupied cell. The region $\mathcal{R}$ is simply the union of each of these disjoint k -occupancy regions. It follows that if we place a unit square with its top-left corner in $\mathcal{R}$, then said square is k -occupied.

We make m translated copies $\mathcal{R}_1, \dots, \mathcal{R}_m$ of $\mathcal{R}$, where $\mathcal{R}_i$ is $\mathcal{R}$ translated by the vector $(-i \bmod m_c, \lfloor i/m_c \rfloor)$. It follows that if we place the top-left corner of a $m_r \times m_c$

grid G at a point that is contained in x of the $\mathcal{R}_i$ then the number of k -occupied cells of G is x . Thus, we have reduced the problem of placing G to a problem of finding a point contained in the largest number of regions $\mathcal{R}_i$. Since each $\mathcal{R}_i$ is a set of orthogonal polygons they can be decomposed into rectangles thereby reducing the problem to maximum-clique in a rectangle intersection graph.

In summary, we compute $\mathcal{R}$ in $O(kn \log n)$ time, partition $\mathcal{R}$ into rectangles and create $\mathcal{R}_1, \dots, \mathcal{R}_m$. This results in a set of rectangles of size $O(mkn)$. We then solve the maximum-clique problem in the resulting set of rectangles in time $O(mkn \log mn)$. To reduce the space requirements we note that $\mathcal{R}_1, \dots, \mathcal{R}_m$ need not be computed explicitly. Indeed, in the priority queue that implements the plane sweep of the maximum clique algorithm, each edge of $\mathcal{R}$ need appear only once at any given time. At all times during the sweep, an edge in the queue represents each of the m_c copies with the same y -coordinate. To initialize the queue the left and upper most copy of each edge is inserted into the queue. When an edge is swept over it generates the appropriate m_c simultaneous events, one for each grid column. After it has been swept the edge is translated downward and reinserted into the queue to represent the next set of m_c events. This translation is performed only m_r times for each edge thus providing the m copies of each edge.

Theorem 4.6. *Given n points $P = \{p_1, \dots, p_n\}$, an integer $1 \leq k \leq n$, and a finite grid G with m cells it is possible to report a translation of G with maximal k -occupancy in $O(mkn \log mn)$ time and $O(m_c kn)$ space.*

4.4 Conclusions

We have studied problems related to placing a regular square grid over a set of points in the plane in order to optimize some function of the k -occupancy. For small values of k most of our algorithms are logarithmic factors off of linear.

To solve these problems we have introduced a new lazy data structure for maintaining a piecewise constant function on the real line. While we use this structure to

find the maximum-clique of an interval graph under insertions and deletions of intervals it could also be used in the weighted case. This data structure is fully dynamic as it doesn't require the interval endpoints to be known in advance.

The data structure is also rather general. Besides the application of storing intervals, it may be used to store piecewise polynomial functions and return a value x that maximizes or minimizes the sum of all the functions evaluated at x . One could imagine using such a data structure with plane-sweep to perform operations like maintaining the minimum in sums of functions, or combining it with the persistence paradigm [48] to get an efficient representation of sums of functions.

An problem left open by this research is that of improving the running time of the algorithm for Problem 4.6. In particular, it would be beneficial to produce an algorithm that is output sensitive in the size of the k -occupied set.

Open Problem 4.1. *Given a planar point set P , contained in a grid cell C and fixed integer k , find an output sensitive algorithm that computes the boundary of the k -occupied region.*

Chapter 5

Rigid Motion Placement Problems

The previous two chapters discuss a variety of geometric translational placement problems. In this chapter we switch our focus to include rotation and translation. Our main contribution here is the solution of a variety of rigid motion placement problems using a single algorithmic approach.

The chapter itself is organized into four main sections. Section 5.1 presents our algorithmic approach in its most general form and leaves the specifics of its application to the remaining sections. The remaining three sections (Sections 5.2–5.4) break the rigid motion problems into three groups according to the class of geometric object to be placed (*i.e.*, stars, polygons and regular rectangular grids, respectively). Each of these sections has a similar layout and begins by specifying the relevant occupancy problems as introduced in Section 2.4.1. Once the problems are specified the number of combinatorially distinct placements is analyzed and simple solutions are reviewed. Then the general approach is adapted for the specific geometric object being considered by describing the computation of a rotation diagram for that class of geometric objects. Finally, each section concludes with a discussion of the algorithmic space and time complexities of each solution.

5.1 Introduction

This section introduces the concepts used to solve rigid motion placement problems throughout this chapter. We begin with fundamental definitions and theorems involving rigid motion placement of objects. Section 5.1.1 introduces the Rotation Diagram which is the main algorithmic tool in all of our solutions in this chapter. Finally, we conclude our introduction in Section 5.1.2 by giving the general algorithmic approach we have developed for rigid motion placement problems.

The term *rigid motion placement* is originally found in the seminal work by Chazelle [31]. In that work Chazelle defined *rigid motions* to be the placement of an object γ under an arbitrary rotation and translation. Chazelle proved facts, crucial to this thesis, concerning the placement of objects consisting of straight line segments¹. These facts are key to counting the number of different placements of such objects. We begin by paraphrasing the proofs of these ideas before presenting them.

We claim that two rigid motion placements of γ are *combinatorially distinct* if and only if the partitioning of P by γ is distinct. As an example consider any rigid motion placement of a grid on a pointset P . Every such placement partitions P into sub-sets by grid cell. That is, each sub-set of P is defined to be a sub-set contained within a single grid cell. By the given definition, two placements are combinatorially distinct if at least one sub-set in one partitioning is not present in the second partitioning.

Consider a placement of γ such that no point of P lies on any segment of γ . Now consider the smallest translation of the first placement such that one point p_0 comes in contact with a segment s_0 . Both of these placements of γ are combinatorially equivalent. Furthermore, the smallest rotation of γ around p_0 such that a second point p_1 comes in contact with a segment s_1 is also possible without changing the partitioning.

Remarkably, there still exist rigid motions that are combinatorially non-distinct and keep p_0 and p_1 in contact with s_0 and s_1 , respectively. This degree of freedom is

¹Here “line segment” may mean line segment, ray or line.

removed when a third point p_2 comes in contact with a segment s_2 . If at least one of these three segments is non-parallel to the other two then the position is called a *stable placement* as originally introduced by Chazelle [31].

Lemma 5.1 (Chazelle 1). *Given any points (p_0, p_1, p_2) and any segments (s_0, s_1, s_2) there are at most two stable placements with p_0 on s_0 , p_1 on s_1 and p_2 on s_2 . Moreover, these placements can be computed in constant time.*

Lastly we mention that Chazelle provides a proof of the following lemma crucial to the analysis of occupancy based algorithms using stable placements.

Lemma 5.2 (Chazelle 3). *If a given subset P_i can be contained within a polygonal region of γ then there exists a stable placement for which P_i and only P_i is contained within γ .*

5.1.1 Rotation Diagrams

The primary functionality of Rotation Diagrams is to represent a large set of rigid motion placements using a single planar arrangement. Dickerson and Scharstein [45] introduce the concept of rotation diagrams as a technique to describe all different stable placements of a convex polygon relative to a point set P . The definition for convex polygons defines a *containing region* for each pair of points such that one point is on the boundary of the polygon while the other is contained in the polygon. A *rotation diagram* is the union of all containing regions in which a single point is on the boundary.

In the remainder of this chapter we develop rotation diagrams for objects other than convex polygons. To do this we focus on Chazelle's lemmata (see Lemmata 5.1–5.2) and stable placements rather than on containing regions. Our goal for this section is to present the fundamentals of rotation diagrams for a wide classification of object types. We begin our description of these fundamentals with a description of rotation diagrams for convex polygons as developed by Dickerson and Scharstein [45].

Figure 5.1: Triangle and 3 points for the rotation diagram example

Figure 5.2: The coloured regions represent the containing regions for the respective points in the figure above.

In examining the convex polygon example we introduce the necessary principle components of the rotation diagram. The usefulness of these diagrams is demonstrated in the general approach described in the following section and their application to problems in the remainder of this chapter.

The primary idea behind rotation diagrams is to characterize many stable placements of an object by a planar arrangement. In the case of placing a convex polygon γ each diagram represents all placements where one of the points p_0 lies on $\partial\gamma$. Thus n rotation diagrams have to be constructed to characterize all stable placements of a convex polygon γ on n points.

For a single rotation diagram each of the possible rotations and translations is characterized by p_0 's position L on the perimeter of γ and the angular rotation Θ of γ . Since the perimeter of γ is W then $0 \leq L \leq W$ and $0 \leq \Theta \leq 2\pi$. Thus the rotation diagram for γ plots the containment region of all p_i when p_0 is on $\partial\gamma$, with respect to (L, Θ) . That is, Θ is the x -axis and L is the y -axis of the rotation diagram. The containment region of p_i is plotted as a region using these variables. An example of a rotation diagram is given in Fig. 5.2. This example is based on the 3 points and triangle given in Fig. 5.1².

The significance of this representation is the meaning of intersecting boundaries of containment regions. For the containment region of p_i , the point p_i must lie on $\partial\gamma$ by the definition. This implies that the intersection of two boundaries of containment regions represents a placement with two points on $\partial\gamma$. By definition of the rotation diagram we know that p_0 lies on the $\partial\gamma$. Therefore the intersection of the boundary of two containment regions in a rotation diagram represents a single stable rigid motion placement of γ . It is proven in [45] that for convex polygons, the boundaries of such containment regions are represented by $\Theta(m^2)$ trigonometric curve and vertical line segments, where $|\gamma| = m$. This leads to the main property of rotation diagrams discussed in this thesis.

²Fig. 5.1 and Fig. 5.2 are screen captures from a Java applet written by D. Scharstein and are used with his permission.

Property 5.1. *A rotation diagram for an object γ , a point set P and a fixed point p_0 is an arrangement of curves and line segments in which each vertex represents a stable rigid motion placement of γ on P .*

All of the rotation diagrams for this thesis use the concepts found in the rotation diagrams for convex polygons. Each diagram represents a specific configuration: in the case of convex polygons the point p_0 is on the boundary of γ . The rigid motion placements are then described with respect to that specific configuration: the rotation Θ of the object is represented as the x -axis of the diagram while constrained translation L is the y -axis. Finally, each rotation diagram focuses on determining when each specific point p_i is on the boundary of the object being placed. In the case of convex polygons this occurs when each point is on an edge of γ .

5.1.2 Approach

The solutions of this chapter hinge on defining rotation diagrams for stars, simple polygons and grids with Property 5.1. Since each rotation diagram represents a set of stable placements we also define a set of rotation diagrams whose union represents all stable placements. This allows us to construct a general approach that examines all stable placements in a single rotation diagram and ensures that repetition of the approach examines all stable placements.

Our general approach is to construct rotation diagrams for specific problems and examine the stable placements in an order that minimizes the differences between the partitionings induced by successive stable placements. Viewing the rotation diagram as an embedded graph implies the existence of a depth first search ordering of the vertices/stable placements. We assume that the arrangement is represented as a vertical decomposition and that additional vertices and segments are only examined to connect otherwise separated components. Theorem 5.1 states the amount of change possible between two vertices with a segment directly connecting them (*i.e.*, directly connected vertices).

Theorem 5.1. *Given two directly connected vertices u and v in a rotation diagram, the total number of points that can be in different sub-sets of the induced partitioning is equal to at most $O(\deg(u) + \deg(v))$.*

Proof. By Property 5.1 we know that the vertex u is a stable placement and that $O(\deg(u))$ points lie on the $\partial\gamma$. In the worst case each of these points changes from its sub-set at u to a new subset. Similarly, $O(\deg(v))$ points are on $\partial\gamma$ at placement v . At most $O(\deg(v))$ of these are different from the points on $\partial\gamma$ at u and thus at most $O(\deg(v))$ points change partitions. Since u and v are directly connected, no other boundaries are crossed and the result follows immediately. $\square$

Then the general technique is to construct the rotation diagram such that each curve and line segment is associated with the point p_i and sub-sets (P_j, P_k) which define it. Thus, given any curve or line segment or vertex of the rotation diagram we can determine in $O(1)$ time the partition changes associated with crossing that segment. The rotation diagram's vertices are examined in depth first search order while the partitioning information is updated using the dynamic structure from Section 2.4.2. This implies that as each vertex of the rotation diagram is examined the partitioning is changed appropriately. Since the structure requires at most $O(1)$ time per partition change we only need to determine the number of changes to the partition to determine the time used to perform the examination of each stable placement.

Theorem 5.2. *For points in general position the amount of time spent updating partition information while examining all of the vertices of the rotation diagram is $O(k)$ where k is the number of vertices in the rotation diagram.*

Proof. We assume that the initial partitioning is calculated for any vertex of the rotation diagram. The partitioning is then updated according to the depth first search ordering of the diagram (*N.B.*: the diagram is viewed as an embedded graph for the remainder of this proof). Examining the segment between vertices u and v can result in at most $O(\deg(u) + \deg(v))$ changes to the partitioning. For points in

general position, at most *three* points are on $\partial\gamma$ for any given placement. This implies that the degree of any vertex of the rotation diagram is of constant complexity. Since depth first search traverses at most each edge of the diagram then each vertex u is visited $\deg(u)$ times. Thus the amount of work is $\sum_{i=1}^k \deg(u_i)^2 = \sum_{i=1}^k O(1)^2 = O(k)$ for points in general position.

It remains to be proven that the depth first search ordering can be achieved in $O(k)$ time. We know from [108] that calculating the ordering takes $O(|V| + |E|)$. Since $|V| = k$ and $|E| = O(|V|)$ (by Euler's formula for planar graphs) it follows that: $O(|V| + |E|) = O(k)$. $\square$

5.2 Star Placement

We begin by reviewing the elementary definitions and notation of stars. The star R , is a collection of m rays $\{r_1, r_2, \dots, r_m\}$ sharing a common endpoint, called its *origin*, and ordered in a clockwise direction around the origin (see Fig. 5.3). For simplicity we assume that the rays are spaced at equal angles, with minor modifications all results apply to the more general case.

Given a star R we further define a set of m *cones* $\{C_1, C_2, \dots, C_m\}$ such that C_i is the region bounded by (r_i, r_{i+1}) and includes r_i (note that $r_{m+1} = r_1$). Lastly, given a point set P , the *occupancy* of R is the number of cones in R that contain at least one point. Similarly *k-occupancy* of a star R is the number of cones containing at least k points.

Having reviewed these definitions it is possible to explicitly define the star rigid motion placement problems that we study in this thesis:

Problem 5.1. *Given a planar point set P and a star R find a rigid motion of R that maximizes the occupancy of R .*

Problem 5.2. *Given a planar point set P , a star R and a fixed integer k , find a rigid motion of R that minimizes/maximizes the k -occupancy of R .*

Figure 5.3: A star with 5 rays

Problem 5.3. *Given a planar point set P , a star R and a fixed integer k find a rigid motion of R (if one exists) such that no cone of R is k -occupied.*

Problem 5.4. *Given a planar point set P and a star R , find the smallest integer k such that there exists a rigid motion of R with k -occupancy 0.*

Problem 5.5. *Given a planar point set P , a star R and a fixed integer c , find the largest integer k such that there exists a rigid motion of R with k -occupancy at least c .*

5.2.1 Rigid Motion Placements

The first algorithm we develop is a brute force technique that solves Problems 5.1–5.5. Essentially, we record necessary occupancy information for each combinatorially distinct rigid motion placement of R and find an optimal placement. We begin by asking: How many combinatorially distinct rigid motion placements are there?

Chazelle proved that Lemma 5.1 holds for a Real RAM model of computation with unit time calculable trigonometric functions. This lemma provides us with a way of

calculating an upper bound on the number of combinatorially distinct placements. Since at most each triple of points (p_0, p_1, p_2) can be matched to $O(m^3)$ triples of rays (s_0, s_1, s_2) each possibly producing a constant number of stable placements then there are at most $O(n^3 m^3)$ different placements. This is an overestimate because we can assume, by star symmetry, that $s_0 = r_1$ and this yields an $O(n^3 m^2)$ bound on the number of different placements.

For each placement we must determine how many points lie in each cone. This takes $O(n)$ time and then determining the cone with maximal occupancy takes a further $O(m)$ time and space. Thus the total time for the algorithm is $O(n^3 m^2 (n + m))$ and only $O(n + m)$ space is required.

5.2.2 Rotation Diagrams for Stars

The goal of this section is to describe a rotation diagram for stars which conforms to Property 5.1. This structure allows us to use the approach outlined in Section 5.1.2 to solve the rigid motion star placement problems. We begin by commenting on the brute force technique outlined at the end of the previous section. We continue by analyzing the placements of a star with two arbitrary points on rays of the star. This analysis provides us with the mathematical apparatus to define rotation diagrams for stars.

In evaluating the brute force approach our first observation is that any method which considers all stable placements to determine the solutions to Problems 5.1–5.5 may use at most $\Omega(n^3 m^2)$ time. The second observation is that constant re-computing of the number of points is inefficient. The following rotation diagram technique allows all of the stable placements to be examined more efficiently.

We present rotation diagrams for stars, determine their complexity and construction time. To construct the rotation diagram it is critical to calculate what rigid motions keep p_j on r_i when p_0 is on r_1 . We define D to be the distance between p_0 and p_j , and β_j to be the c.w. angle between vertical and the vector from p_j to p_0 (see

Figure 5.4: Characterization of rigid motion star placement

Fig. 5.4).

We characterize all rigid motions that force p_0 to be on r_1 using the two variables (Θ, L) where Θ is the c.w. angle of r_1 from the vertical and L is the distance from p_0 to the origin (see Figure 5.4). Following Dickerson and Scharstien's rotation diagrams for convex polygons [45] we create a rotation diagram for the star R by plotting the values of L and Θ that place each point p_j on a ray r_i (see Fig. 5.9 for an example).

The analysis of placing p_0 on r_1 and p_j on r_i breaks into four simple cases all depending on the angle Ψ_i which is the c.w. angle between r_1 and r_i . A summary of all cases is available at the end of this section (see Table 5.1).

Case 1: $\Psi_i = 0$

In this case there are exactly two values of Θ such that p_j is on $r_i = r_1$. Given that $L \geq 0$, then either $\Theta = \beta_j$ or $\Theta = \beta_j + \pi$ (see Figure 5.5). When $\Theta = \beta_j$, the point p_j is on r_i when $D \leq L$. Similarly if $\Theta = \pi + \beta_j$ then p_j is on r_i for all $L \geq 0$.

Case 2: $\Psi_i = \pi$

This case is quite simple as p_j is on r_i iff $\Theta = \beta_j$ and $0 \leq L \leq D$. Proof is by inspection and the fact that $L \geq 0$ (see Figure 5.6).

Figure 5.5: $\Psi_i = 0$ Figure 5.6: $\Psi_i = \pi$

Figure 5.7: $0 < \Psi_i < \pi$ **Case 3:** $0 < \Psi_i < \pi$

This case is best described by Figure 5.7. Two restrictions must be met for this case to occur: $\alpha \geq 0$ and $\gamma \geq 0$. These restrictions decide the range of values of θ and positive L for which p_j is on r_i .

First we calculate the angles α and γ :

$$\alpha = \Theta - \beta_j \quad (5.1)$$

$$\gamma = \pi - \Psi_i - \Theta + \beta_j \quad (5.2)$$

Then, applying these calculated values to the above restrictions yields a restriction on Θ :

$$\Theta - \beta_j \geq 0 \quad (5.3)$$

$$\beta_j \leq \Theta \quad (5.4)$$

$$\pi - \Psi_i + \beta_j - \Theta \geq 0 \quad (5.5)$$

$$\Theta \leq \pi - \Psi_i + \beta_j \quad (5.6)$$

$$\beta_j \leq \Theta \leq \pi - \Psi_i + \beta_j \quad (5.7)$$

For values of Θ in the range described by Eq. 5.7 we can analyze the length of L using the Sine Law and Figure 5.7.

$$L = D \frac{\sin \gamma}{\sin \Psi_i} \quad (5.8)$$

$$= D \frac{\sin(\pi - \Psi_i - \Theta + \beta_j)}{\sin \Psi_i} \quad (5.9)$$

$$= D \frac{\sin(\Theta + \Psi_i - \beta_j)}{\sin \Psi_i} \quad (5.10)$$

The restriction that $L \geq 0$ can also be applied to the final equation. Unfortunately, this yields no further constraint on Θ than already achieved above.

Case 4: $\pi < \Psi_i < 2\pi$

As in Case 3 there exist restraints on angles $\alpha \geq 0$ and $\gamma \geq 0$ which determine where p_j is on r_i . We calculate these angles and constraints as before obtaining:

$$\alpha = \beta_j - \Theta \quad (5.11)$$

$$\gamma = \Psi_i + \Theta - \beta_j - \pi \quad (5.12)$$

$$\pi + \beta_j - \Psi_i \leq \Theta \leq \beta_j \quad (5.13)$$

As before we use the Sine Law and Figure 5.8 to determine appropriate values of L for each value of Θ :

$$L = D \frac{\sin \gamma}{\sin(2\pi - \Psi_i)} \quad (5.14)$$

$$= D \frac{\sin(\Psi_i + \Theta - \beta_j - \pi)}{\sin(-\Psi_i)} \quad (5.15)$$

$$= D \frac{\sin(\Theta - \beta_j + \Psi_i)}{\sin \Psi_i} \quad (5.16)$$

Figure 5.8: $\pi < \Psi_i < 2\pi$

Since $\sin \Psi_i < 0$, this implies $\sin(\Theta - \beta_j + \Psi_i) \leq 0$ and this further implies:

$$\pi \leq \Theta - \beta_j + \Psi_i \leq 2\pi \quad (5.17)$$

$$\pi + \beta_j - \Psi_i \leq \Theta \leq 2\pi + \beta_j - \Psi_i \quad (5.18)$$

Mixing this last equation with the previous restriction on Θ yields no added constraints to Θ :

$$\pi + \beta_j - \Psi_i \leq \Theta \leq \beta_j \quad (5.19)$$

Summary

Given two points $\{p_0, p_j\}$ and two rays $\{r_l, r_i\} \in R$ we define all values of (Θ, L) such that $p_0 \in r_l$ and $p_j \in r_i$ by the four cases in Table 5.1.

An example of a star rotation diagram is presented in Fig. 5.9. The example is based on two points being unit distance apart with the point p_j directly above p_0 (i.e., $D = 1$ and $\beta_j = \pi$).

$\Psi_i = 0$:

a) $\Theta = \beta_j$ and $L \geq D$

b) $\Theta = \beta_j \pm \pi$ and $L \geq 0$

$\Psi_i = \pi$: $\Theta = \beta_j$ and $0 \leq L \leq D$

$0 < \Psi_i < \pi$: $\beta_j \leq \Theta \leq \pi - \Psi_i + \beta_j$ and $L = D \frac{\sin(\Theta + \Psi_i - \beta_j)}{\sin(\Psi_i)}$

$\pi < \Psi_i < 2\pi$: $\pi + \beta_j - \Psi_i \leq \Theta \leq \beta_j$ and $L = D \frac{\sin(\Theta + \Psi_i - \beta_j)}{\sin(\Psi_i)}$

Table 5.1: Summary of star rotation diagram equations

Figure 5.9: Star rotation diagram example ($D = 1$, $\beta_j = \pi$, $m = 9$)

5.2.3 Algorithm and Diagram Complexity

Given the case analysis above, it is possible to construct a diagram where p_o is on r_l with Θ as the x-axis and L as the y-axis. For each point p_j and ray r_i a curve is plotted and the total arrangement of these curves can be computed. Since there are $O(nm)$ monotonic curve segments it is possible to compute this arrangement in $O(nm \log(nm) + k)$ expected time [89] or $O((nm + k) \log nm)$ deterministic time [20] where k is the complexity of the arrangement.

For a given diagram any intersection of two curves represents a stable rigid motion placement of the star on the point set. This is proven by noting that each curve represents a point being on a ray and the diagram forces some point p_o to be on r_l which further implies an upper bound of $k = O(n^2m^2)$.

The algorithm proceeds as follows: choosing a point, calculate the rotation diagram: calculate the partitioning of points into cones for one stable point: perform a depth first search traversal of the arrangement of the diagram, updating the partitioning at each vertex: record the best partitioning seen and repeat the process for all possible points. There can be at most $O(m)$ sub-sets in any partitioning and each curve segment in the arrangement can contain a pointer to the point changing and which partitions are involved. Thus each partitioning can be created and the optimization information kept in $O(1)$ time per stable placement. Thus the algorithm's runtime is bounded by the construction of the rotation diagram. Given that n diagrams must be calculated, Problems 5.1–5.5 can be solved in $O(n^3m^2 \log nm)$ deterministic time or $O(n^3m^2)$ expected time. Thus we have achieved the goal of $O(n^3m^2)$ time complexity, which is the best achievable upper bound for solutions that consider all possible placements.

5.3 Simple Polygon Placement

The main goal of this section is to address the rigid motion placement of a simple polygon on a set of planar points. After examining the occupancy based problems, we note that most of the problems applicable to stars and grids do not apply to simple polygons (see Section 2.4.1). Thus the specific goal of this section is to solve Problem 5.6.

Problem 5.6. *Given a planar point set $P = \{p_1, p_2, \dots, p_n\}$ and a simple polygon γ with m vertices, find a rigid motion placement of γ such that it contains the most points of P of any possible rigid motion placement of γ .*

5.3.1 Introduction

Dickerson and Scharstein solve the version of this problem where γ is a convex polygon. By using rotation diagrams and a bucketing technique they yield a solution in $O(n \log n + nk^2m^2 \log mk)$ time where k is the number of points contained in the optimal solution. A more complete description is given in Section 2.2.3.

The only previous related work specifically on Problem 5.6 is in the work on stable placements by Chazelle [31]. Chazelle provides an algorithm for this problem by noting that each possible placement can be calculated in $O(n^3m^3)$ time and that given each placement the inclusion of these points in the polygon is determined in $O(n \log m)$ time using point location. This results in a $O(n^4m^3 \log m)$ algorithm.

Before describing our algorithm for this problem we review the difficulty in applying Dickerson and Scharstein's results to simple polygons. The first and most difficult problem to overcome is their successful output-sensitivity. Dickerson and Scharstein use the convexity of the polygons to show that a given rotation diagram involves at most $O(k)$ points. Their argument involves tiling a constant number of polygons to cover a square with side length equal to the diameter of the polygon. Although this is possible for most convex polygons it is not true in general for non-convex polygons.

The second difficulty is their use of convexity of the polygon to determine the size of the containment region of a specific point. The prevailing observation being that the region is monotone with segments bounded by trigonometric curves. We focus on surmounting the second problem of finding rotation diagrams for non-convex polygons. The following section presents a method of constructing rotation diagrams for a given simple polygon and two algorithms that use the diagrams. The first algorithm follows the general approach highlighted in Section 5.1.2. but the second algorithm saves space at the expense of time and follows the approach of [45].

5.3.2 Rotation Diagrams for Simple Polygons

We define a rotation diagram for a simple polygon to be analogous to the definition for convex polygons given by Dickerson and Scharstein [45] with one minor change. A rotation diagram for a simple polygon is a plot that assumes that some point $p_0 \in P$ is on segment s_j of the boundary of the polygon γ . Thus placement of γ is exclusively determined by point p_0 , segment s_j and the co-ordinates (Θ, L) of the rotation diagram. The x co-ordinate of the diagram is Θ : the angle which γ is rotated about p_0 and the y co-ordinate of the diagram is L : the distance from the origin of s_j at which p_0 is located.

Dickerson and Scharstein's rotation diagrams for convex polygons did not have the segment restriction on the placement of p_0 . Thus one of their diagrams represents a vertical stacking of m of our diagrams. We have made this subdivision to facilitate the discussion of complexity and to decrease the working space requirement of our algorithm. Dickerson and Scharstein also prove that for a convex polygon γ , the *containment region* for point p_i has size at most $O(m^2)$ in their diagram. They further show that the containment region can be represented by $O(m^2)$ vertical line segments and trigonometric curve segments. Since the previous work provides a framework, our goal is to divide Problem 5.6 into a set of problems involving convex polygons.

The computation of the rotation diagram begins by triangulating the polygon γ .

Figure 5.10: Splitting Δ with s_j

If p_i is contained in γ then it is contained in some triangle Δ of γ . Thus we need only consider the union of the containment regions for which p_i is in some triangle of γ . Thus we have reduced the construction of a rotation diagram for p_0 on s_j to the construction of a set of containment regions where p_0 is on s_j and p_i is in Δ .

Before computing these containment regions we introduce two requirements that must be satisfied by each case. We then show how to break cases which do not fit these requirements into a constant number of cases that do.

Requirement 5.1. *The line passing through s_j does not intersect the interior of triangle Δ .*

If Requirement 5.1 is not true then we divide Δ using the intersecting line and any resulting quadrilateral is arbitrarily halved into two triangles (see Fig. 5.10). This results in a constant number of cases for any Δ all satisfying Requirement 5.1.

Requirement 5.2. *Any line passing through a segment of Δ must fully contain s_j or intersect an endpoint of s_j .*

If Requirement 5.2 is violated then s_j is divided at every point of intersection (see Fig. 5.11). Since there are a constant number of segments of Δ and a line and

Figure 5.11: Splitting s_j with Δ

segment may only cross once this produces at most a constant number of cases that satisfy Requirements 5.1–5.2.

We proceed by calculating the containment region for each of the sub-cases created. The algorithm begins by constructing the convex hull CH of $\{s_j, \Delta\}$. The polygon U is defined to be $U = CH - \Delta$. Two observations are possible: that CH contains s_i as part of its boundary and that all vertices of U are visible from all points of s_j . Both of these observations are true or else one of the two requirements would be violated.

Furthermore, $p_i \in \Delta$ iff $p_i \in CH$ and $p_i \notin U$. Since CH is convex and contains s_j on its boundary, we can determine the containing region for which $p_i \in \Delta$ and $p_0 \in s_j$, using rotation diagrams for convex polygons. This is possible in constant time as $|CH| = O(1)$ [45]. All that remains is to subtract out the regions for which $p_i \in U$. This is also easily accomplished as U is either convex or contains only one reflex vertex (see Fig. 5.12). The visibility observation of the previous paragraph ensures that at most two convex polygons cover U in the case with one reflex vertex. Thus we can determine the containing region(s) that correspond to $p_i \in U$ in constant time as they are also of constant size.

Thus for a given triangle Δ , segment s_j and point p_i the region can be calculated in $O(1)$ time (assuming Requirements 5.1–5.2 are met). For any one diagram there are at most $O(nm)$ such regions as each point p_i may lie within any triangle Δ with

Figure 5.12: CH , Δ , U and the decomposition of U

at most $O(1)$ sub-cases for each of these not satisfying Requirements 1 and/or 2. The arrangement of curve segments for this diagram is calculable in $O(nm \log nm + k)$ expected time [90] or $O((nm + k) \log nm)$ deterministic time [20] where k is the number of intersections between boundaries.

A total of $O(n^2 m^2)$ intersections are possible in the resulting rotation diagram. This directly corresponds to the number of stable placements of $p_0 \in s_j$ that two other points lie on the boundary of γ . We note that the analysis of the following algorithms assumes the points are in general position such that no placement exists in which more than three points lie on the boundary.

5.3.3 Polygon Placement Algorithms

As mentioned in the introduction to this section we present two solutions to Problem 5.6. The algorithms represent a time versus space tradeoff. The first algorithm follows the general approach described in Section 5.1.2 and as such runs in $O(n^3 m^3)$ expected time with $O(n^2 m^2)$ space. The second algorithm follows the approach of [45] and runs in $O(n^3 m^3 \log nm)$ deterministic time and $O(nm)$ space.

It should be noted that the complexities of the rotation diagrams for simple polygons are overestimated. The worst case analysis does not consider the relative distance between points. In fact there can be a number of containment regions that

are empty because there exists no placement of γ which covers p_0 and p_i . Dicker-son and Scharstein use this fact along with the convexity of the polygon to decrease the number of containment regions in a given rotation diagram. Unfortunately their technique does not apply to simple polygons.

Constructing the Rotation Diagrams

The first algorithm begins by constructing the arrangement of containment regions. Once the arrangement has been constructed a single stable placement is calculated, and the number of points lying on the interior counted this requires $O((n + m) \log m)$ time. Then the arrangement can be walked. At each intersection of the arrangement a single point switches from being contained to being uncontained (or vice versa). This information is easily maintained in $O(1)$ time and the optimality of the placement is also checked. Thus the entire set of stable placements involving p_0 on s_j is checked in $O(n^2 m^2)$ time.

Since nm different rotation diagrams exist, all arrangements can be checked in $O(n^3 m^3)$ expected time or $O(n^3 m^3 \log nm)$ deterministic time. This improves on the time complexity of the previous algorithm by $O(n \log m)$ in the expected case but uses more space (*i.e.*, $O(n^2 m^2)$).

Scanning the Arrangement

The second algorithm does not construct the rotation diagram. Instead it computes all of the containment regions and sorts their boundary segments by endpoint from left to right. The goal is to scan a vertical line from left to right through the rotation diagram maintaining the maximum point of overlap. This is accomplished using a segment tree as in [45].

This technique is very similar to the arrangement construction technique by Bentley and Ottmann [20] but does not produce the arrangement. Instead at every event point (boundary intersection or boundary endpoint) the information on the number

of containment regions is updated. Since this information is represented as a single number at each elementary interval this is possible in $O(\log nm)$ time per event. Unfortunately this kind of solution does not work for stars or grids as the information required is a full partitioning of the point set at each elementary interval.

Since each rotation diagram contains at most $O(nm)$ boundaries and an arrangement complexity of $O(n^2m^2)$ this algorithm runs in $O(n^2m^2 \log nm)$ time and $O(nm)$ space per diagram. Since $O(nm)$ such diagrams exist, the program runs in $O(n^3m^3 \log nm)$ time and $O(nm)$ space.

5.4 Grid Placement

In previous related work, by Asano and Tokuyama [12] and Agarwal *et al.* [2] the grid placement problem is to minimize the maximum number of points in a given cell. In the problem that they consider, the grid is allowed to rotate and change its cell sizes in accordance to the bounding box for a given rotation. Thus the grid being placed is finite in size and has a constant number of cells. Asano and Tokuyama's result works in $O(mn^3 \log n)$ time where m is the number of grid cells. Agarwal *et al.* gave a randomized Monte Carlo algorithm that works in $\tilde{O}(\min(m^{\frac{1}{2}}n^{\frac{5}{3}} \log^{\frac{7}{3}} n + m^{\frac{3}{2}}\delta n \log^3 n, n^2))$ time where the maximal number of points in a cell is $\frac{n}{m} + \delta$ and m is the number of grid cells. Note that in both of these cases the only grids considered have an equal number of rows and columns.

We should comment that our model of computation is a real RAM model that uses the floor function. Neither of the previous works [2, 12] explicitly state which model of computation is used. For both papers, hashing with the floor function is the assumed use for the solutions obtained. Thus the Real RAM model with the floor function is required once the solution is obtained. The paper by Asano and Tokuyama [12] does use functionality equivalent to the floor function to calculate their solution. However, the use of the floor function does not appear in the calculations of Agarwal *et al.* [2].

In this section we assume that a grid G is a regular rectangular grid comprised of infinite lines dividing the plane into regular rectangular cells. During the analysis of the algorithms presented we refer either to these lines or to the segments bounding a cell. For notational simplicity assume that the grid is comprised of unit squares. This assumption can be removed and the cells may be rectangles of any size and aspect ratio: but the analysis becomes obscured with an overabundance of notation. The problems that we solve in this section are occupancy based and are substantively identical to the rigid motion star placement problems. Formally these problems are:

Problem 5.7. *Given a planar point set P and a regular rectangular grid G find a rigid motion placement of G that maximizes the occupancy of G .*

Problem 5.8. *Given a planar point set P , a regular rectangular grid G and a fixed integer k , find a rigid motion placement of G that minimizes/maximizes the k -occupancy of G .*

Problem 5.9. *Given a planar point set P , a regular rectangular grid G and a fixed integer k find a rigid motion placement of G (if one exists) such that no cone of G is k -occupied.*

Problem 5.10. *Given a planar point set P and a regular rectangular grid G , find the smallest integer k such that there exists a rigid motion placement of G with k -occupancy 0.*

Problem 5.11. *Given a planar point set P , a regular rectangular grid G and a fixed integer c , find the largest integer k such that there exists a rigid motion placement of G with k -occupancy at least c .*

We begin with a the description of an enumeration algorithm that discusses the number of stable placements and a reasonably efficient method of examining them. The following section describes the construction of a rotation diagram for a grid and applies the general technique to solve Problems 5.7–5.11.

5.4.1 Enumeration Algorithm

As discussed previously, the key to analyzing different rigid motion placements involves computing the number of combinatorially different stable placements. A stable placement is defined to be three points that are coincident to three lines. Since lines of the grid are parallel, at least two of the lines involved in a stable placement must be of a different orientation. Given this restriction and the fact that grid lines only have two orientations, then at least two of the lines have the same orientation. This is a key observation to counting the number of stable placements.

We note that $O(n^3)$ different triples of points can be constructed where two of the three points are marked as having the same orientation. These can be generated by selecting any pair of points to lie on a segment of the same orientation and then selecting one of the $n - 2$ remaining points to lie on a segment of the other orientation.

The question remains: “How many different orientations of parallel lines exist for which a pair of points lie on the parallel lines?”. We provide a constructive proof that for the points $\{p_0, p_j\}$ at a distance d_{0j} there can exist $O(\lceil d_{0j} \rceil)$ orientations with both points on two parallel lines. Without loss of generality assume that the line passing through p_0 and p_j is vertical. Now we rotate the grid around p_0 from $\Theta = 0$ to $\Theta = \pi$ and count all of the times p_j comes in contact with a line parallel to the one passing through p_0 . When $\Theta = \frac{\pi}{2}$ we note that $\lceil d_{0j} \rceil$ lines can intersect the segment between p_0 and p_j . Each of these must have come in contact with p_j during the rotation. By a similar argument for the angles from $\frac{\pi}{2}$ to π : at most $2\lceil d_{0j} \rceil$ different orientations exist for a given pair of points.

For any given pair and orientation the only remaining degree of freedom is to translate the grid along this orientation. Since translation of the grid is identical modulo 1 cell translation there exist at most two different lines for which a third point can be placed. This infers that at most $O(n^2 \sum_i \sum_{i > j} \lceil d_{ij} \rceil)$ different stable placements exist for three points on a unit regular rectangular grid. The naive method of calculating the solution to any rigid motion grid placement problem is to calculate

each of the stable placements and for each placement partition the set and evaluate the partition. This naive method would require $O(n)$ time per placement for a total of $O(n^3 \sum_i \sum_{i>j} [d_{ij}])$.

Fortunately, the analysis of the number of stable placements suggests a better method. This refined technique iterates over each possible pair of points, all possible parallel line placements and orientations. For each such case the algorithm begins by arbitrarily selecting a third point. The point is fixed on the line and the partitioning is calculated. Without loss of generality assume that the orientation of allowable translation is parallel to the y -axis. Sort all of the points according to their y coordinate modulo the grid cell. That is, sort them according to the order in which they would encounter a horizontal line when the grid is translated down. We then simulate such a translation and generate the resulting partitionings.

At each event in the horizontal translation exactly one point changes cells. To maintain this efficiently a priority queue of cells must be maintained to determine the maximum/minimum cell contents. Each update is done in $O(\log n)$ time because there are at most $O(n)$ cells. Since sorting requires $O(n \log n)$ time and the remaining $O(n)$ partitions are created in $O(n \log n)$ time, the refined algorithm requires only $O(n^2 \log n \sum_i \sum_{i>j} [d_{ij}])$ time.

5.4.2 Rotation Diagrams for Grids

The algorithm begins by calculating a rotation diagram where p_0 is on the boundary of a grid cell. Because of the symmetries of the grid it is only necessary to study the case when p_0 is at distance L along on the bottom edge of a cell C_0 and the cell is rotated at some angle $0 \leq \Theta \leq 2\pi$. The regions of this diagram are defined by the rigid motions for which a given point p_j is contained within a cell C_i . It is critical to our solution to associate a given region with the cell of the grid in $O(1)$ time.

Thus before constructing the rotation diagram we determine all of the cells that can be occupied when p_0 is on the boundary of a cell. This is accomplished by sorting

Figure 5.13: Computing the containing region for p_j and C .

all points in P according to their distance to p_i . A single pass is made through the points to group them according to the ceiling and floor of their distances. For each unique group of points the set of cells is created which any point within that group can occupy. The goal is to then calculate for each cell and the points that it can contain, the appropriate containment regions.

Each of these containment regions represents the stable placements for which p_0 is on the boundary of a cell C_0 and the point p_j is in a cell C . These containment regions are generated in a similar fashion to the regions generated for the simple polygon containment regions in Section 5.3.

Consider the polygon rotation diagrams for the three rectangles R_1 , R_2 and R_3 (see Figure 5.13). If we overlap these polygon rotation diagrams when $0 \leq L \leq 1$ and determine the region for which $p_j \in (R_1 - (R_2 \cup R_3))$. This can be done in $O(1)$ time as each diagram is $O(1)$ in size. The resulting region is the desired region and has been calculated in $O(1)$ time.

Theorem 5.3. *Given a specific cell C and a point p_j it is possible in $O(1)$ time and space to calculate the region in the rotation diagram for p_0 for which $p_j \in C$.*

Proof. Follows immediately from discussion above. □

Now the question that remains is: what is the complexity of a single diagram? From the definition of a rotation diagram p_0 is on the boundary of C_0 . This implies

that any of the remaining points can lie on parallel lines of the grid causing a total of $\sum_i \lceil d_{0i} \rceil$ orientations each with $O(n)$ different stable placements. Thus the number of vertices of the diagram is $O(n \sum_i \lceil d_{0i} \rceil)$ and since it is a planar arrangement this is also the complexity of the diagram.

The complexity of solving Problems 5.7–5.11, using the technique of Section 5.1.2, is $O(n^2 \sum_i \sum_{i < j} \lceil d_{ij} \rceil)$ expected time or $O(n^2 \sum_i \sum_{i < j} \lceil d_{ij} \rceil \log(n \sum_{i=\max} \lceil d_{ij} \rceil))$ deterministic time. The space required is $O(n \sum_{i=\max} \lceil d_{ij} \rceil)$ which indicates the size of the rotation diagram with the most number of stable placements.

Although the expected time of this algorithm is better than the worst case algorithm of the previous section this provides a time vs. space tradeoff as the first algorithm requires only linear space while the second does not use more than $O(n^2)$.

Chapter 6

Summary & Conclusions

This thesis focuses on a variety of Object Placement Problems:

Object Placement Problem *Given a set of one or more movable geometric objects, place them optimally with respect to a second set of fixed geometric objects.*

In focusing on such problems we have made contributions ranging from introducing and solving new problems to improving existing solutions. As we stated at the outset the goal of this thesis is to examine these problems using the theory and application of computational geometry and geometric arrangements. We believe that this thesis matches this goal and further believe that in doing so we have made contributions to both fields. The solutions and research provided here leave us with a variety of new questions regarding placement problems and computational geometry.

6.1 Summary

In this section we summarize the research contained within this thesis. The summary is broken into three sections representing the content of the three substantive chapters of this thesis.

Objective	Time	Space
Maximize Occupancy	$O(nm \log nm + nm^3 \log m)$	$O(nm^3 \log m)$
Minimize/Maximize k -Occupancy	$O(nm \log nm + nm^3 \log m)$	$O(nm^3 \log m)$
Minimize Maximum Occupied Cone	$O((nm \log nm + nm^3 \log m) \log n)$	$O(nm^3 \log m)$
Maximize Minimum Occupance of c Cones	$O((nm \log nm + nm^3 \log m) \log n)$	$O(nm^3 \log m)$

Table 6.1: Summary of translational star placement results

Star Translation

The analyses of star translation problems formulated and solved in this thesis are presented in Table 6.1. These solutions hinge on the construction of k Maximal Regions and subsequently finding the common intersection of the constructed regions. The derivation of these layers, their properties and their construction is a contribution of this thesis.

A further contribution is made in the section on finding the common intersection of m simple polygons. Specifically, Section 3.4 details two algorithms for finding the common intersection of polygons and classifies the problem as **THREE-SUM HARD**.

Grid Translation

Chapter 4 presents a number of solutions to grid translation problems (see Table 6.2) and presents a new data structure. Half of our solutions outperform the trivial algorithms which require that $\Omega(n^2)$ positions be examined. The remaining algorithms are dependent on the number of points k found in the solution's "most occupied" grid cell (see Table 6.2 for details). All of the solutions use our data structure which maintains a piecewise constant function on the real line.

This data structure allows insertion and deletion of a value on an interval in

Objective	Time	Space
Maximize Occupancy	$O(n \log n)$	$O(n)$
Minimize/Maximize k -Occupancy	$O(kn \log n)$	$O(kn)$
Maximize k -Occupancy Finite Grids $ G = m$	$O(mkn \log n)$	$O(m_c kn)$
Minimize Maximum Occupied Cells	$O(kn \log n \log k)$	$O(kn)$
Maximize Minimum Occupance of c Cells	$O(kn \log n \log k)$	$O(kn)$

Table 6.2: Summary of translational grid placement results

$O(\log n)$ time and $O(n)$ space (note: n is the number of interval endpoints with differing function values). Furthermore the structure maintains the minimum or maximum over the function and allows queries of function values in $O(\log n)$ time and min/max queries in $O(1)$ time.

Rigid Motion Placement

This thesis makes a variety of contributions regarding rigid motion placement problems. Firstly, we provide methods for calculating rotation diagrams for stars, grids and simple polygons. This extends previous work on rotation diagrams for convex polygons [45].

In the case of placing a simple polygon γ to contain the most number of points from P , we have presented two algorithms which improve on the previous $O(n^4 m^3 \log m)$ time and $O(n + m)$ space result of Chazelle [31] ($|P| = n$ and $|\gamma| = m$). Analysis of these algorithms is given in Tables 6.3–6.4.

The placement of a star or grid to optimize a partitioning is solved by searching rotation diagrams for the object and subsequent use of our data structure to maintain the partitioning. This data structure maintains a pointset partitioning by moving a

Objective	Deterministic Time	Space
Star Placement	$O(n^3 m^2 \log nm)$	$O(n^2 m^2)$
Polygon Placement Alg 1	$O(n^3 m^3 \log nm)$	$O(n^2 m^2)$
Polygon Placement Alg 2	$O(n^3 m^3 \log nm)$	$O(nm)$
Grid Placement Alg 1	$O(n^2 \log n \sum_i \sum_{i>j} \lceil d_{ij} \rceil)$	$O(n)$
Grid Placement Alg 2	$O(n^2 \sum_i \sum_{i>j} \lceil d_{ij} \rceil \log(n \sum_{i=\max} \lceil d_{ij} \rceil))$	$O(n \sum_{i=\max} \lceil d_{ij} \rceil)$

Table 6.3: Summary of deterministic results for rigid motion placement

Objective	Expected Time	Space
Star Placement	$O(n^3 m^2)$	$O(n^2 m^2)$
Polygon Placement Alg 1	$O(n^3 m^3)$	$O(n^2 m^2)$
Grid Placement Alg 2	$O(n^2 \sum_i \sum_{i>j} \lceil d_{ij} \rceil)$	$O(n \sum_{i=\max} \lceil d_{ij} \rceil)$

Table 6.4: Summary of expected results for rigid motion placement

given point from one sub-set to another in $O(1)$ time. The structure also answers queries about maximize or minimize sub-set size, number of sub-sets or number of sub-sets larger than a pre-specified constant all in $O(1)$ time and $O(n)$ space. Summary of the algorithmic complexity for placement solutions is found in Tables 6.3–6.4. Details on the structure can be found in Section 2.4.2.

6.2 Future Work

The contributions of this thesis give rise to several questions and avenues of future research. The first set of open problems is to determine the lower bounds and possible improvements on the algorithms presented herein. Translational placement of a star seems a likely point to begin. Consider the problem of finding a translational placement such that the cone with the largest number of points is minimized. For constant sized stars the solution of this thesis requires $\Omega(n \log^2 n)$ time and space. Unfortunately the method derived in this thesis involves repetitive construction of the boundaries of the k^{th} maximal regions which requires $\Omega(n \log n)$ time for many values of k . Future research should avoid such repetitive constructions.

The rigid motion placement problems are another area where optimality is not certain. As mentioned previously an improvement in the calculation of arrangements of trigonometric curve segments would benefit all of the solutions in Chapter 5. Some progress on curve segment arrangements has been made recently [10]. They clearly state restrictions on the curves which exclude trigonometric curves but do not have space to state the reasons for these restrictions.

Another source of future research is the worst case point set/polygon combination which realizes $\Omega(n^3 m^3)$ stable placements. While this combinatorial result is stated in this thesis there is no known lower bound. A similar worst case involving a point set which realizes the highest complexity for stable placements of a star would be of similar interest. In determining such worst case combinations it could help establish lower bounds for the problems discussed.

Finally, the solution of these partitioning problems also spurs questions about the application of such algorithms to partition based structures. For example, suppose that a point set and an oriented star are given. We could partition the point set finding the placement which minimizes the maximum occupancy. By recursively applying this to the sub-sets we can derive a structure which is a generalization of the k-d tree. This can be done using any of the placement types described each

defining a different type of structure. We conclude by stating that the exploration of these structures' properties and uses is one of the most personally intriguing areas of future research.

Bibliography

- [1] P. Agarwal. *Intersection and decomposition algorithms for planar arrangements*. Cambridge University Press, New York, 1991.
- [2] P. Agarwal, B. Bhattacharya, and S. Sen. Output-sensitive algorithms for uniform partitions of points. In A. Aggarwal and C. Rangan, editors, *Proceedings of the 10th International Symposium on Algorithms and Computation, ISAAC*, volume 1741 of *Lecture Notes in Computer Science*, pages 403–414. Springer-Verlag, December 1999.
- [3] P. Agarwal, M. de Berg, S. Har-Peled, M. Overmars, M. Sharir, and J. Vahrenhold. Reporting intersecting pairs of polytopes in two and three dimensions. In F. Dehne, J.-R. Sack, and R. Tamassia, editors, *Proceedings of the 7th International Workshop on Algorithms and Data Structures*, volume 2125 of *Lecture Notes in Computer Science*, pages 122–134. Springer-Verlag, August 2001.
- [4] P. Agarwal and S. Sen. Randomized algorithms for geometric optimization problems. In P. Pardalos, S. Rajasekaran, J. Reif, and J. Rolim, editors, *Handbook of Randomization*. Kluwer Publishing, to appear.
- [5] P. Agarwal and M. Sharir. Planar geometric location problems and maintaining the width of a planar set. In *Proceedings of the 2nd ACM-SIAM Symposium on Discrete Algorithms*, pages 449–459, 1991.

- [6] P. Agarwal and M. Sharir. Applications of a new space-partitioning technique. *Discrete & Computational Geometry*, 9:11–38, 1993.
- [7] P. Agarwal and M. Sharir. Planar geometric location problems. *Algorithmica*, 11:185–195, 1994.
- [8] P. Agarwal and M. Sharir. Arrangements and their applications. In J.-R. Sack and J. Urrutia, editors, *Handbook of Computational Geometry*, chapter 2, pages 49–120. North-Holland, Amsterdam, 2000.
- [9] P. Agarwal, M. Sharir, and E. Welzl. The discrete 2-center problem. In *Proceedings of the 13th Annual ACM Symposium on Computational Geometry*, pages 147–155, 1997.
- [10] N. Amato, M. Goodrich, and E. Ramos. Computing the arrangement of curve segments: divide-and-conquer algorithms via sampling. In *Proceedings of the 11th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 705–706, January 2000.
- [11] S. Arya, D. Mount, N. Netanyahu, R. Silverman, and A. Wu. An optimal algorithm for approximate nearest neighbor searching fixed dimensions. *Journal of the ACM*, 45(6):891–923, 1998.
- [12] T. Asano and T. Tokuyama. Algorithms for projecting points to give the most uniform distribution with applications to hashing. *Algorithmica*, 9(6):572–590, June 1993.
- [13] T. Asano and G. Toussaint. Computing the geodesic centre of a simple polygon. In D. Johnson, A. Nozaki, T. Nishizeki, and H. Willis, editors, *Perspectives in Computing: Discrete Algorithms and Complexity, Proceedings of Japan-US Joint Seminar*, pages 65–79, Boston, 1986. Academic Press.

- [14] M. Atallah, R. Cole, and M. Goodrich. Cascading divide-and-conquer: a technique for designing parallel algorithms. *SIAM Journal on Computing*, 18:499–532, 1989.
- [15] D. Avis, B. Beresford-Smith, L. Devroye, H. Elgindy, E. Guévremont, F. Hurtado, and B. Zhu. Unoriented θ -maxima in the plane: Complexity and algorithms. *SIAM Journal of Computing*, 28(1):278–296, 1998.
- [16] G. Barequet and M. Dickerson. The translation-scale diagram for point-containing placements of a convex polygon. In *Proceedings of the 12th Canadian Conference on Computational Geometry*, pages 7–12, August 2000.
- [17] G. Barequet, M. Dickerson, and P. Pau. Translating a convex polygon to contain a maximum number of points. *Computational Geometry: Theory & Applications*, 8(4):167–179, September 1997.
- [18] F. Bauernöppel, E. Kranakis, D. Krizanc, A. Maheshwari, J.-R. Sack, and J. Urrutia. Planar stage graphs: characterizations and applications. *Theoretical Computer Science*, 175:239–255, 1997.
- [19] J. Bentley, H. Kung, M. Schkolnick, and C. Thompson. On the average number of maxima in a set of vectors and applications. *Journal of the ACM*, 25:536–543, 1978.
- [20] J. Bentley and T. Ottmann. Algorithms for reporting and counting geometric intersections. *IEEE Transactions on Computing*, C-28:643–647, 1979.
- [21] J. L. Bentley. Algorithms for Klee’s rectangle problems. Carnegie-Mellon University, Department of Computer Science, unpublished notes, 1977.
- [22] S. Bespamyatnikh and D. Kirkpatrick. Rectilinear 2-center problems. In *Proceedings of the 11th Conference on Computational Geometry*, pages 68–71, 1999.

- [23] S. Bespamyatnikh and M. Segal. Rectilinear static and dynamic discrete 2-center problems. In F. Dehne, A. Gupta, J.-R. Sack, and R. Tamassia, editors, *Proceedings of the 6th International Workshop on Algorithms and Data Structures*, volume 1663 of *Lecture Notes in Computer Science*, pages 276–287. Springer-Verlag, August 1999.
- [24] S. Bespamyatnikh and M. Segal. Covering a set of points by two axis-parallel boxes. *Information Processing Letters*, 75(3):95–100, 2000.
- [25] P. Bose, A. Maheshwari, P. Morin, and J. Morrison. The grid placement problem. In F. Dehne, J.-R. Sack, and R. Tamassia, editors, *Proceedings of the 7th Annual Workshop on Algorithms and Data Structures*, volume 2125 of *Lecture Notes in Computer Science*, pages 180–191. Springer-Verlag, August 2001.
- [26] P. Bose and G. Toussaint. Computing the constrained Euclidean, geodesic and link centers of a simple polygon with applications. *Studies in Location Analysis: Special Issue on Computational Geometry*, 15:37–66, 2000.
- [27] P. Bose, M. van Kreveld, A. Maheshwari, P. Morin, and J. Morrison. Translating regular grids over point sets. Accepted to: Special Issue of Computational Geometry: Theory & Applications on EURO CG 2001, 2002.
- [28] P. Bose and Q. Wang. Facility location constrained to a polygonal domain. In *Proceedings of the Latin American Theoretical Informatics Symposium 2002*, page to appear. Springer-Verlag, 2002.
- [29] T. Chan. More planar two-center algorithms. *Computational Geometry: Theory & Applications*, 13:189–198, 1999.
- [30] T. Chan. A fully dynamic algorithm for planar width. In *Proceedings of the 17th Annual Symposium on Computational Geometry*, pages 172–176. ACM Press, 2001.

- [31] B. Chazelle. The polygon placement problem. In F. Preparata, editor. *Advances in Computing Research*, volume 1, pages 1–34. JAI Press, 1983.
- [32] B. Chazelle. On the convex layers of a planar set. *IEEE Transactions on Information Theory*, IT-31(4):509–517, July 1985.
- [33] B. Chazelle. Triangulating a simple polygon in linear time. *Discrete & Computational Geometry*, 6:485–524, 1991.
- [34] B. Chazelle and H. Edelsbrunner. An optimal algorithm for intersecting line segments in the plane. *Journal of the ACM*, 39(1):1–54, January 1992.
- [35] B. Chazelle, H. Edelsbrunner, L. Guibas, and M. Sharir. Algorithms for bichromatic line segment problems and polyhedral terrains. *Algorithmica*, 11:116–132, 1994.
- [36] K. Clarkson and P. Shor. Applications of random sampling in computational geometry II. *Discrete & Computational Geometry*, 4:387–421, 1989.
- [37] R. Cole, M. Sharir, and C. Yap. On k -hulls and related problems. *SIAM Journal on Computing*, 16(1):61–77, 1987.
- [38] D. Comer and M. O'Donnell. Geometric problems with applications to hashing. *SIAM Journal on Computing*, 11(2):217–226, May 1982.
- [39] T. Cormen, C. Leiserson, and R. Rivest. *Introduction to Algorithms*. MIT Press, Cambridge, 1990.
- [40] A. Datta, H. Lenhof, C. Schwarz, and M. Smid. Static and dynamic algorithms for k -point clustering problems. *Journal of Algorithms*, 19:474–503, 1995.
- [41] A. Datta, A. Maheshwari, and J.-R. Sack. Optimal parallel algorithms for direct dominance problems. *Nordic Journal of Computing*, 3:72–88, 1996.

- [42] M. de Berg. *Ray Shooting, Depth Orders and Hidden Surface Removal*, volume 703 of *Lecture Notes in Computer Science*. Springer-Verlag, Heidelberg, 1993.
- [43] M. deBerg, P. Bose, O. Cheong, and P. Morin. On simplifying dot maps. European Workshop on Computational Geometry, March 2002.
- [44] M. deBerg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry*. Springer-Verlag, New York, 1997.
- [45] M. Dickerson and D. Scharstein. Optimal placement of convex polygons to maximize point containment. *Computational Geometry: Theory and Applications*, 11(1):1–16, August 1998.
- [46] H. Djidjev, A. Lingas, and J.-R. Sack. An $O(n \log n)$ algorithm for computing the link center of a simple polygon. *Discrete & Computational Geometry*, 8:131–152, 1992.
- [47] Z. Drezner. On the rectangular p -center problem. *Naval Res. Logist.*, 34:229–234, 1987.
- [48] J. Driscoll, N. Sarnak, D. Sleator, and R. Tarjan. Making data structures persistent. *Journal of Computer and System Sciences*, 38(1):86–124, February 1989.
- [49] C. Duncan, M. Goodrich, and S. Kobourov. Balanced aspect ratio trees: combining the advantages of k-d trees and octrees. In *Proceedings of the 10th ACM-SIAM Symposium on Discrete Algorithms*, pages 300–309, Baltimore, January 1999.
- [50] H. Edelsbrunner, L. Guibas, and M. Sharir. The complexity and construction of many faces in arrangements of lines and segments. *Discrete & Computational Geometry*, 5:161–196, 1990.

- [51] H. Edelsbrunner, D. Kirkpatrick, and R. Seidel. On the shape of a set of points in the plane. *IEEE Transactions on Information Theory*, IT29(4):551–559, 1983.
- [52] A. Efrat, G. Rote, and M. Sharir. On the union of fat wedges and separating a collection of segments by a line. *Computational Geometry: Theory & Applications*, 3:277–288, 1994.
- [53] A. Efrat, M. Sharir, and A. Ziv. Computing the smallest k -enclosing circle and related problems. *Computational Geometry Theory & Applications*, 4:119–136, 1994.
- [54] D. Eppstein. Dynamic three-dimensional linear programming. *ORSA J. Computing*, 4(4):360–368, 1992. Special issue on computational geometry.
- [55] D. Eppstein. Faster construction of planar two-centers. In *8th ACM-SIAM Symposium on Discrete Algorithms*, pages 131–138, New Orleans, 1997.
- [56] D. Eppstein. Incremental and decremental maintenance of planar width. *Journal of Algorithms*, 37(2):570–577, 2000.
- [57] D. Eppstein and J. Erickson. Iterated nearest neighbors and finding minimal polytopes. *Discrete & Computational Geometry*, 11:321–350, 1994.
- [58] A. Gajentaan and M. Overmars. On a class of $o(n^2)$ problems in computational geometry. *Computational Geometry: Theory and Applications*, 5(3):165–185, 1995.
- [59] A. Glozman, K. Kedem, and G. Shpitalnik. On some geometric selection and optimization problems via sorted matrices. *Computational Geometry: Theory & Application*, 11:17–28, 1998.
- [60] J. Goodman. Pseudoline arrangements. In J. Goodman and J. O’Rourke, editors, *Handbook of Discrete and Computational Geometry*, The CRC Press

- Series on Discrete Mathematics and its Applications. pages 83–110. CRC Press. New York. 1997.
- [61] M. Goodrich. Intersecting line segments in parallel with an output sensitive number of processors. *SIAM Journal on Computing*. 20(4):737–755. 1991.
- [62] M. Goodrich. A polygonal approach to hidden-line and hidden-surface elimination. *Computer Vision, Graphics, and Image Processing: Graphical Models and Image Processing*. 54(1):1–12. 1992.
- [63] P. Green and B. Silverman. Constructing the convex hull of a set of points in the plane. *The Computer Journal*. 22(3):262–266. 1979.
- [64] S. Guha. An $o(n)$ time set bipartition algorithm under the l_∞ metric. Technical Report 48109-2122. EECS Department University of Michigan. Ann Arbor. MI. 1989.
- [65] L. Guibas and M. Sharir. Combinatorics and algorithms of arrangements. In J. Pach, editor, *New Trends in Discrete and Computational Geometry*, number 10 in Algorithmics and Combinatorics, chapter 1, pages 9–36. Springer-Verlag, Berlin. 1993.
- [66] P. Gupta, R. Janardan, and M. Smid. Efficient algorithms for counting and reporting pairwise intersections between convex polygons. *Information Processing Letters*. 69(1):7–13. 1999.
- [67] P. Gupta, R. Janardan, M. Smid, and B. Dasgupta. The rectangle enclosure and point-dominance problems revisited. In *Proceedings of the 11th Annual Symposium on Computational Geometry*, pages 162–171. Vancouver. 1995.
- [68] R. Güting, O. Nurmi, and T. Ottman. Fast algorithms for direct enclosures and direct dominances. *Journal of Algorithms*. 10:170–186. 1989.

- [69] S. Har-Peled and M. Sharir. On-line point location in planar arrangements and its applications. In *Proceedings of the 12th Annual Symposium on Discrete Algorithms*, pages 57–66, 2001.
- [70] J. Hershberger. Finding the upper envelope of n line segments in $O(n \log n)$ time. *Information Processing Letters*, 33(4):169–174, December 1989.
- [71] J. Hershberger and S. Suri. Finding tailored partitions. *Journal of Algorithms*, 12:431–463, 1991.
- [72] J. Hershberger and S. Suri. A pedestrian approach to ray shooting: Shoot a ray, take a walk. *Journal of Algorithms*, 18(3):403–431, May 1995.
- [73] F. Hurtado, V. Sacristán, and G. Toussaint. Constrained facility location. *Studies in Locational Analysis: Special Issue on Computational Geometry in Locational Analysis*, 15, December 2000.
- [74] J. Jaromczyk and M. Kowaluk. The two-line center problem from a polar view: A new algorithm and data structure. In *Proceedings of the 4th Workshop on Algorithms and Data Structures*, volume 955 of *Lecture Notes in Computer Science*, pages 13–25. Springer-Verlag, 1995.
- [75] J. Jaromczyk and M. Kowaluk. Orientation independent covering of point sets in r^2 with pairs of rectangles or optimal squares. In *Proceedings of the European Workshop on Computational Geometry*, volume 871 of *Lecture Notes in Computer Science*, pages 71–78. Springer-Verlag, 1996.
- [76] M. Katz, K. Kedem, and M. Segal. Constrained square-center problems. *Computational Geometry: Theory & Applications*, 15:203–214, 2000.
- [77] M. Katz and M. Sharir. An expander-based approach to geometric optimization. *SIAM Journal of Computing*, 26(5):1384–1408, October 1997.

- [78] D. Kirkpatrick and J. Snoeyink. Tentative prune-and-search for computing fixed-points with applications to geometric computation. *Fundamenta Informaticae*. 22(4):353–370. 1995.
- [79] M. Ko and Y. Ching. Linear time algorithms for the weighted tailored 2-partition problem and the weighted rectilinear 2-center problem under l_∞ metric. *Discrete Applied Mathematics*. 40:397–410. 1992.
- [80] H. Kung, F. Luccio, and F. Preparata. On finding the maxima of a set of vectors. *Journal of the ACM*. 22(4):469–476. October 1975.
- [81] D. Lee. Maximum clique problem of rectangle graphs. In F. Preparata, editor. *Advances in computing research*, volume 1, pages 91–107. JAI Press Inc., 1983.
- [82] D. Lee and F. Preparata. An improved algorithm for the rectangle enclosure problem. *Journal of Algorithms*. 3:218–224. 1982.
- [83] A. Maheshwari, J.-R. Sack, and H. Djidjev. Link distance problems. In J.-R. Sack and J. Urrutia, editors. *Handbook of Computational Geometry*, chapter 12, pages 519–558. North-Holland, Amsterdam, 2000.
- [84] J. Matoušek. Range searching with efficient hierarchical cuttings. *Discrete & Computational Geometry*. 10(2):157–182. 1993.
- [85] J. Matoušek. On enclosing k points by a circle. *Information Processing Letters*. 53:217–221. 1995.
- [86] N. Megiddo. Linear-time algorithms for linear programming in $\mathbb{R}^3$ and related problems. *SIAM Journal on Computing*. 12(4):759–776. November 1983.
- [87] J. Mitchell. Geometric shortest paths and network optimization. In J.-R. Sack and J. Urrutia, editors. *Handbook of Computational Geometry*, chapter 15. North-Holland, Amsterdam, 2000.

- [88] D. Mount. Intersection detection and separators for simple polygons. In *Proceedings of the 8th Annual Symposium on Computational Geometry*, pages 303–311. ACM, ACM Press, June 1992.
- [89] K. Mulmuley. A fast planar partition algorithm. *International Journal on Symbolic Computation*, 10(3–4):253–280, 1990.
- [90] K. Mulmuley. A fast planar partition algorithm II. *Journal of the ACM*, 38(1):74–103, January 1991.
- [91] T. Murali. *Efficient hidden-surface removal in theory and practice*. PhD thesis, Brown University, May 1999.
- [92] F. Nielson. Output-sensitive peeling of convex and maximal layers. *Information Processing Letters*, 59:255–259, 1996.
- [93] J. Nievergelt and P. Widmayer. Spatial data structures: concepts and design choices. In M. van Kreveld, J. Nievergelt, T. Roos, and P. Widmayer, editors, *Algorithmic foundations of geographic information systems*, number 1340 in Lecture Notes in Computer Science, pages 153–197. Springer-Verlag, Berlin, 1997.
- [94] D. Nussbaum. Rectilinear p -piercing problems. In W. Kuchlin, editor, *Proceedings of the 1997 International Symposium on Symbolic and Algebraic Computation*, pages 316–323. ACM Press, July 1997.
- [95] A. Okabe, B. Boots, K. Sugihara, and S. Chiu. *Spatial tessellations: concepts and applications of Voronoi diagrams*. John Wiley and Sons, Inc, Chichester, second edition, 2000.
- [96] M. Overmars and J. van Leeuwen. Maintenance of configurations in the plane. *Journal of Computer and System Sciences*, 23:166–204, 1981.

-
- [97] R. Pollack, M. Sharir, and E. Rote. Computing the geodesic center of a simple polygon. *Discrete & Computational Geometry*, 4:611–626, 1989.
 - [98] F. Preparata and M. Shamos. *Computational Geometry: An Introduction*. Texts and Monographs in Computer Science. Springer-Verlag, 1985.
 - [99] J.-R. Sack and J. Urrutia, editors. *Handbook of Computational Geometry*. North-Holland, Amsterdam, 2000.
 - [100] R. Seidel. Backwards analysis of randomized algorithms. In J. Pach, editor, *New trends in discrete and computational geometry*, volume 10 of *Algorithms and Combinatorics*, pages 37–68. Springer-Verlag, 1993.
 - [101] M. Shamos. *Computational Geometry*. PhD thesis, Yale University, New Haven, CT, 1978.
 - [102] M. Sharir. A near-linear algorithm for the planar 2-center problem. *Discrete & Computational Geometry*, 18:125–134, 1997.
 - [103] M. Sharir and P. Agarwal. *Davenport-Schinzel Sequences and Their Applications*. Cambridge University Press, New York, 1995.
 - [104] M. Sharir and M. Overmars. A simple output-sensitive algorithm for hidden surface removal. *ACM Transactions on Graphics*, 11(1):1–11, January 1992.
 - [105] M. Sharir and E. Welzl. Rectilinear and polygonal p -piercing and p -center problems. In *Proceedings of the 12th ACM Symposium on Computational Geometry*, pages 122–132, 1996.
 - [106] M. Smid. 2001. Personal Communication.
 - [107] J. Sylvester. On Poncelet’s approximate linear valuation of surd forms. *Philosophical Magazine, Ser. 4*, 20:203–220, 1860.

- [108] R. Tarjan. Depth-first search and linear graph algorithms. *Siam Journal on Computing*, 1(2):146–160. June 1972.
- [109] G. Toussaint. Solving geometric problems with the rotating calipers. In *Proceedings of IEEE MELECON'83*. May 1983.
- [110] V. Vaishnavi and D. Wood. Data structures for rectangle containment and enclosure problems. *Computer Graphics Image Processing*, 13:372–384. 1980.
- [111] M. van Kreveld. The power of parallel projection. *Information Processing Letters*, 46(4):185–191. June 1993.
- [112] M. van Kreveld and M. Overmars. Concatenable segment trees (extended abstract). In B. Monien and R. Cori, editors. *Proceedings of the 6th Annual Symposium on Theoretical Aspects of Computer Science*, volume 349 of *Lecture Notes in Computer Science*, pages 493–504. Springer-Verlag, February 1989.

Index

- α -hull. 16
- δ Fat Cones. 59
- k -face. 22
- k -hull. 15
- k -occupancy. 41
- k -set. 15
- k^{th} Maximal Region. 47
- Arrangement. 21
 - k -level. 22
 - Cells. 22
 - Computational Complexity. 22
 - Faces. 22
- Boundary
 - Point Set. 17
- Centre Problems
 - Discrete. 33
- Centre Problem
 - Link. 40
- Centre Problems
 - Constrained. 33
 - Unconstrained. 33
- Combinatorially Distinct. 99
- Cone
 - k -occupied. 46
- Containing Region. 31. 100. 101
- Convex
 - Orthogonally. 14
- Convex Hull. 14. 15
- Dominance
 - Direct. 12
- Dominance Pair
 - Direct. 12
 - Redundant. 12
- Dominates. 11
- Envelope
 - Lower. 22
 - Upper. 22
- Event Point. 11
 - Schedule. 11
- Extreme Points. 15
- Fat Cones. 59
- Generalized Disc. 16
- Geodesic Distance. 40
- Hidden Surface Removal(HSR). 68

-
- Homothetic Copy. 27
 - Link Distance. 39
 - Maximal Elements. 12
 - Maximal Layer. 14
 - Maximal Outer Hull. 14
 - Maximal Region. 47
 - Object Placement Problem. 1. 127
 - Objects. 10
 - Occupancy. 41
 - Occupation Rectangles. 90
 - Onion Peeling. 17
 - Placement
 - Rigid Motion. 99
 - Point
 - Cluster. 33
 - Polygon
 - Scaffold. 63
 - Rigid Motion. 30. 99
 - Rotation Diagram. 31
 - Space
 - index. 21
 - Star
 - k -occupancy. 46
 - Occupancy. 46
 - Oriented. 45
 - Sweep. 10
 - Plane. 10
 - Space. 10
 - Translation. 26
 - Vertical Decomposition. 24
 - Visibility Map. 68
 - Width. 31