IN PURSUIT OF LINEAR COMPLEXITY IN DISCRETE AND COMPUTATIONAL GEOMETRY

BY

BENJAMIN ADAM RAICHEL

DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2015

Urbana, Illinois

Doctoral Committee:

Professor Sariel Har-Peled, Chair
Professor Chandra Chekuri
Dr. Kenneth Clarkson, IBM Almaden
Professor Jeff Erickson

Abstract

Many computational problems arise naturally from geometric data. In this thesis, we consider three such problems: (i) distance optimization problems over point sets, (ii) computing contour trees over simplicial meshes, and (iii) bounding the expected complexity of weighted Voronoi diagrams. While these topics are broad, here the focus is on identifying structure which implies linear (or near linear) algorithmic and descriptive complexity.

The first topic we consider is in geometric optimization. More specifically, we define a large class of distance problems, for which we provide linear time exact or approximate solutions. Roughly speaking, the class of problems facilitate either clustering together close points (i.e. netting) or throwing out outliers (i.e pruning), allowing for successively smaller summaries of the relevant information in the input. A surprising number of classical geometric optimization problems are unified under this framework, including finding the optimal k-center clustering, the kth ranked distance, the kth heaviest edge of the MST, the minimum radius ball enclosing k points, and many others. In several cases we get the first known linear time approximation algorithm for a given problem, where our approximation ratio matches that of previous work.

The second topic we investigate is contour trees, a fundamental structure in computational topology. Contour trees give a compact summary of the evolution of level sets on a mesh, and are typically used on massive data sets. Previous algorithms for computing contour trees took $\Theta(n \log n)$ time and were worst-case optimal. Here we provide an algorithm whose running time lies between $\Theta(n\alpha(n))$ and $\Theta(n \log n)$, and varies depending on the shape of the tree, where $\alpha(n)$ is the inverse Ackermann function. In particular, this is the first algorithm with $O(n\alpha(n))$ running time on instances with balanced contour trees. Our algorithmic results are complemented by lower bounds indicating that, up to a factor of $\alpha(n)$, on all instance types our algorithm performs optimally.

For the final topic, we consider the descriptive complexity of weighted Voronoi diagrams. Such diagrams have quadratic (or higher) worst-case complexity, however, as was the case for contour trees, here we push beyond worst-case analysis. A new diagram, called the candidate diagram, is introduced, which allows us to bound the complexity of weighted Voronoi diagrams arising from a particular probabilistic input model. Specifically, we assume weights are randomly permuted among fixed Voronoi sites, an assumption which is weaker than the more typical sampled locations assumption. Under this assumption, the expected complexity is shown to be near linear.

To my family, and to all those who supported me, in ways both big and small.

Acknowledgments

First and foremost, I would like to thank my advisor, Sariel Har-Peled. At an uncertain time, after having just received my bachelor's degree, Sariel helped me find a career path forward. Even though when we first met, I had little knowledge of theoretical computer science, he believed in me, and continued to do so over the years, even at times when I was uncertain of myself. Undoubtedly, without his support, this thesis, and indeed my chosen research path, would not have been possible.

I would also like to thank Seshadhri Comandur for taking me on as an intern the past two summers at Sandia National Labs, and showing me a research world outside of UIUC. In particular, the middle portion of this thesis is the result of the work we started together at Sandia.

Also, thank you to all my committee members. First, thanks to Chandra Chekuri and Jeff Erickson, from whom I have learned a great deal during my time as a grad student. Their door has always been open to me, and each has uniquely provided me with a significant amount of valuable and well-thought-out advice. Also, thank you to Ken Clarkson for graciously agreeing to serve on my committee, despite us not having worked together previously, and for twice traveling a long distance to listen to me speak.

I am also very grateful to all my co-authors. Specifically, thank you to Hsien-Chih Chang, Seshadhri Comandur, and Sariel Har-Peled who worked with me on results contained in this thesis, and also to Anne Driemel, Alina Ene, Nirman Kumar, David Mount, and Amir Nayyeri for their work with me on other results.

In my time as a graduate student I have also been fortunate enough to be a part of the UIUC theory group. The group has provided me with a sense of community right from the start. The faculty deeply care about the success of their students, and the students support one another to succeed. So thank you to the entire theory group, both past and present, for taking me in.

I would also like to thank all those who gave advice and support, both directly and indirectly, on research in and outside of this thesis. This includes, but is not limited to: Pankaj Agarwal, Janine Bennett, Sergio Cabello, Timothy Chan, Esther Ezra, Haim Kaplan, Steve LaValle, Joseph O'Rourke, Ali Pinar, Manoj Prabhakaran, Raimund Seidel, Micha Sharir, Jessica Sherette, Anastasios Sidiropoulos, Yusu Wang, and Carola Wenk. More generally, I would like to thank all the mentors I have had over the years, for pushing me to succeed and for believing in me. Also, thanks to all my classmates who struggled together with me in our quest for understanding.

Last, but certainly not least, thank you to all my friends and family. First, thank you to my parents, to whom I owe the most, and who have always supported me and pushed me to succeed in whatever path I chose in life. Next, I thank my brothers for always being there for me. We complement each other, and I am very proud of both of them. Finally, my love and thanks goes to my fiancée, Aubrey, whose love and support has been immeasurable and who I will forever be grateful to as we continue through life together.

Table of Contents

List of Figures

Chapter 1

Introduction

Understanding the world around us and the physical phenomena it contains, is the basis of much of modern science. This broad area of study typically involves producing representative data sets, either by direct physical measurements or through simulations. Specifically, these data sets consist of (or are represented by) points in low dimensional Euclidean space. Due to the strength of modern computing, producing data is "cheap", leading to massive data sets. The challenge is thus understanding the structure of this data and exploiting that structure to handle it efficiently.

Motived by this fundamental challenge, we investigate the underlying geometric properties that lead to linear time solutions and near linear complexity structures, as linear (or near linear) complexity is required when handling massive data sets. Specifically, this thesis has three main parts. In the first part, we discuss the Net and Prune technique, which gives linear time algorithms for a large and well defined class of classical geometric optimization problems. In the second part, we provide an algorithm for computing the contour tree, a fundamental structure in computational topology. While it is not always possible to compute this structure in linear time, by taking into account the shape of the tree, we provide the first algorithm which (up to a factor of the inverse Ackermann function) is instance class optimal. In the final part, we consider various weighted Voronoi diagrams which have high worst-case complexity. Here we introduce the candidate diagram and use it to prove that by assuming that weights are randomly permuted among the sites, these diagrams instead have near linear expected complexity.

The first two parts are connected by the goal of obtaining linear time algorithms. In particular, for Net and Prune, the problems covered include basic distance queries which are prerequisites for many other computational tasks. For contour trees, the need for linear time is even more immediate, as one of their primary uses is to analyze the output of scientific simulations run on supercomputers. To get this improved running time requires moving beyond worst-case analysis, and this is how the last two sections of the thesis connect. Specifically, we give the first linear time algorithm for balanced trees, and more generally relate the shape of the tree to the hardness of its computation. For weighted Voronoi diagrams, we identify a probabilistic input structure which leads to near linear expected complexity. As our analysis is via randomized incremental construction, a common technique for computing Voronoi diagrams, we get similar near linear bounds on the time to compute these weighted diagrams.

Below we provide a high level sketch of the important aspects of each part.

1.1 Geometric optimization via Net and Prune

We now overview Chapter 2, which is largely based on the paper [HR15a].

Setup. A set of n points in $\mathbb{R}^d$ determines a set of $\binom{n}{2}$ inter-point distances. Many classical geometric optimization problems, such as finding the minimum cost k-center clustering or the longest edge in the Euclidean MST, concern finding such an inter-point distance. The Net and Prune framework, provides linear time approximation algorithms for a large subset of problems which can be phrased as searching for a particular inter-point distance. This subset is defined by two key properties. First, the problem must be efficiently decidable, i.e. in linear time, given a query value, we must be able to decide whether the query is larger, smaller, or approximately equal to the optimum. Second, the problem must be "shrinkable", that is given that we know our query was too small or too large, we must be able to use that information to throw out irrelevant input points to produce a smaller subset of the data which (approximately) preserves the optimum value.

A concrete example. Consider the problem of computing the kth smallest inter-point distance, often referred to as the distance selection problem, for which the Net and Prune framework provides a linear time algorithm. In particular, when $k = 1$ this is the well studied closest pair problem, which already had a linear time solution [Rab76]. However, for larger values of k we provide the first $(1 + \varepsilon)$-approximate solution to run in $O(n/\varepsilon^d)$ time.

One can draw an analogy between this and the well known selection problem, where the goal is to find the value of rank k in an array of n values. Conceptually, our algorithm for distance selection on Euclidean points mimics the standard $\Theta(n)$ time randomized selection algorithm for an array with n values. However, in selection the goal is to find the kth ranked value in a set of n values, whereas for distance selection one works with a much larger $\Theta(n^2)$ sized (implicit) set of distances, yet our algorithm runs in $\Theta(n/\varepsilon^d)$ time.

Contributions. The main contribution of the Net and Prune framework is the identification of a simple pair of conditions which describe a large class of problems for which a linear time decision procedure can be converted into a linear time optimization procedure. These conditions are simple to check for, yet general enough to capture many previously studied geometric optimization problems, such as k-center clustering, distance selection, finding the kth longest MST edge, and more.

For many of these problems, Net and Prune provides the first known linear time algorithm. However, even for problems where previously linear time algorithms were known, Net and Prune unifies them under one framework. Conceptually, they are all described by the problem of looking for a particular inter-point distance, and they are all solvable by the same searching technique.

Additionally, the Net and Prune framework is simple, both at a conceptual level, and from an implementation standpoint (requiring only well optimized basic primitives such as hashing and counting).

Significance of linear time. For the problems handled by Net and Prune, previously $O(n \log n)$ time approximate solutions were known (specifically via well-separated pairs decompositions [CK95, Har11]). However, removing the $O(\log n)$ here is significant for several reasons. First, with our above motivation of massive data sets, an algorithm which takes longer than the time for a few linear scans of the data may be prohibitively expensive. Second, our approach is simpler than the previous $O(n \log n)$ time algorithms.

While in general one may argue that removing an $O(\log n)$ factor is not exciting, note that here we are removing the last $O(\log n)$ factor. In particular, for the problems we consider our running time is optimal. Note that more generally, problems may admit sublinear time solutions, and there has been significant success in identifying and solving such problems via sampling. However, for the problems we consider linear

time will be a lower bound. Specifically these problems are akin to searching for a needle in a haystack, for which sampling fails. As a simple example, consider finding the closest pair distance. For this problem either the removal or addition of a single point can dramatically change the value of the solution, and hence all points must be seen.

Approximation quality. The approximation quality produced by Net and Prune, will be the same as the approximation quality of the best (linear time) decision procedure. For most of the problems we discuss, this will be a $(1+\varepsilon)$-approximation, with a couple of notable exceptions. First, for k-center clustering, our algorithm yields a 2-approximation, however, such a constant approximation is necessary as k-center clustering in the plane is known to be hard to approximate within 1.8, unless $\mathsf{P} = \mathsf{NP}$ [FG88]. Second, consider the set of nearest neighbor distances (i.e. for each input point this set contains the distance to its nearest distinct input point). Surprisingly, for any rank k, we can compute the value of rank k in this set exactly in linear time. In particular, for the simple case of computing the closest pair distance, our approach matches previous results.

Expected running time. Finally, note that the Net and Prune framework provides a Las Vegas algorithm, i.e. it is always correct, but the running time is only linear in expectation. However, we also show how to turn this expected time algorithm into one which runs in linear time with polynomially high probability. Doing so is technically challenging, and while interesting from a theoretical standpoint, practically, the simpler expected linear time algorithm suffices as it quickly reduces the input to a small fraction of its initial size.

1.2 Computing contour trees in shape sensitive time

We now overview Chapter 3, which is largely based on the paper [RS14].

Setup. Consider the manifold in $\mathbb{R}^{d+1}$, induced by interpolating a height function f defined over the vertices of a simplicial mesh in $\mathbb{R}^d$. A contour of this manifold, is defined as a connected component of a level set of f at some height h. As an everyday example, a contour in a barometric pressure map may, for example, represent a storm front. The contour tree is a compact structure which tracks how contours are born, split, merge, and die as h varies, and is an indispensable tool in exploring and visualizing scientific simulation data. More specifically, the set of leaves in the contour tree are in one-to-one correspondence with the set of maxima and minima in the manifold, while internal vertices represents locations where contours merge together or split apart. Returning to the example of barometric pressure maps, intuitively when viewing the pressure map as a planar graph, the contour tree is similar to the dual graph.

Sorting. Along any maxima leaf to minima leaf path in the contour tree, vertices are in sorted height ordering. Therefore, as a natural first step, previous algorithms for computing the contour tree sorted the input, leading to a $\Theta(n \log n)$ running time. Imagine any partition of the contour tree into a set of disjoint and maximal length descending paths, P. Specifically, identify any maximal length descending path, remove it, and recur on the remaining subtrees (i.e. intuitively break the tree into the trunk, limbs, and branches). For any path $p \in P$, the vertices must appear in sorted order in P and so intuitively, $\sum_{p \in P} |p| \log |p|$, where $|p|$ is the length of p, is a lower bound on the time to compute the tree. However, observe that $n = \sum_{p \in P} |p|$,

and moreover for a balanced binary tree $\sum_{p \in P} |p| \log |p| = \Theta(n)$. Therefore, there is a gap between the cost of a global sort and our lower bound for the amount of sorting required by a specific tree structure.

Join and split Trees. There are two sister structures related to the contour tree called the join and split trees. As one varies the height h, the contour tree tracks the components of the level set (i.e. at exactly height h), while the join tree and split tree track the components of the super-level set (i.e. at h and higher) and sub-level set (i.e. at h and lower), respectively. Conceptually, since the manifold is connected, the join tree looks like a rooted tree, where the root is the global minimum (and similarly for the split tree with the global maximum). The standard algorithm for computing the contour tree first computes the join and split trees, and then interleaves them to produce the contour tree. However, we show that there are inputs for which computing the join and split trees takes $\Theta(n \log n)$ time, while the contour tree can be computed in $O(n\alpha(n))$ time, where $\alpha(n)$ is the inverse Ackermann function.

Contributions. There are two main contributions. The first is a new algorithm for computing the contour tree, which (up to a factor of $\alpha(n)$) performs optimally with respect to a given input class, and in particular is the first $O(n\alpha(n))$ time algorithm for computing balanced contour trees. This new algorithm consists of two main pieces. The first piece is a new linear time algorithm to partition the input complex into subcomplexes, such that the contour tree of each subcomplex is the same as its join tree. Naturally, given this partitioning scheme, the second (and more significant) piece is a new algorithm for computing the join tree, whose running time is (near) optimal with respect to the shape of the join tree. Additionally, both the new partitioning and the new join tree algorithm are simple, mainly consisting of several DFS calls. (To optimize the running time, the join tree algorithm further requires a data structure such as binomial heaps.)

The second contribution is a characterization of the hardness of computing the contour tree in terms of path decompositions. Specifically, given a path decomposition P (as described above), we describe a family of $\prod_{p \in P} (|p|!)$ height functions defined over a single simplicial complex such that each function has a distinct contour tree. This implies a comparison lower bound of $\Omega(C(P))$, where $C(P) = \sum_{p \in P} |p| \log |p|$. This lower bound is more input aware than the previous worst-case $\Omega(n \log n)$ bound. Finally, to prove that our algorithm is (near) optimal, we describe a particular canonical path decomposition P', and prove that our algorithm requires $\Theta(C(P'))$ comparisons. This last step involves a sophisticated charging argument, using a variant of heavy path decompositions [ST83].

1.3 Bounding probabilistic inputs for weighted diagrams

We now overview Chapter 4, which is largely based on the paper [CHR15].

Setup. Given a set of n point sites in the plane, the standard Voronoi diagram is the partition of the plane into maximal cells such that every point in a cell has the same nearest site. In other words, which cell you are contained in determines your "best site" under the Euclidean distance. However, often when determining a best site, there are other relevant features. Specifically, consider the more general setting where additionally each site is given a cost vector. For example, if the sites represent stores, then the entries in the vector might represent cost or quality of products at a given store. Here we consider weighted diagrams which take into account both the Euclidean distance as well as the cost vector when determining the best site.

Worst-case versus expected complexity. The standard Voronoi diagram in the plane has many nice features, such as $\Theta(n)$ worst-case complexity, convex cells, and straight line boundary pieces. However, weighted generalizations of this diagram are often not so well behaved. For example, the multiplicative diagram, where the distance to a site is the Euclidean distance times some (site dependent) positive weight, has $\Theta(n^2)$ worst-case complexity, disconnected cells, and curved boundary pieces. Since the worst-case complexity of weighted diagrams is prohibitively expensive, here we focus on expected complexity. Previous papers considered the expected complexity of various geometric structures under sampled site locations, however, this appears to be a stronger assumption than we require. On the other hand, significantly weaker models like smoothed analysis appear to be insufficient (in particular, the standard quadratic lower bound example for multiplicative diagrams appears stable to such perturbations). Instead, here we assume that given any set of fixed weights and site locations, that the weights are randomly permuted among the sites (for weight vectors, this assumption is coordinate-wise).

Candidate diagram. As noted above, it is not immediately clear how to use our probabilistic input assumption since weighted diagrams are difficult to reason about directly (i.e. poorly behaved cell boundaries and shapes). Therefore, instead we introduce a new and more well behaved diagram, called the candidate diagram, which is interesting in its own right. Again, suppose we are given a set of sites, where the ith site has an associated weight vector v^i. For a given location q in the plane, we say a site s_i is a candidate for q if there is no other site which is better than s_i in every way, i.e. there is no site s_j which is both closer to q and coordinate-wise $v^j \leq v^i$. The candidate diagram is then the partition of the plane into maximal cells of points with the same set of candidates.

Contributions. There are two main contributions. The first is the introduction of the candidate diagram. Specifically, consider any weighted distance function, with the natural restriction that a site s_j cannot be the nearest weighted site if there is another site s_i which is better in every way (i.e. nearest sites must be candidates). Observe that the candidate diagram is a single structure which conveys information about all such functions simultaneously. Specifically, your location in the diagram tells you all relevant sites under any distance function, i.e. as you change your preferences over time (i.e. vary your distance function), the candidate diagram remains fixed. As a consequence, given any such weighted distance function, the complexity of the candidate diagram can be used to bound the complexity of the weighted diagram. Moreover, the candidate diagram has convex cells and polygonal boundaries, making it significantly easier to reason about.

As the candidate diagram is more expressive, in general it may have high complexity (i.e. potentially higher than the weighted diagrams mentioned above). Our second contribution is showing that our probabilistic input assumption leads to an expected $O(n \operatorname{polylog} n)$ bound on the complexity of such diagrams. In the process several results are shown which are of independent interest including: (i) with high probability the expected number of staircase points when sampling from the unit hypercube does not (asymptotically) exceed its expectation, (ii) a general lemma giving high probability bounds for various instances of backwards analysis, and (iii) the expected complexity of the overlay of cells during a randomized incremental construction of the kth order Voronoi diagram is bounded by $O\big(k^4 n \log n\big)$.

1.3.1 Related work

While the presentation in this thesis on weighted diagrams largely follows that in [CHR15], the author's work on weighted Voronoi diagrams began with the paper "On the Expected Complexity of Randomly Weighted Multiplicative Voronoi Diagrams" [HR15b], discussed below.

Previous results on weighted diagrams. In [HR15b] the simpler case of scalar weights, rather than weight vectors, was considered. Therefore, for point sites in the plane, [CHR15] largely subsumes this work. However, unlike [CHR15], in [HR15b] a more general class of allowable sites was considered. This includes not only point sites, but also sites which are interior-disjoint convex sets, and more generally sites whose bisectors respect (a superset of) the conditions for abstract Voronoi diagrams [KLN09], a class defined to capture the minimal properties required for linear complexity unweighted diagrams. Specifically, for a set of n such sites, an $O(n \operatorname{polylog} n)$ bound on the expected complexity of the randomly weighted multiplicative Voronoi diagram was shown. As a consequence this yields an alternative proof to that of Agarwal *et al.* [AHKS14] of the near linear complexity of the union of randomly expanded disjoint segments or convex sets (with an improved bound on the latter).

The author has also worked on bounding the expected complexity of other types of Voronoi diagrams, such as in the paper "On the Expected Complexity of Voronoi Diagrams on Terrains" [DHR12], discussed below.

Voronoi diagrams on terrains. In [DHR12], the combinatorial complexity of geodesic Voronoi diagrams on polyhedral terrains was investigated using a probabilistic analysis. Aronov *et al.* [ABT08] proved that, if one makes certain realistic input assumptions on the terrain, this complexity is $\Theta(n + m\sqrt{n})$ in the worst case, where n denotes the number of triangles that define the terrain and m denotes the number of Voronoi sites. In [DHR12] it was shown that under a relaxed set of assumptions the Voronoi diagram has expected complexity $O(n + m)$, given that the sites are sampled uniformly at random from the domain of the terrain (or the surface of the terrain). Furthermore, a construction was presented of a terrain which implies a lower bound of $\Omega(nm^{2/3})$ on the expected worst-case complexity if these assumptions on the terrain are dropped.

1.4 Remarks

Computation model. Throughout this thesis the model of computation will be the real-RAM model. This is the standard model in Computational Geometry, and allows for storing arbitrary real numbers, along with unit cost arithmetic operations and comparisons. Typically, this model does not include a unit cost floor function (i.e. conversion between real numbers and integers), however, in Chapter 2 a unit cost floor function is assumed, as this is required for efficient grid computations. Subtleties about this assumption are discussed further in Chapter 2.

A note on dimension. In this thesis, the Euclidean dimension, d, is assumed to be a small fixed constant, and hence constant factors involving d are ignored. This assumption is required as many standard techniques used in this thesis are exponential in the dimension. However, as discussed above there are many problems related to the physical world in which this is a natural assumption.

More generally, in low dimensional settings, the set of available tools is much broader, leading to a larger class of problems which can be efficiently handled (i.e. the types of problems considered in this thesis). Indeed, even in high dimensional settings, in order to make the problem tractable, often one assumes the input has some "hidden" lower-dimensional structure.

Chapter 2

Net and Prune

2.1 Introduction

In many optimization problems one is given a geometric input and the goal is to compute some real valued function defined over it. For example, for a finite point set, the function might be the distance between the closest pair of points. For such a function, there is a search space associated with the function, encoding possible candidate values, and the function returns the minimum value over all possible solutions. For the closest pair problem, the search space is the set of all pairs of input points and their associated distances, and the target function computes the minimum pairwise distance in this space. As such, computing the value of such a function requires searching for the optimal solution, and returning its value; that is, *optimization*. Other examples of such functions include:

(A) The minimum cost of a clustering of the input. Here, the search space is the possible partitions of the data into clusters together with their associated prices.

(B) The price of a minimum spanning tree. Here, the search space is the set of all possible spanning trees of the input (and their associated prices).

(C) The radius of the smallest enclosing ball. Here, the search space is the set of all possible balls that enclose the input and their radii (i.e., it is not necessarily finite).

Naturally, many other problems can be described in this way.

It is often possible to construct a decision procedure, for such an optimization problem, which given a query value can decide whether the query is smaller or larger than the value of the function, without computing the function explicitly. Naturally, one would like to use this decider to compute the function itself by (say) performing a binary search, to compute the minimum value which is still feasible (which is the value of the function). However, often this is inherently not possible as the set of possible query values is a real interval (which is not finite). Instead, one first identifies a finite set of critical values which must contain the optimum value, and then searches over these values.

Naturally, searching over these values directly can be costly as often the number of such critical values is much larger than the desired running time. Instead one attempts to perform an implicit search.

One of the most powerful techniques to solve optimization problems efficiently in Computational Geometry, using such an implicit search, is the ***parametric search*** technique of Megiddo [Meg83]. It is relatively complicated, as it involves implicitly extracting values from a simulation of a parallel decision procedure (often a parallel sorting algorithm). For this reason it is inherently not possible for parametric search to lead to algorithms which run faster than $O(n \log n)$ time. Nevertheless, it is widely used in designing efficient geometric optimization algorithms, see [AST94, Sal97]. Luckily, in many cases one can replace parametric search by simpler techniques (see prune-and-search below for example) and in particular, it can be replaced

by randomization, see the work by van Oostrum and Veltkamp [vOV04]. Another example of replacing parametric search by randomization is the new simplified algorithm for the Fréchet distance [HR11]. Surprisingly, sometimes these alternative techniques can actually lead to linear time algorithms.

Linear time algorithms. There seems to be three main ways to get linear time algorithms for geometric optimization problems (exact or approximate):

(A) **Coreset/sketch**. One can quickly extract a compact sketch of the input that contains the desired quantity (either exactly or approximately). As an easy example, consider the problem of computing the axis parallel bounding box of a set of points - an easy linear scan suffices. There is by now a robust theory of what quantities one can extract a coreset of small size for, such that one can do the (approximate) calculation on the coreset, where usually the coreset size depends only on the desired approximation quality. This leads to many linear time algorithms, from shape fitting [AHV04], to $(1+\varepsilon)$-approximate k-center/median/mean clustering [Har01, Har04a, HM04, HK07] in constant dimension, and many other problems [AHV05]. The running times of the resulting algorithms are usually $O(n + \text{func}(\text{sketch size}))$. The limitation of this technique is that there are problems for which there is no small sketch, from clustering when the number of clusters is large, to problems where there is no sketch at all [Har04b] – for example, for finding the closest pair of points one needs all the given input and no sketching is possible.

(B) **Prune and search**. Here one prunes away a constant fraction of the input, and continues the search recursively on the remaining input. The paramount example of such an algorithm is the linear time median finding algorithm, however, one can interpret many of the randomized algorithms in Computational Geometry as pruning algorithms [Cla88, CS89, Mul94]. For example, linear programming in constant dimension in linear time [Meg84], and its extension to LP-type problems [SW92, MSW96]. Intuitively, LP-type problems include low dimensional convex programming (a standard example is the smallest enclosing ball of a point set in constant dimension). However, surprisingly, such problems also include problems that are not convex in nature – for example, deciding if a set of (axis parallel) rectangles can be pierced by three points is an LP-type problem. Other examples of prune-and-search algorithms that work in linear time include (i) computing an ear in a triangulation of a polygon [EET93], (ii) searching in sorted matrices [FJ84], and (iii) ham-sandwich cuts in two dimensions [LMS94]. Of course, there are many other examples of using prune and search with running time that is super-linear.

(C) **Grids.** Rabin [Rab76] used randomization, the floor function, and hashing to compute the closest pair of a set of points in the plane, in expected linear time. Golin *et al.* [GRSS95] presented a simplified version of this algorithm, and Smid provides a survey of algorithms on closest pair problems [Smi00]. A prune and search variant of this algorithm was suggested by Khuller and Matias [KM95]. Of course, the usage of grids and hashing to perform approximate point-location is quite common in practice. By itself, this is already sufficient to break lower bounds in the comparison model, for example for k-center clustering [Har04a]. The only direct extension of Rabin's algorithm the author is aware of is the work by Har-Peled and Mazumdar [HM05] showing a linear time 2-approximation to the smallest ball containing k points (out of n given points).

There is some skepticism of algorithms using the floor function, since Schönhage [Sch79] showed how to solve a PSPACE complete problem, in polynomial time, using the floor function in the real RAM model – the trick being packing many numbers into a single word (which can be arbitrarily long

in the RAM model, and still each operation on it takes only constant time). Note, that as Rabin's algorithm does not do any such packing of numbers (i.e., its computation model is considerably more restricted), this criticism does not seem to be relevant in the case of this algorithm and its relatives.

In this chapter, we present a new technique that combines together all of the above techniques to yield linear time approximation algorithms.

Nets. Given a point set P, an r-net $\mathcal{N}$ of P is a subset of P that represents well the structure of P in resolution r. Formally, we require that for any point in P there is a net point in distance at most r from it, and no two net points are closer than r to each other, see Section 2.2.2 for a formal definition. Thus, nets provide a sketch of the point set for distances that are r or larger. Nets are a useful tool in presenting point sets hierarchically. In particular, computing nets of different resolutions and linking between different levels, leads to a tree like data-structure that can be used to facilitate many tasks, see for example the net-tree [KL04, HM06] for such a data-structure for doubling metrics. Nets can be defined in any metric space, but in Euclidean space a grid can sometimes provide an equivalent representation. In particular, net-trees can be interpreted as an extension of (compressed) quadtrees to more abstract settings.

Computing nets is closely related to k-center clustering. Specifically, Gonzalez [Gon85] shows how to compute an approximate net that has k points in $O(nk)$ time, which is also a 2-approximation to the k-center clustering. This was later improved to $O(n)$ time, for low dimensional Euclidean space [Har04a], if k is sufficiently small (using grids and hashing). Har-Peled and Mendel showed how to preprocess a point set in a metric space with constant doubling dimension, in $O(n \log n)$ time, such that an (approximate) r-net can be extracted in (roughly) linear time in the size of the net.

Our contribution

We consider problems of the following form: Given a set P of weighted points in $\mathbb{R}^d$, one wishes to solve an optimization problem whose solution is one of the pairwise distances of P (or "close" to one of these values). Problems of this kind include computing the optimal k-center clustering, or the length of the kth edge in the MST of P, and many others. Specifically, we are interested in problems for which there is a fast approximate decider. That is, given a value $r > 0$ we can, in linear time, decide if the desired value is (approximately) smaller than r or larger than r. The goal is then to use this decider to approximate the optimum solution in linear time. As a first step towards a linear time approximation algorithm for such problems, we point out that one can compute nets in linear time in $\mathbb{R}^d$, see Section 2.2.2.

However, even if we could implicitly search over the critical values (which we cannot) then we still would require a logarithmic number of calls to the decider which would yield a running time of $O(n \log n)$, as the number of critical values is at least linear (and usually polynomial). So instead we use the return values of the decision procedure as we search to thin out the data so that future calls to the decision procedure become successively cheaper. However, we still cannot search over the critical values (since there are too many of them) and so we also introduce random sampling in order to overcome this.

Outline of the new technique. The new algorithm works by randomly sampling a point and computing the distance to its nearest neighbor. Let this distance be r. Next, we use the decision procedure to decide if we are in one of the following three cases.

(A) **Net.** If r is too small then we zoom out to a resolution of r by computing an r-net and continuing the computation on the net instead of on the original point set. That is, we net the point set into a smaller point set, such that one can solve the original problem (approximately) on this smaller sketch of the input.

(B) **Prune.** If r is too large then we remove all points whose nearest neighbor is further than r away (of course, this implies we should only consider problems for which such pruning does not affect the solution). That is, we isolate the optimal solution by pruning away irrelevant data – this is similar in nature to what is being done by prune-and-search algorithms.

(C) **Done.** The value of r is the desired approximation.

We then continue recursively on the remaining data. In either case, the number of points being handled (in expectation) goes down by a constant factor and thus the overall expected running time is linear. In particular, getting this constant factor decrease is the reason we chose to sample from the set of nearest neighbor distances rather than from the set of all inter-point distances.

Significance of results. Our basic framework is presented in a general enough manner to cover, and in many cases greatly simplify, many problems for which linear time algorithms had already been discovered. At the same time the framework provides new linear time algorithms for a large collection of problems, for which previously no linear time algorithm was known. The framework should also lead to algorithms for many other problems which are not mentioned.

At a conceptual level the basic algorithm is simple enough (with its basic building blocks already having efficient implementations) to be highly practical from an implementation standpoint. Perhaps more importantly, with increasing shifts toward large data sets, algorithms with super linear running time can be impractical. Additionally, our framework seems amenable to distributed implementation in frameworks like MapReduce. Indeed, every iteration of our algorithm breaks the data into grid cells, a step that is similar to the map phase. In addition, the aggressive thinning of the data by the algorithm guarantees that after the first few iterations the algorithm is resigned to working on only a tiny fraction of the data.

Significance of the netting step. Without the net stage, our framework is already sufficient to solve the closest pair problem, and in this specific case, the algorithm boils down to the one provided by Khuller and Matias [KM95]. This seems to be the only application of the new framework where the netting is not necessary. In particular, the wide applicability of the framework seems to be the result of the netting step.

Framework and results. We provide a framework that classifies which optimization problems can be solved using the new algorithm. We get the following new algorithms (all of them have an expected linear running time, for any fixed ε):

(A) **k-center clustering** (Section 2.4.1). We provide an algorithm that 2-approximates the optimal k-center clustering of a point set in $\mathbb{R}^d$. Unlike the previous algorithm [Har04a] that was restricted to $k = O(n^{1/6})$, the new algorithm works for any value of k. This new algorithm is also simpler.

(B) **kth smallest distance** (Section 2.4.2). In the distance selection problem, given a set of points in $\mathbb{R}^d$, one would like to compute the kth smallest distance defined by a pair of points of P. It is believed that such exact distance selection requires $\Omega\big(n^{4/3}\big)$ time in the worst case [Eri95], even in the plane (in higher dimensions the bound deteriorates). We present an $O(n/\varepsilon^d)$ time algorithm that

$(1+\varepsilon)$-approximates the kth smallest distance. Previously, Bespamyatnikh and Segal [BS02] presented $O(n \log n + n/\varepsilon^d)$ time algorithm using a well-separated pairs decomposition (see also [DHW12]).

Given two sets of points P and W with a total of n points, using the same approach, we can $(1+\varepsilon)$-approximate the kth smallest distance in a bichromatic set of distances $X = \{\mathsf{d}(\mathsf{p},\mathsf{q}) \mid \mathsf{p} \in \mathsf{P}, \mathsf{q} \in \mathsf{W}\}$.

Intuitively, the mechanism behind distance selection underlines many optimization problems, as it (essentially) performs a binary search over the distances induced by a set of points. As such, being able to do approximate distance selection in linear time should lead to faster approximation algorithms that perform a similar search over such distances.

(C) **The kth smallest m-nearest neighbor distance** (Section 2.4.3). For a set $\mathsf{P} = \{\mathsf{p}_1, \ldots, \mathsf{p}_n\} \subseteq \mathbb{R}^d$ of n points, and a point $\mathsf{p} \in \mathsf{P}$, its ***$m$th nearest neighbor*** in P is the mth closest point to p in $\mathsf{P} \setminus \{\mathsf{p}\}$. In particular, let $\mathsf{d}_m(\mathsf{p},\mathsf{P})$ denote this distance. Here, consider the set of these distances defined for each point of P; that is, $X = \{\mathsf{d}_m(\mathsf{p}_1,\mathsf{P}), \ldots, \mathsf{d}_m(\mathsf{p}_n,\mathsf{P})\}$. We can approximate the kth smallest number in this set in linear time.

(D) **Exact nearest neighbor distances** (Section 2.4.3). For the special case where $m = 1$, one can turn the above approximation into an exact computation of the kth nearest neighbor distance with only a minor post processing grid computation.

As an application, when $k = n$, one can compute in linear time, exactly, the furthest nearest neighbor distance; that is, the nearest neighbor distance of the point whose nearest neighbor is furthest away. This measure can be useful, for example, in meshing applications where such a point is a candidate for a region where the local feature size is large and further refinement is needed.

We are unaware of any previous work directly on this problem, although one can compute this quantity by solving the all-nearest-neighbor problem, which can be done in $O(n \log n)$ time [Cla83]. This is to some extent the "antithesis" to Rabin's algorithm for the closest pair problem, and it is somewhat surprising that it can also be solved in linear time.

(E) **The kth longest MST edge** (Section 2.4.4). Given a set P of n points in $\mathbb{R}^d$, we can $(1+\varepsilon)$-approximate, in $O(n/\varepsilon^d)$ time, the kth longest edge in the MST of P.

(F) **Smallest ball with a monotone property** (Section 2.4.5). Consider a property defined over a set of points, P, that is monotone; that is, if $\mathsf{W} \subseteq \mathsf{Q} \subseteq \mathsf{P}$ has this property then Q must also have this property. Consider such a property that can be easily checked, for example, (i) whether the set contains k points, or (ii) if the points are colored, whether all colors are present in the given point set. Given a point set P, one can $(1+\varepsilon)$-approximate, in $O(n/\varepsilon^d)$ time, the smallest radius ball b, such that $\mathsf{b} \cap \mathsf{P}$ has the desired property. For example, we get a linear time algorithm to approximate the smallest ball enclosing k points of P. The previous algorithm for this problem [HM05] was significantly more complicated. Furthermore, we can approximate the smallest ball such that all colors appear in it (if the points are colored), or the smallest ball such that at least t different colors appear in it, etc. More generally, the kind of monotone properties supported are *sketchable*; that is, properties for which there is a small summary of a point set that enables one to decide if the property holds, and furthermore, given summaries of two disjoint point sets, the summary of the union point set can be computed in constant time. We believe that formalizing this notion of sketchability is a useful abstraction. See Section 2.4.5 for details.

(G) **Smallest connected component with a monotone property** (Section 2.4.5). Consider the connected components of the graph where two points are connected if they are distance at most r from each other. Using our techniques, one can approximate, in linear time, the smallest r such that

there is a connected component of this graph for which a required sketchable property holds for the points in this connected component.

As an application, consider ad hoc wireless networks. Here, we have a set P of n nodes and their locations (say in the plane), and each node can broadcast in a certain radius r (the larger the r the higher the energy required, so naturally we would like to minimize it). Assume there are two special nodes. It is natural to ask for the minimum r, such that there is a connected component of the above graph that contains both nodes. That is, these two special nodes node can send message to each other, by message hopping (with distance at most r at each hop). We can approximate this connectivity radius in linear time.

(H) **Clustering for a monotone property** (Section 2.4.6). Imagine that we want to break the given point set into clusters, such that the maximum diameter of a cluster is minimized (as in k-center clustering), and furthermore, the points assigned to each cluster comply with some sketchable monotone property. We present a $(4+\varepsilon)$-approximation algorithm for these types of problems, that runs in $O(n/\varepsilon^d)$ time. This includes lower bounded clustering (i.e., every cluster must contain at least α points), for which recently the author and his co-authors provided an $O(n \log(n/\varepsilon))$ time $(4+\varepsilon)$-approximation algorithm [ERH12]. One can get a 2-approximation using network flow, but the running is significantly worse [APF+10]. See Section 2.4.6 for examples of clustering problems that can be approximated using this algorithm.

(I) **Connectivity clustering for a monotone property** (Section 2.4.6). Consider the problem of computing the minimum r, such that each connected component (of the graph where points in distance at most r from each other are adjacent) has some sketchable monotone property. We approximate the minimum r for which this holds in linear time.

An application of this for ad hoc networks is the following – we have a set P of n wireless clients, and some of them are base stations; that is, they are connected to the outside world. We would like to find the minimum r, such that each connected component of this graph contains a base station.

(J) **Closest pair and smallest non-zero distance** (Section 2.4.7). Given a set of points in $\mathbb{R}^d$, consider the problem of finding the smallest non-zero distance defined by these points. This problem is an extension of the closest pair distance, as there might be many identical points in the given point set. We provide a linear time algorithm for computing this distance *exactly*, which follows easily from our framework.

High probability. Finally, we show in Section 2.5 how to modify our framework such that the linear running time holds with high probability. Since there is very little elbow room in the running time when committed to linear running time, this extension is quite challenging, and requires several new ideas and insights. See Section 2.5 for details.

What can be done without the net & prune framework. For the problems studied in this chapter, an approximation algorithm with running time $O(n \log n)$ follows by using a $(1+\varepsilon)$-WSPD (well-separated pairs decomposition [CK95, Har11]), for some fixed ε, to compute $O(n)$ relevant resolutions, and then using binary search with the decider to find the right resolution. Even if the WSPD is given, it is not clear though how to get a linear time algorithm for the problems studied without using the net & prune framework. In particular, if the input has bounded spread, one can compute an $O(1)$-WSPD in linear time, and then use the WSPD inside the net & prune framework to get a deterministic linear time algorithm, see Section 2.3.3.

2.2 Preliminaries

2.2.1 Basic definitions

The ***spread*** of an interval $\mathcal{I} = [x, y] \subseteq \mathbb{R}^+$ is $\mathrm{sprd}(\mathcal{I}) = y/x$.

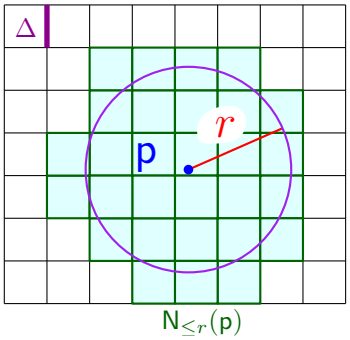
Definition 2.2.1. For a point set P in a metric space with a metric d, and a parameter $r > 0$, an ***r-net*** of P is a subset $\mathcal{C} \subseteq \mathsf{P}$, such that (i) for every $\mathsf{p}, \mathsf{q} \in \mathcal{C}$, $\mathsf{p} \neq \mathsf{q}$, we have that $\mathsf{d}(\mathsf{p}, \mathsf{q}) \geq r$, and (ii) for all $\mathsf{p} \in \mathsf{P}$, we have that $\min_{\mathsf{q} \in \mathcal{C}} \mathsf{d}(\mathsf{p}, \mathsf{q}) < r$.

Intuitively, an r-net represents P in resolution r.

Definition 2.2.2. For a real positive number Δ and a point $\mathsf{p} = (\mathsf{p}_1, \ldots, \mathsf{p}_d) \in \mathbb{R}^d$, define $\mathsf{G}_\Delta(\mathsf{p})$ to be the grid point $(\lfloor \mathsf{p}_1/\Delta \rfloor \Delta, \ldots, \lfloor \mathsf{p}_d/\Delta \rfloor \Delta)$.

The quantity Δ is the ***width*** or ***sidelength*** of the ***grid*** G_Δ. Observe that G_Δ partitions $\mathbb{R}^d$ into cubes, which are grid ***cells***. The grid cell of p is uniquely identified by the integer point $\mathrm{id}(\mathsf{p}) = \left(\lfloor \mathsf{p}_1/\Delta \rfloor, \ldots, \lfloor \mathsf{p}_d/\Delta \rfloor\right)$.

For a number $r \geq 0$, let $\mathsf{N}_{\leq r}(\mathsf{p})$ denote the set of grid cells in distance $\leq r$ from p, which is the ***neighborhood*** of p. Note, that the neighborhood also includes the grid cell containing p itself, and if $\Delta = \Theta(r)$ then $|\mathsf{N}_{\leq r}(\mathsf{p})| = \Theta\left((2 + \lceil 2r/\Delta \rceil)^d\right) = \Theta(1)$. See figure on the right.



2.2.2 Computing nets quickly for a point set in $\mathbb{R}^d$

There is a simple algorithm for computing r-nets. Namely, let all the points in P be initially unmarked. While there remains an unmarked point, p, add p to $\mathcal{C}$, and mark it and all other points in distance $< r$ from p (i.e. we are scooping away balls of radius r). By using grids and hashing one can modify this algorithm to run in linear time. The following is implicit in previous work [Har04a], and we include it here for the sake of completeness* – it was also described by the author and his co-authors in [ERH12].

Lemma 2.2.3. *Given a point set $\mathsf{P} \subseteq \mathbb{R}^d$ of size n and a parameter $r > 0$, one can compute an r-net for P in $O(n)$ time.*

Proof: Let G denote the grid in $\mathbb{R}^d$ with side length $\Delta = r/\left(2\sqrt{d}\right)$. First compute for every point $\mathsf{p} \in \mathsf{P}$ the grid cell in G that contains p; that is, $\mathrm{id}(\mathsf{p})$. Let $\mathcal{G}$ denote the set of grid cells of G that contain points of P. Similarly, for every cell $\square \in \mathcal{G}$ we compute the set of points of P which it contains. This task can be performed in linear time using hashing and bucketing assuming the floor function can be computed in constant time. Specifically, store the $\mathrm{id}(\cdot)$ values in a hash table, and in constant time hash each point into its appropriate bin.

Scan the points of P one at a time, and let p be the current point. If p is marked then move on to the next point. Otherwise, add p to the set of net points, $\mathcal{C}$, and mark it and each point $\mathsf{q} \in \mathsf{P}$ such that $\mathsf{d}(\mathsf{p}, \mathsf{q}) < r$. Since the cells of $\mathsf{N}_{\leq r}(\mathsf{p})$ contain all such points, we only need to check the lists of points stored in these grid cells. At the end of this procedure every point is marked. Since a point can only be marked if it is in distance $< r$ from some net point, and a net point is only created if it is unmarked when visited, this implies that $\mathcal{C}$ is an r-net.

*Specifically, the algorithm of Har-Peled [Har04a] is considerably more complicated than Lemma 2.2.3, and does not work in this settings, as the number of clusters it can handle is limited to $O\left(n^{1/6}\right)$. Lemma 2.2.3 has no such restriction.

As for the running time, observe that a grid cell, c, has its list scanned only if c is in the neighborhood of some created net point. As $\Delta = \Theta(r)$, there are only $O(1)$ cells which could contain a net point p such that $c \in \mathsf{N}_{\leq r}(\mathsf{p})$. Furthermore, at most one net point lies in a single cell since the diameter of a grid cell is strictly smaller than r. Therefore each grid cell had its list scanned $O(1)$ times. Since the only real work done is in scanning the cell lists and since the cell lists are disjoint, this implies an $O(n)$ running time overall. ■

Observe that the closest net point, for a point $\mathsf{p} \in \mathsf{P}$, must be in one of its neighborhood's grid cells. Since every grid cell can contain only a single net point, it follows that in constant time per point of P, one can compute each point's nearest net point. We thus have the following.

Corollary 2.2.4. *For a set $\mathsf{P} \subseteq \mathbb{R}^d$ of n points, and a parameter $r > 0$, one can compute, in linear time, an r-net of P, and furthermore, for each net point the set of points of P for which it is the nearest net point.*

In the following, a ***weighted point*** is a point that is assigned a positive integer weight. For any subset S of a weighted point set P, let $|S|$ denote the number of points in S and let $\omega(S) = \sum_{\mathsf{p} \in S} \omega(\mathsf{p})$ denote the total weight of S.

In particular, Corollary 2.2.4 implies that for a weighted point set one can compute the following quantity in linear time.

Algorithm 2.2.5 (net). *Given a weighted point set $\mathsf{P} \subseteq \mathbb{R}^d$, let $\mathbf{net}(r, \mathsf{P})$ denote an r-net of P, where the weight of each net point p is the total sum of the weights of the points assigned to it. We slightly abuse notation, and also use $\mathbf{net}(r, \mathsf{P})$ to designate the algorithm (of Corollary 2.2.4) for computing this net, which has linear running time.*

2.2.3 Identifying close and far points

For a given point $\mathsf{p} \in \mathsf{P}$, let

$$\nu(\mathsf{p}, \mathsf{P}) = \arg \min_{\mathsf{q} \in \mathsf{P} \setminus \{\mathsf{p}\}} \mathsf{d}(\mathsf{p}, \mathsf{q}) \qquad \text{and} \qquad \mathsf{d}(\mathsf{p}, \mathsf{P}) = \min_{\mathsf{q} \in \mathsf{P} \setminus \{\mathsf{p}\}} \mathsf{d}(\mathsf{p}, \mathsf{q}), \tag{2.1}$$

denote the nearest neighbor of p in $\mathsf{P} \setminus \{\mathsf{p}\}$, and the distance to it, respectively. The quantity $\mathsf{d}(\mathsf{p}, \mathsf{P})$ can be computed (naively) in linear time by scanning the points. For a set of points P, and a parameter r, let $\mathsf{P}^{\geq r}$ denote the set of ***r-far*** points; that is, it is the set of all points $\mathsf{p} \in \mathsf{P}$, such that the nearest-neighbor of p in $\mathsf{P} \setminus \{\mathsf{p}\}$ is at least distance r away (i.e., $\mathsf{d}(\mathsf{p}, \mathsf{P}) \geq r$). Similarly, $\mathsf{P}^{< r}$ is the set of ***r-close*** points; that is, all points $\mathsf{p} \in \mathsf{P}$, such that $\mathsf{d}(\mathsf{p}, \mathsf{P}) < r$.

Lemma 2.2.6. *Given a weighted set $\mathsf{P} \subseteq \mathbb{R}^d$ of n points, and a distance $\ell > 0$, in $O(n)$ time, one can compute the sets $\mathsf{P}^{< \ell}$ and $\mathsf{P}^{\geq \ell}$. Let $\mathbf{delFar}(\ell, \mathsf{P})$ denote the algorithm which computes these sets and returns $\mathsf{P}^{< \ell}$.*

Proof: Build a grid where every cell has diameter ℓ/c, for some constant $c > 1$. Clearly any point $\mathsf{p} \in \mathsf{P}$ such that $\mathsf{d}(\mathsf{p}, \mathsf{P}) \geq \ell$ (i.e., a far point) must be in a grid cell by itself. Therefore to determine all such "far" points only grid cells with singleton points in them need to be considered. For such a point q, to determine if $\mathsf{d}(\mathsf{q}, \mathsf{P}) \geq \ell$, one checks the points stored in all grid cells in distance $\leq \ell$ from it. If q has no such close neighbor, we mark q for inclusion in $\mathsf{P}^{\geq r}$. By the same arguments as in Lemma 2.2.3, the number of such cells is $O(1)$. Again by the arguments of Lemma 2.2.3 every non-empty grid cell gets scanned $O(1)$ times

overall, and so the running time is $O(n)$. Finally, we copy all the marked (resp. unmarked) points to $\mathsf{P}^{\geq \ell}$ (resp. $\mathsf{P}^{<\ell}$). The algorithm **delFar**(ℓ, P) then simply returns $\mathsf{P}^{<\ell}$. ■

2.3 Approximating nice distance problems

2.3.1 Problem definition and an example

Definition 2.3.1. Let P and Q be two sets of weighted points in $\mathbb{R}^d$ (of the same weight). The set Q is a ***Δ-drift*** of P, if Q can be constructed by moving each point of P by distance at most Δ (and not altering its weight). Formally, there is an onto mapping $f : \mathsf{P} \to \mathsf{Q}$, such that (i) For $\mathsf{p} \in \mathsf{P}$, we have that $\mathsf{d}(\mathsf{p}, f(\mathsf{p})) \leq \Delta$, and (ii) for any $\mathsf{q} \in \mathsf{Q}$, we have that $\omega(\mathsf{q})$ is the sum of the weights of all points $\mathsf{p} \in \mathsf{P}$ such that $f(\mathsf{p}) = \mathsf{q}$.

Note that for a (potentially weighted) point set W, **net**(Δ, W) is a Δ-drift of W.

Definition 2.3.2. Given a set X and a function $f : X \to \mathbb{R}$, a procedure **decider** is a ***φ-decider*** for f, if for any $x \in X$ and $r > 0$, **decider**(r, x) returns one of the following: (i) $f(x) \in [\alpha, \varphi\alpha]$, where α is some real number, (ii) $f(x) < r$, or (iii) $f(x) > r$.

Definition 2.3.3 (φ-NDP). An instance of a ***φ-NiceDistanceProblem*** consists of a pair (W, Γ), where $\mathsf{W} \subseteq \mathbb{R}^d$ is a set of n distinct weighted points[†], and Γ is the ***context*** of the given instance (of size $O(n)$) and consists of the relevant parameters for the problem[‡]. For a fixed ***φ-NDP***, the task is to evaluate a function $f(\mathsf{W}, \Gamma) \to \mathbb{R}^+$, defined over such input pairs, that has the following properties:

- Decider: There exists an $O(n)$ time φ-decider for f, for some *constant* $\varphi \geq 1$.
- Lipschitz: Let Q be any Δ-drift of W. Then $|f(\mathsf{W}, \Gamma) - f(\mathsf{Q}, \Gamma)| \leq 2\Delta$.
- Prune: If $f(\mathsf{W}, \Gamma) < r$ then $f(\mathsf{W}, \Gamma) = f(\mathsf{W}^{<r}, \Gamma')$, where $\mathsf{W}^{<r}$ is the set of r-close points, and Γ' is an updated context which can be computed, in $O(n)$ time, by a procedure denoted **contextUpdate**.

Remark 2.3.4 ($(1+\varepsilon)$-NDP). If we are interested in a $(1+\varepsilon)$-approximation, then $\varphi = 1+\varepsilon$. In this case, we require that the running time of the provided $(1+\varepsilon)$-decider (i.e., property Decider above) is $O(n/\varepsilon^c)$, for some constant $c \geq 1$. This covers all the applications presented in this chapter.

Our analysis still holds even if the decider has a different running time, but the overall running time might increase by a factor of $\log(1/\varepsilon)$, see Lemma 2.3.18$_{\text{p21}}$ for details.

An example – k center clustering

As a concrete example of an NDP, consider the problem of k-center clustering.

Problem 2.3.5 (kCenter). *Let W be a set of points in $\mathbb{R}^d$, and let $k > 0$ be an integer parameter. Find a set of* **centers** *$C \subseteq \mathsf{W}$ such that the maximum distance of a point of W to its nearest center in C is minimized, and $|C| = k$.*

Specifically, the function of interest, denoted by $f_{cen}(\mathsf{W}, k)$, is the radius of the optimal k-center clustering of W. We now show that kCenter satisfies the properties of an NDP.

[†]The case when W is a multiset can also be handled. See Remark 2.3.10.
[‡]In almost all cases the context is a set of integers, usually of constant size.

Remark 2.3.6. Usually one defines the input for kCenter to be a set of unweighted points. However, since the definition of NDP requires a weighted point set, we naturally convert the initial input into a set of unit weight points, W. Also, for simplicity we assume $f_{cen}(\mathsf{W}, k) \neq 0$, i.e. $|\mathsf{W}| > k$ (which can easily be checked). Both assumptions would be used implicitly throughout this chapter.

Lemma 2.3.7. *The following relations between* $|\mathbf{net}(r, \mathsf{W})|$ *and* $f_{cen}(\mathsf{W}, k)$ *hold: (A) If we have* $|\mathbf{net}(r, \mathsf{W})| \leq k$ *then* $f_{cen}(\mathsf{W}, k) < r$. *(B) If* $|\mathbf{net}(r, \mathsf{W})| > k$ *then* $r \leq 2 f_{cen}(\mathsf{W}, k)$.

Proof: (A) Observe that if $|\mathbf{net}(r, \mathsf{W})| \leq k$, then, by definition, the set of net points of $\mathbf{net}(r, \mathsf{W})$ are a set of $\leq k$ centers such that all the points of W are in distance strictly less than r from these centers.

(B) If $|\mathbf{net}(r, \mathsf{W})| > k$ then W must contain a set of $k+1$ points whose pairwise distances are all at least r. In particular any solution to kCenter with radius $< r/2$ would not be able to cover all these $k+1$ points with only k centers. ■

Lemma 2.3.8. *For any instance* (W, k) *of kCenter,* $f_{cen}(\mathsf{W}, k)$ *satisfies the properties of Definition 2.3.3; that is, kCenter is a* $(4+\varepsilon)$*-NDP, for any* $\varepsilon > 0$.

Proof: We need to verify the required properties, see Definition 2.3.3$_{\text{p16}}$.

Decider: We need to describe a decision procedure for kCenter clustering. To this end, given a distance r, the decider first calls $\mathbf{net}(r, \mathsf{W})$, see Algorithm 2.2.5$_{\text{p15}}$. If we have $|\mathbf{net}(r, \mathsf{W})| \leq k$, then by Lemma 2.3.7 the answer "$f_{cen}(\mathsf{W}, k) < r$" can be returned. Otherwise, call $\mathbf{net}((2+\varepsilon/2)r, \mathsf{W})$. If $|\mathbf{net}((2+\varepsilon/2)r, \mathsf{W})| \leq k$, then, by Lemma 2.3.7, we have $r/2 \leq f_{cen}(\mathsf{W}, k) < (2+\varepsilon/2)r$ and "$f \in [r/2, (2+\varepsilon/2)r]$" is returned. Otherwise $|\mathbf{net}((2+\varepsilon/2)r, \mathsf{W})| > k$, and Lemma 2.3.7 implies that the answer "$r < f_{cen}(\mathsf{W}, k)$" can be returned by the decider.

Lipschitz: Observe that if Q is a Δ-drift of W, then a point and its respective center in a k-center clustering of W each move by distance at most Δ in the transition from W to Q. As such, the distance between a point and its center changes by at most 2Δ by this process. This argument also works in the other direction, implying that the k-center clustering radius of W and Q are the same, up to an additive error of 2Δ.

Prune: It suffices to show that if $f_{cen}(\mathsf{W}, k) < r$ then $f_{cen}(\mathsf{W}, k) = f_{cen}\left(\mathsf{W}^{<r}, k - \left|\mathsf{W}^{\geq r}\right|\right)$. Now, if $f_{cen}(\mathsf{W}, k) < r$ then any point of W whose neighbors are all $\geq r$ away must be a center by itself in the optimal k-center solution, as otherwise it would be assigned to a center $\geq r > f_{cen}(\mathsf{W}, k)$ away. Similarly, any point assigned to it would be assigned to a point $\geq r > f_{cen}(\mathsf{W}, k)$ away. Therefore, for any point $\mathsf{p} \in \mathsf{W}$ whose nearest neighbor is at least distance r away, $f_{cen}(\mathsf{W}, k) = f_{cen}(\mathsf{W} \setminus \{\mathsf{p}\}, k-1)$. Repeating this observation implies the desired result. The context update (and computing $\mathsf{W}^{<r}$ and $\mathsf{W}^{\geq r}$) can by done in linear time using **delFar**, see Lemma 2.2.6$_{\text{p15}}$.

■

2.3.2 A linear time approximation algorithm

We now describe the general algorithm which given a φ-NDP, (W, Γ), and an associated target function, f, computes in linear time a $O(\varphi)$ spread interval containing $f(\mathsf{W}, \Gamma)$. In the following, let **decider** denote the given φ-decider, where $\varphi \geq 1$ is some parameter. Also, let **contextUpdate** denote the context updater associated with the given NDP problem, see Definition 2.3.3. Both **decider** and **contextUpdate** run in linear time. The algorithm for bounding the optimum value of an NDP is shown in Figure 2.1.

```
ndpAlg(W, Γ):
    Let W_0 = W, Γ_0 = Γ and i = 1.
    while TRUE do
1:      Randomly pick a point p from W_{i-1}.
2:      ℓ_i ← d(p, W_{i-1}).
        // Next, estimate the value of f_{i-1} = f(W_{i-1}, Γ_{i-1})
        res_> = decider(ℓ_i, W_{i-1}, Γ_{i-1})
        res_< = decider(c_net ℓ_i, W_{i-1}, Γ_{i-1})          // c_net = 37:  Corollary 2.3.16.
3:      if res_> = "f_{i-1} ∈ [x, y]" then return "f(W, Γ) ∈ [x/2, 2y]".
4:      if res_< = "f_{i-1} ∈ [x', y']" then return "f(W, Γ) ∈ [x'/2, 2y']".
        if res_> = "ℓ_i < f_{i-1}" and res_< = "f_{i-1} < c_net ℓ_i" then
5:          return "f(W, Γ) ∈ [ℓ_i/2, 2 c_net ℓ_i]".
        if res_> = "f_{i-1} < ℓ_i" then
6:          W_i = delFar(ℓ_i, W_{i-1})                                  // Prune
            Γ_i = contextUpdate(W_{i-1}, Γ_{i-1}, W_i)
        if res_< = "c_net ℓ_i < f_{i-1}" then
7:          W_i = net(3ℓ_i, W_{i-1})                                    // Net
            Γ_i = Γ_{i-1}
        i = i + 1
```

Figure 2.1: The approximation algorithm. The implicit target value being approximated is $f = f(\mathsf{W}, \Gamma)$, where f is a φ-NDP. That is, there is a φ-decider for f, denoted by **decider**, and the only access to the function f is via this procedure.

Remark 2.3.9. For the sake of simplicity of exposition we assume that $f(\mathsf{W}, \Gamma) \neq 0$. Since all our applications have $f(\mathsf{W}, \Gamma) \neq 0$ we choose to make this simplifying assumption. In particular, we later show in Corollary 2.3.16 that if initially $f(\mathsf{W}, \Gamma) \neq 0$, then **ndpAlg** preserves this property.

Remark 2.3.10. Note that the algorithm of Figure 2.1 can be modified to handle inputs where W is a multiset (namely, two points can occupy the same location), and we still assume (as done above) that $f(\mathsf{W}, \Gamma) \neq 0$. Specifically, it must be ensured that the distance computed in Line 2 is not zero, as this is required for the call to **decider**. This can be remedied (in linear time) by first grouping all the duplicates of the point sampled in Line 1 into a single weighted point (with the sum of the weights) before calling Line 2. This can be done by computing a net for a radius that is smaller than (say) half the smallest non-zero distance. How to compute this distance is described in Section 2.4.7.

Analysis

A ***net iteration*** of the algorithm is an iteration where **net** gets called. A ***prune iteration*** is one where **delFar** gets called. Note that the only other type of iteration is the one where the algorithm returns. In the following, $\mathsf{P}^{\leq \ell}$ is defined analogously to $\mathsf{P}^{< \ell}$ (see Section 2.2.3).

Lemma 2.3.11. *Let P be a point set. A $(2+\varepsilon)\ell$-net of P, for any $\varepsilon > 0$, can contain at most half the points of $\mathsf{P}^{\leq \ell}$.*

Proof: Consider any point p in $\mathsf{P}^{\leq \ell}$ which became one of the net points. Since $\mathsf{p} \in \mathsf{P}^{\leq \ell}$, a disk of radius ℓ centered at p must contain another point q from $\mathsf{P}^{\leq \ell}$ (indeed, $\mathsf{p} \in \mathsf{P}^{\leq \ell}$ only if its distance from its nearest neighbor in P is at most ℓ). Moreover, q cannot become a net point since it is too close to p. Now if we place a ball of radius ℓ centered at each point of $\mathsf{P}^{\leq \ell}$ which became a net point, then these balls will be disjoint

because the pairwise distance between net points is $\geq (2+\varepsilon)\ell$. Therefore each point of $\mathsf{P}^{\leq \ell}$ which became a net point can charge to at least one point of $\mathsf{P}^{\leq \ell}$ which did not make it into the net, such that no point gets charged twice. ∎

Lemma 2.3.12. *Given an instance* (W, Γ) *of an NDP, the algorithm* **ndpAlg**(W, Γ) *runs in expected* $O(n)$ *time.*

Proof: In each iteration of the while loop the only non-trivial work done is in computing ℓ_i, the two calls to **decider**, and the one call to either **net** or **delFar**. It has already been shown that all of these can be computed in $O(|\mathsf{W}_{i-1}|)$ time. Hence the total running time for the algorithm is $O\left(\sum_{i=0}^{i=k-1} |\mathsf{W}_i|\right)$, where k denotes the last (incomplete) iteration of the while loop.

So consider the beginning of iteration $i < k$ of the while loop. Let the points in W_{i-1} be labeled $\mathsf{p}_1, \mathsf{p}_2, \ldots, \mathsf{p}_m$ in increasing order of their nearest neighbor distance in W_{i-1}. Let j be the index of the point chosen in Line 1 and let $(\mathsf{W}_{i-1})^{\geq j}$ and $(\mathsf{W}_{i-1})^{\leq j}$ be the subset of the points with index $\geq j$ and index $\leq j$, respectively. Now since a point is randomly picked in Line 1, with probability $\geq 1/2$, $j \in [m/4, 3m/4]$. Lets call this event a ***successful iteration***. We have $\min\left(\left|(\mathsf{W}_{i-1})^{\geq j}\right|, \left|(\mathsf{W}_{i-1})^{\leq j}\right|\right) \geq |\mathsf{W}_{i-1}|/4$ for a successful iteration.

Since $i < k$ is not the last iteration of the while loop, either **delFar** or **net** must get called. If **delFar**$(\ell_i, \mathsf{W}_{i-1})$ gets called (i.e. Line 6) then by Lemma 2.2.6, all of $(\mathsf{W}_{i-1})^{\geq j}$ gets removed. So suppose **net** gets called (i.e. Line 7). In this case Lemma 2.3.11 implies that the call to **net** removes at least $\left|(\mathsf{W}_{i-1})^{\leq j}\right|/2$ points.

Therefore, for any iteration $i < k$, at least $\nu_i = \min\left(\left|(\mathsf{W}_{i-1})^{\geq j}\right|, \left|(\mathsf{W}_{i-1})^{\leq j}\right|/2\right)$ points get removed. If an iteration is successful then $\nu_i \geq |\mathsf{W}_{i-1}|/8$. In particular, $\mathbf{E}\left[\nu_i \mid |\mathsf{W}_{i-1}|\right] \geq |\mathsf{W}_{i-1}|/16$. Now, $|\mathsf{W}_i| \leq |\mathsf{W}_{i-1}| - \nu_i$ and as such $\mathbf{E}\left[|\mathsf{W}_i| \mid |\mathsf{W}_{i-1}|\right] \leq (15/16)|\mathsf{W}_{i-1}|$.

Therefore, for $0 < i < k$,

$$\mathbf{E}\Big[|\mathsf{W}_i|\Big] = \mathbf{E}\Big[\mathbf{E}\Big[|\mathsf{W}_i| \mid \mathsf{W}_{i-1}\Big]\Big] \leq \mathbf{E}\left[\frac{15}{16}|\mathsf{W}_{i-1}|\right] = \frac{15}{16}\mathbf{E}\Big[|\mathsf{W}_{i-1}|\Big].$$

Hence, by induction on i, $\mathbf{E}\left[|\mathsf{W}_i|\right] \leq (15/16)^i |\mathsf{W}_0|$ and so, in expectation, the running time is bounded by

$$O\left(\mathbf{E}\left[\sum_{i=0}^{i=k-1} |\mathsf{W}_i|\right]\right) = O\left(\sum_{i=0}^{i=k-1} \mathbf{E}\Big[|\mathsf{W}_i|\Big]\right) = O\left(\sum_{i=0}^{i=k-1} (15/16)^i |\mathsf{W}_0|\right) = O\Big(|\mathsf{W}_0|\Big).$$

∎

Correctness. The formal proof of correctness is somewhat tedious, but here is the basic idea: At every iteration, either far points are being thrown away (and this does not effect the optimal value), or we net the points. However, the net radius being used is always significantly smaller than the optimal value, and throughout the algorithm execution the radii of the nets being used grow exponentially. As such, the accumulated error in the end is proportional to the radius of the last net computation before termination, which itself is also much smaller than the optimal value.

Before proving that **ndpAlg** returns a bounded spread interval containing $f(\mathsf{W}_0, \Gamma_0)$, several helper lemmas will be needed. For notational ease, in the rest of this section, we use $f(\mathsf{W}_i)$ as shorthand for $f(\mathsf{W}_i, \Gamma_i)$.

Lemma 2.3.13. *Suppose that* **net** *is called in iteration i of the while loop (i.e., Line 7 in Figure 2.1$_{p18}$). Then for any iteration $j > i$ we have, $\ell_j \geq 3\ell_i$.*

Proof: Consider the beginning of iteration j of the while loop. The current set of points, W_{j-1}, are a subset of the net points of a $3\ell_i$-net (it is a subset since Line 6 and Line 7 might have been executed in between rounds i and j). Therefore, being a net, the distance between any two points of W_{j-1} is $\geq 3\ell_i$, see Definition 2.2.1$_{p14}$. In particular, this means that for any point p of W_{j-1}, we have $\mathsf{d}(\mathsf{p}, \mathsf{W}_{j-1}) \geq 3\ell_i$. ■

Lemma 2.3.14. *For $i = 1, \ldots, k$, we have $|f(\mathsf{W}_i) - f(\mathsf{W}_0)| \leq 9\ell_i$.*

Proof: Let I be the set of indices of the net iterations up to (and including) the ith iteration. Similarly, let $\overline{I}$ be the set of iterations where **delFar** get called.

If **net** was called in the jth iteration, then W_j is a $3\ell_j$-drift of W_{j-1} and so by the Lipschitz property, $|f(\mathsf{W}_j) - f(\mathsf{W}_{j-1})| \leq 6\ell_j$. On the other hand, if **delFar** gets called in the jth iteration, then $f(\mathsf{W}_j) = f(\mathsf{W}_{j-1})$ by the Prune property. Let $m = \max I$, we have that

$$\begin{aligned}
|f(\mathsf{W}_i) - f(\mathsf{W}_0)| &\leq \sum_{j=1}^{i} \Big| f(\mathsf{W}_j) - f(\mathsf{W}_{j-1}) \Big| \\
&= \sum_{j \in I} \Big| f(\mathsf{W}_j) - f(\mathsf{W}_{j-1}) \Big| + \sum_{j \in \overline{I}} \Big| f(\mathsf{W}_j) - f(\mathsf{W}_{j-1}) \Big| \\
&\leq \sum_{j \in I} 6\ell_j + \sum_{j \in \overline{I}} 0 \leq 6\ell_m \sum_{j=0}^{\infty} \frac{1}{3^j} \leq 9\ell_m \leq 9\ell_i,
\end{aligned}$$

by Lemma 2.3.13. ■

The following lemma testifies that the radii of the nets computed by the algorithm are always significantly smaller than the value we are trying to approximate.

Lemma 2.3.15. *For any iteration i of the while loop such that* **net** *gets called, we have $\ell_i \leq f(\mathsf{W}_0)/\eta$, where $0 < \eta = c_{\mathrm{net}} - 9$.*

Proof: The proof will be by induction. Let $m_1, \ldots m_t$ be the indices of the iterations of the while loop in which **net** gets called. For the base case, observe that in order for **net** to get called, we must have $\eta\ell_{m_1} < c_{\mathrm{net}}\ell_{m_1} < f(\mathsf{W}_{m_1-1})$. However, since this is the first iteration in which **net** is called it must be that $f(\mathsf{W}_0) = f(\mathsf{W}_{m_1-1})$ (since for all previous iterations **delFar** must have been called).

So now suppose that $\ell_{m_j} \leq f(\mathsf{W}_0)/\eta$ for all $m_j < m_i$. If a call to **net** is made in iteration m_i then again $c_{\mathrm{net}}\ell_{m_i} < f(\mathsf{W}_{(m_i)-1}) = f(\mathsf{W}_{m_{(i-1)}})$. Thus, by Lemma 2.3.14 and induction, we have

$$\ell_{m_i} < \frac{f\big(\mathsf{W}_{m_{(i-1)}}\big)}{c_{\mathrm{net}}} \leq \frac{f(\mathsf{W}_0) + 9\ell_{m_{(i-1)}}}{c_{\mathrm{net}}} \leq \frac{f(\mathsf{W}_0) + 9f(\mathsf{W}_0)/\eta}{c_{\mathrm{net}}} = \frac{1 + 9/\eta}{c_{\mathrm{net}}} f(\mathsf{W}_0) = \frac{f(\mathsf{W}_0)}{\eta},$$

if $\eta = \dfrac{c_{\mathrm{net}}}{1 + 9/\eta}$. This in turn is equivalent to $\eta + 9 = c_{\mathrm{net}}$, which is true by definition. ■

Setting $c_{\mathrm{net}} = 37$, results in $\eta = 28$, and by Lemma 2.3.15, for all i that correspond to a net iteration, $\ell_i \leq f(\mathsf{W}_0)/28$. By Lemma 2.3.14, for any net iteration i, we have

$$\Big| f(\mathsf{W}_i) - f(\mathsf{W}_0) \Big| \leq 9\ell_i \leq f(\mathsf{W}_0)/3.$$

In particular, we conclude that $|f(\mathsf{W}_i) - f(\mathsf{W}_0)| \leq f(\mathsf{W}_0)/3$ for any iteration i. We thus get the following.

Corollary 2.3.16. *For $c_{\text{net}} \geq 37$, and any i, we have:*

(A) $(2/3)f(\mathsf{W}_0) \leq f(\mathsf{W}_i) \leq (4/3)f(\mathsf{W}_0)$.

(B) If $f(\mathsf{W}_i) \in [x, y]$ *then* $f(\mathsf{W}_0) \in [(3/4)x, (3/2)y] \subseteq [x/2, 2y]$.

(C) If $f(\mathsf{W}_0) > 0$ *then* $f(\mathsf{W}_i) > 0$.

Lemma 2.3.17. *For $c_{\text{net}} \geq 37$, given an instance (W, Γ) of a φ-NDP, $\mathbf{ndpAlg}(\mathsf{W}, \Gamma)$ returns an interval $[x', y']$ containing $f(\mathsf{W}, \Gamma)$, where $\operatorname{sprd}([x', y']) \leq 4\max(\varphi, c_{\text{net}})$.*

Proof: Consider the iteration of the while loop at which **ndpAlg** terminates, see Figure 2.1$_{\text{p18}}$. If Line 3 or Line 4 get executed at this iteration, then the interval $[x, y]$ was computed by the φ-decider, and has spread $\leq \varphi$. As such, by Corollary 2.3.16, the returned interval $[x', y'] = [x/2, 2y]$ contains the optimal value, and its spread is $\leq 4\varphi$.

A similar argumentation holds if Line 5 gets executed. Indeed, the returned interval contains the desired value, and its spread is $4c_{\text{net}}$. ■

2.3.3 The result

In Definition 2.3.3 required that the decision procedure runs in $O(n)$ time. However, since our analysis also applies for decision procedures with slower running times, we first state the main result is these more general settings.

Lemma 2.3.18. *Given an instance of a φ-NDP defined by a set of n points in $\mathbb{R}^d$, one can get a φ-approximation to its optimal value, in $O(T(n))$ expected time, where $T(n)$ is the running time of the* **decider** *procedure, and $\varphi \geq 3/2$.*

If $\varphi = 1 + \varepsilon$, for some $\varepsilon \in (0, 1)$, (i.e., **decider** *is $(1+\varepsilon)$-decider), then one can compute a $(1+\varepsilon)$-approximation to the optimal value, and the expected running time is $O\big(T(n) \log \varepsilon^{-1}\big)$.*

Proof: Let (W, Γ) be the given φ-NDP instance, and let **decider** be the corresponding φ-decider. By Lemma 2.3.12 and Lemma 2.3.17, in expected $O(n)$ time, one can get a bounded spread interval $[\gamma, c\gamma]$, for some constant $c \geq 1$ such that $c = O(\varphi)$, such that $f = f(\mathsf{W}, \Gamma) \in [\gamma, c\gamma]$. If $c \leq \varphi$ then we are done. Otherwise, perform a binary search over this interval.

Specifically, for $i = 0, 1, \ldots, m = \lfloor \log_\varphi c \rfloor$, let $\gamma_i = \varphi^i \gamma$ and let $\gamma_{m+1} = c\gamma$. Now perform a binary search over the γ_i's using **decider**. If any of the calls returns an interval $[x, \varphi x]$ that contains the optimal value, then $f \leq \varphi x \leq \varphi f$ and so φx can be returned as the desired φ-approximation. Otherwise, if $f < \gamma_i$ or $f > \gamma_i$ the binary search moves left or right, respectively. In the end, the search ends up with an interval $(\gamma_i, \gamma_{i+1}) = (\gamma_i, \varphi\gamma_i)$ which must contain the optimal value, and again, its spread is φ, and it thus provide the required approximation.

The running time is dominated by the calls to the decider procedure (we are assuming here that $T(n) = \Omega(n)$). Clearly, the number of calls to the decider performed by this algorithm is $O(\log m) = O\big(\log \log_\varphi 4\varphi\big) = O(1)$, if $\varphi \geq 3/2$. Otherwise, if $\varphi = 1 + \varepsilon$, then

$$O(\log m) = O\big(\log \log_\varphi 4\varphi\big) = O\left(\ln\left(1 + \frac{\ln 4}{\ln(1+\varepsilon)}\right)\right) = O\left(\log\left(1 + \frac{1}{\varepsilon}\right)\right) = O\left(\log \frac{1}{\varepsilon}\right),$$

since $\ln(1+\varepsilon) \geq \ln e^{\varepsilon/2} = \varepsilon/2$, for $\varepsilon \in (0, 1/2)$, as can be easily verified. ■

The following is a restatement of the main result in the settings of Definition 2.3.3, where it was assume assumed $T(n) = O(n)$, or $T(n) = O(n/\varepsilon^d)$ when $\varphi = 1 + \varepsilon$. In the latter case, one can remove the $\log(1/\varepsilon)$ factor from the running time.

Theorem 2.3.19. *Given an instance of a φ-NDP defined by a set of n points in $\mathbb{R}^d$, one can get φ-approximate the optimal value, in expected $O(n)$ time, assuming $\varphi \geq 3/2$.*

For the case $\varphi = (1+\varepsilon) \in (1,2)$, given an $(1+\varepsilon)$-decider, with running time $O(n/\varepsilon^c)$, one can $(1+\varepsilon)$-approximate, in $O(n/\varepsilon^c)$ expected time, the given $(1+\varepsilon)$-NDP, where $c \geq 1$ is some constant.

Proof: Lemma 2.3.18 readily implies the first half, as we are now assuming $T(n) = O(n)$. As for the second half, Lemma 2.3.18 implies that one can get a $(1+\varepsilon)$-approximation in expected $O(n \log(1/\varepsilon)/\varepsilon^c)$ time. However, by using the same procedure as in the proof of Lemma 2.3.18 but with exponentially decreasing values for ε in the binary search, a factor of $\log(1/\varepsilon)$ can be avoided. This is a standard idea and was also used by Aronov and Har-Peled [AH08].

Let $\mathbf{decider}_\varepsilon$ be our $(1+\varepsilon)$-decider. If ε is set to any constant (say $1/2$), then by Lemma 2.3.12 and Lemma 2.3.17, in expected linear time, one can get a bounded spread interval $[\gamma, c\gamma]$, for some $c = O(\varphi)$, such that $f = f(\mathsf{W}, \Gamma) \in \mathcal{I}_0 = [\gamma, c\gamma]$.

Set $c = 1 + \varepsilon_0$ and $i = 1$. The algorithm now proceeds in rounds, doing the following in the ith round:

(A) Assume that in the beginning of the ith iteration, we know that

$$f \in \mathcal{I}_{i-1} = \Big[x_{i-1}, y_{i-1}\Big], \qquad \text{where} \qquad y_{i-1} = (1+\varepsilon_{i-1})x_{i-1}.$$

If $\varepsilon_{i-1} \leq \varepsilon$, then we found the desired approximation, and the algorithm stops.

(B) Set $\varepsilon_i = \sqrt{1+\varepsilon_{i-1}} - 1$ and $m_i = (1+\varepsilon_i)x_{i-1}$.

(C) $R_i \leftarrow \mathbf{decider}_{\varepsilon_i}(m_i, \mathsf{W}, \Gamma)$, see Definition 2.3.2$_{\text{p16}}$.

(D) There are three possibilities:

(i) If $R_i =$ "$f < m_i$", then set $\mathcal{I}_i \leftarrow [x_{i-1}, m_i]$.

(ii) If $R_i =$ "$f > m_i$", then set $\mathcal{I}_i \leftarrow [m_i, y_{i-1}]$.

(iii) If $R_i =$ "$f \in \mathcal{I} = [z, (1+\varepsilon_i)z]$", then set $\mathcal{I}_i \leftarrow \mathcal{I}$.

(E) $i \leftarrow i + 1$.

In each round, the algorithm computes an interval of spread $y_i/x_i = 1 + \varepsilon_i$ that contains f, and

$$\varepsilon_i = \sqrt{1+\varepsilon_{i-1}} - 1 = \frac{1+\varepsilon_{i-1}-1}{\sqrt{1+\varepsilon_{i-1}}+1} \leq \frac{\varepsilon_{i-1}}{2}.$$

Since the main bottleneck in each iteration is calling $\mathbf{decider}_{\varepsilon_i}$, which runs in $O(n/\varepsilon_i^c)$ time, the total running time is bounded by,

$$\sum_{i=1} O(n/\varepsilon_i^c) = O(n/\varepsilon^c),$$

since the sum behaves like a geometric series and the last term is $O(n/\varepsilon^c)$. ∎

A deterministic algorithm for the bounded spread case

The random sampling in the algorithm of Theorem 2.3.19 can be removed if the spread of P is polynomially bounded. Indeed, then one can snap the points to a grid, so that the points have integer coordinates

with polynomially bounded values. A compressed quadtree and a $O(1)$-WSPD of the snapped point set can be computed in linear time [Har11, Cha08]. Given such a WSPD, one can compute for each point its (approximate) nearest neighbor, and compute the median distance in this list in linear time. Using this value in each iteration of **ndpAlg** (instead of picking a random point and computing its nearest neighbor distance) results in an algorithm with linear running time. Note that the resulting algorithm is deterministic.

2.4 Applications

We now show that Theorem 2.3.19 can be applied to a wide array of problems. In order to apply it, one needs to show that the given problem meets the requirements of an NDP. Often the inputs to the problems considered are unweighted point sets. As such, when we say such a problem is an NDP, it is actually defined for unit weighted point sets. We also continue making the simplifying assumption that the optimum value is strictly positive (see Remark 2.3.9).

2.4.1 k-center clustering

Since computing a k-center clustering is an NDP problem (Lemma 2.3.8), plugging this into Theorem 2.3.19, immediately yields a constant factor approximation to k-center in linear time. It is easy to convert such an approximation to a 2-approximation using a grid, see Har-Peled [Har04a, Lemma 6.5]. Thus we get the following.

Theorem 2.4.1. *Given a set P of n points in $\mathbb{R}^d$, and a parameter k, $1 \leq k \leq n$, one can compute a 2-approximation to the optimal k-center clustering of P in (expected) linear time.*

A result similar to Theorem 2.4.1 was already known for the case $k = O\big(n^{1/3}/\log n\big)$ [Har04a], and the above removes this restriction. In addition, the new algorithm and its analysis are both simpler than the previous algorithm.

2.4.2 The kth smallest distance

Claim 2.4.2. *Let S and S' be subsets of $\mathbb{R}$ of size n, such that S' is obtained by taking each value in S and increasing or decreasing it by less than Δ. Let v and v' be the kth smallest values in S and S', respectively. Then $|v - v'| \leq \Delta$.*

Proof: Suppose for contradiction that $|v - v'| > \Delta$. If $v' - v > \Delta$ then S' has at least $n - k + 1$ values strictly larger than $v + \Delta$ which implies S has at least $n - k + 1$ values strictly larger than v. Similarly if $v - v' > \Delta$ then S' has at least k values strictly smaller than $v - \Delta$ which implies S has at least k values strictly smaller than v. ■

Lemma 2.4.3. *Let W be a weighted point set in $\mathbb{R}^d$, and let $k > 0$ be an integer parameter. Let $\binom{\mathsf{W}}{2}$ denote the multi-set of pairwise distances determined by W.[§] Given an instance (W, k) the k**thDistance** *problem asks you to output the kth smallest distance in $\binom{\mathsf{W}}{2}$. Given such an instance, one can $(1+\varepsilon)$-approximate the kth smallest distance in (expected) time $O\big(n/\varepsilon^d\big)$.*

[§]In particular a point of weight m is viewed as m unit weight points when determining the values of $\binom{\mathsf{W}}{2}$. For simplicity we assume that the kth distance in W is not zero (i.e. it is determined by two distinct points).

Proof: Let $f(\mathsf{W}, k)$ be the function that returns the kth smallest distance. We prove that this function is an $(1+\varepsilon)$-NDP (see Definition 2.3.3$_{\text{p16}}$).

Decider: Given r and ε, build a grid where every cell has diameter $\delta = \varepsilon r/8$, and store the points of W in this grid. Now, for any non-empty grid cell $\square$, let $\omega(\square)$ be the total weight of points in $\square \cap \mathsf{W}$, and register this weight with all the grid cells in distance at most r from it, i.e. all cells in $\mathsf{N}_{\leq r}(\square)$ (where $\mathsf{N}_{\leq r}(\square)$ is defined analogously for grid cells as $\mathsf{N}_{\leq r}(\mathsf{p})$ was for points in Definition 2.2.2). Let $\omega_{\mathsf{N}}(\square)$ denote the total weight registered with a cell $\square$ (which includes its own weight). Any point in $\square$ determines $\omega_{\mathsf{N}}(\square) - 1$ distances to other points in $\mathsf{N}_{\leq r}(\square)$. Therefore the total number of distances between points in $\square$ and points in $\mathsf{N}_{\leq r}(\square)$ is $\omega(\square)\big(\omega_{\mathsf{N}}(\square) - 1\big)$. Summing this over all cells (and dividing by 2 to avoid double counting) gives the quantity

$$S = \sum_{\square, \omega(\square) \neq 0} \frac{\omega(\square)}{2} \Big(\omega_{\mathsf{N}}(\square) - 1\Big).$$

Note, that the count S includes all distances which are $\leq r$, and no distances $> r + 2\delta$. So let the desired kth distance be denoted by ℓ_k. Thus, if $k \leq S$ then $\ell_k \leq r + 2\delta = (1+\varepsilon/4)r$. Similarly, if $k > S$ then $\ell_k > r$. Therefore, to build a $(1+\varepsilon)$-decider for distance r, we run the above counting procedure on $r_1 = r/(1+\varepsilon/3)$ and $r_2 = r$, and let S_1 and S_2 be the two counts computed, respectively. There are three cases:

(I) If $k \leq S_1$ then $\ell_k \leq (1+\varepsilon/4)r/(1+\varepsilon/3) < r$, and return this result.

(II) If $k \leq S_2$ then $\ell_k \leq (1+\varepsilon/4)r_2 = (1+\varepsilon/4)r$, then $\ell_k \in [r/(1+\varepsilon/3), (1+\varepsilon/4)r]$, and this interval has spread $(1+\varepsilon/4)(1+\varepsilon/3) < 1+\varepsilon$. The decider returns that this interval contains the desired value.

(III) The only remaining possibility is that $\ell_k > r_2$, and the decider returns this.

The running time of this decision procedure is $O\big(n/\varepsilon^d\big)$.

Lipschitz: Since the distance between any pair of points changes by at most 2Δ in a Δ-drift, the Lipschitz condition holds by Claim 2.4.2.

Prune: By assumption, the kth smallest distance is determined by two distinct weighted points p and q. Clearly these points are not in $\mathsf{W}^{\geq r}$ since $\mathsf{d}(\mathsf{p},\mathsf{q}) = f(\mathsf{W}, k) < r$. So consider any point $\mathsf{s} \in \mathsf{W}^{\geq r}$. Since removing s does not remove the distance $\mathsf{d}(\mathsf{p},\mathsf{q})$ from the set of remaining distances, all that is needed is to show how to update k. Clearly, s contributes $\omega(\mathsf{s}) \times \omega(\mathsf{W} \setminus \{\mathsf{s}\})$ distances, all of value $\geq r$, to $\binom{\mathsf{W}}{2}$, and $\binom{\omega(\mathsf{s})}{2}$ pairwise distances of value zero. Thus, after removing s from W, the new context is $k - \binom{\omega(\mathsf{s})}{2}$.

■

The algorithm of Lemma 2.4.3 also works (with minor modifications) if we are interested in the kth distance between two sets of points (i.e., the bipartite version).

Corollary 2.4.4. *Given two sets* P *and* Q *of points in* $\mathbb{R}^d$*, of total size* n*, and parameters* k *and* $\varepsilon > 0$*, one can* $(1+\varepsilon)$*-approximate, in expected* $O(n/\varepsilon^d)$ *time, the following:*

(A) The k*th smallest bichromatic distance. Formally, this is the smallest* k*th number in the multiset* $X = \big\{\mathsf{d}(\mathsf{p},\mathsf{q}) \mid \mathsf{p} \in \mathsf{P}, \mathsf{q} \in \mathsf{W}\big\}$.

(B) The closest bichromatic pair between P *and* Q.

2.4.3 The kth smallest m-nearest neighbor distance

For a set $\mathsf{P} = \{\mathsf{p}_1, \ldots, \mathsf{p}_n\} \subseteq \mathbb{R}^d$ of n points, and a point $\mathsf{p} \in \mathsf{P}$, its ***$m$th nearest neighbor*** in P is the mth closest point to p in $\mathsf{P} \setminus \{\mathsf{p}\}$. In particular, let $\mathsf{d}_m(\mathsf{p}, \mathsf{P})$ denote this distance. Here, consider the multiset of these distances defined for each point of P; that is, $X = \{\mathsf{d}_m(\mathsf{p}_1, \mathsf{P}), \ldots, \mathsf{d}_m(\mathsf{p}_n, \mathsf{P})\}$. Interestingly, for any m, we can approximate the kth smallest number in this set in linear time.

In order to use **ndpAlg**, one needs to generalize the problem to the case of weighted point sets. Specifically, a point p of weight $\omega(\mathsf{p})$ is treated as $\omega(\mathsf{p})$ distinct points at the same location. As such, the set X from the above is actually a multiset containing $\omega(\mathsf{p})$ copies of the value $\mathsf{d}_m(\mathsf{p}, \mathsf{P})$. In particular, if $\omega(\mathsf{p}) > m$ then $\mathsf{d}_m(\mathsf{p}, \mathsf{P}) = 0$.

Theorem 2.4.5. *Let W be a set of n weighted points in $\mathbb{R}^d$, and m, k, ε parameters. Then one can $(1+\varepsilon)$-approximate, in expected $O(n/\varepsilon^d)$ time, the kth smallest m-nearest neighbor distance in W. Formally, the algorithm $(1+\varepsilon)$-approximates the kth smallest number in the multiset $X = \{\mathsf{d}_m(\mathsf{p}_1, \mathsf{W}), \ldots, \mathsf{d}_m(\mathsf{p}_n, \mathsf{W})\}$*

Proof: We establish the required properties, see Definition 2.3.3$_{\text{p16}}$.

Decider: Let $f(\mathsf{W}, k, m)$ denote the desired quantity. We first need a $(1+\varepsilon)$-decider for this problem. To this end, given r, k, m, ε and W as input, create a grid with cells of diameter $\varepsilon r/4$, and mark for each point of W all the grid cells in distance at most r from it. Each non-empty grid cell has a count of the total weight of the points in distance at most r from it. Thus, each point of W can compute the total weight of all points which are approximately in distance at most r from it. Now, a point $\mathsf{p} \in \mathsf{W}$, can decide in constant time if (approximately) $\mathsf{d}_m(\mathsf{p}, \mathsf{W}) \leq r$. If the number of points which declare $\mathsf{d}_m(\mathsf{p}, \mathsf{W}) \leq r$ (where a point p is counted $\omega(\mathsf{p})$ times) is greater than k then the distance r is too large, and if it is smaller than k than r is too small. Being slightly more careful about the details (as was done in the proof of Lemma 2.4.3) one can verify this leads to a $(1+\varepsilon)$-decider for this problem.

Lipschitz: Clearly the Lipschitz property holds in this case, as Δ-drift only changes inter-point distances by at most an additive term of 2Δ.

Prune: We need to show that k can be updated properly for a weighted point p whose nearest neighbor is further away than $f(\mathsf{W}, k, m)$. If $\omega(\mathsf{p}) > m$ then we are throwing away $\omega(\mathsf{p})$ points, all with $\mathsf{d}_m(\mathsf{p}, \mathsf{W}) = 0$, and so k should be update to $k - \omega(\mathsf{p})$ when p is removed. Similarly, if $\omega(\mathsf{p}) \leq m$ then all the points, that p corresponds to, have $\mathsf{d}_m(\mathsf{p}, \mathsf{W})$ larger than the threshold, and k does not have to be updated.

Plugging this into Theorem 2.3.19 implies the desired result. ∎

Remark 2.4.6. Theorem 2.4.5 can be easily extended to work in the bichromatic case. That is, there are two point sets P and Q, and we are interested in the mth nearest neighbor of a point $\mathsf{p} \in \mathsf{P}$ in the set Q. It is easy to verify that the same time bounds of Theorem 2.4.5 hold in this case.

In particular, setting $k = |\mathsf{P}|$ and $m = 1$ in the bichromatic case, the computed distance will be the minimum radius of the balls needed to be placed around the points of Q to cover all the points of P (or vice versa).

Exact nearest neighbor distances, and furthest nearest neighbor

Using Theorem 2.4.5 with $m = 1$ and $\varepsilon = 1$, results in a 2-approximation, in $O(n)$ time, to the ***kth nearest neighbor distance*** of P. We now show how this can be converted into an exact solution.

So we have a quantity r that is larger than the kth nearest neighbor distance, but at most twice larger. We build a grid with a cell diameter being $r/4$. Clearly, any point whose nearest neighbor distance is at least the kth nearest neighbor distance must be the only point in its grid cell. For each such point p, we compute its distance to all the points stored in its neighborhood $\mathsf{N}_{\le r}(\mathsf{p})$, by scanning the list of points associated with these cells. This computes for these "lonely" points their exact nearest neighbor distance. The kth smallest nearest neighbor distance will then be the $(n-k)$th largest distance computed. Clearly, every cell's list of points get scanned a constant number of times, so overall the running time is linear. We summarize the result.

For a point set P, the ***furthest nearest neighbor distance***, is the maximum distance of a point of P, from the remaining points. Formally, it is $\max_{\mathsf{p}\in\mathsf{P}} \mathsf{d}(\mathsf{p},\mathsf{P})$.

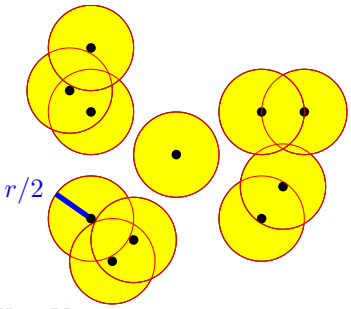
Theorem 2.4.7. *Let P be a set of n points in $\mathbb{R}^d$. For an integer k, $1 \le k \le n$, one can compute exactly, in expected linear time, the kth nearest neighbor distance in P. In particular, setting $k = n$, one can compute the furthest nearest neighbor distance exactly, in expected linear time.*

2.4.4 The spanning forest partitions, and the kth longest MST edge

The net computation provides a natural way to partition the data into clusters, and leads to a fast clustering algorithm. One alternative partition scheme is based on distance connectivity.

Definition 2.4.8. For a set of points P and a number $r > 0$, let $\mathcal{C}_{\le r}(\mathsf{P})$ be the ***r-connectivity clustering*** of P.

Specifically, it is a partition of P into connected components of the MST of P after all edges strictly longer than r are removed from it (alternatively, these are the connected components of the intersection graph where we replace every point of P by a disk of radius $r/2$ centered at that point), see figure on the right. As such, a set in such a partition of P is referred to as a ***connected component***.



Consider two partitions $\mathcal{P}, \mathcal{Q}$ of P. The partition $\mathcal{P}$ is a refinement of $\mathcal{Q}$, denoted by $\mathcal{P} \sqsubseteq \mathcal{Q}$, if for any set $X \in \mathcal{P}$, there exists a set $Y \in \mathcal{Q}$ such that $X \subseteq Y$.

Lemma 2.4.9. *Given a set $\mathsf{P} \subseteq \mathbb{R}^d$, and parameters r and ε, one can compute, in $O\big(n/\varepsilon^d\big)$ time, a partition $\mathcal{P}$, such that $\mathcal{C}_{\le r}(\mathsf{P}) \sqsubseteq \mathcal{P} \sqsubseteq \mathcal{C}_{\le (1+\varepsilon)r}(\mathsf{P})$.*

Proof: Build a grid with every cell having diameter $\varepsilon r/4$. For every point in P mark all the cells in distance $r/2$ from it. That takes $O(n/\varepsilon^d)$ time. Next, for every marked cell create a node v, and let V be the resulting set of nodes. Next, create a bipartite graph $G = (V \cup \mathsf{P}, E)$ connecting every node of V, to all the points of P marking its corresponding grid cell. Now, compute the connected components in G, and for each connected component extract the points of P that belong to it. Clearly, the resulting partition $\mathcal{P}$ of P, is such that any two points in distance $\le r$ are in the same set (since for such points there must be a grid cell in distance $\le r/2$ from both). Similarly, if the distance between two subsets $X, Y \subseteq \mathsf{P}$ is at least $(1+\varepsilon)r$, then they are in different connected components of $\mathcal{P}$. In particular, in order for X and Y to be in the same component, there must exists points $x \in X$ and $y \in Y$ which marked the same grid cell. However, such a grid cell would need to be in distance at most $r/2$ from both x and y, implying $\mathsf{d}(x,y) \le r/2 + r/2 + \varepsilon r/4 < (1+\varepsilon)r$, a contradiction. Thus, this is the desired partition. Clearly, this takes $O(n/\varepsilon^d)$ time overall. ∎

Theorem 2.4.10. *Given a set of n points P in $\mathbb{R}^d$, and parameters k and ε, one can output, in expected $O\big(n/\varepsilon^d\big)$ time, a $(1+\varepsilon)$-approximation to the kth longest (or kth shortest) edge in the MST of P.*

Proof: Consider the cost function $\ell_{MST}(\mathsf{P}, k)$ which returns the kth longest edge of the MST of P. It is easy to verify that $\ell_{MST}(\mathsf{P}, k)$ is the minimum r such that $\mathcal{C}_{\leq r}(\mathsf{P})$ has k connected components. For the sake of simplicity of exposition, we assume all the pairwise distances of points in P are distinct. We need to fill-in/verify the properties of Definition 2.3.3$_{\text{p16}}$:

Decider: The $(1+\varepsilon)$-decider for $\ell_{MST}(\mathsf{P}, k)$ uses the algorithm of Lemma 2.4.9. Specifically, for a specified $r > 0$, compute a partition $\mathcal{P}$, such that $\mathcal{C}_{\leq r}(\mathsf{P}) \sqsubseteq \mathcal{P} \sqsubseteq \mathcal{C}_{\leq (1+\varepsilon)r}(\mathsf{P})$. Now, if $\mathcal{P}$ has $\leq k$ connected components, then $\ell_{MST}(\mathsf{P}, k) \leq (1+\varepsilon)r$. Similarly, if $\mathcal{P}$ has $> k$ connected components, then $\ell_{MST}(\mathsf{P}, k) > r$. By calling on two values, and using slightly smaller ε, it is straightforward to get the desired $(1+\varepsilon)$-decider, as was done in Lemma 2.4.3$_{\text{p23}}$.

Lipschitz: Let P' be some Δ-drift of P, and let G (resp. G') denote the complete graph on P (resp. P'). For an edge $e \in E(G)$ (or in $E(G')$), let $\ell(e)$ denote the length of e. For the sake of simplicity of exposition, we consider P' and P to be multi-sets of the same cardinality (i.e., $n = |\mathsf{P}| = |\mathsf{P}'| > k$). (Note this implies some of the edges may have length 0.) For a point $\mathsf{p} \in \mathsf{P}$, let p' denote its image in P' after the Δ-drift. Let $e_1, \ldots, e_{n-1}$ be the edges in the MST of G sorted by increasing order by their length. Let $e'_1, \ldots, e'_{n-1}$ denote the corresponding edges in G', i.e. for $e = uv$, the corresponding edge is $e' = u'v'$.

Observe that the subgraph of G' having $e'_1, \ldots, e'_i$ as edges, is a forest with $n-i$ connected components. As such, we have that $\ell_{MST}(\mathsf{P}', n-i) \leq \max_{j=1}^{i} \ell(e'_j)$, since by the correctness of the Kruskal's algorithm, the first i edges of the MST form such a forest with the maximum weight edge being minimum (among all such forests). As such, we have that

$$\begin{aligned}\ell_{MST}(\mathsf{P}', n-i) &\leq \max_{j=1}^{i} \ell(e'_j) \leq \max_{j=1}^{i} (\ell(e_j) + 2\Delta) = 2\Delta + \max_{j=1}^{i} \ell(e_j) \\ &= 2\Delta + \ell_{MST}(\mathsf{P}, n-i).\end{aligned}$$

A symmetric argument implies that $\ell_{MST}(\mathsf{P}, n-i) \leq 2\Delta + \ell_{MST}(\mathsf{P}', n-i)$. Setting $i = n-k$, we have that $|\ell_{MST}(\mathsf{P}, k) - \ell_{MST}(\mathsf{P}', k)| \leq 2\Delta$, as desired.

Prune: Consider a point $\mathsf{p} \in \mathsf{W}$, such that $\mathsf{d}(\mathsf{p}, \mathsf{W}) \geq r$, and $r > \ell_{MST}(\mathsf{W}, k)$. This implies that all the MST edges adjacent to p are of distance $\geq r$. Assume there are j such edges, and observe that removing these edges creates $j+1$ connected components of the MST, such that each connected component is of distance $\geq r$ from each other (indeed, otherwise, one of the removed edges should not be in the MST, as can be easily verified). Thus, if we delete p, and recompute the MST, we will get j new edges that replace the deleted edges, all of these edges are of length $\geq r$. That is, deleting p, and decreasing k by one, ensures that the target value remains the same in the pruned instance.

This establish that $\ell_{MST}(\mathsf{P}, k)$ is an $(1+\varepsilon)$-NDP, and by plugging this into Theorem 2.3.19, we get the desired result. ∎

Remark 2.4.11. Note that in the above, as in other parts of this chapter, we treat a point of weight ω as ω copies of the same point. This naturally leads to edges of length zero in the MST, which are being handled correctly by the algorithm.

2.4.5 Computing the minimum cluster

Sketchable families

Definition 2.4.12 (Upward Closed Set System). Let P be a finite ground set of elements, and let $\mathcal{F}$ be a family of subsets of P. Then $(\mathsf{P}, \mathcal{F})$ is an ***upward closed set system*** if for any $X \in \mathcal{F}$ and any $Y \subseteq \mathsf{P}$, such that $X \subseteq Y$, we have that $Y \in \mathcal{F}$. Such a set system is a ***sketchable family***, if for any set $S \subseteq \mathsf{P}$ there exists a constant size ***sketch*** $\mathrm{sk}(S)$ such that:

(A) For any $S, T \subseteq \mathsf{P}$ that are disjoint, $\mathrm{sk}(S \cup T)$ can be computed from $\mathrm{sk}(S)$ and $\mathrm{sk}(T)$ in $O(1)$ time. We assume the sketch of a singleton can be computed in $O(1)$ time, and as such the sketch of a set $S \subseteq \mathsf{P}$ can be computed in $O(|S|)$.

(B) There is a membership oracle for the set system based on the sketch. That is, there is a procedure **orac** such that given the sketch of a subset $\mathrm{sk}(S)$, **orac** returns whether $S \in \mathcal{F}$ or not, in $O(1)$ time.

An example for such a sketchable family, is the set system $(\mathsf{P}, \mathcal{F})$, where $S \subseteq \mathsf{P}$ is in $\mathcal{F}$ if $|S| \geq 10$. Here the sketch of the set is simply the number of elements in the set, and combining two sketches $\mathrm{sk}(S)$ and $\mathrm{sk}(T)$ is adding the numbers to get $\mathrm{sk}(S \cup T)$ (for $S \cap T = \emptyset$).

We will be interested in two natural problems induced by such a family: (i) smallest cluster – find the smallest set in the family with certain properties, and (ii) min-max clustering – find disjoint sets in the family such that they cover the original set, and the maximum price of these sets is minimized.

Remark 2.4.13. In the following, we consider sketchable families $(\mathsf{P}, \mathcal{F})$, where P is a set of Euclidean points. In this case, $\mathsf{P} = \{\mathsf{p}_1, \ldots, \mathsf{p}_n\}$ can be viewed as a set of elements, where each p_i is additionally given a realization at some location in $\mathbb{R}^d$. Specifically, changing the locations of the realizations does not change the subsets in $\mathcal{F}$. This is required to ensure the Lipschitz property holds. Specifically, sketches remain the same after a Δ-drift. Furthermore, we do not have to maintain the sketches under deletions (of the underlying points), as pruned points corresponds to complete sketches that are being thrown away, see Remark 2.4.19.

Remark 2.4.14. Note that it is not necessary for the sketches to have $O(1)$ size or the oracle to run in $O(1)$ time, however, assuming otherwise will affect the running time of **ndpAlg** if it is used to solve a problem involving a sketchable family.

Example 2.4.15. Consider associating a positive k-dimensional vector ψ_{p} with each point $\mathsf{p} \in \mathsf{P}$ (a vector is ***positive*** if it is non-zero, and all its coordinates are non-negative). A ***positive linear inequality*** is an inequality of the form $\sum_i \alpha_i x_i \geq c$, where the coefficients $\alpha_1, \ldots, \alpha_k$ are all non-negative. For such a linear inequality, consider the set system $(\mathsf{P}, \mathcal{F})$, where a set Q is in $\mathcal{F}$ if the linear inequality holds for the vector $\sum_{\mathsf{p} \in \mathsf{Q}} \psi_{\mathsf{p}}$. Clearly, this family is a sketchable family, the sketch being the sum of the vectors associated with the points of the set (here k is assumed to be a constant).

It is easy to verify that sketchable families are closed under finite intersection. Specifically, given a collection of m such positive inequalities, the family of sets such that their sketch vector complies with all these inequalities is a sketchable family (of course, checking if a set, given its sketch, is in the family naively would take $O(mk)$ time).

As a concrete application, consider the scenario where every element in P has $k = 4$ attributes. One might be interested in subsets, Q, such that the sum of the first two attributes of the vector $\sum_{\mathsf{p} \in \mathsf{Q}} \psi_{\mathsf{p}}$ is at least 1, and the sum of last two attributes is at least 2.

Of course, in general, a membership oracle for the attributes space that has the property that if ψ is valid then $\psi + \psi'$ is also valid, for any positive ψ', would define a sketchable family. As a concrete example, consider the non-linear (and not convex) condition that the sum of at least two attributes is larger than 1. Clearly, this defines a sketchable family.

Clearly the above definition of sketchable family is very general and widely applicable. Next, we show how **ndpAlg** can be used to approximate certain objectives over sketchable families.

Min cluster

We now consider the problem of minimizing the cluster radius of a subset of a given point set subject to inclusion in a sketchable family. Specifically, we consider the case when the cluster is defined by a ball of radius r or when the cluster is defined by a connected component of $\mathcal{C}_{\leq r}(\mathsf{P})$ for a radius r. Note that as a ball or component grows, both the set of points the cluster contains and the inclusion of this set of points in the sketchable $\mathcal{F}$, are monotone properties. This correspondence is what allows us to apply our general framework.

Theorem 2.4.16. *For a set of n points $\mathsf{P} \subseteq \mathbb{R}^d$, and a sketchable family $(\mathsf{P}, \mathcal{F})$, one can $(1+\varepsilon)$-approximate, in expected $O\big(n/\varepsilon^d\big)$ time, the radius of the smallest ball, b, such that $\mathsf{b} \cap \mathsf{P} \in \mathcal{F}$.*

Proof: Let $f(\mathsf{P})$ be the diameter of the smallest ball b in $\mathbb{R}^d$ such that $\mathsf{P} \cap \mathsf{b} \in \mathcal{F}$. We claim that $f(\cdot)$ is an NDP, see Definition 2.3.3$_{\text{p16}}$. The Lipschitz and Prune properties readily hold for $f(\cdot)$. Specifically, $f(\mathsf{P})$ was chosen to be the diameter rather than the radius so that the Prune property holds. Note that below, for the Decider property, we refer to approximating the radius, which gives us the same approximation to the diameter.

For the Decider property, given r and $\varepsilon > 0$, construct a grid with cells of diameter $\varepsilon r/4$, and register each point of P in all the grid cells in distance at most r from it. If there is a ball, b, of radius r such that $\mathsf{b} \cap \mathsf{P} \in \mathcal{F}$ then the set of points registered with the grid cell containing the center of this ball will be a superset of $\mathsf{b} \cap \mathsf{P}$ and hence the set is in $\mathcal{F}$, by the upward closed property of sketchable families. Moreover, the set registered at this grid cell requires a ball of radius at most $r + 2\varepsilon r/4$ (centered at any point in this grid cell) to cover it. Thus, the decision procedure simply checks the set of points associated with each grid cell to see whether or not it is in $\mathcal{F}$. The definition of sketchable families implies this can be done in linear time in the total sizes of these sets (i.e., $O(n/\varepsilon^d)$). Furthermore, if there is no ball of radius $(1+\varepsilon)r$ whose point set is in $\mathcal{F}$ then the decision procedure would fail to find such a cell. Thus, this is a $(1+\varepsilon)$-decision procedure, and its running time is $O(n/\varepsilon^d)$.

Plugging this into the algorithm of Theorem 2.3.19 implies the result. ∎

Remark 2.4.17. In the algorithm of the above theorem, as the algorithm progresses, every weighted point that is computed also has an associated sketch from the original points that it corresponds to. One can view this collection of sketches as either an attribute in addition to the weights, or as being passed down via the context.

The following is a sample of what the above theorem implies.

Corollary 2.4.18. *We can $(1+\varepsilon)$-approximate, in $O(n/\varepsilon^d)$ time, the following problems for a set of n points in $\mathbb{R}^d$:*

(A) The smallest ball containing k points of P.

(B) The points of P are weighted, and there is an additional threshold α. Compute the smallest ball containing points with total weight at least α.

(C) If the points of P are colored by k colors, the smallest ball containing points of P, such that they are colored by at least t different colors. (Thus, one can find the smallest non-monochromatic ball [$t = 2$], and the smallest ball having all colors [$t = k$].) The running time is $O\left(nk/\varepsilon^d\right)$, as the sketch here is a k-dimensional vector.

Remark 2.4.19. A careful inspection of our algorithm reveals that we only merge sketches or throw them away. Specifically, a pruning stage of our algorithm corresponds to throwing away some of the sketches, and netting corresponds to merging together some of the sketches. See also Remark 2.4.13.

Min cluster in the spanning forest

Note that $\mathcal{C}_{\le r}(\mathsf{P})$ is a monotone partition as r increases (i.e., components merge, but never split apart). It is natural to ask what is the minimum r, for which there is a connected component in $\mathcal{C}_{\le r}(\mathsf{P})$ that is in a sketchable family.

Theorem 2.4.20. *For a set of n points $\mathsf{P} \subseteq \mathbb{R}^d$, and a sketchable family $(\mathsf{P}, \mathcal{F})$, one can $(1+\varepsilon)$-approximate, in expected $O\left(n/\varepsilon^d\right)$ time, the minimum r, such that there is a connected component in $\mathcal{C}_{\le r}(\mathsf{P})$ that is in $\mathcal{F}$.*

Proof: The target function $f(\mathsf{P})$ is the smallest r such that $\mathcal{C}_{\le r}(\mathsf{P})$ contains a set that is in $\mathcal{F}$. Let $r_{opt} = f(\mathsf{P})$. The algorithm first checks if any of the input points by themselves have the desired property, if so then $r_{opt} = 0$ and the algorithm returns 0. (For the case that P is a multiset, see Remark 2.3.10$_{\text{p18}}$.) As such, we can assume that $f(\mathsf{P}) > 0$, and furthermore, by Corollary 2.3.16 (C), $f(\cdot)$ is always non-zero during the algorithm execution. Again, we need to verify the properties of Definition 2.3.3$_{\text{p16}}$ for $f(\cdot)$:

Decider: One can $(1+\varepsilon/4)$-approximate the connected components of $\mathcal{C}_{\le r/(1+\varepsilon/3)}(\mathsf{P})$, using the algorithm of Lemma 2.4.9, and for each approximate connected component, use their sketch to decide if any of them is in $\mathcal{F}$. If so, return that r is larger than the optimal value. Otherwise, perform a $(1+\varepsilon/3)$-approximate computation of $\mathcal{C}_{\le r}(\mathsf{P})$, and if one of its clusters is in $\mathcal{F}$, then return that the desired value is in the interval $[r/(1-\varepsilon/3), r(1+\varepsilon/3)]$. Otherwise, return that r is smaller than the optimal value. Clearly, this is $(1+\varepsilon)$-decider that works in $O(n/\varepsilon^d)$ time.

Lipschitz: Let $r' = f(\mathsf{P})$. By the definition of $f(\cdot)$, there exists k, such that $r' = \ell_{MST}(\mathsf{P}, k)$. Now, arguing as in the proof of Theorem 2.4.10$_{\text{p27}}$, implies that $f(\cdot)$ has the desired Lipschitz property.

Prune: Consider a point $\mathsf{p} \in \mathsf{W}$, such that $\mathsf{d}(\mathsf{p}, \mathsf{W}) \ge r$, and $r > f(\mathsf{W}) > 0$. Now, p by itself can not have the desired property, as this would imply that $f(\mathsf{W}) = 0$. This implies that in $\mathcal{C}_{\le r_{opt}}(\mathsf{W})$ the point p is by itself (and not in any cluster that is in $\mathcal{F}$). As such, throwing p away does not change the target function. Formally, $f(\mathsf{W}) = f(\mathsf{W} \setminus \{\mathsf{p}\})$.

Plugging this into Theorem 2.3.19 implies the result. ∎

One natural application for Theorem 2.4.20, is for ad hoc wireless networks. Here, we have a set P of n nodes and their locations (say in the plane), and each node can broadcast in a certain radius r (the larger the r the higher the energy required, so naturally we would like to minimize it). It is natural now to ask for the minimum r such that one of the connected components in the resulting ad hoc network has some desired property. For example, in $O(n/\varepsilon^d)$ time, we can $(1+\varepsilon)$-approximate the smallest r such that:

(A) One of the connected components of $\mathcal{C}_{\leq r}(\mathsf{P})$ contains half the points of P, or more generally if the points are weighted, that one of connected component contains points of total weight at least α, for a prespecified α.

(B) If the points are colored, the desired connected component contains all the colors (for example, each color represent some fraction of the data, and the cluster can recover the data if all the pieces are available), or at least two colors, or more generally a different requirement on each color.

2.4.6 Clustering for monotone properties

Definition 2.4.21 (Min-Max Clustering). We are given a sketchable family $(\mathsf{P}, \mathcal{F})$, and a cost function $g : 2^{\mathsf{P}} \to \mathbb{R}^+$. We are interested in finding disjoint sets $S_1, \ldots, S_m \in \mathcal{F}$, such that (i) $\bigcup_i S_i = \mathsf{P}$, and (ii) $\max_i g(S_i)$ is minimized. We will refer to the partition realizing the minimum as the ***optimal clustering*** of P.

Theorem 2.4.22. *Let P be a set of points in $\mathbb{R}^d$, and let $(\mathsf{P}, \mathcal{F})$ be a sketchable family. For a set $\mathsf{W} \in \mathcal{F}$, let $\mathsf{r}_{\min}(\mathsf{W})$ be the radius of the smallest ball centered at a point of and enclosing W. One can $(4+\varepsilon)$-approximate, in expected $O(n/\varepsilon^d)$ time, the min-max clustering under $\mathsf{r}_{\min}$ of P.*

That is, one can cover P by a set of balls, and assign each point of P to one of these balls, such that the set of points assigned to each ball is in $\mathcal{F}$, and the maximum radius of any of these balls is a $(4+\varepsilon)$-approximation to the minimum radius used by any such cover.

Proof: Let $\mathcal{P}_{\text{opt}}$ be the optimal partition with radius r_{opt}, and consider an r-net $\mathcal{N}$ for $r \geq 4r_{opt}$, computed using Corollary 2.2.4. Consider a point $\mathsf{p} \in \mathcal{N}$, and let $\mathsf{P}_{\mathcal{N}}[\mathsf{p}]$ be the set of points of P assigned to p by the nearest net-point assignment.

Next, consider the cluster $\mathsf{W} \in \mathcal{P}_{\text{opt}}$ that contains it. Clearly, $\operatorname{diam}(\mathsf{W}) \leq 2r_{opt}$, and the distance of p from all other net points in $\mathcal{N}$ is at least $4r_{opt}$. It follows that $\mathsf{W} \subseteq \mathsf{P}_{\mathcal{N}}[\mathsf{p}]$, and since $\mathsf{W} \in \mathcal{F}$, it follows that $\mathsf{P}_{\mathcal{N}}[\mathsf{p}] \in \mathcal{F}$.

A 4-decider for this problem works by computing the $4r$-net $\mathcal{N}$, and for each $\mathsf{p} \in \mathcal{N}$, checking the sketchable property for the set $\mathsf{P}_{\mathcal{N}}[\mathsf{p}]$. It is easy to verify that the properties of Definition 2.3.3$_{\text{p16}}$ hold in this case. In particular, throwing a far away isolated point corresponds to a cluster that already fulfill the monotone property, and it is too far away to be relevant. Namely, computing r_{opt} is an NDP and so plugging this into Theorem 2.3.19 implies the result. ■

Lower bounded center clustering

If the required sketchable property is that every cluster contains at least k points, then Theorem 2.4.22 approximates the ***lower bounded center*** problem. That is, one has to cover the points by balls, such that every cluster (i.e., points assigned to a ball) contains at least k points. The price of this clustering is the radius of the largest ball used. A 2-approximation to this problem is known via the usage of flow [APF+10] but the running time is super quadratic. Recently, the author and his co-authors showed a similar result to Theorem 2.4.22 with running time (roughly) $O(n \log n)$ [ERH12]. This paper also shows that this problem cannot be approximated to better than (roughly) 1.8 even for points in the plane.

Corollary 2.4.23. *Let P be a set of points in $\mathbb{R}^d$, and let k and $\varepsilon > 0$ be parameters. One can $(4+\varepsilon)$-approximate the lower bounded center clustering in $O(n/\varepsilon^d)$ time.*

Other clustering problems

One can plug-in any sketchable family into Theorem 2.4.22. For example, if the points have k colors, we can ask for the min-max radius clustering, such that every cluster contains (i) all colors, (ii) at least two different colors, or (iii) a different requirement on each color, etc.

As another concrete example, consider that we have n customers in the plane, and each customer is interested in k different services (i.e., there is a k-dimensional vector associated with each customer specifying his/her demand). There are t types of service centers that can be established, but each such center type requires a minimum level of demand in each of these k categories (i.e., each type is specified by a minimum demand k-dimensional vector, and a set of customers can be the user base for such a service center if the sum of their demands vector is larger than this specification). The problem is to partition the points into clusters (of minimum maximum radius), such that for every cluster there is a valid service center assigned to it. Clearly, this falls into the framework of Theorem 2.4.22, and can be $(4+\varepsilon)$-approximated in $O(nkt/\varepsilon^d)$ time.

Clustering into spanning forests

One can get a similar result to Theorem 2.4.22 for the connectivity version of clustering of P. Formally, a set of points $\mathsf{W} \subseteq \mathsf{P}$ is ***r-valid*** if W is contained in some set of $\mathcal{C}_{\leq r}(\mathsf{P})$. Given a sketchable family $(\mathsf{P}, \mathcal{F})$, a partition $\mathcal{P}$ of P is an ***r-connected clustering*** if all the sets in $\mathcal{P}$ are in $\mathcal{F}$, and are r-valid.

Theorem 2.4.24. *Let P be a set of points in $\mathbb{R}^d$, and let $(\mathsf{P}, \mathcal{F})$ be a sketchable family. One can $(1+\varepsilon)$-approximate r_{opt}, in expected $O(n/\varepsilon^d)$ time, where r_{opt} is the minimum value such that there is a r_{opt}-connected clustering of P.*

Proof: Let $f(\cdot)$ be the target function in this case. We verify the properties of Definition 2.3.3$_{\text{p16}}$:

Decider: Given r, we use Lemma 2.4.9 to compute a partition $\mathcal{P}$ of P such that $\mathcal{C}_{\leq r}(\mathsf{P}) \sqsubseteq \mathcal{P} \subseteq \mathcal{C}_{\leq (1+\varepsilon)r}(\mathsf{P})$. If the sketchable property holds for each cluster of $\mathcal{P}$, then return that r is too large, otherwise return that it is too small. As for the quality of this decider, observe that if the optimal partition has a cluster W that uses points from two different clusters of $\mathcal{P}$, than W is not r-valid, as otherwise these two points would be in the same cluster of $\mathcal{P}$ (namely, $r_{opt} > r$).

Lipschitz: Follows easily by arguing as in Theorem 2.4.10$_{\text{p27}}$.

Prune: If an isolated point exists, then it can be thrown away because the cluster of original points it corresponds to, is a valid cluster that can be used in the final clustering, and it does not interact with any other clusters.

Plugging this into Theorem 2.3.19 now implies the result. ■

A nice application of Theorem 2.4.24 is for ad hoc networks. Again, we have a set P of n wireless clients, and some of them are base stations; that is, they are connected to the outside world. We would like to find the minimum r, such that each connected component of $\mathcal{C}_{\leq r}(\mathsf{P})$ contains a base station.

2.4.7 Smallest non-zero distance

Given a point set P, it was already shown in Section 2.4.3 that, with a small amount of post processing, **ndpAlg** can be used to compute the closest pair distance exactly in expected linear time. If one allows P to

$\mathsf{d}(\mathsf{p},\mathsf{P})$	$\min_{\mathsf{q}\in\mathsf{P}\setminus\{\mathsf{p}\}} \mathsf{d}(\mathsf{p},\mathsf{q})$	Distance from p to its nearest neighbor in $\mathsf{P}\setminus\{\mathsf{p}\}$.
$\mathsf{d}_i(\mathsf{p},\mathsf{P})$		Distance of p to its ith nearest neighbor in $\mathsf{P}\setminus\{\mathsf{p}\}$.
$\mathtt{D}(\mathsf{X},\mathsf{P})$	$\{\mathsf{d}(\mathsf{x},\mathsf{P}) \mid \mathsf{x}\in\mathsf{X}\}$	Multi-set of distances between points in X and nearest neighbors in P.
$\mathtt{D}_i(\mathsf{X},\mathsf{P})$	$\{\mathsf{d}_i(\mathsf{x},\mathsf{P}) \mid \mathsf{x}\in\mathsf{X}\}$	Multi-set of distances between point in X and their ith nearest neighbors in P.
$\mathsf{d}^\delta(\mathsf{X},\mathsf{P})$		Value of rank $\lfloor\delta\,\lvert\mathsf{X}\rvert\rfloor$ in $\mathtt{D}(\mathsf{X},\mathsf{P})$, for some $\delta\in[0,1]$.
$\mathsf{d}_i^\delta(\mathsf{X},\mathsf{P})$		Value of rank $\lfloor\delta\,\lvert\mathsf{X}\rvert\rfloor$ in $\mathtt{D}_i(\mathsf{X},\mathsf{P})$, for some $\delta\in[0,1]$.

Table 2.1: Notation used. If $\mathsf{X}=\mathsf{P}$, the second argument is omitted. Thus, $\mathsf{d}_i^\delta(\mathsf{P})=\mathsf{d}_i^\delta(\mathsf{P},\mathsf{P})$.

contain duplicate points (i.e. P is a multiset) then a couple modifications to **ndpAlg** must be made. First, modify the algorithm so that for the selected point p it finds the closest point to p that has non-zero distance from it. Second, modify **delFar** (see Lemma 2.2.6$_{\text{p15}}$) so that a singleton cell means all points in the cell have the same location, rather than meaning the cell contains a single point. This can easily be checked for all cells in overall linear time. With these two modifications, the above algorithm works verbatim, and we get the following.

Lemma 2.4.25. *Let P be a multiset of weighted points in $\mathbb{R}^d$. Then one can solve the SmallestNonZeroDist problem* exactly *for P, in expected linear time. In particular, if P contains no duplicates then this corresponds to computing the closest pair distance.*

Interestingly, the algorithm of Lemma 2.4.25 is a prune-and-search algorithm, as the net stage never gets executed. Observe, that it is not hard to extend the algorithm of Golin *et al.* [GRSS95] to solve this variant, and the result of Lemma 2.4.25 is included in this chapter only for the sake of completeness. In particular, the resulting (simplified) algorithm is similar to the algorithm of Khuller and Matias [KM95].

2.5 Linear Time with high probability

In Section 2.3, we presented an algorithm (**ndpAlg**, see Figure 2.1$_{\text{p18}}$) that provides a constant factor approximation to a wide variety of problems in expected linear time. Here we present an extension of that algorithm that runs in linear time with high probability.

Our algorithm works by computing a quantity that is not directly related to the problem at hand (but in linear time!), and then relating this quantity to the desired quantity, enabling us to compute it in linear time. Specifically, the resulting algorithm is somewhat counterintuitive and bizarre. As such, to make the algorithm description more accessible to the reader, we introduce the basic ideas in Section 2.5.1. We describe the algorithm in detail in Section 2.5.2, and analyze it in Section 2.5.3.

2.5.1 Preliminaries

Notation. See Table 2.1 for a summary of the notation used in this part of the chapter. As it is sufficient for our purposes, for simplicity (in this section) P will always be an unweighted points set, i.e. $n=\omega(\mathsf{P})=|\mathsf{P}|$. The algorithm can be easily extended to handle the weighted case.

Basic idea

Sample in the middle. Observe that if one could sample a nearest neighbor distance that lies in the interval $[\mathsf{d}^{\delta}(\mathsf{P}), \mathsf{d}^{1-\delta}(\mathsf{P})]$ for some fixed constant $\delta \in (0,1)$, then we are done. Indeed, for each iteration of **ndpAlg** for which the sampled distance was near the middle, it is guaranteed that the algorithm either terminates or removes a constant fraction of the points. As such it is sufficient to present an algorithm which returns a value from this range with probability $\geq 1 - 1/n^c$, for some sufficiently large constant c, where $n = |\mathsf{P}|$. Such a value near the median is a ***middle*** nearest neighbor distance.

For the rest of this section we therefore focus on the problem of computing such an approximate middle nearest neighbor distance, in linear time, with high probability. To turn **ndpAlg** into an algorithm which runs in linear time with high probability, we replace Line 1 and Line 2 in Figure 2.1$_{\text{p18}}$ with this new subroutine.

Sampling for the median. The problem with the random sampling from $\mathtt{D}(\mathsf{P})$ done in **ndpAlg** is that though, with good probability, the sampled distance lies near the middle, it does not do so with high probability. This is the same problem one is faced when performing the standard basic randomized median selection. For median selection one known solution is rather than randomly sampling a pivot, to instead take a sufficiently large random sample, and take the median of the random sample as the pivot. This pivot is good with high probability [MR95], and it is this result we wish to mimic.

The challenge. Given the above discussion, the natural solution to finding a good middle value from $\mathtt{D}(\mathsf{P})$, would be to sample multiple values from $\mathtt{D}(\mathsf{P})$, and then take the median. By the Chernoff inequality, if one samples at least a logarithmic number of values from a set of n values ($\mathtt{D}(\mathsf{P})$ in our case) then, with high probability, the median of the sample will lie near the median of the entire set. (See Lemma 2.5.3$_{\text{p37}}$ for a formal statement and proof of this observation.) The problem with this approach is that sampling $O(\log n)$ values from $\mathtt{D}(\mathsf{P})$ takes $\Theta(n \log n)$ time (at least naively). Specifically, while the sampling is easy, computing the associated values requires $\Omega(n)$ time per sampled point (note this is not a problem for median selection as there we are sampling values directly rather than points from which values are determined).

Therefore, we instead not only take a sample from which to select our median value, but also a sample from which we will determine what these values are. Namely, for each point in a $\Theta(\log n)$ sized random sample, compute it's nearest neighbor in a second $\Theta(n/\log n)$ sized sample, and then take the median of the computed values, and let this quantity be ℓ. Though this approach takes $\Theta(n)$ time, we need to somehow relate ℓ to the desired middle nearest neighbor distance.

Intuition. First note that for most points in the $\Theta(\log n)$ sample, their nearest neighbor in P will not make it into the $\Theta(n/\log n)$ sample from which we compute nearest neighbor distances. This implies ℓ is an over estimate of middle nearest neighbor distances in P. The high level intuition is that how well ℓ approximates middle distances tells us something about the distribution of P. There are two cases.

In the first case, most of the points of P are in small clusters[¶]. This case can be detected since here ℓ will be way off, since for any point in the $\Theta(\log n)$ sample, a $\Theta(n/\log n)$ sample is not likely to have a point from the same small cluster. The advantage is that if points are in small clusters, then one only needs

[¶]Think of constant size clusters here, although in the algorithm they may be up to $\log^2 n$ sized. Also, note the clusters are non-trivial as otherwise we are really in the second case.

```
   midNN(P):
       ℓ ← estLogDist(P).

       res ← deciderM(P, ℓ, 3/4)
       if res = "d^{3/4}(P) ∈ [x, y]" then return x                    // y/x = O(1)
1:     if res = "ℓ < d^{3/4}(P)" then return ℓ                         //Remark 2.5.5

       // Must be that res = "ℓ > d^{3/4}(P)".
       I ← ℓ / 64 ⌈ln² n⌉

       res_log ← deciderM(P, I, 1/2)
       if res_log = "d^{1/2}(P) ∈ [x, y]" then return x                // y/x = O(1)
       if res_log = "I < d^{1/2}(P)" then
2:         return lowSpread(P, I/4, 4ℓ)                                //Section 2.5.2

       // Must be that res_log = "I > d^{1/2}(P)"
3:     return smallComp(ℓ, P)                                          //Section 2.5.2
```

Figure 2.2: Algorithm for computing a constant factor approximation to a value in the interval $[\mathsf{d}^{\delta}(\mathsf{P}), \mathsf{d}^{1-\delta}(\mathsf{P})]$, for some fixed constant $\delta \in (0,1)$.

to look within a small cluster to compute a nearest neighbor distance (and this is done using connectivity clustering).

In the second case, rather than being clustered, P is roughly evenly distributed. Hence, intuitively, for any k, $\mathsf{d}_k(\mathsf{p},\mathsf{P}) \leq k\mathsf{d}(\mathsf{p},\mathsf{P})$. Note that for any point in the $\Theta(\log n)$ sample, with high probability we are expected to see one of its $\log^2 n$ nearest neighbors in a $\Theta(n/\log n)$ sample, and so $\ell \leq \mathsf{d}^{1/2}_{\log^2 n}(\mathsf{P})$. Combining these observations, roughly speaking $\ell \leq (\log^2 n)\mathsf{d}^{1/2}(\mathsf{P})$. In other words, we know the median lies in a $\log^2 n$ spread interval, $[\ell/\log^2 n, \ell]$. This case is then resolved using nets and grids (the details here are non-trivial).

2.5.2 The algorithm

We now present the algorithm in full detail. Specifically, let **midNN** denote the new algorithm for computing a value in $\mathcal{I} = [\mathsf{d}^{\delta}(\mathsf{P}), \mathsf{d}^{1-\delta}(\mathsf{P})]$, for some fixed constant $\delta \in (0,1)$. In Section 2.4.3 it was shown how to convert a constant factor approximation to such a value into an exact value. Hence, computing a constant factor approximation to a value in $\mathcal{I}$ in linear time, with high probability, suffices for our purposes.

The full algorithm is shown in Figure 2.2. Before analyzing the algorithm, we first define several subroutines that it employs.

deciderM: Counting distances

Given a point set P and a distance r, the decision procedure **deciderM**(P, r, α) decides if the tth nearest neighbor distance in $\mathtt{D}(\mathsf{P})$ is smaller than r, larger than r, or roughly equal to it, where $t = \lfloor \alpha |\mathsf{P}| \rfloor$. This decider is implicitly described in Theorem 2.4.5, but a simpler decider suffices, which we describe here in detail for the sake of clarity. To this end, throw the points of P into a grid with cell diameter r. Now, for every point $\mathsf{p} \in \mathsf{P}$, let $w_r(\mathsf{p})$ be the number of the points in $\mathsf{N}_{\leq r}(\mathsf{p})$, not including p itself. Clearly $w_r(\mathsf{p})$ can be computed in constant time (given we have first recorded the number of points in each non-empty grid

estLogDist(P):
$n = |\mathsf{P}|$
$\mathsf{X} \leftarrow$ random sample of P of size $\lceil c_5 \ln n \rceil$.
$\mathsf{S} \leftarrow$ random sample of P of size $\lceil c_6 n / \ln n \rceil$
Compute $\mathtt{D}(\mathsf{X}, \mathsf{S})$
return the element of rank $|\mathsf{X}|/4$ in $\mathtt{D}(\mathsf{X}, \mathsf{S})$.

Figure 2.3: Roughly estimating $\mathtt{d}^{1/4}_{O(\log n)}(\mathsf{P})$. Here c_5 and c_6 are sufficiently large constants.

cell). Now, we compute the count

$$T = \sum_{\mathsf{p} \in \mathsf{P}} \frac{w_r(\mathsf{p})}{2},$$

in $O(|\mathsf{P}|)$ time. Now, if $T < t$ the procedure returns that r is smaller than the tth nearest-neighbor distance. Similarly, if $T \geq t$, then the tth nearest-neighbor distance must be at most $3r$. In this case, by also computing the count for the distance $r/3$, one will be able to either return a constant spread interval that contains the desired distance, or that this distance is strictly smaller than r. Thus yielding the required decider.

estLogDist: Sampling a distance

Given a set P of n points in $\mathbb{R}^d$, take two random samples from P, a sample X of size $\Theta(\log n)$ and a sample S of size $\Theta(n/\log n)$, and compute $\mathtt{d}^{1/4}(\mathsf{X}, \mathsf{S})$. This can be done in $O(n)$ time, by scanning all of S for each point in X to compute the set of distances $\mathtt{D}(\mathsf{X}, \mathsf{S})$. Next, compute value of rank $|\mathsf{X}|/4$ in this set. We denote this algorithm by **estLogDist**(P), and it is shown in Figure 2.3.

lowSpread: Logarithmic Spread

The subroutine **lowSpread** used by **midNN** (i.e. Line 2 in Figure 2.2$_{\text{p35}}$) handles the case when the desired middle nearest neighbor distance lies in a poly-logarithmic spread interval. The analysis of **lowSpread** can be found in Section 2.5.3. In the following, P is a set of unweighted points such that $[\mathtt{d}^{1/2}(\mathsf{P}), \mathtt{d}^{3/4}(\mathsf{P})] \subseteq [r, R]$, for some values $0 < r \leq R$, where $n = |\mathsf{P}|$. (Note, in the following, when sampling from a set S we treat it as a set of $\omega(\mathsf{S})$ points.)

Algorithm 2.5.1 (lowSpread(P, r, R)).

(A) *Compute* $\mathsf{S} =$ **net**$(r/8, \mathsf{P})$.

(B) *Throw the points of* S *into a grid of sidelength* R.

(C) $\mathsf{U} \leftarrow$ *random sample from* S *of* $\Theta(\log(n))$ *points.*

(D) *Let* Z *be an approximation to the set* $\mathtt{D}(\mathsf{U}, \mathsf{S})$.
Specifically, for each point $\mathsf{p} \in \mathsf{U}$ *look within* $\mathsf{N}_{\leq 2R}(\mathsf{p})$ *for its nearest neighbor, see Definition 2.2.2*$_{p14}$*. If such a neighbor is found record this distance (which might be zero if* p *is of weight* > 1*) and if no neighbor is found set this value to* ∞.

(E) *Let* $\mathsf{p} \in \mathsf{P}$ *be the point that corresponds to the number of rank* $(5/8)|\mathsf{Z}|$ *in* Z*, and let* τ *be this number.*

(F) *Return* τ *(*τ *is a constant approximation to* $\mathtt{d}(\mathsf{p}, \mathsf{P})$*).*

smallComp: Many small components

The subroutine **smallComp**, used by **midNN** (i.e. Line 3 in Figure 2.2), handles the case when the desired middle nearest neighbor distance is considerably smaller than the value sampled by **estLogDist**. This corresponds, intuitively, to the situation where the input is mainly clustered in tiny clusters. The analysis of **smallComp** is delegated to Section 2.5.3.

Algorithm 2.5.2 (smallComp(ℓ, P)). *The algorithm works as follows:*

(A) Compute a partition $\mathcal{C}$, such that $\mathcal{C}_{\leq\rho}(\mathsf{P}) \sqsubseteq \mathcal{C} \sqsubseteq \mathcal{C}_{\leq 2\rho}(\mathsf{P})$ in linear time, using the algorithm of Lemma 2.4.9$_{p26}$, where $\rho = \ell/8m$, where $m = \lceil \ln^2 n \rceil$ and $n = |\mathsf{P}|$.

(B) Identify the components of $\mathcal{C}$ which are neither singletons nor contain more than $c_3 \ln^2 n$ points, where c_3 is a sufficiently large constant.

(C) Collect all the points in these components into a set of points X (this set is not empty).

(D) Take a random sample U of $c_4 \log n$ points from X, where c_4 is a sufficiently large constant.

(E) For each point in U, compute its nearest neighbor in P by scanning the non-trivial connected component in $\mathcal{C}$ that contains it. Let Z be this set of numbers.

(F) Return the median value in Z.

2.5.3 Algorithm analysis

Why median of a sample is a good estimate for the median

The following lemma is by now standard, and we include its proof for the sake of completeness.

Lemma 2.5.3. *Let X be a set of n real numbers, constants $\delta \in (0, 1/2)$, $\alpha \in (0,1)$, and a parameter $t > 0$. Let U be a random sample of size t picked (with repetition and uniformly) from X. Then, the element of rank αt in U has rank in the interval $\Big[(1-\delta)\alpha n, (1+\delta)\alpha n\Big]$ in the original set X, with probability $\geq 1 - 2\exp(-\delta^2 \alpha t/8)$. In particular, for $t \geq (8c \ln n)/\big(\delta^2 \alpha\big)$, this holds with probability $\geq 1 - 2/n^c$.*

Proof: Let Y_L be the number of values in U of rank $\leq (1-\delta)\alpha n$ in X. Similarly, let Y_R be the number of values in U of rank $\leq (1+\delta)\alpha n$ in X. Clearly $\mu_L = \mathbf{E}\big[Y_L\big] = (1-\delta)\alpha t$ and $\mu_R = \mathbf{E}\big[Y_R\big] = (1+\delta)\alpha t$.

Observe that if $Y_L \leq \alpha t \leq Y_R$ then the element of rank αt in U has rank in X in the interval $\Big[(1-\delta)\alpha n, (1+\delta)\alpha n\Big]$, as desired. By the Chernoff inequality [MR95], we have that

$$\begin{aligned}\mathbf{Pr}\Big[Y_L > \alpha t\Big] &= \mathbf{Pr}\left[Y_L > \frac{\alpha t}{\mu_L}\mu_L\right] = \mathbf{Pr}\left[Y_L > \frac{\alpha t}{(1-\delta)\alpha t}\mu_L\right] \leq \mathbf{Pr}\Big[Y_L > (1+\delta)\mu_L\Big] \\ &\leq \exp\left(-\frac{\delta^2}{4}\mu_L\right) \leq \exp\left(-\frac{\delta^2}{8}\alpha t\right),\end{aligned}$$

as $1/(1-\delta) > 1+\delta$. Arguing as above, the other bad event probability is

$$\begin{aligned}\mathbf{Pr}\Big[Y_R < \alpha t\Big] &\leq \mathbf{Pr}\left[Y_R < \frac{\alpha t}{\mu_R}\mu_R\right] = \mathbf{Pr}\left[Y_R < \frac{\alpha t}{(1+\delta)\alpha t}\mu_R\right] \\ &\leq \mathbf{Pr}\left[Y_R < \left(1-\frac{\delta}{2}\right)\mu_R\right] \leq \exp\left(-\frac{\delta^2}{8}\mu_R\right) \leq \exp\left(-\frac{\delta^2}{8}\alpha t\right).\end{aligned}$$

■

estLogDist: Analysis

Lemma 2.5.4. *Let* P *be a set of* n *points in* $\mathbb{R}^d$*. Let* ℓ *be the distance returned by executing* **estLogDist**(P)*, see Figure 2.3*$_{p36}$*. With high probability, we have the following:*

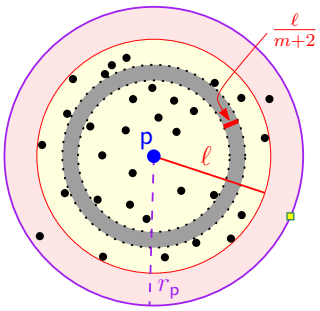
(A) For all points $\mathsf{p} \in \mathsf{P}$*, the set* $|\mathsf{P} \cap \mathsf{b}(\mathsf{p}, r_{\mathsf{p}})|$ *has at most* $m = \lceil \ln^2 n \rceil$ *points, where (i)* $r_{\mathsf{p}} = \mathsf{d}(\mathsf{p}, \mathsf{S})$*, (ii)* S *is the random sample used by* **estLogDist***, and (iii)* $\mathsf{b}(\mathsf{p}, r_{\mathsf{p}})$ *is the ball of radius* r_{p} *centered at* p*.*

(B) Let $\mathcal{P}$ *be a clustering of* P *such that* $\mathcal{C}_{\le \rho}(\mathsf{P}) \sqsubseteq \mathcal{P} \sqsubseteq \mathcal{C}_{\le 2\rho}(\mathsf{P})$*, as computed by the algorithm of Lemma 2.4.9*$_{p26}$*, for* $\rho = \ell/8m$*. Then, at least* $(5/8)n$ *of the points of* P *are in clusters of* $\mathcal{P}$ *that contain at most* m *points.*

(C) $\ell \ge \mathsf{d}^{1/8}(\mathsf{P})$*.*

Proof: (A) Fix a point $\mathsf{p} \in \mathsf{P}$, and consider a ball b centered at p whose interior contains exactly $\lceil \ln^2 n \rceil$ points of P. The probability that a random point from P hits b is $\alpha = \ln^2 n/n$. As such, the probability that all the points of S miss it is at most $(1-\alpha)^{c_5 n/\ln n} \le e^{-c_5 \ln n} \le 1/n^{c_5}$, implying the claim, if c_5 is sufficiently large.

(B) By Lemma 2.5.3, for c_6 sufficiently large, we have that ℓ is in the range $\left[\mathsf{d}^{1/8}(\mathsf{P},\mathsf{S}), \mathsf{d}^{3/8}(\mathsf{P},\mathsf{S})\right]$, and this holds with high probability. That is, for at least $(5/8)n$ points of P have $\mathsf{d}(\mathsf{p},\mathsf{S}) \ge \ell$, and let Q be this set of points. For every point $\mathsf{p} \in \mathsf{Q}$, we have that the ball $\mathsf{b} = \mathsf{b}(\mathsf{p}, \ell) \subseteq \mathsf{b}(\mathsf{p}, r_{\mathsf{p}})$ contains at most m points of P, by (A). Partitioning b into equal radius rings of width $\ell/(m+2)$, one of them must be empty of points of P. In particular, no cluster of $\mathcal{C}_{\le 2\rho}(\mathsf{P})$ can contain points on both sides of this ring, see figure on the right. As such, at least $(5/8)n$ of the points of P are in clusters of $\mathcal{P}$ that contain at most m points.



(C) Follows immediately from (B), as $\ell \ge \mathsf{d}^{1/8}(\mathsf{P},\mathsf{S}) \ge \mathsf{d}^{1/8}(\mathsf{P})$. ■

Remark 2.5.5. Line 1 of **midNN**(P) returns ℓ as the approximate middle nearest neighbor distance if **deciderM** returned $\ell < \mathsf{d}^{3/4}(\mathsf{P})$. Lemma 2.5.4 (C) then implies that $\mathsf{d}^{1/8}(\mathsf{P}) \le \ell < \mathsf{d}^{3/4}(\mathsf{P})$, with high probability, and so ℓ is a valid approximation. In the unlikely event that $\mathsf{d}^{1/8}(\mathsf{P}) > \ell$, one simply runs **midNN**(P) again. This event can be determined with the call **deciderM**$(\mathsf{P}, \ell, 1/8)$.

lowSpread: Analysis

During an execution of **midNN**(P), **lowSpread**(P, r, R) gets called (see Algorithm 2.5.1) only if the interval $[r, R]$ contains the values $\mathsf{d}^{1/2}(\mathsf{P}), \ldots, \mathsf{d}^{3/4}(\mathsf{P})$.

Lemma 2.5.6. *Let* P *be a point set in* $\mathbb{R}^d$ *of size* n*. Given* $r, R \in \mathbb{R}$*, such that (i)* $R/r = O(\log^2 n)$*, and (ii)* $\mathsf{d}^{1/2}(\mathsf{P}), \ldots, \mathsf{d}^{3/4}(\mathsf{P}) \in [r, R]$*, then* **lowSpread**$(\mathsf{P}, r, R)$ *runs in* $O(n)$ *time, and returns a value which is a* $O(1)$*-approximation to some nearest neighbor distance* $\mathsf{d}(\mathsf{p}, \mathsf{P}) \in [\mathsf{d}^{1/2}(\mathsf{P}), \mathsf{d}^{3/4}(\mathsf{P})]$*, with high probability, for some point* $\mathsf{p} \in \mathsf{P}$*.*

Proof: Set $\Delta = r/8$. Using the notation from **lowSpread**, see Algorithm 2.5.1$_{p36}$, let $\mathsf{S} =$ **net**(Δ, P) and U the $\Theta(\log(n))$ size sample from S. By Lemma 2.5.3, with high probability $\mathsf{d}^{5/8}(\mathsf{U},\mathsf{S}) \in \left[\mathsf{d}^{1/2}(\mathsf{S}), \mathsf{d}^{3/4}(\mathsf{S})\right]$,

and assume that this is the case. Since S is an Δ-net of P, by Claim 2.4.2, this changes distances by at most 2Δ, which implies that

$$\tau = \mathsf{d}^{5/8}(\mathsf{U},\mathsf{S}) \in \left[\mathsf{d}^{1/2}(\mathsf{P}) - 2\Delta,\, \mathsf{d}^{3/4}(\mathsf{P}) + 2\Delta\right] \subseteq \left[\frac{1}{2}\mathsf{d}^{1/2}(\mathsf{P}),\, 2\mathsf{d}^{3/4}(\mathsf{P})\right] \subseteq \left[\frac{r}{2}, 2R\right].$$

by assumption.

In particular, this implies that all the relevant distances from the computation of this middle value are in the range $[r/2, 2R]$. As such, when computing the $(5/8)$-middle element, it is allowable to not compute precisely values outside this interval, as done by the algorithm. This implies that **lowSpread** indeed returns the value τ, which is the desired approximation.

As for the running time, observe that S is an Δ-net, and an open ball with this radius around any point must be empty of any other net point. In particular, for any point $\mathsf{p} \in \mathsf{S}$, the maximum number of other points that can be in $\mathsf{N}_{\le 2R}(\mathsf{p})$ is $O\Big((R/r)^d\Big) = O\Big(\log^{2d} n\Big)$. Therefore, for a point $\mathsf{p} \in \mathsf{U}$, computing $\mathsf{d}(\mathsf{p},\mathsf{S})$ takes polylogarithmic time (using linear time preprocessing and a grid). Thus, overall, computing the set of distances $\mathtt{D}(\mathsf{U},\mathsf{S})$ takes polylogarithmic time, since $|\mathsf{U}| = \Theta(\log n)$. Hence, the overall running time is $O(n + \operatorname{polylog} n) = O(n)$. ■

smallComp: Small Components

Lemma 2.5.7. *The call to* **smallComp**(ℓ, P) *by* **midNN** *computes a point* $\mathsf{p} \in \mathsf{P}$*, such that* $\mathsf{d}(\mathsf{p},\mathsf{P}) \in \left[\mathsf{d}^{1/32}(\mathsf{P}), \mathsf{d}^{31/32}(\mathsf{P})\right]$*. The procedure succeeds and its running time is* $O(n)$*, with high probability, where* $n = |\mathsf{P}|$*.*

Proof: Computing the components of $\mathcal{C}$ takes linear time, and given $\mathcal{C}$, computing X takes linear time. For each point in the sample U, it takes $O\big(\log^2 n\big)$ time to compute its nearest neighbor since one only has to scan the point's $O(\log^2 n)$ sized connected component in $\mathcal{C}$. Since $|\mathsf{U}| = O(\log n)$, computing the nearest neighbor distances and taking the median takes polylogarithmic time overall.

Now, **smallComp**(ℓ, P) is called by **midNN** only if the following conditions holds:

(C1) $\ell/64\left\lceil \ln^2 n \right\rceil > \mathsf{d}^{1/2}(\mathsf{P})$. Namely, at least half of the points of P have their nearest neighbor in their own cluster in $\mathcal{C}_{\le\rho}(\mathsf{P})$, where $\rho = \ell/\big(8\left\lceil \ln^2 n \right\rceil\big)$. Let Z_1 be the set of these points.

(C2) Let $\mathcal{P}$ be the partition of P computed by **smallComp**. We have that $\mathcal{C}_{\le\rho}(\mathsf{P}) \sqsubseteq \mathcal{P} \sqsubseteq \mathcal{C}_{\le 2\rho}(\mathsf{P})$. By Lemma 2.5.4 (B), at least $(5/8)n$ of the points of P are in clusters of $\mathcal{P}$ that contain at most m points (note, that some of these points might be a cluster containing only a single point). Let Z_2 be the set of these points.

As such, $|\mathsf{Z}_1 \cap \mathsf{Z}_2| \ge n/8$. Namely, at least $n/8$ of the points of P, are in small clusters and these clusters have strictly more than one point. Furthermore, $\mathsf{Z}_1 \cap \mathsf{Z}_2 \subseteq \mathsf{X}$, and we conclude that $|\mathsf{X}| \ge n/8$. Now, by Lemma 2.5.3, with high probability, the returned value is in the range $\mathsf{d}^{1/4}(\mathsf{X}), \ldots, \mathsf{d}^{3/4}(\mathsf{X})$, which implies that the returned value is in the range $\mathsf{d}^{1/32}(\mathsf{P}), \ldots, \mathsf{d}^{31/32}(\mathsf{P})$, as desired. ■

2.5.4 The result

The analysis in the previous section implies the following.

Lemma 2.5.8. *Given a set $\mathsf{P} \subseteq \mathbb{R}^d$ of n points, **midNN**(P) computes, a value x, such that there is a value of $\mathsf{d}^{1/32}(\mathsf{P}), \ldots, \mathsf{d}^{31/32}(\mathsf{P})$ in the interval $[x/c, cx]$, where $c > 1$ is a constant. The running time bound holds with high probability.*

As shown in Section 2.4.3, given a constant factor approximation to some nearest neighbor distance, we can compute the corresponding nearest neighbor distance exactly in linear time using a grid computation. Let **midNNExact** denote the algorithm that performs this additional linear time step after running **midNN**. As argued above, replacing Line 1 and Line 2 in Figure 2.1$_{\text{p18}}$ with "ℓ_i = **midNNExact**(P_{i-1})", turns **ndpAlg** into an algorithm that runs in linear time with high probability.

We thus have the following analogue of Theorem 2.3.19.

Theorem 2.5.9. *For a constant $\varphi > 3/2$, given an instance of a φ-NDP defined by a set of n points in $\mathbb{R}^d$, with a φ-decider with (deterministic or high-probability) running time $O(n)$, one can get a φ-approximation to its optimal value, in $O(n)$ time, with high probability.*

Similarly, for $\varepsilon \in (0, 1/2)$, given an instance of a $(1+\varepsilon)$-NDP defined by a set of n points in $\mathbb{R}^d$, with a $(1+\varepsilon)$-decider having (deterministic, or high probability) running time $O(n/\varepsilon^c)$, then one can $(1+\varepsilon)$-approximate the optimal value for this NDP, in $O(n/\varepsilon^c)$ time, with high probability.

Corollary 2.5.10. *All the applications of Section 2.4 can be modified so that their running time bound holds not only in expectation, but also with high probability.*

Remark 2.5.11. Note, that the above high probability guarantee is in the size of the input. As such, in later iterations of the algorithm, when the input size of the subproblem is no longer polynomial in n (say, the subproblem size is $O(\text{polylog}\, n)$), the high probability guarantee is no longer meaningful.

Fortunately, this is a minor technicality – once the subproblem size drops below $n/\log n$, the remaining running time of **ndpAlg** is $O(n)$, with high probability. This follows by a simple direct argument – by the Chernoff inequality, after $\Theta(\log n)$ iterations, with high probability at least $\log_{16/15} n$ iterations are successful, see the proof of Lemma 2.3.12 for details. After this number of successful iterations, the algorithm terminates. This implies that the remaining running time is $O(n)$, with high probability, once the subproblem size is $O(n/\log n)$.

As such, our new high-probability algorithm is needed only in the initial "heavy" iterations of the **ndpAlg** algorithm, until the subproblem size drops below $O(n/\log n)$.

2.6 Conclusions

There is further research to be done in investigating the technique presented here. For example, looking into the implementation of this new algorithm in both standard settings and distributed settings (e.g., MapReduce). Additionally, since now one can do approximate distance selection in linear time, maybe now one can get a speed up for other algorithms that do (not necessarily point based) distance selection.

Our framework provides a new way of looking at distance based optimization problems, in particular through the lens of nets. We know how to compute nets efficiently for doubling metrics and it seems one can compute approximate nets in near linear time for planar graphs. For example, it seems the new technique implies that approximate k-center clustering in planar graphs can be done in near linear time. This provides fertile ground for future research.

Open question: When is linear time not possible. Another interesting open problem arising out of the high probability linear time algorithm is the following: Given sets P and X in $\mathbb{R}^d$, of size n and $O(\log n)$, respectively, how long does it take to approximate the values in the set $\mathtt{D}(\mathsf{X}, \mathsf{P})$ (i.e., for each point of X we approximate its nearest-neighbor in P). This problem can be easily solved in $O(n \log n)$ time. The open question is whether it is possible solve this problem in linear time.

The algorithm in this chapter implies that we can compute exactly a value (of a specified rank) in this set in $O(n)$ time (even if $|\mathsf{X}|$ is of size n). We currently have an algorithm with running time $O(n \log |\mathsf{X}|) = O(n \log \log n)$ for approximating $\mathtt{D}(\mathsf{X}, \mathsf{P})$.

More generally, this is a proxy to the meta-question of when linear time algorithms do not exist for distance problems induced by points.

Chapter 3

Contour Trees

3.1 Introduction

Geometric data (potentially massive in size) is often represented as a function $f : \mathbb{R}^d \to \mathbb{R}$. Typically, a finite representation is given by considering f to be piecewise linear over some triangulated mesh (i.e. simplicial complex) $\mathbb{M}$ in $\mathbb{R}^d$. *Contour trees* are a specific topological structure used to represent and visualize the function f. For example, consider a rocket engine during a burn. The 3D mesh represents the locations of points in space around the engine, and f represents the temperature at each point. The boundary of the flame is then represented by an isotherm, i.e. a contour.

It is convenient to think of f as a manifold sitting in $\mathbb{R}^{d+1}$, with the last coordinate (i.e. height) given by f. Imagine sweeping the hyperplane $x_{d+1} = h$ with h going from $+\infty$ to $-\infty$. At every instance, the intersection of this plane with f gives a set of connected components, the *contours* at height h. As the sweeping proceeds various events occur: new contours are created or destroyed, contours merge into each other or split into new components, contours acquire or lose handles. More generally, these are the events where the topology of the level set changes. The contour tree is a concise representation of all these events.

If f is smooth, all points where the gradient of f is zero are *critical points*. These points are the "events" where the contour topology changes and which form the vertices of the contour tree. An edge of the contour tree connects two critical points if one event immediately "follows" the other as the sweep plane makes its pass. (We provide formal definitions later.)

Figure 3.1 and Figure 3.2 show examples of simplicial complexes, with heights and their contour trees. Think of the contour tree edges as pointing downwards, so we can talk of the "up-degree" and "down-degree" of vertices. Leaves of down-degree 1 are maxima, because a new contour is created at such points. Leaves of up-degree 1 are minima, because a contour is destroyed there. Up-degree 2 vertices are "joins", where contours merge, and down-degree 2 vertices are "splits". When $d = 2$ (as in Figure 3.1), contours are closed loops. In dimensions $d \geq 3$, we may have internal vertices of both up and down-degree 1. For example, these could be events where contours gain or lose handles.

Suppose we have $f : \mathbb{M} \to \mathbb{R}$, where $\mathbb{M}$ is a triangulated mesh with n vertices, N faces in total, and $t \leq n$ critical points. It is standard to assume that f is PL-Morse (a well-behaved input assumption) and that the maximum degree in the contour tree is 3. A fundamental result in this area is the algorithm of Carr, Snoeyink, and Axen to compute contour trees, which runs in $O(n \log n + N\alpha(N))$ time [CSA00] (where $\alpha(\cdot)$ denotes the inverse Ackermann function). In practical applications, N is typically $\Theta(n)$ (certainly true for $d = 2$). The most expensive operation is an initial sort of all the vertex heights. Chiang *et al.* build on this approach to get a faster algorithm that only sorts the critical vertices, yielding a running time of $O(t \log t + N)$ [CLLR05]. Common applications for contour trees involve turbulent combustion or noisy

data, where the number of critical points is likely to be $\Omega(n)$. So we are still left with the bottleneck of sorting a large set of numbers. Unfortunately, there is a worst-case lower bound of $\Omega(t \log t)$ by Chiang *et al.* [CLLR05], based on a construction of Bajaj *et al.* [BKO+98].

But this is only a worst-case bound. All previous algorithms begin by sorting (at least) the critical points, and hence must pay $\Omega(t \log t)$ for all instances. Can we beat this sorting bound for certain instances, and can we characterize which inputs are hard? Our main result gives an affirmative answer.

Definition 3.1.1. For a contour tree T, a *leaf path* is any path in T containing a leaf, which is also monotonic in the height values of its vertices. Then a *path decomposition*, $P(T)$, is a partition of the vertices of T into a set of vertex disjoint leaf paths.

Theorem 3.1.2. *Consider a PL-Morse* $f : \mathbb{M} \to \mathbb{R}$, where the contour tree T has maximum degree 3. There is a deterministic algorithm to compute T whose running time is $O(\sum_{p \in P(T)} |p| \log |p| + t\alpha(t) + N)$, where $P(T)$ is a specific path decomposition (constructed implicitly by the algorithm).*

The number of comparisons made is $O(\sum_{p \in P(T)} |p| \log |p| + N)$.

In particular, any comparisons made are only between ancestors and descendants in the contour tree, while previous algorithms compared everything.

Let us interpret the running time. For any path decomposition $P(T)$, obviously $\sum_{p \in P(T)} |p| = |T| = t$, and so the running time is $O(t \log t + N)$. But this is tight only when T is tall and thin. A better upper bound is $O(t \log D + t\alpha(t) + N)$, where D is the diameter of T. This is clearly an improvement when the tree is short and fat, but it can be even better. Consider a contour tree that is constant balanced, meaning that the ratio of sizes of sibling subtrees is $\Theta(1)$. A calculation yields that for any decomposition of such a contour tree, our bound is $O(t\alpha(t) + n)$. Consider the example of Figure 3.2 extended to a larger binary tree both above and below, so $t = \Omega(n)$. The running time of the algorithm of Theorem 3.1.2 is $O(n\alpha(n))$, whereas all previous algorithms require $\Omega(n \log n)$ time.

We complement Theorem 3.1.2 with a lower bound, suggesting that the comparison model complexity of $\sum_{p \in P(T)} |p| \log |p|$ is inherent for any contour tree algorithm. The exact phrasing of the lower bound requires some care. Consider the $d = 2$ (terrain) case. Let a path decomposition P be called *valid* if it is actually output by the algorithm on some input.

Theorem 3.1.3. *Consider a valid path decomposition P. There exists a family $\mathbf{F}_P$ of terrains (i.e. $d = 2$) with the following properties. Any contour tree algorithm makes $\Omega(\sum_{p \in P} |p| \log |p|)$ comparisons in the worst case over $\mathbf{F}_P$. Furthermore, for any terrain in $\mathbf{F}_P$, the algorithm of Theorem 3.1.2 makes $O(\sum_{p \in P} |p| \log |p|)$ comparisons.*

In other words, for any path decomposition, P, we can design a hard family where any algorithm requires $\sum_{p \in P} |p| \log |p|$ time in the worst case, and our algorithm matches this bound.

Beyond worst-case analysis. Our result provides understanding of the complexity of contour tree construction beyond the standard worst-case analysis. Our theorem and lower bound characterize hard instances. Trees that are short and fat are easy to compute, whereas trees with long paths are hard. This is tangentially related to the concept of instance-optimal algorithms, as defined by Afshani *et al.* [ABC09]. Their notion of

*This is a standard "well-behaved" input assumption that implies that all critical points have distinct values, and the contour tree has max-degree 3. Our algorithm does not need this, but it simplifies the proof and presentation.

optimality is far more refined that what we prove, but it shares the flavor of understanding the spectrum of easy to hard instances. In general, our analysis forces one to look closely at the relationship between sorting and the contour tree problem.

3.1.1 Previous Work

Contour trees were first used to study terrain maps by Boyell and Ruston, and Freeman and Morse [BR63, FM67]. They were used for isoline extraction in geometric data by van Krevald *et al.* [vKvOB+97], who provided the first formal algorithm. Contour trees and related topological structures have been applied in analysis of fluid mixing, combustion simulations, and studying chemical systems [LBM+06, BWP+10, BWH+11, BWT+11, MGB+11]. Carr's thesis [Car04] gives various applications of contour trees for data visualization and is an excellent reference for contour tree definitions and algorithms.

There are numerous algorithms for computing contour trees. An $O(N \log N)$ time algorithm for functions over 2D meshes and an $O(N^2)$ algorithm for higher dimensions, was presented by van Kreveld *et al.* [vKvOB+97]. Tarasov and Vyalya [TV98] improved the running time to $O(N \log N)$ for the 3D case. The influential paper of Carr *et al.* [CSA00] improved the running time for all dimensions to $O(n \log n + N\alpha(N))$. This result forms the basis of subsequent practical and theoretical work in computational topology applications. Pascucci and Cole-McLaughlin [PCM02] provided an $O(n + t \log n)$ time algorithm for 3-dimensional structured meshes. Chiang *et al.* [CLLR05] provide an unconditional $O(N + t \log t)$ algorithm. The latter result crucially uses monotone paths to improve the running time, and our algorithm repeatedly exploits numerous properties of monotone paths. Carr's thesis shows relationships between monotone paths in $\mathbb{M}$ and its contour tree $\mathcal{C}(\mathbb{M})$.

Contour trees are a special case of Reeb graphs, a general topological representation for real-valued functions on any manifold. We refer the reader to Chapter 6 in [HE10] for more details. Algorithms for computing Reeb graphs is an active topic of research [SK91, CMEH+03, PSBM07, DN09, HWW10, Par12], where two results explicitly reduce to computing contour trees [TGSP09, DN13].

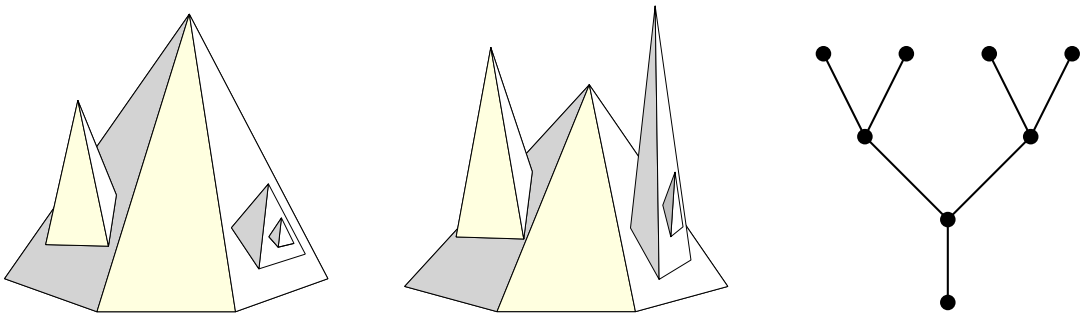


Figure 3.1: Two surfaces with different orderings of the maxima, but the same contour tree.

3.2 Contour tree basics

We detail the basic definitions about contour trees. We follow the terminology of Chapter 6 of Carr's thesis [Car04]. All our assumptions and definitions are standard for results in this area, though there is some variability in notation. The input is a continuous piecewise-linear (PL) function $f : \mathbb{M} \to \mathbb{R}$, where $\mathbb{M}$ is a

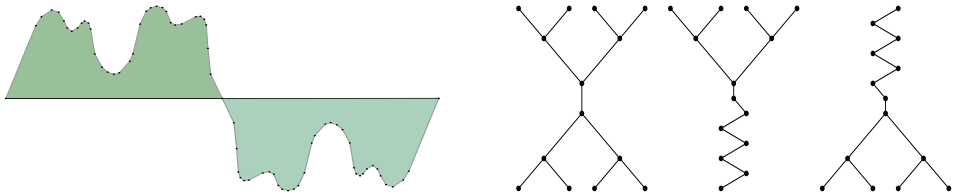


Figure 3.2: On left, a surface with a balanced contour tree, but whose join and split trees have long tails. On right (from left to right), the contour, join and split trees.

simply connected and fully triangulated simplicial complex in $\mathbb{R}^d$, except for specially designated *boundary facets*. So f is defined only on the vertices of $\mathbb{M}$, and all other values are obtained by linear interpolation.

We assume that f has distinct values on all vertices, except for boundaries, as expressed in the following constraint, which intuitively requires that all boundaries look like (local) horizontal cuts.

Definition 3.2.1. The function f is *boundary critical* if the following holds. Consider a boundary facet F. All vertices of F have the same function value. Furthermore, all neighbors of vertices in F, which are not also in F itself, either have all function values strictly greater than or all function values strictly less than the function value at F.

This is convenient, as we can now assume that f is defined on $\mathbb{R}^d$. Any point inside a boundary facet has a well-defined height, including the infinite facet, which is required to be a boundary facet. However, we allow for other boundary facets, to capture the resulting surface pieces after our algorithm make a horizontal cut.

We think of the dimension, d, as constant, and assume that $\mathbb{M}$ is represented in a data structure that allows constant-time access to neighboring simplices in $\mathbb{M}$ (e.g. [BM12]). (This is analogous to a doubly connected edge list, but for higher dimensions.) Observe that $f : \mathbb{M} \to \mathbb{R}$ can be thought of as a d-dimensional simplicial complex living in $\mathbb{R}^{d+1}$, where $f(x)$ is the "height" of a point $x \in \mathbb{M}$, which is encoded in the representation of $\mathbb{M}$. Specifically, rather than writing our input as $(\mathbb{M}, f)$, we abuse notation and typically just write $\mathbb{M}$ to denote the lifted complex.

Definition 3.2.2. The *level set* at value h is the set $\{x | f(x) = h\}$. A *contour* is a maximal connected component of a level set. An *h-contour* is a contour where f-values are h.

Note that a contour that does not contain a boundary is itself a simplicial complex of one dimension lower, and is represented (in our algorithms) as such. We let δ and ε denote infinitesimals. Let $B_\varepsilon(x)$ denote a ball of radius ε around x, and let $f|B_\varepsilon(x)$ be the restriction of f to $B_\varepsilon(x)$.

Definition 3.2.3. The *Morse up-degree* of x is the number of $(f(x) + \delta)$-contours of $f|B_\varepsilon(x)$ as $\delta, \varepsilon \to 0^+$. The *Morse down-degree* is the number of $(f(x) - \delta)$-contours of $f|B_\varepsilon(x)$ as $\delta, \varepsilon \to 0^+$.

We categorize points depending on the local changes in topology.

Definition 3.2.4. A point x is categorized as follows.

- *Regular:* Both Morse up-degree and down-degrees are 1.
- *Maximum:* Morse up-degree is 0.
- *Minimum:* Morse down-degree is 0.
- *Morse Join:* Morse up-degree is strictly greater than 1.

- *Morse Split:* Morse down-degree is strictly greater than 1.

Non-regular points are *critical*. Morse joins and splits are *saddles*. Maxima and minima are *extrema*.

The set of critical points is denoted by $\mathcal{V}(f)$. Because f is piecewise-linear, all critical points are vertices in $\mathbb{M}$. A value h is called *critical*, if $f(v) = h$, for some $v \in \mathcal{V}(f)$. A contour is called *critical*, if it contains a critical point, and it is called *regular* otherwise.

The critical points are exactly where the topology of level sets change. By assuming that our manifold is boundary critical, the vertices on a given boundary are either collectively all maxima or all minima. We abuse notation and refer to this entire set of vertices as a maximum or minimum.

Definition 3.2.5. Two regular contours ψ and ψ' are *equivalent* if there exists an f-monotone path p connecting a point in ψ to ψ', such that no $x \in p$ belongs to a critical contour.

This equivalence relation gives a set *contour classes*. Every such class maps to intervals of the form $(f(x_i), f(x_j))$, where x_i, x_j are critical points. Such a class is said to be created at x_i and destroyed at x_j.

Definition 3.2.6. The *contour tree* is the graph on vertex set $\mathcal{V} = \mathcal{V}(f)$, where edges are formed as follows. For every contour class that is created at v_i and destroyed v_j, there is an edge (v_i, v_j). (Conventionally, edges are directed from higher to lower function value.)

We denote the contour tree of $\mathbb{M}$ by $\mathcal{C}(\mathbb{M})$. The corresponding node and edge sets are denoted as $\mathcal{V}(\cdot)$ and $\mathcal{E}(\cdot)$. It is not immediately obvious that this graph is a tree, but alternate definitions of the contour tree in [CSA00] imply this is a tree. Since this tree has height values associated with the vertices, we can talk about up-degrees and down-degrees in $\mathcal{C}(\mathbb{M})$. We assume there are no "multi-saddles", so up and down-degrees are at most 2, and the total degree is at most 3. This is again a standard assumption in topological algorithms and can be achieved by vertex unfolding (Section 6.3 in [HE10]).

3.2.1 Some technical remarks

Note that if one intersects $\mathbb{M}$ with a given ball B, then a single contour in $\mathbb{M}$ might be split into more than one contour in the intersection. In particular, two $(f(x) + \delta)$-contours of $f|_{B_\varepsilon(x)}$, given by Definition 3.2.3, might actually be the same contour in $\mathbb{M}$. Alternatively, one can define the up-degree (as opposed to *Morse* up-degree) as the number of $(f(x) + \delta)$-contours (in the full $\mathbb{M}$) that intersect $B_\varepsilon(x)$, a potentially smaller number. This up-degree is exactly the up-degree of x in $\mathcal{C}(\mathbb{M})$. (Analogously, for down-degree.) When the Morse up-degree is 2 but the up-degree is 1, the topology of the level set changes but not by the number of connected components changing. For example, when $d = 3$ this is equivalent to the contour gaining a handle. When $d = 2$, this distinction is not necessary, since any point with Morse degree strictly greater than 1 will have degree strictly greater than 1 in $\mathcal{C}(\mathbb{M})$.

As Carr points out in Chapter 6 of his thesis, the term contour tree can be used for a family of related structures. Every vertex in $\mathbb{M}$ is associated with an edge in $\mathcal{C}(\mathbb{M})$, and sometimes the vertex is explicitly placed in $\mathcal{C}(\mathbb{M})$ (by subdividing the respective edge). This is referred to as augmenting the contour tree, and it is common to augment $\mathcal{C}(\mathbb{M})$ with all vertices. Alternatively, one can smooth out all vertices of up-degree and down-degree 1 to get the unaugmented contour tree. (For $d = 2$, there are no such vertices in $\mathcal{C}(\mathbb{M})$.) The contour tree of Definition 3.2.6 is the typical definition in all results on output-sensitive contour trees, and is the smallest tree that contains all the topological changes of level sets. Theorem 3.1.2 is applicable for any augmentation of $\mathcal{C}(\mathbb{M})$ with a predefined set of vertices, though we will not delve into these aspects in this paper.

3.3 A tour of the new contour tree algorithm

Our final algorithm is quite technical and has numerous moving parts. However, for the $d = 2$ case, where the input is just a triangulated terrain, the main ideas of the parts of the algorithm can be explained clearly. Therefore, here we first provide a high level view of the entire result, which we view as essential for understanding our approach.

In the interest of presentation, the definitions and theorem statements in this section will slightly differ from those in the main body. They may also differ from the original definitions proposed in earlier work.

Do not globally sort: Arguably, the starting point for this work is Figure 3.1. We have two terrains with exactly the same contour tree, but different orderings of (heights of) the critical points. Turning it around, we cannot deduce the full height ordering of critical points from the contour tree. Sorting all critical points is computationally unnecessary for constructing the contour tree. In Figure 3.2, the contour tree consists of two balanced binary trees, one of the joins, another of the splits. Again, it is not necessary to know the relative ordering between the mounds on the left (or among the depressions on the right) to compute the contour tree. Yet some ordering information is necessary: on the left, the little valleys are higher than the big central valley, and this is reflected in the contour tree. Leaf paths in the contour tree have points in sorted order, but incomparable points in the tree are unconstrained. How do we sort exactly what is required, without knowing the contour tree in advance?

3.3.1 Breaking $\mathbb{M}$ into simpler pieces

Let us begin with the algorithm of Carr, Snoeyink, and Axen [CSA00]. The key insight is to build two different trees, called the join and split trees, and then merge them together into the contour tree. Consider sweeping down via the hyperplane $x_{d+1} = h$ and taking the *superlevel* sets. These are the connected components of the portion of $\mathbb{M}$ above height h. For a terrain, the superlevel sets are a collection of "mounds". As we sweep downwards, these mounds keep joining each other, until finally, we end up with all of $\mathbb{M}$. The join tree tracks exactly these events. Formally, let $\mathbb{M}_v^+$ denote the simplicial complex induced on the subset of vertices which are higher than v.

Definition 3.3.1. The *critical join tree* $\mathcal{J}_C(\mathbb{M})$ is built on the set $\mathcal{V}$ of all critical points.[†] The directed edge (u, v) is present when u is the smallest valued vertex in $\mathcal{V}$ in a connected component of $\mathbb{M}_v^+$ and v is connected to this component (in $\mathbb{M}$).

Abusing notation, for now we will just call this the join tree. (In Section 3.6 we carefully handle the distinction between the join tree and the critical join tree.) Refer to Figure 3.2 for the join tree of a terrain. Note that nothing happens at splits, but these are still put as vertices in the join tree. They simply form a long path. The split tree is obtained by simply inverting this procedure, sweeping upwards and tracking sublevel sets.

A major insight of [CSA00] is an ingeniously simple linear time procedure to construct the contour tree from the join and split trees. So the bottleneck is computing these trees. Observe in Figure 3.2 that the split vertices form a long path in the join tree (and vice versa). Therefore, constructing these trees forces a global sort of the splits, an unnecessary computation for the contour tree. Unfortunately, in general (i.e.

[†]Later, in Definition 3.6.3, we define the critical join tree on a certain subset of $\mathcal{V}$.

unlike Figure 3.2) the heights of joins and splits may be interleaved in a complex manner, and hence the final merging of [CSA00] to get the contour tree requires having the split vertices in the join tree. Without this, it is not clear how to get a consistent view of both joins and splits, required for the contour tree.

Our aim is to break $\mathbb{M}$ into smaller pieces, where this unnecessary computation can be avoided.

Contour surgery: We first need a divide-and-conquer lemma. Any contour ϕ can be associated with an edge e of the contour tree. Suppose we "cut" $\mathbb{M}$ along this contour. We prove that $\mathbb{M}$ is split into two disconnected pieces, such the contour trees of these pieces is obtained by simply cutting e in $\mathcal{C}(\mathbb{M})$. Alternatively, the contour trees of these pieces can be glued together to get $\mathcal{C}(\mathbb{M})$. This is not particularly surprising, and is fairly easy to prove with the right definitions. The idea of loop surgery has been used to reduce Reeb graphs to contour trees [TGSP09, DN13]. Nonetheless, our theorem appears to be new and works for all dimensions.

Figure 3.3: On left, downward rain spilling only (each shade of gray represents a piece created by each different spilling), producing a grid. Note we are assuming raining was done in reverse sorted order of maxima. On right, flipping the direction of rain spilling.

Cutting $\mathbb{M}$ into extremum dominant pieces: We define a simplicial complex endowed with a height to be *minimum dominant* if there exists only a single minimum. (Our real definition is more complicated, and involves simplicial complexes that allow additional "trivial" minima.) In such a complex, there exists a non-ascending path from any point to this unique minimum. Analogously, we can define maximum dominant complexes, and both are collectively called extremum dominant.

We will cut $\mathbb{M}$ into disjoint extremum dominant pieces, in linear time. One way to think of our procedure is a meteorological analogy. Take an arbitrary maximum x, and imagine torrential rain at the maximum. The water flows down, wetting any point that has a non-ascending path from x. We end up with two portions, the wet part of $\mathbb{M}$ and the dry part. This is similar to *watershed* algorithms used for image segmentation [RM00]. The wet part is obviously connected, while there may be numerous disconnected dry parts. The interface between the dry and wet parts is a set of contours[‡], given by the "water line". The wet part is clearly maximum dominant, since all wet points have a non-descending path to x. So we can simply cut along the interface contours to get the wet maximum dominant piece $\mathbb{M}'$. By our contour surgery theorem, we are left with a set of disconnected dry parts, and we can recur this procedure on them.

But here's the catch. Every time we cut $\mathbb{M}$ along a contour, we potentially increase the complexity of $\mathbb{M}$. Water flows in the interior of faces, and the interface will naturally cut some faces. Each cut introduces new vertices, and a poor choice of repeated raining leads to a large increase in complexity. Consider the left

[‡]Technically, they are not contours, but rather the limits of sequences of contours.

of Figure 3.3. Each raining produces a single wet and dry piece, and each cut introduces many new vertices. If we wanted to partition this terrain into maximum dominant simplicial complexes, the final complexity would be forbiddingly large.

A simple trick saves the day. Unlike reality, we can choose rain to flow solely downwards or solely upwards. Apply the procedure above to get a single wet maximum dominant $\mathbb{M}'$ and a set of dry pieces. Observe that a single dry piece $\mathbb{N}$ is boundary critical with the newly introduced boundary ϕ (the wet-dry interface) behaving as a minimum. So we can rain upwards from this minimum, and get a *minimum dominant* portion $\mathbb{N}'$. This ensures that the new interface (after applying the procedure on $\mathbb{N}$) does not cut any face previously cut by ϕ. For each of the new dry pieces, the newly introduced boundary is now a maximum. So we rain downwards from there. More formally, we alternate between raining upwards and downwards as we go down the recursion tree. We can prove that an original face of $\mathbb{M}$ is cut at most once, so the final complexity can be bounded. In Figure 3.3, this procedure would yield two pieces, one maximum dominant, and one minimum dominant.

Using the contour surgery theorem previously discussed, we can build the contour tree of $\mathbb{M}$ from the contour trees of the various pieces created. All in all, we prove the following theorem.

Theorem 3.3.2. *There is an $O(N)$ time procedure that cuts $\mathbb{M}$ into extremum dominant simplicial complexes $\mathbb{M}_1, \mathbb{M}_2, \ldots$. Furthermore, given the set of contour trees $\{\mathcal{C}(\mathbb{M}_i)\}$, $\mathcal{C}(\mathbb{M})$ can be constructed in $O(N)$ time.*

Extremum dominance simplifies contour trees: We will focus on minimum dominant simplicial complexes $\mathbb{M}$. By Theorem 3.3.2, it suffices to design an algorithm for contour trees on such inputs. For the $d = 2$ case, it helps to visualize such an input as a terrain with no minima, except at a unique boundary face (think of a large boundary triangle that is the boundary). All the valleys in such a terrain are necessarily joins, and there can be no splits. Look at Figure 3.2. The portion on the left is minimum dominant in exactly this fashion. More formally, $\mathbb{M}_v^-$ is connected for all v, so there are no splits.

We can prove that the split tree is just a path, and the contour tree is exactly the join tree. The formal argument is a little involved, and we employ the merging procedure of [CSA00] to get a proof. We demonstrate that the merging procedure will actually just output the join tree, so we do not need to actually compute the split tree. (The real definition of minimum dominant is a little more complicated, so the contour tree is more than just the join tree. But computationally, it suffices to construct the join tree.)

We stress the importance of this step for our approach. Given the algorithm of [CSA00], one may think that it suffices to design faster algorithms for join trees. But this cannot give the sort of optimality we hope for. Again, consider Figure 3.2. Any algorithm to construct the true join tree must construct the path of splits, which implies sorting all of them. It is absolutely necessary to cut $\mathbb{M}$ into pieces where the cost of building the join tree can be related to that of building $\mathcal{C}(\mathbb{M})$.

3.3.2 Join trees from painted mountaintops

Arguably, everything up to this point is a preamble for the main result: a faster algorithm for join trees. Our algorithm does not require the input to be extremum dominant. This is only required to relate the join trees to the contour tree of the initial input $\mathbb{M}$. For convenience, we use $\mathbb{N}$ to denote the input here. Note that in Definition 3.3.1, the join tree is defined purely combinatorially in terms of the 1-skeleton (the underlying graph) of $\mathbb{N}$.

The join tree $\mathcal{J}_C(\mathbb{N})$ is a rooted tree with the dominant minimum at the root, and we direct edges downwards (towards the root). So it makes sense to talk of comparable vs incomparable vertices. We arrive at the main challenge: how to sort only the comparable critical points, without constructing the join tree? The join tree algorithm of [CSA00] is a typical event-based computational geometry algorithm. We have to step away from this viewpoint to avoid the global sort.

The key idea is *paint spilling*. Start with each maximum having a large can of paint, with distinct colors for each maximum. In arbitrary order, we spill paint from each maximum, wait till it flows down, then spill from the next, etc. Paint is viscous, and only flows down edges. *It does not paint the interior of higher dimensional faces.* That is, this process is restricted to the 1-skeleton of $\mathbb{N}$. Furthermore, our paints do not mix, so each edge receives a unique color, decided by the first paint to reach it. In the following, $[n]$ denotes the set $\{1, \ldots, n\}$, for any natural number n.

Definition 3.3.3. Let the 1-skeleton of $\mathbb{N}$ have edge set E and maxima X. A *painting* of $\mathbb{N}$ is a map $\chi : X \cup E \to [|X|]$ with the following property. Consider an edge e. There exists a descending path from some maximum x to e consisting of edges in E, such that all edges along this path have the same color as x.

An *initial* painting has the additional property that the restriction $\chi : X \to [|X|]$ is a bijection.

Note that a painting colors edges, and not vertices (except for maxima). Our definition also does not require the timing aspect of iterating over colors, though that is one way of painting $\mathbb{N}$. We begin with an initial painting, since all maximum colors are distinct. A few comments on paint vs water. The interface between two regions of different color is *not* a contour, and so we cannot apply the divide-and-conquer approach of contour surgery. On the other hand, painting does not cut $\mathbb{N}$, so there is no increase in complexity. Clearly, an initial painting can be constructed in $O(N)$ time. This is the tradeoff between water and paint. Water allows for an easy divide-and-conquer, at the cost of more complexity in the input. For an extremum dominant input, using water to divide the input $\mathbb{N}$ raises the complexity too much.

Our algorithm incrementally builds $\mathcal{J}_C(\mathbb{M})$ from the leaves (maxima) to the root (dominant minimum). We say that vertex v is *touched* by color c, if there is a c-colored edge with lower endpoint v. Let us focus on an initial painting, where the colors have 1-1 correspondence with the maxima. Refer to the left part of Figure 3.4. Consider two sibling leaves ℓ_1, ℓ_2 and their common parent v. The leaves are maxima, and v is a join that "merges" ℓ_1, ℓ_2. In that case, there are "mounds" corresponding to ℓ_1 and ℓ_2 that merge at a valley v. Suppose this was the entire input, and ℓ_1 was colored blue and ℓ_2 was colored red. Both mounds are colored completely blue or red, while v is touched by both colors. So this indicates that v joins the blue maximum and red maximum in $\mathcal{J}_C(\mathbb{M})$.

This is precisely how we hope to exploit the information in the painting. We prove later that when some join v has all incident edges with exactly two colors, the corresponding maxima (of those colors) are exactly the children of v in $\mathcal{J}_C(\mathbb{M})$. To proceed further, we "merge" the colors red and blue into a new color, purple. In other words, we replace all red and blue edges by purple edges. This indicates that the red and blue maxima have been handled. Imagine flattening the red and blue mounds until reaching v, so that the former join v is now a new maximum, from which purple paint is poured. In terms of $\mathcal{J}_C(\mathbb{M})$, this is equivalent to removing leaves ℓ_1 and ℓ_2, and making v a new leaf. Alternatively, $\mathcal{J}_C(\mathbb{M})$ has been constructed up to v, and it remains to determine v's parent. The merging of the colors is not explicitly performed as that would be too expensive; instead we maintain a union-find data structure for that.

Of course, things are more complicated when there are other mounds. There may be a yellow mound, corresponding to ℓ_3 that joins with the blue mound higher up at some vertex u (see the right part of

Figure 3.4). In $\mathcal{J}_C(\mathbb{M})$, ℓ_1 and ℓ_3 are sibling leaves, and ℓ_2 is a sibling of some ancestor of these leaves. So we cannot merge red and blue, until yellow and blue merge. Naturally, we use priority queues to handle this issue. We know that u must also be touched by blue. So all critical vertices touched by blue are put into a priority queue keyed by height, and vertices are handled in that order.

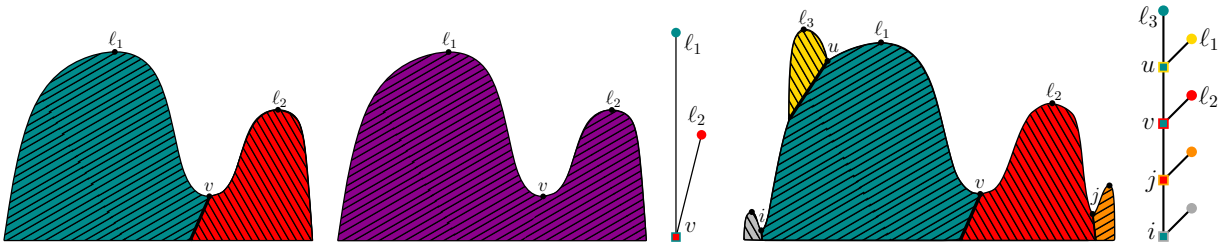


Figure 3.4: On the left, red and blue merge to make purple, followed by the contour tree with initial colors. On the right, additional maxima and the resulting contour tree.

What happens when finally blue and red join at v? We merge the two colors, but now have blue and red queues of critical vertices, which also need to be merged to get a consistent painting. This necessitates using a priority queue with efficient merges. Specifically, we use *binomial heaps* [Vui78], as they provide logarithmic time merges and deletes (though other similar heaps work). We stress that the feasibility of the entire approach hinges on the use of such an efficient heap structure.

In this discussion, we ignored an annoying problem. Vertices may actually be touched by numerous colors, not just one or two as assumed above. A simple solution would be to insert vertices into heaps corresponding to all colors touching it. But there could be super-constant numbers of copies of a vertex, and handling all these copies would lead to extra overhead. We show that it suffices to simply put each vertex v into at most two heaps, one for each "side" of a possible join. We are guaranteed that when v needs to be processed, all edges have at most 2 colors, because of all the color merges that previously occurred.

The running time analysis: Relating the running time to a path decomposition is the most technical part of this chapter. All the non-heap operations can be easily bounded by $O(t\alpha(t) + N)$ (the $t\alpha(t)$ is from the union-find data structure for colors). It is not hard to argue that at all times, any heap always contains a subset of a leaf to root path. Unfortunately, this subset is *not* contiguous, as the right part of Figure 3.4 shows. Specifically, in this figure the far left saddle (labeled i) is hit by blue paint. However, there is another saddle on the far right (labeled j) which is not hit by blue paint. Since this far right saddle is slightly higher than the far left one, it will merge into the component containing the blue mound (and also the yellow and red mounds) before the far left one. Hence, the vertices initially touched by blue are not contiguous in the contour tree.

Nonetheless, we can get a non-trivial (but non-optimal) bound. Let d_v denote the distance to the root for vertex v in the join tree. The total cost (of the heap operations) is at most $\sum_v \log d_v$. This immediately proves a bound of $O(t \log D)$, where D is the maximum distance to the root, an improvement over previous work. But this bound is non-optimal. For a balanced binary tree, this bound is $O(t \log \log t)$, whereas the cost of any path decomposition is $O(t)$.

The cost of heap operations depends on heap sizes, which keep changing because of the repeated merging. This causes the analysis to be technically challenging. There are situations where the initial heap sizes are small, but they eventually merge to create larger heaps. We employ a variant of *heavy path decompositions*,

first used by Sleator and Tarjan for analyzing link/cut trees [ST83]. The final analysis charges expensive heap operations to long paths in the decomposition.

3.3.3 The lower bound

Consider a contour tree T and the path decomposition $P(T)$ used to bound the running time. Denoting $cost(P(T)) = \sum_{p \in P(T)} |p| \log |p|$, we construct a set of $\prod_{p \in P(T)} |p|!$ functions on a fixed domain such that each function has a distinct (labeled) contour tree. By a simple entropy argument, any algebraic decision tree that correctly computes the contour tree on all instances requires worst case $\Omega(cost(P(T)))$ time. We prove that our algorithm makes $\Theta(cost(P(T)))$ comparisons on all these instances.

We have a fairly simple construction that works for terrains. In $P(T)$, consider the path p that involves the root. The base of the construction is a conical "tent", and there will be $|p|$ triangular faces that will each have a saddle. The heights of these saddles can be varied arbitrarily, and that will give $|p|!$ different choices. Each of these saddles will be connected to a recursive construction involving other paths in $P(T)$. Effectively, one can think of tiny tents that are sticking out of each face of the main tent. The contour trees of these tiny tents attach to a main branch of length $|p|$. Working out the details, we get $\prod_{p \in P(T)} |p|!$ terrains each with a distinct contour tree.

3.4 Divide and conquer through contour surgery

The cutting operation: We define a "cut" operation on $f : \mathbb{M} \to \mathbb{R}$ that cuts along a regular contour to create a new simplicial complex with an added boundary. Given a contour ϕ, roughly speaking, this constructs the simplicial complex $\mathbb{M} \setminus \phi$. We will always enforce the condition that ϕ never passes through a vertex of $\mathbb{M}$. Again, we use ε for an infinitesimally small value. We denote ϕ^+ (resp. ϕ^-) to be the contour at value $f(\phi) + \varepsilon$ (resp. $f(\phi) - \varepsilon$), which is at distance ε from ϕ.

An h-contour is achieved by intersecting $\mathbb{M}$ with the hyperplane $x_{d+1} = h$ and taking a connected component. (Think of the $d + 1$-dimension as height.) Given some point x on an h-contour ϕ, we can walk along $\mathbb{M}$ from x to determine ϕ. We can "cut" along ϕ to get a new (possibly) disconnected simplicial complex $\mathbb{M}'$. This is achieved by splitting every face F that ϕ intersects into an "upper" face and "lower" face. Algorithmically, we cut F with ϕ^+ and take everything above ϕ^+ in F to make the upper face. Analogously, we cut with ϕ^- to get the lower face. The faces are then triangulated to ensure that they are all simplices. This creates the two new boundaries ϕ^+ and ϕ^-, and we maintain the property of constant f-value at a boundary.

Note that by assumption ϕ cannot cut a boundary face, and moreover all non-boundary faces have constant size. Therefore, this process takes time linear in $|\phi|$, the number of faces ϕ intersects. This new simplicial complex is denoted by $\texttt{cut}(\phi, \mathbb{M})$. We now describe a high-level approach to construct $\mathcal{C}(\mathbb{M})$ using this cutting procedure.

$\texttt{surgery}(\mathbb{M}, \phi)$

1. Let $\mathbb{M}' = \texttt{cut}(\mathbb{M}, \phi)$.
2. Construct $\mathcal{C}(\mathbb{M}')$ and let A, B be the nodes corresponding to the new boundaries created in $\mathbb{M}'$. (One is a minimum and the other is maximum.)
3. Since A, B are leaves, they each have unique neighbors A' and B', respectively. Insert edge (A', B') and delete A, B to obtain $\mathcal{C}(\mathbb{M})$.

Theorem 3.4.1. *For any regular contour ϕ, the output of* $\texttt{surgery}(\mathbb{M}, \phi)$ *is* $\mathcal{C}(\mathbb{M})$.

To prove the above theorem, we require some theorems from [Car04] (Theorems 6.6 and 6.7) that map paths in $\mathcal{C}(\mathbb{M})$ to $\mathbb{M}$.

Theorem 3.4.2. *For every path P in $\mathbb{M}$, there exists a path Q in the contour tree corresponding to the contours passing through points in P. For every path Q in the contour tree, there exists at least one path P in $\mathbb{M}$ through points present in contours involving Q.*

Theorem 3.4.3. *For every monotone path P in $\mathbb{M}$, there exists a monotone path Q in the contour tree to which P maps (as in the previous theorem), and vice versa.*

Theorem 3.4.1 is a direct consequence of the following lemma.

Lemma 3.4.4. *Consider a regular contour ϕ contained in a contour class (of an edge of $\mathcal{C}(\mathbb{M})$) (u, v) and denote $\mathbb{M}' = \texttt{cut}(\mathbb{M}, \phi)$. Then $\mathcal{V}(\mathcal{C}(\mathbb{M}')) = \{\phi^+, \phi^-\} \cup \mathcal{V}(\mathbb{M})$ and $\mathcal{E}(\mathcal{C}(\mathbb{M}')) = \{(u, \phi^+), (\phi^-, v)\} \cup (\mathcal{E}(\mathbb{M}) \setminus (u, v))$.*

Proof: First observe that since ϕ is a regular contour, the vertex set in the complex $\mathbb{M}'$ is the same as the vertex set in $\mathbb{M}$, except with the addition of the newly created vertices on ϕ^+ and ϕ^-. Moreover, $\texttt{cut}(\mathbb{M}, \phi)$ does not affect the local neighborhood of any vertex in $\mathbb{M}$. Therefore since a vertex being critical is a local condition, with the exception of new boundary vertices, the critical vertices in $\mathbb{M}$ and $\mathbb{M}'$ are the same. Finally, the new vertices on ϕ^+ and ϕ^- collectively behave as a minimum and maximum, respectively, and so $\mathcal{V}(\mathcal{C}(\mathbb{M}')) = \{\phi^+, \phi^-\} \cup \mathcal{V}(\mathbb{M})$.

Now consider the edge sets of the contour trees. Any contour class in $\mathbb{M}'$ (i.e. edge in $\mathcal{C}(\mathbb{M}')$) that does not involve ϕ^+ or ϕ^- is also a contour class in $\mathbb{M}$. Furthermore, a maximal contour class satisfying these properties is also maximal in $\mathbb{M}$. So all edges of $\mathcal{C}(\mathbb{M}')$ that do not involve ϕ^+ or ϕ^- are edges of $\mathcal{C}(\mathbb{M})$. Analogously, every edge of $\mathcal{C}(\mathbb{M})$ not involving ϕ is an edge of $\mathcal{C}(\mathbb{M}')$.

Consider the contour class corresponding to edge (u, v) of $\mathcal{C}(\mathbb{M})$. There is a natural ordering of the contours by function value, ranging from $f(u)$ to $f(v)$. All contours in this class "above" ϕ form a maximal contour class in $\mathbb{M}'$, represented by edge (u, ϕ^+). Analogously, there is another contour class represented by edge (ϕ^-, v). We have now accounted for all contours in $\mathcal{C}(\mathbb{M}')$, completing the proof. ■

A useful corollary of this lemma shows that a contour actually splits the simplicial complex into two disconnected complexes.

Theorem 3.4.5. $\texttt{cut}(\mathbb{M}, \phi)$ *consists of two disconnected simplicial complexes.*

Proof: Denote (as in Lemma 3.4.4) the edge containing ϕ to be (u, v). Suppose for contradiction that there is a path between vertices u and v in $\mathbb{M}' = \texttt{cut}(\mathbb{M}, \phi)$. By Theorem 3.4.2, there is a path in $\mathcal{C}(\mathbb{M}')$ between u and v. Since ϕ^+ and ϕ^- are leaves in $\mathcal{C}(\mathbb{M}')$, this path cannot use their incident edges. Therefore by Lemma 3.4.4, all the edges of this path are in $\mathcal{E}(\mathcal{C}(\mathbb{M})) \setminus (u, v)$. So we get a cycle in $\mathcal{C}(\mathbb{M})$, a contradiction. To show that there are exactly two connected components in $\texttt{cut}(\mathbb{M}, \phi)$, it suffices to see that $\mathcal{C}(\mathbb{M}')$ has two connected components (by Lemma 3.4.4) and then applying Theorem 3.4.2. ■

3.5 Raining to partition $\mathbb{M}$

In this section, we describe a linear time procedure that partitions $\mathbb{M}$ into special *extremum dominant* simplicial complexes.

Definition 3.5.1. A simplicial complex is *minimum dominant* if there exists a minimum x such that every non-minimal *vertex* in the manifold has a non-ascending path to x. Analogously define *maximum dominant*.

The first aspect of the partitioning is "raining". Start at some point $x \in \mathbb{M}$ and imagine rain at x. The water will flow downwards along non-ascending paths and "wet" all the points encountered. Note that this procedure considers all points of the manifold, not just vertices.

Definition 3.5.2. Fix $x \in \mathbb{M}$. The set of points $y \in \mathbb{M}$ such that there is a non-ascending path from x to y is denoted by $\texttt{wet}(x, \mathbb{M})$ (which in turn is represented as a simplicial complex). A point z is at the *interface* of $\texttt{wet}(x, \mathbb{M})$ if every neighborhood of z has non-trivial intersection with $\texttt{wet}(x, \mathbb{M})$.

The following claim gives a description of the interface. While its meaning is quite intuitive, the proof is tedious.

Claim 3.5.3. *For any x, each connected component of the interface of $\texttt{wet}(x, \mathbb{M})$ contains a join vertex.*

Proof: If $p \in \texttt{wet}(x, \mathbb{M})$, all the points in any contour containing p are also in $\texttt{wet}(x, \mathbb{M})$. (Follow the non-ascending path from x to p and then walk along the contour.) The converse is also true, so $\texttt{wet}(x, \mathbb{M})$ contains entire contours.

Let ε, δ be sufficiently small as usual. Fix some y at the interface. Note that $y \in \texttt{wet}(x, \mathbb{M})$. (Otherwise, $B_\varepsilon(y)$ is dry.) The points in $B_\varepsilon(y)$ that lie below y have a descending path from y and hence must be wet. There must also be a dry point in $B_\varepsilon(y)$ that is above y, and hence, there exists a dry, regular $(f(y) + \delta)$-contour ϕ intersecting $B_\varepsilon(y)$.

Let Γ_y be the contour containing y. Suppose for contradiction that $\forall p \in \Gamma_y$, p has up-degree 1 (see Definition 3.2.3). Consider the non-ascending path from x to y and let z be the first point of Γ_y encountered. There exists a wet, regular $(f(y) + \delta)$-contour ψ intersecting $B_\varepsilon(z)$. Now, walk from z to y along Γ_y. If all points w in this walk have up-degree 1, then ψ is the unique $(f(y) + \delta)$-contour intersecting $B_\varepsilon(w)$. This would imply that $\phi = \psi$, contradicting the fact that ψ is wet and ϕ is dry. ■

Note that $\texttt{wet}(x, \mathbb{M})$ (and its interface) can be computed in time linear in the size of the wet simplicial complex. We perform a non-ascending search from x. Any face F of $\mathbb{M}$ encountered is partially (if not entirely) in $\texttt{wet}(x, \mathbb{M})$. The wet portion is determined by cutting F along the interface. Since each component of the interface connected subset of a contour, this is equivalent to locally cutting F by a hyperplane. All these operations can be performed to output $\texttt{wet}(x, \mathbb{M})$ in time linear in $|\texttt{wet}(x, \mathbb{M})|$.

We define a simple `lift` operation on the interface components. Consider such a component ϕ containing a join vertex y. Take any dry increasing edge incident to y, and pick the point z on this edge at height $f(y) + \delta$ (where δ is an infinitesimal, but larger than the value ε used in the definition of `cut`). Let $\texttt{lift}(\phi)$ be the unique contour through the regular point z. Note that $\texttt{lift}(\phi)$ is dry.

Claim 3.5.4. *Let ϕ be a connected component of the interface. Then $\texttt{cut}(\mathbb{M}, \texttt{lift}(\phi))$ results in two disjoint simplicial complexes, one consisting entirely of dry points.*

Proof: By Theorem 3.4.5, $\texttt{cut}(\mathbb{M}, \texttt{lift}(\phi))$ results in two disjoint simplicial complexes. Let $\mathbb{N}$ be the complex containing the point x (the argument in $\texttt{wet}(x, \mathbb{M})$), and let $\mathbb{N}'$ be the other complex. Any path from x to $\mathbb{N}'$ must intersect $\texttt{lift}(\phi)$, which is dry. Hence $\mathbb{N}'$ is dry. ■

We describe the main partitioning procedure that cuts a simplicial complex $\mathbb{N}$ into extremum dominant complexes. It takes an additional input of a maximum x. To initialize, we begin with $\mathbb{N}$ set to $\mathbb{M}$ and x as an arbitrary maximum. One of the critical aspects of the procedure is that the rain alternately flows downwards and upwards, since otherwise faces may be cut a super constant number of times (see Figure 3.3). In other words, when we start, rain flows downwards. In each recursive call, the direction of rain is switched to the opposite direction. While one can implement `rain` this way, it is conceptually easier to think of *inverting* a complex $\mathbb{N}'$ when a recursive call is made. Inversion is easily achieved by just negating the height values. (From a running time standpoint, it suffices to maintain a single bit associated with $\mathbb{N}'$ that determines whether heights are inverted or not.) We can now let rain flow downwards, as it usually does in our world.

$\texttt{rain}(x, \mathbb{N})$

1. Determine interface of $\texttt{wet}(x, \mathbb{N})$.
2. If the interface is empty, simply output $\mathbb{N}$. Otherwise, denote the connected components by $\phi_1, \phi_2, \ldots, \phi_k$ and set $\phi'_i = \texttt{lift}(\phi_i)$.
3. Initialize $\mathbb{N}_1 = \mathbb{N}$.
4. For i from 1 to k:
   (a) Construct $\texttt{cut}(\mathbb{N}_i, \phi'_i)$, consisting of dry complex $\mathbb{L}_i$ and remainder $\mathbb{N}_{i+1}$.
   (b) Let the newly created boundary of $\mathbb{L}_i$ be B_i. Invert $\mathbb{L}_i$ so that B_i is a maximum. Recursively call $\texttt{rain}(B_i, \mathbb{L}_i)$.
5. Output $\mathbb{N}_{k+1}$ together with any complexes output by recursive calls.

For convenience, denote the total output of $\texttt{rain}(x, \mathbb{M})$ by $\mathbb{M}_1, \mathbb{M}_2, \ldots, \mathbb{M}_r$. We first argue the correctness of `rain`.

Lemma 3.5.5. *Each output $\mathbb{M}_i$ is extremum dominant.*

Proof: Consider a call to $\texttt{rain}(x, \mathbb{N})$. If the interface is empty, then all of $\mathbb{N}$ is in $\texttt{wet}(x, \mathbb{N})$, so $\mathbb{N}$ is trivially extremum dominant. So suppose the interface is non-empty and consists of $\phi_1, \phi_2, \ldots, \phi_k$ (as denoted in the procedure). By repeated applications of Claim 3.5.4, $\mathbb{N}_{k+1}$ contains $\texttt{wet}(x, \mathbb{M})$. Consider $\texttt{wet}(x, \mathbb{N}_{k+1})$. The interface must exactly be $\phi_1, \phi_2, \ldots, \phi_k$. So the only dry vertices are those in the boundaries $B_1, B_2, \ldots, B_k$. But these boundaries are maxima. ■

As $\texttt{rain}(x, \mathbb{M})$ proceeds, new faces/simplices are created because of repeated cutting. The key to the running time of $\texttt{rain}(x, \mathbb{M})$ is bounding the number of newly created faces, for which we have the following lemma.

Lemma 3.5.6. *A face $F \in \mathbb{M}$ is cut*[§] *at most once during* $\texttt{rain}(x, \mathbb{M})$.

Proof: All notation here follows that in the pseudocode of `rain`. First, by Theorem 3.4.5, all the pieces on which `rain` is invoked are disjoint. Second, all recursive calls are made on dry complexes.

Consider the first time that F is cut, say, during the call to $\texttt{rain}(x, \mathbb{N})$. Specifically, say this happens when $\texttt{cut}(\mathbb{N}_i, \phi'_i)$ is constructed. $\texttt{cut}(\mathbb{N}_i, \phi'_i)$ will cut F with two horizontal cutting planes, one ε above ϕ'_i

[§]Technically what we are calling a single cut is done with two hyperplanes.

and one ε below ϕ_i'. This breaks F into lower and upper portions which are then triangulated (there is also a discarded middle portion). The lower portion, which is adjacent ϕ_i, gets included in $\mathbb{N}_{k+1}$, the complex containing the wet points, and hence does not participate in any later recursive calls. The upper portion (call it U) is in $\mathbb{L}_i$. Note that the lower boundary of U is in the boundary B_i. Since a recursive call is made to $\texttt{rain}(B_i, \mathbb{L}_i)$ (and $\mathbb{L}_i$ is inverted), U becomes wet. Hence U, and correspondingly F, will not be subsequently cut. ■

Theorem 3.5.7. *The total running time of* $\texttt{rain}(x, \mathbb{M})$ *is* $O(|\mathbb{M}|)$.

Proof: The only non-trivial operations performed are `wet` and `cut`. Since `cut` is a linear time procedure, Lemma 3.5.6 implies the total time for all calls to `cut` is $O(|\mathbb{M}|)$. As for the `wet` procedure, observe that Lemma 3.5.6 additionally implies there are only $O(|\mathbb{M}|)$ new faces created by `rain`. Therefore, since `wet` is also a linear time procedure, and no face is ever wet twice, the total time for all calls to `wet` is $O(|\mathbb{M}|)$. ■

Claim 3.5.8. *Given* $\mathcal{C}(\mathbb{M}_1), \mathcal{C}(\mathbb{M}_2), \ldots, \mathcal{C}(\mathbb{M}_r)$, $\mathcal{C}(\mathbb{M})$ *can be constructed in* $O(|\mathbb{M}|)$ *time.*

Proof: Consider the tree of recursive calls in $\texttt{rain}(x, \mathbb{M})$, with each node labeled with some $\mathbb{M}_i$. Walk through this tree in a leaf first ordering. Each time we visit a node we connect its contour tree to the contour tree of its children in the tree using the `surgery` procedure. Each `surgery` call takes constant time, and the total time is the size of the recursion tree. ■

3.6 Contour trees of extremum dominant manifolds

The previous section allows us to restrict attention to extremum dominant manifolds. We will orient so that the extremum in question is always a *minimum*. We will fix such a simplicial complex $\mathbb{M}$, with the dominant minimum m^*. For vertex v, we use $\mathbb{M}_v^+$ to denote the simplicial complex obtained by only keeping vertices u such that $f(u) > f(v)$. Analogously, define $\mathbb{M}_v^-$. Note that $\mathbb{M}_v^+$ may contain numerous connected components.

The main theorem of this section asserts that the contours trees of minimum dominant manifolds have a simple description. The exact statement will require some definitions and notation. We require the notions of *join* and *split* trees, as given by [CSA00]. Conventionally, all edges are directed from higher to lower function value.

Definition 3.6.1. The *join tree* $\mathcal{J}(\mathbb{M})$ of $\mathbb{M}$ is built on vertex set $V(\mathbb{M})$. The directed edge (u, v) is present when u is the smallest valued vertex in a connected component of $\mathbb{M}_v^+$ *and* v is connected to this component (in $\mathbb{M}$). The *split tree* $\mathcal{S}(\mathbb{M})$ is obtained by looking at $\mathbb{M}_v^-$ (or alternately, by taking the join tree of the inversion of $\mathbb{M}$).

Some basic facts about these trees. All outdegrees in $\mathcal{J}(\mathbb{M})$ are at most 1, all indegree 2 vertices are joins, all leaves are maxima, and the global minimum is the root. All indegrees in $\mathcal{S}(\mathbb{M})$ are at most 1, all outdegree 2 vertices are splits, all leaves are minima, and the global maximum is the root. As these trees are rooted, we can use ancestor-descendant terminology. Specifically, for two adjacent vertices u and v, u is the parent of v if u is closer to the root (i.e. each node can have at most one parent, but can have two children).

The key observation is that $\mathcal{S}(\mathbb{M})$ is trivial for a minimum dominant $\mathbb{M}$.

Lemma 3.6.2. $\mathcal{S}(\mathbb{M})$ *consists of:*

- *A single path (in sorted order) with all vertices except non-dominant minima.*
- *Each non-dominant minimum is attached to a unique split (which is adjacent to it).*

Proof: It suffices to prove that each split v has one child that is just a leaf, which is a non-dominant minimum. Specifically, any minimum is a leaf in $\mathcal{S}(\mathbb{M})$ and thereby attached to a split, which implies that if we removed all non-dominant minima, we must end up with a path, as asserted above.

Consider a split v. For sufficiently small ε, δ, there are exactly two $(f(v)-\delta)$-contours ϕ and ψ intersecting $B_\varepsilon(v)$. Both of these are regular contours. There must be a non-ascending path from v to the dominant minimum m^*. Consider the first edge (necessarily decreasing from v) on this path. It must intersect one of the $(f(v)-\delta)$-contours, say ϕ. By Theorem 3.4.5, $\texttt{cut}(\mathbb{M}, \phi)$ has two connected components, with one (call it $\mathbb{L}$) having ϕ^- as a boundary maximum. This complex contains m^* as the non-ascending path intersects ϕ only once. Let the other component be called $\mathbb{M}'$.

Consider $\texttt{cut}(\mathbb{M}', \psi)$ with connected component $\mathbb{N}$ having ψ^- as a boundary. $\mathbb{N}$ does not contain m^*, so any path from the interior of $\mathbb{N}$ to m^* must intersect the boundary ψ^-. But the latter is a maximum in $\mathbb{N}$, so there can be no non-ascending path from the interior to m^*. Since $\mathbb{M}$ is overall minimum dominant, the interior of $\mathbb{N}$ can only contain a single vertex w, a non-dominant minimum.

The split v has two children in $\mathcal{S}(\mathbb{M})$, one in $\mathbb{N}$ and one in $\mathbb{L}$. The child in $\mathbb{N}$ can only be the non-dominant minimum w, which is a leaf. ■

It is convenient to denote the non-dominant minima as $m_1, m_2, \ldots, m_k$ and the corresponding splits (as given by the lemma above) as $s_1, s_2, \ldots, s_k$.

Using the above lemma we can now prove that computing the contour tree for a minimum dominant manifold amounts to computing its join tree. Specifically, to prove our main theorem, we rely on the correctness of the merging procedure from [CSA00] that constructs the contour tree from the join and split trees. It actually constructs the *augmented contour tree* $\mathcal{A}(\mathbb{M})$, which obtained by replacing each edge in the contour tree with a path of all regular vertices (sorted by height) whose corresponding contour belongs to the equivalence class of that edge.

Consider a tree T with a vertex v of in and out degree at most 1. *Erasing* v from T is the following operation: if v is a leaf, just delete v. Otherwise, delete v and connect its neighbors by an edge (i.e. smooth v out). This tree is denoted by $T \ominus v$.

$\texttt{merge}(\mathcal{J}(\mathbb{M}), \mathcal{S}(\mathbb{M}))$

1. Set $\mathcal{J} = \mathcal{J}(\mathbb{M})$ and $\mathcal{S} = \mathcal{S}(\mathbb{M})$.
2. Denote v as a *candidate* if the sum of its indegree in $\mathcal{J}$ and outdegree in $\mathcal{S}$ is 1.
3. Add all candidates to queue.
4. While candidate queue is non-empty:
   (a) Let v be head of queue. If v is leaf in $\mathcal{J}$, consider its edge in $\mathcal{J}$. Otherwise consider its edge in $\mathcal{S}$. In either case, denote the edge by (v, w).
   (b) Insert (v, w) in $\mathcal{A}(\mathbb{M})$.
   (c) Set $\mathcal{J} = \mathcal{J} \ominus v$ and $\mathcal{S} = \mathcal{S} \ominus v$. Enqueue any new candidates.
5. Smooth out all regular vertices in $\mathcal{A}(\mathbb{M})$ to get $\mathcal{C}(\mathbb{M})$.

Definition 3.6.3. The *critical join tree* $\mathcal{J}_C(\mathbb{M})$ is built on the set V' of all critical points other than the non-dominant minima. The directed edge (u, v) is present when u is the smallest valued vertex in V' in a connected component of $\mathbb{M}_v^+$ and v is connected to this component (in $\mathbb{M}$).

Theorem 3.6.4. *Let $\mathbb{M}$ have a dominant minimum. The contour tree $\mathcal{C}(\mathbb{M})$ consists of all edges $\{(s_i, m_i)\}$ and $\mathcal{J}_C(\mathbb{M})$.*

Proof: We first show that $\mathcal{A}(\mathbb{M})$ is $\mathcal{J}(\mathbb{M}) \ominus \{m_i\}$ with edges $\{(s_i, m_i)\}$. We have flexibility in choosing the order of processing in `merge`. We first put the non-dominant maxima $m_1, \ldots, m_k$ into the queue. As these are processed, the edges $\{(s_i, m_i)\}$ are inserted into $\mathcal{A}(\mathbb{M})$. Once all the m_i's are erased, $\mathcal{S}$ becomes a path, so all outdegrees are at most 1. The join tree is now $\mathcal{J}(\mathbb{M}) \ominus \{m_i\}$. We can now process $\mathcal{J}$ leaf by leaf, and all edges of $\mathcal{J}$ are inserted into $\mathcal{A}(\mathbb{M})$.

Note that $\mathcal{C}(\mathbb{M})$ is obtained by smoothing out all regular points from $\mathcal{A}(\mathbb{M})$. Similarly, smoothing out regular points from $\mathcal{J}(\mathbb{M}) \ominus \{m_i\}$ yields the edges of $\mathcal{J}_C(\mathbb{M})$. ■

3.7 Painting to compute contour trees

The main algorithmic contribution is a new algorithm for computing join trees of any triangulated simplicial complex $\mathbb{M}$.

Painting: The central tool is a notion of *painting* $\mathbb{M}$. Initially associate a color with each maximum. Imagine there being a large can of paint of a distinct color at each maximum x. We will spill different paint from each maximum and watch it flow down. This is analogous to the raining in the previous section, but paint is a much more viscous liquid. *So paint only flows down edges, and it does not color the interior of higher dimensional faces.* Furthermore, paints do not mix, so every edge of $\mathbb{M}$ gets a unique color. This process (and indeed the entire algorithm) works purely on the 1-skeleton of $\mathbb{M}$, which is just a graph.

We now restate Definition 3.3.3.

Definition 3.7.1. Let the 1-skeleton of $\mathbb{M}$ have edge set E and maxima X. A *painting* of $\mathbb{M}$ is a map $\chi : X \cup E \to [|X|]$ with the following property. Consider an edge e. There exists a descending path from some maximum x to e consisting of edges in E, such that all edges along this path have the same color as x.

An *initial* painting has the additional property that the restriction $\chi : X \to [|X|]$ is a bijection.

A painting only colors edges, not vertices. We associate certain sets of colors with vertices.

Definition 3.7.2. Fix a painting χ and vertex v.

- An *up-star* of v is the set of edges that all connected to a fixed component of $\mathbb{M}_v^+$.
- A vertex v is *touched by color* c if x is incident to a c-colored edge with v at the lower endpoint. For v, $col(v)$ is the set of colors that touch v.
- A color $c \in col(v)$ *fully touches* v if all edges in an up-star are colored v.
- For any maximum $x \in X$, we say that x is both touched and fully touched by $\chi(x)$.

Each non-maximum v participates in at least 1 and at most 2 up-stars (the latter iff v is a join).

3.7.1 The data structures

We discuss all the initializations in the next section.

The binomial heaps $T(c)$: For each color c, $T(c)$ is a subset of vertices touched by c, This is stored as a *binomial max-heap* keyed by vertex heights. Abusing notation, $T(c)$ refers both to the set and the data structure used to store it.

The union-find data structure on colors: We will repeatedly perform unions of classes of colors, and this will be maintained as a standard union-find data structure. For any color c, $rep(c)$ denotes the representative of its class.

The stack K: This consists of non-extremal critical points, with monotonically increasing heights as we go from the base to the head.

Attachment vertex $att(c)$: For each color c, we maintain a critical point $att(c)$ of this color. We will maintain the guarantee that the portion of the contour tree above (and including) $att(c)$ has already been constructed.

3.7.2 The algorithm

We formally describe the algorithm below. We require a technical definition of *ripe* vertices.

Definition 3.7.3. A vertex v is *ripe* if: for all $c \in col(v)$, v is present in $T(rep(c))$ and is also the highest vertex in this heap.

`init`($\mathbb{M}$)

1. Construct an initial painting of $\mathbb{M}$ using a descending BFS from maxima that does not explore previously colored edges.
2. Determine all critical points in $\mathbb{M}$. For each v, look at $(f(v) \pm \delta)$-contours in $f|_{B_\varepsilon(v)}$ to determine the up and down degrees.
3. Mark each critical v as unprocessed.
4. For each critical v and each up-star, pick arbitrary color c touching v. Insert v into $T(c)$.
5. Initialize $rep(c) = c$ and set $att(c)$ to be the unique maximum colored c.
6. Initialize K to be an empty stack.

`build`($\mathbb{M}$)

1. Run `init`($\mathbb{M}$).
2. While there are unprocessed critical points:
   (a) Run `update`(K). Pop K to get h.
   (b) Let $cur(h) = \{rep(c) | c \in col(h)\}$.
   (c) For all $c' \in cur(h)$:
      i. Add edge $(att(c'), h)$ to $\mathcal{J}_C(\mathbb{M})$.
      ii. Delete h from $T(c')$.
   (d) Merge heaps $\{T(c') | c' \in cur(h)\}$.
   (e) Take union of $cur(h)$ and denote resulting color by $\widehat{c}$.
   (f) Set $att(\widehat{c}) = h$ and mark h as processed.

`update`(K)

1. If K is empty, push arbitrary unprocessed critical point v.
2. Let h be head of K.
3. While h is not ripe:
   (a) Find $c \in col(h)$ such that h is not the highest in $T(rep(c))$.
   (b) Push the highest of $T(rep(c))$ onto K, and update head h.

A few simple facts:

- At all times, the colors form a valid painting.
- Each vertex is present in at most 2 heaps. After processing, it is removed from all heaps.
- After v is processed, all edges incident to v have the same color (more technically, the same representative).
- Vertices on the stack are in increasing height order.

Observation 3.7.4. Each unprocessed vertex is always in exactly one queue of the colors in each of its up-stars. Specifically, for a given up-star of a vertex v, `init`($\mathbb{M}$) puts v into the queue of exactly one of the colors of the up-star, say c. As time goes on this queue may merge with other queues, but while v remains unprocessed, it is only ever (and always) in the queue of $rep(c)$, since v is never added to a new queue and is not removed until it is processed. In particular, finding the queues of a vertex in `update`(K) requires at most two union find operations (assuming each vertex records its two colors from `init`($\mathbb{M}$)).

3.7.3 Proving correctness

Our main workhorse is the following technical lemma. In the following, the current color of an edge, e, is the value of $rep(\chi(e))$, where $\chi(e)$ is the color of e from the initial painting.

Lemma 3.7.5. *Suppose vertex v is connected to a component $\mathbb{P}$ of $\mathbb{M}_v^+$ by an edge e which is currently colored c. Either all edges in $\mathbb{P}$ are currently colored c, or there exists a critical vertex $w \in \mathbb{P}$ fully touched by c and touched by another color.*

Proof: Since e has color c, there must exist vertices in $\mathbb{P}$ touched by c. Consider the highest vertex w in $\mathbb{P}$ that is touched by c and some other color. If no such vertex exists, this means all edges incident to a vertex touched by c are colored c. By walking through $\mathbb{P}$, we deduce that all edges are colored c.

So assume w exists. Take the $(f(w) + \delta)$-contour ϕ that intersects $B_\varepsilon(w)$ and intersects some c-colored edge incident to w. Note that all edges intersecting ϕ are also colored c, since w is the highest vertex to be touched by c and some other color. (Take the path of c-colored edges from the maximum to w. For any point on this path, the contour passing through this point must be colored c.) Hence, c fully touches w. But w is touched by another color, and the corresponding edge cannot intersect ϕ. So w must have up-degree 2 and is critical. ■

Corollary 3.7.6. *Each time* `update`(K) *is called, it terminates with a ripe vertex on top of the stack.*

Proof: `update`(K) is only called if there are unprocessed vertices remaining, and so by the time we reach step 3 in `update`(K), the stack has some unprocessed vertex h on it. If h is ripe, then we are done, so suppose otherwise.

Let $\mathbb{P}$ be one of the components of $\mathbb{M}_h^+$. By construction, h was put in the heap of some initial adjacent color c. Therefore, h must be in the current heap of $rep(c)$ (see Observation 3.7.4). Now by Lemma 3.7.5, either all edges in $\mathbb{P}$ are colored $rep(c)$ or there is some vertex w fully touched by $rep(c)$ and some other color. The former case implies that if there are any unprocessed vertices in $\mathbb{P}$ then they are all in $T(rep(c))$, implying that h is not the highest vertex and a new higher up unprocessed vertex will be put on the stack for the next iteration of the while loop. Otherwise, all the vertices in $\mathbb{P}$ have been processed. However, it

cannot be the case that all vertices in all components of $\mathbb{M}_h^+$ have already been processed, since this would imply that h was ripe, and so one can apply the same argument to the other non-fully processed component.

Now consider the latter case, where we have a non-monochromatic vertex w. In this case w cannot have been processed (since after being processed it is touched only by one color), and so it must be in $T(rep(c))$ since it must be in some heap of a color in each up-star (and one up-star is entirely colored $rep(c)$). As w lies above h in $\mathbb{M}$, this implies h is not on the top of this heap. ■

We prove a series of claims that will lead to the correctness proof.

Claim 3.7.7. *Consider a ripe vertex v and take the up-star connecting to some component of $\mathbb{M}_v^+$. All edges in this component and the up-star have the same color.*

Proof: Let c be the color of some edge in this up-star. By ripeness, v is the highest in $T(rep(c))$. Denote the component of $\mathbb{M}_v^+$ by $\mathbb{P}$. By Lemma 3.7.5, either all edges in $\mathbb{P}$ are colored $rep(c)$ or there exists critical vertex $w \in \mathbb{P}$ fully touched by $rep(c)$ and another color. In the latter case, w has not been processed, so $w \in T(rep(c))$ (contradiction to ripeness). Therefore, all edges in $\mathbb{P}$ are colored $rep(c)$. ■

Claim 3.7.8. *The partial output on the processed vertices is exactly the restriction of $\mathcal{J}_C(\mathbb{M})$ to these vertices.*

Proof: More generally, we prove the following: all outputs on processed vertices are edges of $\mathcal{J}_C(\mathbb{M})$ and for any current color c, $att(c)$ is the lowest processed vertex of that color. We prove this by induction on the processing order. The base case is trivially true, as initially the processed vertices and attachments of the color classes are the set of maxima. For the induction step, consider the situation when v is being processed.

Since v is being processed, we know by Corollary 3.7.6 that it is ripe. Take any up-star of v, and the corresponding component $\mathbb{P}$ of $\mathbb{M}_v^+$ that it connects to. By Claim 3.7.7, all edges in $\mathbb{P}$ and the up-star have the same color (say c). If some critical vertex in $\mathbb{P}$ is not processed, it must be in $T(c)$, which violates the ripeness of v. Thus, all critical vertices in $\mathbb{P}$ have been processed, and so by the induction hypothesis, the restriction of $\mathcal{J}_C(\mathbb{M})$ to $\mathbb{P}$ has been correctly computed. Additionally, since all critical vertices in $\mathbb{P}$ have processed, they all have the same color c of the lowest critical vertex in $\mathbb{P}$. Thus by the strengthened induction hypothesis, this lowest critical vertex is $att(c)$.

If there is another component of $\mathbb{M}_v^+$, the same argument implies the lowest critical vertex in this component is $att(c')$ (where c' is the color of edges in the respective component). Now by the definition of $\mathcal{J}_C(\mathbb{M})$, the critical vertex v connects to the lowest critical vertex in each component of $\mathbb{M}_v^+$, and so by the above v should connect to $att(c)$ and $att(c')$, which is precisely what v is connected to by `build`$(\mathbb{M})$. Moreover, `build` merges the colors c and c' and correctly sets v to be the attachment, as v is the lowest processed vertex of this merged color (as by induction $att(c)$ and $att(c')$ were the lowest vertices before merging colors). ■

Theorem 3.7.9. *Given an input complex $\mathbb{M}$,* `build`$(\mathbb{M})$ *terminates and outputs $\mathcal{J}_C(\mathbb{M})$.*

Proof: First observe that each vertex can be processed at most once by `build`$(\mathbb{M})$. By Corollary 3.7.6, we know that as long as there is an unprocessed vertex, `update`(K) will be called and will terminate with a ripe vertex which is ready to be processed. Therefore, eventually all vertices will be processed, and so by Claim 3.7.8 the algorithm will terminate having computed $\mathcal{J}_C(\mathbb{M})$. ■

3.7.4 Running time

We now bound the running time of the above algorithm. In the subsequent sections we will then provide a matching lower bound for the running time. Therefore, it will be useful to set up some terminology that can be used consistently in both places. Specifically, the lower bound proof will be a purely combinatorial statement on colored rooted trees, and so the terminology is of this form.

Any tree T considered in following will be a rooted binary tree[¶] where the height of a vertex is its distance from the root r (i.e. conceptually T will be a join tree with r at the bottom). As such, the children of a vertex $v \in T$ are the adjacent vertices of larger height (and v is the parent of such vertices). Then the subtree rooted at v, denoted T_v consists of the graph induced on all vertices which are descendants of v (including v itself). For two vertices v and w in T let $d(v, w)$ denote the length of the path between v and w. We use $A(v)$ to denote the set of ancestors of v. For a set of nodes U, $A(U) = \bigcup_{u \in U} A(u)$.

Definition 3.7.10. A *leaf assignment* χ of a tree T assigns *two* distinct leaves to each internal vertex v, one from the left child and one from the right child subtree of v (naturally if v has only one child it is assigned only one color).

For a vertex $v \in T$, we use H_v to denote the *heap* at v. Formally, $H_v = \{u | u \in A(v), \chi(u) \cap L(T_v) \neq \emptyset\}$. In words, H_v is the set of ancestors of v which are colored by some leaf in T_v.

Definition 3.7.11. Note that the subroutine `init`($\mathbb{M}$) from §3.7.2 naturally defines a leaf assignment to $\mathcal{J}_C(\mathbb{M})$ according to the priority queue for each up-star we put a given vertex in. Call this the *initial coloring* of the vertices in $\mathcal{J}_C(\mathbb{M})$. Note also that this initial coloring defines the H_v values for all $v \in \mathcal{J}_C(\mathbb{M})$.

The following lemma should justify these technical definitions.

Lemma 3.7.12. *Let $\mathbb{M}$ be a simplicial complex with t critical points. For every vertex in $\mathcal{J}_C(\mathbb{M})$, let H_v be defined by the initial coloring of $\mathbb{M}$. The running time of* `build`($\mathbb{M}$) *is $O(N + t\alpha(t) + \sum_{v \in \mathcal{J}_C(\mathbb{M})} \log |H_v|)$.*

Proof: First we look at the initialization procedure `init`($\mathbb{M}$). This procedure runs in $O(N)$ time. Indeed, the painting procedure consists of several BFS's but as each vertex is only explored by one of the BFS's, it is linear time overall. Determining the critical points is a local computation on the neighborhood of each vertex as so is linear (i.e. each edge is viewed at most twice). Finally, each vertex is inserted into at most two heaps and so initializing the heaps takes linear time in the number of vertices.

Now consider the union-find operations performed by `build` and `update`. Initially the union find data structure has a singleton component for each leaf (and no new components are ever created), and so each union-find operation takes $O(\alpha(t))$ time. For `update`, by Observation 3.7.4, each iteration of the while loop requires a constant number of finds (and no unions). Specifically, if a vertex is found to be ripe (and hence processed next) then these can be charged to that vertex. If a vertex is not ripe, then these can be charged to the vertex put on the stack. As each vertex is put on the stack or processed at most once, `update` performs $O(t)$ finds overall. Finally, `build`($\mathbb{M}$) performs one union and at most two finds for each vertex. Therefore the total number of union find operations is $O(t)$.

For the remaining operations, observe that for every iteration of the loop in `update`, a vertex is pushed onto the stack and each vertex can only be pushed onto the stack once (since the only way it leaves the

[¶]Note that technically the trees considered should have a leaf vertex hanging below the root of this in order to represent the global minimum of the complex. This vertex is (safely) ignored to simplify the presentation.

stack is by being processed). Therefore the total running time due to `update` is linear (ignoring the find operations).

What remains is the time it takes to process a vertex v in `build`$(\mathbb{M})$. In order to process a vertex there are a few constant time operations, union-find operations, and queue operations. Therefore the only thing left to bound are the queue operations. Let v be a vertex in $\mathcal{J}_C(\mathbb{M})$, and let c_1 and c_2 be its children (the same argument holds if v has only one child). At the time v is processed, the colors and queues of all vertices in a given component of $\mathbb{M}_v^+$ have merged together. In particular, when v is processed we know it is ripe and so all vertices above v in each component of $\mathbb{M}_v^+$ have been processed, implying these merged queues are the queues of the current colors of c_1 and c_2. Again since v is ripe, it must be on the top of these queues and so the only vertices left in these queues are those in H_{c_1} and H_{c_2}.

Now when v is handled, three queue operations are performed. Specifically, v is removed from the queues of c_1 and c_2, and then the queues are are merged together. By the above arguments the sizes of the queues for each of these operations are H_{c_1}, H_{c_2}, and H_v, respectively. As merging and deleting takes logarithmic time in the heap size for binomial heaps, the claim now follows. ∎

3.8 Leaf assignments and path decompositions

In this section, we set up a framework to analyze the time taken to compute a critical join tree $\mathcal{J}_C(\mathbb{M})$ (see Definition 3.6.3). We adopt all notation already defined in §3.7.4. From here forward we will often assume binary trees are full binary trees (this assumption simplifies the presentation but is not necessary).

Let χ be some fixed leaf assignment to a rooted binary tree T, which in turn fixes all the heaps H_v. We choose a special path decomposition that is best defined as a subset of edges in T such that each internal vertex has degree at most 2. This naturally gives a path decomposition. For each internal vertex $v \in T$, add the edge from v to $\arg\max_{v_l,v_r}\{|H_{v_l}|, |H_{v_r}|\}$ where v_l and v_r are the children of v (if $|H_{v_l}| = |H_{v_r}|$ then pick one arbitrarily). This is called the *maximum* path decomposition, denoted by $P_{\max}(T)$.

Our main goal in this section is to prove the following theorem. We use $|p|$ to denote the number of vertices in p.

Theorem 3.8.1. $\sum_{v\in T} \log |H_v| = O(\sum_{p\in P_{\max}(T)} |p| \log |p|)$.

We conclude this section in §3.8.4 by showing that proving this theorem implies our main result Theorem 3.1.2.

3.8.1 Shrubs, tall paths, and short paths

The paths in $P(T)$ naturally define a tree[‖] of their own. Specifically, in the original tree T contract each path down to its root. Call the resulting tree the *shrub* of T corresponding to the path decomposition $P(T)$. Abusing notation, we simply use $P(T)$ to denote the shrub. As a result, we use terms like 'parent', 'child', 'sibling', etc. for paths as well. The shrub gives a handle on the heaps of a path. We use $b(p)$ to denote the *base* of the path, which is vertex in p closest to root of T. We use $\ell(p)$ to denote the leaf in p. We use H_p to denote the $H_{b(p)}$.

Lemma 3.8.2. *Let p be any path in $P(T)$ and let $\{q_1, \dots q_k\}$ be the children on p. Then $H_{\ell(p)} + \sum_{i=1}^{k} |H_{q_i}| \leq |H_p| + 2|p|$.*

[‖] Please excuse the overloading of the term 'tree', it is the most natural term to use here.

Proof: For convenience, denote $H_i = H_{q_i}$ and $H_0 = H_{\ell(p)}$. Consider $v \in \bigcup_i H_i$ that lies below $b(p)$ in T. Note that such a vertex has only one of its two colors in $L(b(p))$. Since the colors tracked by H_i and H_j for $i \neq j$ are disjoint, such a vertex can appear in only one of the H_i's. On the other hand, a vertex $u \in p$ can appear in more than one H_i, but since any vertex has exactly two colors it can appear in at most two such heaps. Hence, $\sum_i |H_i| \leq |H_p| + 2|p|$. ∎

We wish to prove $\sum_{v \in T} \log |H_v| = O(\sum_{p \in P} |p| \log |p|)$. The simplest approach is to prove $\forall p \in P$, $\sum_{v \in p} \log |H_v| = O(|p| \log |p|)$. This is unfortunately not true, which is why we divide paths into two categories.

Definition 3.8.3. For $p \in P(T)$, p is *short* if $|p| < \sqrt{|H_p|}/100$, and *tall* otherwise.

The following lemma demonstrates that tall paths can "pay" for themselves.

Lemma 3.8.4. *If p is tall, $\sum_{v \in p} \log |H_v| = O(|p| \log |p|)$. If p is short, $\sum_{v \in p} \log |H_v| = O(|H_p| \log |H_p|)$.*

Proof: For $v \in p$, $|H_v| \leq |H_p| + |p|$ (as v is a descendant of $b(p)$ along p). Hence, $\sum_{v \in p} \log |H_v| \leq \sum_{v \in p} \log(|H_p| + |p|) = |p| \log(|H_p| + |p|)$. If p is a tall path, then $|p| \log(|H_p| + |p|) = O(|p| \log |p|)$. If p is short, then $|p| \log(|H_p| + |p|) = O(|p| \log |H_p|)$. For short paths, $|p| = O(|H_p|)$. ∎

There are some short paths that we can also "pay" for. Consider any short path p in the shrub. We will refer to the *tall support chain* of p as the tall ancestors of p in the shrub which have a path to p which does not use any short path (i.e. it is a chain of paths adjacent to p).

Definition 3.8.5. A short path p is *supported* if at least $|H_p|/100$ vertices v in H_p lie in paths in the tall support chain of p.

Let $\mathcal{L}$ be the set of short paths, $\mathcal{L}'$ be the set of supported short paths, and $\mathcal{H}$ be the set of tall paths given by $P_{\max}(T)$. We now construct the shrub of unsupported short paths. Consider $p \in \mathcal{L} \setminus \mathcal{L}'$, and traverse the chain of ancestors from p. Eventually, we must reach another short path q. (If not, we have reached the root r of $P_{\max}(T)$. Hence, p is supported.) Insert edge from p to q, so q is the parent of p in $\mathcal{U}$. This construction leads to the shrub forest of $\mathcal{L} \setminus \mathcal{L}'$, where all the roots are supported short paths, and the remaining nodes are the unsupported short paths.

Most of the work goes into proving the following technical lemma.

Lemma 3.8.6. *Let $\mathcal{U}$ denote a connected component (shrub) in the shrub forest of $\mathcal{L} \setminus \mathcal{L}'$ and let r be the root of $\mathcal{U}$. (i) For any $v \in p$ such that $p \in \mathcal{U}$, $|H_v| = O(|H_r|)$. (ii) $\sum_{p \in \mathcal{U}} |p| = O(|H_r|)$.*

We split the remaining argument into two subsections. We first prove Theorem 3.8.1 from Lemma 3.8.6, which involves routine calculations. Then we prove Lemma 3.8.6, where the interesting work happens.

3.8.2 Proving Theorem 3.8.1

We split the summation into tall, short, and unsupported short paths.

$$\sum_{p \in \mathcal{L}} \sum_{v \in p} \log |H_v| = \sum_{p \in \mathcal{L} \setminus \mathcal{L}'} \sum_{v \in p} \log |H_v| + \sum_{p \in \mathcal{L}'} \sum_{v \in p} \log |H_v| + \sum_{p \in \mathcal{H}} \sum_{v \in p} \log |H_v|$$

The last term can be bounded by $O(\sum_{p \in P_{\max}(T)} |p| \log |p|)$, by Lemma 3.8.4. The second term can be bounded by $O(\sum_{p \in \mathcal{L}'} |H_p| \log |H_p|)$, by Lemma 3.8.4 again. The following claim shows that this in turn is at most the last term.

Claim 3.8.7. $\sum_{p \in \mathcal{L}'} |H_p| \log |H_p| = O(\sum_{q \in \mathcal{H}} \sum_{v \in q} \log |H_v|)$.

Proof: Pick $p \in \mathcal{L}'$. As we traverse the tall support chain of p, there are at least $|H_p|/100$ vertices of H_p that lie in these paths. These are encountered in a fixed order. Let H'_p be the first $|H_p|/200$ of these vertices. When $v \in H'_p$ is encountered, there are $|H_p|/200$ vertices of H_p not yet encountered. Hence, $|H_v| \geq |H_p|/200$. Hence, $|H_p| \log |H_p| = O(\sum_{v \in H'_p} \log |H_v|)$. Since all the vertices lie in tall paths, we can write this as $O(\sum_{q \in \mathcal{H}} \sum_{v \in H'_p \cap q} \log |H_v|)$. Summing over all p, the expression is $\sum_{q \in \mathcal{H}} \sum_{p \in \mathcal{L}'} \sum_{v \in H'_p \cap q} \log |H_v|$.

Consider any $v \in H'_p$. Let S be the set of paths $\widetilde{p} \in \mathcal{L}'$ such that $v \in H'_{\widetilde{p}}$. We now show $|S| \leq 2$ (i.e. it contains at most one path other than p). First observe that any two paths in S must be unrelated (i.e. S is an anti-chain), since paths which have an ancestor-descendant relationship have disjoint tall support chains. However, any vertex v receives exactly one color from each of its two subtrees (in T), and therefore $|S| \leq 2$ since any two paths which share descendant leaves in T (i.e. their heaps are tracking the same color) must have an ancestor-descendant relationship.

In other words, any $\log |H_v|$ appears at most twice in the above triple summation. Hence, we can bound it by $O(\sum_{q \in \mathcal{H}} \sum_{v \in q} \log |H_v|)$. ■

The first term (unsupported short paths) can be charged to the second term (supported short paths). This is where the critical Lemma 3.8.6 plays a role.

Claim 3.8.8. $\sum_{p \in \mathcal{L} \setminus \mathcal{L}'} \sum_{v \in p} \log |H_v| = O(\sum_{p \in \mathcal{L}'} |H_p| \log |H_p|)$.

Proof: Let $\mathcal{U}$ denote a connected component of the shrub forest. We have $\sum_{p \in \mathcal{L} \setminus \mathcal{L}'} \sum_{v \in p} \log |H_v| \leq \sum_{\mathcal{U}} \sum_{p \in \mathcal{U}} \sum_{v \in p} \log |H_v|$. By Lemma 3.8.6, $|H_v| = O(|H_r|)$, where r is the root of $\mathcal{U}$. Furthermore, $\sum_{p \in \mathcal{U}} |p| = O(|H_r|)$. We have $\sum_{p \in \mathcal{U}} \sum_{v \in p} \log |H_v| = O((\log |H_r|) \sum_{p \in \mathcal{U}} |p|) = O(|H_r| \log |H_r|)$. We sum this over all $\mathcal{U}$ in the shrub forest, and note that roots in the shrub forest are supported short paths. ■

3.8.3 Proving Lemma 3.8.6: the root is everything in $\mathcal{U}$

Lemma 3.8.6 asserts the root r in $\mathcal{U}$ pretty much encompasses all sizes and heaps in $\mathcal{U}$. We will work with the *reduced* heap $\widetilde{H}_p$. This is the subset of vertices of H_p that do not appear on the tall support chain of p. By definition, for any unsupported short path (hence, any non-root $p \in \mathcal{U}$), $|\widetilde{H}_p| \geq 99|H_p|/100$. We begin with a key property, which is where the construction of $P_{\max}(T)$ enters the picture.

Lemma 3.8.9. *Let q be the child of some path p in $\mathcal{U}$, then $|H_p| \geq \frac{3}{2}|H_q|$. Moreover, if $p \neq r(\mathcal{U})$, then $|\widetilde{H}_p| \geq \frac{3}{2}|\widetilde{H}_q|$.*

Proof: Let $h(q)$ denote the tall path that is a child of p in $P_{\max}(T)$, and an ancestor of q. If no such tall path exists, then by construction p is the parent of q in $P_{\max}(T)$, and the following argument will go through by setting $h(q) = q$.

The chain of ancestors from q to $h(q)$ consists only of tall paths. Since q is unsupported, these paths contain at most $|H_q|/100$ vertices of H_q. Thus, $|H_{h(q)}| \geq 99|H_q|/100$.

Consider the base of $h(q)$, which is a node w in T. Let v denote the sibling of w in T. Their parent is called u. Note that both u and v are nodes in the path p. Now, the decomposition $P_{\max}(T)$ put u and v in

the same path p. This implies $|H_v| \geq |H_w|$. Since $|H_u| \geq |H_v| + |H_w| - 2$, $|H_u| \geq 2|H_w| - 2$. Let b be the base of p. We have $|H_p| = |H_b| \geq |H_u| - |p| \geq 2|H_w| - |p| - 2$. Since p is a short path, $|p| < \sqrt{|H_p|}/100$. Applying this bound, we get $|H_p| \geq (2-\delta)|H_w|$ (for a small constant $\delta > 0$). Since w is the base of $h(q)$, $H_w = H_{h(q)}$. We apply the bound $|H_{h(q)}| \geq 99|H_q|/100$ to get $|H_p| \geq 197|H_q|/100$, implying the first part of the lemma. For the second part, observe that if $p \neq r(\mathcal{U})$, then p is unsupported and so $|\widetilde{H}_p| \geq 99|H_p|/100$, and therefore the second part follows since $|H_q| \geq |\widetilde{H}_q|$. ■

This immediately proves part (i) of Lemma 3.8.6. Part (ii) requires much more work.

We define a *residue* R_p for each $p \in \mathcal{U}$. Suppose p has children $q_1, q_2, \ldots, q_k$ in $\mathcal{U}$. Then $R_p = |\widetilde{H}_p| - \sum_i |\widetilde{H}_{q_i}|$. By definition, $|\widetilde{H}_p| = \sum_{q \in \mathcal{U}_p} R_p$. Note that R_p can be negative. Now, define $R_p^+ = \max(R_p, 0)$, and set $W_p = \sum_{q \in \mathcal{U}_p} R_p^+$. Observe that $W_p \geq |\widetilde{H}_p|$. We also get an approximate converse.

Claim 3.8.10. *For any path $p \in \mathcal{U}$, $|\widetilde{H}_p| \geq W_p - 2\sum_{q \in \mathcal{U}_p} |q|$.*

Proof: We write $W_p - |\widetilde{H}_p| = \sum_{q \in \mathcal{U}_p} R_q^+ - R_q = -\sum_{q \in \mathcal{U}_p : R_q < 0} R_q$. Consider $q \in \mathcal{U}_p$ and denote the children in $\mathcal{U}_p$ by $q'_1, q'_2, \ldots$. Note that R_q is negative exactly when $|\widetilde{H}_q| < \sum_i |\widetilde{H}_{q'_i}|$. Traverse $P_{\max}(T)$ from q'_i to q. Other than q, all other nodes encountered are in the tall support chain of q'_i and hence do not affect its reduced heap. The vertices of $\widetilde{H}_{q'_i}$ that are deleted are exactly those present in the path q. Any vertex in q can be deleted from at most two of the reduced heaps (of the children of q in $\mathcal{U}_p$), since theses reduced heaps do not have an ancestor-descendant relationship. Therefore when R_q is negative, it is at most by $2|q|$. We sum over all q to complete the proof. ■

The main challenge of the entire proof is bounding the sum of path lengths, which is done next. We stress that the all the previous work is mostly the setup for this claim.

Claim 3.8.11. *Fix any path $p \in \mathcal{U} \setminus \{r(\mathcal{U})\}$. Suppose for any $q, q' \in \mathcal{U}_p$ where q is a parent of q' in $\mathcal{U}_p$, $W_q \geq (4/3)W_{q'}$. Then $\sum_{q \in \mathcal{U}_p} |q| \leq W_p/20$.*

Proof: Since q is an unsupported short path, $|q| < \sqrt{|H_q|}/100 \leq \sqrt{|\widetilde{H}_q|}/99 \leq \sqrt{|W_q|}/99$. We prove that $\sum_{q \in \mathcal{U}_p} \sqrt{|W_q|}/99 \leq W_p/20$ by a charge redistribution scheme. Assume that each $q \in \mathcal{U}_p^-$ starts with $\sqrt{W_q}/99$ units of charge. We redistribute this charge over all nodes in $\mathcal{U}_q$, and then calculate the total charge. For $q \in \mathcal{U}_p$, spread its charge to all nodes in $\mathcal{U}_q$ proportional to R^+ values. In other words, give $(\sqrt{W_q}/99) \cdot (R_{q'}^+/W_q)$ units of charge to each $q' \in \mathcal{U}_q$.

After the redistribution, let us compute the charge deposited at q. Every ancestor in $\mathcal{U}_p^-$ $q = a_0, a_1, a_2, \ldots, a_k$ contributes to the charge at q. The charge is expressed in the following equation. We use the assumption that $W_{a_i} \geq (4/3)W_{a_{i-1}}$ and hence $W_{a_i} \geq (4/3)^i W_{a_0} \geq (4/3)^i$, as a_0 is an unsupported short path and hence $W_{a_0} \geq 1$.

$$(R_q^+/99)\sum_{a_i} 1/\sqrt{W_{a_i}} \leq (R_q^+/99)\sum_{a_i}(3/4)^{i/2} \leq R_q^+/20$$

The total charge is $\sum_{q \in \mathcal{U}_p} R_p^+/20 = W_p/20$. ■

Corollary 3.8.12. *Let r be the root of $\mathcal{U}$, and suppose that for any paths $q, q' \in \mathcal{U} \setminus \{r\}$, where q is a parent of q' in $\mathcal{U}$, $W_q \geq (4/3)W_{q'}$. Then $\sum_{p \in \mathcal{U}} |p| \leq W_r/20 + |r|$.*

Proof: Let $c_1, \ldots, c_m$ be the children of r in $\mathcal{U}$. By definition, $W_r = \sum_i W_{c_i} + R_r^+ \geq \sum_i W_{c_i}$. By Claim 3.8.11, for each c_i we have $W_{c_i}/20 \geq \sum_{p \in \mathcal{U}_{c_i}} |p|$. Combining these to facts yields the claim. ■

We wrap it all up by proving part (ii) of Lemma 3.8.6.

Claim 3.8.13. $\sum_{p\in\mathcal{U}} |p| \leq |H_{r(\mathcal{U})}|/10$.

Proof: We use r for $r(\mathcal{U})$. Suppose $W_q \geq (4/3)W_{q'}$ (for any choice in $\mathcal{U}\setminus\{r\}$ of q parent of q'), then by Corollary 3.8.12, $\sum_{p\in\mathcal{U}} |p| \leq W_r/20 + |r|$. By Claim 3.8.10, $|\widetilde{H}_r| \geq W_r - 2\sum_{p\in\mathcal{U}} |p|$, and so combining these inequalities gives,

$$\sum_{p\in\mathcal{U}} |p| \leq \frac{10}{9}\Big(|\widetilde{H}_r|/20 + |r|\Big) \leq \frac{10}{9}\Big(|H_r|/20 + \sqrt{|H_r|}/100\Big) \leq |H_r|/10.$$

We now prove that for any q parent of q' (other than r), $W_q \geq (4/3)W_{q'}$. Suppose not. Let p, p' be the counterexample furthest from the root, where p is the parent of p'. Note that for q and child q' in $\mathcal{U}_{p'}$, $W_q \geq (4/3)W_{q'}$. We will apply Claim 3.8.11 for $\mathcal{U}_{p'}$ to deduce that $\sum_{q\in\mathcal{U}_{p'}} |q| \leq W_{p'}/20$. Combining this with Claim 3.8.10 gives, $|\widetilde{H}_{p'}| \geq 19W_{p'}/20$. By Lemma 3.8.9, $|\widetilde{H}_p| \geq (3/2)|\widetilde{H}_{p'}|$. Noting that $W_p \geq |\widetilde{H}_p|$, we deduce that $W_p \geq (4/3)W_{p'}$. Hence, p, p' is not a counterexample, and more generally, there is no counterexample. That completes the whole proof. ∎

3.8.4 Our main result

We now show that Theorem 3.8.1 allows us to upper bound the running time for our join tree and contour tree algorithms in terms of path decompositions.

Theorem 3.8.14. *Consider a PL-Morse $f : \mathbb{M} \to \mathbb{R}$, where the join tree $\mathcal{J}_C(\mathbb{M})$ has maximum degree 3. There is an algorithm to compute the join tree whose running time is $O(\sum_{p\in P_{\max}(\mathcal{J}_C)} |p| \log |p| + t\alpha(t) + N)$.*

Proof: By Theorem 3.7.9 we know that `build`$(\mathbb{M})$ correctly outputs $\mathcal{J}_C(\mathbb{M})$, and by Lemma 3.7.12 we know this takes $O(\sum_{v\in\mathcal{J}_C(\mathbb{M})} \log |H_v| + t\alpha(t) + N)$ time, where the H_v values are determined as in Definition 3.7.11. Therefore by Theorem 3.8.1, `build`$(\mathbb{M})$ takes $O(\sum_{p\in P_{\max}(\mathcal{J}_C)} |p| \log |p| + t\alpha(t) + N)$ time to correctly compute $\mathcal{J}_C(\mathbb{M})$. ∎

This result for join trees easily implies our main result, Theorem 3.1.2, which we now restate and prove.

Theorem 3.8.15. *Consider a PL-Morse $f : \mathbb{M} \to \mathbb{R}$, where the contour tree $\mathcal{C} = \mathcal{C}(\mathbb{M})$ has maximum degree 3. There is an algorithm to compute $\mathcal{C}$ whose running time is $O(\sum_{p\in P(\mathcal{C})} |p| \log |p| + t\alpha(t) + N)$, where $P(T)$ is a specific path decomposition (constructed implicitly by the algorithm).*

Proof: First, lets review the various pieces of our algorithm. On a given input simplicial complex, we first break it into extremum dominant pieces using `rain`$(\mathbb{M})$ (and in $O(|\mathbb{M}|)$ time by Theorem 3.5.7). Specifically, Lemma 3.5.5 proves that the output of `rain`$(\mathbb{M})$ is a set of extremum dominant pieces, $\mathbb{M}_1, \ldots, \mathbb{M}_k$, and Claim 3.5.8 shows that given the contour trees, $\mathcal{C}(\mathbb{M}_1), \ldots, \mathcal{C}(\mathbb{M}_k)$, the full contour tree, $\mathcal{C}(\mathbb{M})$, can be constructed (in $O(|\mathbb{M}|)$ time).

Now one of the key observations was that for extremum dominant manifolds, computing the contour tree is roughly the same as computing the join tree. Specifically, Theorem 3.6.4 implies that given $\mathcal{J}_C(\mathbb{M}_i)$, we can obtain $\mathcal{C}(\mathbb{M}_i)$ by simply sticking on the non-dominant minima at their respective splits (which can easily be done in linear time). However, by the above theorem we know $\mathcal{J}_C(\mathbb{M}_i)$ can be computed in $O(\sum_{p\in P_{\max}(\mathcal{J}_C(\mathbb{M}_i))} |p| \log |p| + t_i\alpha(t_i) + N_i)$ (where t_i and N_i are the number of critical points and faces when restricted to $\mathbb{M}_i$).

At this point we can now see what the path decomposition referenced in theorem statement should be. It is just the union of all the maximum path decomposition across the extremum dominant pieces, $P_{\max}(\mathcal{C}(\mathbb{M})) = \cup_{i=1}^{k} P_{\max}(\mathcal{J}_C(\mathbb{M}_i))$. Since all procedures besides computing the join trees take linear time in the size of the input complex, we can therefore compute the contour tree in time

$$O\left(N + \sum_{i=1}^{k}\left(\sum_{p\in P_{\max}(\mathcal{J}_C(\mathbb{M}_i))} |p|\log|p|\right) + t_i\alpha(t_i) + N_i\right) = O\left(\left(\sum_{p\in P_{\max}(\mathcal{C}(\mathbb{M}))} |p|\log|p|\right) + t\alpha(t) + N\right)$$

■

3.9 Lower bound by path decomposition

We first prove a lower bound for join trees, and then generalize to contour trees.

3.9.1 Join trees

We focus on terrains, so $d = 2$. Consider any path decomposition P of a valid join tree (i.e. any rooted binary tree). When we say "compute the join tree", we require the join tree to be labeled with the corresponding vertices of the terrain.

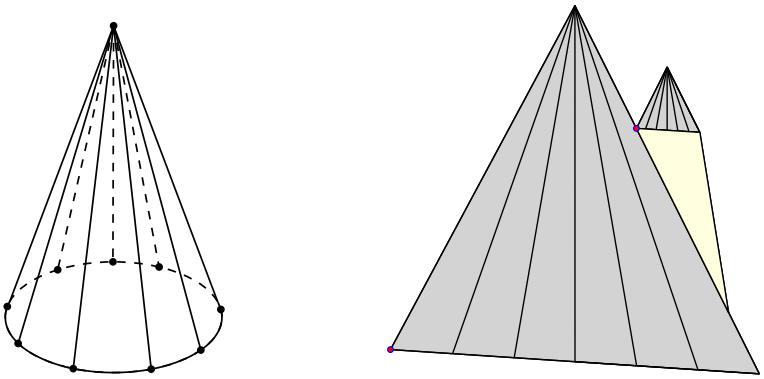


Figure 3.5: Left: angled view of a diamond / Right: a parent and child diamond put together

Lemma 3.9.1. *Fix any path decomposition P. There is a family of terrains, $\mathbf{F}_P$, all with the same triangulation, such that $|\mathbf{F}_P| = \Pi_{p_i\in P}(|p_i| - 1)!$, and no two terrains in $\mathbf{F}_P$ define the same join tree.*

Proof:

We describe the basic building block of these terrains, which corresponds to a fixed path $p \in P$. Informally, a *diamond* is an upside down cone with m triangular faces (see Figure 3.5). Construct a slightly tilted cycle of length m with the two antipodal points at heights 1 and 0. These are called the anchor and trap of the diamond, respectively. The remaining $m - 2$ vertices are evenly spread around the cycle and heights decrease monotonically when going from the anchor to the trap. Next, create an apex vertex at some appropriately large height, and add an edge to each vertex in the cycle.

Now we describe how to attach two different diamonds. In this process, we glue the base of a scaled down "child" diamond on to a triangular cone face of the larger "parent" diamond (see Figure 3.5). Specifically,

the anchor of the child diamond is attached directly to a face of the parent diamond at some height h. The remaineder of the base of the child cone is then extended down (at a slight angle) until it hits the face of the parent.

The full terrain is obtained by repeatedly gluing diamonds. For each path $p_i \in P$, we create a diamond of size $|p_i|+1$. The two faces adjacent to the anchor are always empty, and the remaining faces are for gluing on other diamonds. (Note that diamonds have size $|p_i|+1$ since $|p_i|-1$ faces represent the joins of p_i, the apex represents the leaf, and we need two empty faces next to the anchor.) Now we glue together diamonds of different paths in the same way the paths are connected in the shrub $P_{\mathcal{S}}$ (see §3.8.1). Specially, for two paths $p, q \in P$ where p is the parent of q in $P_{\mathcal{S}}$, we glue q onto a face of the diamond for p as described above. (Naturally for this construction to work, diamonds for a given path will be scaled down relative to the size of the diamond of their parent). By varying the heights of the gluing, we get the family of terrains.

Observe now that the only saddle points in this construction are the anchor points. Moreover, the only maxima are the apexes of the diamonds. We create a global boundary minimum by setting the vertices at the base of the diamond representing the root of $P_{\mathcal{S}}$ all to the same height (and there are no other minima). Therefore, the saddles on a given diamond will appear contiguously on a root to leaf path in the join tree of the terrain, where the leaf corresponds to the maximum of the diamond (since all these saddles have a direct line of sight to this apex). In particular, this implies that, regardless of the heights assigned to the anchors, the join tree has a path decomposition whose corresponding shrub is equivalent to $P_{\mathcal{S}}$.

There is a valid instance of this described construction for any relative ordering of the heights of the saddles on a given diamond. In particular, there are $(|p_i|-1)!$ possible orderings of the heights of the saddles on the diamond for p_i, and hence $\Pi_{p_i \in P}(|p_i|-1)!$ possible terrains we can build. Each one of these functions will result in a different (labeled) join tree. All saddles on a given diamond will appear in sorted order in the join tree. So, any permutation of the heights on a given diamond corresponds to a permutation of the vertices along a path in P. ■

Two path decompositions P_1 and P_2 (of potentially different complexes and/or height functions) are equivalent if: there is a 1-1 correspondence between the sizes of the constituent paths, and the shrubs are isomorphic.

Lemma 3.9.2. *For all $\mathbb{M} \in \mathbf{F}_P$, the number of heap operations performed by* `build`$(\mathbb{M})$ *is* $O(\sum_{p \in P} |p| \log |p|)$.

Proof: The primary "non-determinism" of the algorithm is the initial painting constructed by `init`$(\mathbb{M})$. We show that regardless of how paint spilling is done, the number of heap operations is bounded as above.

Consider an arbitrary order of the initial paint spilling over the surface. Consider any join on a face of some diamond, which is the anchor point of some connecting child diamond. The join has two up-stars, each of which has exactly one edge. Each edge connects to a maximum and must be colored by that maximum. Hence, the two colors touching this join (according to Definition 3.7.11) are the colors of the apexes of the child and parent diamond.

Take any join v, with two children w_1 and w_2. Suppose w_1 and v belong to the same path in the decomposition. The key is that any color from a maximum in the subtree at w_2 cannot touch any ancestor of v. This subtree is exactly the contour tree of the child diamond attached at v. The base of this diamond is completely contained in a face of the parent diamond. So all colors from the child "drain off" to the base of the parent, and do not touch any joins on the parent diamond.

Hence, $|H_v|$ is at most the size of the path in P containing v. By Lemma 3.7.12, the total number of heap operations is at most $\sum_v \log |H_v|$, completing the proof.

■

The following is the equivalent of Theorem 3.1.3 for join trees, and immediately follows from the previous lemmas.

Theorem 3.9.3. *Consider a rooted tree T and an arbitrary path decomposition P of T. There is a family $\mathbf{F}_P$ of terrains such that any algebraic decision tree computing the contour tree (on $\mathbf{F}_P$) requires $\Omega(\sum_{p\in P} |p| \log |p|)$ time. Furthermore, our algorithm makes $O(\sum_{p\in P} |p| \log |p|)$ comparisons on all these instances.*

Proof: The proof is a basic entropy argument. Any algebraic decision tree that is correct on all of $\mathbf{F}_P$ must distinguish all inputs in this family. By Stirling's approximation, the depth of this tree is $\Omega(\sum_{p_i\in P} |p_i| \log |p_i|)$. Lemma 3.9.2 completes the proof. ■

3.9.2 Contour trees

We first generalize previous terms to the case of contour trees. In this section T will denote an arbitrary contour tree with every internal vertex of degree 3.

For simplicity we now restrict our attention to path decompositions consistent with the raining procedure described in §3.5 (more general decompositions can work, but it is not needed for our purposes).

Definition 3.9.4. A path decomposition, $P(T)$, is called *rain consistent* if its paths can be obtained as follows. Perform an downward BFS from an arbitrary maximum v in T, and mark all vertices encountered. Now recursively run a directional BFS from all vertices adjacent to the current marked set. Specifically, for each BFS run, make it an downward BFS if it is at an odd height in the recursion tree and upward otherwise.

This procedure partitions the vertex set into disjoint rooted subtrees of T, based on which BFS marked a vertex. For each such subtree, now take any partition of the vertices into leaf paths.**

The following is analogous to Lemma 3.9.1, and in particular uses it as a subroutine.

Lemma 3.9.5. *Let P be any rain consistent path decomposition of some contour tree. There is a family of terrains, $\mathbf{F}_P$, all with the same triangulation, such that the size of $\mathbf{F}_P$ is $\Pi_{p_i\in P}(|p_i| - 1)!$, and no two terrains in $\mathbf{F}_P$ define the same contour tree.*

Proof: As P is rain consistent, the paths can be partitioned into sets $P_1, \ldots, P_k$, where P_i is the set of all paths with vertices from a given BFS, as described in Definition 3.9.4. Specifically, let T_i be the subtree of T corresponding to P_i and let r_i be the root vertex of this subtree. Note that the P_i sets naturally define a tree where P_i is the parent of P_j if r_i (i.e. the root of T_i) is adjacent to a vertex in P_j.

As the set P_i is a path decomposition of a rooted binary tree T_i, the terrain construction of Lemma 3.9.1 for P_i is well defined. Actually the only difference is that here the rooted tree is not a full binary tree, and so some of the (non-achor adjacent) faces of the constructed diamonds will be blank. Specifically, these blank faces correspond to the adjacent children of P_i, and they tell us how to connect the terrains of the different P_i's.

**Note that the subtree of the initial vertex is rooted at a maximum. For simplicity we require that the path this vertex belongs to also contains a minimum.

So for each P_i construct a terrain as described in Lemma 3.9.1. Now each T_i is (roughly speaking) a join or a split tree, depending on whether the BFS which produced it was an upward or downward BFS, respectively. As the construction in Lemma 3.9.1 was for join trees, each terrain we constructed for a P_i which came from a split tree, must be flipped upside down. Now we must described how to glue the terrains together.

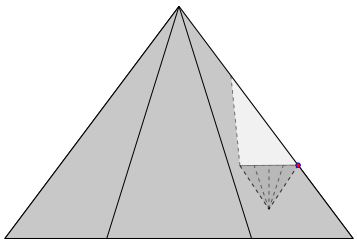


Figure 3.6: A child diamond attached to a parent diamond with opposite orientation.

By construction, the diamonds corresponding to the paths in P_i are connected into a tree structure (i.e. corresponding to the shrub of P_i). Therefore the bottoms of all these diamonds are covered except for the one corresponding to the path containing the root r_i. If r_i corresponds to the initial maximum that the rain consistent path decomposition was defined from, then this will be flat and corresponds to the global outer face. Otherwise, P_i has some parent P_j in which case we connect the bottom of the diamond for r_i to a free face of a diamond in the construction for P_j, specifically, the face corresponding to the vertex in T which r_i is adjacent to. This gluing is done in the same manner as in Lemma 3.9.1, attaching the anchor for the root of P_i directly the corresponding face of P_j, except that now P_i and P_j have opposite orientations. See Figure 3.6.

Just as in Lemma 3.9.1 we now have one fixed terrain structure, such that each different relative ordering of the heights of the join and split vertices on each diamond produces a surface with a distinct contour tree. The specific bound on the size of $\mathbf{F}_P$, defining these distinct contour trees, follows by applying the bound from Lemma 3.9.1 to each P_i. ■

Lemma 3.9.6. *For all $\mathbb{M} \in \mathbf{F}_P$, the number of heap operations is $\Theta(\sum_{p \in P} |p| \log |p|)$*

Proof: This lemma follows immediately from Lemma 3.9.2. The heap operations can be partitioned into the operations performed in each P_i. Apply Lemma 3.9.2 to each of the P_i separately and take the sum. ■

We now restate Theorem 3.1.3, which follows immediately from an entropy argument, analogous to Theorem 3.9.3.

Theorem 3.9.7. *Consider any rain consistent path decomposition P. There exists a family $\mathbf{F}_P$ of terrains ($d = 2$) with the following properties. Any contour tree algorithm makes $\Omega(\sum_{p \in P} |p| \log |p|)$ comparisons in the worst case over $\mathbf{F}_P$. Furthermore, for any terrain in $\mathbf{F}_P$, our algorithm makes $O(\sum_{p \in P} |p| \log |p|)$ comparisons.*

Remark 3.9.8. Note that for the terrains described in this section, the number of critical points is within a constant factor of the total number of vertices. In particular, for this family of terrains, all previous algorithms required $\Omega(n \log n)$ time.

Chapter 4

Weighted Voronoi Diagrams

4.1 Introduction

Informal description of the candidate diagram. Suppose you open your refrigerator one day to discover it is time to go grocery shopping.* Which store you go to will be determined by a number of different factors. For example, what items you are buying, and do you want the cheapest price or highest quality, and how much time you have for this chore. Naturally the distance to the store will also be a factor. On different days which store is the best to go to will differ based on that day's preferences. However, there are certain stores you will never shop at. These are stores which are worse in every way than some other store (i.e., further, more expensive, lower quality, etc.). Therefore, the stores that are relevant and therefore in the *candidate set* are those that are not strictly worse in every way than some other store. Thus, every point in the plane is mapped to a set of stores that a client at that location might use. The *candidate diagram* is the partition of the plane into regions, where each candidate set is the same for all points in the same region. Naturally, if your only consideration is distance, then this is the (classical) Voronoi diagram of the sites. However, here deciding which shop to use is an instance of multi-objective optimization—there are multiple, potentially competing, objectives to be optimized, and the decision might change as the weighting and influence of these objectives mutate over time (in particular, you might decide to do your shopping in different stores for different products). The concept of relevant stores discussed above is often referred as the *Pareto optima*.

Pareto optima in welfare economics. Pareto efficiency, named after Vilfredo Pareto, is a core concept in economic theory and more specifically in welfare economics. Here each point in $\mathbb{R}^d$ represents the corresponding utilities of d players for a particular allocation of finite resources. A point is said to be *Pareto optimal* if there is no other allocation which increases the utility of any individual without decreasing the utility of another. The *First Fundamental Theorem of Welfare Economics* states that any competitive equilibrium (i.e., supply equals demand) is Pareto optimal. The origins of this theorem date back to 1776 with Adam Smith's famous (and controversial) work, "The Wealth of Nations," but was not formally *proven* until the 20th century by Lerner, Lange, and Arrow (see [Fel08]). Naturally such proofs rely on simplifying (i.e., potentially unrealistic) assumptions such as perfect knowledge, or absence of externalities. The *Second Fundamental Theorem of Welfare Economics* states that any Pareto optimum is achievable through lump-sum transfers (i.e. taxation and redistribution). In other words each Pareto optima is a "best solution" under some set of societal preferences, and is achievable through redistribution in one form or another (see [Fel08] for a more in depth discussion).

*Unless you are feeling adventurous enough that day to eat the frozen mystery food stuck to the back of the freezer, which we strongly discourage you from doing.

Pareto optima in computer science. In computational geometry such Pareto optima points relate to the *orthogonal convex hull* [OSW84], which in turn relates to the well known convex hull (the input points that lie on the orthogonal convex hull is a super set of those which lie on the convex hull). Pareto optima are also of importance to the database community [BKS01, HTC13], in which context such points are called *maximal* or *skyline points*. Such points are of interest as they can be seen as the relevant subset of the (potentially much larger) result of a relational database query. The standard example is querying a database of hotels for the cheapest and closest hotel, where naturally hotels which are farther and more expensive than an alternative hotel are not relevant results. There is a significant amount of work on computing these points, see Kung *et al.* [KLP75]. More recently, Godfrey *et al.* [GSG07] compared various approaches for the computation of these points (from a databases perspective), and also introduced their own new external algorithm.[†]

Modeling uncertainty. Recently, there is a growing interest in modeling uncertainty in data. As real data is acquired via physical measurements, noise and errors are introduced. This can be addressed by treating the data as coming from a distribution (e.g., a point location might be interpreted as a center of a Gaussian), and computing desired classical quantities adapted for such settings. Thus, a nearest-neighbor query becomes a probabilistic question—what is the expected distance to the nearest-neighbor? What is the most likely point to be the nearest-neighbor? (See [AAH+13] and references therein for more information.)

This in turn gives rise to the question of what is the expected complexity of geometric structures defined over such data. The case where the data is a set of points, and the locations of the points are chosen randomly was thoroughly investigated (see [SW93, WW93, HR14] and references therein). The problem, when the locations are fixed but the weights associated with the points are chosen randomly, is relatively new. Agarwal *et al.* [AHKS14] showed that for a set of disjoint segments in the plane, if they are being expanded randomly, then the expected complexity of the union is near linear. This result is somewhat surprising as in the worst case the complexity of such a union is quadratic.

Here we are interested in bounding the expected complexity of weighted generalizations of Voronoi diagrams (described below), where the weights (not the site locations) are randomly sampled. Note that the result of Agarwal *et al.* [AHKS14] can be interpreted as bounding the expected complexity of level sets of the multiplicatively weighted Voronoi diagram (of segments). On the other hand, here we want to bound the entire lower envelope (which implies the same bound on any level set). For the special case of multiplicative weighted Voronoi diagrams, a near linear expected complexity bound was provided by Har-Peled and Raichel [HR14]. In this work we consider a much more general class of weighted diagrams which allow multiple weights and non-linear distance functions.

Our contributions

Conceptual contribution. We define formally the *candidate diagram* in Section 4.2.1—a new geometric structure that combines proximity information with utility. For every point x in the plane, the diagram associates a *candidate set* $\mathsf{L}(\mathsf{x})$ of sites that are relevant to x; that is, all the sites that are Pareto optima for x. Putting it differently, a site is not in $\mathsf{L}(\mathsf{x})$ if it is further away from and worse in all parameters than some other site. Significantly, unlike the traditional Voronoi diagram, the candidate diagram allows the user to change their distance function, as long as the function respects the domination relationship. This diagram is

[†]There is of course a lot of other work on Pareto optimal points, from connections to Nash equilibrium to scheduling. We resisted the temptation of including many such references which are not directly related to this work.

a significant extension of the Voronoi diagram, and includes other extensions of Voronoi diagrams as special subcases, like multiplicative weighted Voronoi diagrams. Not surprisingly, the worst case complexity of this diagram can be quite high.

Technical contribution. We consider the case where each site chooses its jth attribute from some distribution $\mathcal{D}_j$ independently for each j. We show that the candidate diagram in expectation has near linear complexity, and that, with high probability, the candidate set has poly-logarithmic size for any point in the plane. In the process we derive several results which are interesting in their own right.

(A) **Low complexity of the minima for random points in the hypercube.** We prove that if n points are sampled from a fixed distribution (see Section 4.2.2 for assumptions on the distribution) over the d-dimensional hypercube then, with probability $1 - 1/n^{\Omega(1)}$, the number of Pareto optima points is $O(\log^{d-1} n)$, which is within a constant factor of the expectation (see Lemma 4.5.4). Previously, this result was only known in a weaker form that is insufficient to imply our other results. Specifically, Bai *et al.* [BDHT05] proved that after normalization the cumulative distribution function of the number of Pareto optima points is normal, up to an additive error $O(1/\operatorname{polylog} n)$. (See [BR10a, BR10b] as well.) In particular, their results (which are quite nice and mathematically involved) only imply the statement with probability $1 - 1/\operatorname{polylog} n$. To the best of our knowledge this result is new—we emphasize, however, that for our purposes a weaker bound of $O(\log^d n)$ is sufficient, and such a weaker result follows readily from the ε-net theorem [HW87] (naturally, this would add a log factor to later results in this chapter).

(B) **Backward analysis with high probability.** To get this result, we prove a lemma providing high probability bounds when applying backwards analysis [Sei93] (see Lemma 4.7.2). Such tail estimates are known in the context of randomized incremental algorithms [CMS93, BCKO08], but our proof is arguably more direct and cleaner, and should be applicable to more cases. See Section 4.2.3 and Section 4.7.

(C) **Overlay of the *k*th order Voronoi cells in randomized incremental construction.** We prove that the overlay of cells during a randomized incremental construction of the kth order Voronoi diagram is of complexity $O\big(k^4 n \log n\big)$ (see Lemma 4.4.7).

(D) **Complexity of the candidate diagram.** Combining the above results carefully yields a near-linear upper bound on the complexity of the candidate diagram (see Theorem 4.6.1).

Outline. In Section 4.2 we formally define our problem and introduce some tools that will be used later on. Specifically, after some required preliminaries, we formally introduce the candidate diagram in Section 4.2.1. The sampling model used is described in detail in Section 4.2.2. In Section 4.2.3, we discuss backward analysis with high-probability bounds.

To bound the complexity of the candidate diagram (i.e., both the size of the planar partition and the total size of the associated candidate sets), in Section 4.3, we use the notion of *proxy set*. Defined formally in Section 4.3.1, it is (informally) an enlarged candidate set. Section 4.3.2 bounds the size of the proxy set using backward analysis, both in expectation and with high probability. Section 4.3.3 shows that mucking around with the proxy set is useful, by proving that the proxy set contains the candidate set, for any point in the plane.

In Section 4.4, we show that the diagram induced by the proxy sets can be interpreted as the arrangement formed by the overlay of cells during the randomized incremental construction of the kth order Voronoi

diagram. To this end, Section 4.4.1 defines the kth order Voronoi diagram, the k *environment* of a site, and states some basic properties of these entities. For our purposes, we need to bound the size of the conflict lists encountered during the randomized incremental construction, and this is done in Section 4.4.2 using the Clarkson-Shor moment technique. Next, in Section 4.4.3, we bound the expected complexity of the proxy diagram.

In Section 4.5, we bound the expected size of the candidate set for any point in the plane. First, in Section 4.5.1, we analyze the number of staircase points of random point sets in hypercubes, and we use this bound, in Section 4.5.2, to bound the size of the candidate set.

In Section 4.6, we put everything together, and prove our main result, showing the desired bound on the complexity of the candidate diagram.

In Section 4.7, we fill in the missing details for the results of Section 4.2.3, proving a high-probability bound for backward analysis.

4.2 Problem definition and preliminaries

Throughout, we assume the reader is familiar with standard computational geometry terms, such as arrangements [SA95], vertical-decomposition [BCKO08], etc. In the same vein, we assume that the variable d, the *dimension*, is a small constant and the O notation hides constants that are potentially exponential (or worse) in d.

A quantity is ***bounded by $O(f)$ with high probability*** with respect to n, if for any large enough constant $\gamma > 0$, there is another constant c depending on γ such that the quantity is at most $c \cdot f$ with probability at least $1 - n^{-\gamma}$. In other words, the bound holds for any sufficiently small polynomial error with the expense of a multiplicative constant factor on the size of the bound. When there's no danger of confusion, we sometimes write $O_{\text{whp}}(f)$ for short.

Definition 4.2.1. Consider two points $\mathsf{p} = (\mathsf{p}_1, \ldots, \mathsf{p}_d)$ and $\mathsf{q} = (\mathsf{q}_1, \ldots, \mathsf{q}_d)$ in $\mathbb{R}^d$. The point p ***dominates*** q (denoted by $\mathsf{p} \preceq \mathsf{q}$) if $\mathsf{p}_i \leq \mathsf{q}_i$, for all i.

Given a point set $\mathsf{P} \subseteq \mathbb{R}^d$, there are several terms for the subset of P that is not dominated, as discussed above, such as *Pareto optima* or *minima*. Here, we use the following term.

Definition 4.2.2. For a point set $\mathsf{P} \subseteq \mathbb{R}^d$, a point $\mathsf{p} \in \mathsf{P}$ is a ***staircase point*** of P if no other point of P dominates it. The set of all such points, denoted by $\overset{\circ}{\preceq}(\mathsf{P})$, is the ***staircase*** of P.

Observe that for a finite point set P, the staircase $\overset{\circ}{\preceq}(\mathsf{P})$ is never empty.

4.2.1 Formal definition of the candidate diagram

Let $\mathsf{S} = \{\mathsf{s}_1, \ldots, \mathsf{s}_n\}$ be a set of n distinct ***sites*** in the plane. For each site s in S, there is an associated list $\alpha = \langle a_1, \ldots, a_d \rangle$ of d real-valued attributes, each in the interval $[0, 1]$. When viewed as a point in the unit hypercube $[0, 1]^d$, this list of attributes is the ***parametric point*** of the site s_i. Specifically, a site is a point in the plane encoding a facility location, while the term *point* is used to refer to the (parametric) point encoding its attributes in $\mathbb{R}^d$.

Preferences. Fix a client location x in the plane. For each site, there are $d+1$ associated variables for the client to consider. Specifically, the client distance to the site, and d additional attributes (e.g., prices of d different products) associated with the site. Conceptually, the goal of the client is to "pay" as little as possible by choosing the best site (e.g., minimize the overall cost of buying these d products together from a site, where the price of traveling the distance to the site is also taken into account).

Definition 4.2.3. A client x has a ***dominating preference*** if for any two sites s, s' in the plane, with parametric points α, $\alpha' \in \mathbb{R}^d$ respectively, the client would prefer the site s over s' if $\|\mathsf{x}-\mathsf{s}\| \leq \|\mathsf{x}-\mathsf{s}'\|$ and $\alpha \preceq \alpha'$ (that is, α dominates α').

Note that a client having a dominating preference does not identify a specific optimum site for the client, but rather a set of potential optimum sites. Specifically, given a client location x in the plane, let its distance to the ith site be $\ell_i = \|\mathsf{x}-\mathsf{s}_i\|$. The set of sites the client might possibly use (assuming the client uses a dominating preference) are the staircase points of the set $\mathsf{P}(\mathsf{x}) = \{(\alpha_1,\ell_1),\ldots,(\alpha_n,\ell_n)\}$ (i.e., we are adding the distance to each site as an additional attribute of the site—this attribute depends on the location of x). The set of sites realizing the staircase of $\mathsf{P}(\mathsf{x})$ (i.e., all the sites relevant to x) is the ***candidate set*** $\mathsf{L}(\mathsf{x})$ of x:

$$\mathsf{L}(\mathsf{x}) = \left\{ \mathsf{s}_i \in \mathsf{S} \;\middle|\; (\alpha_i,\ell_i) \text{ is a staircase point of } \mathsf{P}(\mathsf{x}) \text{ in } \mathbb{R}^{d+1} \right\}. \tag{4.1}$$

The ***candidate cell*** of x is the set of all the points in the plane that have the same candidate set associated with them; that is, $\left\{\mathsf{p} \in \mathbb{R}^2 \mid \mathsf{L}(\mathsf{p}) = \mathsf{L}(\mathsf{x})\right\}$. The decomposition of the plane into these cells is the ***candidate diagram***.

Now, the client x has the candidate set $\mathsf{L}(\mathsf{x})$, and it chooses some site (or potentially several sites) from $\mathsf{L}(\mathsf{x})$ that it might want to use. Note that the client might decide to use different sites for different acquisitions. As an example, consider the case when each site s_i is attached with weights $\alpha_i = (a_{i,1}, a_{i,2})$. If the client x has the preference of choosing the site with smallest value $a_{i,1}\cdot\ell_i$ among all the sites, then this preference is a dominating preference, and therefore the client will choose one of the sites from the candidate list $\mathsf{L}(\mathsf{x})$. (Observe that the preference function corresponds to the weighted Voronoi diagram with respect to the first coordinate of the weights.) Similarly, if the preference function is to choose the smallest value $a_{i,1}\cdot\ell_i^2 + a_{i,2}$ among all the sites (which again is a dominating preference), then this corresponds to a power diagram of the sites.

Complexity of the diagram. The ***complexity*** of a planar arrangement is the total number of edges, faces, and vertices. A candidate diagram can be interpreted as a planar arrangement, and its complexity is defined analogously. The ***space complexity*** of the candidate diagram is the total amount of memory needed to store the diagram explicitly, and is bounded by the complexity of the candidate diagram together with the sum of the sizes of candidate sets over all the faces in the arrangement of the diagram (which is potentially larger by a factor of n, the number of sites). Note, that the space complexity is a somewhat naïve upper bound, as using persistent data-structures might significantly reduce the space needed to store the candidate lists.

Lemma 4.2.4. *Given n sites in the plane, the complexity of the candidate diagram of the sites is $O(n^4)$. The space complexity of the candidate diagram of the sites is $\Omega(n^2)$ and $O(n^5)$.*

Proof: The lower bound is easy, and is left as an exercise to those interested reader. A naïve upper bound of $O(n^5)$ on the space complexity, follows from the fact that (i) all possible pairs of sites induce together

$\binom{n}{2}$ bisectors, (ii) the complexity of the arrangement of the bisectors is $O(n^4)$, and (iii) the candidate set of each face in this arrangement might have n elements inside. ■

We leave the question of closing the gap in the bounds of Lemma 4.2.4 as an open problem for further research.

4.2.2 Sampling model

Fortunately, the situation changes when randomization is involved. Let S be a set of n sites in the plane. For each site $\mathsf{s} \in \mathsf{S}$, a parametric point $\alpha = (\alpha_1, \ldots, \alpha_d)$ is sampled independently from $[0,1]^d$, with the following constraint: each coordinate α_i is sampled from a (continuous) distribution $\mathcal{D}_i$, independently for each coordinate. In particular, the sorted order of the n parametric points by a specific coordinate yields a uniform random permutation (for the sake of simplicity of exposition we assume that all the values sampled are distinct).

Our main result shows that, under the above assumptions, both the complexity and the space complexity of the candidate diagram is near-linear in expectation—see Theorem 4.6.1 for the exact statement.

4.2.3 A short detour into backward analysis

Randomized incremental construction is a powerful technique used by geometric algorithms. Here, one is given a set of elements S (e.g., segments in the plane), and one is interested in computing some structure induced by these elements (e.g., the vertical decomposition formed by the segments). To this end, one computes a random permutation $\Pi = \langle \mathsf{s}_1, \ldots, \mathsf{s}_n \rangle$ of the elements of S, and in the ith iteration one computes the structure V_i induced by the ith prefix $\Pi_i = \langle \mathsf{s}_1, \ldots, \mathsf{s}_i \rangle$ of Π, by inserting the ith element s_i into V_{i-1} and updating it so it becomes V_i (e.g., split all the vertical trapezoids of V_{i-1} that intersect s_i, and merge together adjacent trapezoids with the same floor and ceiling).

In ***backward analysis*** one is interested in computing the probability that a specific object that exists in V_i was actually created in the ith iteration (e.g., a specific vertical trapezoid in the vertical decomposition V_i). If the object of interest is defined by at most b elements of Π_i, for some constant b, then the desired quantity is the probability that s_i is one of these defining elements, which is at most b/i. In some cases, the sum of these probabilities, over the n iterations, count the number of times certain events happen during the incremental construction. However, this yields only a bound in expectation. For a high probability bound, one can not apply this argument directly, as there is a subtle dependency leakage between the corresponding indicator variables involved between different iterations. (Without going into a detailed example, this is because the defining sets of the objects of interest can have different sizes, and these sizes depend on which elements were used in the permutation in earlier iterations.)

Let P be a set of n elements. A ***property*** $\mathcal{P}$ of P is a function that maps any subset X of P to a subset $\mathcal{P}(\mathsf{X})$ of X. Intuitively the elements in $\mathcal{P}(\mathsf{X})$ have some desired property with respect to X (for example, let X be a set of points in the plane, then $\mathcal{P}(\mathsf{X})$ may be those points in X who lie on the convex hull of X). The following corollary to Lemma 4.7.2 provides a high probability bound for backward analysis, and while the proof is an easy application of the Chernoff inequality, it nevertheless significantly simplifies some classical results on randomized incremental construction algorithms. See Section 4.7 for a more detailed discussion and a proof.

Corollary 4.2.5. *Let P be a set of n elements, $c > 1$ and $k \geq 1$ prespecified numbers, and let $\mathcal{P}(\mathsf{X})$ be a property defined over any subset $\mathsf{X} \subseteq \mathsf{P}$. Now, consider a uniform random permutation $\langle \mathsf{p}_1, \ldots, \mathsf{p}_n \rangle$ of P, and let $\mathsf{P}_i = \{\mathsf{p}_1, \ldots, \mathsf{p}_i\}$. Furthermore, assume that, for all i, we have, with probability at least $1 - n^{-c}$, that $|\mathcal{P}(\mathsf{P}_i)| \leq k$. Let X_i be the indicator variable of the event $\mathsf{p}_i \in \mathcal{P}(\mathsf{P}_i)$. Then, for any constant $\gamma \geq 2e$, we have*

$$\mathbf{Pr}\left[\sum_{i=1}^{n} X_i > \gamma \cdot (2k \ln n)\right] \leq n^{-\gamma k} + n^{-c}.$$

(If for all $\mathsf{X} \subseteq \mathsf{P}$ we have that $|\mathcal{P}(\mathsf{X})| \leq k$, then the additional error term n^{-c} is not necessary.)

4.3 The proxy set

Providing a reasonable bound on the complexity of the candidate diagram directly seems challenging. Therefore, we instead define for each point x in the plane a slightly different set, called the *proxy set*. First we prove that the proxy set for each point in the plane has small size (see Lemma 4.3.2 below); then we prove that, with high probability, the proxy set of x contains the candidate set of x for all points x in the plane simultaneously (see Lemma 4.3.4 below).

4.3.1 Definitions

As before, the input is a set of sites S. For each site $\mathsf{s} \in \mathsf{S}$, we randomly pick a parametric point $\alpha \in [0,1]^d$ according to the sampling method described in Section 4.2.2.

Volume ordering. Given a point $\mathsf{p} = (\mathsf{p}_1, \ldots, \mathsf{p}_d)$ in $[0,1]^d$, the ***point volume*** $\mathrm{pv}(\mathsf{p})$ of point p is defined to be $\mathsf{p}_1\mathsf{p}_2 \cdots \mathsf{p}_d$; that is, the volume of the hyperrectangle with p and the origin as a pair of opposite corners. When p is specifically the associated parametric point of an input site s, we refer to the point volume of p as the ***parametric volume*** of s. Observe that if point p dominates another point q then p must have smaller point volume (i.e., p lies in the hyperrectangle defined by q).

The ***volume ordering*** of sites in S is a permutation $\langle \mathsf{s}_1, \ldots, \mathsf{s}_n \rangle$ ordered by increasing parametric volume of the sites; that is, $\mathrm{pv}(\alpha_1) \leq \mathrm{pv}(\alpha_2) \leq \ldots \leq \mathrm{pv}(\alpha_n)$, where α_i is the parametric point of s_i. If α_i dominates α_j then s_i precedes s_j in the volume ordering. So if we add the sites in volume ordering, then when we add the ith site s_i we can ignore all later sites when determining its region of influence—that is, the region of points whose candidate set s_i belongs to—as no later sites can have their parametric point dominate the one of s_i.

k* nearest neighbors.** For a set of sites S and a point x in the plane, let $\mathsf{d}_k(\mathsf{x}, \mathsf{S})$ denote the ***kth nearest neighbor distance to x in S; that is, the kth smallest value in the multiset $\left\{ \|\mathsf{x} - \mathsf{s}\| \,\middle|\, \mathsf{s} \in \mathsf{S} \right\}$. The ***k nearest neighbors*** to x in S is the set

$$\mathsf{N}_k(\mathsf{x}, \mathsf{S}) = \left\{ \mathsf{s} \in \mathsf{S} \,\middle|\, \|\mathsf{x} - \mathsf{s}\| \leq \mathsf{d}_k(\mathsf{x}, \mathsf{S}) \right\}.$$

Definition 4.3.1. Let S be a set of sites in the plane, and let $\mathsf{V}(\mathsf{S}) = \langle \mathsf{s}_1, \ldots, \mathsf{s}_n \rangle$ be the volume ordering of S. Let S_i denote the underlying set of the ith prefix $\langle \mathsf{s}_1, \ldots, \mathsf{s}_i \rangle$ of $\mathsf{V}(\mathsf{S})$. For a parameter k and a point x in

the plane, the ***kth proxy set*** of x is the set of sites

$$\mathsf{C}_k(\mathsf{x}, \mathsf{S}) = \bigcup_{i=1}^{n} \mathsf{N}_k(\mathsf{x}, \mathsf{S}_i),$$

In words, site s is in $\mathsf{C}_k(\mathsf{x}, \mathsf{S})$ if it is one of the k nearest neighbors to point x in some prefix of the volume ordering $\mathsf{V}(\mathsf{S})$.

4.3.2 Bounding the size of the proxy set

The desired bound now follows by using backward analysis and Corollary 4.2.5.

Lemma 4.3.2. *Let S be a set of n sites in the plane, and let $k \geq 1$ be a fixed parameter. Then we have $|\mathsf{C}_k(\mathsf{x}, \mathsf{S})| = O_{whp}(k \log n)$ simultaneously for all points x in the plane.*

Proof: Fix a point x in the plane. A site s gets added to the proxy set $\mathsf{C}_k(\mathsf{x}, \mathsf{S})$ if site s is one of the k nearest neighbors of x among the underlying set S_i of some prefix of the volume ordering of S. Therefore a direct application of Corollary 4.2.5 implies (by setting $\mathcal{P}(\mathsf{S}_i)$ to be $\mathsf{N}_k(\mathsf{x}, \mathsf{S}_i)$), with high probability, that $\left|\mathsf{C}_k(\mathsf{x}, \mathsf{S})\right| = O(k \log n)$.

Furthermore, this holds for all points in the plane simultaneously. Indeed, consider the arrangement determined by the $\binom{n}{2}$ bisectors formed by all the pairs of sites in S. This arrangement is a simple planar map with $O\left(n^4\right)$ vertices and $O\left(n^4\right)$ faces. Observe that within each face the proxy set cannot change since all points in this face have the same ordering of their distances to the sites in S. Therefore, picking a representative point from each of these $O\left(n^4\right)$ faces, applying the high probability bound to each one of them, and then applying the union bound implies the claim. ■

4.3.3 The proxy set contains the candidate set

The following corollary is implied by a careful (but straightforward) integration argument (see Lemma 4.8.2).

Corollary 4.3.3. *Let $F_d(\Delta)$ be the total measure of the points $\mathsf{p} \in [0,1]^d$, such that the point volume $\mathrm{pv}(\mathsf{p})$ is at most Δ. Then for $\Delta \geq (\log n)/n$ we have $F_d(\Delta) = \Theta(\Delta \log^{d-1} n)$; in particular, $F_d(\log n/n) = \Theta((\log^d n)/n)$.*

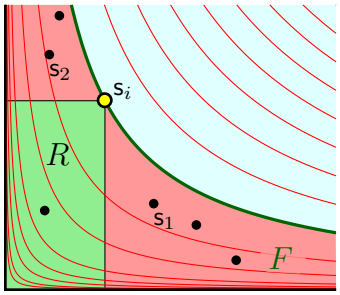
Lemma 4.3.4. *Let S be a set of n sites in the plane, and let $k = \Theta(\log^d n)$ be a fixed parameter. For all points x in the plane, we have that $\mathsf{L}(\mathsf{x}) \subseteq \mathsf{C}_k(\mathsf{x}, \mathsf{S})$, and this holds with high probability.*

Proof: Fix a point x in the plane, and let s_i be any site *not* in $\mathsf{C}_k(\mathsf{x}, \mathsf{S})$, and let α_i be the associated parametric point. We claim that, with high probability, the site s_i is dominated by some other site which is closer to x, and hence by the definition of dominating preference (Definition 4.2.3), s_i cannot be a site used by x (and thus $\mathsf{s}_i \notin \mathsf{L}(\mathsf{x})$). Taking the union bound over all sites not in $\mathsf{C}_k(\mathsf{x}, \mathsf{S})$ then implies this claim.

By Corollary 4.3.3, the total measure of the points in $[0,1]^d$ with point volume at most $\Delta = \log n/n$ is $\Theta((\log^d n)/n)$. As such, by Chernoff's inequality, with high probability, there are $K = O(\log^d n)$ sites in S such that their parametric points have point volume smaller than Δ. In particular, by choosing k to be sufficiently large (i.e., $k > K$), the underlying set S_k of the kth prefix of the volume ordering of S will contain

all these small point volume sites, and since $\mathsf{S}_k \subseteq \mathsf{C}_k(\mathsf{x}, \mathsf{S})$, so will $\mathsf{C}_k(\mathsf{x}, \mathsf{S})$. Therefore, from this point on, we will assume that $\mathsf{s}_i \notin \mathsf{C}_k(\mathsf{x}, \mathsf{S})$ and $\Delta_i = \mathrm{pv}(\alpha_i) = \Omega(\log n/n)$.

Now any site s with smaller parametric volume than s_i is in the (unordered) prefix S_i. In particular, the k nearest neighbors $\mathsf{N}_k(\mathsf{x}, \mathsf{S}_i)$ of x in S_i all have smaller parametric volume than s_i. Hence $\mathsf{C}_k(\mathsf{x}, \mathsf{S})$ contains k points all of which have smaller parametric volume than s_i, and which are closer to x. Therefore, the claim will be implied if one of these k points dominates s_i.



The probability of a site s (that is closer to x than s_i) with parametric point α to dominate s_i is the probability that $\alpha \preceq \alpha_i$ given that $\alpha \in F$, where $F = \left\{\alpha \in [0,1]^d \,\middle|\, \mathrm{pv}(\alpha) \leq \Delta_i\right\}$. By Corollary 4.3.3, we have $\mathrm{vol}(F) = F_d(\Delta_i) = \Theta(\Delta_i \log^{d-1} n)$. The probability that a random parametric point in $[0,1]^d$ dominates α_i is exactly $\Delta_i = \mathrm{pv}(\alpha_i)$, and as such the desired probability $\mathbf{Pr}\left[\alpha \preceq \alpha_i \mid \alpha \in F\right]$ is equal to $\Delta_i / F_d(\Delta_i)$, which is $O(1/\log^{d-1} n)$. This is depicted in the figure on the right—the probability of a random point picked uniformly from the region F under the curve $y = \Delta_i/x$, induced by s_i, to fall in the rectangle R.

As the parametric point of each one of the k points in $\mathsf{N}_k(\mathsf{x}, \mathsf{S}_i)$ has equal probability to be anywhere in F, this implies the expected number of points in $\mathsf{N}_k(\mathsf{x}, \mathsf{S}_i)$ which dominate s_i is $\mathbf{Pr}\left[\alpha \preceq \alpha_i \mid \alpha \in F\right] \cdot k = \Theta(\log n)$. Therefore by making k sufficiently large, Chernoff's inequality implies the desired result.

It follows that this holds, for all points in the plane simultaneously, by following the argument used in the proof of Lemma 4.3.2. ■

4.4 Bounding the complexity of the kth order proxy diagram

The ***kth proxy cell*** of x is the set of all the points in the plane that have the same kth proxy set associated with them; that is, $\left\{\mathsf{p} \in \mathbb{R}^2 \,\middle|\, \mathsf{C}_k(\mathsf{p}, \mathsf{S}) = \mathsf{C}_k(\mathsf{x}, \mathsf{S})\right\}$. The decomposition of the plane into these faces is the ***kth order proxy diagram***. In this section, our goal is to prove that the expected total diagram complexity of the kth order proxy diagram is $O\left(k^4 n \log n\right)$. To this end, we bound the complexity by relating it to the overlay of star-polygons that rise out of the kth order Voronoi diagram.

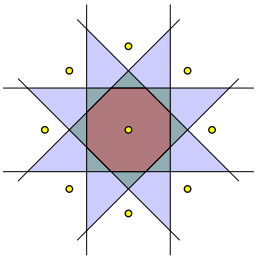
4.4.1 Preliminaries

The kth order Voronoi diagram

Let S be a set of n sites in the plane. The ***kth order Voronoi diagram*** of S is a partition of the plane into faces such that each cell is the locus of points which have the same set of k nearest sites of S (the internal ordering of these k sites, by distance to the query point, may vary within the cell). It is well known that the worst case complexity of this diagram is $\Theta(k(n-k))$ (see [AKL13, Section 6.5]).

Environments and overlays. For a site s in S and a constant k, the ***k environment*** of s, denoted by $\mathrm{env}_k(\mathsf{s}, \mathsf{S})$, is the set of all the points in the plane such that s is one of their k nearest neighbors in S; that is,

$$\mathrm{env}_k(\mathsf{s}, \mathsf{S}) = \left\{\mathsf{x} \in \mathbb{R}^2 \mid \mathsf{s} \in \mathsf{N}_k(\mathsf{x}, \mathsf{S})\right\}.$$



See the figure on the right for an example what this environment looks like for different values of k. One can view the k environment of s as the union of the kth order Voronoi cells which have s as one of the k nearest sites. Observe that the overlay of the polygons $\text{env}_k(\mathsf{s}_1, \mathsf{S}), \ldots, \text{env}_k(\mathsf{s}_n, \mathsf{S})$ produces the kth order Voronoi diagram of S. Indeed, for any point x in the plane, if s is one of x's k nearest sites, then by definition x is covered by $\text{env}_k(\mathsf{s}, \mathsf{S})$; and conversely if x is covered by $\text{env}_k(\mathsf{s}, \mathsf{S})$ then s is one of x's k nearest neighbors. It is also known that each k environment of a site is a star-shaped polygon; this was previously observed by Aurenhammer and Schwarzkopf [AS92].

Lemma 4.4.1. *The set* $\text{env}_k(\mathsf{s}, \mathsf{S})$ *is a star shaped polygon with respect to the point* s.

Proof: Consider the set of all $n-1$ bisectors determined by s and any other site in S. For any point x in the plane, $\mathsf{p} \in \text{env}_k(\mathsf{s}, \mathsf{S})$ holds if the segment from s to p crosses at most $k-1$ of these bisectors. The star-shaped property follows as when walking along any ray emanating from s, the number of bisectors crossed is a monotonically increasing function of distance from s. Moreover, $\text{env}_k(\mathsf{s}, \mathsf{S})$ is a polygon as its boundary is composed of subsets of straight line bisectors. ■

Going back to our original problem, let k be a fixed constant, and let $\mathsf{V}(\mathsf{S}) = \langle \mathsf{s}_1, \ldots, \mathsf{s}_n \rangle$ be the volume ordering of S. As usual, we use S_i to denote the unordered ith prefix of $\mathsf{V}(\mathsf{S})$. Let $\text{env}_i := \text{env}_k(\mathsf{s}_i, \mathsf{S}_i)$, that is, the union of all the cells in the kth order Voronoi diagram of S_i where s_i is one of the k nearest neighbors.

Observation 4.4.2. The arrangement determined by the overlay of the polygons $\text{env}_1, \ldots, \text{env}_n$ is the kth order proxy diagram of S.

Arrangements of planes and lines

One can interpret the kth order Voronoi diagram in terms of an arrangement of planes in $\mathbb{R}^3$. Specifically, "lift" each site to the paraboloid $\big(x, y, -(x^2+y^2)\big)$. Consider the arrangement of planes H tangent to the paraboloid at the lifted locations of the sites. A point on the union of these planes is of *level* k if there are exactly k planes strictly below it. The ***k-level*** is the closure of the set of points of level k.[‡] (For any set of n hyperplanes in $\mathbb{R}^d$, one can define k-levels of arrangement of hyperplanes analogously.) Consider a point x in the xy-plane. The decreasing z-ordering of the planes vertically below x is the same as the ordering, by decreasing distance from x, to the corresponding sites. Hence, let $\mathsf{E}_k(\mathsf{H})$ denote the set of edges in the arrangement H on the k-level, where an edge is a maximal portion of the k-level that lies on the intersection of two planes (induced by two sites). Then the projection of the edges in $\mathsf{E}_{k-1}(\mathsf{H})$ onto the xy-plane results in the edges of the kth order Voronoi diagram. When there is no risk of confusion, we also use $\mathsf{E}_k(\mathsf{S})$ to denote the set of edges in $\mathsf{E}_k(\mathsf{H})$, where H is obtained by lifting the sites in S to the paraboloid and taking the tangential planes, as described above.

We need the notion of k-levels of arrangement of lines as well. For a set of lines L in the plane, let $\mathsf{E}_k(L)$ denote the set of edges in the arrangement of L on the k-level.

Lemma 4.4.3. *Let L be a set of n lines in general position in the plane. Fix any arbitrary insertion ordering of the lines in L, and let m be the total number of distinct vertices on the k-level of the arrangement of L seen over all iterations of this insertion process. We have $m = O(nk)$.*

[‡]The lifting of the sites to the paraboloid $z = -(x^2+y^2)$ is done so that the definition of the k-level coincide with the standard definition.

Proof:Let ℓ_i be the ith line inserted, and let L_i be the set of the first i inserted lines. Any new vertex on the kth level created by the insertion must lie on ℓ_i. However, by Lemma 4.8.1 at most $k+2$ edges from $\mathsf{E}_k(L_i)$ can lie on ℓ_i. As each such edge has at most two endpoints, the insertion of ℓ_i contributes $O(k)$ vertices to the k-level. The bound now follows by summing over all n lines. ■

4.4.2 Bounding the size of the below conflict-lists

The below conflict lists

Let H be a set of n planes in general position in $\mathbb{R}^3$. (For example, in the setting of the kth order Voronoi diagram, H is the set of planes that are tangent to the paraboloid at the lifted locations of the sites.) For any subset $\mathsf{R} \subseteq \mathsf{H}$, let $\mathsf{V}_k(\mathsf{R})$ denote the vertices on the k-level of the arrangement of R. Similarly, let $\mathsf{V}_{\leq k}(\mathsf{R}) = \bigcup_{i=0}^{k} \mathsf{V}_k(\mathsf{R})$ be the set of vertices of level at most k in the arrangement of R, and let $\mathsf{E}_{\leq k}(\mathsf{R})$ be the set of edges of level at most k in the arrangement of R. For a vertex v in the arrangement of R, the ***below conflict list*** $\mathsf{B}(v)$ of v is the set of those planes in H that lie strictly below v; denote b_v to be $|\mathsf{B}(v)|$. For an edge e in the arrangement of R, the ***below conflict list*** $\mathsf{B}(e)$ of e is the set of planes of H which lie below e (i.e., there is at least one point on e that lies above such a plane); denote b_e to be $|\mathsf{B}(e)|$. Our purpose here is to bound the quantities $\mathbf{E}\left[\sum_{v \in \mathsf{V}_{\leq k}(\mathsf{R})} \mathsf{b}_v\right]$ and $\mathbf{E}\left[\sum_{e \in \mathsf{E}_{\leq k}(\mathsf{R})} \mathsf{b}_e\right]$.

The Clarkson-Shor technique

In the following, we use the Clarkson-Shor technique [CS89], stated here without proof (see [Har11] for details). Specifically, let S be a set of elements such that any subset $\mathsf{R} \subseteq \mathsf{S}$ defines a corresponding set of objects $\mathcal{T}(\mathsf{R})$ (e.g., S is a set of planes and any subset $\mathsf{R} \subseteq \mathsf{S}$ induces a set of vertices in the arrangement of planes R). Each potential object, τ, has a defining set and a stopping set. The ***defining set***, $D(\tau)$, is a subset of S that must appear in R in order for the object to be present in $\mathcal{T}(\mathsf{R})$. We require that the defining set has at most a constant size for every object. The ***stopping set***, $\kappa(\tau)$, is a subset of S such that if any of its member appear in R then τ is not present in $\mathcal{T}(\mathsf{R})$. We also naturally require that $\kappa(\tau) \cap D(\tau) = \emptyset$ for all object τ. Surprisingly, this already implies the following.

Theorem 4.4.4 (Bounded Moments [CS89]). *Using the above notation, let S be a set of n elements, and let R be a random sample of size r from S. Let $f(\cdot)$ be a polynomially bounded function[§]. We have that*

$$\mathbf{E}\left[\sum_{\tau \in \mathcal{T}(\mathsf{R})} f\Big(|\kappa(\tau)|\Big)\right] = O\Big(\mathbf{E}\Big[|\mathcal{T}(\mathsf{R})|\Big]\, f\Big(\frac{n}{r}\Big)\Big),$$

where the expectation is taken over random sample R.

Bounding the below conflict-lists

The technical challenge. The proof of the next lemma is technically interesting as it does not follow in a straightforward fashion from the Clarkson-Shor technique. Indeed, the below conflict list is *not* the standard conflict list. Specifically, the decision whether a vertex v in the arrangement of R is of level at most k is a "global" decision of R, and as such the defining set of this vertex is neither of constant size, nor unique, as required to use the Clarkson-Shor technique. If this was the only issue, the extension by Agarwal

[§]A function f is ***polynomially bounded***, if (i) f is monotonically increasing, and (ii) $f(n) = n^{O(1)}$.

et al. [AMS98] could handle this situation. However it is even worse: a plane $\mathsf{h} \in \mathsf{H} \setminus \mathsf{R}$ that is below a vertex $v \in \mathsf{V}_{\leq k}(\mathsf{R})$ is not necessarily conflicting with v (i.e., in the stopping set of v)—as its addition to R will not necessarily remove v from $\mathsf{V}_{\leq k}(\mathsf{R} \cup \{\mathsf{h}\})$.

The solution. Since the standard technique fails in this case, we need to perform our argument somehow indirectly. Specifically, we use a second random sample and then deploy the Clarkson-Shor technique on this smaller sample—this is reminiscent of the proof bounding the size of $\mathsf{V}_{\leq k}(\mathsf{H})$ by Clarkson-Shor [CS89], and the proof of the exponential decay lemma of Chazelle and Friedman [CF90].

Lemma 4.4.5. *Let k be a fixed constant, and let R be a random sample (without replacement) of size r from a set of H of n planes in $\mathbb{R}^3$, we have $\mathbf{E}\Big[\sum_{v \in \mathsf{V}_{\leq k}(\mathsf{R})} \mathsf{b}_v\Big] = O\big(nk^3\big)$.*

Proof: From the sake of simplicity of exposition, let us assume that the sampling here is done by picking every element into the random sample R with probability r/n. Doing the computations below using sampling without replacement (so we get the exact size) requires modifying the calculations so that the probabilities are stated using binomial coefficients—this makes the calculation messier, but the results remain the same. See [Sha03] for further discussion of this minor issue.

So, fix a sample R and sample each plane in R, with probability $1/k$, to be in R'. Let us consider the probability that a vertex $v \in \mathsf{V}_{\leq k}(\mathsf{R})$ ends up on the lower envelope of R'. A lower bound can be achieved by the standard argument of Clarkson-Shor. Specifically, if a vertex v is on the lower envelope then its three defining planes must be in R' and moreover as $v \in \mathsf{V}_{\leq k}(\mathsf{R})$ by definition there are at most k planes below it that must not be in R'. So let X_v be an indicator variable for whether v appears on the lower envelope of R', we then have

$$\mathbf{E}_{\mathsf{R}'}\left[X_v \;\middle|\; \mathsf{R}\right] \geq \frac{1}{k^3}(1-1/k)^k \geq \frac{1}{e^2 k^3}.$$

Observe that

$$\mathbf{E}_{\mathsf{R}'}\left[\sum_{v \in \mathsf{V}_0(\mathsf{R}')} \mathsf{b}_v\right] = \mathbf{E}_{\mathsf{R}}\left[\mathbf{E}_{\mathsf{R}'}\left[\sum_{v \in \mathsf{V}_0(\mathsf{R}')} \mathsf{b}_v \;\middle|\; \mathsf{R}\right]\right] = \mathbf{E}_{\mathsf{R}}\left[\mathbf{E}_{\mathsf{R}'}\left[\sum_{v \in \mathsf{V}_{\leq k}(\mathsf{R})} X_v \mathsf{b}_v \;\middle|\; \mathsf{R}\right]\right]. \tag{4.2}$$

Fixing the value of R, the lower bound above implies

$$\mathbf{E}_{\mathsf{R}'}\left[\sum_{v \in \mathsf{V}_{\leq k}(\mathsf{R})} X_v \mathsf{b}_v \;\middle|\; \mathsf{R}\right] = \sum_{v \in \mathsf{V}_{\leq k}(\mathsf{R})} \mathbf{E}_{\mathsf{R}'}\left[X_v \mathsf{b}_v \;\middle|\; \mathsf{R}\right] = \sum_{v \in \mathsf{V}_{\leq k}(\mathsf{R})} \mathsf{b}_v \mathbf{E}_{\mathsf{R}'}\left[X_v \;\middle|\; \mathsf{R}\right] \geq \sum_{v \in \mathsf{V}_{\leq k}(\mathsf{R})} \frac{\mathsf{b}_v}{e^2 k^3},$$

by linearity of expectations and as b_v is a constant for v. Plugging this into Eq. (4.2), we have

$$\mu = \mathbf{E}_{\mathsf{R}'}\left[\sum_{v \in \mathsf{V}_0(\mathsf{R}')} \mathsf{b}_v\right] \geq \mathbf{E}_{\mathsf{R}}\left[\sum_{v \in \mathsf{V}_{\leq k}(\mathsf{R})} \frac{\mathsf{b}_v}{e^2 k^3}\right] = \frac{1}{e^2 k^3}\mathbf{E}_{\mathsf{R}}\left[\sum_{v \in \mathsf{V}_{\leq k}(\mathsf{R})} \mathsf{b}_v\right]. \tag{4.3}$$

Observe that R' is a random sample of R which by itself is a random sample of H. As such, one can interpret R' as a direct random sample of H. The lower envelope of a set of planes has linear complexity, and for a vertex v on the lower envelope of R' the set $\mathsf{B}(v)$ is the standard conflict list of v. As such,

Theorem 4.4.4 implies

$$\mu = \mathbf{E}_{\mathsf{R}'}\left[\sum_{v \in \mathsf{V}_0(\mathsf{R}')} \mathsf{b}_v\right] = O\left(|\mathsf{R}'| \cdot \frac{n}{|\mathsf{R}'|}\right) = O(n).$$

Plugging this into Eq. (4.3) implies the claim. ∎

Corollary 4.4.6. *Let R be a random sample (without replacement) of size r from a set H of n planes in $\mathbb{R}^3$. We have that $\mathbf{E}_{\mathsf{R}}\left[\sum_{e \in \mathsf{E}_{\leq k}(\mathsf{R})} \mathsf{b}_e\right] = O\left(nk^3\right)$.*

Proof: Under general position assumption every vertex in the arrangement of H is adjacent to 8 edges. For an edge $e = uv$, it is easy to verify that $\mathsf{B}(e) \subseteq \mathsf{B}(u) \cup \mathsf{B}(v)$, and as such we charge the conflict list of e to its two endpoints u and v, and every vertex get charged $O(1)$ times. Now, the claim follows by Lemma 4.4.5.

This argument fails to capture edges that are rays in the arrangement, but this is easy to overcome by clipping the arrangement to a bounding box that contains all the vertices of the arrangement. We omit the easy but tedious details. ∎

4.4.3 Putting it all together

The proof of the following lemma is similar in spirit to the argument of Har-Peled and Raichel [HR14].

Lemma 4.4.7. *Let S be a set of n sites in the plane, $\langle \mathsf{s}_1, \ldots, \mathsf{s}_n \rangle$ be a random permutation of S, and let k be a fixed number. The expected complexity of arrangement determined by the overlay of the polygons $\mathrm{env}_1, \ldots, \mathrm{env}_n$ (and therefore, the expected complexity of the kth order proxy diagram) is $O\left(k^4 n \log n\right)$, where $\mathrm{env}_i = \mathrm{env}_k(\mathsf{s}_i, \mathsf{S}_i)$ and $\mathsf{S}_i = \{\mathsf{s}_1, \ldots, \mathsf{s}_i\}$ is the underlying set of the ith prefix of $\langle \mathsf{s}_1, \ldots, \mathsf{s}_n \rangle$, for each i.*

Proof: As the arrangement of the overlay of the polygons $\mathrm{env}_1, \ldots, \mathrm{env}_n$ is a planar map it suffices to bound the number of edges in the arrangement. For each i, let $\mathsf{E}(\mathrm{env}_i)$ be the edges in $\mathsf{E}_{\leq k}(\mathsf{S}_i)$ that appear on the boundary of env_i (for simplicity we do not distinguish between edges in $\mathsf{E}_{\leq k}(\mathsf{S}_i)$ in $\mathbb{R}^3$ and their projection in the plane). Created in the ith iteration, an edge e in $\mathsf{E}(\mathrm{env}_i)$ is going to be broken into several pieces in the final arrangement of the overlay. Let n_e be the number of such pieces that arise from e.

Fix an integer i. As S_i is fixed, $\mathsf{B}(e)$ is also fixed, for all $e \in \mathsf{E}_{\leq k}(\mathsf{S}_i)$. Moreover, we claim that $n_e \leq c \cdot k\mathsf{b}_e$ for some constant c. Indeed, n_e counts the number of future intersections of e with the edges of $\mathsf{E}(\mathrm{env}_j)$, for any $j > i$. As the edge e is on the k-level at the time of creation, and the edges in $\mathsf{E}(\mathrm{env}_j)$ are on the k-level when they are being created (in the future), these edges must lie below e. Namely, any future intersect on e are caused by intersections of (pairs of) planes in $\mathsf{B}(e)$. So consider the intersection of all planes in $\mathsf{B}(e)$ on the vertical plane containing e. On this vertical plane, $\mathsf{B}(e)$ is a set of b_e lines, whose insertion ordering is defined by the suffix of the permutation $\langle \mathsf{s}_{i+1}, \ldots, \mathsf{s}_n \rangle$. Now any edge of $\mathsf{E}(\mathrm{env}_j)$, for some $j > i$, that intersects e must appear as a vertex on the k-level at some point during the insertion of these lines. However, by Lemma 4.4.3, applied to the lines of $\mathsf{B}(e)$ on the vertical plane of e, under any insertion ordering there are at most $O(k\mathsf{b}_e)$ vertices that ever appear on the k-level.

For an edge $e \in \mathsf{E}_{\leq k}(\mathsf{S}_i)$, let X_e be the indicator variable of the event that e was created in the ith iteration, and furthermore, lies on the boundary of env_i. Observe that $\mathbf{E}[X_e \mid \mathsf{S}_i] \leq 4/i$, as an edge appears for the first time in round i only if one of its (at most) four defining sites was the ith site inserted.

Let $Y_i = \sum_{e \in \mathsf{E}(\mathrm{env}_i)} n_e = \sum_{e \in \mathsf{E}_{\leq k}(\mathsf{S}_i)} n_e X_e$ be the total (forward) complexity contribution to the final arrangement of edges added in round i. We thus have

$$\mathbf{E}\Big[Y_i \;\Big|\; \mathsf{S}_i\Big] = \mathbf{E}\left[\sum_{e \in \mathsf{E}_{\leq k}(\mathsf{S}_i)} n_e X_e \;\middle|\; \mathsf{S}_i\right] \leq \mathbf{E}\left[\sum_{e \in \mathsf{E}_{\leq k}(\mathsf{S}_i)} ck\mathsf{b}_e X_e \;\middle|\; \mathsf{S}_i\right] = \sum_{e \in \mathsf{E}_{\leq k}(\mathsf{S}_i)} ck\mathsf{b}_e \,\mathbf{E}\Big[X_e \;\Big|\; \mathsf{S}_i\Big]$$
$$\leq \frac{4ck}{i} \sum_{e \in \mathsf{E}_{\leq k}(\mathsf{S}_i)} \mathsf{b}_e.$$

The total complexity of the overlay arrangement of the polygons $\mathrm{env}_1, \ldots, \mathrm{env}_n$ is asymptotically bounded by $\sum_i Y_i$, and so by Corollary 4.4.6 we have

$$\mathbf{E}\Big[\textstyle\sum_i Y_i\Big] = \sum_i \mathbf{E}\Big[\mathbf{E}\Big[Y_i \;\Big|\; \mathsf{S}_i\Big]\Big] \leq \sum_i \mathbf{E}\left[\frac{4ck}{i} \sum_{e \in \mathsf{E}_{\leq k}(\mathsf{S}_i)} \mathsf{b}_e\right] = O\left(\sum_i \frac{nk^4}{i}\right) = O\Big(k^4 n \log n\Big).$$

■

4.5 On the expected size of the staircase

4.5.1 Number of staircase points

The two dimensional case

Corollary 4.5.1. *Let P be a set of n points sampled uniformly at random from the unit square $[0,1]^2$. Then the number of staircase points $\text{°}(\mathsf{P})$ in P is $O_{whp}(\log n)$.*

Proof: If we order the points in P by increasing x-coordinate, then the staircase points are exactly the points which have the smallest y-values out of all points in their prefix in this ordering. As the x-coordinates are sampled uniformly at random, this ordering is a random permutation $\langle y_1, \ldots, y_n \rangle$ of the y-values Y. Let X_i be the indicator variable of the event that y_i is the smallest number in $\mathsf{Y}_i = \{y_1, \ldots, y_i\}$ for each i. By setting property $\mathcal{P}(\mathsf{Y}_i)$ to be the smallest number in the prefix Y_i, we have $\sum_{i=1}^{n} X_i = O(\log n)$ with high probability by Corollary 4.2.5. ■

Higher dimensions

Lemma 4.5.2. *Fix a dimension $d \geq 2$. Let m and n be parameters, such that $m \leq n$. Let $\mathsf{Q} = \langle \mathsf{q}_1, \ldots, \mathsf{q}_m \rangle$ be an ordered set of m points picked randomly from $[0,1]^d$ as described in Section 4.2.2. Assume that we have $\big|\text{°}(\mathsf{Q}_i)\big| = O(c_d \log^{d-1} n)$, with high probability with respect to m for all i simultaneously, where $\mathsf{Q}_i = \{\mathsf{q}_1, \ldots, \mathsf{q}_i\}$ is underlying set of the ith prefix of Q. Then, the set $\text{°} = \bigcup_{i=1}^{m} \text{°}(\mathsf{Q}_i)$ has size $O(c_d \log^d n)$, with high probability with respect to m.*

Proof: Let $\big|\text{°}(\mathsf{Q}_i)\big| \leq c' \cdot c_d \ln^{d-1} n$ with probability $1 - m^{-c}$ for large enough constant c and some constant c' depending on c. By setting $\mathcal{P}(\mathsf{Q}_i) = \text{°}(\mathsf{Q}_i)$, we have that $\mathbf{Pr}\big[\big|\text{°}\big| > \gamma(2k \ln m)\big] \leq m^{-\gamma k} + m^{-c}$ for $k = c' \cdot c_d \ln^{d-1} n$ and any $\gamma \geq 2e$, by Corollary 4.2.5. Setting $\gamma = 2e \ln n / \ln m$ implies the claim. ■

Lemma 4.5.3. *Fix a dimension $d \geq 2$. Let m, n be parameters, such that $m \leq n$. Let P be a set of m points picked randomly from $[0,1]^d$ as described in Section 4.2.2. Then, $\big|\text{°}(\mathsf{P})\big| = O(c_d \log^{d-1} n)$ holds, with high probability with respect to m, for some constant c_d that depends only on d.*

Proof: The argument follows by induction on dimension. The two-dimensional case follows from Corollary 4.5.1. Assume we have proven the claim for all dimension smaller than d.

Now, sort P by increasing value of the dth coordinate, and let $\mathsf{p}_i = (\mathsf{q}_i, \ell_i)$ be the ith point in P in this order for each i, where q_i is a $(d-1)$-dimensional vector and ℓ_i is the value of the dth coordinate of p_i. Observe that the points $\mathsf{q}_1, \ldots, \mathsf{q}_m$ are randomly, uniformly, and independently picked from the hypercube $[0,1]^{d-1}$. Now, if p_i is a minima point of P, then it is a minima point of $\{\mathsf{p}_1, \ldots, \mathsf{p}_i\}$. But this implies that q_i is a minima point of $\mathsf{Q}_i = \{\mathsf{q}_1, \ldots, \mathsf{q}_i\}$ as well; namely, $\mathsf{q}_i \in \text{°} := \bigcup_{i=1}^{m} \text{°}(\mathsf{Q}_i)$. This implies that $\left|\text{°}(\mathsf{P})\right| \leq \left|\text{°}\right|$. Now, applying induction hypothesis on each Q_i in dimension $d-1$ we have $\left|\text{°}(\mathsf{Q}_i)\right| = O(c_{d-1} \log^{d-2} n)$ holds for all i, with high probability with respect to m; and plugging it into Lemma 4.5.2 we have $\left|\text{°}(\mathsf{P})\right| \leq \left|\text{°}\right| = O(c_{d-1} \log^{d-1} n)$, with high probability with respect to m. Choosing a proper constant c_d now implies the claim. ■

Lemma 4.5.4. *Fixed a dimension $d \geq 2$. Let $\mathsf{Q} = \langle \mathsf{q}_1, \ldots, \mathsf{q}_n \rangle$ be an ordered set of n points picked randomly from $[0,1]^d$ (as described in Section 4.2.2), and $\mathsf{Q}_i = \{\mathsf{q}_1, \ldots, \mathsf{q}_i\}$ is the ith (unordered) prefix of Q. Then, the set $\bigcup_{i=1}^{n} \text{°}(\mathsf{Q}_i)$ is of size $O_{whp}(c_d \log^d n)$, and the staircase $\text{°}(\mathsf{P})$ is of size $O_{whp}(c_d \log^{d-1} n)$.*

Proof: By Lemma 4.5.2, the set $\bigcup_{i=1}^{n} \text{°}(\mathsf{Q}_i)$ is of size $O(c_d \log^d n)$, with high probability. By Lemma 4.5.3, the set $\text{°}(\mathsf{P})$ is of size $O(c_d \log^{d-1} n)$, with high probability. ■

Remark 4.5.5. In the proof of Lemma 4.5.3 whether a point is on the staircase (or not) only depends on the coordinate orderings of the points and not their actual values.

The basic recursive argument used in Lemma 4.5.3 was used by Clarkson [Cla04] to bound the expected number of k-sets for a random point set. Here, using Corollary 4.2.5 enables us to get a high-probability bound.

Note that the definition of the staircase can be made with respect to any corner of the hypercube (i.e., this corner would replace the origin in the definition dominance, point volume, the exponential grid, etc.). Taking the union over all 2^d such staircases gives us the subset of P on the orthogonal convex hull of P. Therefore Lemma 4.5.4 also bounds the number of input points on the orthogonal convex hull. As the vertices on the convex hull of P are a subset of the points in P on the orthogonal convex hull, the above also implies the same bound on the number of vertices on the convex hull.

4.5.2 Bounding the size of the candidate set

We can now readily bound the size of the candidate set for any point in the plane.

Lemma 4.5.6. *Let S be a set of n sites in the plane, where for each site s in S, a parametric point from a distribution over $[0,1]^d$ is sampled (as described in Section 4.2.2). Then, the candidate set has size $O_{whp}(\log^d n)$ simultaneously for all points in the plane.*

Proof: Consider the arrangement of bisectors of all pairs of points of S. This arrangement has complexity $O(n^4)$; inside each cell the candidate set is the same. Now for any point in a cell of the arrangement, Lemma 4.5.4 immediately gives us the stated bound, with high probability. Therefore picking a representative point from each cell in this arrangement and applying the union bound imply the claim. ■

4.6 The main result

We now use the bound on the complexity of the proxy diagram, as well as our knowledge of the relationship between the candidate set and the proxy set to bound the complexity of the candidate diagram.

Theorem 4.6.1. *Let S be a set of n sites in the plane, where for each site in S we sample an associated parametric point in $[0,1]^d$, as described in Section 4.2.2. Then, the expected complexity of the candidate diagram is $O\Big(n \log^{8d+5} n\Big)$. The expected space complexity of this candidate diagram is $O\Big(n \log^{9d+5} n\Big)$.*

Proof: Fix k to be sufficiently large such that $k = \Theta(\log^d n)$. By Lemma 4.4.7 the expected complexity of the proxy diagram is $O(k^4 n \log n)$. Triangulating each polygonal cell in the diagram does not increase its asymptotic complexity. Lemma 4.3.2 implies that, (simultaneously) for all the points in the plane, the proxy set has size $O(k \log n)$, with high probability. Now, Lemma 4.3.4 implies that, with high probability, the proxy set contains the candidate set for any point in the plane.

The resulting triangulation has $O(k^4 n \log n)$ faces, and inside each face all the sites that might appear in the candidate set are all present in the proxy set of this face. By Lemma 4.2.4, the complexity of an m-site candidate diagram is $O(m^4)$. Therefore the complexity of the candidate diagram per face is $O\big((k \log n)^4\big)$, with high probability (clipping the candidate diagram of these sites to the containing triangle does not increase the asymptotic complexity). Multiplying the number of faces, $O(k^4 n \log n)$, by the complexity of the arrangement within each face, $O\big((k \log n)^4\big)$, yields the desired result.

The bound on the space complexity follows readily from the bound on the size of the candidate set from Lemma 4.5.6. ∎

4.7 Backward analysis with high probability

Let P be a set of n elements. Recall that a ***property*** $\mathcal{P}$ of P is a function that maps any subset X of P to a subset $\mathcal{P}(\mathsf{X})$ of X. (See Section 4.2.3.) The following two lemmas, once proven, implies Corollary 4.2.5.

Lemma 4.7.1. *Let P be a set of n elements, and let $k_1, \ldots, k_n$ be n fixed non-negative integers depending only on P. Let $\mathcal{P}$ be a property of P satisfying the following condition: if $|\mathsf{X}| = i$ then $|\mathcal{P}(\mathsf{X})| = k_i$. Now, consider a uniform random permutation $\langle \mathsf{p}_1, \ldots, \mathsf{p}_n \rangle$ of P. For each i, denote $\mathsf{P}_i = \{\mathsf{p}_1, \ldots, \mathsf{p}_i\}$ and let X_i be an indicator variable of the event $\mathsf{p}_i \in \mathcal{P}(\mathsf{P}_i)$. Then the variables X_i are mutually independent.*

Proof: Let E_i denote the event that $\mathsf{p}_i \in \mathcal{P}(\mathsf{P}_i)$. It suffices to show that the events $E_1, \ldots, E_n$ are mutually independent. The insight is to think about the sampling process of creating the random permutation $\langle \mathsf{p}_1, \ldots, \mathsf{p}_n \rangle$ in a different way, and then the result readily follows. Imagine we randomly pick a permutation of elements in P, and set the last element to be p_n. Next, pick a random permutation of the remaining elements of $\mathsf{P} \setminus \{\mathsf{p}_n\}$ and set the last element to be p_{n-1}. Repeat this process until the whole permutation is generated. Observe that E_j is determined before E_i for any $j > i$.

Now, consider arbitrary indices $1 \leq i_1 < i_2 < \ldots < i_\psi \leq n$. Observe that by our thought experiment, when determining the i_1th value in the permutation, the suffix $\langle \mathsf{p}_{i_1+1}, \ldots, \mathsf{p}_n \rangle$ is fixed. Moreover, the property defined on the remaining set of elements marks k_{i_1} elements, and these elements are randomly permuted before determining the i_1th value. Therefore, for any fixed permutation $\langle \mathsf{p}_{i_1+1}, \ldots, \mathsf{p}_n \rangle$, we have $\mathbf{Pr}\big[E_{i_1} \mid \langle \mathsf{p}_{i_1+1}, \ldots, \mathsf{p}_n \rangle\big] = k_{i_1}/i_1$. This implies that conditioning on the union of any collection of permutations of the last $n - i_1$ elements also yields the same probability. In particular, $\mathbf{Pr}\big[E_{i_1}\big] = k_{i_1}/i_1$ (conditioning

on the union of all possible permutations of the last $n-i_1$ elements), and $\mathbf{Pr}\big[E_{i_1} \;\big|\; E_{i_2} \cap \ldots \cap E_{i_\psi}\big] = \mathbf{Pr}\big[E_{i_1}\big]$ (conditioning on $E_{i_2} \cap \ldots \cap E_{i_\psi}$, which is a union of some collection of permutations of the last $n-i_1$ elements). By induction,

$$\begin{aligned}\mathbf{Pr}\big[E_{i_1} \cap \ldots \cap E_{i_\psi}\big] &= \mathbf{Pr}\big[E_{i_1} \;\big|\; E_{i_2} \cap \ldots \cap E_{i_\psi}\big]\,\mathbf{Pr}\big[E_{i_2} \cap \ldots \cap E_{i_\psi}\big] \\ &= \mathbf{Pr}\big[E_{i_1}\big]\,\mathbf{Pr}\big[E_{i_2} \cap \ldots \cap E_{i_\psi}\big] = \prod_{j=1}^{\psi} \mathbf{Pr}\big[E_{i_j}\big]\,,\end{aligned}$$

which implies that the events are mutually independent. ■

Lemma 4.7.2. *Let P be a set of n elements, and $k \geq 1$ be a fixed integer depending only on P. Let $\mathcal{P}$ be a property of P. Now, consider a uniform random permutation $\langle \mathsf{p}_1, \ldots, \mathsf{p}_n \rangle$ of P. For each i, denote $\mathsf{P}_i = \{\mathsf{p}_1, \ldots, \mathsf{p}_i\}$ and let X_i be an indicator variable of the event $\mathsf{p}_i \in \mathcal{P}(\mathsf{P}_i)$. Then we have*

(A) If $|\mathcal{P}(\mathsf{X})| = k$ whenever $|\mathsf{X}| \geq k$, and $|\mathcal{P}(\mathsf{X})| = |\mathsf{X}|$ whenever $|\mathsf{X}| < k$, then for any $\gamma \geq 2e$, $\mathbf{Pr}\Big[\sum_{i=1}^{n} X_i > \gamma \cdot (2k \ln n)\Big] \leq n^{-\gamma k}$.

(B) The bound in (A) holds under a weaker condition: For all $\mathsf{X} \subseteq \mathsf{P}$ we have $|\mathcal{P}(\mathsf{X})| \leq k$.

(C) An even weaker condition suffices: For a random permutation $\langle \mathsf{p}_1, \ldots, \mathsf{p}_n \rangle$ of P, assume $|\mathcal{P}(\mathsf{P}_i)| \leq \Bbbk$ for all i, with probability $1 - n^{-c}$, where $\Bbbk = c' \cdot k$, c is an arbitrary constant, and $c' > 1$ is a constant that depends only on c. Then $\mathbf{Pr}\Big[\sum_{i=1}^{n} X_i > \gamma \cdot (2c'k \ln n)\Big] \leq n^{-\gamma k} + n^{-c}$ for any $\gamma \geq 2e$.

Proof: (A) Let E_i be the event that $\mathsf{p}_i \in \mathcal{P}(\mathsf{P}_i)$. By Lemma 4.7.1 the events $E_1, \ldots, E_n$ are mutually independent, and $\mathbf{Pr}\big[E_i\big] = |\mathcal{P}(\mathsf{P}_i)|\,/i = \min\big(k/i, 1\big)$. Thus, we have

$$\mu = \mathbf{E}\left[\sum_i X_i\right] \leq k + \sum_{i=k+1}^{n} \frac{k}{i} \leq k\Big(1 + \ln n + 1 - \ln k\Big) \leq k\Big(2 + \ln \frac{n}{k}\Big) \leq 2k \ln n.$$

For any constant $\delta \geq 2e$, by Chernoff's inequality, we have $\mathbf{Pr}\big[\sum_i X_i > \delta\mu\big] < 2^{-\delta\mu}$. Therefore by setting $\delta = \gamma(2k \ln n)/\mu$ (which is at least $2e$ by the assumption that $\gamma \geq 2e$), we have

$$\mathbf{Pr}\left[\sum_i X_i > \gamma(2k \ln n)\right] = \mathbf{Pr}\left[\sum_i X_i > \frac{\gamma(2k \ln n)}{\mu}\mu\right] < 2^{-\gamma(2k \ln n)} < n^{-\gamma k}.$$

(B) In order to extend the result using the weaker condition, we augment the given property $\mathcal{P}$ to a new property $\mathcal{P}'$ that holds for *exactly* k elements. So, fix an arbitrary ordering $\prec$ on the elements of P. Now given any set X with $|\mathsf{X}| \geq k$, if $|\mathcal{P}(\mathsf{X})| = k$ then let $\mathcal{P}'(\mathsf{X}) = \mathcal{P}(\mathsf{X})$; otherwise, add the $k - |\mathcal{P}(\mathsf{X})|$ smallest elements in $\mathsf{X} \setminus \mathcal{P}(\mathsf{X})$ according to $\prec$ to $\mathcal{P}(\mathsf{X})$, and let $\mathcal{P}'(\mathsf{X})$ be the resulting subset of size k. We also set $\mathcal{P}'(\mathsf{X}) = \mathsf{X}$ for all X with $|\mathsf{X}| < k$. The new property $\mathcal{P}'$ complies with the original condition. For any X, $\mathcal{P}(\mathsf{X}) \subseteq \mathcal{P}'(\mathsf{X})$, which implies that an upper bound on the probability that the ith element is in the property set $\mathcal{P}'$ is an upper bound on the corresponding probability for $\mathcal{P}$.

(C) We truncate the given property $\mathcal{P}$ if needed, so that it compiles with (B). Specifically, fix an arbitrary ordering $\prec$ on the elements of P. Given any set X, if $|\mathcal{P}(\mathsf{X})| \leq \Bbbk$ then $\mathcal{P}'(\mathsf{X}) = \mathcal{P}(\mathsf{X})$; otherwise, $|\mathcal{P}(\mathsf{X})| > \Bbbk$, and set $\mathcal{P}'(\mathsf{X})$ to be the first $\Bbbk$ of $\mathcal{P}(\mathsf{X})$ according to $\prec$. Clearly, the new property $\mathcal{P}'$ complies with the condition in (B). Let $\mathcal{E}$ denote the event $\mathcal{P}'(\mathsf{P}_i) = \mathcal{P}(\mathsf{P}_i)$, for all i. By assumption, we have $\mathbf{Pr}[\mathcal{E}] \geq 1 - n^{-c}$.

Similarly, let $\mathcal{F}$ be the event that $\sum_i X_i > \gamma(2\Bbbk \ln n)$. We now have that

$$\mathbf{Pr}\Big[\mathcal{F}\Big] \leq \mathbf{Pr}\big[\mathcal{F} \mid \mathcal{E}\big]\,\mathbf{Pr}\Big[\mathcal{E}\Big] + \mathbf{Pr}\Big[\overline{\mathcal{E}}\Big] < \left(1 - \frac{1}{n^c}\right) n^{-\gamma\Bbbk} + n^{-c} \leq n^{-\gamma k} + n^{-c}$$

for any $\gamma \geq 2e$. ∎

The result of Lemma 4.7.2 is known in the context of randomized incremental construction algorithms (see [BCKO08, Section 6.4]). However, the known proof is more convoluted—indeed, if the property $\mathcal{P}(\mathsf{X})$ has different sizes for different sets X, then it is no longer true that variables X_i in the proof of Lemma 4.7.2 are independent. Thus the padding idea in part (B) of the proof is crucial in making the result more widely applicable.

Example. To see the power of Lemma 4.7.2 we provide two easy applications—both results are of course known, and are included here to make it clearer in what settings Lemma 4.7.2 can be applied. The impatient reader is encouraged to skip this example.

(A) **QuickSort**: We conceptually can think about **QuickSort** as being a randomized incremental algorithm, building up a list of numbers in the order they are used as pivots. Consider the execution of **QuickSort** when sorting a set P of n numbers. Let $\langle \mathsf{p}_1, \ldots, \mathsf{p}_n \rangle$ be the random permutation of the numbers picked in sequence by **QuickSort**. Specifically, in the ith iteration, it randomly picks a number p_i that was not handled yet, pivots based on this number, and then recursively handles the subproblems. At the ith iteration, a set $\mathsf{P}_i = \{\mathsf{p}_1, \ldots, \mathsf{p}_i\}$ of pivots has already been chosen by the algorithm. Consider a specific element $\mathsf{x} \in \mathsf{P}$. For any subset $\mathsf{X} \subseteq \mathsf{P}$, let $\mathcal{P}(\mathsf{X})$ be the two numbers in X having x in between them in the original ordering of P and are closest to each other. In other words, $\mathcal{P}(\mathsf{X})$ contains the (at most) two elements that are the endpoints of the interval of $\mathbb{R} \setminus \mathsf{X}$ that contains x. Let X_i be the indicator variable of the event $\mathsf{p}_i \in \mathcal{P}(\mathsf{P}_i)$—that is, x got compared to the ith pivot when it was inserted. Clearly, the total number of comparisons x participates in is $\sum_i X_i$, and by Lemma 4.7.2 the number of such comparisons is $O(\log n)$, with high probability, implying that **QuickSort** takes $O(n \log n)$ time, with high probability.

(B) **Point-location queries in a history dag**: Consider a set of lines in the plane, and build their vertical decomposition using randomized incremental construction. Let $L_n = \langle \ell_1, \ldots, \ell_n \rangle$ be the permutation used by the randomized incremental construction. Given a query point p, the point-location time is the number of times the vertical trapezoid containing p changes in the vertical decomposition of $L_i = \langle \ell_1, \ldots, \ell_i \rangle$, as i increases. Thus, let X_i the indicator variable of the event that ℓ_i is one of the (at most) four lines defining the vertical trapezoid containing p the vertical decomposition of L_i. Again, Lemma 4.7.2 applies and implies that the query time is $O(\log n)$, with high probability. This result is well known, see [CMS93] and [BCKO08, Section 6.4], but our proof is arguably more direct and cleaner.

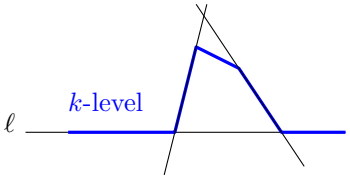
4.8 Basic facts

This section contains some basic facts required earlier in this chapter.

4.8.1 Arrangements of lines

Lemma 4.8.1. *Let L be set of lines in general position in the plane, and let ℓ be any line in L. Then at most $k+2$ edges from $\mathsf{E}_k(L)$, the k-level of the arrangement of L, can lie on ℓ.*

Proof: This lemma is well known, and its proof is included here for the sake of completeness.



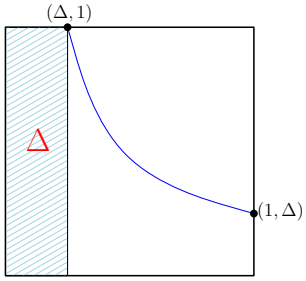
Perform a linear transformation such that ℓ is horizontal and the k-level is preserved. As we go from left to right along the now horizontal line ℓ (starting at infinity), we may leave and enter the k-level multiple times. However, every time we leave and then return to the k-level we must intersect a negative slope line. Specifically, both when we leave and return to the k-level, there must be an intersection with another line. If when leaving, this intersection is with a negative slope line then we are done, so assume it has positive slope. In this case the level on ℓ decreases as we leave the k-level, therefore when we return to the k-level, the point of return must be at an intersection with a negative slope line (since only negative slope intersections can increase the level), see figure on the right.

So after leaving and returning to the k-level $k+1$ times, there must be at least $k+1$ negative slope lines below, which implies that the remaining part of ℓ is on level strictly larger than k. ■

4.8.2 An integral calculation

Lemma 4.8.2. *Let $F_d(\Delta)$ be the total measure of the points $\mathsf{p} = (\mathsf{p}_1, \ldots, \mathsf{p}_d)$ in the hypercube $[0,1]^d$, such that $\mathrm{pv}(\mathsf{p}) = \mathsf{p}_1 \mathsf{p}_2 \cdots \mathsf{p}_d \leq \Delta$. That is, $F_d(\Delta)$ is the measure of all points in hypercube with point volume $\leq \Delta$. Then $F_d(\Delta) = \sum_{i=0}^{d-1} \frac{\Delta}{i!} \ln^i \frac{1}{\Delta}$.*

Proof: The claim follows by tedious but relatively standard calculations. As such, the proof is included for the sake of completeness.



The case for $d = 1$ is trivial. Consider the $d = 2$ case; here the points whose point volume equals Δ are defined by the curve $xy = \Delta$. This curve intersects the unit square at the point $(\Delta, 1)$. As $F_d(\Delta)$ is the total volume under this curve in the unit square we have that

$$F_2(\Delta) = \Delta + \int_{x=\Delta}^{1} \frac{\Delta}{x} \mathrm{d}x = \Delta + [\Delta \ln x]_{x=\Delta}^{1} = \Delta + \Delta \ln \frac{1}{\Delta}.$$

In general, we have

$$\frac{1}{(d-1)!} \int_{x=\Delta}^{1} \frac{\Delta}{x} \ln^{d-1} \frac{x}{\Delta} \mathrm{d}x = \frac{\Delta}{(d-1)!} \left[\frac{1}{d} \ln^d \frac{x}{\Delta} \right]_{x=\Delta}^{1} = \frac{\Delta}{d!} \ln^d \frac{1}{\Delta}.$$

Now assume inductively that

$$F_{d-1}(\Delta) = \sum_{i=0}^{d-2} \frac{1}{i!} \Delta \ln^i \frac{1}{\Delta},$$

then we have

$$F_d(\Delta) = \Delta + \int_{x_d=\Delta}^{1} F_{d-1}\left(\frac{\Delta}{x_d}\right)\mathrm{d}x_d = \Delta + \int_{x_d=\Delta}^{1}\left(\sum_{i=0}^{d-2}\frac{\Delta}{i!x_d}\ln^i\frac{x_d}{\Delta}\right)\mathrm{d}x_d$$
$$= \Delta + \sum_{i=0}^{d-2}\frac{1}{i!}\left(\int_{x_d=\Delta}^{1}\frac{\Delta}{x_d}\ln^i\frac{x_d}{\Delta}\mathrm{d}x_d\right) = \Delta + \sum_{i=1}^{d-1}\frac{\Delta}{i!}\ln^i\frac{1}{\Delta} = \sum_{i=0}^{d-1}\frac{\Delta}{i!}\ln^i\frac{1}{\Delta}.$$

■

References

[AAH+13] P. K. Agarwal, B. Aronov, S. Har-Peled, J. M. Phillips, K. Yi, and W. Zhang. Nearest neighbor searching under uncertainty II. In *Proc. 32nd ACM Sympos. Principles Database Syst.* (PODS), pages 115–126, 2013.

[ABC09] P. Afshani, J. Barbay, and T. Chan. Instance-optimal geometric algorithms. In *Proceedings of the Foundations of Computer Science (FOCS)*, pages 129–138, 2009.

[ABT08] B. Aronov, M. de Berg, and S. Thite. The complexity of bisectors and Voronoi diagrams on realistic terrains. In *Proc. 16th Annu. European Sympos. Algorithms* (ESA), pages 100–111, 2008.

[AH08] B. Aronov and S. Har-Peled. On approximating the depth and related problems. *SIAM J. Comput.*, 38(3):899–921, 2008.

[AHKS14] P. K. Agarwal, S. Har-Peled, H. Kaplan, and M. Sharir. Union of random minkowski sums and network vulnerability analysis. *Discrete Comput. Geom.*, 52(3):551–582, 2014.

[AHV04] P. K. Agarwal, S. Har-Peled, and K. R. Varadarajan. Approximating extent measures of points. *J. Assoc. Comput. Mach.*, 51(4):606–635, 2004.

[AHV05] P. K. Agarwal, S. Har-Peled, and K. Varadarajan. Geometric approximation via coresets. In J. E. Goodman, J. Pach, and E. Welzl, editors, *Combinatorial and Computational Geometry*, Math. Sci. Research Inst. Pub. Cambridge, New York, NY, USA, 2005.

[AKL13] F. Aurenhammer, R. Klein, and D.-T. Lee. *Voronoi Diagrams and Delaunay Triangulations.* World Scientific, 2013.

[AMS98] P. K. Agarwal, J. Matoušek, and O. Schwarzkopf. Computing many faces in arrangements of lines and segments. *SIAM J. Comput.*, 27(2):491–505, 1998.

[APF+10] G. Aggarwal, R. Panigrahy, T. Feder, D. Thomas, K. Kenthapadi, S. Khuller, and A. Zhu. Achieving anonymity via clustering. *ACM Trans. Algo.*, 6(3):49:1–49:19, 2010.

[AS92] F. Aurenhammer and O. Schwarzkopf. A simple on-line randomized incremental algorithm for computing higher order Voronoi diagrams. *Internat. J. Comput. Geom. Appl.*, pages 363–381, 1992.

[AST94] P. K. Agarwal, M. Sharir, and S. Toledo. Applications of parametric searching in geometric optimization. *J. Algorithms*, 17:292–318, 1994.

[BCKO08] M. de Berg, O. Cheong, M. van Kreveld, and M. H. Overmars. *Computational Geometry: Algorithms and Applications.* Springer-Verlag, Santa Clara, CA, USA, 3rd edition, 2008.

[BDHT05] Z.-D. Bai, L. Devroye, H.-K. Hwang, and T.-H. Tsai. Maxima in hypercubes. *Random Struct. Alg.*, 27(3):290–309, 2005.

[BKO+98] C. Bajaj, M. van Kreveld, R. W. van Oostrum, V. Pascucci, and D. R. Schikore. Contour trees and small seed sets for isosurface traversal. Technical Report UU-CS-1998-25, Department of Information and Computing Sciences, Utrecht University, 1998.

[BKS01] S. Börzsönyi, D. Kossmann, and K. Stocker. The skyline operator. In *ICDE*, pages 421–430, 2001.

[BM12] J.-D. Boissonnat and C. Maria. The simplex tree: An efficient data structure for general simplicial complexes. In *Proceedings of the European Symposium on Algorithms (ESA)*, pages 731–742, 2012.

[BR63] R. Boyell and H. Ruston. Hybrid techniques for real-time radar simulation. In *Proceedings of Fall Joint Computer Conference*, pages 445–458, 1963.

[BR10a] I. Bárány and M. Reitzner. On the variance of random polytopes. *Adv. Math.*, 225(4):1986–2001, 2010.

[BR10b] I. Bárány and M. Reitzner. Poisson polytopes. *Annals. Prob.*, 38(4):1507–1531, 2010.

[BS02] S. Bespamyatnikh and M. Segal. Fast algorithms for approximating distances. *Algorithmica*, 33(2):263–269, 2002.

[BWH+11] K. Beketayev, G. Weber, M. Haranczyk, P.-T. Bremer, M. Hlawitschka, and B. Hamann. Visualization of topology of transformation pathways in complex chemical systems. In *Computer Graphics Forum (EuroVis 2011)*, pages 663–672, 2011.

[BWP+10] P.-T. Bremer, G. Weber, V. Pascucci, M. Day, and J. Bell. Analyzing and tracking burning structures in lean premixed hydrogen flames. *IEEE Transactions on Visualization and Computer Graphics*, 16(2):248–260, 2010.

[BWT+11] P.-T. Bremer, G. Weber, J. Tierny, V. Pascucci, M. Day, and J. Bell. Analyzing and tracking burning structures in lean premixed hydrogen flames. *IEEE Transactions on Visualization and Computer Graphics*, 17(9):1307–1325, 2011.

[Car04] H. Carr. *Topological Manipulation of Isosurfaces.* PhD thesis, University of British Columbia, 2004.

[CF90] B. Chazelle and J. Friedman. A deterministic view of random sampling and its use in geometry. *Combinatorica*, 10(3):229–249, 1990.

[Cha08] T. M. Chan. Well-separated pair decomposition in linear time? *Inform. Process. Lett.*, 107(5):138–141, 2008.

[CHR15] H.-C. Chang, S. Har-Peled, and B. Raichel. From proximity to utility: A Voronoi partition of Pareto optima. In *Proc. 31st Annu. Sympos. Comput. Geom.* (SoCG), 2015. To appear.

[CK95] P. B. Callahan and S. R. Kosaraju. A decomposition of multidimensional point sets with applications to k-nearest-neighbors and n-body potential fields. *J. Assoc. Comput. Mach.*, 42:67–90, 1995.

[Cla83] K. L. Clarkson. Fast algorithms for the all nearest neighbors problem. In *Proc. 24th Annu. IEEE Sympos. Found. Comput. Sci.* (FOCS), pages 226–232, Washington, DC, USA, 1983. IEEE Computer Society.

[Cla88] K. L. Clarkson. Applications of random sampling in computational geometry, II. In *Proc. 4th Annu. Sympos. Comput. Geom.* (SoCG), pages 1–11, New York, NY, USA, 1988. ACM.

[Cla04] K. L. Clarkson. On the expected number of k-sets of coordinate-wise independent points. manuscript, 2004.

[CLLR05] Y. Chiang, T. Lenz, X. Lu, and G. Rote. Simple and optimal output-sensitive construction of contour trees using monotone paths. *Computational Geometry: Theory and Applications*, 30(2):165–195, 2005.

[CMEH+03] K. Cole-McLaughlin, H. Edelsbrunner, J. Harer, V. Natarajan, and V. Pascucci. Loops in reeb graphs of 2-manifolds. In *Proceedings of the Symposium on Computational Geometry (SoCG)*, pages 344–350, 2003.

[CMS93] K. L. Clarkson, K. Mehlhorn, and R. Seidel. Four results on randomized incremental constructions. *Comput. Geom. Theory Appl.*, 3(4):185–212, 1993.

[CS89] K. L. Clarkson and P. W. Shor. Applications of random sampling in computational geometry, II. *Discrete Comput. Geom.*, 4:387–421, 1989.

[CSA00] H. Carr, J. Snoeyink, and U. Axen. Computing contour trees in all dimensions. In *Proceedings of the Symposium on Discrete Algorithms*, pages 918–926, 2000.

[DHR12] A. Driemel, S. Har-Peled, and B. Raichel. On the expected complexity of Voronoi diagrams on terrains. In *Proc. 28th Annu. Sympos. Comput. Geom.* (SoCG), pages 101–110, 2012.

[DHW12] A. Driemel, S. Har-Peled, and C. Wenk. Approximating the Fréchet distance for realistic curves in near linear time. *Discrete Comput. Geom.*, 48:94–127, 2012.

[DN09] H. Doraiswamy and V. Natarajan. Efficient algorithms for computing reeb graphs. *Computational Geometry: Theory and Applications*, 42:606–616, 2009.

[DN13] H. Doraiswamy and V. Natarajan. Computing reeb graphs as a union of contour trees. *IEEE TRANSACTIONS ON VISUALIZATION AND COMPUTER Graphics*, 19(2):249–262, 2013.

[EET93] H. ElGindy, H. Everett, and G. Toussaint. Slicing an ear using prune-and-search. *Pattern Recogn. Lett.*, 14(9):719–722, 1993.

[ERH12] A. Ene, B. Raichel, and S. Har-Peled. Fast clustering with lower bounds: No customer too far, no shop too small. In submission. `http://sarielhp.org/papers/12/lbc/`, 2012.

[Eri95] J. Erickson. On the relative complexities of some geometric problems. In *Proc. 7th Canad. Conf. Comput. Geom.* (CCCG), pages 85–90, Ottawa, Canada, 1995. Carleton University.

[Fel08] A. Feldman. Welfare economics. In S. Durlauf and L. Blume, editors, *The New Palgrave Dictionary of Economics*. Palgrave Macmillan, 2008.

[FG88] T. Feder and D. H. Greene. Optimal algorithms for approximate clustering. In *Proc. 20th Annu. ACM Sympos. Theory Comput.* (STOC), pages 434–444, 1988.

[FJ84] G. N. Frederickson and D. B. Johnson. Generalized selection and ranking: Sorted matrices. *SIAM J. Comput.*, 13:14–30, 1984.

[FM67] H. Freeman and S. Morse. On searching a contour map for a given terrain elevation profile. *Journal of the Franklin Institute*, 284(1):1–25, 1967.

[Gon85] T. Gonzalez. Clustering to minimize the maximum intercluster distance. *Theoret. Comput. Sci.*, 38:293–306, 1985.

[GRSS95] M. Golin, R. Raman, C. Schwarz, and M. Smid. Simple randomized algorithms for closest pair problems. *Nordic J. Comput.*, 2:3–27, 1995.

[GSG07] P. Godfrey, R. Shipley, and J. Gryz. Algorithms and analyses for maximal vector computation. *VLDB J.*, 16(1):5–28, 2007.

[Har01] S. Har-Peled. Clustering motion. In *Proc. 42nd Annu. IEEE Sympos. Found. Comput. Sci.* (FOCS), pages 84–93, Washington, DC, USA, 2001. IEEE Computer Society.

[Har04a] S. Har-Peled. Clustering motion. *Discrete Comput. Geom.*, 31(4):545–565, 2004.

[Har04b] S. Har-Peled. No coreset, no cry. In *Proc. 24th Conf. Found. Soft. Tech. Theoret. Comput. Sci.* (FSTTCS), pages 324–335, Berlin, Heidelberg, 2004. Springer-Verlag.

[Har11] S. Har-Peled. *Geometric Approximation Algorithms*, volume 173 of *Mathematical Surveys and Monographs*. Amer. Math. Soc., Boston, MA, USA, 2011.

[HE10] J. Harer and H. Edelsbrunner. *Computational Topology*. AMS, 2010.

[HK07] S. Har-Peled and A. Kushal. Smaller coresets for k-median and k-means clustering. *Discrete Comput. Geom.*, 37(1):3–19, 2007.

[HM04] S. Har-Peled and S. Mazumdar. Coresets for k-means and k-median clustering and their applications. In *Proc. 36th Annu. ACM Sympos. Theory Comput.* (STOC), pages 291–300, New York, NY, USA, 2004. ACM.

[HM05] S. Har-Peled and S. Mazumdar. Fast algorithms for computing the smallest k-enclosing disc. *Algorithmica*, 41(3):147–157, 2005.

[HM06] S. Har-Peled and M. Mendel. Fast construction of nets in low dimensional metrics, and their applications. *SIAM J. Comput.*, 35(5):1148–1184, 2006.

[HR11] S. Har-Peled and B. Raichel. The Fréchet distance revisited and extended. In *Proc. 27th Annu. Sympos. Comput. Geom.* (SoCG), pages 448–457, New York, NY, USA, 2011. ACM. http://sarielhp.org/papers/10/frechet3d/.

[HR14] S. Har-Peled and B. Raichel. On the expected complexity of randomly weighted Voronoi diagrams. In *Proc. 30th Annu. Sympos. Comput. Geom.* (SoCG), pages 232–241, 2014.

[HR15a] S. Har-Peled and B. Raichel. Net and prune: A linear time algorithm for euclidean distance problems. *J. Assoc. Comput. Mach.*, 2015. Accepted.

[HR15b] S. Har-Peled and B. Raichel. On the complexity of randomly weighted Voronoi diagrams. *Discrete Comput. Geom.*, 53(3):547–568, 2015.

[HTC13] H.-K. Hwang, T.-H. Tsai, and W.-M. Chen. Threshold phenomena in k-dominant skylines of random samples. *SIAM J. Comput.*, 42(2):405–441, 2013.

[HW87] D. Haussler and E. Welzl. ε-nets and simplex range queries. *Discrete Comput. Geom.*, 2:127–151, 1987.

[HWW10] W. Harvey, Y. Wang, and R. Wenger. A randomized o(m log m) time algorithm for computing reeb graph of arbitrary simplicial complexes. In *Proceedings of the Symposium on Computational Geometry (SoCG)*, pages 267–276, 2010.

[KL04] R. Krauthgamer and J. R. Lee. Navigating nets: simple algorithms for proximity search. In *Proc. 15th ACM-SIAM Sympos. Discrete Algs.* (SODA), pages 798–807, Philadelphia, PA, USA, 2004. Society for Industrial and Applied Mathematics.

[KLN09] R. Klein, E. Langetepe, and Z. Nilforoushan. Abstract Voronoi diagrams revisited. *Comput. Geom.*, 42(9):885–902, 2009.

[KLP75] H. Kung, F. Luccio, and F. Preparata. On finding the maxima of a set of vectors. *J. Assoc. Comput. Mach.*, 22(4):469–476, 1975.

[KM95] S. Khuller and Y. Matias. A simple randomized sieve algorithm for the closest-pair problem. *Inform. Comput.*, 118:34–37, 1995.

[LBM+06] D. Laney, P.-T. Bremer, A. Macarenhas, P. Miller, and V. Pascucci. Understanding the structure of the turbulent mixing layer in hydrodynamic instabilities. *IEEE Transactions on Visualization and Computer Graphics*, 12(6):1053–1060, 2006.

[LMS94] C.-Y. Lo, J. Matoušek, and W. L. Steiger. Algorithms for ham-sandwich cuts. *Discrete Comput. Geom.*, 11:433–452, 1994.

[Meg83] N. Megiddo. Applying parallel computation algorithms in the design of serial algorithms. *J. Assoc. Comput. Mach.*, 30(4):852–865, 1983.

[Meg84] N. Megiddo. Linear programming in linear time when the dimension is fixed. *J. Assoc. Comput. Mach.*, 31:114–127, 1984.

[MGB+11] A. Mascarenhas, R. Grout, P.-T. Bremer, V. Pascucci, E. Hawkes, and J. Chen. Topological feature extraction for comparison of length scales in terascale combustion simulation data. In *Topological Methods in Data Analysis and Visualization: Theory, Algorithms, and Applications*, pages 229–240, 2011.

[MR95] R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge University Press, Cambridge, UK, 1995.

[MSW96] J. Matoušek, M. Sharir, and E. Welzl. A subexponential bound for linear programming. *Algorithmica*, 16:498–516, 1996.

[Mul94] K. Mulmuley. *Computational Geometry: An Introduction Through Randomized Algorithms*. Prentice Hall, Englewood Cliffs, NJ, 1994.

[OSW84] T. Ottmann, E. Soisalon-Soininen, and D. Wood. On the definition and computation of rectlinear convex hulls. *Inf. Sci.*, 33(3):157–171, 1984.

[Par12] S. Parsa. A deterministic o(m log m) time algorithm for the reeb graph. In *Proceedings of the Symposium on Computational Geometry (SoCG)*, pages 269–276, 2012.

[PCM02] V. Pascucci and K. Cole-McLaughlin. Efficient computation of the topology of level set. In *IEEE Visualization*, pages 187–194, 2002.

[PSBM07] V. Pascucci, G. Scorzelli, P.-T. Bremer, and A. Mascarenhas. Robust on-line computation of reeb graphs: simplicity and speed. *ACM Transactions on Graphics*, 26(58), 2007.

[Rab76] M. O. Rabin. Probabilistic algorithms. In J. F. Traub, editor, *Algorithms and Complexity: New Directions and Recent Results*, pages 21–39. Academic Press, Orlando, FL, USA, 1976.

[RM00] J. B. T. M. Roerdink and A. Meijster. The watershed transform: definitions, algorithms, and parallelization strategies. *Fundamenta Informaticae*, 41:187–228, 2000.

[RS14] B. Raichel and C. Seshadhri. Avoiding the global sort: A faster contour tree algorithm. *CoRR*, abs/1411.2689, 2014.

[SA95] M. Sharir and P. K. Agarwal. *Davenport-Schinzel Sequences and Their Geometric Applications*. Cambridge University Press, New York, 1995.

[Sal97] J. Salowe. Parametric search. In J. E. Goodman and J. O'Rourke, editors, *Handbook of Discrete and Computational Geometry*, chapter 37, pages 683–698. CRC Press LLC, Boca Raton, FL, 1997.

[Sch79] A. Schönhage. On the power of random access machines. In *Proc. 6th Internat. Colloq. Automata Lang. Prog.*, volume 71 of *Lecture Notes Comput. Sci.*, pages 520–529, London, UK, 1979. Springer-Verlag.

[Sei93] R. Seidel. Backwards analysis of randomized geometric algorithms. In J. Pach, editor, *New Trends in Discrete and Computational Geometry*, volume 10 of *Algorithms and Combinatorics*, pages 37–68. Springer-Verlag, 1993.

[Sha03] M. Sharir. The Clarkson-Shor technique revisited and extended. *Comb., Prob. & Comput.*, 12(2):191–201, 2003.

[SK91] Y. Shinagawa and T. Kunii. Constructing a reeb graph automatically from cross sections. *IEEE Comput. Graphics Appl.*, 11(6):44–51, 1991.

[Smi00] M. Smid. Closest-point problems in computational geometry. In J.-R. Sack and J. Urrutia, editors, *Handbook of Computational Geometry*, pages 877–935. Elsevier, Amsterdam, The Netherlands, 2000.

[ST83] D. Sleator and R. Tarjan. A data structure for dynamic trees. *Journal of Computing and System Sciences*, 26(3):362–391, 1983.

[SW92] M. Sharir and E. Welzl. A combinatorial bound for linear programming and related problems. In *Proc. 9th Sympos. Theoret. Aspects Comput. Sci.*, volume 577 of *Lect. Notes in Comp. Sci.*, pages 569–579, London, UK, 1992. Springer-Verlag.

[SW93] R. Schneider and J. A. Wieacker. Integral geometry. In P. M. Gruber and J. M. Wills, editors, *Handbook of Convex Geometry*, volume B, chapter 5.1, pages 1349–1390. North-Holland, 1993.

[TGSP09] J. Tierny, A. Gyulassy, E. Simon, and V. Pascucci. Loop surgery for volumetric meshes: Reeb graphs reduced to contour trees. *IEEE Trans. on Visualization and Computer Graphics*, 15(6):1177–1184, 2009.

[TV98] S. Tarasov and M. Vyalyi. Construction of contour trees in 3d in $O(n \log n)$ steps. In *Proceedings of the Symposium on Computational Geometry (SoCG)*, pages 68–75, 1998.

[vKvOB+97] M. van Kreveld, R. van Oostrum, C. Bajaj, V. Pascucci, and D. Schikore. Contour trees and small seed sets for isosurface traversal. In *Proceedings of the Symposium on Computational Geometry (SoCG)*, pages 212–220, 1997.

[vOV04] R. van Oostrum and R. C. Veltkamp. Parametric search made practical. *Comput. Geom. Theory Appl.*, 28(2-3):75–88, 2004.

[Vui78] J. Vuillemin. A data structure for manipulating priority queues. *Communications of the ACM*, 21:309–314, 1978.

[WW93] W. Weil and J. A. Wieacker. Stochastic geometry. In P. M. Gruber and J. M. Wills, editors, *Handbook of Convex Geometry*, volume B, chapter 5.2, pages 1393–1438. North-Holland, 1993.