Chapter 1

Introduction

The overwhelming majority of research in computational geometry is devoted to the design and analysis of algorithms and data structures, in an effort to find the fastest possible solutions to geometric problems. This thesis, however, is concerned with lower bounds on the complexity of these problems. Here, the goal is to describe inherent limits on the ability of a model of computation to solve a geometric problem, or in other words, to define the phrase “fastest possible solution”.

This thesis describes new lower bounds on the complexity of several fundamental decision problems that arise in computational geometry. For example: Given a set of points in the plane, are any three colinear? Given a set of points and lines in the plane, does any point lie on any line? These and similar questions arise as subproblems or special cases of a large number of more complicated geometric problems, including point location, range searching, motion planning, collision detection, ray shooting, visibility, and hidden surface removal [86, 75].

In the last several years, computational geometers have developed powerful techniques for solving these problems efficiently, at least in theory, and there is a common belief that the best known algorithms for these problems are optimal or very close to optimal. Unfortunately, there are still large gaps between the running times of these algorithms and the best known lower bounds. Currently available methods for deriving lower bounds in general models of computation (algebraic decision trees, for example) are simply not powerful enough to give very good results.

The approach taken in this thesis, therefore, is to develop new lower bounds in models of computation that are more specialized to each problem. For each problem, the model we consider is powerful enough to describe all known algorithms for solving it. Thus, the lower bounds we develop formally demonstrate the limitations of solving the problem using currently available techniques.

Results in this area are important for two reasons. A good lower bound for a problem, in a sufficiently general model of computation, indicates that an efficient solution to the problem is essentially impossible. In such a situation, research is more profitably aimed at deriving approximate solutions, deriving algorithms that are efficient (or correct) only with high probability (either with respect to some input distribution or internal randomization), or considering useful special cases.

Lower bounds in specialized models of computation are also useful, provided those models accurately describe all known algorithms. Such results direct algorithms researchers to consider fundamentally new techniques for solving geometric problems, in an effort to avoid the limitations of current approaches. Moreover, by describing exactly why current approaches fail, these results ease the discovery of new approaches.

1.1 Old Results

Before describing our specific new results, we begin with a whirlwind tour of previously known lower bounds in computational geometry. Along the way, we will point out inherent limitations of the techniques used to derive these bounds.

In order to say anything meaningful about algorithmic complexity, we must first agree on what an algorithm is. All lower bounds are expressed in terms of a *model of computation*, a set of assumptions about the operations algorithms are allowed to perform and the how much "time" each of these operations costs. (Actually, all upper bounds are expressed in terms of a model of computation, too, but the model is rarely described explicitly.) For purposes of proving lower bounds, we almost always ignore all but a few simple operations such as branches, input/output, assignments, pointer traversals, or arithmetic, and then restrict further which of these operations we allow.

1.1.1 Output Size

For many geometric problems, the only available way to prove tight lower bounds is by bounding the combinatorial complexity of the output. For example, any algorithm that constructs an arrangement of n hyperplanes in $\mathbb{R}^d$ must take time $\Omega(n^d)$[1] in the worst case, because the arrangement can have that many cells [62]. Similarly, any algorithm that constructs the convex hull of a set of n points in $\mathbb{R}^d$ must take time $\Omega(n^{\lfloor d/2 \rfloor})$ in the worst case, because there are polytopes with that many facets [87].

This approach is clearly worthless if the output size is always small, or if we want to prove output-sensitive lower bounds of the form $\Theta(f(n)+r)$, where r is the output size. For almost all the problems we consider in this thesis, the output size is a single bit.

1.1.2 Decision Trees

Decision trees are one of the simplest and most widely studied models of computation. A k-*ary decision tree* is a rooted directed tree in which every internal node has k children. If the degree k is unspecified, we take it to be a small constant; in practice k is almost always either 2 or 3. Associated with each internal node is a question about the input, or *query*, with k possible answers. Each answer is associated with a unique outgoing edge. Each leaf is labeled with an output value. To compute with such a tree, we start at the root and proceed down to a leaf. At each internal node, the answer to its associated query tells us which child to go to next. The running time of the algorithm is the number of queries asked, which in the worst case is the depth of the tree. Memory management, data movement, arithmetic, and other aspects of real-world algorithms are simply ignored. Typically, a decision tree is designed only to deal with inputs of a particular size. Thus, an algorithm in this model is represented by a family of decision trees, one for each possible input size.

One of the easiest techniques for proving lower bounds on the depth of decision trees is based on *information theory*: if a problem has N possible outputs, then any decision tree that solves it must have at least N leaves, and thus its depth must be at least $\lceil \log_k N \rceil$. For example, finding an unknown integer between 1 and 1 000 000 using only yes-no questions requires at least $\lceil \log_2 1\,000\,000 \rceil = 20$ questions. Since n items can be ordered in $n!$ different ways, any k-ary decision tree that sorts a list of n items must have depth $\lceil \log_k(n!) \rceil = \Omega(n \log n)$.

There are many problems for which the information-theoretic bound is far too weak to be useful. For example, consider the simple problem of choosing the largest of n

[1] We assume the reader is familiar with the standard asymptotic notations $o(\cdot), O(\cdot), \Theta(\cdot), \Omega(\cdot)$, and $\omega(\cdot)$. Otherwise, see [52, 103].

input items. Any reasonable algorithm obviously requires linear time[2], but since there are only n possible outputs, the information-theoretic lower bound is only $\Omega(\log n)$. In fact, a matching $O(\log n)$ upper bound can be obtained by a decision tree that uses queries like "Is the largest item in the first half of the list?"

If we restrict the questions the algorithm can ask, we can often improve the information-theoretic lower bound by using an *adversary argument*. The argument works as follows. Instead of choosing a single input in advance and letting the algorithm ask questions about it, an all-powerful malicious adversary *pretends* to choose an input, and answers questions in whatever way will make the algorithm do the most work. If the algorithm doesn't ask enough queries, then there will be several different inputs, each consistent with the adversary's answers, that should result in different outputs. Whatever the algorithm outputs, the adversary can "reveal" an input that is consistent with all of its answers, but inconsistent with the algorithm's output. The adversary approach hinges critically on the fact that a decision tree only has access to its input through its queries; the algorithm cannot distinguish the adversary from an honest user that chooses an input in advance.

Adversary algorithms have proven particularly useful in proving lower bounds on the depth of *comparison trees*. In a comparison tree, the input is a set of items from some totally ordered domain (typically $\mathbb{Z}$ or $\mathbb{R}$), and every query is a comparison between two input values. The following simple adversary argument implies that any comparison tree that chooses the largest of n input values $x_1, x_2, \ldots, x_n$ must have depth at least $n-1$. The adversary initially presents an arbitrary list of distinct values, say $x_i = i$ for all i. If an algorithm declares that x_n is the largest input value after fewer than $n-1$ comparisons, there must be at least one other input value $x_i \neq x_n$ that is bigger than anything the algorithm compared it to. The adversary can change the value of x_i to $n+1$, and then "reveal" its modified input, proving the algorithm wrong. Since the algorithm cannot distinguish between the original input and the modified input, it cannot possibly give the correct result for both.

Adversary arguments have also been used to prove lower bounds on the number of comparisons required to choose the largest and smallest elements in a list, the second-largest element [102, 104] (see also [60]), the median element [21, 122], or the kth largest element for arbitrary values of k [122, 99]. The minimum number of comparisons required

[2] unless we allow a small probability of error [148]

to solve the last two problems is still not known; the simplest open case is $k = 3$. Several more references can be found in [17].

We can generalize comparison trees by allowing more complicated query expressions. A d*th order algebraic decision tree* [125, 126, 138] is a ternary decision tree whose input is a vector $(x_1, x_2, \dots, x_n)$ of real numbers, and each of whose queries asks for the sign of a multivariate *query polynomial* $f(x_1, x_2, \dots, x_d)$ of degree at most d. For example, in a comparison tree, every query is of the form $x_i - x_j$. Typically, the parameter d is omitted, and assumed to be a fixed constant. An important special case of algebraic decision trees are *linear decision trees*, in which every query polynomial is linear (*i.e.*, $d = 1$) [58, 117].

The *element uniqueness problem* asks, given a list of n numbers, whether any two are equal. It seems "obvious" that any algorithm that solves this problem must sort the input values[3], and so the decision tree complexity "ought" to be $\Omega(n \log n)$. Unfortunately, since there are only two possible outputs, YES and NO, the information-theoretic bound is trivial. A simple adversary argument establishes a lower bound of $n - 1$ in the comparison tree model, but that is hardly satisfactory.

Dobkin and Lipton [58] proved a lower bound of $\Omega(n \log n)$ for the element uniqueness problem in the linear decision tree model, and therefore also in the comparison tree model, as follows. Fix a linear decision tree, and for each leaf ℓ, let X_ℓ be the set of input vectors for which computation reaches ℓ. X_ℓ is the intersection of several hyperplanes and linear halfspaces, and is therefore a convex polytope; in particular, X_ℓ is connected. Let $W \subset \mathbb{R}^n$ be the set of input vectors whose elements are distinct. In order for the decision tree to be correct, W must equal the union of X_ℓ over all leaves ℓ whose output label is YES. Clearly W has $n!$ disjoint connected components. It follows that any linear decision tree that solves the element uniqueness problem must have at least $n!$ YES leaves. Otherwise, there must be some leaf ℓ such that X_ℓ intersects more than one component of W and therefore intersects the complement of W, a contradiction. Any ternary tree with $n!$ leaves has depth at least $\lceil \log_3(n!) \rceil = \Omega(n \log n)$. More generally, any linear decision tree that solves the *set membership problem* for a set W with $\#W$ connected components must have depth at least $\lceil \log_3(\#W) \rceil$.

Following ideas of Yao [154], this argument was later generalized to higher-order

[3] Like many "obvious" statements, this is actually false! The discriminant $\prod_{i<j}(x_i - x_j)$ can be directly computed using $O(n \log n)$ multiplications (and $O(n \log^2 n)$ additions) without sorting the inputs [15, 141]. The input elements x_i are distinct if and only if this expression is not equal to zero, but the expression gives us (almost) no information about the sorted order of the inputs.

algebraic decision trees by Steele and Yao [138] and Ben-Or [16]. The generalization hinges on the following theorem of algebraic geometry independently proven by Petrovskiĭ and Oleĭnik [120, 121], Thom [147], and Milnor [115]. (See also [13, 14, 152].) A *semialgebraic* set is the set of points satisfying a finite number of polynomial equations and inequalities.

Theorem 1.1 (Petrovskiĭ/Oleĭnik/Thom/Milnor). *Let V be a semialgebraic set in $\mathbb{R}^n$, defined by t polynomial inequalities of maximum degree d. The sum of the Betti numbers of V is $(td)^{O(n)}$; in particular, V has $(td)^{O(n)}$ connected components.*

This theorem implies that, if t is the depth of a dth order algebraic decision tree that solves the set membership problem for W, then W has at most $3^t(td)^{O(n)}$ connected components. It follows immediately that the depth must be $\Omega(\log \#W - n \log d)$; in particular, the complexity of the element uniqueness problem is $\Omega(n \log n)$. Ben-Or [16] strengthened the argument further, deriving a similar lower bound in the stronger *algebraic computation tree* model.

More recent techniques imply lower bounds based on different complexity measures of the set W, such as its higher order Betti numbers [158], its Euler characteristic [19, 159], or the number of its lower-dimensional faces [118, 157, 94]. Lower bounds can also be derived by considering the complexity of the complement of W, the interior of W, the closure of W, or the intersection of W with another semi-algebraic subset of $\mathbb{R}^n$. For further generalizations, see [156, 93]. All of these techniques are essentially information-theoretic; in every case, the implied lower bound is the logarithm of the complexity of W.

A large class of geometric problems can be formalized as asking whether a point lies in a (semi-)algebraic set W defined by a polynomial number of constant-degree polynomial (in-)equalities. Fortunately, this is precisely the framework in which these lower bound arguments apply. As a consequence, it is quite easy to derive $\Omega(n \log n)$ lower bounds for many of these problems. Unfortunately, this is the best we can do. The Petrovskiĭ/Oleĭnik/Thom/Milnor theorem and its generalizations [13, 14, 152] imply that the complexity of W, in any reasonable sense of the word "complexity", is at most $n^{O(n)}$. Thus, for these problems, *no known lower bound technique can imply lower bounds bigger than $\Omega(n \log n)$ in the algebraic decision tree model.* An $\omega(n \log n)$ lower bound for *any* natural problem solvable in polynomial time would be a major breakthrough. Quadratic lower bounds are known for a few NP-complete problems [59, 16, 93], but again, this is the best lower bound we can prove, since for these problems the number of defining inequalities is only singly-exponential.

1.1.3 Semigroup Arithmetic

The *semigroup arithmetic model* was introduced by Fredman [82, 81] and refined by Yao [155] and Chazelle [34] to study the complexity of range searching data structures. In this model, each point in the input is given a value from an abelian semigroup satisfying a mild technical condition[4]. These points are preprocessed into a data structure, so that the sum of the weights of the points contained in an arbitrary query range can be computed quickly. The data structure consists of a collection of partial sums, called *generators*, which are added together to produce the answer to each range query. The size of the data structure is the total number of generators. The time required to answer a range query is the minimum number of generators whose sum is the correct answer. The only computational activity considered in this model is the addition of semigroup values; branches, pointer traversals, memory allocation, and other aspects of real-world range searching algorithms are ignored.

Algorithms in the semigroup model can exploit special properties of the semigroup, but they are *not* allowed to exploit the particular weights on the points; the same algorithm must work for *any* assignment of weights to the points. In effect, answers must be computed symbolically. Subtraction of semigroup values is also disallowed, even if the semigroup is actually a group, as is frequently the case.

The semigroup model was originally introduced by Fredman [81, 82], who derived lower bounds for dynamic orthogonal and halfplane range searching data structures, which must support the insertion and deletion of points as well as range queries. The model was first applied to static range searching problems by Yao [155], who proved lower bounds for orthogonal range searching in two dimensions. Vaidya [150] proved lower bounds for orthogonal range searching in higher dimensions, which were later improved by Chazelle [34]. Chazelle also derived lower bounds for simplex range searching [33], and with Brönnimann and Pach, halfspace range searching [23]. All of these online lower bounds give tradeoffs between the space required by the data structure and the resulting query time. The model

[4] Specifically, the semigroup must be *faithful*: any two identically equal linear forms must involve exactly the same set of variables, although not necessarily with the same coefficients. For example, $(\mathbb{Z}, +)$, $(\{0,1\}, \vee)$, $(\mathbb{R}, \max)$, and $(2^P, \cup)$ (for any nonempty set P) are faithful semigroups, but $(\{0\}, +)$ and $\mathbb{Z}/2\mathbb{Z}$ are not [155].

was later generalized to deal with offline problems, where the ranges are all known in advance. Chazelle and Rosenberg [42, 41] derived lower bounds for computing partial sums in multi-dimensional arrays (a special case of orthogonal range searching where the points lie on a lattice), and Chazelle [38] proved lower bounds for both orthogonal and simplex range searching. With the exception of halfspace range searching [23], all of these lower bounds are optimal, up to polylogarithmic or n^{ε} factors, in the semigroup model. For further details, we refer the reader to an excellent survey by Matoušek [112].

Every set of points and set of ranges defines a bipartite incidence graph, where the presence of an edge denotes that a point lies in a range. A key step in the proofs of many semigroup lower bounds is the construction of a set of points and ranges, such that (some subgraph of) the incidence graph has several edges but no large complete bipartite subgraphs. For example, Fredman's lower bounds for dynamic halfplane range searching [82] rely on a construction of Erdős of n points and n lines in the plane with $\Omega(n^{4/3})$ point-line incidences; since any pair of lines intersects in at most one point, the incidence graph for this point-line configuration has no $K_{2,2}$. We will use a similar technique in the second part of this thesis.

Unfortunately, the semigroup arithmetic model is inappropriate for studying the complexity of range *emptiness* problems, where we just want to know whether any points lie inside a given query range. If the range is empty, then the algorithm will perform no additions; conversely, if we perform even a single addition, the range must not be empty. The problem is that algorithms in this model are not allowed to "know" the weights assigned to the points. All of the previous lower bounds hold when weights are taken from the faithful semigroup $(\{0, 1\}, \vee)$, but *not* under the assumption that every point is given weight 1. This may be somewhat counterintuitive, since we can "obviously" remove the points with weight zero before doing anything else. However, in many range searching applications, weights are often not assigned in advance, but are determined implicitly by some other criterion, for example, presence in or absence from some other query range.

Very little is known about the complexity of range searching in the *group* arithmetic model, where subtractions are also allowed. Willard [153] considers dynamic orthogonal range searching data structures and shows that under some fairly restrictive assumptions, allowing subtractions cannot make these structures more efficient. Chazelle derives nontrivial (but very weak) lower bounds for offline halfplane [37] and orthogonal range counting [38] by examining the spectra of point-range incidence matrices. Absolutely nothing is known about range searching over more complicated domains such as rings or fields.

A useful special case of range searching is range *reporting*, where we want to know which points are in each query range, not just how many. Since the output from a reporting query can be quite large, we would like bounds of the form $\Theta(f(n) + r)$, where r is the number of points reported. Techniques similar to those used in the semigroup model imply lower bounds of this form in Tarjan's *pointer machine* model [144]. In this model, a data structure is represented by a directed graph in which every node has constant outdegree. The graph has a special starting node, or *source*. Each node in the graph is either unlabeled or is labeled with the index of exactly one point. A query is answered by visiting nodes in the graph, starting at the source and traversing edges in arbitrary order, until the index of every point in the query range has been seen at least once. Query algorithms are also allowed to modify the data structure by adding new nodes and adding or deleting edges between previously visited nodes. The query time is the number of nodes visited; all other aspects of computation are ignored. Chazelle [34] derives lower bounds for orthogonal range reporting in this model; his techniques were applied to simplex range reporting by Chazelle and Rosenberg [44]. Again, this model is inappropriate for studying range emptiness problems, since if a query range is empty, we don't need to do any work at all.

1.2 New Results

The remainder of this thesis divides naturally into two parts.

In Part I (Chapters 2 through 5), we derive lower bounds for several degeneracy-detection problems. For each of the problems we consider, the previously best lower bound, in any model of computation, was only $\Omega(n \log n)$ [138, 16].

In Chapter 2, we show that, in the worst case, $\Omega(n^d)$ sidedness queries are required to decide, given n points in $\mathbb{R}^d$, whether any $d + 1$ lie on a common hyperplane. Since there is an algorithm that solves this problem in time $O(n^d)$ [39, 68, 69], our lower bound is tight. Our lower bound follows from an extremely simple adversary argument, based on the construction of a set of points in general position with $\Omega(n^d)$ "collapsible" simplices, any one of which can be made degenerate without changing the result of any other sidedness query. If the algorithm doesn't do enough work, then the adversary can collapse some unchecked simplex, resulting in a degenerate set of points that the algorithm cannot distinguish from the original nondegenerate set. We also show that our lower bound still holds if we allow a wide variety of other computational primitives, such as coordinate comparisons and slope comparisons.

In Chapter 3, using similar techniques, we show that $\Omega(n^{\lceil d/2 \rceil - 1})$ sidedness queries are required to determine is the convex hull of n points in $\mathbb{R}^d$ is simplicial, or to count the number of convex hull facets. This matches known upper bounds when d is odd [36].

In Chapter 4, we show that $\Omega(n^{d+1})$ insphere queries are required to decide if any $d+2$ points lie on a common finite-radius sphere in $\mathbb{R}^d$. In the plane, $\Omega(n^3)$ incircle queries are required to decide if any four points lie on a common circle or line. These lower bounds are optimal [68, 69].

In Chapter 5, we prove an $\Omega(n^{\lceil r/2 \rceil})$ lower bound for the following problem: For some fixed linear equation in r variables, given a set of n real numbers, do any r of them satisfy the equation? Our lower bound holds in a restricted linear decision tree model, in which each decision is based on the sign of an arbitrary linear combination of r or fewer inputs. In this model of computation, our lower bound is as large as possible. Previously, this lower bound was known only for even r, and only for one special case [55, 56, 80]. A key step in the lower bound proof is the introduction of formal infinitesimals into the adversary configuration. We use a theorem of Tarski [146] to show that if we can construct a hard input containing infinitesimals, then for every decision tree algorithm, there exists a corresponding set of real numbers which is hard for that particular algorithm.

In Part II (Chapters 6 and 7), we derive lower bounds for problems that are typically solved by geometric divide-and-conquer techniques.

In Chapter 6, we establish new lower bounds on the complexity of the following basic geometric problem, attributed to John Hopcroft: Given a set of n points and m hyperplanes in $\mathbb{R}^d$, is any point contained in any hyperplane? We define a general class of *partitioning algorithms*, and show that in the worst case, for all m and n, any such algorithm requires time $\Omega(n \log m + n^{2/3} m^{2/3} + m \log n)$ in two dimensions, or $\Omega(n \log m + n^{5/6} m^{1/2} + n^{1/2} m^{5/6} + m \log n)$ in three or more dimensions. We obtain slightly higher bounds for the counting version of Hopcroft's problem in four or more dimensions. Informally, a partitioning algorithm divides space into a constant number of regions, determines which points and lines intersect which regions, and recursively solves the resulting subproblems. Our planar lower bound is within a factor of $2^{O(\log^*(n+m))}$ of the best known upper bound [111].[5] The previously best lower bound was only $\Omega(n \log m + m \log n)$ [138, 16]. We develop our lower bounds in two stages. First we define a combinatorial representation of the relative order type of a set of points and hyperplanes, called a *monochromatic cover*, and derive lower bounds on its size in the worst case. We then show that the running time of any partitioning algorithm is bounded below by the size of some monochromatic cover. As a related result, using a straightforward adversary argument, we derive a *quadratic* lower bound on the complexity of Hopcroft's problem in a surprisingly powerful decision tree model of computation.

Finally, in Chapter 7, we derive a lower bound of $\Omega(n \log m + n^{2/3} m^{2/3} + m \log n)$ for the following halfspace emptiness problem: Given a set of n points and m hyperplanes in $\mathbb{R}^5$, is every point above every hyperplane? This matches the best known upper bound up to polylogarithmic factors [107, 3, 29], and improves the previously best lower bound $\Omega(n \log m + m \log n)$ [138, 16]. We also obtain marginally better bounds in higher dimensions. Our lower bound applies to partitioning algorithms in which every query region is a polyhedron with a constant number of facets.

At the end of each chapter, we outline some relevant open problems and suggest directions for further research.

> *Sadly enough, surveying the status of lower bounds in computational geometry is a fairly easy task.*
>
> — Bernard Chazelle, "Computational Geometry: A Retrospective", 1994

[5] The iterated logarithm $\log^* n$ is 1 for all $n \leq 2$ and $1 + \log^*(\log_2 n)$ for all $n > 2$.