On the Relative Complexities of Some Geometric Problems*

Jeff Erickson
Computer Science Division
University of California
Berkeley, CA 94720-1776
jeffe@cs.berkeley.edu

October 17, 1995

Abstract

We consider the relative complexities of a large number of computational geometry problems whose complexities are believed to be roughly $\Theta(n^{4/3})$. For certain pairs of problems, we show that the complexity of one problem is asymptotically bounded by the complexity of the other. Almost all of the problems we consider can be solved in time $O(n^{4/3+\delta})$ or better, and there are $\Omega(n^{4/3})$ lower bounds for a few of them in specialized models of computation. However, the best known lower bound in any general model of computation is only $\Omega(n \log n)$.

The paper is naturally divided into two parts. In the first part, we consider a large number of problems that are harder than Hopcroft's problem. These problems include various ray shooting problems, sorting line segments in $\mathbb{R}^3$, collision detection in $\mathbb{R}^3$, and halfspace emptiness checking in $\mathbb{R}^5$. In the second, we survey known reductions among problems involving lines in three-space, and among higher dimensional closest-pair problems. Some of our results rely on the introduction of formal infinitesimals during reduction; we show that such a reduction is meaningful in the algebraic decision tree model.

Keywords: relative complexity, Hopcroft's problem, infinitesimal reduction.

Submitted to *Computational Geometry: Theory and Applications*

*Portions of this research were done while the author was visiting Universität des Saarlandes, Freie Universität Berlin, and Universiteit Utrecht. An extended abstract of this paper was presented at the 7th Canadian Conference on Computational Geometry [25].

1 Introduction

In this paper, we consider a number of problems whose best known algorithms run in roughly $O(n^{4/3})$ time. While it is generally believed that these algorithms are optimal, at least up to polylogarithmic or n^ε factors, the best known lower bound in any general model of computation is only $\Omega(n \log n)$. We characterize these problems by their *relative* complexities. That is, for certain pairs of problems, we show that the complexity of one problem is asymptotically bounded by the complexity of the other. Thus, a $o(n^{4/3})$-time algorithm for the "harder" problem is impossible without a similar algorithm for the "easier" one. Conversely, any significantly better lower bounds for the easier problem, in a sufficiently powerful model of computation, would immediately apply to the harder problem as well.

This paper is similar in spirit to the earlier work of Gajentaan and Overmars [28]. They introduce the class of 3SUM-hard problems, all of which are at least as hard as the following simple base problem: Given n numbers, do any three sum to zero? All of these problems seem to require $\Omega(n^2)$ time to solve; thus, some earlier papers describe them with the more suggestive but potentially misleading term "n^2-hard" [29]. (But see [7].) The present classification of what might be *informally* called "$n^{4/3}$-hard" problems is not so clean. Many of our reductions introduce extra logarithmic factors. More importantly, while we can easily describe artificial problems that are easier than anything presented here[1], we do not know if there is a single *natural* base problem to which all these problems can be reduced.

The paper is organized as follows. In Section 2, we give relevant definitions and background information. In Section 3 we describe a several problems that can all be reduced to Hopcroft's problem; a few related problems are described in Section 4. In Sections 5 and 6, we survey known reductions among problems involving lines in three-space and proximity problems in higher dimensions. We summarize our results in Section 7. We propose a number of open problems throughout the paper. Finally, we include some technical details regarding infinitesimal reductions in in a separate appendix.

2 Definitions and Background

2.1 Relative Complexity

Let $T_1(n)$ and $T_2(n)$ denote the complexities of two problems Π_1 and Π_2, expressed as functions of the input size n. If $T_1(n) = \Omega(T_2(n))$, or equivalently, if $T_2(n) = O(T_1(n))$, we say that **Π_1 is harder than Π_2**.[2] If $T_2(n) = O(T_1(n) \log^c n)$ for some constant c, we say that **Π_1 is almost harder than Π_2**. Typically, this means that Π_2 is solved by a binary search or parametric search [19, 37] using an algorithm for Π_1 as an oracle, although more complicated reductions are also possible. Finally, we say that **Π_1 is probably harder than Π_2** if Π_1 is almost harder than Π_2, and $T_1(n) = \Omega(n^{1+\delta})$ for some $\delta > 0$ implies that Π_1 is harder than Π_2.

We derive many our relative complexity results through the standard technique of *reductions*. Suppose we want to show that problem Π_1 is harder than problem Π_2. In the simplest reduction argument, given an instance X of Π_2 of size n, we transform it into an instance Y of Π_1 of size $O(n)$, call an algorithm for Π_1 as a subroutine, and transform the output of this subroutine to the

[1] Given any two problems Π_1 and Π_2, the following artificial problem is trivially easier than both of them: Given inputs X_1 and X_2 for Π_1 and Π_2, respectively, solve either problem. This construction can clearly be extended to any finite number of problems.

[2] It would be more accurate, but considerably unwieldier, to say "at least as hard as" or "no easier than".

result of Π_2. This reduction gives us the bound

$$T_2(n) \leq T_{2\to1}(n) + T_1(\mathrm{O}(n)) = T_{2\to1}(n) + \mathrm{O}(T_1(n)),$$

where $T_{2\to1}(n)$ is the time required to transform X into Y. (We assume here that the output can be transformed in constant time.) If $T_{2\to1}(n) = \mathrm{O}(T_1(n))$, this bound simplifies to $T_2(n) = \mathrm{O}(T_1(n))$, as desired. Because of the known lower bounds of $\Omega(n \log n)$ on each of the problems we consider, it will suffice that $T_{2\to1}(n) = \mathrm{O}(n \log n)$.

Our terminology is somewhat more flexible than the notation PR1 $\lll_{f(n)}$ PR2 ("PR1 is $f(n)$-solvable using PR2") used by Gajentaan and Overmars [28], or the earlier equivalent notation $\mathcal{A} \propto_{\tau(N)} \mathcal{B}$ ("$\mathcal{A}$ is $\tau(N)$-transformable to $\mathcal{B}$") used by Preparata and Shamos [41], both of which denote the existence of a *direct* reduction from one problem to another. Our terminology allows for more complicated reductions[3], but it hides the actual reduction time.

For example, suppose Π_2 is solved using a recursive divide-and-conquer strategy, similar to quicksort or mergesort: the input is divided into two halves in linear time, an algorithm for Π_1 is applied to the partitioned input, and problem Π_2 is recursively solved for each of the two partitions. Then we have the following upper bound for $T_2(n)$ in terms of $T_1(n)$.

$$T_2(n) \leq \mathrm{O}(T_1(n)) + 2T_2(n/2) = \sum_{i=0}^{\lceil \lg n \rceil} 2^i \mathrm{O}(T_1(n/2^i))$$

We can always simplify this upper bound to $T_2(n) = \mathrm{O}(T_1(n) \log n + n \log n) = \mathrm{O}(T_1(n) \log n)$, but if $T_1(n) = \Omega(n^{1+\delta})$, we can improve the bound to $T_2(n) = \mathrm{O}(T_1(n))$. Thus, Π_1 is probably harder than Π_2. The proofs of Theorems 5.1 and 6.2 include even more complex reductions.

We develop our relative complexity results primarily in the algebraic decision tree model of computation [43]. Most of our results hold in more general models of computation such as algebraic computation trees [5] or real RAMs [41] (and in some cases, even integer RAMs or Turing machines), but a few results rely on specific properties of the algebraic decision tree model. Specifically, some of our reductions introduce formal infinitesimals into the input before passing it to the Π_1 subroutine. See, for example, the proof of Theorem 3.4. In the appendix, we prove a few technical results that make this practice rigorous. These results follow almost immediately from simple properties of algebraic decision trees and real closed fields.

2.2 Known Lower Bounds

For each of the problems we consider, the best lower bound known in any general model of computation is only $\Omega(n \log n)$. These lower bounds follow from results of Steele and Yao in the algebraic decision tree model [43], and from results of Ben-Or in the algebraic computation tree model [5] or the equivalent real RAM model [41].

Better lower bounds are known for a few of these problems in less powerful models, but these models are inappropriate in more general settings. Chazelle [14, 13] has proven a number of lower bounds for online and offline range counting problems in the Fredman/Yao semigroup arithmetic model [27]. In this model, the points are given arbitrary weights from a fixed semigroup, and the complexity of a problem is given by the worst-case number of semigroup additions required to

[3]Formally, the reductions in [29] and [41] are quasilinear-time *many-one* reductions, and the reductions we consider here are quasilinear-time *oracle* reductions. The difference is exactly analogous to that between Karp reduction and Cook reduction in the theory of NP-completeness [30].

compute the answer. While this model works quite well for studying range counting problems, it is not at all applicable to decision or optimization problems.

Similarly, Erickson [24] has proven $\Omega(n^{4/3})$ lower bounds for a number of problems, including Hopcroft's problem (Problem A) and unit distance detection (Problem G), in what he calls the partitioning algorithm model. Informally, a partitioning algorithm splits the plane up into a constant number of regions, determines which of the input objects intersects which regions, and recursively solves the resulting subproblems. While this model describes existing algorithms for incidence detection problems reasonably well, it does not generalize to other contexts. Some problems, such as halfspace emptiness checking (Problem F), can be solved in linear time in this model. Others, such as line towering (Problem K), apparently cannot be solved at all.

3 Hopcroft's Problem and Its Friends

We begin with one of the oldest problems in computational geometry, first posed by John Hopcroft in the early 1980's.

> Problem A. *Hopcroft's Problem:* Given a set of points in the plane, does any point lie on the dual line of any other point?
> Best known upper bound: $O(n^{4/3}2^{O(\log^* n)})$ [36]

A number of different results suggest that the true complexity of Hopcroft's problem is $\Theta(n^{4/3})$. Erdős has constructed a set of n points and n lines with $\Omega(n^{4/3})$ point-line incidences. (See [27].) The existence of such a configuration implies that any algorithm that solves the *reporting* version of Hopcroft's problem must take time $\Omega(n^{4/3})$ in the worst case. Lower bounds due to Fredman [27] and Chazelle [13] imply that any algorithm that solves the online *counting* version of Hopcroft's problem — How many points are on each line? — by building a general purpose range searching data structure over the points and querying it once for each line, must take $\Omega(n^{4/3})$ time. More recent results of Chazelle [14] imply the same lower bound for the offline counting version. Finally, Erickson [24] has shown a lower bound of $\Omega(n^{4/3})$ for any partitioning algorithm that solves the original decision problem.

Hopcroft's problem is a special case of a large number of other more general problems, including the following. We leave the reductions as easy exercises for the reader.

- Detecting, counting, or enumerating incidences between a set of "point-like" geometric objects (points, line segments, circles, triangles, etc.) and a set of "line-like" geometric objects (lines, line segments, rays, circles, etc.)
- Finding the closest pair between a set of points (or point-like objects) and a set of lines (or line-like objects)
- Locating a set of points in an arrangements of lines
- Counting intersecting pairs among a single set of line segments (In contrast, counting intersection *points* is 3SUM-hard [28].)
- Triangular emptiness checking — Given a set of points and triangles, does any triangle contain a point?
- Halfplane range counting — Given a set of points and halfplanes, how many points are in each halfplane?

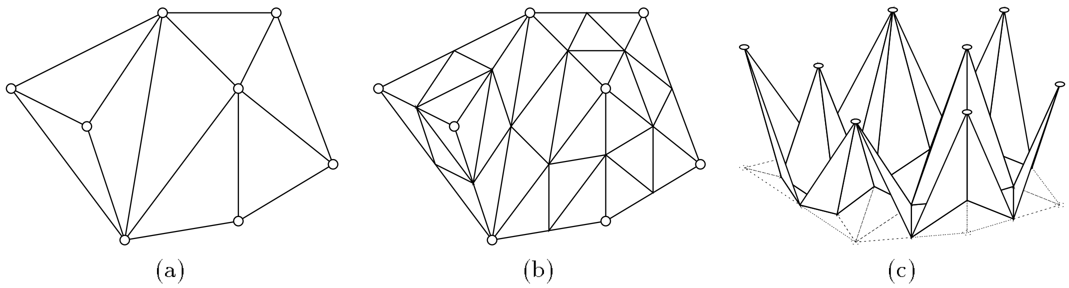

Figure 1. Transforming Hopcroft's problem into ray shooting over a polyhedral terrain (Theorem 3.1). (a) The initial triangulation of the points. (b) The refined triangulation. (c) The final terrain.

- Detecting intersections, or finding the closest pair, among lines (or line segments, circles, polyhedra, etc.) in $\mathbb{R}^3$

Rather than attempting to give an exhaustive list of easy reductions, we will describe only a few specific problems, for which the reductions may be less obvious.

3.1 Ray Shooting

Problem B. *Ray Shooting over a Polyhedral Terrain:* Given a polyhedral terrain and a set of rays in $\mathbb{R}^3$, does any ray hit the terrain?
Best known upper bound: $O(n^{4/3+\delta})$[4] [16]

The ray shooting problem is normally described as a selection problem: Which facet (if any) of the terrain does each ray hit first? In order to solve the usual ray shooting problem, it is obviously necessary to solve the decision problem we consider here. In other words, the selection problem is harder than the decision problem.

Theorem 3.1. *Ray shooting over a polyhedral terrain is harder than Hopcroft's problem.*

Proof: Given a set of points and lines, we construct a polyhedral terrain and a set of rays as follows. Construct an arbitrary triangulation of the points. Divide each triangle into four sub-triangles by bisecting its edges. See Figure 1(b). This ensures that no edge in the triangulation joins two of the original points. To create the terrain, lift the triangulation into $\mathbb{R}^3$, placing each of the original points on the plane $z = 1$ and all the other vertices on the plane $z = 0$. See Figure 1(c). To create the rays, chop each line at some very large x- or y-coordinate, and lift the resulting ray to the plane $z = 1$. A ray hits the terrain if and only if the corresponding line contains one of the original points. The entire reduction can be carried out in time $O(n \log n)$. □

This reduction may be unsatisfactory, as it is not clear, from a practical standpoint, whether the intersection of a ray and a single vertex of the terrain should qualify as a "hit". In most applications of the ray shooting problem, it suffices to consider only proper intersections, where the ray passes through the relative interior of a terrain facet. We can avoid this ambiguity altogether, at least in the algebraic decision tree model, by moving the rays from the plane $z = 1$ to the plane $z = 1 - \varepsilon$, where ε is a formal infinitesimal. (See the appendix.) We emphasize that infinitesimals

[4] In upper bounds of this form, δ represents an arbitrarily small positive constant. Multiplicative constants hidden in the big-Oh notation may depend on δ.

are required for this modified reduction to work in all cases. In order to determine a suitable real value to replace ε, we would have to compute a lower bound on the minimum non-zero distance between a point and a line in the original input configuration. Unfortunately, determining such a bound is itself harder than Hopcroft's problem.

Several other versions of ray-shooting are also harder than Hopcroft's problem. For example, given a set of non-intersecting triangles and a set of vertical rays in $\mathbb{R}^3$, does any ray hit any triangle? Given a set of line segments and a set of vertical rays in the plane, which segment does each ray hit? We leave the reductions as easy exercises.

3.2 Sorting Line Segments in Space

Problem C. *Segment Depth Order Verification:* Given a sequence of non-intersecting line segments in $\mathbb{R}^3$, is any segment below a segment following it in the sequence?
Best known upper bound: $O(n^{4/3+\delta})$ [6]

The segment depth order problem arises in the context of hidden surface removal. An extremely simple solution to the hidden surface removal problem is the "painter's algorithm": simply draw the objects from back to front, painting newer objects over the old ones. In order for this algorithm to succeed, however, it must first determine a valid back-to-front order for the objects.

In common applications of hidden surface removal, the objects in question are triangles. Here we consider the easier special case where the objects are line segments. Our reductions can be generalized to the case of nondegenerate triangles, at least in the algebraic decision tree model, by replacing any line segment with a long thin triangle, one of whose sides has infinitesimal length.

Theorem 3.2. *Segment depth order verification is harder than Hopcroft's problem.*

Proof: Suppose we are given a set $\{p_1, \ldots, p_n\}$ of points and a set $\{l_1, \ldots, l_n\}$ of lines. Lift each point p_i onto the horizontal plane $z = -i$ and each line l_j onto the plane $z = j$. Present the sequence $(p_1, p_2, \ldots, p_n, l_1, l_2, \ldots, l_n)$ to a depth order verification algorithm, where each point is considered a segment of zero length and each line a segment of infinite length. The depth order algorithm reports that some pair of segments is out of order if and only if there is a point-line incidence in the original input. □

Once again, this reduction may be unsatisfactory, since in most applications we never need to consider zero-length segments or infinite segments. Consider a restricted version of the depth order problem, where the inputs must be finite nontrivial segments. We can reduce Hopcroft's problem to this restricted problem, at least in the algebraic decision tree model, by replacing the points by segments of length ε, and the lines by segments of length $1/\varepsilon$, where ε is a formal infinitesimal. We emphasize that infinitesimals are necessary for this modified reduction to work in all cases.

It is possible that the set of objects being drawn does not have a valid depth order, and some preprocessing must be done before the painter's algorithm can be applied. This observation leads de Berg *et al.* [6] to consider the following problem.

Problem D. *Segment Cyclic Overlap:* Given a set of non-intersecting line segments in $\mathbb{R}^3$, does any subset overlap cyclically?
Best known upper bound: $O(n^{4/3+\delta})$ [6]

Theorem 3.3 (De Berg *et al.* [6]). *Segment depth order verification is harder than segment cyclic overlap.*

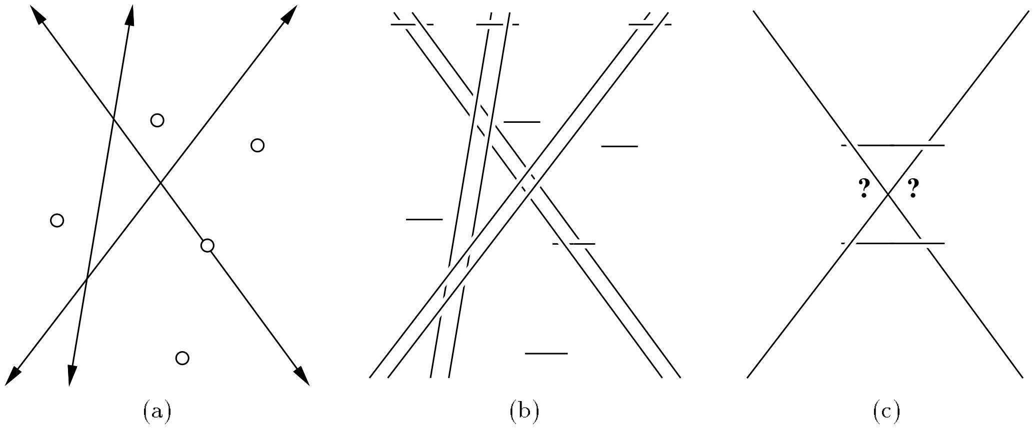

Figure 2. Transforming Hopcroft's problem into segment cyclic overlap (Theorem 3.4). (a) The initial configuration of points and lines. (b) The transformed set of segments, as seen from above (not to scale). The incidence in the original configuration is transformed into an overlap cycle. (c) Why no real number can be substituted for ε.

Proof: If the segments do not cyclically overlap, then they can be sorted using any $O(n \log n)$-time sorting algorithm. That is, the segment cyclic overlap problem can be decided by applying an $O(n \log n)$ sorting algorithm to the segments, and checking to see if the resulting sequence is a valid depth order. □

Theorem 3.4. *Segment cyclic overlap is harder than Hopcroft's problem, in the algebraic decision tree model of computation.*

Proof: Suppose we are given a set $P = \{p_1, p_2, \ldots, p_n\}$ of points and a set $L = \{l_1, l_2, \ldots, l_n\}$ of lines. Without loss of generality, none of the lines is horizontal. We produce a set of non-intersecting segments as follows. Replace each line l_i with two parallel lines l_i^+ and l_i^-, at horizontal distance ε to the right and left of l_i, respectively, where ε is a formal infinitesimal. (See the appendix.) Lift each line l_i^+ to the plane $z = 3i + 1$ and each line l_i^- to the plane $z = 3i - 1$.

Next, for each line l_i, add a segment s_i parallel to the x-axis and in the plane $z = 3i$, such that the projection of s_i intersects the projections of both l_i^+ and l_i^-, but is far away from every other point and line. It suffices to put s_i at a y-coordinate larger than that of any of the original points or line intersections. Alternately, we can make this y-coordinate equal to $1/\varepsilon$.

Finally, replace each point $p_j = (x_j, y_j)$ with a line segment $\hat{p}_j$ with endpoints $(x_j - 2\varepsilon, y_j, -3n)$ and $(x_j + 2\varepsilon, y_j, 6n)$. These line segments are almost parallel to the z-axis. Our final collection of $4n$ line segments consists of l_i^+, l_i^-, s_i, and $\hat{p}_j$ for all i, j. The entire transformation can be performed in time $O(n \log n)$.

If there are no incidences between P and L, then none of the segments $\hat{p}_j$ are above or below any other segment. Since the other segments are all parallel to the xy-plane, there are no cyclic overlaps. On the other hand, if there is an incidence between l_i and p_j, then the segment $\hat{p}_j$ slips between the segments l_i^+ and l_i^-, going over the former and under the latter, introducing a cyclic overlap. See Figure 2. □

Unlike Theorems 3.1 and 3.2, we are unable to extend this result into the algebraic computation tree model by reducing to a "boundary case". The primary reason for introducing the infinitesimals into the reduction is to eliminate interactions between points and *pairs* of lines. Fix any real number ε. If a point is within, say, $\varepsilon/2$ of more than one line, then replacing the point by a segment whose vertical projection has length ε may introduce a cycle. We can get rid of such a cycle by changing the vertical order of the lines, but there are situations in which no order can eliminate all such cycles. See Figure 2(c). Determining a real value for ε that is sufficiently small to avoid these difficulties would involve finding a lower bound on the smallest non-zero point-line distance, which is itself harder than Hopcroft's problem.

We close this section with an open problem related to hidden surface removal. Agarwal and Matoušek [4] describe an output-sensitive hidden surface removal algorithm that runs in time $O(n^{2/3+\delta}k^{2/3} + n^{1+\delta})$, where n is the input size and k is the combinatorial complexity of the resulting image. If $k = \Theta(n)$, this running time reduces to $O(n^{4/3+\delta})$. Is hidden surface removal harder than Hopcroft's problem, if the complexity of the output is known in advance to be linear? The reductions we have described so far lead to images with quadratic complexity.

3.3 Collision Detection

Problem E. *Polyhedron Intersection:* Given two triangulated simple polyhedra in $\mathbb{R}^3$, do they intersect?
Best known upper bound: $O(n^{8/5+\delta})$ [40]

This problem is an important special case of three-dimensional motion planning. A typical motion planning problem asks for a sequence of translations (and/or rotations) that take a "robot" from one position (and/or orientation) to another, such that the moving object never collides with a given set of obstacles. Of course, in order to solve this motion planning problem, an algorithm must be able to decide if the robot intersects any of the obstacles. In other words, motion planning is harder than detecting intersections.

Extremely simple versions of translational motion planning in $\mathbb{R}^3$ are 3SUM-hard [28]. Here we are considering the case of a non-convex robot in the presence of a single non-convex obstacle [45].

Theorem 3.5. *Polyhedron intersection is harder than Hopcroft's problem, in the algebraic decision tree model of computation.*

Proof: Given a set $P = \{p_1, p_2, \ldots, p_n\}$ of points and a set $L = \{l_1, l_2, \ldots, l_n\}$ of non-horizontal lines, we produce two simple polyhedra, one corresponding to P and the other to H, as follows. We begin with everything in the plane $z = 0$ in $\mathbb{R}^3$.

To produce the point polyhedron, surround each point by a triangle of infinitesimal diameter ε, and surround the entire set of triangles with a rectangle. Let the top and bottom y-coordinates of this rectangle be $y_{\max}$ and $y_{\min}$, respectively. Triangulate the collection of points and line segments. Lift the original points up to the plane $z = 3n$, creating long vertical spikes. Finally, to close the polyhedron, add four triangles, forming an inverted pyramid "underneath" the rectangle. The resulting simple polyhedron clearly has total complexity $O(n)$, and can be constructed in $O(n \log n)$ time. See Figure 3(b).

To produce the line polyhedron, start by lifting each line l_i to the plane $z = n + i$. Place a rectangle in the plane $y = y_{\max} + 1$ that is large enough in the x- and z-directions to intersect all the lines. Now we essentially repeat the previous process. Replace the intersection points of the lines and the rectangle with triangles of infinitesimal diameter ε, triangulate everything inside the rectangle, and lift the intersection points *along the lines* to the plane $y = y_{\min} - 1$, creating long horizontal spikes. These spikes clearly do not intersect. Finally, to turn the polyhedral surface into a polyhedron, add four triangles "behind" the rectangle. The resulting simple polyhedron has total complexity $O(n)$ and can be constructed in $O(n \log n)$ time. See Figure 3(c).

If any point $p_i \in P$ lies on a a line $l_j \in L$, then the ith vertical spike intersects the jth horizontal spike. On the other hand, if the point is at distance $d > 0$ from the line, then these two spikes are never closer than $d - 2\varepsilon > 0$ apart, since ε is a formal infinitesimal. Thus, if there are no incidences in the original input, the polyhedra do not intersect. See Figure 3(d). □

3.4 Range Searching

Hopcroft's problem is one of the simplest examples of range searching. In a typical range searching problem, we are given a set of points and a set of ranges (typically axis-parallel boxes, hyperplanes, halfspaces, spheres, or simplices), and are asked to report or count the points contained in each range. Results in this area are surveyed by Matoušek [33].

We have already mentioned a number of two-dimensional range searching problems that are harder than Hopcroft's problem. Here we consider a higher-dimensional example.

Problem F. *Halfspace Emptiness Checking in* $\mathbb{R}^5$*:* Given a set of points and hyperplanes in $\mathbb{R}^5$, is every point above every hyperplane?
Best known upper bound: $O(n^{4/3} \log^{O(1)} n)$ [34]

Counting the number of points in each halfspace is apparently much more difficult than deciding if every halfspace is empty. Recall that halfspace range counting is already harder than Hopcroft's problem in two dimensions, whereas the corresponding emptiness problem can be decided in $O(n \log n)$ time using any convex hull algorithm.

Theorem 3.6. *Halfspace emptiness checking in* $\mathbb{R}^5$ *is harder than Hopcroft's problem.*

Proof: Every point and hyperplane in $\mathbb{R}^d$ can be represented in homogeneous coordinates by a vector in $\mathbb{R}^{d+1}$, so that the relative orientation of any point p and any hyperplane h is determined by the sign of the inner product $\langle p, h\rangle$. Specifically, if $\langle p, h\rangle > 0$ then p is above h; if $\langle p, h\rangle = 0$, then p is contained in h; and if $\langle p, h\rangle < 0$, then p is below h.

Using this observation, we can reformulate the two problems as follows.

- Hopcroft's problem: Given a set of red and blue vectors in $\mathbb{R}^3$, is there a red vector p and a blue vector h such that $\langle p, h\rangle = 0$?
- Halfspace emptiness checking in $\mathbb{R}^5$: Given a set of red and blue vectors in $\mathbb{R}^6$, is there a red vector p and a blue vector h such that $\langle p, h\rangle \le 0$?

Now consider the function $\sigma : \mathbb{R}^3 \to \mathbb{R}^6$ defined as

$$\sigma(x, y, z) = (x^2, y^2, z^2, \sqrt{2}xy, \sqrt{2}yz, \sqrt{2}zx).$$

This function squares inner products: $\langle \sigma(p), \sigma(h)\rangle = \langle p, h\rangle^2$. Thus, we can transform any set of planar points and lines into a set of of five-dimensional points and hyperplanes in linear time, by applying the function σ to the entire set, so that no point is below any hyperplane and all incidences are preserved. □

The best known upper bound for halfspace emptiness checking in *four* dimensions is also $O(n^{4/3} \log^{O(1)} n)$ [34]. Is four-dimensional halfspace emptiness checking also harder than Hopcroft's problem?

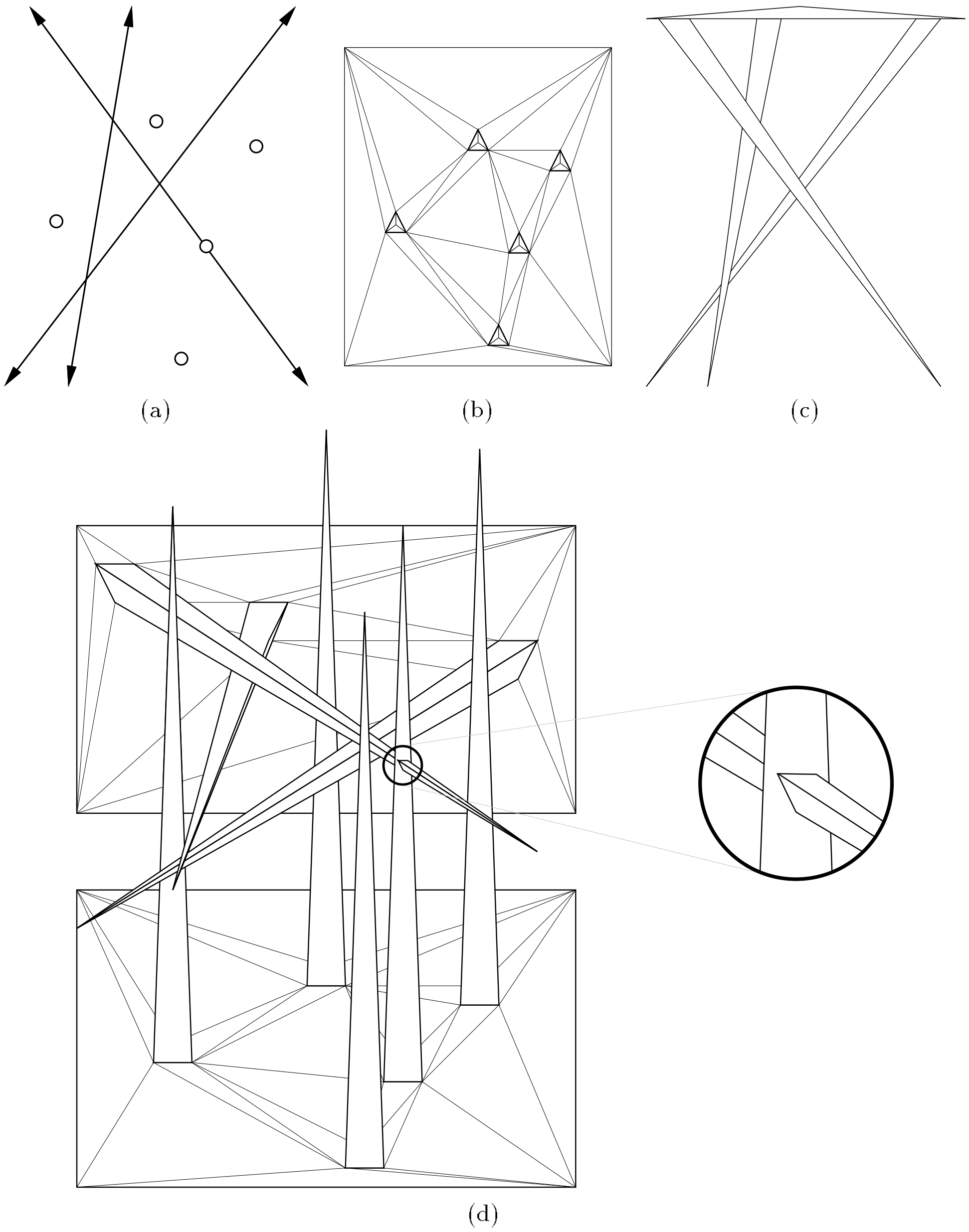

Figure 3. Transforming Hopcroft's problem into polyhedron intersection (Theorem 5) (a) The initial configuration of points and lines. (b) The point polyhedron, seen from above. (c) The line polyhedron, seen from above. (d) Oblique view of the two polyhedra together. The incidence is transformed into an intersection. Drawings are not to scale — the spikes are actually infinitesimally thin.

4 Planar Distance Problems

Problem G. *Unit Distance Detection:* Given a set of points in the plane, is any pair of points at unit distance?
Best known upper bound: $O(n^{4/3} \log^{2+\delta} n)$ [32].

This problem is actually quite similar to Hopcroft's problem in many respects. Both problems can be solved using similar divide-and-conquer algorithms; compare [32] and [36]. Where algorithms for Hopcroft's problem typically exploit the duality between points and lines, algorithms for detecting unit distances typically exploit the similar duality between points and unit circles. Any collection of points and lines with no incidences can be approximated arbitrarily closely by a combinatorially equivalent set of points and unit circles. Using this observation, Erickson proves a $\Omega(n^{4/3})$ lower bound for this problem in the partitioning algorithm model [24].

We conjecture, but are unable to prove, that detecting unit distances is harder than detecting point-line incidences. Both the unit distance problem and Hopcroft's problem are special cases of several other harder problems, including as point-circle incidence detection in the plane, point-plane incidence detection in $\mathbb{R}^3$, and unit-distance detection in $\mathbb{R}^3$.

Problem H. *Distance Selection:* Given a set of points in the plane and an integer k, what is the kth smallest interpoint distance?
Best known upper bound: $O(n^{4/3} \log^{3+\delta} n)$ [32]

Theorem 4.1. *Distance selection is almost harder than unit distance detection.*

Proof: We can detect unit distances with a binary search over the $\binom{n}{2}$ possible values for k, using a distance selection algorithm at each step in the search. □

Problem I. *Distance Ranking:* Given a set of points in the plane, how many pairs of points are closer than unit distance apart?
Best known upper bound: $O(n^{4/3} \log^{3+\delta} n)$ [32]

Theorem 4.2. *Distance selection is almost harder than distance ranking. Distance ranking is almost harder than distance selection.*

Proof: The binary search algorithm described in the proof of Theorem 4.1 can also be used to solve the distance ranking problem. To select the kth smallest distance, we can perform a parametric search over the space of interpoint distances, using a distance ranking algorithm as an oracle [2]. □

Theorem 4.3. *Distance ranking is harder than unit distance detection in the algebraic decision tree model.*

Proof: To detect the presence or absence of unit distances in a set of points, we call a distance ranking algorithm twice, once on the original set of points, and once on the set of points scaled by a factor of $1+\varepsilon$, where ε is an infinitesimal. The unit distance rank is the same in both sets if and only if the original set contains no unit distances. □

Katoh and Iwano [31] consider the related problem of actually enumerating the k farthest pairs among a given set of points in the plane. They describe an algorithm that runs in time $O(\min\{n^2, n \log n + k^{4/3} \log n / \log^{1/3} n\})$, which simplifies to $O(n^{4/3} \log^{2/3} n)$ when $k = \Theta(n)$. Is enumerating the n farthest pairs harder than detecting unit distances?

5 More Lines and Segments in Space

In this section, we describe a number of known reductions between problems involving lines and line segments in 3-space. We first consider the following special case of polyhedron intersection (Problem E). A *polyhedral terrain* is a polyhedral surface that intersects any vertical line in at most one point.

Problem J. *Polyhedral Terrain Intersection:* Given two triangulated polyhedral terrains in $\mathbb{R}^3$, do they intersect?
Best known upper bound: $O(n^{4/3}\log^{O(1)} n)$ [16, 15, 34]

Whether terrain intersection is harder than Hopcroft's problem is completely open. Note that reduction in Theorem 3.5 does not produce two polyhedral terrains. We can modify the reduction so that it does produce a polyhedral terrain from the lines, but the resulting terrain would have quadratic complexity, and thus would be worthless for our purposes.

Problem K. *Line Towering:* Given a set of red and blue lines in $\mathbb{R}^3$, are all the red lines above all the blue lines?
Best known upper bound: $O(n^{4/3}\log^{O(1)} n)$ [15, 34]

Theorem 5.1 (Chazelle *et al.* [16]). *Line towering is almost harder than polyhedral terrain intersection.*

Proof: There are two distinct ways that two polyhedral terrains can intersect: either a vertex of one terrain is on the wrong side of a facet of the other, or an edge of the higher terrain is below an edge of the lower terrain. The first case can be decided in $O(n \log n)$ time using any optimal algorithm for point location in planar subdivisions [20]. The second case is what makes the problem difficult.

Chazelle *et al.* [16] show that this second case can be reduced to several instances of line towering. Let $\hat{S}$ and $\hat{T}$ be the edges of the two terrains, and let S and T be their vertical projections onto the plane $z = 0$. To see if any segment in $\hat{S}$ lies below a segment in $\hat{T}$, it suffices to consider all pairs of segments in S and T that intersect. To do this, we divide S into several (not necessarily disjoint) subsets S_i, and T into corresponding subsets T_i, such that every segment in S_i intersects every segment in T_i, and every intersection is represented in some subset pair (S_i, T_i). If we use the algorithm in [16], the total size of these subsets is $\sum_i |S_i| + |T_i| = O(n \log^2 n)$, where $n = |S| + |T|$. We refer the reader to [16] for details on how these subsets are actually constructed.

Finally, it suffices to check, for each pair of subsets S_i and T_i, whether every segment in $\hat{S}_i$ is above every segment in $\hat{T}_i$. To do this, we extend the segments $\hat{S}_i$ and $\hat{T}_i$ into lines and invoke a line towering subroutine. This extension does not introduce any new bichromatic intersections in the projection.

Let $T_{LT}(n)$ be the time taken by the line towering subroutine, given a set of n lines as input. The total time for all calls to the line towering subroutine is

$$\begin{aligned}
\sum_i T_{LT}(|S_i| + |T_i|) &\leq T_{LT}\left(\sum_i (|S_i| + |T_i|)\right) \\
&= T_{LT}(O(n \log^2 n)) \\
&= T_{LT}(n) \cdot O(\log^4 n)
\end{aligned}$$

The last inequality follows from the crude estimate $T_{LT}(n) = O(n^2)$. The calls to the line towering subroutine dominate the running time of the terrain intersection algorithm. $\square$

Problem L. *Line Cyclic Overlap:* Given a set of non-intersecting lines in $\mathbb{R}^3$, do any three lines overlap cyclically?
Best known upper bound: $O(n^{4/3} \log^{O(1)} n)$ [17, 15, 34]

This problem is clearly a special case of segment cyclic overlap (Problem D). We conjecture, but are unable to prove, that this special case is harder than Hopcroft's problem.

Theorem 5.2 (Chazelle *et al.* [17]). *Line towering is almost harder than line cyclic overlap.*

Proof: The line cyclic overlap problem can be solved by applying any sorting algorithm to the lines, and checking whether the resulting sequence is a valid depth order. (Compare Theorem 3.3.) The sequence can be verified as follows. Split the sequence of lines into two equal halves, verify each of the two halves recursively, and use a line towering algorithm to check that every line in the first half is above every line in the second half. □

Problem M. *Largest Vertical Distance:* Given a set of lines in $\mathbb{R}^3$, find the largest vertical distance between two lines.
Best known upper bound: $O(n^{4/3} \log^{O(1)} n)$ [39, 15, 34]

Theorem 5.3 (Pellegrini [39]). *Line towering is almost harder than largest vertical distance.*

Proof: The largest vertical distance among a given set of lines can be solved as follows. Split the set of lines into two subsets R and B, and recursively find the largest vertical distance within each subset. Let B' be the result of moving the set B down a distance of δ. If our line towering subroutine tells us that R lies completely above B', then the largest distance between R and B is less than δ. Thus, we can find the largest bichromatic vertical distance with parametric search, using a line towering algorithm as an oracle. □

We emphasize that the previous reduction does not actually give us the farthest pair of lines, unless a bichromatic line intersection algorithm is also called.

Theorem 5.4 (Chazelle *et al.* [15]). *Halfspace emptiness checking in $\mathbb{R}^5$ is probably harder than line towering.*

Proof: Consider the special case of *consistently oriented* sets of lines, in which the projections of the blue lines onto the xy-plane all have higher slope than the projections of the red lines. In this case, using Plücker coordinates [44], we can express each red line as a point in $\mathbb{R}^5$ and each blue line as a hyperplane in $\mathbb{R}^5$, so that relative orientation is preserved.

The general problem can be solved using the following divide-and-conquer approach. The median slope among the xy-projections of all the lines naturally partitions the red lines into two subsets R_1 and R_2, and the blue lines into B_1 and B_2, so that the projected slopes in $R_1 \cup B_1$ are all larger than the projected slopes of $R_2 \cup B_2$. The subset pairs (R_1, B_2) and (R_2, B_1) are consistently oriented, and thus can be checked using the Plücker space algorithm described previously. The other two pairs of subsets are checked recursively. □

6 Higher-Dimensional Distance Problems

Problem N. *Points Outside Intersecting Unit Balls in* $\mathbb{R}^3$*:* Given a set of points and a set of unit balls in $\mathbb{R}^3$, such that every ball contains the origin, is any point contained in any ball?
Best known upper bound: $O(n^{4/3}\log^{4/3} n)$ [3]

This computational problem is closely related to the following open combinatorial problem: What is the worst-case combinatorial complexity of the union of n intersecting unit balls in $\mathbb{R}^3$? The best known bounds are only $O(n^2)$ and $\Omega(n)$. If the complexity of the union is linear, then Problem N can be solved in $O(n \log n)$ expected time using random sampling techniques [18]. If we allow balls of even two different sizes, or do not require the balls to have a common intersection, their union can have complexity $\Omega(n^2)$. The *intersection* of unit balls, on the other hand, always has complexity $O(n)$.

Problem N is a special case of several other harder range searching algorithms. The reductions derive directly from simple geometric transformations. We leave the details as exercises.

- Unit-spherical emptiness checking in $\mathbb{R}^3$ — Given a set of points and a set of unit balls, is any point inside any ball?
- Anti-spherical emptiness checking in $\mathbb{R}^3$ — Given a set of points and a set of balls, is *every* point inside *every* ball?
- Halfspace emptiness checking in $\mathbb{R}^4$ — Given a set of points and halfspaces, does any halfspace contain a point? (See Problem F.)
- Unit anti-spherical emptiness checking in $\mathbb{R}^4$ — Given a set of points and a set of unit balls, is *every* point inside *every* ball?

In the remainder of this section, we describe known reductions from proximity problems in three and four dimensions to some of the range searching problems listed above. The reductions we describe immediately extend to problem N.

Problem O. *Bichromatic Closest Pair in* $\mathbb{R}^3$*:* Given a set of red and blue points in $\mathbb{R}^3$, find the closest red-blue pair.
Best known upper bound: $O(n^{4/3}\log^{4/3} n)$ [3]

Theorem 6.1. *Bichromatic closest pair is harder than unit spherical range checking, and unit spherical range checking is almost harder than bichromatic closest pair.*

Proof: We can solve any unit spherical range checking problem by looking for the closest foreign neighbor among the points and the centers of the spheres. If the distance separating the closest point-center pair is less than unity, then the point is in the ball; otherwise, every point is outside every ball. Conversely, we can find the bichromatic closest pair with a parametric search over the space of interpoint distances, using a unit spherical range checking algorithm as an oracle. □

Problem P. *Euclidean Minimum Spanning Tree in* $\mathbb{R}^3$*:* Given a set of points in $\mathbb{R}^3$, construct its Euclidean minimum spanning tree.
Best known upper bound: $O(n^{4/3}\log^{4/3} n)$ [3]

Theorem 6.2 (Agarwal *et al.* [3]). *Bichromatic closest pair is probably harder than Euclidean minimum spanning tree, and Euclidean minimum spanning tree is harder than bichromatic closest pair.*

Proof: Agarwal *et al.* [3, Theorem 5] describe a rather complicated reduction for the first half of this theorem. Their algorithm decomposes the input set S into several pairs of "α-separated" subsets (R_i, B_i), such that for any two points $r, b \in S$, there is some subset pair such that $r \in R_i$ and $b \in B_i$. These subset pairs are constructed essentially by traversing a constant number of implicit three-dimensional range trees [41]. The total number of α-separated subset pairs constructed by their algorithm is $O(n \log^2 n)$. Each edge of the Euclidean minimum spanning tree is guaranteed to join the bichromatic nearest neighbors between some pair of α-separated subsets. Thus, once all the bichromatic nearest neighbors are found, the minimum spanning tree can be constructed in time $O(n \log^2 n)$. We refer the reader to [47] and [3] for definitions and further details.

Let $T_{EMST}(n)$ denote the overall time to construct the Euclidean minimum spanning tree, $T_i(n)$ the time to traverse an i-dimensional range tree, and $T_{BCP}(n)$ the time to find the bichromatic closest pair, given a total of n points as input. We have the following recurrences.

$$\begin{aligned} T_{EMST}(n) &\leq O(T_3(n)) + O(n \log^2 n) \\ T_i(n) &\leq 2T_i(n/2) + T_{i-1}(n) \qquad [i = 1, 2, 3] \\ T_0(n) &\leq T_{BCP}(n) \end{aligned}$$

With no additional assumptions, we immediately have $T_{EMST}(n) = O(T_{BCP}(n) \log^3 n)$. Moreover, if $T_{BCP}(n) = \Omega(n^{1+\delta})$ for some $\delta > 0$, then $T_{EMST}(n) = O(T_{BCP}(n))$.

The second half of the theorem is obvious. □

Problem Q. *Nearest Foreign Neighbors in* $\mathbb{R}^3$*:* Given a set of colored points in $\mathbb{R}^3$, find for each point the closest point with a different color.
Best known upper bound: $O(n^{4/3} \log^{4/3} n)$ [1]

This problem is the obvious generalization of the bichromatic closest pair problem (Problem O) to more than two colors.

Theorem 6.3 (Yao [47]). *Nearest foreign neighbors is almost harder than Euclidean minimum spanning tree.*

Proof: We can construct the minimum spanning tree using the following algorithm, originally published by Borůvka in 1926 [9, 46]. We start with a forest of n one-vertex trees. In each phase of the algorithm, we find the minimum weight edge leaving each tree in the forest, and add these edges to the evolving forest. In each phase, the number of trees drops by at least a factor of two. Thus, after $O(\log n)$ phases, the forest contains only the minimum spanning tree. In the geometric setting, each phase can be easily implemented using a nearest foreign neighbors algorithm, by coloring each point according to the tree that contains it. □

Problem R. *Bichromatic Farthest Pair in* $\mathbb{R}^4$*:* Given a set of red and blue points in $\mathbb{R}^4$, find the farthest red-blue pair.
Best known upper bound: $O(n^{4/3} \log^{O(1)} n)$ [34]

The following result is exactly analogous to Theorem 6.1.

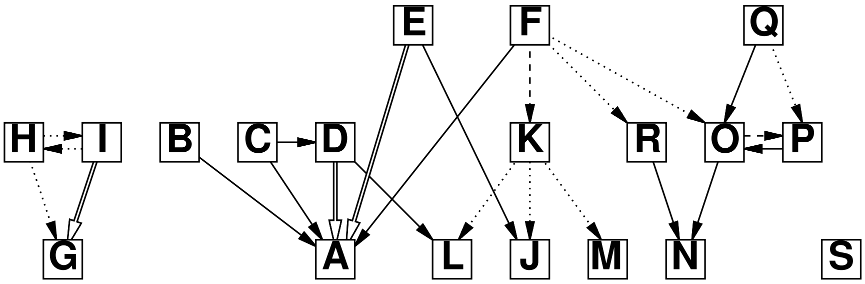

Figure 4. Summary of results. Arrows point from harder to easier problems. Dotted arrows indicate "almost harder". Dashed arrows indicate "probably harder". Outlined arrows indicate results that only hold in the algebraic decision tree model.

Theorem 6.4. *Bichromatic farthest pair is harder than unit anti-spherical range checking, and unit anti-spherical range checking is almost harder than farthest foreign pair.*

The algorithm in [34] can also be used to compute the diameter of a single set of points in $\mathbb{R}^4$, in time $O(n^{4/3} \log^{O(1)} n)$. Is computing diameters in $\mathbb{R}^4$ harder than (say) halfspace emptiness checking in $\mathbb{R}^3$?

7 Summary

We have described reductions relating the computational complexities of a number of geometric problems. Figure 4 summarizes our results, and suggests a number of open problems. Which is easier: Hopcroft's problem or unit distance detection? Can we better relate the complexities of the problems in Section 5? Is there a single natural problem that is easier than all the problems we have considered?

We mention in closing one more interesting problem that we have been unable to relate to any of the others.

Problem S. *Extreme Points:* Given a set of points in $\mathbb{R}^4$, is any point a convex combination of other points? Equivalently, is every point a vertex of the set's convex hull?
Best known upper bound: $O(n^{4/3} \log^{O(1)} n)$ [35, 12]

Ultimately, we would like a proof that all these problems require $\Omega(n^{4/3})$ time in algebraic decision tree model, as we strongly suspect. Unfortunately, proving a lower bound of $\omega(n \log n)$ for *any* decision problem in *any* general model of computation seems to be completely out of reach at present.

Appendix: Infinitesimal Reductions and Algebraic Decision Trees

A.1 Algebraic Decision Trees

We begin with some definitions. A dth-order algebraic decision tree is a finite, rooted, directed ternary tree. If the order d is unspecified, we take it to be a constant. Each internal node v is labeled with a *query polynomial* $q_v \in \mathbb{R}[t_1, \ldots, t_n]$ of degree at most d. The outgoing edges of each internal node are labeled -1, 0, and $+1$. Each leaf is given a label from some label set $\mathcal{L}$. In Steele and Yao's original formulation [43], every leaf is labeled either "yes" or "no", but to reflect common usage of the model, we allow for a much larger variety of outputs. The label set $\mathcal{L}$ is a product of a finite number of *primitive* label sets $\mathcal{L}_1 \times \mathcal{L}_2 \times \cdots \times \mathcal{L}_k$, each of which is either a set of combinatorial objects (booleans, integers, permutations, graphs, convex polytopes, etc.) or a set of polynomials of degree at most d. At the risk of confusing the reader, we will use the same symbol ℓ to refer both to a leaf and its label.

Given a vector $X \in \mathbb{R}^n$ as input, we compute with an algebraic decision tree by traversing a path from the root to a leaf. At each internal node v in the path, the sign of the query polynomial $q_v(X)$ determines which edge to traverse next. We write $A \xrightarrow{X} v$ when computation with the tree A on input X reaches the node v. When we reach a leaf ℓ, the leaf's label $\ell = (\ell_1, \ell_2, \ldots, \ell_k) \in \mathcal{L}$ determines the algorithm's output. For each i, if $\mathcal{L}_i$ is a set of polynomials, we return the real value $\ell_i(X)$; otherwise, we return the object ℓ_i directly. We denote by $\ell(X)$ the output produced at the leaf ℓ given the input configuration X.

Every algebraic decision tree computes a function from some real vector space $\mathbb{R}^n$ to some output space $\mathbb{R}^m \times \mathcal{O}$, where $\mathcal{O}$ is a discrete set. In contexts where the structure of the output space is unimportant, we let $\mathcal{O}$ refer to the entire space, rather than just its discrete component. The complexity of a function Π is the minimum depth of any algebraic decision tree that computes Π. In the main body of the paper, when we speak of a "problem", we mean a family of functions $\mathbf{\Pi} = \{\Pi^1, \Pi^2, \ldots\}$, where each function has the form $\Pi^n : \mathbb{R}^n \to \mathcal{O}$. The complexity[5] of a problem $\mathbf{\Pi}$ is a function that expresses the complexity of each Π^n in terms of the input size n.

A.2 Reductions

One problem with applying reduction arguments in the algebraic decision tree model is that the model doesn't allow for the storage of temporary results. Nevertheless, we can still apply the standard reduction argument by representing the transformed input implicitly, as described in the following lemma.

Lemma A.1. *Let $A_{1\to 2}$ and A_2 be two algebraic decision trees that compute functions $\Pi_{1\to 2} : \mathbb{R}^n \to \mathbb{R}^m$ and $\Pi_2 : \mathbb{R}^m \to \mathcal{O}$, respectively. Then there is an algebraic decision tree A_1 that computes the composition $\Pi_1 = \Pi_2 \circ \Pi_{1\to 2}$, whose depth (resp. order) is the sum of the depths (resp. orders) of $A_{1\to 2}$ and A_2.*

Proof: Let $X \in \mathbb{R}^n$ denote a typical input to $A_{1\to 2}$, and $Y = \Pi_{1\to 2}(X) \in \mathbb{R}^m$ be the corresponding output. By definition, each leaf ℓ in $A_{1\to 2}$ is labeled with a vector $\ell = (\ell_1, \ell_2, \ldots, \ell_m)$ of polynomials such that if $A_{1\to 2} \xrightarrow{X} \ell$, then $Y = \ell(X) = (\ell_1(X), \ell_2(X), \ldots, \ell_m(X))$.

To create the new algebraic decision tree A_1, we replace each leaf ℓ of $A_{1\to 2}$ with a modified copy of A_2, in which we replace each query polynomial q_v with the polynomial $q_v(\ell)$ and each leaf polynomial ℓ'_i with the polynomial $\ell'_i(\ell)$. Thus, during computation in the copy of A_2 replacing the leaf ℓ, instead of evaluating $q_v(Y)$ directly, we evaluate $q_v(\ell(X))$. We easily verify that A_1 has the desired properties. $\square$

[5] Strictly speaking, this defines the *nonuniform* complexity, since in principle there is no similarity between the trees used for different input sizes. Unfortunately, we can say little or nothing about uniform complexity; in fact, it is not entirely clear how to formally *define* uniform complexity in the algebraic decision tree model. There are problems for which the best known upper bounds on their uniform and nonuniform complexities differ significantly. See, for example, Fredman's $O(n^2)$-time algorithm for sorting $X + Y$ [26] or Meyer auf der Heide's polynomial-time algorithm for the knapsack problem [38].

A.3 Infinitesimals

Let $\mathbb{R}[\varepsilon]$ denote the ring of real polynomials in ε, ordered so that ε is positive, but less than any positive real number. The symbol ε represents a formal *infinitesimal.* Previous applications of infinitesimals in computational geometry include various perturbation techniques [21, 22, 48], algorithms dealing with real semi-algebraic sets [10, 11], and at least one lower bound argument [23]. The ring $\mathbb{R}[\varepsilon]$ is an ordered subset of the field of rational functions $\mathbb{R}(\varepsilon)$, which is in turn an ordered subset of the real closed field $\overline{\mathbb{R}(\varepsilon)}$. For an introduction to the theory of real closed fields, including formal definitions, we refer the reader to [8] or [42].

An *elementary formula* is a finite quantified boolean formula, each of whose clauses is a multivariate polynomial inequality with real coefficients. An elementary formula *holds in* an ordered subset S of an ordered ring R if and only if the formula has no free variables, and the formula is true if we interpret each variable as an element of S, addition and multiplication as ring operations in R, and comparisons according to the linear order on R. A classic result of Tarski [46], called the Transfer Principle, states that an elementary formula holds over $\mathbb{R}$ if and only if it holds over any other real closed field.

If the ordered ring R contains the reals, we can think of any real polynomial as a function from R to itself. Thus, we can reasonably consider the behavior of any algebraic decision tree given elements of R, or any ordered subset $S \subseteq R$, as input instead of real numbers. If an algebraic decision tree A computes a function $\Pi : \mathbb{R}^n \to \mathbb{R}^m \times \mathcal{O}$ over the reals, then it also computes a function $\Phi : S^n \to R^m \times \mathcal{O}$ over the ordered set S. Our first technical result implies that in a sense these two functions are the same.

Lemma A.2. *If two algebraic decision trees compute the same function over the reals, then they also compute the same function over any ordered subset of any real closed field.*

Proof: Let $(v_0, v_1, \ldots, v_t)$ be the path of vertices from the root v_0 to another node v_t in some algebraic decision tree A. We can write the expression $A \xrightarrow{X} v_t$ formally as

$$\bigwedge_{i=0}^{t-1} (q_{v_i}(X) \diamond_i 0),$$

where $\diamond_i$ is $<$, $=$, or $>$, depending on whether the edge from v_i to v_{i-1} is labeled -1, 0, or $+1$, respectively.

Let A and B be two algebraic decision trees, and let L_A and L_B denote the sets of leaves in A and B, respectively. We can write the statement "A and B compute the same function" as

$$\forall X : \bigwedge_{\ell \in L_A} \bigwedge_{\ell' \in L_B} \Big(\big(A \xrightarrow{X} \ell \wedge B \xrightarrow{X} \ell'\big) \implies \ell(X) = \ell'(X) \Big), \qquad (*)$$

where $A \xrightarrow{X} \ell$ and $B \xrightarrow{X} \ell'$ are written as above. If some of the primitive label sets are discrete sets, then the clauses $\ell(X) = \ell'(X)$ can be simplified, possibly to a single boolean value.

Suppose A and B compute the same function over the reals, and therefore, by construction, the elementary formula $(*)$ holds over the reals. Then Tarski's Transfer Principle [46] implies that formula $(*)$ holds over any real closed field. Moreover, since the formula is only universally quantified, it also hold over any subset of any real closed field. $\square$

Corollary A.3. *Let S be an ordered subset of a real closed field, such that $\mathbb{R} \subseteq S$. The complexity of any function over S that can be computed by an algebraic decision tree is the same as the complexity of that function restricted to the reals.*

Proof: Let A be a dth-order algebraic decision tree that computes some function $\Pi : S^n \to S^m \times \mathcal{O}$. The tree A also computes a function $\Phi : \mathbb{R}^n \to \mathbb{R}^m \times \mathcal{O}$ over the reals. Since $\mathbb{R} \subset S$, it follows that Φ is the restriction of Π to the reals, that is, $\Pi(X) = \Phi(X)$ for any vector $X \in \mathbb{R}^n$. Lemma A.2 implies that any algebraic decision tree that computes Φ also computes Π. Thus, both functions are computed by exactly the same set of algebraic decision trees. □

An important implication of this lemma is that any function over the reals can be extended uniquely to any ordered subset of any real closed field.

A.4 Infinitesimal Reductions

We are now in a position to discuss our more complicated reduction argument. Suppose we want to show that the complexity of the function $\Pi_1 : \mathbb{R}^m \to \mathcal{O}$ is asymptotically less than the complexity of the function $\Pi_2 : \mathbb{R}^n \to \mathcal{O}$, or in the looser terminology of the main body of the paper, that Π_2 is harder than Π_1. Given a real configuration $X \in \mathbb{R}^n$, we compute $\Pi_1(X)$ by (implicitly) transforming X into another configuration $Y \in \mathbb{R}[\varepsilon]^m$ and computing $\Phi_2(Y)$, where Φ_2 is the unique extension of Π_2 to $\mathbb{R}[\varepsilon]$. Since the complexity of the two functions Π_2 and Φ_2 are equal, the complexity of Π_1 is at most the complexity of Π_2 plus the reduction time.

The only problem with this argument is that algebraic decision trees cannot compute functions from $\mathbb{R}^n$ to $\mathbb{R}[\varepsilon]^m$. Therefore, in order to make the reduction work, we must *simulate* the introduction of infinitesimals.

Let $\mathbb{R}[\varepsilon^{\leq k}] \subset \mathbb{R}[\varepsilon]$ denote the set of real polynomials in ε with degree $\leq k$, ordered consistently with the polynomial ring $\mathbb{R}[\varepsilon]$. As an ordered set, $\mathbb{R}[\varepsilon^{\leq k}]$ is isomorphic to the lexicographically ordered vector space $\mathbb{R}^{k+1}$. Thus, given any function of the form

$$\Pi : \mathbb{R}[\varepsilon^{\leq k}]^n \to \mathbb{R}[\varepsilon^{\leq dk}]^m \times \mathcal{O},$$

we easily define an equivalent function over the reals:

$$\Pi^{[k]} : \mathbb{R}^{(k+1)n} \to \mathbb{R}^{(dk+1)m} \times \mathcal{O}.$$

Lemma A.4. *Let A be a dth-order algebraic decision tree that computes a function Π over $\mathbb{R}[\varepsilon^{\leq k}]$. There exists a dth-order algebraic decision tree $A^{[k]}$ that computes the function $\Pi^{[k]}$ and whose depth is at most $dk+1$ times the depth of A.*

Proof: We construct $A^{[k]}$ by replacing the real arithmetic in A by polynomial arithmetic. Let $X \in \mathbb{R}[\varepsilon^{\leq k}]$ be a typical input to A. Any query polynomial $q_v(X)$ can be rewritten as a polynomial in ε of degree at most $dk+1$, and the sign of $q_v(X)$ is given by the sign of the lowest-degree non-zero term of this polynomial. To construct $A^{[k]}$, we replace each node v in A with a sequence of up to $dk+1$ nodes, which compute the terms of $q_v(X)$ in increasing order of degree. We modify the leaf labels similarly. □

Finally, we formalize our infinitesimal reduction. As before, we want to show that the complexity $T_1(n)$ of the function $\Pi_1 : \mathbb{R}^n \to \mathcal{O}$ is asymptotically less than the complexity $T_2(m)$ of the function $\Pi_2 : \mathbb{R}^m \to \mathcal{O}$. Given a real configuration $X \in \mathbb{R}^n$, we compute $\Pi_1(X)$ by (implicitly) transforming X into another configuration $Y^{[k]} \in \mathbb{R}^{(k+1)m} \cong \mathbb{R}[\varepsilon^{\leq k}]^n$ and then computing $\Pi_2^{[k]}(Y^{[k]})$. The reduction gives us an inequality of the form

$$T_1(n) \leq T_{1\to 2}(n) + T_2^{[k]}(m) = T_{1\to 2}(n) + \mathrm{O}(k) \cdot T_2(m),$$

where $T_{1\to 2}(n)$ is the complexity of the transformation $X \mapsto Y^{[k]}$ and $T_2^{[k]}(m)$ is the complexity of $\Pi_2^{[k]}$. Provided $T_{1\to 2}(n)$ is dominated by the other term, k is a constant, and $m = \mathrm{O}(n)$, we have the desired inequality

$$T_1(n) = \mathrm{O}(T_2(n)).$$

A.5 Other Models of Real Computation

Results similar to Lemma A.2 and Lemma A.3 can be also proven in the algebraic computation tree [5] and real RAM [41] models of computation. However, since in these models, algorithms can compute polynomials of exponentially high degree in linear time, the simple simulation outlined in Lemma A.4 might increase the running time of an algorithm exponentially.

References

[1] P. Agarwal and J. Matoušek. Personal communication, reported in [3], 1991.

[2] P. K. Agarwal, B. Aronov, M. Sharir, and S. Suri. Selecting distances in the plane. *Algorithmica*, 9:495–514, 1993.

[3] P. K. Agarwal, H. Edelsbrunner, O. Schwarzkopf, and E. Welzl. Euclidean minimum spanning trees and bichromatic closest pairs. *Discrete Comput. Geom.*, 6(5):407–422, 1991.

[4] P. K. Agarwal and J. Matoušek. Ray shooting and parametric search. *SIAM J. Comput.*, 22(4):794–806, 1993.

[5] M. Ben-Or. Lower bounds for algebraic computation trees. In *Proc. 15th Annu. ACM Sympos. Theory Comput.*, pages 80–86, 1983.

[6] M. de Berg, M. Overmars, and O. Schwarzkopf. Computing and verifying depth orders. In *Proc. 8th Annu. ACM Sympos. Comput. Geom.*, pages 138–145, 1992.

[7] S. Bloch, J. Buss, and J. Goldsmith. How hard are n^2-hard problems? *SIGACT News*, 25(2):83–85, 1994.

[8] J. Bochnak, M. Coste, and M.-F. Roy. *Géométrie algébraique réelle*. Number 12 in Ergebnisse der Mathematik und ihrer Grenzgebeite 3. Springer-Verlag, 1987.

[9] O. Borůvka. O jistém problému minimálím. *Práca Moravské Přírodovědecké Společnosti*, 3:37–58, 1926.

[10] J. Canny. Some algebraic and geometric configurations in PSPACE. In *Proc. 20th Annu. ACM Sympos. Theory Comput.*, pages 460–467, 1988.

[11] J. Canny. Computing roadmaps of semi-algebraic sets. In *Proc. 9th Annu. Sympos. Algebraic Algorithms and Error Corr. Codes*, pages 94–107, 1991.

[12] T. M. Y. Chan. Output-sensitive results on convex hulls, extreme points, and related problems. In *Proc. 11th Annu. ACM Sympos. Comput. Geom.*, pages 10–19, 1995.

[13] B. Chazelle. Lower bounds on the complexity of polytope range searching. *J. Amer. Math. Soc.*, 2:637–666, 1989.

[14] B. Chazelle. Lower bounds for off-line range searching. In *Proc. 27th Annu. ACM Sympos. Theory Comput.*, 1995. To appear.

[15] B. Chazelle, H. Edelsbrunner, L. Guibas, and M. Sharir. Diameter, width, closest line pair and parametric searching. *Discrete Comput. Geom.*, 10:183–196, 1993.

[16] B. Chazelle, H. Edelsbrunner, L. Guibas, and M. Sharir. Algorithms for bichromatic line segment problems and polyhedral terrains. *Algorithmica*, 11:116–132, 1994.

[17] B. Chazelle, H. Edelsbrunner, L. J. Guibas, R. Pollack, R. Seidel, M. Sharir, and J. Snoeyink. Counting and cutting cycles of lines and rods in space. In *Proc. 31st Annu. IEEE Sympos. Found. Comput. Sci.*, pages 242–251, 1990.

[18] K. L. Clarkson and P. W. Shor. Applications of random sampling in computational geometry, II. *Discrete Comput. Geom.*, 4:387–421, 1989.

[19] R. Cole. Slowing down sorting networks to obtain faster sorting algorithms. *J. ACM*, 34:200–208, 1987.

[20] H. Edelsbrunner, L. J. Guibas, and J. Stolfi. Optimal point location in a monotone subdivision. *SIAM J. Comput.*, 15:317–340, 1986.

[21] H. Edelsbrunner and E. P. Mücke. Simulation of simplicity: a technique to cope with degenerate cases in geometric algorithms. *ACM Trans. Graph.*, 9:66–104, 1990.

[22] I. Emiris and J. Canny. A general approach to removing degeneracies. In *Proc. 32nd Annu. IEEE Sympos. Found. Comput. Sci.*, pages 405–413, 1991.

[23] J. Erickson. Lower bounds for linear satisfiability problems. In *Proc. 6th Annu. ACM-SIAM Sympos. Discrete Algorithms*, pages 388–395, 1995.

[24] J. Erickson. New lower bounds for Hopcroft's problem. In *Proc. 11th Annu. ACM Sympos. Comput. Geom.*, pages 127–137, 1995.

[25] J. Erickson. On the relative complexities of some geometric problems. In *Proc. 7th Canad. Conf. Comput. Geom.*, pages 85–90, 1995.

[26] M. L. Fredman. How good is the information theory bound in sorting? *Theoret. Comput. Sci.*, 1:355–361, 1976.

[27] M. L. Fredman. Lower bounds on the complexity of some optimal data structures. *SIAM J. Comput.*, 10:1–10, 1981.

[28] A. Gajentaan and M. Overmars. On a class of $O(n^2)$ problems in computational geometry. *Comput. Geom. Theory Appl.*, to appear.

[29] A. Gajentaan and M. H. Overmars. n^2-hard problems in computational geometry. Report RUU-CS-93-15, Dept. Comput. Sci., Utrecht Univ., Utrecht, Netherlands, Apr. 1993.

[30] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman, New York, NY, 1979.

[31] N. Katoh and K. Iwano. Finding k farthest pairs and k closest pairs/farthest bichromatic pairs for points in the plane. In *Proc. 8th Annu. ACM Sympos. Comput. Geom.*, pages 320–329, 1992.

[32] M. J. Katz and M. Sharir. An expander-based approach to geometric optimization. In *Proc. 9th Annu. ACM Sympos. Comput. Geom.*, pages 198–207, 1993.

[33] J. Matoušek. Geometric range searching. *ACM Comput. Surv.*, 26: 421–461, 1994.

[34] J. Matoušek and O. Schwarzkopf. On ray shooting in convex polytopes. *Discrete Comput. Geom.*, 10(2):215–232, 1993.

[35] J. Matoušek. Linear optimization queries. *J. Algorithms*, 14:432–448, 1993. The results combined with results of O. Schwarzkopf also appear in *Proc. 8th ACM Sympos. Comput. Geom.*, 1992, pages 16–25.

[36] J. Matoušek. Range searching with efficient hierarchical cuttings. *Discrete Comput. Geom.*, 10(2):157–182, 1993.

[37] N. Megiddo. Applying parallel computation algorithms in the design of serial algorithms. *J. ACM*, 30:852–865, 1983.

[38] F. Meyer auf der Heide. A polynomial time linear search algorithm for the n-dimensional knapsack problem. *J. ACM*, 31:668–676, 1984.

[39] M. Pellegrini. Incidence and nearest-neighbor problems for lines in 3-space. In *Proc. 8th Annu. ACM Sympos. Comput. Geom.*, pages 130–137, 1992.

[40] M. Pellegrini. Ray shooting on triangles in 3-space. *Algorithmica*, 9:471–494, 1993.

[41] F. P. Preparata and M. I. Shamos. *Computational Geometry: An Introduction*. Springer-Verlag, New York, NY, 1985.

[42] A. Prestel. *Lectures on Formally Real Fields*, volume 1093 of *Lecture Notes in Mathematics*. Springer-Verlag, 1984.

[43] J. M. Steele and A. C. Yao. Lower bounds for algebraic decision trees. *J. Algorithms*, 3:1–8, 1982.

[44] J. Stolfi. *Oriented Projective Geometry: A Framework for Geometric Computations*. Academic Press, New York, NY, 1991.

[45] E. Szemerédi and W. T. Trotter, Jr. Extremal problems in discrete geometry. *Combinatorica*, 3:381–392, 1983.

[46] R. E. Tarjan. *Data Structures and Network Algorithms*, volume 44 of *CBMS-NSF Regional Conference Series in Applied Mathematics*. Society for Industrial Applied Mathematics, 1983.

[47] A. C. Yao. On constructing minimum spanning trees in k-dimensional spaces and related problems. *SIAM J. Comput.*, 11:721–736, 1982.

[48] C. K. Yap. A geometric consistency theorem for a symbolic perturbation scheme. *J. Comput. Syst. Sci.*, 40:2–18, 1990.