UNIVERSITY OF CALIFORNIA
Santa Barbara

Computing Volumes and Convex Hulls: Variations and Extensions

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

Doctor of Philosophy

in

Computer Science

by

Hakan Yıldız

Committee in Charge:

Professor Subhash Suri, Chair

Professor Amr El Abbadi

Professor John Gilbert

June 2014

The dissertation of
Hakan Yıldız is approved:

Professor Amr El Abbadi

Professor John Gilbert

Professor Subhash Suri, Committee Chairperson

May 2014

Computing Volumes and Convex Hulls:

Variations and Extensions

by

Hakan Yıldız

Acknowledgements

I have no doubt that I would not be able to write this dissertation without the encouragement and contributions of some wonderful and bright people. I, therefore, would like to devote this section to acknowledge those whose support made the completion of this dissertation possible.

First and foremost, I would like to thank my supervisor, Prof. Subhash Suri, for his guidance during my Ph.D. studies. In addition to his continuous encouragement for research and scientific contributions, I think I learned a lot from him in terms of what teaching is about. It is my hope that, if I ever end up in a teaching position in the future, I can put into use what I have learnt from him.

Besides my supervisor, I would like to thank the rest of my Ph.D. committee; Prof. Amr El Abbadi and Prof. John Gilbert; for their encouragement, insightful comments and sincere suggestions.

This section cannot be complete without acknowledging my labmates: Luca, Pegah, Kyle, Kevin, Lucas, and Jonathan. I am grateful that they generously shared their knowledge with me and made my research experience enjoyable.

I also would like to thank my collaborators from outside UCSB; John Hershberger, Pankaj Agarwal, Wuzhou Zhang, and Sariel Har-Peled; for their important contributions to my research.

Finally, I would like to express my gratefulness to my family. With their unconditional support throughout my entire life, they made it possible for me to become who I am now.

Curriculum Vitæ

Hakan Yıldız

Education

2014 (Expected)	Doctor of Philosophy in Computer Science, University of California, Santa Barbara.
2013	Master of Science in Computer Science, University of California, Santa Barbara.
2008	Bachelor of Science in Computer Engineering, Middle East Technical University, Ankara, Turkey

Professional Experience

2010-2013	Research Assistant, Department of Computer Science, UC Santa Barbara, CA, USA.
2013	SDE Intern, Microsoft Corporation, Redmond, WA, USA.
2011	SDE Intern, Microsoft Corporation, Redmond, WA, USA.
2008-2010	Teaching Assistant, Department of Computer Science, UC Santa Barbara, CA, USA.
2006	Intern, METU Research and Development Center, Ankara, Turkey.
2005	Intern, IBM Türk, Ankara, Turkey.

Publications

- **P. K. Agarwal, S. Har-Peled, S. Suri, H. Yıldız, W.Zhang** *Convex Hulls under Uncertainty*, Under preparation, 2014
- **S. Eriksson-Bique, J. Hershberger, V. Polishchuk, B. Speckmann, S. Suri, T. Talvitie, K, Verbeek, H. Yıldız,** *Geometric kth Shortest Paths*, Under preparation, 2014
- **S. Suri, K. Verbeek, H. Yıldız,** *On the Most Likely Convex Hull of Uncertain Points*, 21st European Symposium on Algorithms (ESA), 2013
- **J. Hershberger, S. Suri, H. Yıldız,** *A Near-Optimal Algorithm for Shortest Paths Among Curved Obstacles in the Plane*, 29th Symposium on Computational Geometry (SoCG), 2013
- **H. Yıldız, S. Suri,** *Computing Klee's Measure of Grounded Boxes*, Algorithmica Online, 2013
- **H. Yıldız, S. Suri,** *On Klee's Measure for Grounded Boxes*, 28th Symposium on Computational Geometry (SoCG), 2012
- **H. Yıldız, C. Kruegel,** *Detecting Social Cliques for Automated Privacy Control in Online Social Networks*, 4th Workshop on Security and Social Networking (SESOC), 2012
- **H. Yıldız, L. Foschini, J. Hershberger, S. Suri,** *The Union of Probabilistic Boxes: Maintaining the Volume*, 19th European Symposium on Algorithms (ESA), 2011
- **H. Yıldız, J. Hershberger, S. Suri,** *A Discrete and Dynamic Version of Klee's Measure Problem*, 23rd Canadian Conference on Computational Geometry (CCCG), 2011

Achievements and Awards

- **Dean's fellowship,** Univerity of California, Santa Barbara, 2013-2014
- **4th Place** in DEFCON CTF hacking competition as a member of Shellphish team, Las Vegas, USA, 2009
- **5th Place** (among 71 teams) at ACM-Programming Contest Southern California Regionals 2008 as a member of the team representing University of California Santa Barbara
- **Top Graduate** in Middle East Technical University, 2008
- **Best Senior Project** in METU Computer Engineering Senior Project Competition 2008 as a member of Opus 144
- **First Place** at METU National Programming Contests 2005-2008
- **Gökçe Karataş 2005 Education Award** for his accomplisments in the field of Informatics
- **Silver Medal** at 16th International Olympiad in Informatics, Athens, Greece, 11-18 September 2004
- **Silver Medal** at 12th Balkan Olympiad in Informatics, Plovdiv, Bulgaria, 3-9 July 2004
- **Gold Medal** at 11th National Olympiad in Informatics, Ankara, Turkey, December 2003
- **Bronze Medal** at 15th International Olympiad in Informatics, Wisconsin, USA, 16-23 August 2003
- **Bronze Medal** at 11th Balkan Olympiad in Informatics, Iai, Romania, 14-20 July 2003
- **Silver Medal** at 10th National Olympiad in Informatics, Ankara, Turkey, December 2002
- **5th Place** in 2001 Turkish National High School Exam among 560.000 students

Fields of Study

Major Field: Computational Geometry

Studies in Computational Geometry with Professor Subhash Suri

Abstract

Computing Volumes and Convex Hulls: Variations and Extensions

Hakan Yıldız

Geometric techniques are frequently utilized to analyze and reason about multi-dimensional data. When confronted with large quantities of such data, simplifying geometric statistics or summaries are often a necessary first step. In this thesis, we make contributions to two such fundamental concepts of computational geometry: Klee's Measure and Convex Hulls. The former is concerned with computing the total volume occupied by a set of overlapping rectangular boxes in d-dimensional space, while the latter is concerned with identifying extreme vertices in a multi-dimensional set of points. Both problems are frequently used to analyze optimal solutions to multi-objective optimization problems: a variant of Klee's problem called the Hypervolume Indicator gives a quantitative measure for the quality of a discrete Pareto Optimal set, while the Convex Hull represents the subset of solutions that are optimal with respect to at least one linear optimization function.

In the first part of the thesis, we investigate several practical and natural variations of Klee's Measure Problem. We develop a specialized algorithm for a specific case of Klee's problem called the "grounded" case, which also solves the Hypervolume Indicator problem faster than any earlier solution for certain dimensions. Next, we extend

Klee's problem to an uncertainty setting where the existence of the input boxes are defined probabilistically, and study computing the expectation of the volume. Additionally, we develop efficient algorithms for a discrete version of the problem, where the volume of a box is redefined to be the cardinality of its overlap with a given point set.

The second part of the thesis investigates the convex hull problem on uncertain input. To this extent, we examine two probabilistic uncertainty models for point sets. The first model incorporates uncertainty in the existence of the input points. The second model extends the first one by incorporating locational uncertainty. For both models, we study the problem of computing the probability that a given point is contained in the convex hull of the uncertain points. We also consider the problem of finding the most likely convex hull, i.e., the mode of the convex hull random variable.

Table of Contents

List of Figures

Introduction

Geometry, "the science of shapes", has an essential place in mathematics. Many subfields of mathematics are, directly or indirectly, involved with geometry. One might even argue that mathematics is mostly geometry itself, considering that a real number can be represented as a point on the number line. Inevitably, the field of algorithms, which can be viewed (to some extent) as the mathematics of computer science, has studied many problems dealing with geometry. In 1970s, the scientists who study such problems started to use the term "Computational Geometry" to categorize their work. Today, Computational Geometry is a widely recognized subfield of algorithms, with applications in many subdisciplines of computer science including but not limited to computer graphics, computer-aided design, databases, geographical information systems, machine learning and robotics.

A good deal of applications in computer science, including many in computational geometry, require computing statistics or simplifications of the data at hand, either as part of their main intent or in order to improve efficiency. As an example that relates

to geometry, one can consider certain collision detection algorithms used in computer graphics, which replace complicated objects (e.g., 3D meshes) with simpler shapes such as bounding boxes or enclosing spheres. In some cases, the sought statistic may even be as simple as a single number. For instance, (following the earlier example in computer graphics) one may want to compute the volume of a 3D mesh (which is a single number) in order to assign it a weight proportional to its volume and then make physical computations on it.

In this dissertation, we make contributions to computing two such statistics from geometry: volumes and convex hulls. Both computing volumes and computing convex hulls are operations that are frequently utilized in practice to simplify or summarize geometric data. As a result, computational geometers have studied both problems in various forms for decades, publishing a large amount of related work. This dissertation studies algorithms for computing these two statistics on various scenarios. The results that we present either improve the best results on old problems or provide non-trivial solutions for novel variations of these problems. Before we elaborate on the content of the dissertation, we give some brief information about computing volumes and convex hulls.

Volumes: Going back to the basics, the *volume* of an object is the amount of space it occupies. While the word "volume" normally refers to a measure of three-dimensional space, the concept applies in any number of dimensions. The one-dimensional vol-

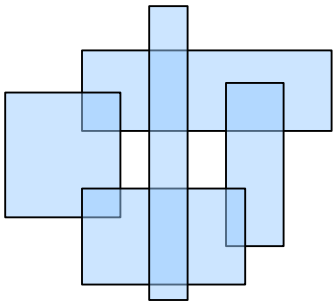


Figure 0.1: An instance of Klee's Measure Problem in two dimensions.

ume is called the *length* whereas the two-dimensional volume is called the *area*. In mathematical terminology, volume for dimensions higher than three is referred as *hypervolume*. In this dissertation, we use the term "volume" to refer to volume in any dimension, including length, area, three-dimensional volume and hypervolume.

Calculating volumes is one of the core operations in geometry, with countless applications in science and engineering. As a result, there is a significant amount of work in computational geometry literature that addresses computing volumes. Since it is straightforward to compute the volume of most simple shapes (e.g. triangles, ellipses, spheres, etc.), the focus of this work has been on collections of shapes or shapes with arbitrary descriptive complexity. Some of the studied problems involve computing the volume of a "union" of shapes (e.g., [5, 8, 56, 69]). In this dissertation, we mainly work on a specific problem in this class called Klee's Measure Problem, which is named after the famous mathematician Victor Klee. In this problem, one wants to compute the volume of a union of axis-aligned boxes in d-dimensional space, where d could be any

positive integer. In the two-dimensional special case of the problem (where $d = 2$), the aim is to compute the area covered by a union of axis-aligned rectangles. (See Figure 0.1.)

Despite that the main motivation for studying Klee's problem is theoretical, the problem is also useful in some practical scenarios. One well-known example is the *hypervolume indicator*, whose motivation comes from the field of multi-objective optimization [39, 48]. Conceptually, a feasible solution to a multi-objective optimization problem with d objectives is a point with positive coordinates in d-dimensional space, where its coordinates represent its evaluation with respect to the d objectives. Ideally, the goal of a multi-objective optimization algorithm is to produce the set of Pareto Optimal solutions: no point in the solution set is dominated by another point representing a feasible solution.[1] In many cases, however, only an approximate set is obtainable due to efficiency or feasibility reasons. The hypervolume indicator is a scalar measure used to evaluate how close an approximate solution set is to the true Pareto Optimal set. The indicator is simply equal to the volume in the positive orthant that is the dominated by the points in the approximate set. See Figure 0.2a for a two-dimensional example. Computing the hypervolume indicator can be seen as a special case of Klee's problem. In particular, the hypervolume indicator of n points, is equivalent to the volume of a

[1] A point p dominates another point q if all of p is larger than q on all coordinates.

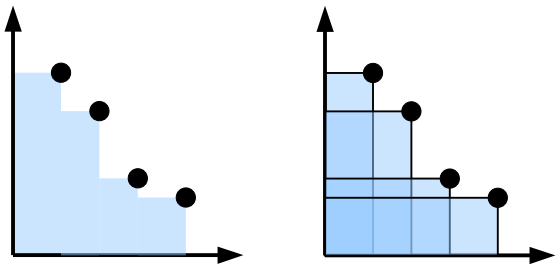


Figure 0.2: (a) The area dominated by a set of points in the positive quadrant of the plane. (b) The dominated area shown as a union of axis-aligned rectangles.

union of n boxes, where all boxes have one of the corners located on the origin. (See Figure 0.2b.)

Although Klee's Measure Problem has a long history in computational geometry (dating back to 1977 [56]), the only (unconditional) lower bound known for the problem is $\Omega(n \log n)$ for all dimensions, where n is the number of boxes in the input. The problem can be solved in optimal $O(n \log n)$ time for $d = 1, 2$ [12] and $O(n^{d/2})$ time for $d \geq 3$ [24], where n is the number of boxes. For the hypervolume indicator problem, faster specialized algorithms exist.

Convex Hulls: The second geometric concept that we study in this dissertation is a structure known as the convex hull. Given a set of points in d-space (for any positive integer d), their *convex hull* is the smallest convex set that contains all the points. If the points are given in the plane (i.e., $d = 2$), their convex hull would be the smallest convex polygon containing all of the points. (See Figure 0.3.)

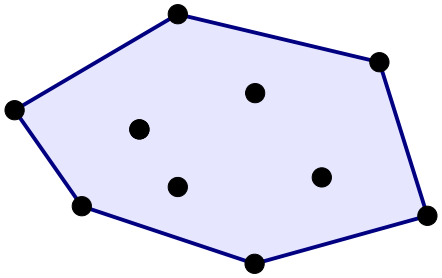


Figure 0.3: Convex hull of a set of points in two dimensions.

The convex hull is a fundamental structure in mathematics and computational geometry, with applications in a variety of areas including but not limited to computer graphics, image processing, pattern recognition and statistics. For instance, in computer graphics, convex hulls provide a more accurate simplification of 3D meshes compared to bounding boxes and can be used to accelerate collision detection in certain scenarios.

Convex hulls are also frequently utilized when dealing with optimization problems. The vertices of the convex hull represent a "Linear Optimization Pareto Front" among the input point set: each vertex is a point that is optimal with respect to at least one linear optimization function. Convex hulls also allow efficient queries of these optimal points for given linear optimization functions.

Owing to their importance in practice, the algorithms for computing convex hulls are well-studied and optimal worst-case algorithms have been discovered for all dimensions. The convex hull of a set of n points can be computed in $O(n \log n)$ time for

$d = 2, 3$ [46, 65] and $O(n^{\lfloor d/2 \rfloor})$ for $d > 3$ [25]. There also exist faster output-sensitive algorithms [21, 26, 55, 61, 67].

Contribution of the Dissertation

In this dissertation, we present new results on computing volumes and convex hulls. In particular, we study several interesting and practical problems involving volumes and convex hulls and either solve these problems with novel algorithms or prove hardness results on them. Most of the problems that we consider are variations or extensions of Klee's Measure and Convex Hull problems. To the best of our knowledge, we are the first to study these variations/extensions. We also study a special case of Klee's Measure Problem. In that case, our results imply a faster solution compared to earlier (more general) solutions.

We emphasize that the problem variations we work on are not arbitrarily chosen. In addition to that we study theoretically interesting and (potentially) practical problems, it is worth to point out two aspects of our work. First, all of the problems are studied for an arbitrary number dimensions of Euclidean space. In other words, we do not limit our solution techniques to two and three dimensions where the data is easily "visualizable", but instead develop solutions that extends to higher dimensions whenever possible. Such high-dimensional geometric techniques are frequently utilized in prac-

tice, especially in areas such as databases, machine learning and data mining. Thus, our techniques have potential applications in these fields.

The second major aspect of our work is that most of our problems focus on or can be extended to involve uncertainties in input. In many practical scenarios, the data of concern may be subject to uncertainties. For instance, if the data is collected by a sensor device, there may be measurement errors. In some cases, the uncertainty is intentionally introduced by perturbing data (e.g., data anonymization for privacy protection). Depending on the cause of the uncertainty, it may be possible to model it probabilistically. For instance, one can gather statistical information from past certain data to infer a probability distribution on the uncertain data. It may also be possible infer such a distribution directly from the process generating the data. If such probabilistic information is available, then one can extract more meaningful information about the effects of uncertainties on the output.

To better understand the nature of our work on uncertain data, consider the following example on the convex hull problem. Suppose that we are given a set of points in the plane, but we are uncertain about the existence of each point and only know it probabilistically. In particular, assume that each point is assigned an independent probability of existence. That is, we know, for each point p, the likelihood that p really exists. Under these conditions, the convex hull becomes a random variable. It is then possible

to analyze various statistics of the convex hull. For instance, what is the expected area of the convex hull? One can also ask for the most likely convex hull of the points.

Organization of Chapters

The dissertation consists of two main parts (Chapters 1-3 and Chapters 4-5). In Chapters 1 to 3, we study variations of Klee's Measure Problem. In Chapters 4 and 5, we study convex hulls under uncertainty. We now give detailed information about the content of our chapters.

In Chapter 1, we study a special case of Klee's problem called the "grounded" case. Given a set of boxes in d-space, we say that the boxes are *k-grounded*, if all boxes have a common corner point when they are projected to a particular subset of k dimensions ($1 \leq k \leq d$). Based on this definition, the hypervolume indicator problem is the d-grounded case of Klee's Measure Problem. We show that if the boxes are k-grounded for any $k \geq 2$, then the volume of their union can be computed in $O(n^{(d-1)/2} \log^2 n)$ time, which is roughly $\sqrt{n}$ faster than the general upper bound of $O(n^{d/2})$. Our algorithm also solves the hypervolume indicator problem faster than any previously published algorithm for dimensions $d = 4, 5, 6$.

In Chapter 2, we study Klee's Measure Problem on uncertain boxes. In particular, we examine the problem of computing volume in the case where each box is known to exist with an independent probability. In this setting, we describe a data structure

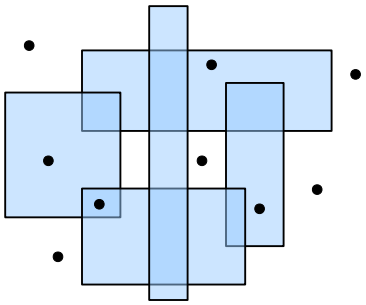


Figure 0.4: An arrangement of boxes and points in two dimensions. The discrete volume of the union of boxes with respect to the point set is 4.

that can maintain the *expected volume* as the set of boxes undergoes insertions and deletions. The data structure has constant query time and an amortized update time of $O(n^{(d-1)/2} \log n)$. Surprisingly, we also show that querying the probability distribution of the volume for a specific value is an NP-hard problem despite that the expected volume can be computed in polynomial time.

Chapter 3 examines a discrete version of Klee's Measure Problem. In this version, we redefine the volume of each box to be the number of points it contains from a given point set. Then, the volume of a union of boxes is simply the number points that are contained by at least one box. (See Figure 0.4 for a two-dimensional example.) The discrete measure problem have natural applications in areas that deal with multi-attribute data. For instance, in databases, it can be used to compute the number of records that satisfy a disjunction of ranges queries. As our solution to this problem, we describe a dynamic data structure that can maintain the discrete volume of a set of boxes with respect to a set of points, as both sets undergo modifications. This structure has

constant query and sublinear update times. We also extend our result to a probabilistic setting.

In Chapter 4, we present our first set of results for the convex hull problem under uncertainty. Our results are based on two probabilistic uncertainty models. In the first model, the uncertainty is existential only. In particular, each point has a fixed position but exists with an independent probability. In the second model, we also incorporate locational uncertainty and allow each point to appear in a number different positions based on a probability distribution. For both settings, we examine the problem of finding the probability that a given point is contained in the convex hull and solve it with polynomial algorithms. We also describe a data structure that can efficiently report the probability that a given query point is included in the convex hull of a set of uncertain points in two dimensions.

Chapter 5 considers the problem computing the "most likely convex hull", the mode of the convex hull variable, based on the same uncertainty models from Chapter 4. The most likely convex hull can be utilized in practice as an estimator for the random convex hull variable. We show that the most likely hull of a set of n uncertain points can be computed in $O(n^3)$ if the points are in two-dimensional space and their uncertainties are existential only. For dimensions higher than two or in the presence of locational uncertainty, we prove that computing the most likely hull is an NP-hard problem and also show an inapproximability result.

Chapter 1

Computing Klee's Measure of Grounded Boxes*

1.1 Introduction

In this chapter, we study algorithms for computing the volume of a union of axis-aligned boxes, for a special case that we call "grounded boxes". As mentioned before, computing the volume of a union of boxes is known as Klee's Measure Problem. Since our problem is a special case of Klee's Measure Problem, we first give some brief information about Klee's problem before we elaborate on our problem and results.

*This chapter is based on a joint work with Subhash Suri and parts of this chapter appeared in the following publications: [80] (Published and copyright held by ACM. The definitive version available at http://doi.acm.org/10.1145/2261250.2261267.) [81] (Published and copyright held by Springer. The final publication available at http://link.springer.com/10.1007/s00453-013-9797-9.)

1.1.1 Klee's Measure Problem

Klee's Measure Problem is a classical problem in computational geometry, dating back to the 1970s: Given a set of n axis-aligned boxes in d-space, compute the volume of their union. In the two-dimensional special case of the problem (where $d = 2$), the problem asks for the area of a union of axis-aligned rectangles. Similarly, in three dimensions, the aim is to compute the volume of a union of axis-aligned rectangular prisms.

In the literature, this problem was first studied by Victor Klee in 1977 [56] in its one-dimensional form and hence is named after him. Klee described how to compute the length of a union of n segments on the number line in $O(n \log n)$ time and pointed out that the problem extends to dimensions higher than one. The first non-trivial solution for higher dimensions, with running time $O(n^{d-1} \log n)$, was given by Bentley using his space-sweep approach [12], which was quickly improved to $O(n^{d-1})$ time by van Leeuwen and Wood [76]. Several years later, Overmars and Yap [64] made a breakthrough and achieved the bound of $O(n^{d/2} \log n)$, which essentially remains unbeaten more than 20 years later. All improvements to this bound during these intervening years have come in the form of reducing the log factor [23, 24]. Today, the best upper-bound for Klee's problem is $O(n \log n)$ for $d = 1, 2$ [12] and $O(n^{d/2})$ for $d \geq 3$ [24].

The only lower bound known for the problem is $\Omega(n \log n)$ for all dimensions, in the linear decision tree model [41]. This is quite unsatisfactory, especially considering

that there is a big cap between the upper and lower bounds in high dimensions. On the other hand, the problem is known to be $\#P$-hard when the dimension is part of the input [17], so the exponential dependence on the dimension seems unavoidable for an algorithm with running time polynomial in n. In fact, the problem has also been shown to be $W[1]$-hard [23], meaning that it is unlikely (based on commonly accepted conjectures) that an algorithm with time complexity $O(f(d) \times P(n))$ exists, where $f(d)$ is a function of d and $P(n)$ is a polynomial of n independent of d.

Since beating the $O(n^{d/2})$ bound proved to be difficult task over the years, scientists have also studied special cases of Klee's problem, with hopes to get better bounds. The most well-studied special cases are the ones where the boxes are in the form of hypercubes [2, 5, 15, 22, 24, 31]. For $d \geq 4$, the best bound for computing the volume of a union of hypercubes is $O(n^{(d+1)/3} \operatorname{polylog} n)$, which can further be improved to $O(n^{d/3} \operatorname{polylog} n)$ if the hypercubes have unit size [24]. Notice that this is significantly better than the general $O(n^{d/2})$ bound. For $d \leq 3$, it is possible to solve the hypercube problem in $O(n \operatorname{polylog} n)$ time [2].

Another well-studied special case of Klee's problem is known as the *hypervolume indicator*, which is a metric frequently used in multi-objective optimization and evolutionary computing [39, 48, 83]. In this special case, all boxes have one of their corners located on the origin and extend towards the positive orthant. This special case can be reduced to Klee's problem on unit hypercubes and therefore be solved in

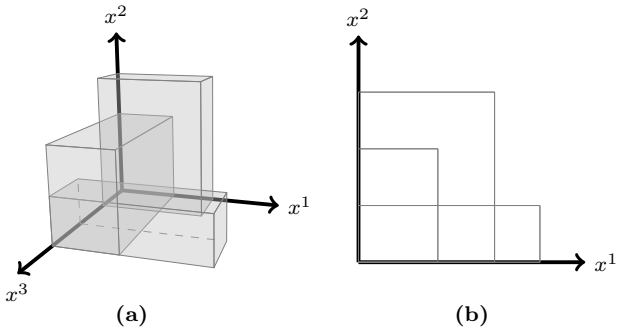


Figure 1.1: A set of three boxes in 3-space are shown in (a). The coordinate axes are labeled as x^1, x^2 and x^3. These boxes are 2-grounded because when they are projected to the 2-dimensional space formed by axes x^1 and x^2 (as shown in (b)), they share a common bottom-left corner, which is the origin.

$O(n^{d/3} \operatorname{polylog} n)$ for $d \geq 4$. For smaller dimensions, algorithms with $O(n \log n)$ running time exist [14, 40].

Finally, there also exist some work on streaming and space-efficient algorithms for Klee's Measure Problem [27, 68, 71].

1.1.2 Problem and Results

The main contribution of this chapter is a new result for computing Klee's measure under the assumption that a 2-dimensional orthogonal projection of all the boxes has a common corner. We call such a collection of boxes 2-*grounded*. (See Figure 1.1 for an example.) When the boxes share a common corner in a k-dimensional projection, for $1 \leq k \leq d$, we call the set k-*grounded*. Naturally, the algorithm for 2-grounded boxes work trivially for k-grounded, where $k \geq 2$.

Besides their intrinsic theoretical interest as a special case of the general Klee's problem, the k-grounded boxes also arise frequently in applications when some of the coordinate axes have a natural "start" or "end" position for ranges. For instance, if one of the axes represents time recording the duration of some event, then the origin marks the natural start point, and therefore a source of grounding on that axis. Similarly, in sensor databases, physical attributes such as temperature, humidity etc. naturally have a common "lower bound," and the measured quantity is really the deviation from this "grounded" value. In these cases, as long as two or more dimensions are grounded, our algorithm leads to an improved bound for computing the volume of the union.

Our main result is an $O(n^{(d-1)/2} \log^2 n)$ time algorithm for computing Klee's measure for a set of n 2-grounded boxes. This is an improvement of roughly $O(\sqrt{n})$ compared to the fastest solution of the general problem. Recently, it has been shown that breaking the bound $O(n^{(d-1)/2})$ for 2-grounded boxes would imply a breakthrough for the general Klee's problem [16]. Assuming the correctness of the conjecture that $\Omega(n^{d/2})$ is the right lower bound for Klee's problem, our algorithm is optimal for 2-grounded boxes, with the exception of its logarithmic factors.

If the boxes are d-grounded, namely they all share a common corner (without any projection), the time complexity of our algorithm is further improved by a $\log n$ factor. Recall that the hypervolume indicator problem deals with boxes with a common corner (i.e., they are d-grounded), so our algorithm also works for computing the hypervol-

ume indicator. We note that our algorithm is asymptotically faster than any previously published hypervolume indicator algorithm for dimensions 4, 5 and 6.

At the high level, our algorithm has the following general form. We transform the d-dimensional problem into one of maintaining its $(d-1)$-dimensional cross-section with a sweeping plane. The sweeping is a standard step used in algorithms for the Klee's measure, but instead of solving the cross-section problem directly in $(d-1)$-space, we exploit the grounding property and transform the problem into a $(d-2)$-dimensional *weighted volume* problem. In this problem, each box has a non-negative weight and, in computing the volume, the contribution of each point in space equals the weight of the heaviest box containing it. Solving the weighted volume problem efficiently is the main result of this chapter. In particular, we show that the d-dimensional weighted volume can be maintained under insertions at the amortized cost of $O(n^{(d-1)/2}\log^2 n)$, which results in an $O(n^{(d-1)/2}\log^2 n)$ time algorithm for the d-dimensional Klee's Measure Problem on 2-grounded boxes. In solving the weighted volume problem, we also introduce and solve a combinatorial problem of maintaining the *sum of ordered products*, which may be of independent interest.

Chapter Organization

The chapter is organized in seven sections. In Section 1.2, we explain how to reduce the d-dimensional problem to a $(d-2)$-dimensional weighted volume problem.

In Sections 1.3 and 1.4, we solve a special case of the weighted volume involving *half-spaces*, which turns out to be the key problem. In Section 1.5, we show how the general weighted problem can be solved based on the solution of the halfspace problem. In Section 1.6, we briefly mention our results for the hypervolume indicator. In Section 1.7, we conclude with a summary and discussion.

1.2 Dimension Reduction: Sweeping and Weighting

The classical approach of Overmars and Yap [64] solves the d-dimensional Klee's Measure Problem by reducing it to a dynamic $(d-1)$-dimensional problem. Our key idea is to reduce the d-dimensional problem to a $(d-2)$-dimensional *weighted* volume problem with *insert-only* updates. We reduce one dimension by plane sweep (a standard technique used in previous algorithms for Klee's Measure [12, 23, 64, 76]), and another by converting the unweighted problem to a weighted problem. We begin with the high level ideas behind the plane sweep, and the weighting.

Reducing a Dimension by Plane Sweep: We first briefly introduce some formal definitions. A d-dimensional axis-aligned box B is the Cartesian product of d one-dimensional ranges, namely, $B = \Pi_{i=1}^{d}(a_i, b_i)$. The *volume* of a box B is $vol(B) = \Pi_{i=1}^{d}|b_i - a_i|$. Given a set $\mathcal{B}$ of boxes $\{B_1, B_2, \ldots, B_n\}$, its *union* $\bigcup B_i$ is the set of points contained in at least one box of $\mathcal{B}$.

Consider a set of axis-aligned boxes $\mathcal{B} = \{B_1, \ldots, B_n\}$ in d-space, for which we want to compute the volume of their union. Without loss of generality, we assume that all the boxes are contained in the positive orthant $\mathbb{R}^d_+$—otherwise, we can divide the problem into 2^d groups, one for each orthant. We say that a box B is *grounded* with respect to dimension k if B's extent along the kth dimension has the form $(0, B^k)$, where $B^k > 0$ is the length of B along dimension k. Supposing that all boxes in $\mathcal{B}$ are grounded with respect to dimension d, we can compute the volume of their union as follows. We sort the boxes in $\mathcal{B}$ in the descending order of their dth coordinate. Without loss of generality, let the resulting order be $B_1, B_2, \ldots, B_n$, meaning that $B_1^d > B_2^d > \cdots > B_n^d > 0$. We further set $B_{n+1}^d = 0$. Then, it is easy to see that the total volume covered by $\mathcal{B}$ is given by the formula

$$\sum_{i=1}^{n} V_{i,d-1} \times (B_i^d - B_{i+1}^d),$$

where $V_{i,d-1}$ is the volume of the set $\{B_1, \ldots, B_i\}$ *projected* onto the plane $\mathbf{x}^d = 0$.[2] This is easily seen by visualizing a plane parallel to the plane $\mathbf{x}^d = 0$, sweeping the space from B_1^d to $B_{n+1}^d = 0$, and by observing that the intersection of this plane with the boxes of $\mathcal{B}$ is constant between two consecutive coordinates in the sorted list, determined entirely by the boxes that precede B_i. If we write $V_{i,d-1}$ for the $(d-1)$-

[2] $\mathbf{x}^d$ denotes the dth coordinate of point $\mathbf{x}$.

dimensional volume of this intersection, then the d-dimensional volume that lies between B_i^d and B_{i+1}^d is precisely $V_{i,d-1} \times (B_i^d - B_{i+1}^d)$.

The $(d-1)$-dimensional slice changes only when the sweeping plane encounters a new box B_i, and thus we can compute the d-dimensional volume of n boxes by maintaining the $(d-1)$-dimensional volume of their projections subject to n insert-only updates. If the *amortized* update and query cost is $O(F(n))$, then the d-dimensional problem can be solved in time $O(nF(n))$ plus the preprocessing cost including the initial sorting.

Reducing a Dimension by Weighting: Another conceptual way to reduce the dimension of the volume problem is the following. Assuming again that the boxes of $\mathcal{B}$ are grounded with respect to the dth dimension, we orthogonally project each box B_i in $\mathcal{B}$ onto the plane $\mathbf{x}^d = 0$, obtaining a $(d-1)$-dimensional box, denoted B_i', and assign a *weight* $w(B_i') = B_i^d$ to this projected box. The *weighted volume* of B_i' is defined as $w(B_i')$ times the (ordinary, $(d-1)$-dimensional) volume of B_i'. See Figure 1.2 for illustration.

This transformation converts the set $\mathcal{B}$ into a set $\mathcal{B}' = \{B_1', B_2', \ldots, B_n'\}$ of $(d-1)$-dimensional weighted boxes. For each point $\mathbf{x} \in \mathbb{R}_+^{d-1}$ on the plane $\mathbf{x}^d = 0$, assign to it the weight of the heaviest box containing $\mathbf{x}$. That is,

$$w(\mathbf{x}) = \max\{w(B_i') \mid \mathbf{x} \in B_i'\}$$

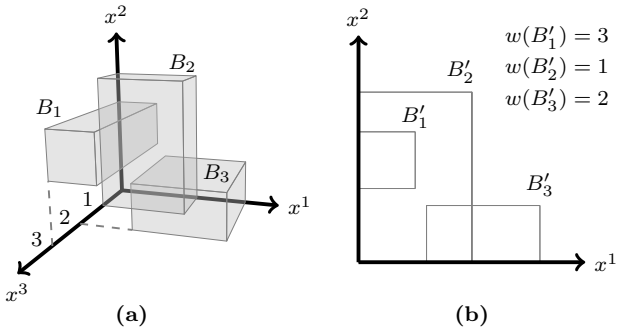


Figure 1.2: (a) A set of 3 boxes B_1, B_2, B_3 in 3-space, grounded with respect to the x^3 axis, and (b) their 2-dimensional weighted projections.

Then the volume covered by the boxes in $\mathcal{B}$ is given by the integral

$$\int_{\mathbf{x}\in\mathbb{R}^{(d-1)}} w(\mathbf{x})\, \mathrm{d}\mathbf{x}$$

We call this expression the *weighted volume* of the union of the set $\mathcal{B}'$. Intuitively, this expression weights each point $\mathbf{x}$ on the projection plane by the maximum *height* of a box in $\mathcal{B}$ that contains it, thus correctly computing the volume contribution of each box in $\mathcal{B}$.

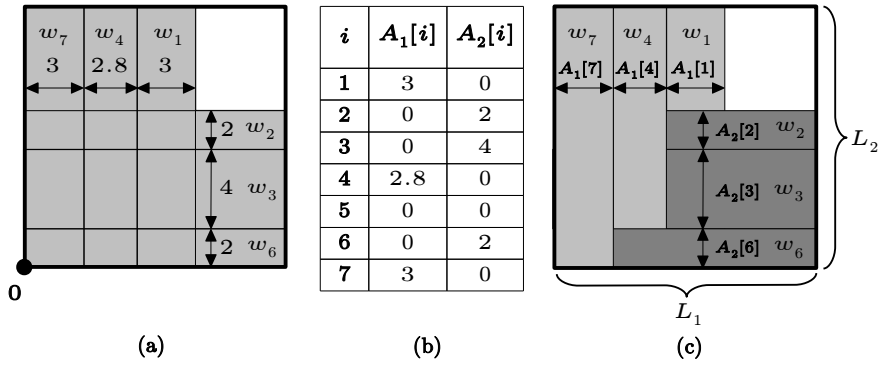
Joint Sweep and Weighting for 2-grounded Boxes: Suppose $\mathcal{B}$ is a set of boxes, grounded along dimensions d and $d-1$, and that their orthogonal projection onto the dimensions $(d-1)$ and d has the origin as the common corner. We can now apply both of the above dimension-reduction techniques simultaneously, as follows. We sweep the space along the dth dimension, which requires us to dynamically maintain the $(d-1)$-dimensional volume of the set of boxes intersecting the sweep plane. The updates are *insert only* because boxes are only inserted, and never deleted. Depending on that

the boxes are grounded with respect to $(d-1)$th dimension, we maintain the dynamic $(d-1)$-dimensional volume by converting it to a *weighted* $(d-2)$-dimensional volume.

Thus, the d-dimensional Klee's Measure Problem is transformed into a $(d-2)$-dimensional problem of maintaining the weighted volume under insertion of boxes. Solving this problem efficiently is the most essential part of our algorithm, and the focus of the next three sections. In fact, the crux of the problem proves to be the following special case:

> *Under insert-only updates, maintain the weighted volume of a set of axis-parallel halfspaces.*

Our algorithm needs to compute the weighted volume of axis-parallel *strips*, not halfspaces. However, the solution is easier to describe for the halfspace case, and generalizes to strips with minor modifications that incur only an extra $\log n$ factor in time complexity. Therefore, in the next two sections, we focus on halfspaces, and then return to the weighted volume of arbitrary boxes in Section 1.5. For ease of presentation, we first describe the halfspace algorithm in two dimensions (Section 1.3), and then its generalization to higher dimensions (Section 1.4). Section 1.5 describes how to solve the general weighted volume by combining the halfspaces problem with a space partition technique.



i	$A_1[i]$	$A_2[i]$
1	3	0
2	0	2
3	0	4
4	2.8	0
5	0	0
6	0	2
7	3	0

Figure 1.3: (a) Illustrating the gradient structures. The vertical gradient has three strips, with weights in descending order $w_7 > w_4 > w_1$, and widths $3, 2.8$ and 3, respectively. (b) The array representation of the two gradients shown in (a). The halfplane H_5 is either not inserted yet or covered by halfplanes with higher weights, and so its array entries are zero. (c) Illustrating the "intersection" of the two gradients. The "light gray" (resp., "dark gray") section shows the portion where the weight of the vertical (resp., horizontal) gradient dominates. Recall that the halfplane with larger index has higher weight than the one with smaller index.

1.3 Weighted Volume of Halfspaces: the Planar Case

Consider a set $\mathcal{H}$ of n axis-parallel weighted halfplanes in 2-space, each containing the origin, and a positive orthant axis-aligned rectangle R anchored at the origin. For a point $\mathbf{p}$ in R, define the weight of $\mathbf{p}$ with respect to a subset $\mathcal{H}' \subseteq \mathcal{H}$, denoted $w(\mathbf{p}, \mathcal{H}')$, as the weight of the heaviest halfplane in $\mathcal{H}'$ that contains $\mathbf{p}$; if no such halfplane exists, then the weight is zero. The weighted volume (area) of $\mathcal{H}'$ over R is the integral $\int_{\mathbf{p}\in R} w(\mathbf{p}, \mathcal{H}')\ \mathrm{d}\mathbf{p}$. Our goal is to maintain the weighted volume as $\mathcal{H}'$ undergoes insert-only updates. We will show a data structure with amortized cost of $O(\log n)$ per insertion. The set of axis-aligned halfplanes in $\mathcal{H}'$ is naturally divided into two groups: vertical and horizontal. The vertical halfplanes have the form $\{\mathbf{x} \in \mathbb{R}^2 \mid 0 \leq \mathbf{x}^1 \leq a\}$,

and the horizontal ones have the form $\{\mathbf{x} \in \mathbb{R}^2 \mid 0 \leq \mathbf{x}^2 \leq b\}$, where a and b are arbitrary reals. We maintain the weight distribution imposed by these two classes separately, and then show how to compute the joint weight implicitly and efficiently.

1.3.1 Vertical and Horizontal Gradients

The intersection of R with a halfplane H is a "strip," either vertical or horizontal, containing the origin. Let us focus on the vertical halfplanes of $\mathcal{H}'$, and consider the *partition* they induce on R where each point of R is "claimed" by the maximum weight halfplane containing it. This partition is a sequence of vertical strips in which each strip belongs entirely to one halfplane, each halfplane contributes at most one strip, and the strips are ordered in *descending* weight order from left to right. This follows because all halfplanes contain the origin, and a larger weight halfplane completely overrides the smaller weight halfplane to its left. Visually, the resulting structure looks like a "waterfall", and for ease of reference, we call it the *vertical gradient*.[3] Similarly, the horizontal halfplanes, considered in isolation, induce a *horizontal gradient* of strips ordered in descending order of weights from bottom to top. (Figure 1.3a shows an example, where the vertical gradient consists of three strips of respective weights $w_7 > w_4 > w_1$, and respective widths 3, 2.8, and 3.)

[3]This name is inspired by color gradients, which are images with decreasing color intensity in one direction.

The gradients give a nice and compact representation of the weight structure imposed by the vertical and the horizontal halfplanes *separately*. In order to compute the weighted volume of $\mathcal{H}'$ over R, however, we need to *intersect* the two gradients. An explicit intersection of two size n gradient structures entails $\Omega(n^2)$ complexity [64], and so the key is to perform this intersection *implicitly*.

Let w_i be the weight of the ith halfplane $H_i \in \mathcal{H}$, and assume that the halfplanes are ordered in increasing order of their weights. Without loss of generality, we assume that the weights w_i's are distinct. We maintain two arrays, A_1 and A_2, storing the *widths* of the strips contributed by vertical and horizontal halfplanes, respectively. Both arrays have size n, and the ith entry of each corresponds to the halfplane H_i. The entry $A_1[i]$ or $A_2[i]$ stores the width of the strip contributed by H_i: if H_i is vertical, it contributes to $A_1[i]$, otherwise to $A_2[i]$. (A halfplane contributes to neither gradient if it is dominated by heavier halfplanes, in which case both entries are zero.) (See Figure 1.3b for an illustration.)

Let L_1, L_2 be the dimensions of the rectangular region R. Then it is easy to see that $\sum_i A_1[i] \leq L_1$ and $\sum_i A_2[i] \leq L_2$. (The vertical strips whose widths populate A_1 are disjoint, and their sum cannot exceed the width of R.) Furthermore, *considering each gradient in isolation*, the weighted volume contribution of H_i is precisely equal to $w_i \times (A_1[i] \cdot L_2 \ + \ A_2[i] \cdot L_1)$: this follows because only one of $A_1[i]$ or $A_2[i]$ can be non-zero, and so this term is precisely the weighted area of the rectangular strip of H_i.

The next problem is to determine how much of each gradient is claimed by the other. We do that in the next subsection, but first let us consider how to maintain the arrays A_1 and A_2 under *insertion of new halfplanes*. Consider an update to A_1; the horizontal case A_2 is entirely symmetric. When a vertical halfplane H, with weight w, is to be inserted, we first determine its *rank*, namely, the *index* i for the weight w in the ordered sequence $w_1, \ldots, w_n$, which can be done in $O(\log n)$ time. Suppose the rank of H is i, namely, $H \equiv H_i$. Two changes occur in the gradient structure: (1) some of the vertical strips that overlap with H_i are deleted—in particular, those with weights less than w_i, and (2) the strip containing the vertical line defining H_i "shrinks" in width. We can locate the strip to be shrunk in logarithmic time by maintaining a binary search tree on the strips, keyed by their position, and then appropriately update its array value. Starting with that strip, we then traverse the list of vertical strips to the left, deleting each strip as long as their weights are less than w_i. These strips are deleted also from the search tree, in logarithmic time per strip. (We note that once a strip is deleted from the gradient, it is never reinserted.) We can easily compute the width of the strip produced by H_i and write this value to $A_1[i]$, leading to the following lemma.

Lemma 1.3.1. *The arrays A_1 and A_2 representing the vertical and horizontal gradients can be updated in $O(\log n)$ amortized time after the insertion of a halfplane. The number of array entries whose values change is $O(1)$ amortized.*

1.3.2 Intersecting Gradient Volumes via Partial Sums

The key problem now is to deduce the correct weighted volume by overlapping the two gradients. Any point that is covered by both the vertical and the horizontal gradients should only be counted once, and receive the weight of the heavier halfplane containing it. In particular, let V_1 denote the weighted volume *contributed* by the vertical gradient: this is the weighted sum over the points where the vertical halfplanes prevail. Similarly, V_2 is the contribution of the horizontal gradient. We claim that V_1 has the following form:

$$V_1 \;=\; \sum_{i=1}^{n} \left(w_i \times A_1[i] \times \left(L_2 - \sum_{j=i+1}^{n} A_2[j] \right) \right)$$

This follows because $\sum_{i=1}^{n} w_i \cdot A_1[i] \cdot L_2$ is the weighted volume *without* considering the horizontal gradient, and the subtracted term is precisely what the horizontal gradient claims away from vertical strips. More precisely, for the halfplane H_i, the portion claimed by the horizontal gradient involves only those strips whose weight is larger than w_i, and if such a strip has "thickness" $A_2[j]$, then the area claimed by the vertical strip away from H_i is $A_1[i] \cdot A_2[j]$. (See Figure 1.3c.) By subtracting the total over all such horizontal strips leaves the portion that H_i contributes to the final weighted volume. The complementary term V_2 has the similar form:

$$V_2 \;=\; \sum_{i=1}^{n} \left(w_i \times A_2[i] \times \left(L_1 - \sum_{j=i+1}^{n} A_1[j] \right) \right)$$

We need to maintain these weighted volumes under insert-only updates, which we do by using a dynamic partial sums structure. Recall that the *partial sums* problem is the following:

> Maintain an array $\mathcal{A}$ of size n under an intermixed sequence of *update* and *query* operations, where $update(i, \Delta)$ changes $A[i]$ to $A[i] + \Delta$, and $query(k)$ reports the partial sum $\sum_{i=1}^{k} A[i]$.

Using a balanced binary tree, one can easily support these operations in $O(\log n)$ time each using linear space. The partial sums structure allows us to maintain our weighted volumes V_1 and V_2 as the gradient structures change due to insertion of new halfplanes. In particular, when a halfplane insertion changes $A_1[i]$ by Δ, the value V_1 changes by

$$w_i \times \Delta \times \left(L_2 - \sum_{j=i+1}^{n} A_2[j] \right)$$

This requires computing the partial sum $\sum_{i=j+1}^{n} A_2[j]$, which is easily done by computing the partial sum $query(j)$, and subtracting it from $query(n)$, in $O(\log n)$ time. On the other hand, if the halfplane H changes the entry $A_2[j]$ by Δ, then the change in V_1 is

$$-\left(\Delta \times \sum_{i=1}^{j-1} w_i \times A_1[i] \right)$$

By maintaining a partial sums structure for the array $A[i] = w_i \times A_1[i]$, this update can also be implemented in logarithmic time. In summary, the disjoint weighted volume

contributions V_1 and V_2 of the two gradients can be maintained at the cost of $O(\log n)$ time per update, and we have the following key result.

Theorem 1.3.2. *The weighted volume of a set of axis-aligned halfplanes in a rectangle R can be maintained in $O(\log n)$ amortized time per insertion, with a linear-space data structure.*

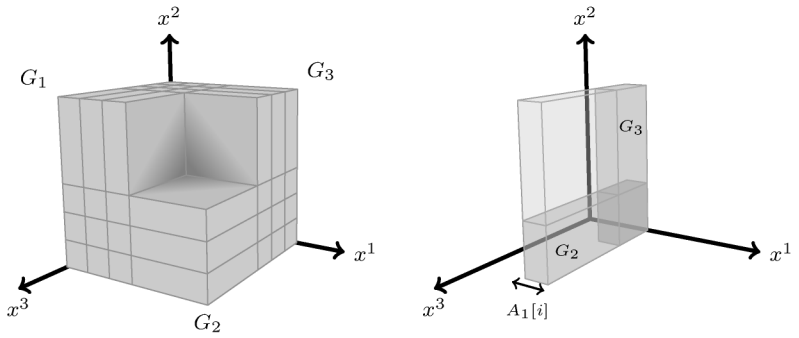
Proof. We use the array data structure described above, with the total weighted volume maintained explicitly, so the query time is $O(1)$. By Lemma 1.3.1, inserting a new hyperplane into the array representations requires $O(\log n)$ amortized time, and affects $O(1)$ array entries. For each affected entry, we update two partial-sum structures (one each for V_1 and V_2) and then compute the change in V_1 and V_2, which suffices to recompute the new weighted volume, in $O(\log n)$ time via partial-sums queries. Thus, the total amortized cost of an insertion is $O(\log n)$. Finally, all data structures consume linear space. □

1.4 Multidimensional Weighted Volume

In dimension $d > 2$, the basic idea is the same: we organize the halfspaces in d independent gradient structures, which can be updated efficiently when a new halfspace is added. The key difference from the two-dimensional problem arises in how we compute the final weighted volume by intersecting the d gradients structures. Unlike

the planar case, where we only needed to compute partial sums, the higher dimensional problems involves a more complex *sum of ordered products*. We discuss the details of these sums in the next two subsections, but first let us express the general form of the d-dimensional gradients.

We have a set $\mathcal{H}$ of n axis-parallel weighted halfspaces, each containing the origin, in d-space. We also have an orthogonal region R in the positive orthant, anchored at the origin. We wish to maintain the weighted volume of a subset $\mathcal{H}' \subseteq \mathcal{H}$ over R, under insertions of halfspaces from $\mathcal{H}$. We maintain d gradient structures, where the jth structure, denoted G_j, is formed by the halfspaces normal to the jth axis, for $j = 1, 2, \ldots, d$. The gradient G_j consists of strips, in descending weight order, along the positive $\mathbf{x}^j$ direction. (See Figure 1.4a for a three-dimensional example.) We represent G_j as an array A_j of size n, where the entry $A_j[i]$ contains the width of the strip contributed by the halfspace H_i to the gradient G_j, with weight w_i. The array indices are arranged in the increasing weight order of the halfspaces. Each of these d arrays can be maintained at the amortized cost of $O(\log n)$, per insertion of a halfspace, requiring modifications to a constant (amortized) number of array entries, using the technique of the previous section.



(a) (b)

Figure 1.4: (a) A three-dimensional halfspace arrangement forming three gradients G_1, G_2 and G_3. (b) The ith strip on G_1 (light gray) and the portion of its volume claimed by gradients G_2 and G_3 (areas with darker gray).

1.4.1 Intersecting the Gradients

Let us now consider how to maintain the weighted volume of the current set of halfspaces $\mathcal{H}'$, given the d gradient arrays. Write the weighted volume as the sum $V_1 + \cdots + V_d$, where V_j is the contribution by G_j to the total volume. We show how to maintain V_1; the others are analogous. In the absence of the other gradients, the weighted volume induced by a strip in G_1 is simply the product of its weight, its width and the $(d-1)$-dimensional volume of its projection on the plane $\mathbf{x}^1 = 0$. Formally, the weighted volume of ith strip is $w_i \times A_1[i] \times \prod_{\alpha=2}^{d} L_\alpha$, where L_α is the length of R along the αth coordinate axis. But some of this volume is *lost* to other gradients because of their higher weight. In particular, for a nonempty set $S = \{\alpha_1, \ldots, \alpha_m\} \subseteq \{2, \ldots, d\}$, let $Loss_i(S)$ be the weighted volume of the ith strip in G_1 that is claimed by all of the

gradients in the set $\{G_{\alpha_1}, \ldots, G_{\alpha_m}\}$. In other words, $Loss_i(S)$ is the intersection of the weighted volumes (on the ith strip of G_1) claimed by the gradients $G_{\alpha_1}, \ldots, G_{\alpha_m}$. We can write $Loss_i(S)$ as $w_i \times A_1[i]$ times the $(d-1)$-dimensional volume on the plane $\mathbf{x}^1 = 0$ that is covered by the intersection of heavier subsections of the gradients $G_{\alpha_1}, \ldots, G_{\alpha_m}$. (See the three-dimensional example in Figure 1.4b.) Formally, we write

$$Loss_i(S) = w_i \times A_1[i] \times \prod_{\alpha \in \{2,\ldots,d\} \setminus S} L_\alpha \times \sum_{i < j_{\alpha_1} \leq n} A_{\alpha_1}[j_{\alpha_1}] \times \cdots \times \sum_{i < j_{\alpha_m} \leq n} A_{\alpha_m}[j_{\alpha_m}]$$

Let $Loss(S)$ be the portion of G_1's weighted volume (over all of its strips) that is claimed by the intersection of the set of gradients $\{G_{\alpha_1}, \ldots, G_{\alpha_m}\}$. Then, we clearly have

$$\begin{aligned} Loss(S) &= \sum_{1 \leq i \leq n} Loss_i(S) \\ &= \sum_{1 \leq i \leq n} \left(w_i \times A_1[i] \times \prod_{\alpha \in \{2,\ldots,d\} \setminus S} L_\alpha \times \sum_{i < j_{\alpha_1} \leq n} A_{\alpha_1}[j_{\alpha_1}] \times \cdots \times \sum_{i < j_{\alpha_m} \leq n} A_{\alpha_m}[j_{\alpha_m}] \right) \\ &= \prod_{\alpha \in \{2,\ldots,d\} \setminus S} L_\alpha \times \sum_{1 \leq i < j_{\alpha_1}, j_{\alpha_2}, \ldots, j_{\alpha_m} \leq n} \left(w_i \times A_1[i] \times A_{\alpha_1}[j_{\alpha_1}] \times \cdots \times A_{\alpha_m}[j_{\alpha_m}] \right) \end{aligned}$$

The volume contributed by G_1, namely, V_1 can be written as the quantity *not* claimed by any of the other gradients. Using the inclusion-exclusion principle, we get

$$\begin{aligned} V_1 &= \prod_{2\leq\alpha\leq d} L_\alpha \times \sum_{1\leq i\leq n} w_i \times A_1[i] \;\; - \sum_{S\subseteq\{2,\ldots,d\}\wedge S\neq\emptyset} (-1)^{|S|+1} Loss(S) \\ &= Loss(\{\}) \;\; - \sum_{S\subseteq\{2,\ldots,d\}\wedge S\neq\emptyset} (-1)^{|S|+1} Loss(S) \\ &= \sum_{S\subseteq\{2,\ldots,d\}} (-1)^{|S|} Loss(S) \end{aligned}$$

Observe that this expression contains 2^{d-1} inner terms of the following general form:

$$C \times \sum_{1\leq i_1<i_2,i_3,\ldots,i_k\leq n} \mathcal{A}_1[i_1] \times \cdots \times \mathcal{A}_k[i_k],$$

where C is constant, the number of indices k is at most d, and each array $\mathcal{A}_j[\,]$ corresponds to either a gradient array or the array $A[i] = w_i \times A_1[i]$. We further decompose each term of the above form into $(k-1)!$ terms by grouping the inner terms of the summation by the ordering of their indices $i_2, \ldots, i_k$. This yields at most $\lfloor \mathsf{e}(d-1)! \rfloor$ terms[4] (where e is the natural logarithm base) such that each term has the following

[4] Note that $\sum_{k=1}^{d} \binom{d-1}{k-1} \times (k-1)! = \sum_{k=0}^{d-1} \frac{(d-1)!}{(d-1-k)!} = \sum_{k=0}^{d-1} \frac{(d-1)!}{k!} \leq \mathsf{e}(d-1)!\,.$

form:[5]

$$C \times \sum_{1 \le i_1 < \cdots < i_k \le n} \mathcal{A}_1[i_1] \times \cdots \times \mathcal{A}_k[i_k].$$

Let us call this expression a *sum of ordered products*. All we need now is a data structure that can maintain the sum of ordered products efficiently as array entries are modified. We show in the following subsection (Section 1.4.2) how to maintain the sum of ordered products in $O(\log n)$ time per update and $O(1)$ time per query, which is sufficient for the following key theorem for the weighted volume of halfspaces in any fixed dimension d.

Theorem 1.4.1. *The weighted volume of a set of axis-aligned halfspaces in a rectangular box R in d dimensions can be maintained in $O(\log n)$ amortized time per insertion, with a linear-space data structure.*

1.4.2 Maintaining the Sum of Ordered Products

Given d arrays $\mathcal{A}_i$, $i = 1, 2, \ldots, d$, each of size n, we want to efficiently maintain the following sum of ordered products, under updates to individual array entries:

$$\sum_{1 \le i_1 < \ldots < i_d \le n} \mathcal{A}_1[i_1] \times \ldots \times \mathcal{A}_d[i_d]$$

[5] Since we assume that all halfspaces have distinct weights, all inner terms that contain equal indices are 0, and so we can safely ignore these terms and use a strict ordering on indices $i_2, \ldots, i_k$.

For simplicity, let us assume that n is a power of two. For $0 \leq j \leq \log n$ and $1 \leq k \leq n/2^j$, let $S(j,k)$ denote the set of consecutive integers in the range $[(k-1)2^j + 1, k2^j]$. Observe that $S(0,k) = \{k\}$ and $S(\log n, 1) = \{1, \ldots, n\}$. Moreover, $S(j,k)$ is the concatenation of $S(j-1, 2k-1)$ and $S(j-1, 2k)$, for $j \geq 1$. Conceptually, S represents a hierarchically ordered binary partition of the set $\{1, \ldots, n\}$ into singleton integers. If the partition is viewed as a tree, then $S(j,k)$ refers to the kth node from the left at the jth level and it is the parent of $S(j-1, 2k-1)$ and $S(j-1, 2k)$.

Let $T(j,k,l,r)$ denote the following sum of ordered products

$$\sum_{i_l < \ldots < i_r \ \wedge \ i_l, \ldots, i_r \in S(j,k)} \mathcal{A}_l[i_l] \times \ldots \times \mathcal{A}_r[i_r],$$

where $0 \leq j \leq \log n$, $1 \leq k \leq n/2^j$ and $1 \leq l \leq r \leq d$. We observe that $T(\log n, 1, 1, d)$ is the sum of ordered products that we want to maintain. Additionally, $T(0,k,l,l) = \mathcal{A}_l[k]$ and $T(0,k,l,r) = 0$ for $l \neq r$. For $j \geq 1$, we can write $T(j,k,l,r)$ in terms of $T()$ values whose first parameter is $(j-1)$, as shown in the following lemma.

Lemma 1.4.2. *For $j \geq 1$, we have the following recurrence:*

$$\begin{aligned} T(j,k,l,r) = {} & T(j-1, 2k-1, l, r) \ + \ T(j-1, 2k, l, r) \\ & + \sum_{l \leq c < r} \Big(T(j-1, 2k-1, l, c) \times T(j-1, 2k, c+1, r) \Big) \end{aligned}$$

Proof. A member of $S(j,k)$ is either a member of $S(j-1,2k-1)$ or $S(j-1,2k)$. We can, therefore, decompose $T(j,k,l,r)$ into sums of products based on the sets to which the indices belong, as follows:

$$\begin{aligned} T(j,k,l,r) = & \sum_{i_l<\ldots<i_r \ \wedge \ i_l,\ldots,i_r \in S(j-1,2k-1)} A_l[i_l] \times \ldots \times A_r[i_r] \\ & + \sum_{i_l<\ldots<i_r \ \wedge \ i_l,\ldots,i_r \in S(j-1,2k)} A_l[i_l] \times \ldots \times A_r[i_r] \\ & + \sum_{l \le c < r} \left(\sum_{\substack{i_l<\ldots<i_r \\ \wedge \ i_l,\ldots,i_c \in S(j-1,2k-1) \\ \wedge \ i_{c+1},\ldots,i_r \in S(j-1,2k)}} A_l[i_l] \times \ldots \times A_r[i_r] \right) \end{aligned}$$

By the distributive property of multiplication over addition, we get

$$\begin{aligned} T(j,k,l,r) = & \sum_{i_l<\ldots<i_r \ \wedge \ i_l,\ldots,i_r \in S(j-1,2k-1)} A_l[i_l] \times \ldots \times A_r[i_r] \\ & + \sum_{i_l<\ldots<i_r \ \wedge \ i_l,\ldots,i_r \in S(j-1,2k)} A_l[i_l] \times \ldots \times A_r[i_r] \\ & + \sum_{l \le c < r} \left(\begin{array}{c} \displaystyle\sum_{\substack{i_l<\ldots<i_c \\ \wedge \ i_l,\ldots,i_c \in S(j-1,2k-1)}} A_l[i_l] \times \ldots \times A_c[i_c] \\ \times \\ \displaystyle\sum_{\substack{i_{c+1}<\ldots<i_r \\ \wedge \ i_{c+1},\ldots,i_r \in S(j-1,2k)}} A_{c+1}[i_{c+1}] \times \ldots \times A_r[i_r] \end{array} \right) \end{aligned}$$

This expression is equivalent to

$$T(j,k,l,r) = T(j-1,2k-1,l,r) \;+\; T(j-1,2k,l,r) \\ + \sum_{l \le c < r} \Big(T(j-1,2k-1,l,c) \times T(j-1,2k,c+1,r) \Big)$$

The lemma follows. □

We maintain $T(\cdot)$ values in a table: for each valid selection of (j,k,l,r), the table has an entry storing $T(j,k,l,r)$. Then, a query can be answered in $O(1)$ time by reporting $T(\log n, 1, 1, d)$. Moreover, it is easy to show that the table has $O(n)$ entries. When an array entry $A_s[k]$ is updated, we update the table entry for $T(0,k,s,s)$ and the table entries for all $T(\cdot)$ values that are dependent on $T(0,k,s,s)$ through the recurrence relation given in Lemma 1.4.2. Notice that updating an entry takes constant time, assuming d is a constant. It can be easily seen that all table entries dependent on $T(0,k,s,s)$ are in the form $T(j, \lceil \frac{k}{2^j} \rceil, l, r)$ such that $0 \le j \le \log n$ and $1 \le l \le s \le r \le d$. Thus, assuming d is a constant, $O(\log n)$ entries are updated in total. This leads to the following theorem.

Theorem 1.4.3. *The sum of ordered products of d arrays of size n can be maintained with an update time of $O(\log n)$, query time of $O(1)$, using a linear-space data structure.*

1.4.3 Higher Order Partial Sums and Sum of Ordered Products

The reader will notice that the partial sum problem utilized in the halfplane solution is replaced by sum of ordered products in higher dimensions, suggesting a link between the two problems. In fact, the sum of ordered products can be viewed as an *iterated* or *higher-order* generalization of the classical partial sum problem.

Given an array $\mathcal{A}$ of n numbers, the basic *partial sum* problem, addresses the following query operation: report $query(k) = \sum_{i=1}^{k} \mathcal{A}[i]$. There are n different partial sum queries, for $1 \leq k \leq n$, and one can view them as forming another array $\mathcal{A}'[k] = query(k)$. We can then ask the partial sum problem on $\mathcal{A}'$, which could be considered the *second order* partial sum of $\mathcal{A}$. An iterated application of this process leads to higher order partial sums, with the following general form.

> Given an array $\mathcal{A}$ of size n, its *kth partial sum of order d*, denoted $P_d(k)$, is defined recursively as follows:
>
> $$P_d(k) = \begin{cases} \sum_{i=1}^{k} \mathcal{A}[i] & \text{if } d = 1 \\ \sum_{i=1}^{k} P_{d-1}(i) & \text{if } d > 1 \end{cases}$$

Considering the d arrays $\mathcal{A}_1, \ldots, \mathcal{A}_d$ of size n, the following definition is a further generalization of the iterated partial sums problem for $\mathcal{A}_1$:

> Given d arrays $\mathcal{A}_1, \ldots, \mathcal{A}_d$ of size n, *their kth weighted partial sum (of order d)*, denoted $W_d(k)$, is defined as follows:
>
> $$W_d(k) = \begin{cases} \sum_{i=1}^{k} \mathcal{A}_1[i] & \text{if } d = 1 \\ \sum_{i=1}^{k} \mathcal{A}_d[i] \times W_{d-1}(i) & \text{if } d > 1 \end{cases}$$

The following lemma shows that the sum of ordered products is actually a weighted partial sum.[6]

Lemma 1.4.4. *For d arrays $\mathcal{A}_1, \ldots, \mathcal{A}_d$,*

$$W_d(n) = \sum_{1 \le i_1 \le \cdots \le i_d \le n} \mathcal{A}_1[i_1] \times \cdots \times \mathcal{A}_d[i_d]$$

Proof. By induction. The lemma clearly holds for $d = 1$. For $d > 1$,

$$\begin{aligned} W_d(n) &= \sum_{i=1}^{n} \mathcal{A}_d[i] \times W_{d-1}(i) \\ &= \sum_{i=1}^{n} \left(\mathcal{A}_d[i] \times \sum_{1 \le i_1 \le \cdots \le i_{d-1} \le i} \mathcal{A}_1[i_1] \times \cdots \times \mathcal{A}_{d-1}[i_{d-1}] \right) \\ &= \sum_{1 \le i_1 \le \cdots \le i_d \le n} \mathcal{A}_1[i_1] \times \cdots \times \mathcal{A}_d[i_d] \end{aligned}$$

□

Our sum of ordered products structure can be easily used to maintain weighted (or iterated) partial sums of d arrays $\mathcal{A}_1, \ldots, \mathcal{A}_d$. Let $\mathcal{A}_{d+1}$ be an additional array such that $\mathcal{A}_{d+1}[i] = 0$ for all i. Then, the weighted partial sum $W_d(i)$ can be obtained by simply incrementing $\mathcal{A}_{d+1}[i]$ by 1 and then querying for the sum of ordered products of the set $\{\mathcal{A}_1, \ldots, \mathcal{A}_{d+1}\}$.

[6]For simplicity of presentation only, we use a non-strict ordering of the array indices (i.e., $i_1 \le \cdots \le i_d$) in this lemma. The sum of products with strictly ordered indices (i.e., $i_1 < \cdots < i_d$, as defined in Section 1.4.2) can be easily reduced to this form by shifting arrays. In particular, the strictly ordered sum of products for d arrays $\mathcal{A}_1, \ldots, \mathcal{A}_d$ is equal to the non-strictly ordered sum of products for arrays $\mathcal{A}'_1, \ldots, \mathcal{A}'_d$ where $\mathcal{A}'_s[i]$ is defined to as $\mathcal{A}_s[i + s - 1]$.

1.5 Dynamic Weighted Volume For Arbitrary Boxes

Computing the Klee's measure of n 2-grounded boxes in d-space requires maintaining the weighted volume, under insertion-only updates, of n boxes in $(d-2)$-space. In this section, we complete the discussion of this last step, and show how to maintain the weighted volume under insert-only updates in $O(n^{(d-1)/2}\log^2 n)$ amortized time in d dimensions. Since we require n such insertions in $(d-2)$-space, the final complexity of our algorithm will be $O(n \cdot n^{(d-3)/2}\log^2 n) \;=\; O(n^{(d-1)/2}\log^2 n)$, as desired.

Our scheme is based on the space partition technique originally proposed by Overmars and Yap [64]. In particular, we partition the space into to a set of axis-parallel regions such that the boxes form a set of *axis-parallel strips*[7] inside each region (a structure known as a *trellis*). See Figure 1.5 for an illustration. By utilizing a binary space partition tree on the partition, we reduce the main problem to the simpler case of maintaining the weighted volume of axis-parallel strips. This case can be relatively easily solved by extending our halfspace solution.

For simplicity, we first describe a solution for the two-dimensional case in Section 1.5.1, which we later extend into d dimensions in Section 1.5.2. We defer the details of how we generalize our halfspace solution to the case of axis-parallel strips to Section

[7] In d dimensions, an axis-parallel strip has the form $\{\mathbf{x} \in \mathbb{R}^d \mid a \leq \mathbf{x}^k \leq b\}$ where a and b are reals and k is an integer between 1 and d.

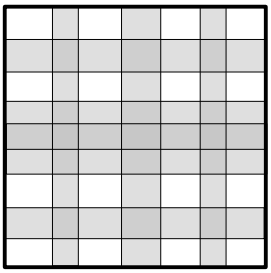


Figure 1.5: An arrangement of axis-parallel strips inside a rectangular region in two dimensions.

1.5.3. The following theorem, which we utilize in our descriptions, is proved in Section 1.5.3 and summarizes our data structure for axis-parallel strips.

Theorem 1.5.1. *The weighted volume of a set of axis-aligned strips in a rectangular box R in d dimensions can be maintained in $O(\log^2 n)$ amortized time per insertion, with a linear-space data structure. The data structure additionally supports an elimination operation in which all strips with weight less than a given weight are deleted. The cost of the elimination operation can be charged to the preceding insertions, and thus is also $O(\log^2 n)$ amortized.*

1.5.1 The Planar Case

Let $\mathcal{B}$ be a set of n weighted boxes in 2-space. Our objective is to maintain the weighted volume of a dynamic set of boxes that undergoes insertions from $\mathcal{B}$. We propose a structure that achieves this in $O(\sqrt{n}\log^2 n)$ amortized time per update.

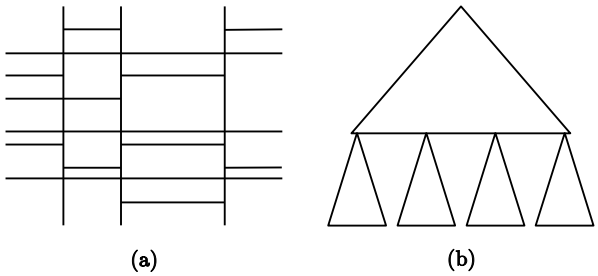


Figure 1.6: (a) An example partition. (b) Its binary space partition tree.

We first create a partition of the space into rectangular regions such that the set of boxes, $\mathcal{B}$, forms a set of axis-parallel strips inside each region. The partition is formed in two steps as follows. Each box in $\mathcal{B}$ has 2 vertical boundaries (left and right sides) and 2 horizontal boundaries (top and bottom sides). Thus, there are $2n$ vertical and $2n$ horizontal boundaries in total. In the first step of the partition, we sort the vertical boundaries with respect to their first (horizontal) coordinates. Then, we draw a vertical line through each $\sqrt{n}$-th boundary in the sorted list, dividing the space into $O(\sqrt{n})$ slabs, each of which contain $O(\sqrt{n})$ vertical boundaries in its interior. In the second step of the partition, we individually divide each slab along the second axis. We do this by sorting the horizontal boundaries by their second (vertical) coordinates and then drawing a horizontal line through: (1) each $\sqrt{n}$-th horizontal boundary of boxes passing through the slab, and (2) each box corner inside the slab. Consequently, each slab is partitioned into $O(\sqrt{n})$ final regions containing $O(\sqrt{n})$ horizontal boundaries (See Figure 1.6a). Note that each slab contains $O(\sqrt{n})$ box corners because there are only $O(\sqrt{n})$ vertical boundaries inside.

We call the final regions of the partition *cells*. The following lemma, which is proved in [64], summarizes the properties of the partition. We include the proof for completeness.

Lemma 1.5.2. *The partition has the following properties.*

(a) There are $O(n)$ cells.

(b) Any box intersects $O(\sqrt{n})$ cells on its boundary.

(c) Each cell intersects the boundary of $O(\sqrt{n})$ boxes in $\mathcal{B}$.

(d) The boxes of $\mathcal{B}$ contain no corner in the interior of any cell.

Proof. (a) is obvious from construction. We show (b) as follows. Each vertical boundary of a box intersects $O(\sqrt{n})$ cells that are contained in the same single slab. Moreover, each horizontal boundary of a box intersects at most one cell in each slab, thus, $O(\sqrt{n})$ cells in total. It follows that each box intersects $O(\sqrt{n})$ cells on its boundaries. (c) is due to that each cell intersects at most $O(\sqrt{n})$ vertical and horizontal boundaries. Finally, (d) holds because each slab is partitioned into cells by drawing horizontal lines through the box corners. □

The last property implies that $\mathcal{B}$ forms a set of strips inside each cell. As the first key part of our algorithm, we maintain, for each cell C, the weighted volume on C contributed *only* by the boxes that intersect C on their boundary. We do this by utilizing

a two-dimensional instance of the strips structure given in Theorem 1.5.1 for each cell. By Lemma 1.5.2, there are $O(\sqrt{n})$ boxes that intersect any cell on their boundary, thus, the size of each strip structure is $O(\sqrt{n})$. Also, during the insertion of a box B, we can update all strip structures in $O(\sqrt{n}\log^2 n)$ amortized time, because B intersects $O(\sqrt{n})$ cells on its boundary and updating the strip structure of each of these cells takes $O(\log^2 n)$ time. We note that the cells lying on the boundary of B can be efficiently obtained by using the partition tree that we describe below.

Clearly, the weighted volumes maintained in the cells exclude the contributions of the boxes that *entirely* contain the cells. The second key part of our algorithm is to include these contributions so that the overall weighted volume is correctly maintained. To do this efficiently, we utilize a *binary space partition tree*. In particular, we maintain a balanced binary tree in which every node v is associated with a rectangular region of the space R_v, such that for all internal nodes v with children v_l and v_r, R_v is divided into R_{v_l} and R_{v_r} by a vertical or horizontal line. Moreover, the root is associated with the whole space and each leaf is associated with one cell of our partition bijectively. Such a balanced space partition tree can be easily constructed in $O(n \log n)$ time by constructing a balanced tree for the slabs first, and then for the cells within each slab (See Figure 1.6b).

When a box B is inserted to the structure, we store an entry for B at every node v such that B subsumes R_v but not $R_{parent(v)}$. This storage scheme conceptually parti-

tions B into a set of maximal fragments, where each fragment is the intersection $B \cap R_v$ for a node v that B is stored. Note that this fragmentation excludes the sections of B that partially overlap the cells, which are already stored in the corresponding strip structures. The following lemma is borrowed from [64], and included here with proof only for the sake of completeness.

Lemma 1.5.3. *Each box B is stored in $O(\sqrt{n}\log n)$ nodes in the binary space partition tree and these nodes can be traversed in $O(\sqrt{n}\log n)$ steps.*

Proof. By Lemma 1.5.2(b), there are $O(\sqrt{n})$ leaves v in the tree such that B partially overlaps R_v, i.e., $R_v \cap B \neq \emptyset$ and $R_v \not\subseteq B$. Then, there are $O(\sqrt{n}\log n)$ nodes u (internal or leaf) such that B partially overlaps R_u. This follows from the fact that each such node u is above at least one leaf v that is partially overlapped by B and the tree height is $O(\log n)$. Since the tree is binary and each node that stores B has a parent u that is partially overlapped, B is stored at $O(\sqrt{n}\log n)$ nodes.

The nodes storing B can be traversed by recursively descending in the tree through the nodes that are partially overlapped until nodes subsumed by B are reached. Since the number of partially overlapped nodes is $O(\sqrt{n}\log n)$, so is the traversal time. □

For efficient weighted volume maintenance, we remove any box fragments that are entirely contained by boxes of higher weight. In particular, when we insert a box entry B to a node v, we delete all box entries B' stored below v such that $w(B') < w(B)$. See

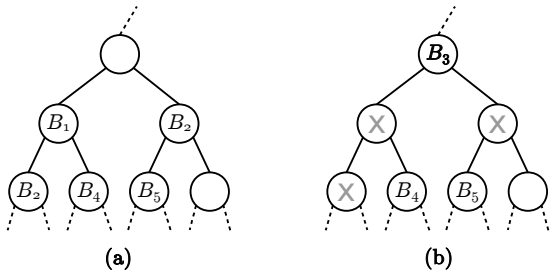


Figure 1.7: (a) A set of boxes stored in the partition tree. Box with larger index has higher weight. (b) The tree after inserting an entry for box B_3. All lower weight entries below the inserted entry are deleted.

Figure 1.7 for an example. Note that the deleted fragments have no contribution to the weighted volume because they are contained by a heavier box, and thus they are safe to be deleted. Similarly, for each cell that a newly inserted box subsumes, we delete all lower weight boxes in the corresponding strip structure by using its elimination operation. Finally, we keep only the highest weight box at each node. The result of this policy is an invariant that for any particular box B at a node v, all boxes stored above v have lower weights while all boxes stored below (including the strip structures) have higher weights. Consequently, the weighted volumes of the strip structures directly contribute to the overall. Moreover, we can write the weighted volume contribution of a box B stored at node v as

$$w(B) \times (|R_v| - A(v))$$

where $|R_v|$ is the ordinary volume (area) of region R_v and $A(v)$ is the ordinary volume of the union of the boxes stored strictly below v (including the strip structures). Note

that $A(v)$ equals the volume that is claimed from B by higher weight fragments, which are all stored below v. The maintenance of $A(v)$ for leaf nodes v can be done with no extra cost by utilizing a copy of the strips structure at v that maintains the *ordinary* volume rather than the weighted volume. (In this case, the weights of the boxes are set to 1.) For each internal node v with children v_l and v_r, $A(v)$ can be written recursively in terms of $A(v_l)$ and $A(v_r)$, and thus can easily be updated during the tree traversals. In particular, we have:

$$A(v) = \begin{cases} |R_{v_l}| + |R_{v_r}| & \text{if } v_l \text{ and } v_r \text{ both store a box} \\ |R_{v_l}| + A(v_r) & \text{if } v_l \text{ stores a box and } v_r \text{ does not} \\ A(v_r) + |R_{v_r}| & \text{if } v_r \text{ stores a box and } v_l \text{ does not} \\ A(v_r) + A(v_l) & \text{if neither } v_l \text{ nor } v_r \text{ store a box} \end{cases}$$

The overall weighted volume is easily maintained as the sum of weighted volume contributions of each strip structure and each box entry in the tree. It remains to show that removing lower weight box entries during insertions do not increase the overall amortized cost of $O(\sqrt{n}\log^2 n)$ per insertion. Recall that the tree traversal performed to insert a box takes $O(\sqrt{n}\log n)$ steps. To efficiently delete the lower weight boxes after an insertion, we maintain at each node v, the weight of the lowest weight box stored below v (including the boxes stored in the strip structures). This allows us to locate the

box entries that we need to delete, without searching the whole tree. In particular, we descend down the tree as long as the lowest weight stored below the current node is lower than weight of the inserted box. The additional traversals performed to locate the deleted entries in the tree can be charged to the traversals which inserted these entries. Notice that each step of an insertion traversal (i.e., traversal of an edge) is charged at most once by a deletion traversal. Finally, as we mention in Theorem 1.5.1, the time spent to eliminate the box entries in the strip structures can also be charged to the preceding insertions. It follows that the amortized cost of a box insertion remains $O(\sqrt{n}\log^2 n)$.

The binary space partition tree uses $O(n)$ space and can be constructed in $O(n \log n)$ time. Each of the $O(n)$ strip structures uses $O(\sqrt{n})$ space (by Lemma 1.5.2(c)) and can be constructed in $O(\sqrt{n}\log n)$ time. This yields the following theorem.

Theorem 1.5.4. *There exists a data structure that maintains the weighted volume of n 2-dimensional boxes in $O(\sqrt{n}\log^2 n)$ amortized time per insertion. This structure can be constructed in $O(n\sqrt{n}\log n)$ time and consumes $O(n\sqrt{n})$ space.*

1.5.2 The d-dimensional Case

We now extend our weighted volume data structure to d dimensions. In d dimensions, the high-level algorithm is the same. We construct a d-dimensional partition whose cells do not contain any $(d-2)$-dimensional facets of any box, i.e., all boxes are

strips inside the cells. In this section, we briefly describe how this construction is done. (Details on this construction may be found in [64].) Note that the rest of the algorithm can be adapted trivially to d-dimensions: we organize the final partition in a balanced binary space partition tree and apply our maintenance scheme. We emphasize that we utilize the d-dimensional version of our strips structure in the leaves of the partition tree.

Let B be a set of n weighted boxes in d-space. We construct the partition in d steps as follows. Each box has two $(d-1)$-dimensional boundaries that are orthogonal to the ith coordinate axis, for a particular i ($1 \leq i \leq d$). For simplicity, we call these boundaries *i-bounds*. In the first step of construction, we sort the 1-bounds of the boxes with respect to their first coordinate. Then, we draw a $(d-1)$-dimensional hyperplane (orthogonal to the first axis) through each $\sqrt{n}$-th 1-bound in the sorted list, dividing the space into $O(\sqrt{n})$ sections, each of which contains $O(\sqrt{n})$ 1-bounds. In the second step, we individually divide each section along the second coordinate. We do this by sorting the 2-bounds and then drawing a hyperplane (orthogonal to the second axis) through: (1) each $\sqrt{n}$-th 2-bound in the section, and (2) each 2-bound whose owner box contains a 1-bound in the section. Consequently, each section is partitioned into $O(\sqrt{n})$ subsections containing $O(\sqrt{n})$ 1-bounds and 2-bounds. In the third step, we cut each section with respect to the third axis through: (1) each $\sqrt{n}$-th 3-bound in the section and (2) each 3-bound whose owner box contains a 1-bound or 2-bound in the

section. We continue in this fashion for a total of d steps, creating a partition with $O(n^{d/2})$ cells each containing $O(\sqrt{n})$ i-bounds for any $1 \leq i \leq d$. The following is the d-dimensional analog of Lemma 1.5.2, also from [64].

Lemma 1.5.5. *The partition has the following properties.*

(a) There are $O(n^{d/2})$ cells.

(b) Any box intersects $O(n^{(d-1)/2})$ cells on its boundary.

(c) Each cell intersects the boundary of $O(\sqrt{n})$ boxes in $\mathcal{B}$.

(d) The boxes of $\mathcal{B}$ contain no $(d-2)$-dimensional facet in the interior of any cell.

Proof. (a) is true because at each step of the construction, the number of sections increase by a factor of $O(\sqrt{n})$. We prove (b) as follows. Each i-bound intersects at most $O(n^{(i-1)/2})$ sections just before the ith step of the construction. After the ith step, for any section that an i-bound intersected previously, the i-bound can intersect at most one of its resulting subsections (because the cuts are parallel to the i-bound). Thus, each i-bound still intersects $O(n^{(i-1)/2})$ sections after the ith step. It follows that each i-bound intersects $O(n^{(d-1)/2})$ sections when all d steps are completed. Since each box has $2d$ i-bounds in total, (b) follows. (c) is due to that each cell intersects at most $O(\sqrt{n})$ i-bounds for each $1 \leq i \leq d$. Finally, we prove (d) as follows. A $(d-2)$-dimensional facet is the intersection of a j-bound and a k-bound of a box for $1 \leq j < k \leq d$. By

induction, after ith step of the construction, no resulting subsection contains a j-bound and a k-bound of the same box such that $1 \leq j < k \leq i$. Thus, after the dth step, no cell contains a $(d-2)$-dimensional facet. □

Given the above partition, we can utilize our partition tree algorithm of the previous section (with d-dimensional strip structures). By following similar proof steps, we deduce the following result.

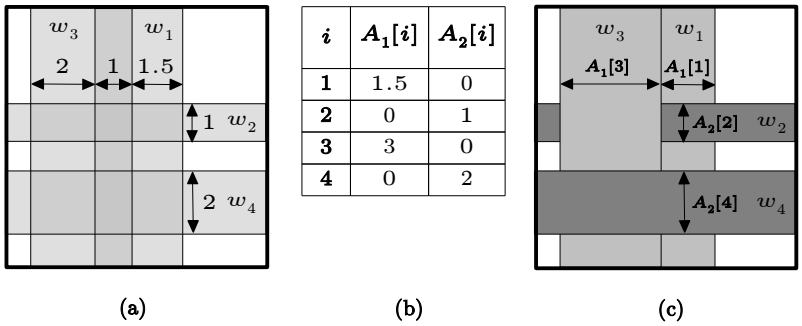
Theorem 1.5.6. *We can maintain the weighted volume of a set of n d-dimensional boxes in $O(n^{(d-1)/2}\log^2 n)$ amortized time per box insertion. This data structure can be constructed in $O(n^{(d+1)/2}\log n)$ time using $O(n^{(d+1)/2})$ space.*

Computing the Klee's measure for 2-grounded boxes in d-dimensional space requires n box insertions into a $(d-2)$-dimensional structure, for the total complexity of $O(n \cdot n^{(d-3)/2}\log^2 n) \;=\; O(n^{(d-1)/2}\log^2 n)$, giving us the main result of this chapter.

Theorem 1.5.7. *Klee's measure for n 2-grounded boxes in d-space can be computed in worst-case time $O(n^{(d-1)/2}\log^2 n)$ and space $O(n^{(d-1)/2})$.*

1.5.3 Weighted Volume of Axis-parallel Strips

Our technique for maintaining the volume of halfspaces extends easily to strips, by keeping a separate array for strips orthogonal to each axis. In this section, we describe this extension, and show how to maintain the weighted volume of axis-parallel strips



i	$A_1[i]$	$A_2[i]$
1	1.5	0
2	0	1
3	3	0
4	0	2

Figure 1.8: (a) A two-dimensional arrangement of four strips with weights in descending order $w_4 > w_3 > w_2 > w_1$. Notice the intersection between the strips with weights w_1 and w_3, where the strip with weight w_3 dominates. (b) The array representation of the strips. (c) Illustrating the intersection of the strips. The "light gray" (resp.,"dark gray") section shows the portion where the weights of the vertical (resp., horizontal) strips dominate.

that are in the form $\{\mathbf{x} \in \mathbb{R}^d \mid a \leq \mathbf{x}^k \leq b\}$ (where a and b are reals). In particular, let $S_1, \ldots, S_n$ be a set of n strips such that the indices are ordered by weight, i.e., $w(S_1) < \cdots < w(S_n)$. We maintain d arrays $A_1, \ldots, A_d$ of size n. If the strip S_i is orthogonal to the kth axis, then the entry $A_k[i]$ represents the total width of S_i not claimed by the strips orthogonal to the kth axis. Otherwise, $A_k[i]$ is zero. See Figure 1.8. The algorithm from Section 1.4 can then be applied on these arrays to maintain the weighted volume, at the cost of $O(\log n)$ per array entry change.

We utilize a standard data structure known as a *segment tree* to maintain the arrays. In particular, the segment tree used to maintain the array A_k stores the strips orthogonal to the kth axis as one-dimensional *weighted* intervals on the kth axis. (The intervals are the projections of these strips.) The segment tree stores each interval in $O(\log n)$ of its nodes, which conceptually partitions the interval into $O(\log n)$ subintervals in a

way very similar to the binary space partition tree described in Section 1.5.1. (For more details on segment trees, see [35].)

By applying the same maintenance ideas from the space partition tree, we can maintain the length dominated by each interval (which corresponds to the array entry of its strip) in $O(\log n)$ time per interval insertion. Each interval insertion affects $O(\log n)$ array entries amortized. This follows from the fact that each inserted interval is stored on $O(\log n)$ nodes in the tree, with $O(\log n)$ total ancestors. The array contribution of the lower weight intervals stored in these $O(\log n)$ ancestors are possibly modified, whereas the contributions of the lower weight intervals in all descendants are *deleted*, for which we charge to their corresponding insertions.

Each array entry change is coupled with an $O(\log n)$ time update in our sums of ordered products structure. It follows that we can maintain the weighted volume of strips in $O(\log^2 n)$ amortized time per insertion.

We finally note that we also need an elimination operation as part of the main algorithm described in Section 1.5.1. In this elimination operation, we delete all strips with weights less than a given weight w. This can be easily achieved by doing a binary search on the weights to identify the strips to delete and then deleting the corresponding intervals from the segment trees. As usual, we can charge the deletions to the corresponding insertions, thus this operation can be achieved in $O(\log^2 n)$ amortized time as well.

1.6 Klee's Measure for d-Grounded Boxes

When the boxes are d-grounded, namely, they are all anchored at a common corner, the time complexity of our algorithm improves by a factor of $\log n$. This follows from the fact that the Overmars-Yap partition, when applied on d-grounded boxes, yields to regions that contain half-spaces rather than strips. By Theorem 1.4.1, we can maintain weighted volume of each region in $O(\log n)$ time per update (improving on $O(\log^2 n)$ for strips), reducing the running time of our algorithm to $O(n^{(d-1)/2} \log n)$. The d-grounded case of the problem is also known as the hypervolume indicator problem, which is utilized in evolutionary computing often to assess the quality of multi-objective optimization algorithms. Several techniques in the computational geometry literature can be used solve the problem efficiently. Prior to our algorithm, the best bounds with respect to the number of dimensions are as follows:

- For $d \leq 3$, $O(n \log n)$, based on space-sweep [40].
- For $d = 4$, $O(n^{3/2}\text{polylog } n)$ by Chan [22], using reduction to unit-cubes.
- For $d = 5$, $O(n^2\text{polylog } n)$ by Kaplan et al. [52].
- For $d \geq 6$, $O(n^{(d+2)/3})$ by Bringmann [15], using reduction to fat-boxes.

Compared to the above results, our algorithm is faster in dimensions 4, 5 and 6, improving the bound for computing the hypervolume indicator in these dimensions, and using a simpler approach.

1.7 Conclusion

We have proposed a new method for computing Klee's measure on grounded boxes, which includes the hypervolume indicator as a special case. In particular, we obtained a bound of $O(n^{(d-1)/2} \log^2 n)$ for the k-grounded problem for $2 \leq k < d$, which is an improvement of roughly $\sqrt{n}$ over the general Klee's bound. In a follow-up work by Bringmann [16], it has been shown that a significant breakthrough of our bound for 2-grounded boxes would imply a breakthrough for the general Klee's problem. On the other hand, we emphasize that even if our bound is (nearly) tight for 2-grounded boxes, it is possible that better algorithms exist for k-grounded boxes where k is strictly larger than 2.

Our techniques also lead to an algorithm for the hypervolume indicator problem that is faster than any previously published algorithm for dimensions 4, 5 and 6. The research on hypervolume indicator is still very active. A recent work by Chan [24] has improved our bounds for hypervolume indicator even further, to $O(n^{d/3} \operatorname{polylog} n)$ for $d \geq 4$.

Given the long and distinguished history of Klee's measure problem, where all the previous improvements have been limited to cube-like boxes, the grounded boxes offer an interesting new direction to pursue.

Chapter 2

The Union of Uncertain Boxes*

2.1 Introduction

In this chapter, we consider the problem of estimating the volume covered by a set of overlapping boxes in d-space when the existence of each box is known only with partial certainty. Specifically, we are given a set of n axis-aligned boxes, in which the ith box is known to be present only with probability p_i. (The probabilities of different boxes are independent of each other.) This is a *probabilistic* version of Klee's Measure Problem. (A brief history of Klee's Measure Problem is given in Section 1.1.1 in Chapter 1.) Besides being a fundamental problem in its own right, it is also a natural framework

*This chapter is based on a joint work with Luca Foschini, John Hershberger and Subhash Suri. Parts of this chapter appeared in the following publications: [78] (Published and copyright held by Springer. The final publication available at `http://link.springer.com/10.1007/978-3-642-23719-5_50`.)

to model risk and uncertainty in geometric settings. In order to motivate the problem, consider the following scenario.

Suppose that a tract of land has a variety of health hazards, occurring in possibly overlapping regions. The virulence of each hazard is expressed as a survival rate, i.e., the probability that an entity survives after being exposed to the hazard. Assuming independence of the hazards, the probability of survival at a point is the product of the survival probabilities for the different hazards affecting the point. The *average survival rate* within the whole tract is the integral of the survival probabilities of all points in the tract divided by the area the tract. It is easy to see that the integral of concern equals the expected area of covered by the hazardous regions, if we treat the survival rates as the probability of *absence* for each region.

Let us now introduce the problem more formally. Recall (from Chapter 1) that a d-dimensional box B is the Cartesian product of d one-dimensional ranges, namely, $B = \Pi_{i=1}^{d}[a_i, b_i]$ and its volume is $vol(B) = \Pi_{i=1}^{d}|b_i - a_i|$. In our problem, we consider *uncertain* boxes. In particular, each box B_i is assumed to be *present* (or active) with an independent probability p_i, and absent otherwise. Under these conditions, we wish to compute the *expected value* of the total volume occupied by such a collection of boxes. More generally, we may wish to compute the probability distribution of the volume—for each value V, the probability that the volume of the union is V. In fact, we wish to maintain a collection of uncertain boxes so that their volume statistics (expectation or

tail bounds) are easily updated as boxes (along with their activation probabilities) are inserted or deleted.

Our main result in this chapter is a data structure for maintaining the expected volume over a dynamic set of uncertain boxes in amortized time $O(n^{(d-1)/2} \log n)$ per insert or delete. Any major improvement in the update complexity of our data structure will imply a breakthrough on Klee's Measure Problem because the d-dimensional Klee's problem can be solved by maintaining a $(d-1)$-dimensional volume over n insertions and deletions [23, 64]. In particular, one can utilize our data structure to compute the expected volume of a static set of uncertain box in $O(n \cdot n^{(d-2)/2} \log n) = O(n^{d/2} \log n)$ time, which nearly matches the best bound for computing the volume of a set of regular (certain) boxes.

The core problem in computing the volume of uncertain boxes arises already in one dimension, and leads us to a new data structure called an *anonymous segment tree*. This structure allows us to compute the expected length covered by a set of uncertain segments in logarithmic time per update. Building on this foundation, we then generalize the problem to d dimensions by combining it with the ideas of Overmars and Yap [64]. (The issues underlying this extension are mostly technical, albeit somewhat non-trivial, since the Overmars-Yap scheme uses a space-sweep that requires *a priori* knowledge of the box coordinates, while we assume a fully dynamic setting with no prior knowledge of future boxes.)

Surprisingly, while the expected value of the volume can be efficiently maintained, we show that computing the tail bounds, or the probability distribution, of the volume is intractable—specifically, it is *NP*-hard to compute the *probability* that the volume of the union exceeds a given value V even when the dimension is $d = 1$.

Chapter Organization

The remaining of the chapter is organized as follows. In Section 2.2, we describe our *NP*-hardness result for computing tail bounds. In Section 2.3, we solve the dynamic expected volume problem on one-dimensional uncertain segments and describe the anonymous segment tree data structure. In Section 2.4, we describe our solution for maintaining the expected volume in higher dimensions. We finish with a conclusion in Section 2.5.

2.2 Probabilistic Volume: Expectation and Tail Bounds

If algorithmic efficiency were not the main concern, then the *expected* volume of the union of n boxes would be easy to compute in polynomial time. A set of n boxes in d-space is defined by $O(dn)$ facets, and the hyperplanes determined by those facets partition the space into $O(n^d)$ rectangular cells, with each cell fully contained in all the boxes that intersect it. For each cell, compute the probability that at least one of

its covering boxes is active. By the linearity of expectation, the expected volume is simply the sum of the volumes of these cells weighted by their probability of being covered. This naïve algorithm runs in $O(n^{d+1})$ time, which is polynomial in n for fixed dimension. In Section 2.4, we present our main result: how to maintain the expected volume much more efficiently under dynamic updates.

On the other hand, we argue below that computing the probability distribution (i.e., the tail bounds) is intractable. In particular, even in one dimension, computing the probability that the union of n uncertain line segments has volume (length) at least V is *NP*-hard. The following theorem shows that it is hard to compute the probability that the union has length precisely L for some integer L—since we use only integer-valued lengths, computing the tail bound is at least as hard (the probability of length exactly L can be determined from those of lengths at least L and at least $L + 1$.)

Theorem 2.2.1. *Given n disjoint line segments on the integer line, where the ith segment is active with probability p_i, it is NP-hard to compute the probability that the union of the active segments has length L.*

Proof. We show a reduction from the well-known *NP*-complete problem SUBSET-SUM [42]. The subset-sum problem takes as input a set of positive integers $A = \{a_1, \ldots, a_n\}$, a *target* integer L, and asks if there is an index subset $I \subseteq \{1, \ldots, n\}$ whose elements sum to the target value L, that is, $\sum_{i \in I} a_i = L$. Given an instance of the subset-sum problem (A, L), we create a set of uncertain segments $\{s_1, s_2, \ldots, s_n\}$,

as follows. The segment s_i begins at point $\sum_{j<i} a_j$ and has length a_i. Each of the n segments occurs with probability $p_i = 1/2$. We observe that because of the uniform probability, each of the 2^n subsets of $\{s_1, s_2, \ldots, s_n\}$ is equally likely, each occurring with probability 2^{-n}.

Since the segments $s_1, \ldots, s_n$ are disjoint, for any index subset $I \subseteq \{1, \ldots, n\}$, the union of the segments $\{s_i | i \in I\}$ has length precisely $\sum_{i \in I} a_i$. Thus, I is a solution to the subset sum problem (A, L) if and only if the union of the segments indexed by I has length L. However, the probability that the union of the active segments has length L is precisely equal to the number of index subsets that are valid solutions of the subset sum problem divided by 2^n. Thus, given the probability that the union of the active segments is L, we can deduce whether the subset sum problem has a solution. This completes the proof. □

2.3 Maintaining the Expected Measure in 1D

We now describe our data structure for one-dimensional expected volume problem. In the next section, we show how to embed it in an appropriately generalized version of the Overmars-Yap structure for the d-dimensional problem. We describe the data structure, called the *anonymous segment tree*, first without the probabilities, focusing

on its form and updates, and then present an *abstraction* that retains all the key elements and yet accommodates probabilistic segments.

2.3.1 Anonymous Segment Tree

Let $\mathcal{S}$ be a dynamic set of n line segments on the number line that undergoes insertions and deletions. Our goal is to maintain the length covered by the union of the segments in $\mathcal{S}$. We simply call this length the *measure* of $\mathcal{S}$. The segments in $\mathcal{S}$ split the number line into at most $(2n + 1)$ disjoint intervals, called *primitive intervals*. We maintain a balanced binary tree (e.g., red-black tree) whose keys are the coordinates of the segment endpoints, and whose leaves correspond to the primitive intervals. Each internal node represents the union of all its leaf descendants' intervals. (See Figure 2.1a.)

Consider a leaf v and its associated primitive interval $I = (x_1, x_2)$. Let $\mathcal{S}_I \subset \mathcal{S}$ be the subset of segments that cover I, and define the *coverage count* of v, denoted $cover(v)$, as $|\mathcal{S}_I|$. The *measure* of v, denoted $\mu(v)$, is clearly zero if $cover(v) = 0$ and $x_2 - x_1$ otherwise. The measure of $\mathcal{S}$ is the sum of $\mu(v)$ over all leaves v. The coverage count is an inefficient mechanism for maintaining the measure when segments are inserted or deleted, so we use a secondary quantity, called $ccover(v)$. (The name *ccover* derives from *complete coverage count*.) The *ccover* values satisfy the two invariants described below.

> SUM INVARIANT: *For any leaf v, $cover(v)$ is the sum of $ccover(a)$ over all ancestors a of v (including v itself).*

A trivial way to achieve the invariant is to set $ccover(v) = cover(v)$ for each leaf v and $ccover(u) = 0$ for each non-leaf node u. But, as we show below, $ccover()$ allows us to support maintenance of the measure through its flexibility. We use $ccover()$ values to maintain the measure as follows, where $L(v)$ is the length of v's interval.

$$\mu(v) = \begin{cases} L(v) & \text{if } ccover(v) > 0 \\ 0 & \text{if } ccover(v) = 0 \ \wedge \ v \text{ is a leaf} \\ \mu(v_l) + \mu(v_r) & \text{if } ccover(v) = 0 \ \wedge \ v \text{ has children } v_l, v_r \end{cases} \tag{2.1}$$

The following lemma is easily established.

Lemma 2.3.1. *Let a node v be called* exposed *if $ccover(a) = 0$ for all ancestors a of v (excluding v). Then for any exposed node v, $\mu(v)$ is the measure of $\mathcal{S}$ restricted to v's interval. In particular, $\mu(root)$ is the measure of $\mathcal{S}$.*

Proof. Consider an exposed node v such that $ccover(v) > 0$. Then the measure of $\mathcal{S}$ restricted to v's interval is $L(v)$. This is captured by the first line of (2.1). Now consider an exposed leaf v such that $ccover(v) = 0$. Then $cover(v) = 0$ and consequently the measure of $\mathcal{S}$ in v's interval is 0. This is captured by the second line of (2.1). Finally, consider an exposed internal node v such that $ccover(v) = 0$. Then v_l and v_r are

exposed. By induction, $\mu(v_l)$ is the measure of $\mathcal{S}$ restricted to v_l's interval. A similar observation applies to $\mu(v_r)$. Then the measure restricted to v's interval is the sum of these two quantities and this is captured by the third line of (2.1). □

> PUSHUP INVARIANT: *For each non-leaf node v with children v_l and v_r, at least one of $ccover(v_l)$ and $ccover(v_r)$ is zero.*

We achieve this invariant by applying the following *push-up* operation at each internal node v: Let v_l and v_r be children of v. Decrement $ccover(v_l)$ and $ccover(v_r)$ by $\min(ccover(v_l), ccover(v_r))$ and increment $ccover(v)$ by the same amount. (See Figure 2.1b.) Notice that this operation restores the push-up invariant in v while preserving the sum invariant. (It, however, possibly breaks the push-up invariant in its parent, but this can be corrected by applying the same operation at the parent.) By applying push-ups throughout the whole tree in bottom-up fashion, from the leaves to the root, we proagate the values of $ccover()$ up the tree as much as possible, which in turn allows us to update the $\mu()$ values efficiently.[8]

We now discuss how to preserve the sum and push-up invariants during insertion and deletion of segments so that the measure of $\mathcal{S}$ is correctly maintained in $\mu(root)$. For simplicity, we focus our attention on the changes in $ccover()$ values. The $\mu()$ values are updated as necessary when the $ccover()$ values change. If $ccover(v)$ or the interval

[8]The ability to move $ccover()$ values between nodes is the inspiration for the name *anonymous segment tree*: the coverage representation for a node is independent of the covering segments. Coverage of an interval by a single segment is indistinguishable from coverage by an arbitrary number of consecutive short segments.

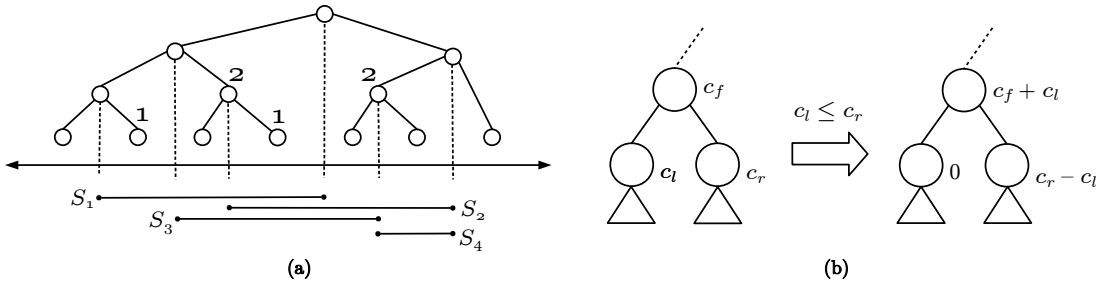


Figure 2.1: (a) An anonymous segment tree, positive $ccover$ values are shown. (b) the push-up operation, c_i's stand for $ccover$ values.

of v changes at a node v, then $\mu(v)$ must be updated at all ancestors of v, including v itself. In each operation that we discuss below, the total cost of updating the $\mu()$ values is logarithmic.

We insert a segment s in two steps. In the first step, we insert the segment's endpoints as keys into the tree. This ensures that the leaves represent the new set of primitive intervals. As a result of a single key insertion, some leaf v is replaced by a new internal node u with two new leaves attached to it. (See Figure 2.2.) This corresponds to splitting the primitive interval represented by v. To preserve our invariants, we set $ccover(u)$ to $ccover(v)$ and the $ccover()$ values of the new leaves to 0. After the key insertion, the balance of the tree may need to be restored by rotations. Before a rotation, we push the $ccover()$ values of the participating nodes down to their children. (See Figure 2.3.) This ensures that the sum invariant is preserved during the rotation; however, the push-up invariant may be violated. After the rotation, we restore the push-up

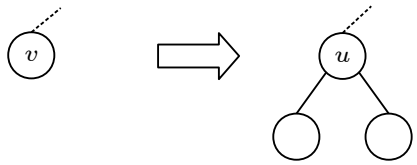


Figure 2.2: A key insertion.

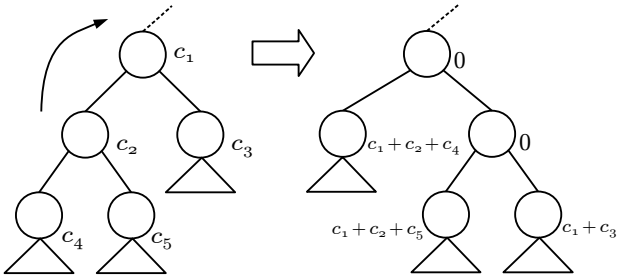


Figure 2.3: A rotation.

invariant by performing push-ups on the participating nodes. The total cost of the key insertion is then $O(\log n)$.

After the key insertions, the structure still stores the old measure. In the second step, we insert s into our structure. This operation is applied as in segment trees: segment s is associated with a set $\mathcal{C}(s)$ of *canonical* nodes such that for every node v in $\mathcal{C}(s)$, s covers the interval of v but not its parent's. There are logarithmically many such nodes and they can be traversed in logarithmic time [35]. Moreover, the intervals associated with the canonical nodes are disjoint and their union equals s. We increment the $ccover()$ value at each canonical node to preserve the sum invariant, then apply push-ups at these nodes and their ancestors to restore the push-up invariant. Since the number of

ancestors of the canonical nodes is logarithmic, the total cost of the insertion operation is $O(\log n)$.

Similarly to insertion, deletion is also performed in two steps. This is the inverse of insertion—we decrement $ccover(v)$ for each canonical node $v \in \mathcal{C}(s)$. We then perform push-ups at all the nodes of $\mathcal{C}(s)$ and their ancestors to restore the push-up invariant. One issue that is worth mentioning is that the $ccover()$ values of the canonical nodes may drop below zero during deletion. We now argue that after the push-ups, all $ccover()$ values in the tree remain non-negative. Notice that the push-ups push all the negativity up to the root. This negativity is guaranteed to be cancelled out at the root, because the push-up invariant ensures that $ccover(root)$ is non-negative. To see this, imagine a path starting at root, drawn by following the children whose $ccover()$ values are zero. The destination of this path is a leaf whose $cover()$ value equals $ccover(root)$. Then, $ccover(root)$ should be non-negative.

After deleting s, we remove the keys corresponding to its endpoints from the tree as follows. Let v be the node whose key is to be deleted. Let u be the rightmost leaf in the left subtree of v, and let w be the leftmost leaf in the right subtree of v. Notice that the deletion corresponds to the merge of the primitive intervals of w and u. Let u' be the father and u'' be the sibling of u. (See Figure 2.4.) Following the standard deletion procedure for binary search trees, we replace the key of v with the key of u', which is the largest key in v's left subtree. Afterwards, we replace u' with u''. Thus u'

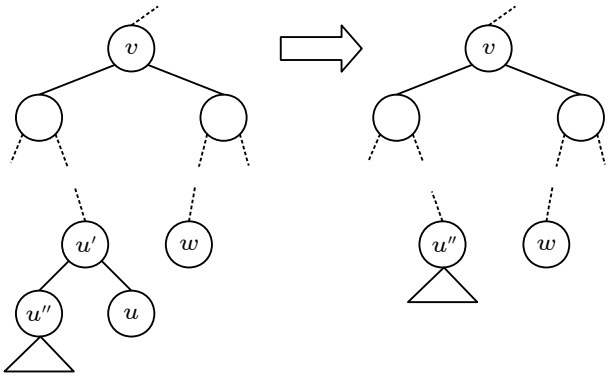


Figure 2.4: The deletion.

and u are deleted, and w now corresponds to the merged primitive interval. In fact, the intervals of all the nodes on the rightmost path of the left subtree of v and the leftmost path of the right subtree of v have changed. (Recall that one has to recompute $\mu()$ values for these nodes and their ancestors). In accordance with the structural changes, we set $ccover(u'')$ to $ccover(u'') + ccover(u')$ to preserve the sum invariant. The push-up invariant may be violated at u'', so we apply push-ups to u'' and possibly to its ancestors. Any rotations needed to balance the structure are handled as in the insertion case. It easily follows that the cost of a segment deletion is also $O(\log n)$.

We now can deduce the following lemma.

Lemma 2.3.2. *We can maintain the measure of a dynamic set of n segments in $O(\log n)$ time for insert or delete operations, and $O(1)$ time for measure query.*

We next show how to generalize the anonymous segment tree to deal with probabilistic segments. Towards that goal, we introduce an abstract framework that includes the measure of probabilistic segments as a special case.

2.3.2 An Abstract Anonymous Segment Tree

Let f be a function mapping the segments in $\mathcal{S}$ to some range set G, and let $\oplus$ be a commutative and associative binary operation on G. We consider the problem of maintaining the following sum for each primitive interval I of the set $\mathcal{S}$: $F(v) = \sum_{s \in \mathcal{S}_I} f(s)$, where v is the leaf associated with I and the summation uses the $\oplus$ operation.[9]

We compute $F(v)$ indirectly by storing a quantity called $FF(v)$ at each node v of the tree, and maintain the invariant that the sum of $FF(a)$ over all ancestors a of v equals $F(v)$. (This is the equivalent of the sum invariant from Section 2.3.1.) However, such a maintenance needs additional requirements to be efficient. We require that $\oplus$ is invertible, and there is a total order $\leq_G$ on G such that

$$A \leq_G B \iff A \oplus C \leq_G B \oplus C. \qquad (*)$$

[9]Observe that if G is the set of integers, $\oplus$ is integer addition, and $f(s) = 1$ for every s, then $F(v) = cover(v)$, as in the preceding section.

In other words, $(G, \oplus, \leq_G)$ forms a *totally ordered abelian group*. Finally, we reduce the range of $f()$ from G to G^+, defined as $G^+ = \{g \mid g \in G \ \wedge \ e \leq_G g\}$, where e is the identity element of $\oplus$.

The push-up invariant in this abstract setting is that, for each internal node v with children v_l and v_r, at least one of $FF(v_l)$ and $FF(v_r)$ is e and the other is in G^+. Repeated push-up operations in the tree, starting from the leaves, establish this invariant. In particular, let v be an internal node with children v_l and v_r, and without loss of generality assume that $FF(v_l) \leq_G FF(v_r)$. The push-up operation sets $FF(v) = FF(v) \oplus FF(v_l)$, $FF(v_l) = e$ and $FF(v_r) = FF(v_r) \oplus FF(v_l)^{-1}$, where $^{-1}$ denotes the inverse with respect to $\oplus$. Due to condition (*), this operation restores the push-up invariant in v. Following the same ideas in Section 2.3.1, one can easily show that setting $FF(v) = F(v)$ for all leaves v in the tree and then applying push-ups in bottom-up fashion produces a configuration of $FF()$ values that satisfy both the sum and the push-up invariants.

Under this scheme, we handle the insertion of a segment s as follows. We first insert the endpoints of s as keys to the tree. During the key insertions, we preserve our invariants the same way we did in Section 2.3.1. Then, s itself is inserted to the tree. We do this by adding $f(s)$ to $FF(v)$ for each canonical node v of s. The invariants are preserved by push-ups as usual.

In the case of deletion, we first remove s from the tree by adding $f(s)^{-1}$ to $FF(v)$ for each canonical node v of s. Notice that the $FF()$ values of the canonical nodes may get negative (i.e., outside G^+). However, similar to the case in Section 2.3.1, the push-up invariant guarantees that all $FF()$ values remain non-negative after the push-ups. Once s is removed from the tree, we remove the keys corresponding to the endpoints of s and this is done as in Section 2.3.1. It is easy to verify that the insertion and deletion operations cost $O(\log n)$ time.

2.3.3 Measure of Probabilistic Segments

We now show how to use the abstract anonymous segment tree framework for maintaining the measure of probabilistic segments. For the sake of simplicity, we maintain the *complement* of the expected measure: the expected value of the length *not* covered by any active segment.[10] In order to maintain the measure for probabilistic segments, we apply our abstract framework twice. First, for each leaf v, we maintain the number of segments that cover its interval *and* have probability 1. We denote this by $cover(v)$, and use the deterministic coverage count algorithm to maintain it. Second, we maintain the probability that the primitive interval of a leaf v is uncovered by the segments whose probability is strictly less than 1. (The segments with probability 1 are handled separately, and more easily.) We denote this quantity by $prob(v)$, and maintain it us-

[10] We assume that all segments are contained in a finite, bounded range, ensuring that the complement is bounded.

ing our generalized scheme as follows. We define G as the set of positive reals, $\oplus$ as multiplication, $\leq_G$ as $\geq$, and set $f(s)$ to

$$f(s) = \begin{cases} (1 - p_s) & \text{if } p_s < 1 \\ 1 & \text{if } p_s = 1 \end{cases}$$

Observe that $F(v)$ represents $prob(v)$. For ease of reference we denote the $FF(v)$ values used to maintain $prob()$ by $pprob(v)$. We can define the *uncovered measure* of a node v, denoted $\nu(v)$, recursively as follows:

$$\nu(v) = \begin{cases} 0 & \text{if } ccover(v) > 0 \\ pprob(v) \cdot L(v) & \text{if } ccover(v) = 0 \ \wedge \ v \text{ is a leaf} \\ pprob(v) \cdot (\nu(v_l) + \nu(v_r)) & \text{if } ccover(v) = 0 \ \wedge \ v \text{ has children } v_l \text{ and } v_r \end{cases} \tag{2.2}$$

Lemma 2.3.3. *Let $\bar{\mu}(v)$ denote the complement of the expected measure of $\mathcal{S}$ restricted to v's interval, let $aprob(v)$ be the product of $pprob(a)$ over all ancestors a of v (excluding v), and let a node v be called* exposed *if $ccover(a) = 0$ at all strict ancestors a of v. Then, for any exposed node v, we have $\bar{\mu}(v) = aprob(v) \cdot \nu(v)$.*

Proof. If an exposed node v has $ccover(v) > 0$, then $\bar{\mu}(v) = 0$. By the first line of (2.2), $\bar{\mu}(v)$ equals $aprob(v) \cdot \nu(v)$. Now consider an exposed leaf v such that $ccover(v) = 0$.

Then $cover(v) = 0$. We write

$$\bar{\mu}(v) \;=\; prob(v) \cdot L(v) \;=\; aprob(v) \cdot pprob(v) \cdot L(v)$$

By the second line of (2.2), this expression equals $aprob(v) \cdot \nu(v)$. Finally, consider an exposed internal node v such that $ccover(v) = 0$. Then v_l and v_r are exposed. By induction, $\bar{\mu}(v_l) = aprob(v_l) \cdot \nu(v_l)$ and $\bar{\mu}(v_r) = aprob(v_r) \cdot \nu(v_r)$. Then

$$\begin{aligned} \bar{\mu}(v) \;&=\; \bar{\mu}(v_l) + \bar{\mu}(v_r) \;=\; aprob(v_l) \cdot \nu(v_l) + aprob(v_r) \cdot \nu(v_r) \\ &=\; aprob(v) \cdot pprob(v) \cdot (\nu(v_l) + \nu(v_r)) \end{aligned}$$

By the third line of (2.2), the expression equals $aprob(v) \cdot \nu(v)$. □

Corollary 2.3.4. *$\nu(root)$ equals the complement of the expected measure of $\mathcal{S}$.*

By Corollary 2.3.4, one can report the expected measure of $\mathcal{S}$ simply by returning the complement of $\nu(root)$. By maintaining $\nu()$ the same way we maintain $\mu()$ in Section 2.3.1, we end up with a structure that solves the uncertain measure problem:

Theorem 2.3.5. *The expected measure of a dynamic set of segments can be maintained in $O(1)$ query time, $O(\log n)$ insertion/deletion time and $O(n)$ space.*

2.4 Dynamic Expected Volume in d Dimensions

We now show how to maintain the expected volume of the union of a dynamic set of uncertain boxes in d-space. The main idea is to adapt the framework of Overmars

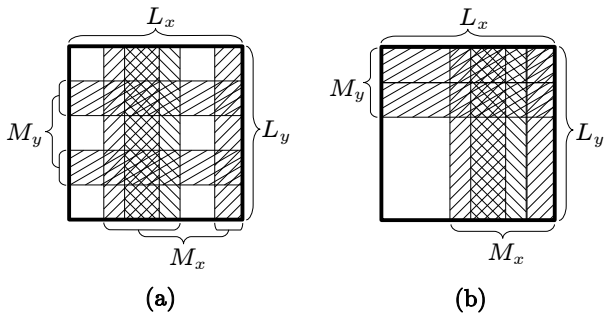


Figure 2.5: (a) A two-dimensional trellis formed by 5 boxes. (b) The shape with the same area formed by moving strips.

and Yap's solution for Klee's problem to a dynamic and probabilistic setting [64]. In the following three subsections, we explain this adaptation in detail.

2.4.1 The Trellis Structure

The first step is to apply Theorem 2.3.5 to a very special kind of d-dimensional box arrangement called a *trellis*. (This is the same arrangement discussed in Chapter 1, Section 1.5.3.) A trellis in d dimensions is a rectangular region R and a collection of boxes $\mathcal{B}$ that such that each box in $\mathcal{B}$ forms of an axis-parallel *strip* inside R. In other words, no $(d-2)$-dimensional face (a corner in two dimensions) of a box in $\mathcal{B}$ intersects the interior of R. A two-dimensional example is shown in Figure 2.5a, where each box is either a vertical or a horizontal strip.

The volume of a trellis is easy to compute efficiently. First consider the problem in two dimensions. Suppose the horizontal and vertical side lengths of R are L_x and L_y, respectively. Let M_x be the length of the portion of the x-interval of R covered by

the vertical strips, and M_y the length of R's y-interval covered by the horizontal strips. Then it is easy to see that the area covered in R is $L_x \times L_y - (L_x - M_x) \times (L_y - M_y)$. (A visual proof is offered in Figure 2.5b.)

It follows that computing the area of a trellis reduces to maintaining M_x and M_y separately, i.e., to solving two *one-dimensional* volume problems. In $d > 2$ dimensions, the volume formula for a trellis generalizes easily to

$$\prod_i L_i - \prod_i (L_i - M_i),$$

where the product index ranges from 1 to d, L_i is the side length of R along the ith axis and M_i is the sublength of L_i that is covered by strips orthogonal to the ith axis [64].

To maintain the expected volume within a trellis for uncertain boxes, we use the same formula, except that all the variables in the formula are replaced by their expectations. Specifically, the formula for the d-dimensional case becomes

$$\prod_{1 \le i \le d} L_i - \prod_{1 \le i \le d} (L_i - E(M_i))$$

where $E(M_i)$ is the expected value of M_i. Note that M_i's are independent. Then, by linearity and multiplicativity of expectation over independent variables, the formula correctly represents the expected volume.

It is clear that the expected volume in a trellis can be maintained in logarithmic time per update by using d instances of anonymous segment tree, each maintaining M_i for $1 \le i \le d$. While several efficient solutions are known for maintaining the one-

dimensional measure of *non-probabilistic* segments [30,45,73], they all seem quite specialized, focusing on a particular application. It is unclear whether they can be adapted to our probabilistic setting without fairly complicated modifications. The anonymous segment tree, on the other hand, offers a simple, and general, framework suitable for probabilistic measure maintenance.

2.4.2 Overmars-Yap Partition

The second step is to partition the space hierarchically such that the leaves of the partition contain trellis structures. This the same partition described in Section 1.5 of Chapter 1. We therefore only briefly redescribe the partition in this chapter and omit the proofs about its properties. We, however, warn the reader that we define some additional terms while describing the partition in this section, which will be useful for our complexity analysis.

The partition proceeds in d steps. Let us call a face of a box orthogonal to the ith axis an *i-face* and the hyperplane it sits on an *i-bound*. In the first step of the partition, we divide the space into regions called 1-slabs by cutting it with hyperplanes through every $\sqrt{n}$th 1-bound of the boxes along the first axis. Consequently, $O(\sqrt{n})$ 1-slabs are formed, each of which contains $O(\sqrt{n})$ 1-faces. In the second step, each 1-slab is split into 2-slabs by hyperplanes perpendicular to the second coordinate axis. These hyperplanes are introduced as follows: A hyperplane is drawn along every $\sqrt{n}$th 2-bound in

$\mathcal{B}$. Additionally, for each box B that has a 1-face inside the 1-slab, two hyperplanes are drawn along both of its 2-bounds. Consequently, each 1-slab is partitioned into $O(\sqrt{n})$ 2-slabs, each of which intersects with $O(\sqrt{n})$ 1-faces and 2-faces of the boxes in $\mathcal{B}$. In the third step, each 2-slab is partitioned into $O(\sqrt{n})$ 3-slabs. This time, the splitting hyperplanes pass along every $\sqrt{n}$th 3-bound and the 3-bounds of each box that has a 1-face or a 2-face intersecting the inside of the 2-slab. This partitioning strategy is continued until the dth step, in which each $(d-1)$-slab is divided into $O(\sqrt{n})$ cells. The following lemma, whose proof can be found in [64], summarizes the key properties of this orthogonal partition.

Lemma 2.4.1. *The orthogonal partition contains $O(n^{d/2})$ cells such that each box of $\mathcal{B}$ partially covers $O(n^{(d-1)/2})$ cells, each cell partially intersects $O(\sqrt{n})$ boxes in $\mathcal{B}$, and the boxes partially overlapping a cell form a trellis.*

We can now maintain the uncovered expected volume as follows. For each cell C, we maintain uncovered volume of the boxes that partially intersects C restricted to C. By using the trellis structure from Section 2.4.1, this is doable in logarithmic time per update and constant time per query. Since a box partially intersects $O(n^{(d-1)/2})$ cells, we can update all trellis structures in $O(n^{(d-1)/2} \log n)$ time during a box insertion/deletion.

The cells in a $(d-1)$-slab form a linear sequence. We can track the boxes that completely overlap the cells of a $(d-1)$-slab using a structure called a *slab tree*, which

is just an anonymous segment tree with trellises at its leaves. To be precise, the slab tree is obtained by the following modifications to the anonymous segment tree:

- The keys stored in the tree correspond to the d-bounds bounding the cells in the $(d-1)$-slab. Consequently, each leaf corresponds to a cell and each internal node corresponds to the union of the cells represented by its descendant leaves. In this case, $cover(v)$ is defined to be the number of boxes that have probability 1 and fully contain the cell of v. Similarly, $prob(v)$ stands for the probability that no box that has probability strictly less than 1 and fully contains the cell of v exists. $ccover()$ and $pprob()$ values are maintained accordingly.

- We redefine $\nu(v)$ so that it represents an expected uncovered d-dimensional volume and also takes into account the volume inside the cells (trellises). In particular, we define $\nu(v)$ as:

$$\nu(v) = \begin{cases} 0 & \text{if } ccover(v) > 0 \\ pprob(v) \cdot \nu_T(v) & \text{if } ccover(v) = 0 \ \wedge \ v \text{ is a leaf} \\ pprob(v) \cdot (\nu(v_l) + \nu(v_r)) & \text{if } ccover(v) = 0 \ \wedge \ v \text{ has children} \\ & \quad v_l \text{ and } v_r \end{cases}$$

where $\nu_T(v)$ is the expected uncovered volume in the trellis structure associated with leaf v. We emphasize that the only change to the recursive definition of $\nu(v)$ is in the second line.

By following the same proof strategy for Lemma 2.3.3 and Corollary 2.3.4, one can easily prove the following lemma.

Lemma 2.4.2. *$\nu(root)$ is the expected uncovered volume inside the $(d-1)$-slab.*

The uncovered measure of the whole space is the sum of $\nu(r)$ over all the roots r of the $(d-1)$-slab trees. Updating a slab tree with a box takes logarithmic time. Since there are $O(n^{(d-1)/2})$ $(d-1)$-slabs, updating all slab trees takes $O(n^{(d-1)/2}\log n)$ during a box insertion/deletion. Also, when there is change in the volume of trellis, the hosting slab tree can be updated in $O(\log n)$ time to reflect this change. Since a box affects $O(n^{(d-1)/2})$ trellises, the total cost of this update is also $O(n^{(d-1)/2}\log n)$. This yields a total update time of $O(n^{(d-1)/2}\log n)$ for box insertions and deletions.

2.4.3 Dynamic Partition

We have a final missing ingredient: a dynamic version of the hierarchical orthogonal partition. At the high level, we maintain such a partition as follows. Recall that Overmars and Yap's partition is based on making slab cuts at the $\sqrt{n}$th coordinate in each dimension. We relax this constraint by maintaining the sorted sequence of box

coordinates in each dimension and cutting along a fairly stable—but not static—set of slab boundaries. The slab boundaries for each dimension partition the corresponding sorted coordinate sequence into *buckets*. We maintain the invariant that each bucket contains at most $2\sqrt{n}$ coordinates, and any two adjacent buckets contain at least $\sqrt{n}$. Whenever one of these invariants is violated, we split or merge buckets as necessary by introducing or removing slab boundaries. This invalidates some trellises and slab trees; we restore them by rebuilding. By a potential argument, it can be shown that the amortized cost of all the rebuilding is also $O(n^{(d-1)/2} \log n)$ time per box insertion/deletion, matching the direct cost of data structure updates. We now describe this algorithm in depth.

The main idea is to use a different sequence of i-bounds instead of the consecutive sequence of $\sqrt{n}$th i-bounds used in the original Overmars-Yap partition. Let $\mathcal{B}$ be a set of n boxes in the plane. We are interested in maintaining a sequence K_i of i-bounds for each $1 \leq i \leq d$ to replace the sequence of $\sqrt{n}$th i-bounds. Let I be the set of i-bounds in $\mathcal{B}$. At any time, K_i is a sorted sequence $(k_1, \ldots, k_m)$ of distinct i-bounds (not necessarily in I). The elements of K_i partition the i-bounds in I into a sequence $\mathcal{Q}_i = (Q_1, \ldots, Q_{m+1})$ of buckets. If we define sentinel i-bounds $k_0 = -\infty$ and $k_{m+1} = \infty$, then $Q_j = \{x \mid x \in I \ \wedge \ k_{j-1} < x \leq k_j\}$ for every $j \in [1, m+1]$.

Moreover, we require that the following size conditions hold on $\mathcal{Q}_i$:

- For each bucket Q_j, $|Q_j| \leq 2\sqrt{n} + 1$.

- For two adjacent buckets Q_j and Q_{j+1}, $|Q_j| + |Q_{j+1}| \geq \sqrt{n}$.

The elements of K_i are not updated as long as the size conditions are satisfied. An update is required in the following two cases:

- The size of a bucket Q_j exceeds $2\sqrt{n} + 1$. This is most likely due to an increase in the size of Q_j as a result of a box insertion into $\mathcal{B}$. However, this situation may also arise when a box is deleted from $\mathcal{B}$, reducing n. In either case, we resolve the situation by inserting the median of Q_j into K_i. Consequently, Q_j is split into two buckets of almost equal size.

- The total size of two adjacent buckets Q_j and Q_{j+1} drops below $\sqrt{n}$. This might be due to either a decrease in the size of one of these buckets or an increase in n. In either case, we merge Q_j and Q_{j+1} by removing k_j from K_i.

To keep track of when one of these events occurs, we maintain all buckets in a max-heap keyed on the bucket size, and we maintain all adjacent bucket pairs in a min-heap keyed on the total size of the pair.

It is easily seen that $|K_i|$ is $O(\sqrt{n})$ and the number of i-bounds in $\mathcal{B}$ between two consecutive elements of K_i is $O(\sqrt{n})$. These properties are precise enough that we can use K_is to define an orthogonal partition that is asymptotically equivalent to the static partition that we described. This partition is defined as follows. The space is divided into $O(\sqrt{n})$ 1-slabs by cuts along each 1-bound in K_1. Each 1-slab is divided

into $O(\sqrt{n})$ 2-slabs by cuts along each 2-bound in K_2 and the 2-bounds of each box that has a 1-face inside the 1-slab. Each 2-slab is divided into $O(\sqrt{n})$ 3-slabs by cuts along each 3-bound in K_3 and the 3-bounds of each box that has a 1-face or 2-face in the 2-slab. This process continues until the $(d-1)$-slabs are cut into cells. It can be easily shown that this orthogonal partition satisfies Lemma 2.4.1.

Remark: In the remainder of this discussion, we also use the term d-slab to refer to a cell of the partition.

During insertions and deletions of boxes, we rebuild portions of the partition together with the associated data structures whenever they become invalid. To make the analysis of these rebuilding operations easier, we first present the following lemma.

Lemma 2.4.3. *The following properties are true for the orthogonal partition.*

1. *There are $O(n^{i/2})$ i-slabs.*
2. *An i-slab contains $O(n^{(d-i)/2})$ cells and $O(n^{(d-i-1)/2})$ $(d-1)$-slabs for $i < d$.*
3. *A j-bound of a box possibly intersects the interior of $O(n^{(i-1)/2})$ i-slabs where $j \leq i$.*

Proof. Properties 1 and 2 are obvious from the structure of the orthogonal partition. To deduce Property 3, we consider the partition level by level. Observe that each level in the partition except the jth level, multiplies the number of slabs intersecting a particular

j-bound by a factor of $O(\sqrt{n})$. Thus, at the ith level (for $i \geq j$), a j-bound intersects $O(n^{(i-1)/2})$ i-slabs. □

In our modifications to the partition, we make use of the following rebuilding operations:

- **Split of an i-slab:** In this operation, we split an i-slab R (possibly a cell) into two new i-slabs R_1 and R_2 by cutting R through an i-bound. As part of this operation, we destroy the trellis structures and the slab tree of R and construct the trellises and the slab trees of R_1 and R_2 from scratch. By Lemma 2.4.3, there are $O(n^{(d-i)/2})$ trellis structures and $O(n^{(d-i-1)/2})$ slab trees in an i-slab. The costs of constructing a trellis and a slab tree are $O(\sqrt{n}\log n)$ and $O(n \log n)$ respectively, thus, the total cost of this operation is $O(n^{(d-i+1)/2}\log n)$.

- **Merge of two i-slabs:** In this operation, we merge two neighboring i-slabs R_1 and R_2 that are separated by a i-bound into a single i-slab R. As part of this operation, we destroy the trellis structures and the slab trees of R_1 and R_2 and construct the trellises and the slab tree of R from scratch. Similar to the split operation, the total cost is $O(n^{(d-i+1)/2}\log n)$.

The following list describes the events that invalidate our partition, along with the modifications they cause and the costs they incur:

1. For $1 \leq j \leq i \leq (d-1)$, a set $\mathcal{R}$ of i-slabs intersects a new particular j-face in their interior as the result of the insertion of a box B. In this case, we further partition each i-slab in $\mathcal{R}$ by introducing cuts through the $(i+1)$-bounds of B. As a result of each cut, an $(i+1)$-slab is split into two. By Lemma 2.4.3, there are $O(n^{(i-1)/2})$ i-slabs intersecting the new j-face. Thus, the overall cost of this event is $2 \times O(n^{(i-1)/2}) \times O(n^{(d-i)/2}) = O(n^{(d-1)/2} \log n)$.

2. For $1 \leq j \leq i \leq (d-1)$, a set $\mathcal{R}$ of i-slabs lose a particular j-face from their interior as the result of the deletion of a box B. In this case, for each i-slab in $\mathcal{R}$, we remove the cuts through the $(i+1)$-bounds of B. As a result of each cut removal, two $(i+1)$-slabs are merged into one. Similar to Event 1, the overall cost is $O(n^{(d-1)/2} \log n)$.

3. An i-bound x_i is inserted to K_i (where $1 \leq i \leq d$). In this case, we split each i-slab containing x_i into two new i-slabs by cutting it through x_i. By Lemma 2.4.3, there are $O(n^{(i-1)/2})$ such i-slabs. Then, the overall cost is $O(n^{(i-1)/2}) \times O(n^{(d-i)/2}) = O(n^{d/2} \log n)$.

4. An i-bound x_i is deleted from K_i. In this case, we merge each neighboring pair of i-slabs separated by x_i. Similar to Event 3, the total time spent is $O(n^{d/2} \log n)$.

This implies that the worst-case cost of rebuilding is $O(n^{(d-1)/2} \log n)$ per update if there are no changes in K_i for any i. However, each single change in K_i's costs

$O(n^{d/2} \log n)$ time in the worst-case. In the following lemma, we show that the frequency of updates to K_i's is so low that the amortized cost stays at $O(n^{(d-1)/2} \log n)$.

Lemma 2.4.4. *The amortized cost of a box insertion or deletion in the relaxed partition is* $O(n^{(d-1)/2} \log n)$.

Proof. Recall that each change in K_i is due to either a bucket split or merge. Our proof argument assigns a *potential* to each bucket in all K_i's. In addition to the actual $O(n^{(d-1)/2} \log n)$ time required by an insertion or deletion, each operation also contributes a similar amount to the bucket potentials; the potential associated with a bucket pays for the cost of bucket splits and merges.

The potential of a bucket Q is $\Phi(Q) = n^{(d-1)/2} \log n \times \big| |Q| - \sqrt{n} \big|$. When an i-bound is inserted or deleted, potentials are affected in three ways: (1) $|Q|$ changes by 1 for a single bucket; (2) n changes by 1 for all buckets; and (3) some buckets may be split or merged. We now analyze the effect on the potential for all three cases.

In the discussion below, we use the notation ΔA to denote amount of change in some quantity A after an update. Notice that if A is a function of n, i.e., $A = f(n)$ for some f, then ΔA is either $f(n+1) - f(n)$ or $f(n-1) - f(n)$. Then, $|\Delta(n^k)| = O(n^{k-1})$ and $|\Delta(\log n)| = O(1/n)$. Also, recall that $\Delta(AB) = A\Delta B + B\Delta A + \Delta A \cdot \Delta B$. Hence $\Delta(n^{(d-1)/2} \log n) = O(n^{(d-3)/2} + n^{(d-3)/2} \log n + n^{(d-5)/2}) = O(n^{(d-3)/2} \log n)$.

Consider the first two cases. If a bound is inserted into or deleted from a bucket Q, then $\Delta\big|\,|Q| - \sqrt{n}\big| \leq \big|\Delta|Q|\big| + \big|\Delta(\sqrt{n})\big| = 1 + O(1/\sqrt{n})$. Hence

$$\begin{aligned}
\Delta\Phi(Q) &= \Delta\left(n^{(d-1)/2}\log n \times \big||Q| - \sqrt{n}\big|\right) \\
&= n^{(d-1)/2}\log n \times \Delta\big||Q| - \sqrt{n}\big| \;+\; \Delta\left(n^{(d-1)/2}\log n\right) \times \big||Q| - \sqrt{n}\big| \\
&\quad + \Delta\left(n^{(d-1)/2}\log n\right) \times \Delta\big||Q| - \sqrt{n}\big| \\
&= n^{(d-1)/2}\log n \times (1 + O(1/\sqrt{n})) \;+\; O(n^{(d-3)/2}\log n) \times O(\sqrt{n}) \\
&\quad + O(n^{(d-3)/2}\log n) \times (1 + O(1/\sqrt{n})) \\
&= O(n^{(d-1)/2}\log n).
\end{aligned}$$

If n changes without affecting the contents of a bucket Q, then $\Delta\big|\,|Q| - \sqrt{n}\big| \leq |\Delta(\sqrt{n})| = O(1/\sqrt{n})$, and hence

$$\begin{aligned}
\Delta\Phi(Q) &= \Delta\left(n^{(d-1)/2}\log n \times \big||Q| - \sqrt{n}\big|\right) \\
&= n^{(d-1)/2}\log n \times \Delta\big||Q| - \sqrt{n}\big| \;+\; \Delta\left(n^{(d-1)/2}\log n\right) \times \big||Q| - \sqrt{n}\big| \\
&\quad + \Delta\left(n^{(d-1)/2}\log n\right) \times \Delta\big||Q| - \sqrt{n}\big| \\
&= n^{(d-1)/2}\log n \times O(1/\sqrt{n}) \;+\; O(n^{(d-3)/2}\log n) \times O(\sqrt{n}) \\
&\quad + O(n^{(d-3)/2}\log n) \times O(1/\sqrt{n}) \\
&= O(n^{(d-2)/2}\log n).
\end{aligned}$$

The total number of buckets is $O(\sqrt{n})$, so the total potential change over all buckets due to a single insertion or deletion is $O(n^{(d-1)/2}\log n)$. This change (if positive) is

charged to the insertion/deletion operation. It does not affect the asymptotic cost of the operation, since the direct data structure modifications take the same time.

Now consider the effect of a split or merge on a bucket's potential. When a bucket is about to be split, its size is greater than $2\sqrt{n}+1$, and so its potential is at least $n^{(d-1)/2}\log n(\sqrt{n}+1) \geq n^{d/2}\log n$. The size of the bucket is at most $2\sqrt{n}+2$, so the potential of each of the two buckets resulting from the split is at most $n^{(d-1)/2}\log n$. When two buckets are about to be merged, the sum of their sizes is less than $\sqrt{n}$, and so the sum of their potentials is at least $n^{(d-1)/2}\log n \times \sqrt{n} = n^{d/2}\log n$. The size of the merged bucket is at least $\sqrt{n}-1$, so its potential is at most $n^{(d-1)/2}\log n$. In both cases, the decrease in total potential is $\Omega(n^{d/2}\log n)$. This potential change is used to pay for the $O(n^{d/2}\log n)$ cost of the direct data structure manipulations to perform the split or merge. It follows that the amortized cost of inserting or deleting a bound is $O(n^{(d-1)/2}\log n)$. □

We can now state our final theorem.

Theorem 2.4.5. *The expected volume of the union of a dynamic set of n uncertain boxes can be maintained with a data structure that uses $O(n^{(d+1)/2})$ space, $O(1)$ time for query and $O(n^{(d-1)/2}\log n)$ amortized time for insertions and deletions of boxes.*

2.5 Conclusion

In this chapter, we considered the problem of maintaining the volume of the union of n boxes in d-space when each box is known to exist with an arbitrary, but independent, probability. We showed that, even in one dimension, computing the probability distribution, namely the probability that the volume equals (or exceeds) a given value, is *NP*-hard. On the other hand, we showed that the expected volume of the union can be maintained, nearly as efficiently as in the static and deterministic case. Along the way we introduced a data structure called *anomymous segment tree* that may be of independent interest in dealing with dynamic segment problems with abstract measures.

Our volume data structure was also implemented and evaluated experimentally [78]. In this evaluation, it has been shown that it performs as predicted by theory, and indeed significantly outperforms a naïve solution. On the other hand, the same experimental evaluation highlighted a particular limitation of an Overmars-Yap type of approach. The data structure is memory-intensive, which makes it unsuitable for large data sets. Thus, an interesting future research question is to explore better space-time tradeoffs that might yield scalable solutions to our dynamic and stochastic Klee's problem.

Chapter 3

A Discrete and Dynamic Version of Klee's Measure Problem*

3.1 Introduction

In this chapter, we consider a *discrete* and *dynamic* version of Klee's problem, in which the volume of a box is defined as the cardinality of its intersection with a finite point set $\mathcal{P}$, and both the boxes and the points are subject to insertion and deletion. In particular, we have a set of axis-aligned boxes $\mathcal{B} = \{B_1, B_2, \ldots, B_n\}$, a set of points $\mathcal{P} = \{p_1, p_2, \ldots, p_m\}$ in d-space, and we wish to maintain the *discrete measure* of

*This chapter is based on a joint work with John Hershberger and Subhash Suri. Parts of this chapter appeared in the following publications: [79] (Publication available at `http://2011.cccg.ca/proceedings/main.pdf`.)

$\mathcal{B}$ with respect to $\mathcal{P}$, namely, $meas(\mathcal{B}, \mathcal{P}) = |\mathcal{P} \cap \{\bigcup_{i=1}^{n} B_i\}|$, under insertion and deletion of both points and boxes.

The problem is fundamental, and arises naturally in several applications dealing with multi-attribute data. In databases, for instances, data records with d independent attributes are viewed as d-dimensional points, and selection rules are given as ranges over these attributes. A conjunction of ranges over d attributes is then equivalent to a d-dimensional box. Given a set of selection rules, the problem of *counting* all the data records that satisfy the union (namely, the disjunction) of all the rules is our discrete measure problem. Similarly, one may ask for the set of records that fail to satisfy *any of the rules*, and thus form the set of points "not covered" by the union of boxes.

Similarly, the management of firewall rules for network access can also be formulated as a discrete measure problem. The data packets in the Internet are classified by a small number of fields, such as IP address of the source and destination, the network port number, etc. The managers of a local area network (LAN) use a number of "firewall rules" based on these attributes to block some external services (such as ftp) from their network. The discrete measure problem in this setting keeps track of the number of services blocked by all the firewall rules; conversely, one can keep track of the number of services that become "exposed" by the deletion of a box.

Problem Formulation and Our Results

We begin with a formal definition of the problem. A d-dimensional *box* B is the Cartesian product of d one-dimensional ranges, namely $B = \prod_{i=1}^{d}[a_i, b_i]$, where a_i and b_i are reals. The *discrete measure* of a single box B with respect to a finite set of points $\mathcal{P}$ is the cardinality of the intersection $\mathcal{P} \cap B$. The discrete measure of the set of boxes $\mathcal{B}$ with respect to $\mathcal{P}$, denoted $meas(\mathcal{B}, \mathcal{P})$, is the cardinality of $\mathcal{P} \cap \{\bigcup_{B \in \mathcal{B}} B\}$. (Because a point may lie in multiple boxes, the discrete measure of $\mathcal{B}$ is not the sum of the measures of the individual boxes.) In this chapter, we consider the problem of maintaining the discrete measure under insertion and deletion of both points and boxes. Specifically, we propose a data structure that supports modifying $\mathcal{P}$ through insertion or deletion of a point, modifying $\mathcal{B}$ through insertion or deletion of a box, and querying for the current discrete measure $meas(\mathcal{B}, \mathcal{P})$.

Despite its natural formulation, the problem appears not to have been studied in this form. This may be partially attributed to the fact that the *static* version of the problem is easy to solve using standard data structures of computational geometry: build a d-dimensional version of a segment tree for the set of boxes, and then query separately for each point to determine whether any box contains it, for a total of $O(n \log^d n + m \log^{d-1} n)$ time. This approach, however, is inefficient when the set of boxes is dynamic, because each insertion or deletion can affect a large number of points, requiring $\Omega(m)$ recomputation time per update.

A technique of Chan [22] can be used to solve the dynamic problem. In [22], he describes a data structure for maintaining a set of points and a set of hyperplanes in d-space to answer queries of the form "does any of the points lie below the lower envelope of the hyperplanes." One can use this data structure in combination with standard range searching structures and a dynamization technique by Overmars and van Leeuwen [63] to solve our discrete measure problem so that point insertions and box updates require $O(\log^2 m + \log^d n)$ and $O(m^{1-\frac{1}{d}} \log m + \log^d n)$ time respectively.[11]

Compared to this bound, the time complexity of our data structure is better by a factor of $\log m$. However, a more important contribution may be the simplicity of our method and the fact that it solves the problem in a more *direct* way, making it more appealing for implementation. Specifically, our result gives a dynamic data structure for the discrete measure problem with the following performance: a box can be inserted or deleted in time $O(m^{1-\frac{1}{d}} + \log^d n)$; a point can be inserted in time $O(\log m + \log^d n)$ and deleted in time $O(\log m)$. The data structure always updates its measure, so a query takes $O(1)$ time.

The data structure also solves the *reporting* problem in output-sensitive time. Specifically, if k is the number of points in the union of the boxes, then they can be found in $O(k + k \log \frac{m}{k})$ worst-case time. The same bound also holds if one wants to report the points *not* contained in the union. Finally, we extend our results to a *uncertain* version

[11] Reducing box update time is possible at the expense of increasing the cost of point insertions and vice versa.

of the problem, in which each point and each box is associated with an independent probability of being present. In this case, one can naturally define an *expected discrete measure*, which is the expected number of points present that are covered by the union of the boxes present. Our bounds for the uncertain case are the same as the deterministic one.

Chapter Organization

In Section 3.2, we describe our solution for the dynamic version of the discrete measure problem. In Section 3.3, we discuss the extension of our solution to reporting queries and uncertain measure maintenance. We sum up with a conclusion in Section 3.4.

3.2 Maintaining the Discrete Measure

In the following discussion we assume that all the boxes in $\mathcal{B}$ and points in $\mathcal{P}$ have distinct coordinates.[12] Before we describe our dynamic data structure, it is helpful to consider a solution for the static problem. Let $\mathcal{B}$ be a set of n boxes and $\mathcal{P}$ a set of m points in d-space. For each point $p \in \mathcal{P}$, we define its *stabbing count*, denoted $stab(p)$, as the number of boxes in $\mathcal{B}$ that contain p. The *measure* of a single point p,

[12]This assumption merely simplifies the presentation; one can use symbolic perturbation to break ties between identical coordinates without affecting the result.

$meas(\mathcal{B}, \{p\})$, is 1 if $stab(p) > 0$ and 0 otherwise. One can easily see that the overall discrete measure can be written as the sum of point measures; that is, $meas(\mathcal{B}, \mathcal{P}) = \sum_{p \in \mathcal{P}} meas(\mathcal{B}, \{p\})$. The stabbing count of a point can be efficiently obtained using a *multi-level segment tree* [72], which achieves the following performance bounds.

Lemma 3.2.1 ([72])**.** *The multi-level segment tree represents a set of n boxes in d-space. The structure can report the stabbing count of any query point in $O(\log^{d-1} n)$ time. It requires $O(n \log^{d-1} n)$ space and $O(n \log^{d} n)$ preprocessing time for construction.*

By building a multi-level segment tree and then querying it for the stabbing count of each point in $\mathcal{P}$, we can calculate the measure $meas(\mathcal{B}, \mathcal{P})$ for the static problem in $O(n \log^{d} n + m \log^{d-1} n)$ time using $O(n \log^{d-1} n)$ space.

3.2.1 Invariants for Stabbing and Measure

The static solution described above loses its appeal in the dynamic setting because each box insertion or deletion can invalidate the stabbing count of $\Omega(m)$ points. We circumvent this problem by storing the stabbing counts *indirectly*, using an idea from *anonymous segment trees* (presented in Chapter 2, Section 2.3), so that only a small number of these indirect values need to be modified after a box update. We describe the technique in general first, deferring its specialization for the efficient maintenance of the discrete measure until later.

Consider a balanced tree (not necessarily binary) whose leaves are in one-to-one correspondence with the points of $\mathcal{P}$. The point corresponding to a leaf v is denoted p_v. In order to represent the stabbing counts of the points, we store a *non-negative* integer field $\sigma(w)$ at each node w of the tree subject to the following *sum invariant*: *for each leaf v, the sum of $\sigma(a)$ over all ancestors a of v (including v itself) equals $stab(p_v)$*. By assigning $\sigma(v) = stab(p_v)$ to each leaf v and $\sigma(w) = 0$ to all internal nodes w, we may obtain a trivial assignment with the sum invariant. But, as we will see, the flexibility afforded by these σ values allows us to update the stabbing counts of many points by modifying only a few σ values. As an example, if a box covering all the points of $\mathcal{P}$ were inserted, then incrementing the single value $\sigma(root)$ by 1 suffices, where $root$ denotes the root of the tree.

We will maintain the discrete measure, $meas(\mathcal{B}, \mathcal{P})$, through the σ values. In particular, at each node v, we store a quantity $\bar{\mu}(v)$ representing the number of points that have a stabbing count of 0, considering only the information stored in the subtree rooted at v. (The notation $\bar{\mu}$ is meant to suggest that it represents the complement of the measure.) The quantity $\bar{\mu}(v)$ is defined recursively using the σ values as follows:

$$\bar{\mu}(v) = \begin{cases} 0 & \text{if } \sigma(v) > 0 \\ 1 & \text{if } \sigma(v) = 0 \ \wedge \ v \text{ is a leaf} \\ \sum_{w \in child(v)} \bar{\mu}(w) & \text{if } \sigma(v) = 0 \ \wedge \ v \text{ is an internal node} \end{cases}$$

where $child(v)$ represents the children of a non-leaf node v. The following lemma is used to show that $\bar{\mu}(root)$ is the number of points in $\mathcal{P}$ whose stabbing counts are 0.

Lemma 3.2.2. *Let a node v be called* exposed *if for all ancestors a of v (excluding v) $\sigma(a)$ is zero. Then, for an exposed node v, $\bar{\mu}(v)$ is the number of points in v's subtree whose stabbing counts are 0.*

Proof. If $\sigma(v) > 0$, then $\bar{\mu}(v) = 0$ and $stab(p_w)$ is positive for all leaves w in v's subtree by the sum invariant, and thus the lemma holds. If v is a leaf with $\sigma(v) = 0$, then $stab(p_v) = 0$ and the lemma again holds. Otherwise, v is an internal node with $\bar{\mu}(v) = 0$. It follows that v's children are exposed. Let $N_0(v)$ be the number of points in v's subtree with zero stabbing count. Then, we can write by induction:

$$\bar{\mu}(v) = \sum_{w \in child(v)} \bar{\mu}(w) = \sum_{w \in child(v)} N_0(w) = N_0(v)$$

The lemma follows. □

Corollary 3.2.3. *$\bar{\mu}(root)$ is the total number of points with zero stabbing count.*

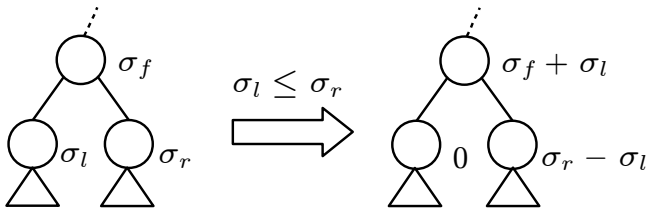


Figure 3.4: The push-up operation on a node with two children.

Consequently, $meas(\mathcal{B}, \mathcal{P}) = m - \bar{\mu}(root)$, and one can report $meas(\mathcal{B}, \mathcal{P})$ in $O(1)$ time.

We add one final constraint on σ values to achieve uniqueness, which also contributes to the efficiency of our specialized structure. In particular, we push the σ values as high up the tree as possible to enforce the following *push-up invariant*: *at least one child of every non-leaf node v has a σ value of* 0. This specifies σ uniquely, as shown by the following lemma.

Lemma 3.2.4. *Let T be a tree representing a set of points $\mathcal{P}$ and their stabbing counts as described above. Then there exists a unique configuration of σ values satisfying the sum and the push-up invariants in T.*

Proof. We first prove the existence of the desired configuration. Consider an arbitrary configuration of σ values satisfying the sum invariant. (For instance, $\sigma(v) = stab(p_v)$ for each leaf and $\sigma(v) = 0$ for each non-leaf.) We then apply the following *push-up* operation at each non-leaf node v to revise its value: increment $\sigma(v)$ by Δ and decrement $\sigma(w)$ by Δ for each child w of v, where $\Delta = \min_{w \in child(v)} \sigma(w)$. (This

is the same push-up operation presented in Chapter 2, Section 2.3.) See Figure 3.4. This achieves the push-up invariant at v while preserving the sum invariant in the tree. Repeated applications of the push-up operation from the leaves to the root produce a configuration of σ values satisfying both invariants.

For the uniqueness of this configuration, consider any node v. Let $parent(v)$ be the parent of v and $minstab(v)$ the minimum stabbing count in the subtree rooted at v. By the push-up invariant, there is a path in the tree from v to a leaf w such that every node on the path below v has $\sigma = 0$. Since this path clearly has the minimum possible sum of σ values, the stabbing count $stab(p_w)$ must be the one achieving $minstab(v)$. Because all σ values on the path are zero, the sum of $\sigma(a)$ over all ancestors a of v (including v itself) equals $stab(p_w) = minstab(v)$. By this reasoning, $\sigma(v) = minstab(v) - minstab(parent(v))$ for every node v except the root node. Therefore, the σ values are directly and uniquely determined by the configuration of stabbing counts. □

3.2.2 The Measure Tree and Dynamic Updates

In order to allow efficient insertion and deletion of boxes, and the corresponding updates of the points' stabbing counts, we organize $\mathcal{P}$ in a balanced tree that supports efficient range queries. A k-d tree, where points are stored at the leaves, allows efficient range queries, but is inefficient for insertion and deletion of *points*.[13] The structure we

[13]There is also no easy way to implement our scheme using range trees because they contain multiple copies of the points.

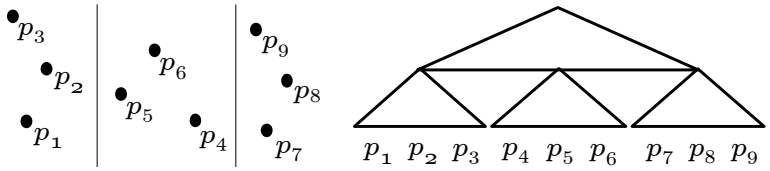


Figure 3.5: A measure tree of 9 points on the plane.

propose, which we call a *measure tree*, is a variant of divided k-d trees [74], and allows both efficient range queries and updates on the set of points. We note that the tree described in this section has slightly slower amortized bounds but these can be easily improved to achieve our main result as explained in Section 3.2.5.

We describe the measure tree in two dimensions for simplicity; the extension to d dimensions is conceptually straightforward, but we defer those details for later. Given a dynamic set of points $\mathcal{P}$ in the plane, we represent $\mathcal{P}$ as a two-level tree. The first level consists of an upper tree that partitions the points of $\mathcal{P}$ into at most $2\sqrt{m}$ subsets along the x-axis, each containing at most $2\sqrt{m}$ points, where m is the current size of $\mathcal{P}$. Each leaf of the upper tree acts as a root for a lower level tree that further partitions the corresponding subset of points using their y-coordinates. These lower trees form the second level of our tree. Figure 3.5 shows an example. Both levels of the tree are organized using 2-3 trees in which each data element is stored in a single leaf. Consequently, each leaf of the measure tree corresponds to a single point of $\mathcal{P}$ and we can use our measure maintenance scheme to store σ and $\bar{\mu}$ values on the nodes. We now discuss how to perform updates on the measure tree while preserving the invariants.

Insertion or Deletion of a box B. Let us consider insertion first. We find a set $\mathcal{C}$ of subtrees whose leaves correspond to the points covered by B. This is a range query, where we first perform a one-dimensional range search on the upper tree to locate the subsets of points that are completely or partially covered by the x-range of B. Observe that at most two subsets are partially covered. We then search the lower level trees corresponding to the partially covered subsets to find the points contained in B. The leaves corresponding to these points are included in $\mathcal{C}$. For each subset that is completely covered by the x-range of B, we perform a one-dimensional range search on the corresponding lower tree to find a set of maximal subtrees containing the points that lie in B. These maximal subtrees are also included in $\mathcal{C}$. It is straightforward to show that the subtrees in $\mathcal{C}$ span the set of points covered by B and the total cost of the range query is $O(\sqrt{m}\log m)$.

The insertion of B causes the stabbing count of each point contained in B to increase by 1. We effect this by incrementing the σ value of the root of each subtree in $\mathcal{C}$ by 1. This corrects the sum invariant in the tree, but may invalidate the push-up invariant. We therefore apply push-ups on the nodes whose σ values are updated. Since each push-up may introduce a violation of the push-up invariant at the parent, we continue applying push-ups until all violations are resolved. Finally, we recompute $\bar{\mu}$ for all ancestors of nodes whose σ values changed. This recomputation is also done bottom-up, since the $\bar{\mu}$ value of a node depends on the $\bar{\mu}$ values of its descendants. We

note that both the push-ups and the recomputations of $\bar{\mu}$ values can be done as part of the tree traversal of the range query. It follows that the total cost of the box insertion is $O(\sqrt{m}\log m)$ time.

The handling of deletion is similar to insertion, except that we decrement the σ value of the root of each subtree found by the range query. The time complexity is $O(\sqrt{m}\log m)$, as for insertion. Decrementing the σ's may cause some values to drop below zero, but the push-up operations eliminate these negative values. In particular, observe that a push-up at a node v restores not only the push-up invariant but also the non-negativity of v's children. To see that the final value of $\sigma(root)$ is non-negative, imagine a root-to-leaf path (as in the proof of uniqueness for Lemma 3.2.4) such that σ is zero for all nodes on the path except *root*. The path ends at a leaf v such that $stab(p_v)$ equals $\sigma(root)$, and so it follows that $\sigma(root)$ is non-negative.

Insertion or Deletion of a point p. When inserting a point p, we search the upper tree with the x-coordinate of p to find the lower tree in which p should be inserted, and then insert p using the standard 2-3 tree insertion algorithm. This creates a new leaf v with $p_v = p$. We need to know the stabbing count of p in order to initialize $\sigma(v)$ correctly. For the moment, let us assume that we know $stab(p)$—see Lemma 3.2.7—and focus on the update of the tree. In order to preserve the sum invariant, we set $\sigma(v)$ to $stab(p) - \Sigma$, where Σ is the sum of $\sigma(a)$ over all strict ancestors a of v. If $\sigma(v)$ is

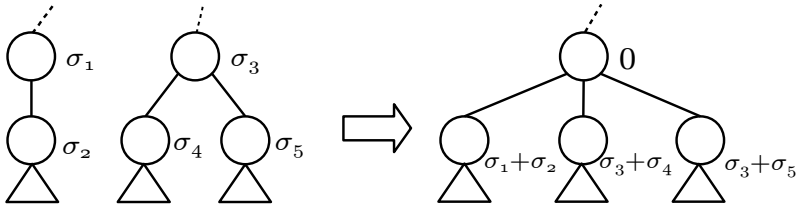


Figure 3.6: Push-down in a merge.

less than 0, we apply push-ups to v and all of its ancestors to push the negativity to the root, where it is canceled out. The 2-3 tree insertion may split one or more ancestors of v, and during those splits, the σ values of the resulting nodes are set to the original node's σ value, thereby preserving the sum invariant. After the split, we apply push-ups on the resulting nodes to re-establish the push-up invariant. Altogether, $O(\log m)$ splits and push-ups are performed, and so the cost of the insertion is $O(\log m)$. The insertion might cause the size of the lower tree to exceed $2\sqrt{m}$, but this is discussed in Section 3.2.3.

When a point p is deleted, we locate the lower tree containing it and simply delete the leaf corresponding to p, and restructure the tree to reestablish the balance of the 2-3 tree. The sum invariant is unaffected by the deletion, but we may need to apply push-ups to the ancestors of v to restore the push-up invariant. The deletion may also cause 2-3 tree merge or redistribution operations, and to preserve the sum invariant during these operations, we push the σ values of the participating nodes down to their children (see Figure 3.6). After the operation, push-ups are applied on these nodes to restore

the push-up invariant. If the lower tree becomes empty as a result of the deletion, we simply delete it and apply the same deletion algorithm on the upper tree. Due to the decrease in the value of m, the sizes of some upper or lower trees may exceed $2\sqrt{m}$; we deal with this in Section 3.2.3.

3.2.3 Complexity Analysis

We use two types of operations to ensure that the upper and the lower trees do not exceed their size thresholds. First, when a lower tree's size exceeds $2\sqrt{m}$, we split it into two new lower trees of equal size, destroying the original tree and constructing the new trees from scratch. During this process, we traverse the original tree to obtain the stabbing counts of the points and use those to construct the new trees. This split operation takes $O(\sqrt{m})$ time since the y-order of the points is known. Second, we avoid violating the upper tree's threshold by periodically rebuilding the entire measure tree. This reconstruction creates a lower tree for each set of $\lceil \sqrt{m} \rceil$ points along the x-axis (except perhaps the last one in the sequence, which may be smaller). Consequently, the size of the upper tree is at most $\sqrt{m}$. The reconstruction takes $O(m \log m)$ time. (It can be done in $O(m)$ time if we maintain the x- and y-orders of the points separately.) We determine when to do the reconstruction as follows. Assume that the most recent reconstruction of the tree was done when $m = m_0$. We reconstruct the tree after $\frac{1}{5}m_0$

point insertions or deletions. As the following two lemmas show, this ensures that the upper tree does not exceed its threshold.

Lemma 3.2.5. *Consider a measure tree constructed for m_0 points, where m_0 is sufficiently large. Then a newly constructed lower tree T requires at least $\frac{1}{2}\sqrt{m_0}$ point insertions into T before T splits.*

Proof. T has at most $\sqrt{m}+1$ leaves when it is constructed and at least $2\sqrt{m}$ leaves when it is split. Moreover, m satisfies $\frac{4}{5}m_0 \leq m \leq \frac{6}{5}m_0$. Consequently, there must be at least $(2\sqrt{\frac{4}{5}m_0} - (\sqrt{\frac{6}{5}m_0}+1))$ point insertions into T before it splits. For sufficiently large m_0, this quantity is lower-bounded by $\frac{1}{2}\sqrt{m_0}$. The lemma follows. □

Lemma 3.2.6. *The size of the upper tree is at most $2\sqrt{m}$.*

Proof. When the tree is constructed, it contains at most $\sqrt{m_0}$ lower trees. By Lemma 3.2.5, the number of additional lower trees constructed during the lifetime of the tree is at most $\frac{1}{5}m_0/(\frac{1}{2}\sqrt{m_0}) = \frac{2}{5}\sqrt{m_0}$. Thus, the size of the upper tree at any time is at most $\frac{7}{5}\sqrt{m_0} \leq \frac{7}{5}\sqrt{\frac{5}{4}m} \leq 2\sqrt{m}$. □

Next, we discuss how to initialize the stabbing count of a point when it is first inserted. We enable this by maintaining a separate *dynamic multi-level segment tree* [36], which provides the following functions dynamically.

Lemma 3.2.7 ([36])**.** *The dynamic multi-level segment tree represents a dynamic set of n boxes in d-space. The structure uses $O(n \log^{d-1} n)$ space and can report the stabbing*

count of any query point in $O(\log^d n)$ *time. It supports insertion or deletion of boxes in* $O(\log^d n)$ *time apiece.*

Putting together these pieces, we obtain our main result in two dimensions.

Theorem 3.2.8. *We can maintain the discrete measure in two-dimensional space using* $O(n \log n + m)$ *space, with constant time measure queries,* $O(\log^2 n + \sqrt{m} \log m)$ *time for insertion or deletion of a box,* $O(\log^2 n + \log m)$ *time for a point insertion, and* $O(\log m)$ *time for a point deletion time. (The* $\log m$ *term in the bounds is amortized.)*

Proof. We use the measure tree along with a two-dimensional dynamic segment tree. The bound on the space complexity follows because the measure tree requires linear space and the multi-level segment tree requires $O(n \log n)$ space by Lemma 3.2.7. The query complexity is obviously constant. The insertion or deletion of a box takes $O(\sqrt{m} \log m)$ time for the measure tree and $O(\log^2 n)$ time for the segment tree. The cost of inserting or deleting a point is $O(\log m)$ for the measure tree if there is no reconstruction of a lower tree or the whole measure tree. Reconstruction of the measure tree costs $O(m \log m)$. We charge the cost of this construction to the $\Omega(m)$ point updates that must precede it, which gives us an amortized cost of $O(\log m)$ per update. The reconstruction of a lower tree costs $O(\sqrt{m})$. One can easily show that $\Omega(\sqrt{m})$ point insertions precede the construction, which gives us an amortized cost of $O(1)$.

Finally, we do a stabbing count query costing $O(\log^2 n)$ time when we insert a point. This completes the proof. □

3.2.4 Extension to Higher Dimensions

The measure tree naturally extends to higher dimensions, as a d-level tree, with each level partitioning the points along one of the coordinate axes. The tree at the top level partitions the set of points into at most $2m^{1/d}$ subsets, each of which is partitioned into at most $2m^{1/d}$ subsets by a second level tree. This partitioning continues through d levels The measure is maintained using the σ and $\bar{\mu}$ values, as in two dimensions.

The box and point insertions/deletions are performed the same way they are done in two dimensions. The extension of point updates are straightforward, and they take $O(\log m)$ time per operation (aside from the initialization of stabbing counts and the reconstruction operations which will be discussed shortly). For box insertions and deletions, the idea is again to do a range query to find a set $\mathcal{C}$ of subtrees whose leaves correspond to the points covered by the box. Then, we simply increment or decrement (depending on whether it is an insertion or deletion) σ value of the root of each subtree in $\mathcal{C}$, and perform the necessary updates to maintain the sum and push-up invariants. The following lemma summarizes the cost of a box update.

Lemma 3.2.9. *The cost of a box insertion or deletion in the d-dimensional measure tree is $O(m^{1-\frac{1}{d}} \log m)$.*

Proof. The cost of a box update is dominated by the range query to find the node set $\mathcal{C}$. All additional modifications happen on the paths traversed by the range query and thus does not increase complexity. To analyze the complexity of the range query, we examine it level by level. At the top level, we have the upper tree that subdivides the set of points into disjoint subsets along the first coordinate axis. Each subset is represented by a subtree at the second level. For at most two of these point subsets, the box covers the subset *partially* along the first coordinate axis. Consequently, the corresponding two subtrees (which are of size $O(m^{1-\frac{1}{d}})$) may need to be traversed completely. For each subset that is completely covered by the box along the first coordinate axis, the range query performs a subquery in the corresponding subtree. This query is equivalent to performing a range query in a $(d-1)$-dimensional measure tree with $O(m^{1-\frac{1}{d}})$ nodes. This process recursively traverses down the tree and is finalized at the dth level. It is easy to see that the range query in a single subtree in the dth level takes $O(\log m)$.

Let $F(m, d)$ be the cost of the range query on a d-dimensional measure tree with m points. Then, we can write the following recurrence:

$$F(m,d) = \begin{cases} O(\log m) & \text{if } d = 1 \\ O(m^{1-\frac{1}{d}}) + m^{\frac{1}{d}} F(O(m^{1-\frac{1}{d}}), d-1) & \text{if } d \geq 2 \end{cases}$$

This easily solves to $F(m, d) = O(m^{1-1/d} \log m)$. The lemma follows. □

All reconstruction procedures are natural extensions of their two-dimensional counterparts. The initial tree size is at most $\lceil m^{1/d} \rceil$ at all levels; a tree is split when its size becomes larger than $2m^{1/d}$. Moreover, we reconstruct the whole tree after each Cm_0 point insertions and deletions, where C can be any constant satisfying $C^{1/d} < \frac{1}{4}$, for instance $C = \frac{1}{5^d}$. The following two lemmas show that the tree reconstructions are frequent enough that the size of the upper tree is at most $2m^{1/d}$ at all times.

Lemma 3.2.10. *Consider a measure tree constructed for m_0 points. Then a newly constructed inner tree T requires at least $(1 - 3C^{1/d})m_0^{1/d}$ leaf splits (splits of child trees) before T itself splits.*

Proof. T has at most $(m^{1/d} + 1)$ leaves when it is constructed and at least $2m^{1/d}$ leaves when it is split. Moreover, at any time, m satisfies $(1 - C)m_0 \leq m \leq (1 + C)m_0$. Consequently, the following is a lowerbound on the number of leaf splits required to split T:

$$\begin{aligned} 2\left((1-C)m_0\right)^{1/d} - \left((1+C)m_0\right)^{1/d} - 1 &= 2(1-C)^{1/d}m_0^{1/d} - (1+C)^{1/d}m_0^{1/d} - 1 \\ &\geq 2(1-C^{1/d})m_0^{1/d} - (1+C^{1/d})m_0^{1/d} \\ &= (1-3C^{1/d})m_0^{1/d} \end{aligned}$$

The lemma follows. □

Lemma 3.2.11. *The size of the upper tree is at most $2m^{1/d}$.*

Proof. By Lemma 3.2.10, when the tree is constructed for m_0 points, it requires at least $(1 - 3C^{1/d})m_0^{1/d}$ point insertions to a dth level inner tree to split. Similarly, the split of a tree at the $(d-1)$th level requires the split of $(1 - 3C^{1/d})m_0^{1/d}$ inner trees at the dth level. This implies that $\left((1 - 3C^{1/d})m_0^{1/d}\right)^2$ points insertions are necessarry to split an inner tree at level $(d-1)$. Working the same logic all the way up to the top level, we see that it requires $\left((1 - 3C^{1/d})m_0^{1/d}\right)^d$ point insertions for the upper tree to exceed its size threshold of $2m^{1/d}$. By definition of C, this is more than Cm_0 and thus the tree is reconstructed before the upper tree can exceed its threshold. The lemma follows. □

Theorem 3.2.12. *We can maintain the discrete measure in d dimensions, for $d \geq 2$, using $O(n \log^{d-1} n + m)$ space, with constant time measure queries, $O(\log^d n + m^{1-\frac{1}{d}} \log m)$ time for insertion or deletion of a box, $O(\log^d n + \log m)$ time for insertion of a point, and $O(\log m)$ time for the deletion of a point. (The $\log m$ term in the bounds is amortized.)*

Proof. We use the measure tree along with a d-dimensional dynamic segment tree. The bound on the space complexity follows because the measure tree requires linear space and the multi-level segment tree requires $O(n \log^{d-1} n)$ space by Lemma 3.2.7. The query complexity is obviously constant. The insertion or deletion of a box takes $O(m^{1-\frac{1}{d}} \log m)$ time for the measure tree and $O(\log^d n)$ time for the segment tree. The cost of inserting or deleting a point is $O(\log m)$ for the measure tree if there is no reconstruction of a subtree or the whole measure tree. Reconstruction of the whole

measure tree costs $O(m \log m)$. We charge the cost of this construction to the $\Omega(m)$ point updates that must precede it, which gives us an amortized cost of $O(\log m)$ per update. The reconstruction of a subtree at the ith level (due to leaf split in its parent tree) costs $O(m^{1-\frac{i-1}{d}} \log m)$. By Lemma 3.2.10, one can easily show that $\Omega(m^{1-\frac{i-1}{d}})$ point insertions precede the reconstruction, which gives us an amortized cost of $O(\log m)$. Finally, we do a stabbing count query costing $O(\log^d n)$ time when we insert a point. This completes the proof. □

3.2.5 Further Improvements

The amortized bounds of our structure can be converted to worst case bounds, using a general technique called *global rebuilding* [62]. The idea, in brief, is to spread out the process of subtree reconstruction over time, operating on a shadow copy of the main data structure and then swapping in the result when the reconstruction is finished.

Finally, the term $m^{1-\frac{1}{d}} \log m$ in box update bounds can be improved to $m^{1-\frac{1}{d}}$ by using an optimized version of the measure tree. For instance, in two dimensions, the partitioning parameter can be tuned to achieve $O(\sqrt{m} + \log^2 n)$ time for inserting or deleting a box, by mimicking the construction of [51].

3.3 Extensions

3.3.1 Reporting Queries

In some applications, it is useful to report explicitly the points covered (or uncovered) by the union of boxes. Our structure can be used to answer such queries in output-sensitive time. In this section, we describe how this can be done.

We report the covered (uncovered) points in an output-sensitive manner simply by traversing the tree top-down, visiting each node whose subtree contains at least one point to be reported. These *active* nodes can be identified as follows: When reporting the covered points, a node v of the measure tree is *active* if it has an ancestor a with $\sigma(a) > 0$ or $\bar{\mu}(v) < leaves(v)$, where $leaves(v)$ is the number of leaves below v. (The quantity $leaves(v)$ is easily maintained.) When reporting the uncovered points, a node v is *active* if $\bar{\mu}(v) > 0$ for v and all its ancestors. In either case, we can decide whether a node is active in constant time during the tree traversal.

To analyze the total cost of the traversal, we first introduce the following lemma, which is an extension of a result on perfectly balanced trees [57] to 2-3 trees.

Lemma 3.3.1. *Let L be a set of k leaves in a 2-3 tree with M leaves. Then the total number of nodes v such that v has a descendant leaf in L is $O(k + k \log \frac{M}{k})$.*

Proof. Let T be a 2-3 tree with M leaves and let L be a set of k leaves in T. We call a leaf v *selected* if $v \in L$. Let $T(L)$ denote the subtree of T consisting of the selected

nodes and all their ancestors. We want to bound $|T(L)|$. Assume that L is chosen to maximize $|T(L)|$. We now make a series of deductions.

1. Call an internal node v in $T(L)$ an i-node if it has i children in $T(L)$. Then the highest 1-node is not higher than the lowest 2-node or 3-node.

   Proof. Suppose to the contrary that v is a 2-node or 3-node that is lower than a 1-node w, that is, $height(v) < height(w)$. Let v' be a selected descendant leaf of v. One of w's children, say w', has no selected leaves in its subtree. Let w'' be a descendant leaf of w'. Since T is a 2-3 tree, v' and w'' have the same depth. We can increase $|T(L)|$ by at least $(height(w) - height(v))$ by removing v' from L and inserting w'' into L. This contradicts the assumption that L was chosen to maximize $|T(L)|$. □

2. On the path from the root to a leaf in $T(L)$, all 1-nodes are below all 2-nodes and 3-nodes.

   Proof. This follows from Property 1. □

3. The number of 2-nodes and 3-nodes is at most $k - 1$.

   Proof. Each 2- or 3-node corresponds to one step in a hierarchical partition of the set L into singletons, so there can be at most $k - 1$ of them. □

4. Let T' be a 2-3 tree formed by pruning some nodes of T such that all nodes of $T(L)$ are in T' and every internal node in T' has two children except the 3-nodes of $T(L)$. Let M' denote the number of leaves in T'. Then, the size (number of leaves) of each maximal subtree of T' containing exactly one selected leaf is at most $\frac{2M'}{k}$.

   Proof. The maximal subtree containing a single selected leaf v is the subtree rooted at the highest 1-node ancestor of v (or v itself if it has no 1-node parent). By Property 1, the height difference between two such maximal subtrees is at most 1. Consequently, since all such maximal trees are complete binary trees, the ratio between the sizes of two maximal subtrees is at most 2. Moreover, there is at least one maximal subtree whose size is at most M'/k. It follows that every maximal subtree has size at most $\frac{2M'}{k}$. □

5. The total number of 1-nodes is at most $k \log_2 \frac{2M}{k}$.

   Proof. Consider a selected leaf v. Let T_v be the maximal subtree of T' containing v as its only selected leaf. Since T_v is binary and by Property 4, the height of T_v is at most $\log_2 \frac{2M'}{k}$. This number also bounds the number of 1-node ancestors of v. By summing over all selected leaves, we get $k \log_2 \frac{2M'}{k} \leq k \log_2 \frac{2M}{k}$ as an upper bound on the total number of 1-nodes. □

By Properties 3 and 5, the number of internal nodes in $T(L)$ is at most $(k-1) + k\log_2 \frac{2M}{k}$. It follows that $|T(L)|$ is $O(k + k\log \frac{M}{k})$. □

We continue with the following mathematical property.

Lemma 3.3.2. *Let m, k, m_1, m_2, k_1 and k_2 be positive real numbers such that $m_1 + m_2 = m$ and $k_1 + k_2 = k$. Then, $k\log \frac{m}{k} \geq k_1 \log \frac{m_1}{k_1} + k_2 \log \frac{m_2}{k_2}$.*

Proof. By simple calculus, the function $f(x) = x\log \frac{m_1}{x} + (k-x)\log \frac{m_2}{x}$ attains its maximum value when $x = \left(\frac{m_1}{m_1+m_2}\right)k$. This implies that the expression $k_1 \log \frac{m_1}{k_1} + k_2 \log \frac{m_2}{k_2}$ has its maximum value when $k_1 = \left(\frac{m_1}{m_1+m_2}\right)k$, in which case $\frac{m_1}{k_1} = \frac{m_2}{k_2} = \frac{m}{k}$. Then, we can write

$$k_1 \log \frac{m_1}{k_1} + k_2 \log \frac{m_2}{k_2} \;\leq\; k_1 \log \frac{m}{k} + k_2 \log \frac{m}{k} \;=\; k\log \frac{m}{k}$$

The lemma follows. □

Corollary 3.3.3. *Let m and k be positive real numbers. Also, let $m_1, \ldots, m_t$ and $k_1, \ldots, k_t$ be positive real numbers such that $\sum m_i = m$ and $\sum k_i = k$. Then, $k\log \frac{m}{k} \geq \sum k_i \log \frac{m_i}{k_i}$.*

We now prove the following lemma.

Lemma 3.3.4. *Let L be a set of k leaves in a d-dimensional measure tree of m points. Then the total number of nodes v such that v has a descendant leaf in L is $O(k + k\log \frac{m}{k})$.*

Proof. Suppose that T is a measure tree of m points. Let L be a set of k selected leaves of T. Let $T(L)$ be the subtree consisting of L and all ancestors of the leaves in L. Notice that we want to bound $|T(L)|$.

In order to simplify the math in the remaining of the proof, we assume that $k < m/\mathsf{e}$, where e is the natural logarithm base. Notice that if $k \geq m/\mathsf{e}$, then both $|T|$ and $|T(L)|$ are clearly $O(k)$ and thus $|T(L)|$ satisfies the bound.

To show that $|T(L)| = O(k + k \log \frac{m}{k})$, we argue that the size of $T(L)$ restricted to each of the d levels of the measure tree is $O(k + k \log \frac{m}{k})$. (Recall that we treat d as a constant in our analyses.) In particular, let $\{T_1, \ldots, T_t\}$ be the set of inner trees in the ith level such that each T_j has some overlap with $T(L)$, i.e., $|T_j \cap T(L)| \geq 1$. The size of $T(L)$ constrained to the ith level is simply $\sum |T_j \cap T(L)|$.

Recall that each inner tree is a 2-3 tree. Let m_j be the number of leaves of T_j and k_j be the number of leaves in $T_j \cap T(L)$. By Lemma 3.3.1, $|T_j \cap T(L)| = O(k_j + k_j \log \frac{m_j}{k_j})$. Clearly, $\sum k_j \leq k$ and $\sum m_j \leq m$. The function $f(x, y) = x + x \log \frac{y}{x}$ is increasing with respect to y, and also with respect x if $x < y/\mathsf{e}$. Then, based on the assumption that $k < m/\mathsf{e}$ and Corollary 3.3.3, the following is a bound for the size of

$T(L)$ constrained to the ith level:

$$\begin{aligned}\sum |T_j \cap T(L)| &= \sum O(k_j + k_j \log \frac{m_j}{k_j}) \\ &\leq O\left(\sum k_j + \sum k_j \times \log \frac{\sum m_j}{\sum k_j}\right) \\ &\leq O(k + k \log \frac{m}{k})\end{aligned}$$

The lemma follows. □

It follows that the reporting k points takes $O(k + k \log \frac{m}{k})$ traversal steps, proving the following theorem.

Theorem 3.3.5. *A reporting query can be answered in $O(k + k \log \frac{m}{k})$ time, where k is the size of the output.*

3.3.2 Uncertain Discrete Measure

The recent proliferation of data mining applications has created an urgent need to deal with *data uncertainty*, which may arise because the mining algorithms output probability distributions over an output space, or because attributes whose values are not explicitly known are modeled with a discrete set of probabilistic values. This motivates a natural *probabilistic* extension of our discrete measure problem, in which both the underlying set of points $\mathcal{P}$ and the set of boxes $\mathcal{B}$ are associated with independent probabilities. Specifically, each point p in $\mathcal{P}$ occurs with probability π_p and each box B in $\mathcal{B}$

occurs with probability π_B. The probabilities are independent, but otherwise can take any real values. A natural problem in this setting is to compute the *expected size* of the discrete measure—that is, how large is $meas(\mathcal{B}, \mathcal{P})$ on average for a random sample of boxes and points drawn from the given probability distribution?

Our measure maintenance algorithm is adapted to the uncertain problem as follows. We represent the measure as the difference between the expected number of points that are present and the expected number of those points that remain uncovered by the boxes that are present. This difference is the desired expected measure. The expected number of points present can be written simply as $\sum_{p \in \mathcal{P}} \pi_p$. In order to maintain the expected number of these points that remain uncovered, we proceed as follows. Let $prob(p)$ denote the probability that p is *uncovered* by the set of boxes present in B. We can write $prob(p)$ as the product of the complements of the existence probabilities of the boxes that cover p, namely, $prob(p) = \prod_{B \in \mathcal{B} \wedge p \in B} (1 - \pi_B)$. The expected number of uncovered points can then be written as $\sum_{p \in \mathcal{P}} prob(p)\pi_p$. We maintain this quantity with a measure tree in which the auxiliary information (σ and $\bar{\mu}$) is modified in the following way.

Each σ value is a real number between 0 and 1. We maintain the *product invariant* that the product of $\sigma(a)$ over all ancestors a of a leaf v equals $prob(p_v)$. (Cf. the sum invariant of Section 3.2.) The quantity $\bar{\mu}(v)$ is used to represent the expected number of uncovered points, considering only the information stored in the subtree of v. In

particular,

$$\bar{\mu}(v) = \begin{cases} \sigma(v)\pi_{p_v} & \text{if } v \text{ is a leaf} \\ \sigma(v)\sum_{w \in child(v)} \bar{\mu}(w) & \text{if } v \text{ is an internal node} \end{cases}$$

We now show that $\bar{\mu}(root)$ is the expected number of uncovered points in a random sample of the instance.

Lemma 3.3.6. *For a node v, let P_v be the set of points which are stored in the subtree v. Also, let $\sigma^*(v)$ be the product of the σ values of the ancestors of v. Then, $\sigma^*(v)\bar{\mu}(v)$ is the expected number of uncovered points in P_v, i.e., $\sigma^*(v)\bar{\mu}(v) = \sum_{p \in P_v} prob(p)\pi_p$.*

Proof. By induction. If v is a leaf, then $\sigma^*(v)\bar{\mu}(v) = \sigma^*(v)\sigma(v)\pi_{p_v} = prob(p_v)\pi_{p_v}$ by the product invariant. If v is an internal node, we can write

$$\begin{aligned} \sigma^*(v)\bar{\mu}(v) &= \sigma^*(v)\sigma(v) \sum_{w \in child(v)} \bar{\mu}(w) \\ &= \sum_{w \in child(v)} \sigma^*(v)\sigma(v)\bar{\mu}(w) \\ &= \sum_{w \in child(v)} \sigma^*(w)\bar{\mu}(w) \\ &= \sum_{w \in child(v)} \sum_{p \in P_w} prob(p)\pi_p \\ &= \sum_{p \in P_v} prob(p)\pi_p \end{aligned}$$

□

Corollary 3.3.7. *$\bar{\mu}$(root) is the expected number of uncovered points.*

The insertion of a box B is performed using a range search similar to that in the non-stochastic setting. The σ values of the roots of the reported subtrees are multiplied by $(1 - \pi_B)$ to preserve the product invariant. The deletion of a box is done similarly, except that the σ values are divided by $(1 - \pi_B)$.

The *push-up invariant* in the uncertainty setting requires that at least one child of each internal node has a σ value of 1. To perform a push-up operation on a node v, we multiply $\sigma(v)$ by Δ and divide $\sigma(w)$ by Δ for all children w of v, where $\Delta = \max_{w \in child(v)} \sigma(w)$. Push-downs are performed analogously.

We note that when a point p is inserted, we need to know $prob(p)$. (This is similar to $stab(p)$ in the non-stochastic setting.) The multi-level segment tree can easily be modified to answer this kind of query.

One technical issue is that a division by zero may occur if $\mathcal{B}$ contains boxes with an existence probability of 1. We work around this problem by symbolic perturbation, by treating each box with probability 1 as if it had a probability of $(1-\epsilon)$, for an arbitrarily small $\epsilon > 0$. We then represent each σ value symbolically with an expression $r\epsilon^t$ where r is a real number and t is a nonnegative integer. When we evaluate $\bar{\mu}(v)$ at a node v, if $\sigma(v)$ contains any positive power of ϵ, then $\bar{\mu}(v) = 0$. With these adaptations, we can deduce the result given in Theorem 3.3.8.

Theorem 3.3.8. *The d-dimensional discrete measure problem on uncertain boxes and points can be solved with a data structure that requires $O(n \log^{d-1} n + m)$ space, $O(1)$ query time, $O(\log^d n + m^{1-\frac{1}{d}})$ time for insertion or deletion of a box, $O(\log^d n + \log m)$ time for a point insertion and $O(\log m)$ time for a point deletion.*

3.4 Conclusion

We introduced a discrete measure problem, and presented a data structure that supports dynamic updates to both the set of points and the set of boxes. The queries for the current measure take constant time, the updates to the set of points take polylogarithmic time, while updates to the set of boxes take time polylogarithmic in the number of boxes and sub-linear in the number of points. The data structure permits output-sensitive enumeration of the points covered by the union of the boxes, and also lends itself to a stochastic setting in which points and boxes are present with independent, but arbitrary, probabilities.

Our work leads to a number of research problems. First, can the update bounds be improved? Second, is there a trade-off between the update time for boxes and the update time for points? In particular, can one achieve polylogarithmic complexity in both n and m?

Chapter 4

Convex Hulls under Uncertainty: Membership Probability*

4.1 Introduction

In this chapter, we present our first set of results on convex hulls under uncertainty. The focus of this chapter is computing the membership probability, which is the probability that a given point is contained in the convex hull of an uncertain set of points. Our work on convex hulls under uncertainty also spans the next chapter, where we consider the problem of computing the most likely convex hull, which is the mode of the convex hull random variable. We start this chapter with a brief information on convex hulls.

*This chapter is based on a joint work with Pankaj K. Agarwal, Sariel Har-Peled, Subhash Suri and Wuzhou Zhang. Parts of this chapter are under preparation for submission.

The convex hull is a fundamental structure in mathematics and computational geometry. Given a set of points $\mathcal{P}$ in d-space, the *convex hull* of $\mathcal{P}$ is the minimal convex polytope that contains all points in $\mathcal{P}$. Convex hulls have applications in a variety of areas including but not limited to computer graphics, image processing, pattern recognition and statistics. Owing to their importance in practice, the algorithms for computing convex hulls are well-studied. In two and three dimensions, there are algorithms to compute the convex hull of n points in $O(n \log n)$ time [46, 65] or even in $O(n \log h)$ output-sensitive time where h is the complexity of the hull [21,26,55]. In d dimensions, for $d > 3$, the best algorithm takes $O(n^{\lfloor d/2 \rfloor})$ time in the worst-case [25], but there also exist faster output-sensitive algorithms [61, 67]. In all dimensions, the algorithms are optimal in the worst-case for algebraic decision tree and algebraic computation tree models of computation [11, 25].

In many applications, such as sensor databases, location-based services or computer vision, the location and sometimes even the existence of the data is uncertain, but statistical information can be used as a probability distribution guide for data. This raises the natural computational question: what is a robust and useful convex hull representation for such an uncertain input, and how well can we compute it? We explore this problem under two simple models where both the location and the existence (presence) of each point are described probabilistically, and study basic questions such as what is the

probability that a query point lies inside the convex hull, or what does the probability distribution of the convex hull over the space look like.

Uncertainty Models

We focus on two models of uncertainty: unipoint and multipoint. In the *unipoint model*, each input point has a fixed location but it only exists probabilistically. Specifically, the input $\mathcal{P}$ is a set of pairs $\{(s_1, \pi_1), \ldots, (s_n, \pi_n)\}$ where each s_i is a point in d-space and each π_i is a real number in the range $(0, 1]$ denoting the probability of s_i's existence. The existence probabilities of different points are independent.

In the *multipoint model*, each point probabilistically exists at one of multiple possible sites. Specifically, $\mathcal{P}$ is a set of pairs $\{(S_1, \Pi_1), \ldots, (S_m, \Pi_m)\}$ where each S_i is a set of points and each Π_i is a set of reals in the range $(0, 1]$, with $|S_i| = |\Pi_i|$. The set $S_i = \{s_i^1, \ldots, s_i^{|S_i|}\}$ describes the possible sites for the ith point of $\mathcal{P}$ and the set $\Pi_i = \{\pi_i^1, \ldots, \pi_i^{|\Pi_i|}\}$ describes the associated probability distribution. In an experiment, the ith point of $\mathcal{P}$ is located at s_i^j with probability π_i^j. The probabilities π_i^j correspond to disjoint events and therefore sum to at most 1. By allowing the sum to be less than one, this model also accounts for the possibility that the point does not exist (i.e. the *null* location)—thus, the multipoint model strictly generalizes the unipoint model. In the multipoint model, the number of points in $\mathcal{P}$ is referred by m and we use n to refer to the total number of possible sites, i.e., $n = \sum_i |S_i|$.

The Results

The main results of this chapter can be summarized as follows.

1. We show that the membership probability, namely, the probability of a query point being inside the convex hull, can be computed in $O(n \log n)$ time for $d = 2$, both in the unipoint and the multipoint models.

2. We show that the membership probability can be computed in $O(n^d)$ time for $d \geq 3$, both in the unipoint and the multipoint models, *given that all points including the query point are in general position.*

3. We describe a structure called the *probability map*, denoted $\mathbb{M}(\mathcal{P})$, which decomposes $\mathbb{R}^d$ into cells so that all points in a single cell have the same membership probability. We show that $\mathbb{M}(\mathcal{P})$ has size $O(n^{d^2})$. The map can be computed in optimal $O(n^4)$ time for $d = 2$.

Previous Work

In the multipoint uncertainty model, numerous results have been obtained on nearest neighbor queries, top-k queries, range queries, clustering, ranking, etc [1, 3, 4, 7, 13, 28, 29, 33, 47, 49], in both the database and the computational geometry community; see [6, 34] for surveys on uncertain data. Some problems become indeed intractable in the presence of uncertainty, and one has to turn to approximation. Suri et al. [50]

showed that, even in the unipoint model, it is NP-hard to compute the probability that the closest pair distance is less than a given value l, even in $\mathbb{R}^2$ and L_2 norm, and they gave a linear-space data structure with $O(\log n)$ query time to compute the expected distance of a given query point to its $(1+\varepsilon)$-approximate nearest neighbor when the dimension d is a constant.

The convex hull problem over uncertain data has received some attention very recently. Zhao et al. investigated the problem of computing the probability of each uncertain point lying on the convex hull [82], where they aimed to return the set of (uncertain) input points whose probability of being on the convex hull is at least some threshold.

Some scientists have also considered geometric problems on "imprecise" objects: each object, such as a point, can be anywhere inside a simple geometric region [18, 19, 58, 59, 60, 75]. Convex hull was examined in the imprecise point model [58], and problems like (approximate) largest convex hull and smallest convex hull were investigated. However, we emphasize that this line of research looks at the worst-case behavior, and not the stochastic behavior, which is the main focus of our work. Another line of research has focused on uncertainty caused by the finite machine precision [53, 66, 77]. The goal there is to achieve robustness under bounded precision, and not to compute structures that are most representative under a probability distribution.

Chapter Organization

In Section 4.2, we describe our algorithm for the membership probability in two dimensions. This is followed by our algorithm for higher dimensions in Section 4.3. We investigate probability maps in Section 4.4. Finally, we sum up with a conclusion in 4.5

4.2 Membership Probability in the Plane

In this section, we describe how to compute the convex hull membership probability of a given point with respect to a given uncertain point set in the plane. For simplicity, we assume that the input is non-degenerate, meaning that all possible point sites, including the query point q, are in general position: no two sites have the same coordinate along any dimension and no three sites are collinear. We defer the discussion of how to handle such degeneracies to Section 4.2.3. We begin our discussion with the unipoint case.

4.2.1 The Unipoint Model

Let $\mathcal{P} = \{(s_1, \pi_1), \ldots, (s_n, \pi_n)\}$ be a set of n uncertain points in the plane under the unipoint model. We let S denote the set of all sites of $\mathcal{P}$, namely, $\{s_1, \ldots, s_n\}$. A

subset $A \subseteq S$ is the outcome of a probabilistic experiment with probability

$$\pi(A) = \prod_{i \,|\, s_i \in A} \pi_i \;\times \prod_{i \,|\, s_i \notin A} \overline{\pi_i}$$

where $\overline{\pi_i}$ is the complementary probability $(1 - \pi_i)$. Given a query point q, we want to compute its membership probability, namely the probability that q lies in the convex hull of A. For simplicity, we denote this probability by $\mu(q)$. Let $\mathrm{conv}(A)$ denote the convex hull of A. By definition, $\mu(q)$ can be written as

$$\mu(q) = \sum_{\substack{A \subseteq S \\ q \,\in\, \mathrm{conv}(A)}} \pi(A),$$

which unfortunately involves an exponential number of terms (possible subsets A). Our polynomial-time scheme for computing $\mu(q)$ builds on the following simple observation: for an outcome A, q is *outside* $\mathrm{conv}(A)$ if and only if q is a vertex of the convex hull $\mathrm{conv}(A \cup \{q\})$. For ease of reference, let C denote $\mathrm{conv}(A \cup \{q\})$ and V denote the set of vertices of C. Then, the probability we want is $\mu(q) \;=\; 1 - \Pr\big[\, q \in V \,\big]$.

If $A = \emptyset$, then clearly $C = \{q\}$ and $q \in V$. Otherwise, $|V| \geq 2$ and $q \in V$ implies that q is an endpoint of exactly two edges on the boundary of C.[14] In this case, we define the first edge following q in the counter-clockwise order of C as the *witness edge* of the outcome A. (See Figure 4.1a.) It is easy to see that $q \in V$ if and only if

[14] If A consists of a single site s_i, then C is the line segment qs_i. In this case, we consider the boundary of C to be a cycle formed by two edges: one going from q to s_i, and one going from s_i back to q.

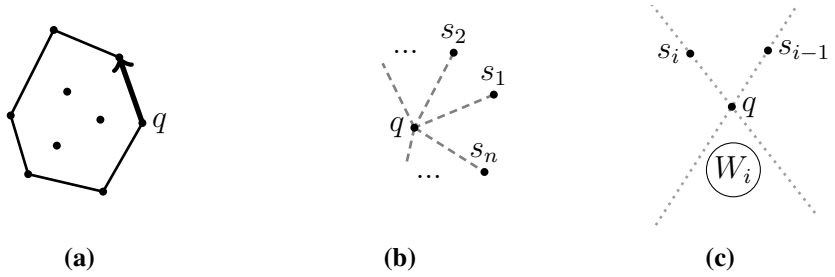


Figure 4.1: (a) A witness edge. (b) Sites in radial order around q. (c) The set W_i.

$A = \emptyset$ or (exclusively) A has a witness edge. Thus,

$$\Pr\big[\, q \in V \,\big] = \Pr\big[\, A = \emptyset \,\big] + \sum_{1 \le i \le n} \Pr\big[\, qs_i \text{ is the witness edge of } A \,\big]$$

The first term is easily computed in $O(n)$ time. To compute the ith term in the summation, we observe that qs_i is the witness edge of A if and only if $s_i \in A$ and A contains no sites to the right of the line $\overleftrightarrow{qs_i}$, where the right direction is with respect to the vector $\overrightarrow{qs_i}$. The corresponding probability is easily written as:

$$\pi_i \cdot \prod_{j \,|\, s_j \in G_i} \overline{\pi_j}$$

where G_i is the set of sites to the right of the line $\overleftrightarrow{qs_i}$. This expression can be easily computed in $O(n)$ time. It follows that one can compute $\Pr\big[\, q \in V \,\big]$, and therefore $\mu(q)$, in $O(n^2)$ time.

The computation time can be improved to $O(n \log n)$ by computing the witness edge probabilities in radial order around q. We sort all sites in counter-clockwise order around q. Without loss of generality, assume that the circular sequence $s_1, \ldots, s_n$ is

the resulting order. (See Figure 4.1b.) We first compute the probability that qs_1 is the witness edge in $O(n)$ time. Then, for increasing values of i from 2 to n, we compute the probability that qs_i is the witness edge by updating the probability for qs_{i-1}, in $O(1)$ amortized time. In particular, let W_i denote the set of sites in the open wedge bounded by the lines $\overleftrightarrow{qs_{i-1}}$ and $\overleftrightarrow{qs_i}$. (See Figure 4.1c.) Notice that $G_i = G_{i-1} \cup \{s_{i-1}\} \setminus W_i$. It follows that the probability for qs_i can be computed by multiplying the probability for qs_{i-1} with $\frac{\pi_i}{\pi_{i-1}} \times \frac{\overline{\pi_{i-1}}}{\prod_{s_j \in W_i} \overline{\pi_j}}$. The cost of a single update is $O(1)$ amortized because total number multiplications in all the updates is at most $4n$. (Each site affects at most 4 updates.) Finally, notice that we can easily keep track of the set W_i during our radial sweep, as changes to this set follow the same radial order.

Theorem 4.2.1. *Given a set of n uncertain points in the unipoint model and a point q in the plane, one can compute the membership probability of q in $O(n \log n)$ time and linear space.*

4.2.2 The Multipoint Model

Our algorithm extends to the multipoint model easily by modifying how we compute the probability for an edge. The key change is in the expression used to compute the probability that no sites exists in A from a given set G. In the unipoint model, we use an expression of the form $\prod_{s_i \in G} \overline{\pi_i}$ to write the corresponding probability. All such expressions are replaced by $\prod_{1 \leq i \leq m} \left(1 - \sum_{j \mid s_i^j \in G} \pi_i^j\right)$ in the multipoint model, where

$\left(1 - \sum_{j \mid s_i^j \in G} \pi_i^j\right)$ is the probability that ith point does not exist on a site from G. Before we describe the changes in more detail, we first give some preliminary information on the membership probability in the multipoint model.

Recall that in the multipoint model, $\mathcal{P}$ is a set of pairs $\{(S_1, \Pi_1), \ldots, (S_m, \Pi_m)\}$ where each set $S_i = \{s_i^1, \ldots, s_i^{|S_i|}\}$ describes the possible sites for the ith point of $\mathcal{P}$ and each set $\Pi_i = \{\pi_i^1, \ldots, \pi_i^{|\Pi_i|}\}$ describes the associated probability distribution. In an experiment, the ith point of $\mathcal{P}$ is located at s_i^j with probability π_i^j. We use S to denote the set of all sites, i.e., $S = \{s_i^j\}$ and set $n = S$.

For a subset $A \subseteq S$, we denote the probability that A is the outcome of a probabilistic experiment by $\pi(A)$. Similarly to the unipoint model, the definition of $\pi(A)$ involves a product of existence probabilities for all sites in A. The sites that are not in A, however, contribute to $\pi(A)$ in a different way. Specifically, let s_i^j be a site that is not in A. If A contains another $s_i^{j'}$ site from the ith point, then the non-existence probability of s_i^j is irrelevant to $\pi(A)$, because existence of $s_i^{j'}$ already implies non-existence of s_i^j. If there is no such site $s_i^{j'}$, then no site from the tuple of the ith point is in A. In that case, we just consider the probability that ith point does not exist at all, which is $1 - \sum_{1 \leq j \leq |S_i|} \pi_i^j$. Finally, notice that if A contains two sites from the same uncertain point, then it cannot be the outcome of an experiment. This implies the following definition for $\pi(A)$:

$$\pi(A) = \begin{cases} 0 & \text{if there are two distinct sites } s_i^j \text{ and } s_i^{j'} \text{ in } A \\ \displaystyle\prod_{s_i^j \in A} \pi_i^j \times \prod_{i \mid \nexists j. s_i^j \in A} \left(1 - \sum_{1 \le j \le |S_i|} \pi_i^j\right) & \text{otherwise} \end{cases}$$

The definition of $\mu(q)$ remains the same but is based on the new definition of $\pi(A)$.

We now describe how $\mu(q)$ is computed in the multipoint model. For an outcome $A \subseteq S$, let us define C and V as before, i.e., $C = \mathrm{conv}(A \cup \{q\})$ and V is the set of vertices of C. As in the unipoint model, q is in the convex hull of A if and only if $q \in V$. We follow a similar strategy and decompose $\Pr[\, q \in V \,]$ as follows:

$$\Pr[\, q \in V \,] = \Pr[\, A = \emptyset \,] + \sum_{\substack{1 \le i \le m \\ 1 \le j \le |S_i|}} \Pr[\, qs_i^j \text{ is the witness edge of } A \,]$$

The first term is trivial to compute in $O(n)$ time. We compute the probability that qs_i^j forms a witness edge of A as follows. Let $G_{i,j}$ be the set of sites to the right of the line $\overleftrightarrow{qs_i^j}$ where the right direction is with respect to the vector $\overrightarrow{qs_i^j}$. As in the unipoint model, the segment qs_i^j is the witness edge of A if and only if $s_i^j \in A$ and $A \cap G_{i,j} = \emptyset$. We

can write the corresponding probability as follows:

$$\begin{aligned}
\Pr\big[\, s_i^j \in A \;\wedge\; A \cap G_{i,j} = \emptyset \,\big] &= \Pr\big[\, s_i^j \in A \,\big] \times \Pr\big[\, A \cap G_{i,j} = \emptyset \;\mid\; s_i^j \in A \,\big] \\
&= \Pr\big[\, s_i^j \in A \,\big] \times \prod_{1 \le k \le m} \Pr\big[\, A \cap G_{i,j} \cap S_k = \emptyset \;\mid\; s_i^j \in A \,\big] \\
&= \Pr\big[\, s_i^j \in A \,\big] \times \prod_{\substack{1 \le k \le m \\ k \ne i}} \Pr\big[\, A \cap S_k \cap G_{i,j} = \emptyset \,\big] \\
&= \pi_i^j \times \prod_{\substack{1 \le k \le m \\ k \ne i}} \left(1 - \sum_{l \,\mid\, s_k^l \in G_{i,j}} \pi_k^l \right)
\end{aligned}$$

This expression can be easily computed in $O(n)$ time. It follows that one can compute $\overline{\mu(q)}$, thus $\mu(q)$, in $O(n^2)$ time.

As before, the computation time can be improved to $O(n \log n)$ by computing the witness edge probabilities in radial order around q. Let the circular sequence $s'_1, s'_2, \ldots, s'_n$ be the counter-clockwise order of all sites around q, where each s'_u is a distinct site s_a^b. We first compute the probability that qs'_1 is the witness edge in $O(n)$ time and also remember the values of the intermediate factors used in the computation. (The factors inside the $\prod_{1 \le k \le m}$ expression.) Then, for increasing values of u from 2 to n, we compute the probability that qs'_u is the witness edge by updating the probability for qs'_{u-1}. As a first step to this update, we update the values of the intermediate factors. To be more specific, let W_u denote the set of sites in the open wedge bounded by the lines $\overleftrightarrow{qs'_u}$ and $\overleftrightarrow{qs'_{u-1}}$. Also, for simplicity, assume that $s'_u = s_a^b$ and $s'_{u-1} = s_c^d$. Notice

that $G_{a,b} = G_{c,d} \cup \{s_c^d\} \setminus W_u$. Then, for each site s_e^f in W_u, the eth factor increases by π_e^f. Also, the cth factor decreases by π_c^d. Finally, we temporarily set the value of the ath factor to 1 (to cover the case $k \neq i$ in the expression). Then, we can compute the witness edge probability for qs'_u by multiplying the probability of qs'_{u-1} with π_a^b/π_c^d and the multiplicative change in each intermediate factor. The cost of a single update is $O(1)$ amortized, as each site can contribute to at most 4 updates as in the unipoint case.

Theorem 4.2.2. *Given a set of uncertain points in the multipoint model (with n sites in total) and a point q in the plane, one can compute the membership probability of q in $O(n \log n)$ time and linear space.*

4.2.3 Dealing with Degeneracies

In this section, we briefly explain how our algorithm for the planar case can be adapted to handle degeneracies. A degeneracy can happen in one of two ways: (i) Two sites may occur at the same location, and (ii) three sites may be collinear. There are two main issues to be handled in the presence of degeneracies:

1. **A site may coincide with the query point q:** If this is the case, then the existence of such a site in A implies that q is in the convex hull $\mathrm{conv}(A)$. We compute the probability that q coincides with a site in A separately. The remaining portion of $\mu(q)$ is computed as before, however is conditioned on the non-existence of

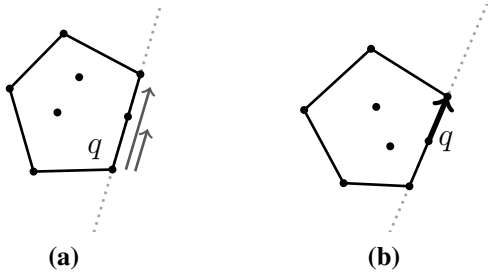


Figure 4.2: (a) An outcome with two possible witness edges, each shown by a gray arrow. (b) An outcome where there is a witness edge but q is not a vertex of $\mathrm{conv}(A \cup \{q\})$.

all sites that coincide with q. To be precise, we again compute the probability $1 - \Pr\big[\, q \in V \,\big]$, but this time on a reduced set of sites which does not involve the sites coinciding q. (In the multipoint model, this also requires adjusting the probabilities of the sites which belong to the same uncertain point with another site coinciding q.) Once this probability is computed, we further multiply it with the probability that no site coinciding q exists in A.

2. q **might be collinear with two or more other sites:** In such a case, it is possible for an outcome to have multiple witness edges. (See Figure 4.2a.) We can overcome this issue by selecting the witness edge that is the longest. If there are multiple longest edges because their corresponding sites coincide, we simply choose the one that has the lowest site index (the i in s_i). It may also be the case that an outcome has a witness edge but $q \notin V$. (See Figure 4.2b.) Both cases imply that a witness edge is valid only if there is no site in the outcome that is

collinear with the witness edge and but not contained in the edge. Based on the new definition of a witness edge, we tweak the way we compute the witness edge probability as follows. To compute the probability for an edge qs_i (in the unipoint model), we include the following additional sets of sites in G_i:

- All sites collinear with q and s_i but not contained in the segment qs_i.
- All sites s_j coinciding s_i such that $j < i$.

A similar approach also applies to the multipoint model.

4.3 Membership Probability in d Dimensions

We now describe our algorithm for computing the membership probability for dimensions higher than two. This algorithm works correctly only for non-degenerate input. To be precise, we require that all possible point sites, including the query point q, are in general position: no two sites have the same coordinate along any dimension and no $k+2$ sites lie on a k-dimensional hyperplane in any projection to a subset of the coordinates. We note that computing the membership probability for degenerate point sets in high dimensions is still an open problem. Our work can be considered a first step into understanding the complexity of the membership probability in high dimensions.

For simplicity, we describe our algorithm in the unipoint model and then briefly explain how it extends to the multipoint model.

4.3.1 The Unipoint Model

The main difficulty in extending the two-dimensional solution idea to higher dimensions is an appropriate generalization of *witness edges*, which allow us to implicitly sum over exponentially many outcomes without *double counting*. The following discussion explains the main idea we use for this generalization.

In keeping with the planar case, let V be the vertices of C and let $\lambda(A \cup \{q\})$ denote the point with lowest dth coordinate in $A \cup \{q\}$. Clearly, if q is $\lambda(A \cup \{q\})$ then $q \in V$. We decompose the probability that $q \in V$ based on which point among $A \cup \{q\}$ is $\lambda(A \cup \{q\})$. In particular, we write

$$\Pr\big[\, q \in V \,\big] = \Pr\big[\, q \equiv \lambda(A \cup \{q\}) \,\big] + \sum_{1 \le i \le n} \Pr\big[\, s_i \equiv \lambda(A \cup \{q\}) \ \wedge \ q \in V \,\big].$$

It is trivial to compute the first term. We show below how to compute each term of the summation in $O(n^{d-1})$ time, which gives the desired bound of $O(n^d)$.

Consider an outcome A with $s_i \in A$. We project the d-space onto the first $(d-1)$ dimensions, and let A', s_i', q', respectively, denote the projections of A, s_i and q. Let C' denote $\mathrm{conv}(A' \cup \{q'\})$, let V' be the vertices of C'. Observe that V' is not necessarily the projection of V because some vertices of V may fall inside C' when projected.

Let $\overrightarrow{r}(s_i', q')$ denote the ray obtained by removing the segment $s_i'q'$ from the ray $\overrightarrow{s_i'q'}$. (See Figure 4.3a for a three-dimensional example.) We say that a facet[15] f of C

[15] A facet of a d-dimensional polytope is a $(d-1)$-dimensional face on its boundary.

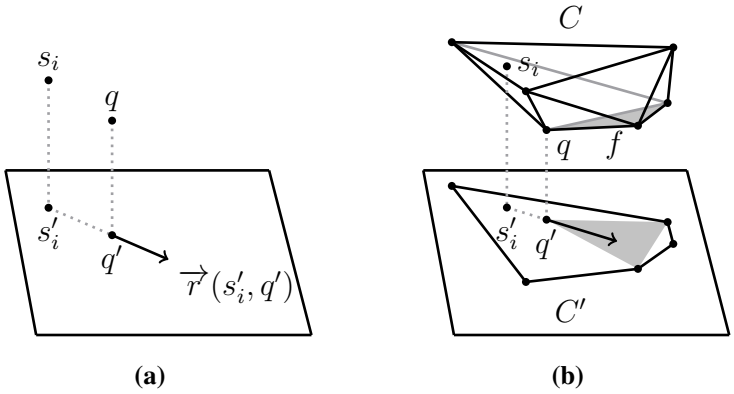


Figure 4.3: (a) The ray $\overrightarrow{r}(s'_i, q')$. (b) A s_i-escaping facet f for q on C.

is a *s_i-escaping facet* for q, if q is a vertex of C adjacent to f and the projection of f into the first $(d-1)$ dimensions intersects $\overrightarrow{r}(s'_i, q')$. (See Figure 4.3b for a three-dimensional example.) The following lemma is key to our algorithm.

Lemma 4.3.1. *If $s_i \in A$, (i) q has at most one s_i-escaping facet on C; (ii) $q \in V$ if and only if q has an s_i-escaping facet on C or (exclusively) $q' \in V'$.*

Proof. Proof of part (i) : Suppose that $s_i \in A$ and $q \in V$. Assume to the contrary that two s_i-covering facets f_1 and f_2 exists. Recall that a convex hull is defined by an intersection of halfspaces. Let H_1 be the defining halfspace that is bounded by the hyperplane supporting f_1 and contains C. Define H_2 similarly for f_2.

H_1 and H_2 grow towards opposite directions along the dth axis, otherwise C would be unbounded. To see this, consider a line ℓ parallel to the dth axis and projecting to the point $q' + \epsilon\overrightarrow{v}$ where ϵ is an infinitesimal and $\overrightarrow{v}$ is the direction vector of $\overrightarrow{r}(s'_i, q')$. Clearly, ℓ intersects both f_1 and f_2 and no other facet of C. Unless, H_1 and H_2 have

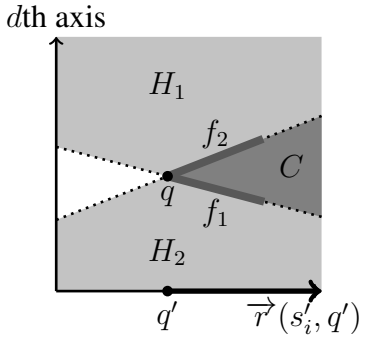


Figure 4.4: The cross-section of the space on the plane defined by the dth coordinate axis and the line supporting $\overrightarrow{r}(s_i', q')$.

opposite directions, the intersection of ℓ with H_1 and H_2 would be infinitely long and thus C would be unbounded.

Since the projections of f_1 and f_2 extend towards $\overrightarrow{r}(s_i', q')$ (and due to the general position assumption), H_1 and H_2 have a positive intersection length on all lines parallel to dth axis along $\overrightarrow{r}(s_i', q')$. The hyperplanes bounding H_1 and H_2 intersect at q. Due to the linearity of hyperplanes, the intersection of H_1 and H_2 is empty along the reverse ray $\overrightarrow{q's_i'}$. (H_1 and H_2 switch sides at q. See Figure 4.4 for an illustration.) It follows that s_i is not in C (and thus not in A), yielding a contradiction.

Proof of part (ii) : We first argue that both conditions imply $q \in V$. If q has an s_i-escaping facet on C then it is a vertex of C, thus $q \in V$. Also, if $q' \in V'$, then it has to be a vertex of C and thus $q \in V$.

We now prove the converse. In the course of the proof, we also show that the conditions are mutually exclusive. Suppose that $s_i \in A$ and $q \in V$. We argue that

$q' \in V'$ if and only if there is no s_i-escaping facet f on q. If such a facet f exists, then q' is strictly contained within a line segment in C', in particular the line segment between s'_i and $q' + \epsilon\overrightarrow{v}$ where ϵ is an infinitesimal and $\overrightarrow{v}$ is the direction vector of $\overrightarrow{r}(s'_i, q')$. It follows that q' cannot be a vertex of C'. If such a facet f does not exist, then the ray $\overrightarrow{r}(s'_i, q')$ has an empty intersection with C' and thus q' is on the boundary of C'. Since points in $A \cup \{q\}$ are in general position, q' is a vertex. □

Given a subset of sites $S_\alpha \subseteq S\backslash\{s_i\}$ of size $(d-1)$, define $f(L_\alpha)$ to be the $(d-1)$-dimensional simplex whose vertices are the points in S_α and q. Since $s_i \equiv \lambda(A \cup \{q\})$ implies $s_i \in A$, we can use Lemma 4.3.1 to decompose the ith term as follows:

$$\Pr\big[\, s_i \equiv \lambda(A \cup \{q\}) \ \wedge\ q \in V \,\big] = \Pr\big[\, s_i \equiv \lambda(A \cup \{q\}) \ \wedge\ q' \in V' \,\big]$$
$$+ \sum_{\substack{L_\alpha \subseteq S\backslash\{s_i\} \\ |L_\alpha| = (d-1) \\ f(L_\alpha) \text{ is } s_i\text{-escaping for } q}} \Pr\big[\, s_i \equiv \lambda(A \cup \{q\}) \ \wedge\ f(L_\alpha) \text{ is a facet of } C \,\big].$$

The first term is an instance of the same problem in $(d-1)$ dimensions (for the point q' and the projection of S), and thus is computed recursively. For the second term, we compute the probability that $f(L_\alpha)$ is a facet of C as follows. Let $G_1 \subseteq S$ be the subset of sites which are on the other side of the hyperplane supporting $f(L_\alpha)$ with respect to s_i. Let $G_2 \subseteq S$ be the subset of sites that are below s_i along dth axis. Clearly, $f(L_\alpha)$ is a facet of C (and $s_i \equiv \lambda(A \cup \{q\})$) if and only if all points in L_α and s_i exist in A, and

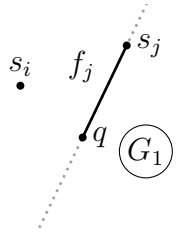


Figure 4.5: A facet f_j projected to the orthogonal complement plane.

all points in $G_1 \cup G_2$ is absent from A. The corresponding probability can be written as

$$\pi_i \times \prod_{j \mid s_j \in L_\alpha} \pi_j \times \prod_{j \mid s_j \in (G_1 \cup G_2)} \overline{\pi_j}$$

This formula is valid only if $L_\alpha \cap G_2 = \emptyset$ and s_i has a higher dth coordinate than q; otherwise we set the probability to zero. This expression takes linear time, and the whole summation term can be computed in $O(n^d)$ time. Then, by induction, the computation of the ith term takes $O(n^d)$ time. The base case of our induction is a two-dimensional membership probability problem with the additional condition $s_i \equiv \lambda(A \cup \{q\})$. Our two dimensional algorithm can be easily adapted to solve this problem in $O(n \log n)$ time as well.

Similar to the planar case, we can improve the computation time for the ith term to $O(n^{d-1})$ by considering the facets $f(L_\alpha)$ in radial order. In particular, let $L_\beta \subseteq S$ be a subset of $(d-2)$ sites. Let f_j denote the $(d-1)$-dimensional simplex $f(L_\beta \cup \{s_j\})$ where $s_j \notin L_\beta$ and $s_j \neq s_i$. We can compute the probability that f_j is a facet of C for all facets f_j in constant amortized time as follows. We project all sites to the two-dimensional plane passing through q and orthogonal to the $(d-2)$-dimensional hyperplane defined

by $L_\beta \cup \{q\}$. (Such a plane is known as an orthogonal complement.) The hyperplane defined by $L_\beta \cup \{q\}$ projects onto q on this plane. Moreover, each facet f_j projects to a line segment extending from q. When we need to compute the probability that f_j is a facet of C, the set G_1 includes the sites on the other side of the line supporting f_j's projection with respect to s_i. (See Figure 4.5.) We compute probabilities for the facets f_j based on their radial order around q. The probability for the next facet in the sweep can be computed by modifying the probability of the previous facet in constant amortized time as we have done for the planar case, as we can efficiently track how G_1 changes. As a final note, we point out that the total cost of all sorting involved is $O(n^{d-1} \log n)$ which is less than the overall cost of $O(n^d)$.

Theorem 4.3.2. *Given a set $\mathcal{P}$ of n uncertain points in the unipoint model and a point q in d-space (where $d \geq 3$), one can compute the membership probability of q in $O(n^d)$ time and linear space, provided that $\mathcal{P} \cup \{q\}$ is non-degenerate.*

4.3.2 The Multipoint Model

As in the planar case, the d-dimensional algorithm easily extends to the mulitpoint model. As before, we compute $\mu(q)$ by computing the probability $\Pr[\, q \in V \,]$. Follow-

ing the same strategy, we decompose it as

$$\Pr\Big[\, q \in V \,\Big] = \Pr\Big[\, q \equiv \lambda(A \cup \{q\}) \,\Big] + \sum_{1 \le i \le m} \left(\sum_{1 \le j \le |S_i|} \Pr\Big[\, s_i^j \equiv \lambda(A \cup \{q\}) \ \wedge \ q \in V \,\Big] \right).$$

It is trivial to compute the first term in $O(n)$ time. We now show how to compute each term inside the summations in $O(n^{d-1})$ time. This implies a total time of $O(n^d)$.

Clearly, Lemma 4.3.1 extends to the multipoint model, so we can use s_i^j-escaping facets to decompose our probability. Given a subset of sites $S_\alpha \subseteq S \setminus \{s_i^j\}$ of size $(d-1)$, define $f(L_\alpha)$ to be the $(d-1)$-dimensional simplex whose vertices are the points in S_α and q. Then,

$$\Pr\Big[\, s_i^j \equiv \lambda(A \cup \{q\}) \ \wedge \ q \in V \,\Big] = \Pr\Big[\, s_i^j \equiv \lambda(A \cup \{q\}) \ \wedge \ q' \in V' \,\Big] + \sum_{\substack{L_\alpha \subseteq S \setminus \{s_i^j\} \\ |L_\alpha| = (d-1) \\ f(L_\alpha) \text{ is } s_i^j\text{-escaping for } q}} \Pr\Big[\, s_i^j \equiv \lambda(A \cup \{q\}) \ \wedge \ f(L_\alpha) \text{ is a facet of } C \,\Big].$$

The first term is computed recursively. We compute each term of the summation as follows. Let I_α be the set of uncertain point indices of the sites in L_α, i.e., $I_\alpha = \{u \mid \exists v \,.\, s_u^v \in L_\alpha\}$. As before, let $G_1 \subseteq S$ be the subset of sites which are on the other side of the hyperplane supporting $f(L_\alpha)$ with respect to s_i^j. Let $G_2 \subseteq S$ be the subset of sites that are below s_i^j along the x_d-axis. Following the same strategy, we write the desired probability as the probability that all points in L_α and s_i^j exist in A, and all

points in $G_1 \cup G_2$ are absent from A. This probability is clearly zero, if any of the following conditions hold:

- $L_\alpha \cap G_2 \neq \emptyset$.
- s_i^j has a higher x_d-coordinate than q.
- L_α contains any two sites from the same uncertain point S_k.
- L_α contains any site from S_i.

Otherwise, we can write the probability as follows:

$$\begin{aligned}
&\Pr\big[\, s_i^j \in A \ \wedge\ L_\alpha \cap A = L_\alpha \ \wedge\ A \cap (G_1 \cup G_2) = \emptyset \,\big] \\
&\quad = \Pr\big[\, s_i^j \in A \,\big] \ \times\ \Pr\big[\, L_\alpha \cap A = L_\alpha \mid s_i^j \in A \,\big] \ \times \\
&\qquad \Pr\big[\, A \cap (G_1 \cup G_2) = \emptyset \mid s_i^j \in A \ \wedge\ L_\alpha \cap A = L_\alpha \,\big] \\
&\quad = \Pr\big[\, s_i^j \in A \,\big] \ \times\ \Pr\Big[\, L_\alpha \cap A = L_\alpha \,\Big] \ \times \\
&\qquad \Pr\big[\, A \cap (G_1 \cup G_2) = \emptyset \mid s_i^j \in A \ \wedge\ L_\alpha \cap A = L_\alpha \,\big] \\
&\quad = \Pr\big[\, s_i^j \in A \,\big] \ \times\ \Pr\Big[\, L_\alpha \cap A = L_\alpha \,\Big] \ \times \\
&\qquad \prod_{\substack{1 \le u \le m \\ u \neq i \\ u \notin I_\alpha}} \Big(\Pr[\, S_u \cap A \cap (G_1 \cup G_2) = \emptyset \,] \Big) \\
&\quad = \pi_i^j \ \times \prod_{u,v \,\mid\, s_u^v \in L_\alpha} \pi_u^v \ \times \prod_{\substack{1 \le u \le m \\ u \neq i \\ u \notin I_\alpha}} \left(1 - \sum_{v \,\mid\, s_u^v \in (G_1 \cup G_2)} \pi_u^v \right).
\end{aligned}$$

The expression takes linear time to compute and thus the summation term can be computed in $O(n^d)$ time. Then, by induction, the computation of the term for the site s_i^j takes $O(n^d)$ time. As before, we can improve this computation time to $O(n^{d-1})$ by considering the facets $f(L_\alpha)$ in radial order. This implies a total complexity of $O(n^d)$ for the algorithm.

Theorem 4.3.3. *Given a set $\mathcal{P}$ of uncertain points in the multipoint model (with n sites in total) and a point q in d-space (where $d \geq 3$), one can compute the membership probability of q in $O(n^d)$ time and linear space, provided that $\mathcal{P} \cup \{q\}$ is non-degenerate.*

4.4 The Probability Map

In some applications, one might be interested in seeing how the uncertain convex hull is distributed over the space. In particular, one can construct a probability map that plots the membership probability throughout the whole space. Figure 4.6 shows how such a probability map might look like in two dimensions. In this section, we investigate complexity of probability maps.

Given a set $\mathcal{P}$ of uncertain points, the *probability map* of $\mathcal{P}$, denoted $\mathbb{M}(\mathcal{P})$, is the subdivision of $\mathbb{R}^d$ into maximal connected regions so that $\mu(q)$ is the same for all query points q in a region. The following lemma gives a tight bound on the size of $\mathbb{M}(\mathcal{P})$.

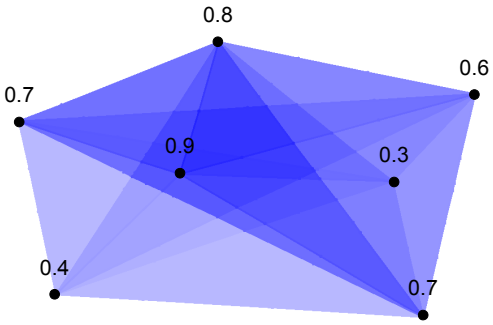


Figure 4.6: A set of uncertain points in the unipoint model and the corresponding probability map. The existence probability of each point is shown above it. In the map, the higher the color intensity, the higher the membership probability is.

Lemma 4.4.1. *The worst-case complexity of the probability map of a set of uncertain points in $\mathbb{R}^d$ is $O(n^{d^2})$, under both the unipoint and the multipoint model, where n is the total number of sites in the input.*

Proof. We prove the result for the unipoint model; extension to the multipoint is straightforward. Consider the set Φ of $O(n^d)$ hyperplanes formed by all d-tuple of points in $\mathcal{P}$. In the arrangement $\mathcal{A}(\Phi)$ formed by these planes, each cell (including the lower dimensional cells bounding the d-dimensional cells) has a constant value of $\mu(q)$, because all points in it are contained by the same set of convex hulls formed among the uncertain point set. It follows that the probability map is a refinement of this arrangement. Since the arrangement has size $O((n^d)^d) = O(n^{d^2})$ [37], the upper bound follows. □

4.4.1 Computing the Probability Map in the Plane

In this section, we describe how to compute the *probability map* for a given set of uncertain points on the plane in $O(n^4)$ time. For simplicity, we assume that the input is given in the unipoint model, however, we briefly explain how to extend the algorithm to the multipoint model.

The high level idea of our algorithm is as follows. Recall that the structure of the probability map is an arrangement of $O(n^2)$ lines that contains $O(n^4)$ faces, edges and vertices. We first construct the whole arrangement in $O(n^4)$ time by using an algorithm by Edelsbrunner et al. [37]. Next, we compute the membership probability of one of the faces in the arrangement, say F, in $O(n \log n)$ time. We then compute the membership probabilities of the vertices, edges and faces neighboring F, in $O(1)$ time per each, by modifying the probability of F.[16] We then apply the same process for all faces neighboring F. By repeatedly expanding into the neighboring faces, we can compute the probability for all of the arrangement in $O(n^4)$ time.

We now show how to compute the probability of a face F' by using the already computed probability of one of its neighboring faces F. We later explain how this algorithm can be adapted to compute the probability of neighboring edges and vertices.

[16]For ease of presentation, we assume that the arrangement is non-degenerate. It is straightforward to apply our technique on degenerate arrangements by using standard techniques (such as perturbation) to create a non-degenerate arrangement. We note that, even if we perturb the points to create a non-degenerate arrangement, we still use the old coordinates of the points and utilize the degeneracy handling rules of Section 4.2.3 while computing probabilities.

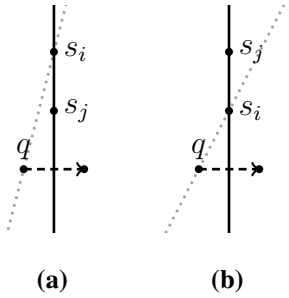


Figure 4.7: The cases to consider for computing the probability of F' from F.

Without loss of generality, assume that F and F' are separated by a vertical line passing through the sites s_i and s_j and F is to the left of F'. Notice that the boundary separating F and F' is only a segment of the vertical line and does not contain s_i or s_j. Now imagine that a point q moves through this boundary, crossing from F to F'. It is easy to see that the change in the membership probability of q is due to the changes in witness edge probabilities of the segments qs_i and qs_j, as other sites are irrelevant. We now describe the change in the witness edge probability of qs_i. The probability of qs_j changes analogously. The change in the probability of qs_i happens differently for two cases (See Figure 4.7):

(a) s_i **is above** s_j**:** Then, s_j switches from the right side of the line $\overleftrightarrow{qs_i}$ to its left side (where right direction is with respect to the vector $\vec{qs_i}$). Consequently, the probability of qs_i changes by a factor of $\frac{1}{\pi_j}$.

(b) s_i **is below** s_j**:** Then, s_j switches from the left side of the line $\overleftrightarrow{qs_i}$ to its right side. Consequently, the probability of qs_i changes by a factor of $\overline{\pi_j}$.

The changes clearly require constant time operations, and thus the membership probability of F' can be computed in $O(1)$ time.

We now describe how to compute the membership probability of an edge e bounding F by using the already computed probability of F. Notice that any query point q on an edge e is degenerate with respect to the uncertain points. Therefore, we have to make use of the degeneracy handling rules from Section 4.2.3. Without loss of generality, assume that e is on a vertical line passing through the sites s_i and s_j and F is to the left of e. Notice that e is only a segment of the vertical line and does not contain s_i or s_j. Now imagine that a point q moves from F onto e. Again, the change in the membership probability of q is due to the changes in witness edge probabilities of the segments qs_i and qs_j. We describe the change in the witness edge probability of qs_i, the change for qs_j is analogous. We consider six different cases based on the vertical order of the points q,s_i and s_j (See Figure 4.8):

(a) **Order** q, s_i, s_j**:** s_j switches from the right side of $\overleftrightarrow{qs_i}$ to $\overleftrightarrow{qs_i}$. Based on the degeneracy rules, it still requires s_j to be non-existent for qs_i to be a witness edge. Hence, there is no change to the witness edge probability of qs_i.

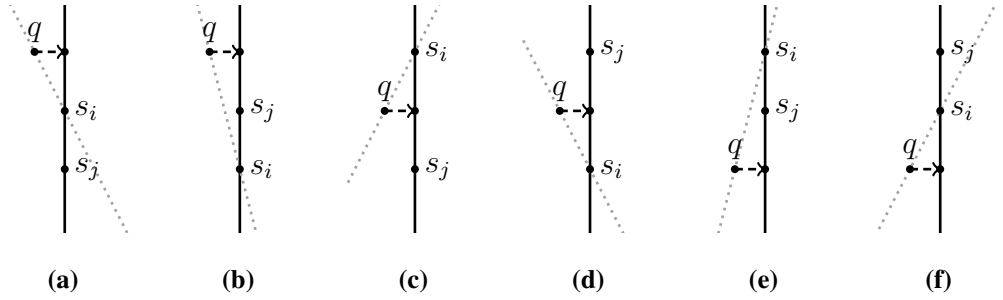


Figure 4.8: The cases to consider for computing the probability of e from F.

(b) **Order** q, s_j, s_i**:** s_j switches from the left side of $\overleftrightarrow{qs_i}$ to $\overleftrightarrow{qs_i}$. Based on the degeneracy rules, s_j has still no effect on the witness edge property of qs_i. Hence, there is no change.

(c) **Order** s_i, q, s_j**:** s_j switches from the right side of $\overleftrightarrow{qs_i}$ to $\overleftrightarrow{qs_i}$. Based on the degeneracy rules, it still requires s_j to be non-existent for qs_i to be a witness edge. Hence, there is no change.

(d) **Order** s_j, q, s_i**:** s_j switches from the left side of $\overleftrightarrow{qs_i}$ to $\overleftrightarrow{qs_i}$. Based on the degeneracy rules, it requires s_j to be non-existent for qs_i to be a witness edge. Hence, the witness edge probability of qs_i changes by $\overline{\pi_j}$.

(e) **Order** s_i, s_j, q**:** s_j switches from the right side of $\overleftrightarrow{qs_i}$ to $\overleftrightarrow{qs_i}$. Based on the degeneracy rules, s_j has no effect on the witness edge property of qs_i. Hence, the witness edge probability of qs_i changes by $\frac{1}{\overline{\pi_j}}$.

(f) **Order** s_j, s_i, q**:** s_j switches from the left side of $\overleftrightarrow{qs_i}$ to $\overleftrightarrow{qs_i}$. Based on the degeneracy rules, it requires s_j to be non-existent for qs_i to be a witness edge. Hence, the witness edge probability of qs_i changes by $\overline{\pi_j}$.

The changes clearly require $O(1)$ time.

Finally, we explain how to compute the probability of the vertices. Let v be a vertex of the arrangement such that v is an endpoint of two edges e_1 and e_2 that bound F. Then, the membership probability of v is computed by applying the same probability changes that is applied to e_1 and e_2. The only exception to this is when v coincides a site s_i. In that case, we compute the membership probability of v from scratch in $O(n \log n)$ time. Since the number of such vertices is linear, it does not increase our overall cost of $O(n^4)$.

The extension of our technique to the multipoint model is straightforward. The only major difference is that we need to remember (similar to what is done in Section 4.3.2) the intermediate factors when computing face probabilities, as updating the witness edge probabilities requires updating these factors first. The total cost of a single update remains $O(1)$ because it requires updating one intermediate factor of two witness edge probabilities.

Theorem 4.4.2. *Given a set $\mathcal{P}$ of uncertain points in the plane (with n sites in total), one can compute $\mathbb{M}(\mathcal{P})$ in $O(n^4)$ time.*

Once the probability map is computed, one can construct a data structure to answer membership probability queries. Using the point location structure by Kirkpatrick [54], we can construct such a data structure in $O(n^4)$ time. The structure consumes $O(n^4)$ space and can answer queries in $O(\log n)$ time.

4.5 Conclusion

In this chapter, we investigated convex hulls under uncertainty with an emphasis on computing the membership probability. One closely related problem that we leave open is the efficient computation of the membership probability *for degenerate input* in dimensions higher than 3. While we believe an adaptation of our high-dimensional algorithm to handle degeneracies is possible, it seems to be a difficult task. Being unable to compute the membership probability efficiently for degenerate input sets also means that we cannot efficiently compute complete probability maps as such maps would include many points that are degenerate with respect to the uncertain point set.

Making sense of uncertain data is a complex and challenging task. For simple numerical data, elementary statistics such as mean, median, or mode serve a useful first order approximation. For multi-dimensional spatial data, however, there are no universally agreed upon summaries of similar generality. Our work on membership proba-

bility and probability maps is an attempt to explore some natural geometric structures, and their complexity, over probabilistic data.

Chapter 5

The Most Likely Convex Hull of Uncertain Points*

5.1 Introduction

In this chapter, we study the problem of computing the *most likely convex hull* of uncertain points. The problem is fundamental in its own right, extending the notion of minimal convex enclosure to probabilistic input, but is also motivated by a number of applications dealing with noisy data. Before formalizing the problem, let us mention some motivating scenarios for our problem. In *movement ecology* [43, 44], scientists track the movements of a group of animals using sensors with the goal of inferring

*This chapter is based on a joint work with Subhash Suri and Kevin Verbeek. Parts of this chapter appeared in the following publications: [70] (Published and copyright held by Springer. The final publication available at http://link.springer.com/10.1007/978-3-642-40450-4_67.)

their natural "home range". The ecologists have long known that the smallest convex polygon containing all possible locations visited by the animals is a gross overestimation of the home range, due to the outlier problem, and instead have begun to consider probability-based isopleths. The most likely hull is one possible tool in this analysis: use a discrete set of landmarks (points), assign probability to each based on the frequency of the animals' visits to the landmarks, and compute the most likely convex hull of this probabilistic set of points as the most probable home range. As another example, consider monitoring of a large geographic area for physical activity (e.g., earthquake tremors). After collecting data over a period of time, we want to estimate the most likely region of activity. Since the value of a prediction decreases sharply with the rate of false positives, we want to find the tightest region for expected activity, and the most likely hull is a natural candidate. Finally, as a growing number of applications rely on machine learning and data mining for classification, we are inevitably forced to work with data whose attributes are inherently probabilistic. Computing meaningful geometric structures over these data is an interesting, and challenging, algorithmic problem. The most likely hull is a convenient vehicle to investigate these types of problems, although our methods and results are applicable more broadly, as discussed later.

As done in the previous chapter, our study focusses on two models of uncertainty: the unipoint model and the multipoint model. We briefly explain two models again for the completeness of this chapter.

In the *unipoint model*, each input point has a fixed location but it only exists probabilistically. Specifically, the input $\mathcal{P}$ is a set of pairs $\{(s_1, \pi_1), \ldots, (s_n, \pi_n)\}$ where each s_i is a point in d-space and each π_i is a real number in the range $(0, 1]$ denoting the probability of s_i's existence. The existence probabilities of different points are independent.

The *multipoint* model generalizes the unipoint model to incorporate *locational uncertainty*. In the *multipoint model*, each point probabilistically exists at one of multiple possible sites. Specifically, $\mathcal{P}$ is a set of pairs $\{(S_1, \Pi_1), \ldots, (S_m, \Pi_m)\}$ where each S_i is a set of points and each Π_i is a set of reals in the range $(0, 1]$, with $|S_i| = |\Pi_i|$. The set $S_i = \{s_i^1, \ldots, s_i^{|S_i|}\}$ describes the possible sites for the ith point of $\mathcal{P}$ and the set $\Pi_i = \{\pi_i^1, \ldots, \pi_i^{|\Pi_i|}\}$ describes the associated probability distribution. The probabilities π_i^j correspond to disjoint events and therefore sum to at most 1. By allowing the sum to be less than one, this model also accounts for the possibility that the point does not exist (i.e. the *null* location)—thus, the multipoint model strictly generalizes the unipoint model. In the multipoint model, the number of points in $\mathcal{P}$ is referred by m and we use n to refer the total number of possible sites, i.e., $n = \sum_i |S_i|$.

In either setting, the convex hull of $\mathcal{P}$ is a random variable, which assumes values over the convex hulls of the (at most) 2^n possible subsets. We are interested in computing the *most likely convex hull* for $\mathcal{P}$, which is the mode of the convex hull variable.

Results

Our first result shows that the most likely hull of points in 2 dimensions in the unipoint model can be found in $O(n^3)$ time. We then show that the problem becomes NP-hard for dimensions $d \geq 3$. We also show an inapproximability results. In particular, computing a hull whose likelihood is within factor $2^{-O(n^{1-\epsilon})}$ of the optimal is NP–hard. This is nearly tight because a factor-(2^{-n}) approximate hull is easily computed by a simple greedy algorithm. Under the multipoint model, we show that the most likely hull problem is NP-hard even in two dimensions, and also inapproximable to a factor better than $2^{-O(n^{1-\epsilon})}$ unless P=NP. Note that in both models the problem is clearly in P for $d = 1$, since the number of distinct convex hulls in one dimension is only polynomial. While we focus on the most likely hull as a natural and concrete example, our algorithms and techniques apply more broadly to other possible ways of defining a probabilistic convex hull.

Chapter Organization

In Section 5.2, we explain our dynamic programming algorithm to compute the most likely hull in the planar unipoint model. In Sections 5.3, we prove that computing the most likely hull in three or higher dimensions is NP-hard. In Section 5.4, we describe our NP-hardness for the multipoint model. We finish with a brief overview of possible extensions and concluding remarks in 5.5.

5.2 Two-Dimensional Most Likely Hull in the Unipoint Model

In this section, we describe a dynamic programming algorithm for computing the most likely hull of n points in the plane under the unipoint model of uncertainty. For simplicity, we assume that no three points are collinear, but the algorithm is easily modified to handle such degeneracies. We begin with some general technical facts related to convex hulls of uncertain points in the unipoint model.

Let $\mathcal{P} = \{(s_1, \pi_1), \ldots, (s_n, \pi_n)\}$ denote the input to the uncertain convex hull problem in d-space and let S be the set of sites, i.e., $S = \{s_1, \ldots, s_n\}$. A subset $A \subseteq S$ occurs as an outcome of a probabilistic experiment with probability $\pi(A)$ given by

$$\pi(A) \;=\; \prod_{s_i \in A} \pi_i \;\times\; \prod_{s_i \notin A} \overline{\pi_i}$$

where we use the notation $\overline{\pi_i} = (1 - \pi_i)$. Given an outcome A, its convex hull is denoted as $\mathrm{conv}(A)$. For a convex polytope C, we define its *likelihood*, denoted $\mathcal{L}(C)$, as the probability that C is the convex hull of the random outcome of a probabilistic experiment on $\mathcal{P}$. In other words,

$$\mathcal{L}(C) \;=\; \Pr\big[\; \mathrm{conv}(A) \equiv C \;\big] = \sum_{\substack{A \subseteq S \\ \mathrm{conv}(A) \equiv C}} \pi(A)$$

The *most likely hull* of $\mathcal{P}$ is the polytope C with the maximum value of $\mathcal{L}(C)$. Our first lemma shows that $\mathcal{L}(C)$ can be written as a product of two factors where the first factor involves only the *vertices* of C, and not all the sites that fall inside C.

Lemma 5.2.1. *Let C be a convex polytope, $V \subseteq S$ be its vertex set, and $S_{out} \subseteq S$ the set of sites lying outside C. Then, we have the following:*

$$\mathcal{L}(C) \;=\; \prod_{s_i \in V} \pi_i \times \prod_{s_i \in S_{out}} \overline{\pi_i},$$

Proof. Let S_{in} denote the sites contained by C (possibly on the boundary). Then,

$$\begin{aligned}
\mathcal{L}(C) &= \sum_{A \subseteq S \;\wedge\; \mathrm{conv}(A) = C} \pi(A) \\
&= \sum_{V \subseteq A \subseteq S_{in}} \pi(A) \\
&= \sum_{V \subseteq A \subseteq S_{in}} \left(\prod_{s_i \in A} \pi_i \times \prod_{s_i \notin A} \overline{\pi_i} \right)
\end{aligned}$$

$$= \sum_{\substack{A=V \uplus A' \\ A' \subseteq (S_{in} \setminus V)}} \left(\prod_{s_i \in V} \pi_i \times \prod_{s_i \in S_{out}} \overline{\pi_i} \times \prod_{s_i \in A'} \pi_i \times \prod_{s_i \in (S_{in} \setminus V) \setminus A'} \overline{\pi_i} \right)$$

$$= \prod_{s_i \in V} \pi_i \times \prod_{s_i \in S_{out}} \overline{\pi_i} \times \sum_{A' \subseteq S_{in} \setminus V} \left(\prod_{s_i \in A'} \pi_i \times \prod_{s_i \in (S_{in} \setminus V) \setminus A'} \overline{\pi_i} \right)$$

$$= \prod_{s_i \in V} \pi_i \times \prod_{s_i \in S_{out}} \overline{\pi_i} \times \prod_{s_i \subseteq S_{in} \setminus V} (\pi_i + \overline{\pi_i})$$

$$= \prod_{s_i \in V} \pi_i \times \prod_{s_i \in S_{out}} \overline{\pi_i}$$

□

5.2.1 Likelihood Contributions of Edges

We now describe how to find the most likely hull for a 2-dimensional input under the unipoint model. Our algorithm computes, for each site s_i, the most likely hull with s_i as its lowest (minimum y-coordinate) vertex, and then outputs the best hull over all choices of s_i. For ease of reference, let us call a convex polygon with s_i as its lowest vertex, a *hull rooted at* s_i. We decompose the likelihood of a convex hull into several components, each associated with an edge of the hull. The key to the computational efficiency is to ensure that the component associated with an edge *does not depend on the hull* in which the edge participates. Geometrically, we associate a *wedge shaped region* with each edge, depending only on the choice of the lowest vertex, and define

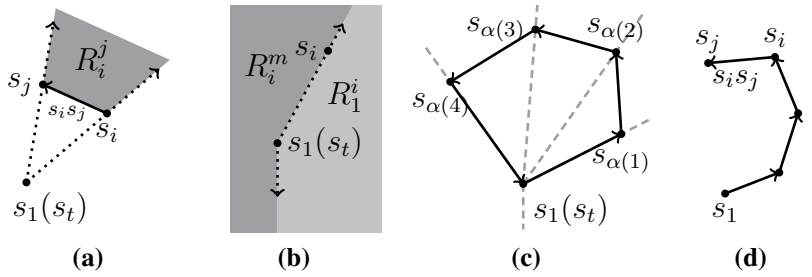


Figure 5.1: Illustrations for the two-dimensional most likely hull in the unipoint model.

the contribution based only on the sites contained in this wedge. We now discuss this in more details.

Suppose we want to compute the most likely hull rooted at s_1. Without loss of generality, let $s_2, \ldots, s_{t-1}$ be the sequence of sites (all lying above s_1) in the counter-clockwise order around s_1, for $(t-1) \leq n$. Any hull rooted at s_1 has a subsequence of $s_1, \ldots, s_{t-1}$ as its vertex set. Finally, for notational convenience, we add an artificial site $s_t = s_1$ (a copy of the root point) with probability zero.

Given two sites s_i and s_j, with $1 \leq i < j \leq t$, we use $s_i s_j$ to denote the *directed edge* drawn from s_i to s_j. To each directed edge $s_i s_j$, we associate a region of space R_i^j. For an edge not involving s_1 or its copy s_t, namely $s_i s_j$, for $1 < i < j < t$, R_i^j is the region bounded by the segment $s_i s_j$ and the rays $\overrightarrow{s_1 s_i}$ and $\overrightarrow{s_1 s_j}$. See Figure 5.1a for illustration. For edges with the first endpoint at s_1, namely $s_1 s_i$, for $1 < i < t$, R_1^i is the region bounded (on its left) by the downward ray extending from s_1 and the ray $\overrightarrow{s_1 s_i}$. The complementary region of R_1^i is also important, and we call it R_i^t, associated with the edge $s_i s_t$, which is the reverse edge of $s_1 s_i$. See Figure 5.1b.

We now define the *contribution* of the directed edge $s_i s_j$, denoted $\mathcal{C}(s_i s_j)$, as π_i times the probability that none of the sites in the region R_i^j (except s_i and s_j) are present, including the sites that may lie below s_1. That is,

$$\mathcal{C}(s_i s_j) = \pi_i \times \prod_{s_k \in R_i^j} \overline{\pi_k}$$

The following lemma shows how these edge contributions help us compute the likelihood of a convex hull C.

Lemma 5.2.2. *Let C be a hull rooted at s_1, with vertices $s_1, s_{\alpha(1)}, \ldots, s_{\alpha(\ell)}$ in the counter-clockwise order. Then,*

$$\mathcal{L}(C) = \mathcal{C}(s_1 s_{\alpha(1)}) \times \mathcal{C}(s_{\alpha(1)} s_{\alpha(2)}) \times \cdots \times \mathcal{C}(s_{\alpha(\ell-1)} s_{\alpha(\ell)}) \times \mathcal{C}(s_{\alpha(\ell)} s_t)$$

Proof. Partition the space outside C into the regions $R_1^{\alpha(1)}, R_{\alpha(1)}^{\alpha(2)}, \ldots, R_{\alpha(\ell-1)}^{\alpha(\ell)}, R_{\alpha(\ell)}^{t}$ by drawing a downward ray from s_1 and drawing rays $\overrightarrow{s_1 s_{\alpha(j)}}$ for each $1 \leq j \leq \ell$. (See Figure 5.1c for an example.) Then, by Lemma 5.2.1, it is easy to see that the $\mathcal{L}(C)$ is the product of the contributions of the edges of C. □

The contribution of each edge can be computed in constant time after an $O(n^2)$-time preprocessing. The main idea is to utilize a modified version of a triangle query structure by [38]. In particular, we have the following lemma from [38].

Lemma 5.2.3. *Given a set $\mathcal{P}$ of n points in the plane, one can preprocess $\mathcal{P}$ in $O(n^2)$ time and space, so that the number of points in $\mathcal{P}$ contained by a given query triangle (with corners among $\mathcal{P}$) can be reported in constant time.*

It is trivial to modify this data structure so that, under an assignment of weights to the set of points, one can report the product of the weights of the points in the query triangle. In particular, we have the following lemma.

Lemma 5.2.4. *Let $\mathcal{P}$ be a set of n points in the plane such that each point is assigned a weight. One can preprocess $\mathcal{P}$ in $O(n^2)$ time and space, so that the product of the weights of all points in $\mathcal{P}$ contained by a given query triangle (with corners among $\mathcal{P}$) can be reported in constant time.*

We now show how to query edge contributions using this data structure. Recall that S is the set of all sites, and we want to compute edge contributions with respect to lowest vertex s_1. Let U be the set of the four corners of the bounding box of S. Moreover, let V be the set of points produced by intersecting the bounding box of S with the downward ray extending from s_1 and the rays $s_1 s_i$ for all s_i. Clearly, $|V| \leq n$. We construct an instance of the weighted triangle query structure on $S \cup U \cup V$. In this structure, we define the weight of each point s_i in S as its corresponding complementary probability, i.e., $\overline{\pi_i}$. For all points in U and V, we define the weight as 1.

Given an edge $s_i s_j$, we can compute its contribution as follows. The region R_i^j restricted to the bounding box of S is a polygon of constant complexity. We triangulate this polygon, and for each triangle, query the product of the weights of the points in the triangle. (See Figure 5.2.) The results of these queries, when multiplied, gives the

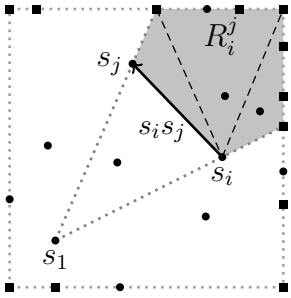


Figure 5.2: Triangulating R_i^j inside the bounding box of S. The black circles are the sites in S. The black squares the points in U and V.

product of complementary probabilities of all sites in R_i^j, which is what we need to compute $\mathcal{C}(s_i s_j)$.

5.2.2 The Dynamic Programming Algorithm

Our dynamic programming algorithm computes, for each edge $s_i s_j$, the convex chain whose edges yield the *maximum product of contributions* under the following constraints:

1. The sequence of vertices in the chain is a subsequence of $s_1, \ldots, s_t$.

2. The first vertex of the chain is s_1.

3. The last edge of the chain is $s_i s_j$. (See Figure 5.1d for an example.)

We denote this maximum chain by $\mathcal{T}(s_i s_j)$. With a slight abuse of notation, we also use $\mathcal{T}(s_i s_j)$ to denote the product of the edge contributions of this chain. Clearly, all chains of the form $\mathcal{T}(s_i s_t)$ correspond to polygons rooted at s_1, and the one with the maximum

contribution is the most likely hull we want. Our dynamic programming formulation is fairly standard, and similar style of algorithms have been used in the past for computing largest convex subsets [9, 32] and monochromatic islands [10].

We now describe an optimal substructure property crucial for our dynamic programming algorithm. Consider a chain $\mathcal{T}(s_i s_j)$. This, by definition, has the maximum likelihood of all chains terminating with the edge $s_i s_j$. If we remove the last vertex s_j of $\mathcal{T}(s_i s_j)$, and the corresponding edge $s_i s_j$, then the remaining chain should be the optimal chain terminating at s_i *that can be extended to* s_j *without violating convexity*. In other words, the remaining chain is the maximum among all chains $\mathcal{T}(s_k s_i)$ (where $1 \leq k < i$) such that the path $s_k \to s_i \to s_j$ is a left turn. This implies the following recurrence:

$$\mathcal{T}(s_i s_j) = \begin{cases} \mathcal{C}(s_1 s_j) & \text{if } i = 1 \\ \mathcal{C}(s_i s_j) \times \max\limits_{\substack{1 \leq k < i \\ s_k \to s_i \to s_j \text{ is a left turn}}} \Big(\mathcal{T}(s_k s_i)\Big) & \text{otherwise} \end{cases}$$

We use this recurrence to compute all the chains $\mathcal{T}(s_i s_j)$ as follows. We begin by setting $\mathcal{T}(s_1 s_i)$ to $\mathcal{C}(s_1 s_i)$ for all $1 < i \leq t$. Then, we process all sites s_i in increasing order of i. When we process a site s_i, we compute all chains $\mathcal{T}(s_i s_j)$ by using the previously computed chains. This can be done in $O(n)$ time as follows. Let S_{prec} be the set of sites $\{s_1, \ldots, s_{i-1}\}$ and S_{succ} be the set $\{s_{i+1}, \ldots, s_t\}$. Let $s_{\beta(1)}, \ldots, s_{\beta(\ell)}$ be

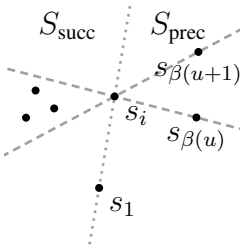


Figure 5.3: Wedge for $\mathcal{T}(s_i s_j)$.

the sites in S_{prec} in counter-clockwise order around s_i, starting with s_1.[17] For each site $s_{\beta(u)}$ in S_{prec}, we define s_u^* to be the site s_k among the sequence $s_{\beta(1)}, \ldots, s_{\beta(u)}$ that maximizes $\mathcal{T}(s_k s_i)$. The site s_u^* can be computed for all sites $s_{\beta(u)}$ with a linear sweep of the sites in S_{prec} in order.

For each site $s_{\beta(u)}$ in S_{prec}, we set the value $\mathcal{T}(s_i s_j)$ to $\mathcal{C}(s_i s_j) \times \mathcal{T}(s_u^* s_i)$ for all sites s_j in S_{succ} inside the wedge bounded by the lines $\overleftrightarrow{s_{\beta(u)} s_i}$ and $\overleftrightarrow{s_{\beta(u+1)} s_i}$.[18] (See Figure 5.3.) Note that the sites in this wedge are the sites that form a left turn when connected to $s_{\beta(1)}, \ldots, s_{\beta(u)}$ through s_i (the condition in the recurrence relation). By considering the sites $s_{\beta(u)}$ in radial order around s_i, we can locate each site in the wedge of interest in constant time.

The processing of a single point s_i takes $O(n)$ time, and thus we can find the most likely hull rooted at s_1 in $O(n^2)$ time, and the global most likely hull of $\mathcal{P}$ in $O(n^3)$ time. The algorithm needs $O(n^2)$ space, dominated by the storage of the $\mathcal{T}(\cdot)$ values.

[17] This counter-clockwise order for all sites s_i can be precomputed in $O(n^2 \log n)$ time.

[18] We also remember how $\mathcal{T}(s_i s_j)$ is computed, so the corresponding chain can be constructed later.

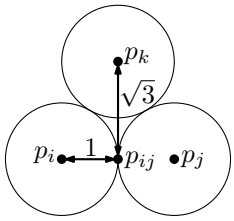


Figure 5.4: Lemma 5.3.1.

Theorem 5.2.5. *The most likely convex hull of an uncertain point set defined by n sites in the unipoint model can be computed in $O(n^3)$ time and in $O(n^2)$ space.*

5.3 Hardness of the 3-Dimensional Most Likely Hull

We now show that computing the most likely hull in 3 or more dimensions is NP-hard in the unipoint model. In particular, we give a reduction from the vertex cover problem in penny graphs to the 3-dimensional most likely hull problem.

A *penny graph* is a graph $G = (V, E)$ along with an embedding $\rho : V \to \mathbb{R}^2$ such that $\|\rho(u) - \rho(v)\|_2 = 2$ if $(u, v) \in E$, and $\|\rho(u) - \rho(v)\|_2 > 2$ if $(u, v) \notin E$, where $\|.\|_2$ denotes the L_2 norm. In other words, a penny graph admits a planar drawing where vertices are represented as unit disks with pairwise disjoint interiors, and two disks make contact if and only if there is an edge between the two corresponding vertices. We denote the centers of the unit disks by the points $p_1, \ldots, p_n$, and the point of contact between two adjacent disks with centers p_i and p_j by p_{ij}. The following

simple observation about the penny graph embedding will be critical in our reduction. See Figure 5.4 for an illustration.

Lemma 5.3.1. $\|p_k - p_{ij}\|_2 \geq \sqrt{3}$, *for all* $k \neq i, j$.

Proof. Consider the triangle formed by p_i, p_j, and p_k. By Heron's formula, the area A of this triangle is at least $\sqrt{3}$ (the sides have length at least 2). Alternatively, the area can be computed as $A = bh/2$, where $b = \|p_i - p_j\|_2 = 2$ and h is the height of triangle. Thus we get that $\|p_k - p_{ij}\|_2 \geq h = A \geq \sqrt{3}$. □

The vertex cover problem for penny graphs is to find the smallest subset $U \subseteq V$ of vertices such that every edge of the graph has an endpoint in U. This problem was shown to be NP-hard in [20]. Our reduction relies on the following simple but important property of the most likely hull in the unipoint model.

Lemma 5.3.2. *Any point* (s_i, π_i) *with* $\pi_i \geq 1/2$ *is in the most likely hull.*

Proof. For the sake of contradiction assume that a site s_k has probability $\pi_k > \frac{1}{2}$ and is outside the most likely hull C. Let $V \subseteq S$ be the set of sites that appear on C as a vertex, and let S_{out} be the set of sites outside C. Now consider adding s_k to C. For the resulting hull C', let V' be the set of vertices of C', and let S'_{out} be the set of sites outside C'. Note that $V' \subseteq V \cup \{s_k\}$ and $S'_{out} \subseteq S_{out} \setminus \{s_k\}$. From Lemma 5.2.1 we

can obtain:

$$\begin{aligned}
\mathcal{L}(C') &= \prod_{s_i \in V'} \pi_i \times \prod_{s_i \in S'_{out}} \overline{\pi_i} \\
&\geq \left(\frac{\pi_k}{1-\pi_k}\right) \prod_{s_i \in V} \pi_i \times \prod_{s_i \in S_{out}} \overline{\pi_i} \\
&= \left(\frac{\pi_k}{1-\pi_k}\right) \mathcal{L}(C) \\
&> \mathcal{L}(C)
\end{aligned}$$

This implies that C is not the most likely hull, contradicting the initial assumption. □

5.3.1 The Reduction

Consider an instance of the vertex cover problem for a penny graph G, with the set $\{p_1, \ldots, p_n\}$ being the disk centers of the embedding of G. We create an instance of the most likely hull problem in three dimensions, as follows. All the sites lie on one of the two paraboloids, $\mathcal{P}_1 : z = x^2 + y^2$ or $\mathcal{P}_2 : z = x^2 + y^2 - 2$. In particular, for each disk center p_i, we create a site u_i by vertically lifting p_i onto the paraboloid $\mathcal{P}_2$. All these points are assigned a fixed probability $\pi_i = \alpha < \frac{1}{2}$.

The sites on $\mathcal{P}_1$ are associated with the contact points p_{ij} but are not a direct lifting of the contact points themselves. Instead, for each contact point $p_{ij} = (x_{ij}, y_{ij})$, we define four new points $p_{ij}^N = (x_{ij}, y_{ij} + \delta)$, $p_{ij}^E = (x_{ij} + \delta, y_{ij})$, $p_{ij}^S = (x_{ij}, y_{ij} - \delta)$, and $p_{ij}^W = (x_{ij} - \delta, y_{ij})$, for some $\delta > 0$. (We set the value of δ later.) Next, we add a set

X_{ij} of t arbitrary points inside the quadrilateral formed by p_{ij}^e ($e \in \{N, E, S, W\}$). We lift each of the p_{ij}^e onto $\mathcal{P}_1$ to obtain a site u_{ij}^e, for $e \in \{N, E, S, W\}$, and each of these points is assigned a probability of 1. Finally, the subsets X_{ij} are lifted onto $\mathcal{P}_1$ to get subsets Y_{ij}, and each of these points are assigned a fixed probability $\beta > \frac{1}{2}$. All these points, lying on the paraboloids $\mathcal{P}_1$ and $\mathcal{P}_2$, along with their associated probabilities form the input for our most likely hull problem.

The main idea of the reduction is that we want to "cover" each set Y_{ij} by putting either u_i or u_j on the most likely hull. In the penny graph, this corresponds to covering the edge associated with the contact point p_{ij} by the vertex associated with p_i or p_j. We now describe this relation in more depth, starting with a well-known lemma about the lifting transform.

Lemma 5.3.3. *Consider a point $p \in \mathbb{R}^2$, and let $u(p)$ be its vertical projection (lifting) onto the paraboloid $\mathcal{P}_1$, and $H(p)$ the hyperplane tangent to $\mathcal{P}_1$ at $u(p)$. Then, the vertical projections $u(p')$ of all points $p' \in \mathbb{R}^2$ at distance r from p lie on a hyperplane parallel to $H(p)$ whose vertical distance from $H(P)$ is r^2.*

Proof. Every plane parallel to $H(p)$ can be represented by the equation $Ax + By + C(z - h) = 0$, where (A, B, C) (with $C \neq 0$) is the normal of the plane with length 1, and h is the vertical shift from origin. By intersecting such a plane with $\mathcal{P}_1$ we obtain the equation $Ax+By+C(x^2+y^2-h) = 0$, which we can rewrite as $(x+\frac{A}{2C})^2+(y+\frac{B}{2C})^2 = h + \frac{A^2+B^2}{4C^2}$. This equation describes a circle with center $(-\frac{A}{2C}, -\frac{B}{2C})$ (independent of

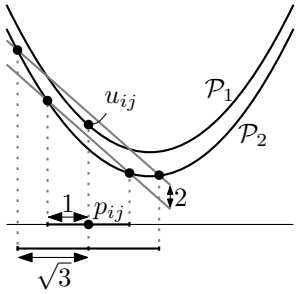


Figure 5.5: Lift to $\mathcal{P}_1$ and $\mathcal{P}_2$ (vertically scaled).

h) and radius $r^2 = h + \frac{A^2+B^2}{4C^2}$. Since the plane is tangent to $\mathcal{P}_1$ when $r = 0$, the result follows. □

The points u_i's (liftings of p_i's) lie on $\mathcal{P}_2$, which is a vertical downward shift of $\mathcal{P}_1$. Now, if u_{ij} is the point obtained by lifting p_{ij} to $\mathcal{P}_1$, then by Lemma 5.3.3 the points u_i and u_j are vertically 1 unit below the tangent plane of $\mathcal{P}_1$ at u_{ij}, while the points u_k ($k \neq i, j$) are at least vertically 1 unit above this plane by Lemma 5.3.1 (see Figure 5.5). If we treat $\mathcal{P}_1$ as an "obstacle", then u_i and u_j can "see" u_{ij} from below, while the points u_k ($k \neq i, j$) cannot. Thus there exists a small enough $\delta > 0$ such that Y_{ij} is contained in the convex hull of u_{ij}^e ($e \in \{N, E, S, W\}$) with either of u_i and u_j, but not with u_k ($k \neq i, j$). The following lemma describes a sufficient upper-bound on δ.

Lemma 5.3.4. *If $\delta < \sqrt{3}-\sqrt{2}$, then the points u_i and u_j can see the entire quadrilateral on $\mathcal{P}_1$ formed by u_{ij}^e ($e \in \{N, E, S, W\}$) from below, but no u_k ($k \neq i, j$) can see any part of the quadrilateral from below.*

Proof. Let $p \in \mathbb{R}^2$ be a point inside the quadrilateral formed by p_{ij}^e ($e \in \{N, E, S, W\}$) and let $q \in \mathcal{P}_1$ be the point obtained by lifting p. By definition, $\|p_i - p\| \leq 1 + \delta$ (same for p_j), and by Lemma 5.3.1 $\|p_k - p\| \geq \sqrt{3} - \delta$ for $k \neq i, j$. We need that u_i and u_j are below the tangent plane of $\mathcal{P}_1$ at q, and u_k ($k \neq i, j$) is above this plane. Since $\mathcal{P}_2$ is 2 units below $\mathcal{P}_1$ and by Lemma 5.3.3, u_i is below the tangent plane if and only if $\|p_i - p\| < \sqrt{2}$. The analogue holds for u_j. Similarly, we need $\|p_k - p\| > \sqrt{2}$ for all u_k. Consequently, we obtain two bounds on δ, namely $\delta < \sqrt{2} - 1$ and $\delta < \sqrt{3} - \sqrt{2}$, of which the latter is the strongest. □

Theorem 5.3.5. *Computing the most likely hull in three dimensions is NP-hard.*

Proof. We show that computing the likelihood of the most likely hull is NP-hard. Given an instance of the vertex cover problem for penny graphs, we construct an instance of the most likely hull problem in three dimensions as described above (e.g., with $\delta = 0.25$). We choose t, α, and β such that $\beta^t < \alpha$, and $\alpha < 0.5 < \beta$; e.g., $t = 3$, $\alpha = 0.25$, and $\beta = 0.6$. By Lemma 5.3.2 all points on $\mathcal{P}_1$ must be on or inside the most likely hull, and so we only need to choose which points u_i ($1 \leq i \leq n$) are on the most likely hull. No point from a set Y_{ij} can be on the most likely hull because then we could add either u_i or u_j to the hull and increase the likelihood of the hull, since $\beta^t(1 - \alpha) < \alpha$. Thus, the likelihood of the most likely hull is determined by the number κ of points u_i ($1 \leq i \leq n$) that are on the most likely hull, and its likelihood is $\alpha^\kappa(1 - \alpha)^{n-\kappa}$. Every point u_i on the most likely hull corresponds to a vertex of the penny graph, and by

construction and Lemma 5.3.4, these vertices form a vertex cover of the penny graph. Thus the penny graph has a vertex cover of size κ if and only if the likelihood of the most likely hull is at least $\alpha^{\kappa}(1-\alpha)^{n-\kappa}$. Finally, it is easy to see that the construction can be performed in polynomial time. □

The proof above directly implies that there is no polynomial-time $(\frac{\alpha}{1-\alpha})$-approximation algorithm to compute the likelihood of the most likely hull unless $P = NP$. Although we can change the value of α to obtain a stronger bound, we give a more general argument below.

5.3.2 Inapproximability

The likelihood of a hull is a product of terms. We show that, under mild conditions, NP-hard optimization problems of this form cannot be approximated well by a multiplicative factor, unless $P = NP$.

Let $\mathcal{O} = (\mathcal{I}, \mathcal{F}, f)$ be an optimization problem where $\mathcal{I}$ is the set of instances, $\mathcal{F}$ is a function over $\mathcal{I}$ such that $\mathcal{F}(I)$ describes the set of feasible solutions for instance I, and f is an optimization function over all feasible solutions. For an instance $I \in \mathcal{I}$, let $|I|$ denote the size of I. We say that $\mathcal{O}$ is *product composable* if, given any collection of problem instances $I_1, \ldots, I_k \in \mathcal{I}$, we can construct a new instance $I^* \in \mathcal{I}$ in polynomial time (w.r.t. $|I^*|$) satisfying the following:

1. $|I^*| = \sum_{i=1}^{k} |I_i|$.

2. There is a bijection between $\mathcal{F}(I^*)$ and $\mathcal{F}(I_1) \times \ldots \times \mathcal{F}(I_k)$ such that for each solution $S \in \mathcal{F}(I^*)$ with the matching tuple $(S_1, \ldots, S_k)$, $f(S) = \prod_{1 \leq i \leq k} f(S_i)$.

3. Given a solution $S \in \mathcal{F}(I^*)$, one can construct the solutions in its matching tuple in polynomial time.

In other words, we can form a new instance I^* by combining the instances $I_1, \ldots, I_k$ in an independent way.

Lemma 5.3.6. *If a maximization problem $\mathcal{O}$ is product composable and cannot be approximated within a constant $c < 1$ in polynomial time, then there exists no polynomial-time $2^{-O(n^{1-\epsilon})}$-approximation algorithm for $\mathcal{O}$, where n is the size of the instance and $\epsilon > 0$.*

Proof. By changing the constant in the big O notation, we can rewrite the approximation factor as $2^{-O(n^{1-\epsilon})} = c^{O(n^{1-\epsilon})}$. For the sake of contradiction, assume that there is a polynomial-time $c^{O(n^{1-\epsilon})}$-approximation algorithm of $\mathcal{O}$, and that its output for instance I is given by the function $A(I) \in \mathcal{F}(I)$. For any instance I, let $Opt(I)$ denote its optimal solution. Now, consider any instance I of $\mathcal{O}$ and let $n = |I|$. Since $\mathcal{O}$ is product composable, we can construct an instance I^* containing $m = n^k$ copies of I. We get $|I^*| = N = n^{k+1}$ and by the bijection property of product composability $f(Opt(I^*)) = f(Opt(I))^m$. Let $(S_1, \ldots, S_m)$ (where each $S_i \in \mathcal{F}(I)$) be the matching tuple of $A(I^*)$ in the bijection. At least one solution in this tuple, say S_1, satisfies

$f(S_1) \geq f(A(I^*))^{1/m}$. By assumption, $f(A(I^*)) \geq c^{O(N^{1-\epsilon})} \cdot f(Opt(I^*))$. It follows that

$$f(S_1) \geq f(A(I^*))^{1/m} \geq c^{\frac{O(N^{1-\epsilon})}{m}} \cdot f(Opt(I^*))^{1/m} = c^{\frac{O(N^{1-\epsilon})}{m}} \cdot f(Opt(I))$$

Since $m = N^{\frac{k}{k+1}}$ we can choose, for any $\epsilon > 0$, a large enough k such that $m = \omega(N^{1-\epsilon})$. For such an assignment, S_1 is computable in polynomial time (in n) and $f(S_1) \geq c \cdot f(Opt(I))$. This contradicts with the premise that there is no polynomial-time c-approximation algorithm for $\mathcal{O}$. □

Although the most likely hull problem is not product composable itself, this property only needs to hold for a subproblem. The subproblem formed by the instances used in our NP-hardness reduction is product composable, as we show in the following lemma.

Lemma 5.3.7. *Let $\mathcal{I}$ be the set of most likely hull problem instances constructed from penny graph embeddings using the described construction. Then, the most likely hull problem restricted to $\mathcal{I}$ is product composable.*

Proof. Given any collection of instances $I_1, \ldots, I_k \in \mathcal{I}$, we can construct an instance $I^* \in \mathcal{I}$ satisfying the product composability conditions as follows. Let $E_1, \ldots, E_k$ be the penny graph embeddings corresponding to $I_1, \ldots, I_k$. Without loss of generality, we assume that the embeddings $E_1, \ldots, E_k$ are separated by at least a distance of 1.[19]

[19]If this is not the case, one can easily translate each E_i to form a new embedding E'_i so that E'_i is separated from the other translated embeddings E'_j. By construction, the problem instance I'_i that is

In other words, the penny graphs corresponding to $E_1, \ldots, E_k$ are disconnected. We create I^* by simply taking the union of $I_1, \ldots, I_k$, i.e., $I^* = \bigcup_i I_i$.

Let $\{s_1, \ldots, s_{|I^*|}\}$ denote the points in I^* and $\{\pi_1, \ldots, \pi_{|I^*|}\}$ be the corresponding probabilities. Notice that the probability assigned to each point s_u is the same in both its original instance (I_i) and the composed instance (I^*).

We now argue that the I^* satisfies the product composability conditions. It is easy to see that $|I^*| = \sum_{i=1}^{k} |I_i|$. Before we describe the bijection property, we need some preliminary arguments.

A feasible solution to a most likely hull problem instance I is a convex polyhedron C such that the vertices C is a subset of the points in I and C includes all permanent points in I (points probability with 1). Let C_i be a feasible solution to instance I_i, i.e., $C_i \in \mathcal{F}(I_i)$. Let V_i be its non-permanent vertices (the ones with probability strictly less than 1). The following "vertex preservation property" is crucial for our proof: the convex hull of $C_i \cup P$ for any point set P such that $P \subseteq (I^* \setminus I_i)$ contains all points in V_i as vertices as well. To see this, notice that all points of V_i on the paraboloid $\mathcal{P}_2$ are in convex position with respect to all points in I^* and thus cannot be covered by any point combination from I^*. Also, since the embeddings of $E_1, \ldots, E_k$ are separated and by Lemma 5.3.4, the remaining points of V_i (on the paraboloid $\mathcal{P}_1$) cannot be covered by

implied by E'_i has combinatorially the exact same set of feasible solutions and the optimization function values with the original instance I_i. In other words, given a feasible solution S'_i for I'_i, one can compute in polynomial time a feasible solution S_i for I_i such that $f(S_i) = f(S'_i)$ and vice versa.

any combination of points from $(I^* \setminus I_i)$. This property also implies that any point $p \in I_i$ that is outside C_i is also outside $\mathrm{conv}(C_i \cup P)$ for any $P \subseteq (I^* \setminus I_i)$.

We now describe our bijective mapping. Let C^* be a feasible solution for I^*, i.e., $C^* \in \mathcal{F}(I^*)$. Let V^* be the non-permanent vertices of C^*. Let C_i be the convex hull of the permanent points in I_i and the points in $I_i \cap V^*$. It is easy to see that all points in $I_i \cap V^*$ are the non-permanent vertices of C_i (i.e., V_i as defined earlier). We map C^* to $(C_1, \ldots, C_k)$. The vertex preservation property implies that this mapping is one-to-one. Given any tuple $(C_1, \ldots, C_k) \subseteq \mathcal{F}(I_i) \times \cdots \times \mathcal{F}(I_k)$, one can construct the corresponding C^* by simply taking the convex hull of $C_1, \ldots, C_k$.

We now show that the bijective mapping satisfies the product composability. In particular, we show that the likelihood of C^* is the product of the likelihoods of $C_1, \ldots, C_k$ (restricted to the points in $I_i, \ldots, I_k$ respectively). Let V^* be the non-permanent vertices of C^* and V_i be the non-permanent vertices of C_i as before. Similarly, let O^* be the points in I^* outside C^* and let O_i be the points in I_i outside C_i. Finally, let $\mathcal{L}^*(C^*)$ be the likelihood of C^* (with respect to points in I^*) and let $\mathcal{L}_i(C_i)$ be the likelihood of C_i (with respect to points in I_i). Making use of Lemma 5.2.1 and the vertex preservation property, we write

$$\mathcal{L}^*(C^*) = \prod_{s_u \in V^*} \pi_u \times \prod_{s_u \in O^*} \overline{\pi_u}$$

$$
\begin{aligned}
&= \prod_{1 \le i \le k} \prod_{s_u \in V_i} \pi_u \times \prod_{1 \le i \le k} \prod_{s_u \in O_i} \overline{\pi_u} \\
&= \prod_{1 \le i \le k} \left(\prod_{s_u \in V_i} \pi_u \times \prod_{s_u \in O_i} \overline{\pi_u} \right) \\
&= \prod_{1 \le i \le k} \mathcal{L}_i(C_i)
\end{aligned}
$$

Finally, it is easy to see that the bijective mapping is computable in polynomial time. This completes the proof. □

Corollary 5.3.8. *For any $\epsilon > 0$, there exists no polynomial-time $2^{-O(n^{1-\epsilon})}$-approximation algorithm for the most likely hull problem in three dimensions, unless P=NP.*

Finally we observe that one can trivially achieve a 2^{-n}-approximation of the most likely hull problem as follows: simply take the convex hull of all sites with probability at least $\frac{1}{2}$. If $\pi_i < \frac{1}{2}$ for all i, then the convex hull is empty.

5.4 Most Likely Hull in the Multipoint Model

In this section, we show that computing the most likely hull in the multipoint model is NP–hard even for two dimensions. For completeness, we begin with some preliminary information about the most likely hull definition under the multipoint model.

In the multipoint model, the ith point of the input is described by a pair (S_i, Π_i), where $S_i = \{s_i^1, \ldots, s_i^{|S_i|}\}$, $\Pi_i = \{\pi_i^1, \ldots, \pi_i^{|\Pi_i|}\}$ and $|S_i| = |\Pi_i|$. The interpretation is that the ith point appears at the position s_i^j with probability π_i^j, for $j = 1, 2, \ldots, |S_i|$.

If the sum of probabilities for the ith point (i.e., $\sum_{1 \leq j \leq |S_i|} \pi_i^j$) is not 1 (in which case it is strictly less than 1), then it is possible that the ith point does not appear at all in a probabilistic experiment.

We use S to denote the set of all sites, i.e., $S = \{s_i^j\}$ and set $n = S$. For a subset $A \subseteq S$, we denote the probability that A is the outcome of a probabilistic experiment by $\pi(A)$. Similarly to the unipoint model, the definition of $\pi(A)$ involves a product of existence probabilities for all sites in A. The sites that are not in A, however, contribute to $\pi(A)$ in a different way. Specifically, let s_i^j be a site that is not in A. If A contains another $s_i^{j'}$ site from the ith point, then the non-existence probability of s_i^j is irrelevant to $\pi(A)$, because existence of $s_i^{j'}$ already implies non-existence of s_i^j. If there is no such site $s_i^{j'}$, then no site from the tuple of the ith point is in A. In that case, we just consider the probability that ith point does not exist at all, which is $1 - \sum_{1 \leq j \leq |S_i|} \pi_i^j$. Finally, notice that if A contains two sites from the same uncertain point, then it cannot be the outcome of an experiment. This implies the following definition for $\pi(A)$:

$$\pi(A) = \begin{cases} 0 & \text{if there are two distinct sites } s_i^j \text{ and } s_i^{j'} \text{ in } A \\ \prod_{s_i^j \in A} \pi_i^j \times \prod_{i \,|\, \nexists j . s_i^j \in A} \left(1 - \sum_{1 \leq j \leq |S_i|} \pi_i^j\right) & \text{otherwise} \end{cases}$$

The definition for the most likely hull follows from $\pi(A)$ as in the unipoint model case. That is, the most likely convex hull is the polytope C which maximizes the likelihood function $\mathcal{L}(C)$, which is defined as

$$\mathcal{L}(C) \;=\; \Pr\big[\; \mathrm{conv}(A) \equiv C \;\big] \;=\; \sum_{\substack{A \subseteq S \\ \mathrm{conv}(A) \equiv C}} \pi(A)$$

5.4.1 The Reduction

Our NP-hardness proof uses a reduction from NP-hard problem 3-SAT [42]. The 3-SAT problem takes as input a conjunctive normal form boolean formula where each clause contains 3 variables and asks if this formula is satisfiable. Consider a 3-SAT instance (V, U) where V is the set of the variables and U is the set of clauses. We first construct $6|U|$ points on the unit circle. We call these points the *anchors* and use them as permanent points (i.e., points with probability 1) in our hull problem instance. Between each pair of consecutive anchors, we place a single point on the unit circle that we call a *spike*. (See Figure 5.6a.) We assign an independent existence probability of $\frac{1}{2}$ to each spike. As we will explain shortly, the main idea of our construction is that the most likely hull includes all spikes in its interior if and only if the 3-SAT instance is satisfiable.

For each variable v, we construct two additional sets of points, one corresponding to the case that v is true and one corresponding to the case that v is false. In particular, for each clause u that v appears in positive form, we construct a point p_v^u covering a

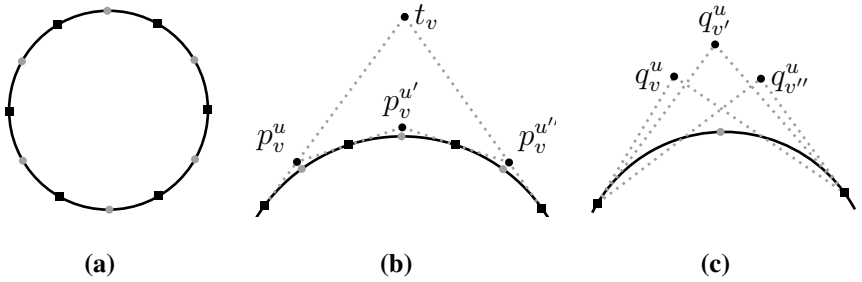


Figure 5.6: (a) Anchors (black squares) and spikes (gray circles) on the unit circle. (b) Construction of t_v. (c) The three points constructed for clause u.

single spike, at the intersection of the lines tangent to the unit circle at the two anchors next to the spike. We assign each p_v^u a probability of $\frac{1}{2}$ but this probability is dependent, as we will put p_u^v; we will put more sites for the uncertain point that hosts p_u^v in the rest of the construction. We construct all points p_u^v for a single variable v over a consecutive sequence of spikes, and then put a single point t_v covering the constructed points. (See Figure 5.6b.)

We apply the same construction for all clauses that v appears in negated form. This creates an additional set of points p_u^v, all of which we cover with a single point f_v as we did for t_v. We associate t_v and f_v with the same uncertain point and assign each a probability of $\frac{1}{2}$. That is, in a probabilistic experiment, either t_v or f_v is present (with equal probability), but not both. Existence of t_v is meant to imply that v is assigned true, whereas the existence of f_v is meant to imply that v is assigned false.

Finally, for each clause u, we construct three additional points covering a single spike. These points are constructed in such a way that: (1) they do not cover any other spike, and (2) they are in convex position with respect to each other and the two anchors next to the covered spike. Each of these points corresponds to a distinct variable v that appears in the clause. We denote the point associated with variable v by q_v^u. (See Figure 5.6c.) We assign each q_v^u to the uncertain point that the previously constructed point p_v^u was assigned to and set its probability to $\frac{1}{2}$. That is, in an experiment, either q_v^u or p_v^u exists (with equal probability), but not both.

Lemma 5.4.1. *The most likely hull has likelihood* $(1/2)^{3|U|+|V|}$ *if and only if it contains all spikes in its interior. Otherwise, its likelihood is at most* $(1/2)^{3|U|+|V|+1}$.

Proof. Let C be the most likely hull. For ease of reference, let us say that the outcome A of a probabilistic experiment is compatible with C if $\mathrm{conv}(A) = C$. Notice that all experiment outcomes A compatible with C contain a particular configuration of the dependent point pairs. In particular, if C contains a point t_v as a vertex, then all compatible outcomes contain t_v and not f_v. Otherwise, all compatible outcomes contain f_v and not t_v. Similarly, if C contains q_v^u as a vertex, then all compatible outcomes contain q_v^u or p_v^u otherwise. The probability that these configurations exists in the outcome of an experiment is $(1/2)^{3|U|+|V|}$ because there are $3|U| + |V|$ dependent point pairs. If C contains all spikes in its interior, then the existence of spikes are irrelevant to the likelihood of C, thus $\mathcal{L}(C) = (1/2)^{3|U|+|V|}$. Otherwise, compatibility with C is also

conditioned on either existence or non-existence of at least one spike. This implies $\mathcal{L}(C) \leq (1/2)^{3|U|+|V|+1}$. □

We now describe how the satisfiability of the 3-SAT instance relates to our construction. We first give a high-level idea and then give the formal proof in a lemma. Consider a variable v. Notice that, if the most likely hull covers all spikes below t_v, then either t_v or all points p_v^u below t_v appears in the hull as a vertex. If t_v appears in the hull, then the hull can pass through the points q_v^u (which are in the same probabilistic tuples with points p_v^u), and cover spikes representing the clauses that v appears in positive form. This corresponds to the case that v is assigned true and all corresponding clauses are satisfied. Similar notion also applies to f_v and the clauses that v appears in negated form. If all spikes are covered, then all clauses are satisfied and so is the 3-SAT instance. Combining this idea with Lemma 5.4.1, we deduce the following lemma.

Lemma 5.4.2. *The 3-SAT instance is satisfiable if and only if the most likely hull has likelihood* $(1/2)^{3|U|+|V|}$.

Proof. We first show that if the most likely hull has likelihood $(1/2)^{3|U|+|V|}$ then the 3-SAT instance is satisfiable. Let C be the most likely hull with likelihood $(1/2)^{3|U|+|V|}$. By construction, C contains exactly one of the sites t_v and f_v as a vertex for each variable v. Consider the boolean assignment where we assign the variable v to true if t_v is a vertex, and to false if f_v is a vertex. We now argue that this assignment satisfies

all clauses in the 3-SAT instance. Take any clause u. By Lemma 5.4.1, C contains all spikes in its interior. Consequently at least one point q_v^u is a vertex of C. Then, the dependent point p_v^u is not a vertex of C. By construction, C covers the underlying spike with t_v if v appears in positive form in u or with f_v if v appears in negated form. This implies that u is satisfied by the assignment of v.

We now prove the converse. Suppose that there is a satisfying variable assignment for the 3-SAT instance. We construct a subset Q of points as follows. We insert to Q t_v if v is assigned true and f_v if v is assigned false. Additionally, for each variable-clause pair (v, u) we insert q_v^u if v is a satisfying variable for u or p_v^u otherwise. Finally, we insert all anchor points. Observe that Q is a valid outcome of a probabilistic experiment. We now argue that the convex hull of Q covers all spikes and thus has likelihood $(1/2)^{3|U|+|V|}$ by Lemma 5.4.1. The spikes under all points $p_v^u \in Q$ are trivially covered. For each point $p_v^u \notin Q$, v is a satisfying variable for u, and thus the spike under p_v^u is covered by either t_v or f_v (whichever is the one above p_v^u). Finally, since all clauses are satisfied, each spike under a triplet of points q_v^u, $q_{v'}^u$ and $q_{v''}^u$ are also covered (at least by one of them). This completes the proof. □

Easily following from Lemma 5.4.2, we state the following theorem.

Theorem 5.4.3. *Computing the most likely hull in the multipoint model is NP-hard.*

Lemma 5.4.1 in fact implies a stronger result: It is NP-hard to compute the likelihood of the most likely hull within any factor $c > \frac{1}{2}$. By construction, the problem instances that we create are product composable. Then, by Lemma 5.3.6, we can state the following theorem.

Theorem 5.4.4. *For any $\epsilon > 0$, there exists no polynomial-time $2^{-O(n^{1-\epsilon})}$-approximation algorithm for the most likely hull problem in the multipoint model unless P=NP.*

5.5 Extensions and Concluding Remarks

Algorithms for computing or estimating succinct summary hulls are a useful tool in the analysis of uncertain geometric data. While we focused exclusively on the Most Likely Hull, our techniques are applicable to several other ways of defining the "best" hull. Any useful definition of the likely hull must include a penalty function for misclassifying points, *both false positives and false negatives*. If only false negatives (points outside the hull) are penalized, then the convex hull of *all* the points has the best score. Our dynamic programming algorithm for the unipoint model in 2 dimensions can be extended for several natural scoring functions. Although entries may need to be computed differently, the subproblem structure utilized by the dynamic programming algorithm also applies to these other settings.

For instance, one simple scoring function measures the *agreement* on the "in" and "out" classification. A convex hull C splits the point set into two parts: inside and outside. We can measure the "quality" $Q(C)$ of a hull C by its expected agreement with a random hull's classification: the number of points of S whose classification (in or out) is the same for both C and the hull of a random outcome. Both our dynamic programming algorithm for computing the hull in 2 dimensions, and the hardness in 3 dimensions, under the unipoint model carry over to this "Symmetric Difference Hull" definition. Similarly, another scoring function for measuring the fraction of points correctly classified counts the number of points in the random outcome that lie in C plus the number of non-sample points that lie outside C. Our results for the unipoint model hold for this type of scoring as well.

In summary, we believe that the study of geometric structures over probabilistic data is a fundamental problem, and our results are only a first, but promising, step.

Conclusion

In this dissertation, we have made contributions to the computational study of two geometric concepts: volumes and convex hulls. The focus of our work was on variations of Klee's Measure Problem and convex hulls under uncertainty. We developed efficient algorithms and showed hardness results for both type of problems. Nevertheless, many problems still remain open and both concepts are likely to be a subject of substantial future research.

In Chapter 1, we studied the "grounded" case of Klee's Measure Problem. We showed that improvements on the general upper bound are possible as long as the input boxes are k-grounded for $k \geq 2$. Our algorithm also improved the bound for the hypervolume indicator problem for certain dimensions. There is currently an active research on Klee's problem for both its general form and its special cases in the computational geometry community. Therefore, future research is likely to bring even more improvements.

In Chapter 2, we investigated Klee's problem on uncertain boxes. We showed that the expected volume of uncertain boxes is computable in polynomial time despite the fact that its probability distribution is NP-hard to compute. We also developed an efficient data structure to maintain the expected volume of a set of boxes that undergoes insertions and deletions. However, our structure has a significant space usage and the problem of maintaining the volume with a near-linear space data structure is still open.

In Chapter 3, we developed an efficient dynamic data structure for a discrete version of Klee's problem. The structure also allowed reporting queries and the maintenance of an uncertain measure. All our update times are sublinear, however, they are significantly better for point updates than for box updates. It is not yet known if there is a tradeoff between point and box updates, and if one can design a structure with better box update times while keeping point update costs acceptable.

In Chapter 4, we investigated convex hulls under uncertainty. To this extent, we examine two probabilistic models: one has only existential uncertainty, while the other couples it with locational uncertainty. For both models, we described efficient algorithms to compute the convex hull membership probability of a given query point. We also introduced the probability map structure and showed how to compute it efficiently in two dimensions. On the other hand, it remains an open problem to efficiently compute the membership probability for degenerate point sets in high dimensions.

In Chapter 5, we studied the most likely convex hull, the mode of the convex hull variable, under both probabilistic models from Chapter 4. We described a cubic-time dynamic programming algorithm to compute the most likely hull in two dimensions when only existential uncertainty is present. We also showed that it is NP-hard to compute the most likely hull in higher dimensions or in the presence of locational uncertainty and gave an inapproximability result. Our results are a promising step towards understanding the complexity of geometric structures over uncertain data and future research on uncertain data is likely to produce many more interesting results.

Bibliography

[1] P. Afshani, P. K. Agarwal, L. Arge, K. G. Larsen, and J. M. Phillips. (Approximate) uncertain skylines. *Theory of Computing Systems*, 52(3):342–366, 2013.

[2] P. K. Agarwal. An improved algorithm for computing the volume of the union of cubes. In *Proceedings of the 26th Annual Symposium on Computational Geometry*, pages 230–239, 2010.

[3] P. K. Agarwal, S.-W. Cheng, Y. Tao, and K. Yi. Indexing uncertain data. In *Proceedings of the ACM Symposium on Principles of Database Systems*, pages 137–146, 2009.

[4] P. K. Agarwal, S.-W. Cheng, and K. Yi. Range searching on uncertain data. *ACM Transactions on Algorithms*, 8(4):43, 2012.

[5] P. K. Agarwal, H. Kaplan, and M. Sharir. Computing the volume of the union of cubes. In *Proceedings of the 23rd Annual Symposium on Computational Geometry*, pages 294–301, 2007.

[6] C. C. Aggarwal. *Managing and Mining Uncertain Data*. Springer-Verlag, 2009.

[7] C. C. Aggarwal and P. S. Yu. A survey of uncertain data algorithms and applications. *IEEE Transactions on Knowledge and Data Engineering*, 21(5):609–623, 2009.

[8] D. Avis, B. K. Bhattacharya, and H. Imai. Computing the volume of the union of spheres. *The Visual Computer*, 3(6):323–328, 1988.

[9] D. Avis and D. Rappaport. Computing the largest empty convex subset of a set of points. In *Proceedings of the 1st Annual Symposium on Computational Geometry*, pages 161–167, 1985.

[10] C. Bautista-Santiago, J. M. Díaz-Báñez, D. Lara, P. Pérez-Lantero, J. Urrutia, and I. Ventura. Computing optimal islands. *Operations Research Letters*, 39(4):246–251, 2011.

[11] M. Ben-Or. Lower bounds for algebraic computation trees. In *Proceedings of the 15th Annual ACM Symposium on Theory of Computing*, pages 80–86, 1983.

[12] J. L. Bentley. Solutions to Klee's rectangle problems. *Unpublished manuscript, Department of Computer Science, Carnegie Mellon University, Pittsburgh PA*, 1977.

[13] G. Beskales, M. A. Soliman, and I. F. Ilyas. Efficient search for the top-k probable nearest neighbors in uncertain databases. *Proceedings of the VLDB Endowment*, 1(1):326–339, 2008.

[14] N. Beume, C. Fonseca, M. López-Ibáñez, L. Paquete, and J. Vahrenhold. On the complexity of computing the hypervolume indicator. *IEEE Transactions on Evolutionary Computation*, 13(5):1075–1082, 2009.

[15] K. Bringmann. Klee's measure problem on fat boxes in time $O(n^{(d+2)/3})$. In *Proceedings of the 26th Annual Symposium on Computational Geometry*, pages 222–229, 2010.

[16] K. Bringmann. Bringing order to special cases of Klee's measure problem. In *Proceedings of 39th Intenational Symposium on Mathematical Foundations of Computer Science*, volume 8087 of *Lecture Notes in Computer Science*, pages 207–218. 2013.

[17] K. Bringmann and T. Friedrich. Approximating the volume of unions and intersections of high-dimensional geometric objects. In *Proceedings of 19th International Symposium on Algorithms and Computation*, pages 436–447, 2008.

[18] S. Cabello. Approximation algorithms for spreading points. *Journal of Algorithms*, 62(2):49–73, 2007.

[19] S. Cabello and M. J. van Kreveld. Approximation algorithms for aligning points. In *Proceedings of the 19th ACM Symposium on Computational Geometry*, pages 20–28, 2003.

[20] M. R. Cerioli, L. Faria, T. O. Ferreira, and F. Protti. On minimum clique partition and maximum independent set on unit disk graphs and penny graphs: Complexity and approximation. *Electronic Notes in Discrete Mathematics*, 18:73–79, 2004.

[21] T. M. Chan. Optimal output-sensitive convex hull algorithms in two and three dimensions. *Discrete & Computational Geometry*, 16:361–368, 1996.

[22] T. M. Chan. Semi-online maintenance of geometric optima and measures. In *Proceedings of the 13th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 474–483, 2002.

[23] T. M. Chan. A (slightly) faster algorithm for Klee's measure problem. *Computational Geometry*, 43(3):243–250, 2010.

[24] T. M. Chan. Klee's measure problem made easy. In *Proceedings of the 54th Annual Symposium of Foundations of Computer Science (FOCS)*, pages 410–419, 2013.

[25] B. Chazelle. An optimal convex hull algorithm in any fixed dimension. *Discrete & Computational Geometry*, 10:377–409, 1993.

[26] B. Chazelle and J. Matoušek. Derandomizing an output-sensitive convex hull algorithm in three dimensions. *Computational Geometry*, 5(1):27–32, 1995.

[27] E. Y. Chen and T. M. Chan. Space-efficient algorithms for Klee's measure problem. In *17th Canadian Conference on Computational Geometry (CCCG)*, 2005.

[28] R. Cheng, J. Chen, M. Mokbel, and C. Chow. Probabilistic verifiers: Evaluating constrained nearest-neighbor queries over uncertain data. In *Proceedings of the 24th International Conference on Data Engineering*, pages 973–982, 2008.

[29] R. Cheng, L. Chen, J. Chen, and X. Xie. Evaluating probability threshold k-nearest-neighbor queries over uncertain data. In *Proceedings of the 12th International Conference on Extending Database Technology*, pages 672–683, 2009.

[30] S. W. Cheng and R. Janardan. Efficient maintenance of the union intervals on a line, with applications. In *Proceedings of the ACM Symposium on Discrete Algorithms*, pages 74–83, 1990.

[31] L. Chew, D. Dor, A. Efrat, and K. Kedem. Geometric pattern matching in d-dimensional space. *Discrete and Computational Geometry*, 21(2):257–274, 1999.

[32] V. Chvátal and G. Klincsek. Finding largest convex subsets. *Congresus Numeratium*, 29:453–460, 1980.

[33] G. Cormode and A. McGregor. Approximation algorithms for clustering uncertain data. In *Proceedings of the 27th ACM Symposium on Principles Database Systems*, pages 191–200, 2008.

[34] N. N. Dalvi, C. Ré, and D. Suciu. Probabilistic databases: Diamonds in the dirt. *Communications of the ACM*, 52(7):86–94, 2009.

[35] M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars. *Computational geometry: Algorithms and applications*. Springer-Verlag New York Inc, 2008.

[36] H. Edelsbrunner. Dynamic data structures for orthogonal intersection queries. *Report F59, Institut für Inform., TU Graz*, 1980.

[37] H. Edelsbrunner, J. O'Rourke, and R. Seidel. Constructing arrangements of lines and hyperplanes with applications. In *24th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 83–91, 1983.

[38] D. Eppstein, M. Overmars, G. Rote, and G. Woeginger. Finding minimum area k-gons. *Discrete & Computational Geometry*, 7(1):45–58, 1992.

[39] M. Fleischer. The measure of pareto optima. Applications to multi-objective metaheuristics. In *Proceedings of the 2nd International Conference on Evolutionary Multi-Criterion Optimization*, pages 519–533, 2003.

[40] C. Fonseca, L. Paquete, and M. López-Ibáñez. An improved dimension-sweep algorithm for the hypervolume indicator. In *IEEE Congress on Evolutionary Computation*, pages 1157–1163, 2006.

[41] M. Fredman and B. Weide. On the complexity of computing the measure of $\cup[a_i, b_i]$. *Communications of the ACM*, 21(7):540–544, 1978.

[42] M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-completeness*. 1979.

[43] W. M. Getz, S. Fortmann-Roe, P. C. Cross, A. J. Lyons, S. J. Ryan, and C. C. Wilmers. Locoh: Nonparameteric kernel methods for constructing home ranges and utilization distributions. *PLOS ONE*, 2(2), 02 2007.

[44] W. M. Getz and C. C. Wilmers. A local nearest-neighbor convex-hull construction of home ranges and utilization distributions. *Ecography*, 27(4):489–505, 2004.

[45] G. H. Gonnet, J. I. Munro, and D. Wood. Direct dynamic structures for some line segment problems. *Computer Vision, Graphics, and Image Processing*, 23(2):178–186, 1983.

[46] R. L. Graham. An efficient algorithm for determining the convex hull of a finite planar set. *Information Processing Letters*, 1(4):132–133, 1972.

[47] S. Guha and K. Munagala. Exceeding expectations and clustering uncertain data. In *Proceedings of the 28th ACM Symposium on Principles Database Systems*, pages 269–278, 2009.

[48] S. Huband, P. Hingston, L. While, and L. Barone. An evolution strategy with probabilistic mutation for multi-objective optimisation. In *The 2003 Congress on Evolutionary Computation*, volume 4, pages 2284–2291, 2003.

[49] A. Jørgensen, M. Löffler, and J. M. Phillips. Geometric computations on indecisive and uncertain points. *arXiv:1205.0273*, `http://arxiv.org/abs/1205.0273`, 2012.

[50] P. Kamousi, T. M. Chan, and S. Suri. Closest pair and the post office problem for stochastic points. In *Proceedings of the 12th Algorithms and Data Structures Symposium*, pages 548–559, 2011.

[51] K. Kanth and A. Singh. Optimal dynamic range searching in non-replicating index structures. In *Proceedings of the 7th International Conference on Database Theory*, pages 257–276. Springer, 1999.

[52] H. Kaplan, N. Rubin, M. Sharir, and E. Verbin. Counting colors in boxes. In *Proceedings of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 785–794, 2007.

[53] L. Kettner, K. Mehlhorn, S. Pion, S. Schirra, and C.-K. Yap. Classroom examples of robustness problems in geometric computations. *Computational Geometry*, 40(1):61–78, 2008.

[54] D. G. Kirkpatrick. Optimal search in planar subdivisions. *SIAM Journal on Computing*, 12(1):28–35, 1983.

[55] D. G. Kirkpatrick and R. Seidel. The ultimate planar convex hull algorithm? *SIAM Journal of Computing*, 15(1):287–299, 1986.

[56] V. Klee. Can the measure of $\cup_1^n [a_i, b_i]$ be computed in less than O($n \log n$) steps? *American Mathematical Monthly*, 84(4):284–285, 1977.

[57] W. Lipski and F. Preparata. Finding the contour of a union of iso-oriented rectangles. *Journal of Algorithms*, 1(3):235–246, 1980.

[58] M. Löffler. *Data Imprecision in Computational Geometry*. PhD thesis, Utrecht University, 2009.

[59] M. Löffler and M. van Kreveld. Largest and smallest convex hulls for imprecise points. *Algorithmica*, 56:235–269, 2010.

[60] M. Löffler and M. J. van Kreveld. Largest bounding box, smallest diameter, and related problems on imprecise points. *Computational Geometry*, 43(4):419–433, 2010.

[61] J. Matoušek. Linear optimization queries. *Journal of Algorithms*, 14(3):432–448, 1993.

[62] M. Overmars. *The design of dynamic data structures*. Springer, 1983.

[63] M. Overmars and J. van Leeuwen. Worst-case optimal insertion and deletion methods for decomposable searching problems. *Information Processing Letters*, 12(4):168–173, 1981.

[64] M. H. Overmars and C.-K. Yap. New upper bounds in Klee's measure problem. *SIAM Journal on Computing*, 20(6):1034–1045, 1991.

[65] F. P. Preparata and S. J. Hong. Convex hulls of finite sets of points in two and three dimensions. *Communications of the ACM*, 20(2):87–93, 1977.

[66] D. Salesin, J. Stolfi, and L. J. Guibas. Epsilon geometry: Building robust algorithms from imprecise computations. In *Proceedings of the 5th Symposium on Computational Geometry*, pages 208–217, 1989.

[67] R. Seidel. Constructing higher-dimensional convex hulls at logarithmic cost per face. In *Proceedings of the 18th Annual ACM Symposium on Theory of Computing*, pages 404–413, 1986.

[68] G. Sharma, C. Busch, R. Vaidyanathan, S. Rai, and J. L. Trahan. An efficient transformation for Klee's measure problem in the streaming model. In *24th Canadian Conference on Computational Geometry (CCCG)*, 2012.

[69] P. G. Spirakis. Very fast algorithms for the area of the union of many circles. Technical report, Courant Institute of Mathematical Sciences, 1983.

[70] S. Suri, K. Verbeek, and H. Yıldız. On the most likely convex hull of uncertain points. In *Proceedings of the 21st European Symposium on Algorithms (ESA)*, volume 8125 of *Lecture Notes in Computer Science*, pages 791–802, 2013.

[71] J. Vahrenhold. An in-place algorithm for Klee's measure problem in two dimensions. *Information Processing Letters*, 102(4):169 – 174, 2007.

[72] V. K. Vaishnavi. Computing point enclosures. *IEEE Transactions on Computers*, 31(1):22–29, 1982.

[73] G. van den Bergen, A. Kaldewaij, and V. J. Dielissen. Maintenance of the union of intervals on a line revisited. In *Computing Science Reports*. Eindhoven University of Technology, 1998.

[74] M. van Kreveld and M. Overmars. Divided k-d trees. *Algorithmica*, 6(1):840–858, 1991.

[75] M. J. van Kreveld, M. Löffler, and J. S. B. Mitchell. Preprocessing imprecise points and splitting triangulations. *SIAM Journal on Computing*, 39(7):2990–3000, 2010.

[76] J. van Leeuwen and D. Wood. The measure problem for rectangular ranges in d-space. *Journal of Algorithms*, 2(3):282–300, 1981.

[77] C.-K. Yap and S. Pion. Special issue on robust geometric algorithms and their implementations. *Computational Geometry*, 33(1-2), 2006.

[78] H. Yıldız, L. Foschini, J. Hershberger, and S. Suri. The union of probabilistic boxes: Maintaining the volume. In *Proceedings of the 19th Annual European Symposium on Algorithms (ESA)*, 2011.

[79] H. Yıldız, J. Hershberger, and S. Suri. A discrete and dynamic version of Klee's measure problem. In *Proceedings of the 23rd Canadian Conference on Computational Geometry (CCCG)*, pages 211–216, 2011.

[80] H. Yıldız and S. Suri. On Klee's measure problem for grounded boxes. In *Proceedings of the 28th Annual Symposium on Computational Geometry (SoCG)*, pages 111–120, 2012.

[81] H. Yıldız and S. Suri. Computing Klee's measure of grounded boxes. *Algorithmica (Online)*, 2013.

[82] Z. Zhao, D. Yan, and W. Ng. A probabilistic convex hull query tool. In *Proceedings of the 15th International Conference on Extending Database Technology*, pages 570–573, 2012.

[83] E. Zitzler and S. Künzli. Indicator-based selection in multiobjective search. In *Proceedings of 8th International Conference on Parallel Problem Solving from Nature*, pages 832–842, 2004.