SPACE-TIME TRADEOFFS FOR EMPTINESS QUERIES*

JEFF ERICKSON†

Abstract. We develop the first nontrivial lower bounds on the complexity of online hyperplane and halfspace emptiness queries. Our lower bounds apply to a general class of geometric range query data structures called *partition graphs*. Informally, a partition graph is a directed acyclic graph that describes a recursive decomposition of space. We show that any partition graph that supports hyperplane emptiness queries implicitly defines a halfspace range query data structure in the Fredman/Yao semigroup arithmetic model, with the same asymptotic space and time bounds. Thus, results of Brönnimann, Chazelle, and Pach imply that any partition graph of size s that supports hyperplane emptiness queries in time t satisfies the inequality $st^d = \Omega((n/\log n)^{d-(d-1)/(d+1)})$. Using different techniques, we improve previous lower bounds for Hopcroft's problem—Given a set of points and hyperplanes, does any hyperplane contain a point?—in dimensions four and higher. Using this offline result, we show that for online hyperplane emptiness queries, $\Omega(n^d/\operatorname{polylog} n)$ space is required to achieve polylogarithmic query time, and $\Omega(n^{(d-1)/d}/\operatorname{polylog} n)$ query time is required if only $O(n \operatorname{polylog} n)$ space is available. These two lower bounds are optimal up to polylogarithmic factors. For two-dimensional queries, we obtain an optimal continuous tradeoff $st^2 = \Omega(n^2)$ between these two extremes. Finally, using a lifting argument, we show that the same lower bounds hold for both offline and online halfspace emptiness queries in $\mathbb{R}^{d(d+3)/2}$.

Key words. lower bounds, range searching, space-time tradeoff, partition graph

AMS subject classifications. 68Q25 (68U05)

1. Introduction.

A geometric range searching data structure stores a finite set of points so that we can quickly compute some function of the points inside an arbitrary region of space, or *query range*. For example, a reporting query asks for the list of points in the query range, and a counting query asks for their number. Perhaps the simplest type of query is an *emptiness* query (also called an *existential* query [6, 45]), which asks whether the query range contains any points in the set. Emptiness query data structures have been used to solve several geometric problems, including point location [20], ray shooting [2, 20, 41, 44], nearest and farthest neighbor queries [2], linear programming queries [40, 11], depth ordering [25], collision detection [19], and output-sensitive convex hull construction [40, 12].

This paper presents the first nontrivial lower bounds on the complexity of data structures that support online emptiness queries, where the query ranges are either arbitrary hyperplanes or arbitrary halfspaces. Most of our results take the form of tradeoffs between space and query time; that is, we prove lower bounds on the size of the data structure as a function of its worst-case query time, or vice versa. We also prove tradeoffs between preprocessing time and query time. These are the first such lower bounds for any range searching problem in any model of computation; earlier models do not even define preprocessing time.

1.1. Previous Results.

Most geometric range searching lower bounds are presented in the Fredman/Yao *semigroup arithmetic* model [33, 56]. In this model, the

*This research was done at the Computer Science Division, U. C. Berkeley, and at the Center for Geometric Computing, Department of Computer Science, Duke University, with the support of a Graduate Assistance in Areas of National Need fellowship, NSF grant DMS-9627683, and U. S. Army Research Office MURI grant DAAH04-96-1-0013. Portions of this paper were presented at the 37th IEEE Symposium on Foundations of Computer Science [30] and at the 13th ACM Symposium on Computational Geometry [32].

†Department of Computer Science, University of Illinois, Urbana-Champaign; jeffe@cs.uiuc.edu; http://www.uiuc.edu/~jeffe

points are given weights from a semigroup, and the goal of a range query is to determine the total weight of the points in a query region. A data structure in this model can be informally regarded as a set of precomputed partial sums in the underlying semigroup. The size of such a data structure is the number of partial sums, and the query time is the minimum number of semigroup additions performed on these partial sums to obtain the required answer. (We define the model more formally in Section 2.) Lower bounds have been established in this model for several types of query ranges [10, 14, 16, 56], in many cases matching the complexities of the corresponding data structures, at least up to polylogarithmic factors.

Unfortunately, emptiness queries are completely trivial in the semigroup arithmetic model. If the query range is empty, we perform no additions; conversely, if we perform even a single addition, the query range must not be empty. Similar arguments apply to Tarjan's *pointer machine* model [54], which has been used to derive output-sensitive lower bounds for several types of reporting queries [15, 22]. In fact, the only lower bounds previously known for hyperplane emptiness queries are essentially trivial. The size of any range searching data structure must be $\Omega(n)$, since it must store each of the points. The time to answer any range query must be at least $\Omega(\log n)$, even for a fixed one-dimensional point set, in any reasonable model of computation such as algebraic decision trees [50], algebraic computation trees [8], pointer machines [54], or real RAMs [49].[1] Since there are both linear-size data structures (with large query times) [38] and data structures with logarithmic query time (with large space requirements) [17, 42], any better lower bound must take the form of a tradeoff between space and time.

The only nontrivial lower bound previously known for *any* class of online emptiness queries, in *any* model of computation, is due to Anderson and Swanson [6]. They show that $\Omega(n \log n / \log t)$ space is required to answer axis-aligned rectangular emptiness queries in time t, in the so-called layered partition model. In particular, $\Omega(n \log n / \log \log n)$ space is required to achieve polylogarithmic query time in this model (but see [13] for better upper bounds in the integer RAM model).

Very recently, Borodin *et al.* [9] derived lower bounds for hyperplane and halfspace emptiness, nearest-neighbor, point-location, and related queries in high-dimensional spaces, in Yao's extremely general *cell probe* model [57]. (See also [46].) In the cell probe model, a data structure is an array of s cells, each containing b bits. A query is answered by probing t of these cells in sequence; the address of each probe may depend arbitrarily on the query and the results of previous probes. Borodin *et al.* show that for hyperplane queries among n points in the d-dimensional Hamming cube $\{0,1\}^d$, either $t \log s = \Omega(\log n \log d)$ or $tb = \Omega(n^{1-\varepsilon})$ for any fixed $\varepsilon > 0$. Unfortunately, this lower bound is trivial for any fixed dimension d, since it requires $n \ll 2^d$. Even when the dimension is allowed to vary, the bound is extremely weak unless the number of probes t is nearly constant.

1.2. New Results. We derive our new lower bounds with respect to a general class of geometric range query data structures called *partition graphs*. Informally, a partition graph is a directed acyclic graph that describes a recursive decomposition of space into connected regions. This recursive decomposition provides a natural search

[1] Sublogarithmic or even constant query times can be obtained for axis-aligned rectangular queries in models of computation that allow bit manipulation and require integer inputs within a known bounded universe; see, for example, [4, 5, 13, 46, 47, 55]. No such result is known for non-orthogonal ranges, however. We will take the traditional computational-geometric view that geometric objects are represented by arbitrary real coordinates, for which bit manipulation is impossible.

TABLE 1
Best known upper bounds for online hyperplane emptiness queries.

Space	Preprocessing	Query Time	Source
$O(n^d/\log^d n)$	$O(n^d/\log^{d-\varepsilon} n)$	$O(\log n)$	[17, 42]
$O(n)$	$O(n^{1+\varepsilon})$	$O(n^{1-1/d})$	[42]
$O(n)$	$O(n \log n)$	$O(n^{1-1/d} \operatorname{polylog} n)$	[38]
$n \le s \le n^d/\log^d n$	$O(n^{1+\varepsilon} + s \log^{\varepsilon} n)$	$O(n/s^{1/d})$	[17, 42]

structure that is used both to preprocess points and to answer queries. (A formal definition is provided in Section 3.) Our model is powerful enough to describe most, if not all, known hyperplane range searching data structures.[2] Partition graphs were originally introduced to study the complexity of Hopcroft's problem—Given a set of points and hyperplanes, does any hyperplane contain a point?—and similar offline geometric searching problems [31].

We summarize our results below. In each of these results and throughout the paper, s denotes space, p denotes preprocessing time, and t denotes worst-case query time. For comparison, the best known upper bounds are listed in Table 1. For a thorough overview of range searching techniques, results, and applications, see the surveys by Matoušek [43] and by Agarwal and Erickson [1].

- Any partition graph that supports hyperplane queries requires $\Omega(n)$ space, $\Omega(n \log n)$ preprocessing time, and $\Omega(\log n)$ query time.
- Any partition graph that supports hyperplane emptiness queries implicitly defines a halfspace range query data structure in the Fredman/Yao semigroup arithmetic model, with the same time and space bounds. Thus, results of Brönnimann, Chazelle, and Pach [10] immediately imply that $st^d = \Omega((n/\log n)^{d-(d-1)/(d+1)})$. This lower bound applies with high probability to a randomly generated set of points. This is the first nontrivial lower bound for hyperplane emptiness queries in any model of computation.
- We generalize earlier lower bounds on the complexity of Hopcroft's problem [31] for the special case of *polyhedral* partition graphs. Specifically, we prove that in the worst case, the time to preprocess n points in $\mathbb{R}^d$ and perform k hyperplane emptiness queries is

  $$\Omega(n \log k + n^{1-2/d(d+1)} k^{2/(d+1)} + n^{2/(d+1)} k^{1-2/d(d+1)} + k \log n),$$

  even if the query hyperplanes are specified in advance. This lower bound was previously known in dimensions less than four, and in arbitrary dimensions for offline counting and reporting queries, for arbitrary partition graphs [31].
- The previous result implies the worst-case tradeoffs $pt^{(d+2)(d-1)/2} = \Omega(n^d)$ and $pt^{2/(d-1)} = \Omega(n^{(d+2)/d})$. These lower bounds match known upper bounds up to polylogarithmic factors when $d = 2$, $p = O(n \operatorname{polylog} n)$, or $t = O(\operatorname{polylog} n)$ [38, 42]. These results apply to polyhedral partition graphs when $d \ge 4$ and to all partition graphs when $d \le 3$. Under a mild assumption about the partition graphs, these bounds imply similar tradeoffs between space and query time, improving the earlier space-time tradeoffs whenever $s = \Omega(n^{d-1})$ or $s = O(n^{1+2/(d^2+d)})$ and giving us the optimal lower bound $st^2 = \Omega(n^2)$ in two dimensions.

[2] Difficulties in directly modeling existing range searching data structures as partition graphs are discussed in [31, Section 3.5].

- Finally, using a lifting argument, we show that all of these lower bounds also apply to online and offline halfspace emptiness queries in $\mathbb{R}^{d(d+3)/2}$, at least for *semialgebraic* partition graphs. The lower bounds we obtain match existing upper bounds up to polylogarithmic factors in dimensions 2, 3, and 5. In most other cases, our bounds are extremely weak; nevertheless, they are an improvement over all previous (trivial) lower bounds for halfspace emptiness searching.

All our lower bounds for hyperplane emptiness queries in $\mathbb{R}^d$ also apply to hyperplane and halfspace counting, reporting, or semigroup queries in $\mathbb{R}^d$.

We also show that lower bounds in the semigroup arithmetic model imply lower bounds for the corresponding counting or reporting queries in the partition graph model. Thus, any partition graph supporting halfspace counting or reporting queries satisfies $st^d = \Omega((n/\log n)^{d-(d-1)/(d+1)})$, even in the average case. We also derive the worst-case lower bound $st^{d(d+1)/2} = \Omega(n^d)$ for hyperplane queries in the semigroup model (no lower bound was previously known in this model), and thus for hyperplane counting or reporting in the partition graph model as well. Surprisingly, in two dimensions, the lower bound $st^3 = \Omega(n^2)$ is tight in the semigroup model.

From a practical standpoint, our results are quite strong. Even for very simple query ranges, and even if we only want to know whether the range is empty, range searching algorithms based on geometric divide-and-conquer techniques cannot be significantly faster than the naïve linear-time algorithm that simply checks each point individually, unless the dimension is very small or we have almost unlimited storage.[3] For example, answering 10-dimensional hyperplane emptiness queries in, say, $\sqrt{n}$ time—quite far from the desired $O(\log n)$ time bound—requires $\Omega(n^4)$ space, which is simply impossible for large data sets. In practice, we can only afford to use linear space, and this drives the worst-case query time up to roughly $n^{9/10}$. This behavior is unfortunately not a feature of some pathological input; most of our space-time tradeoffs hold in the average case, so in fact, most point sets are this difficult to search. The lower bounds that do not (as far as we know) hold in the average case apply to extremely simple point sets, such as regular lattices.

1.3. Outline. The rest of the paper is organized as follows. In Section 2, we review the definition of the semigroup arithmetic model, state a few useful results, and derive new bounds on the complexity of hyperplane queries in this model. Section 3 defines partition graphs, describes how they are used to answer hyperplane and halfspace queries, and states a few of their basic properties. We prove our new space-time tradeoff lower bounds for hyperplane emptiness queries in Section 4. In Section 5, we define *polyhedral covers* and develop bounds on their worst-case complexity. Using these combinatorial bounds, in Section 6, we (slightly) improve earlier lower bounds on the complexity of Hopcroft's offline point-hyperplane incidence problem in dimensions four and higher. From these offline results, we derive new tradeoffs between preprocessing and query time for online hyperplane queries in Section 7. Section 8 describes a reduction argument that implies lower bounds for halfspace emptiness queries in both the online and offline settings. Finally, in Section 9, we offer our conclusions.

2. Semigroup Arithmetic.

[3] This "curse of dimensionality" can sometimes be avoided by requiring only an approximation of the correct output; see, for example, [7, 24, 37].

2.1. Definitions. We begin by reviewing the definition of the semigroup arithmetic model, originally introduced by Fredman to study dynamic range searching problems [33], and later refined for the static setting by Yao [56].

A *semigroup* $(S, +)$ is a set S equipped with an associative addition operator $+ : S \times S \to S$. A semigroup is *commutative* if the equation $x + y = y + x$ is true for all $x, y \in S$. A *linear form* is a sum of variables over the semigroup, where each variable can occur multiple times, or equivalently, a homogeneous linear polynomial with positive integer coefficients. A commutative semigroup is *faithful* if any two identically equal linear forms have the same set of variables, although not necessarily with the same set of coefficients.[4] For example, $(\mathbb{Z}, +)$ and $(\{\text{true}, \text{false}\}, \vee)$ are faithful semigroups, but $(\{0, 1\}, + \bmod 2)$ is not faithful.

A semigroup is *idempotent* if $x + x = x$ for all semigroup elements x, and *integral* if $x \neq \alpha x$ for all semigroup elements x and all integers $\alpha > 1$. For example, the semigroups $(\{\text{true}, \text{false}\}, \vee)$ and $(\mathbb{Z}, \max)$ are idempotent, $(\mathbb{Z}, +)$ and $(\mathbb{R}, \times)$ are integral, and $(\mathbb{C}, \times)$ is neither. (All these semigroups are faithful.)

Let P be a set of n points in $\mathbb{R}^d$, let $(S, +)$ be a faithful commutative semigroup, and let $w : P \to S$ be a function that assigns a weight $w(p)$ to each point $p \in P$. For any subset $P' \subseteq P$, let $w(P') = \sum_{p \in P'} w(P)$, where addition is taken over the semigroup.[5] The range searching problem considered in the semigroup model is to preprocess P so that $w(P \cap q)$ can be calculated quickly for any query range q.

Let $x_1, x_2, \ldots, x_n$ be a set of n variables over S. A *generator* $g(x_1, \ldots, x_n)$ is a linear form $\sum_{i=1}^{n} \alpha_i x_i$, where the α_i's are non-negative integers, not all zero. Given a class Q of query ranges (subsets of $\mathbb{R}^d$), a *storage scheme* for (P, Q, S) is a collection of generators $\{g_1, g_2, \ldots, g_s\}$ with the following property: For any query range $q \in Q$, there is an set of indices $I_q \subseteq \{1, 2, \ldots, s\}$ and an indexed set of non-negative integers $\{\beta_i \mid i \in I_q\}$ such that

$$w(P \cap q) = \sum_{i \in I_q} \beta_i g_i(w(p_1), w(p_2), \ldots, w(p_n))$$

holds for *any* weight function $w : P \to S$. The size of the smallest such set I_q is the *query time* for q.

We emphasize that although a storage scheme can take advantage of special properties of the semigroup S or the point set P, it must work for *any* assignment of weights to P. In particular, this implies that lower bounds in the semigroup model do *not* apply to the problem of counting the number of points in the query range, even though $(\mathbb{Z}, +)$ is a faithful semigroup, since a storage scheme for that problem only needs to work for the particular weight function $w(p) = 1$ for all $p \in P$ [14]. For the same reason, even though the semigroups $(\{\text{true}, \text{false}\}, \vee)$ and $(2^P, \cup)$ are faithful, the semigroup model cannot be used to prove lower bounds for emptiness or reporting queries. Emptiness queries can also be formulated as queries over the one-element semigroup $(\{*\}, * + * = *)$, but this semigroup is not faithful.

[4] More formally, $(S, +)$ is faithful if for each $n > 0$, for any sets of indices $I, J \subseteq \{1, \ldots, n\}$ where $I \neq J$, and for all indexed sets of positive integers $\{\alpha_i \mid i \in I\}$ and $\{\beta_j \mid j \in J\}$, there are semigroup elements $s_1, s_2, \ldots, s_n \in S$ such that

$$\sum_{i \in I} \alpha_i s_i \neq \sum_{j \in J} \beta_j s_j.$$

[5] Since S need not have an identity element, we may need to assign a special value *nil* to the empty sum $w(\varnothing)$.

For any linear form $\sum_{1=1}^{n} \alpha_i x_i$, we call the set of points $\{p_i \mid \alpha_i \neq 0\}$ its *cluster*. The faithfulness of the semigroup S implies that the union of the clusters of the generators used to determine $w(P \cap q)$ is precisely $P \cap q$ (since each $\alpha_i > 0$):

$$\bigcup_{i \in I_q} \operatorname{cluster}(g_i) = P \cap q.$$

Thus, we can think of a storage scheme as a collection of clusters, such that any set of the form $P \cap q$ can be expressed as the (not necessarily disjoint) union of several of these clusters. The size of a storage scheme is the number of clusters, and the query time for a range q is the minimum number of clusters whose union is $P \cap q$. This is the formulation actually used to prove lower bounds in the semigroup arithmetic model[6] [10, 14, 16, 56].

Whether or not the clusters used to answer a query must be disjoint depends on the semigroup. If the semigroup is integral, the clusters must be disjoint for every query; on the other hand, if the semigroup is idempotent, clusters can overlap arbitrarily. Thus, upper bounds developed for integral semigroups and lower bounds developed for idempotent semigroups apply to all other semigroups as well.

Some of our lower bounds derive from the following result.

Theorem 2.1 (Brönnimann, Chazelle, Pach [10]). *Let P be a uniformly distributed set of points in the d-dimensional unit hypercube $[0,1]^d$. With high probability, any storage scheme of size s that supports halfspace queries over P in time t satisfies the inequality $st^d = \Omega((n/\log n)^{d-(d-1)/(d+1)})$.*

2.2. Unreasonably Good Bounds for Hyperplane Queries. Although lower bounds are known for offline hyperplane searching in the semigroup model [18, 31], we are unaware of any previous results for online hyperplane queries. In particular, Chazelle's lower bound $st^d = \Omega(n^d/\log^d n)$ for simplex range searching [14], which holds when the query ranges are slabs bounded by two parallel hyperplanes, does not apply when the ranges are hyperplanes; Chazelle's proof requires a positive lower bound on the width of the slabs.

We easily observe that for any set of points in general position, the smallest possible storage scheme, consisting of n singleton sets, allows hyperplane queries to be answered correctly in constant "time". We can obtain better lower bounds by considering degenerate point sets, as follows.

First consider the two-dimensional case. Let $\mathcal{C}$ be (the set of clusters associated with) an optimal storage scheme of size s that supports line queries for some n-point set P in the plane. The storage scheme $\mathcal{C}$ must contain n singleton sets, one containing each point in P. Without loss of generality, each of the other $s - n$ clusters in $\mathcal{C}$ is a maximal colinear subset of P; that is, each has the form $P \cap \ell$ for some line ℓ. (If a cluster contains three non-colinear points, it can be discarded. If a cluster contains at least two points on a line ℓ, but not every point on ℓ, then adding the missing points decreases the query time for ℓ without changing the number of clusters or the query time for any other line.) Thus, the query time for any line ℓ is 1 if $P \cap \ell \in \mathcal{C}$, and $|P \cap \ell|$ otherwise. It follows that an optimal storage scheme of size s consists of n singleton sets plus the $s-n$ largest maximal colinear subsets of P, and the worst-case query time is the size of the $(s - n + 1)$th largest maximal colinear subset of P.

Theorem 2.2. *A storage scheme of size $s \geq 2n$ that supports line queries in time t satisfies the inequality $st^3 = \Omega(n^2)$ in the worst case.*

[6]despite the complete absence of both semigroups and arithmetic!

Proof. Let P be a $\sqrt{n} \times \sqrt{n}$ integer lattice of points. Erdős showed that for any integer k, there is a set of k lines L such that the number of point-line incidences between P and L is $\Omega(n^{2/3}k^{2/3})$; see [33] or [48, p. 177]. In particular, we can take L to be the lines containing the k largest colinear subsets of P. Erdős's lower bound implies that the kth largest maximal colinear subset of P contains at least $\Omega(n^{2/3}/k^{1/3})$ points. Thus, the worst-case query time for any storage scheme of size s is $\Omega(n^{2/3}/(s-n+1)^{1/3}) = \Omega(n^{2/3}/s^{1/3})$. ☐

Surprisingly, this lower bound is optimal for most values of s.

THEOREM 2.3. *For any set P of n points in the plane and any integer s with $2n \le s \le n^2$, there is a storage scheme of size s that supports line queries for P in time t, where $st^3 = O(n^2)$.*

Proof. Szemerédi and Trotter [53] proved that there are at most $O(n+n^{2/3}k^{2/3}+k)$ incidences between any set of n points and any set of k lines. (See also [19, 52].) Thus, for any k in both $\Omega(\sqrt{n})$ and $O(n^2)$, the kth largest maximal collinear subset of any n-point set has at most $O(n^{2/3}/k^{1/3})$ elements. The theorem follows by setting $k = s - n + 1$. ☐

In order to derive bounds in higher dimensions, we focus our attention on *restricted* point sets [35], in which any d points lie on a unique hyperplane. The optimal storage scheme for a restricted point set again consists of n singleton sets plus the $s-n$ largest subsets of the form $P \cap h$ for some hyperplane h, and the worst-case query time is the size of the $(s-n+1)$th largest subset of the form $P \cap h$. Of course, lower bounds for restricted point sets also apply to the general case, but the upper bounds do not similarly generalize.

Given a set P of points and a set H of hyperplanes, let $I(P,H)$ denote the number of incidences between P and H, that is, point-hyperplane pairs $(p,h) \in P \times H$ such that $p \in h$. Our higher-dimensional lower bounds, both here and later in the paper, use the following generalization of the Erdős point-line construction.

LEMMA 2.4 (Erickson [31]). *For any integers n and k with $n > \lfloor k^{1/d} \rfloor$, there is a restricted set P of n points and a set H of k hyperplanes in $\mathbb{R}^d$, such that $I(P,H) = \Omega(n^{2/(d+1)}k^{1-2/d(d+1)})$.*

THEOREM 2.5. *A storage scheme of size s that supports d-dimensional hyperplane queries in time t satisfies the inequality $st^{d(d+1)/2} = \Omega(n^d)$ in the worst case.*

Using probabilistic counting techniques of Clarkson *et al.* [23], Guibas, Overmars, and Robert [35] prove that for any restricted set P of n points and any set H of k hyperplanes, $I(P,H) = O(n+n^{d/(2d-1)}k^{(2d-2)/(2d-1)}+k)$. The following upper bound follows immediately from their result.

THEOREM 2.6. *For any restricted set P of n points in $\mathbb{R}^d$ and any integer s such that $2n \le s \le n^d$, there is a storage scheme of size s that supports hyperplane queries for P in time t, where $st^{2d-1} = O(n^d)$.*

The general case is unfortunately not so straightforward. Optimal storage schemes could have clusters contained in lower-dimensional flats, in which case the query time for a hyperplane h is no longer necessarily either 1 or $|P \cap h|$. If the semigroup is idempotent, every cluster in an optimal storage scheme still has the form $P \cap h$ for some hyperplane h, but this may not be true for all semigroups. We leave further generalization of our upper bounds as an open problem.

Except when s is near n^d, our upper bounds in the semigroup model are significantly better than the best known upper bounds in more realistic models of computation. The most efficient data structure known satisfies the upper bound $st^d = O(n^d)$ [17, 42], and this is believed to be optimal, especially in light of Chazelle's simplex range searching lower bounds. We are not suggesting that this data structure can be significantly improved, but rather that the semigroup model is too powerful to permit tight lower bounds for this range searching problem. This raises the frustrating possibility that closing the existing gaps between upper and lower bounds for other types of ranges, such as halfspaces [10], will be impossible unless more realistic computational models are considered.

In the remainder of this paper, we derive tighter lower bounds by considering a model that more accurately describes the behavior of geometric range searching algorithms.

3. Partition Graphs.

3.1. Definitions. A *partition graph* is a directed acyclic (multi-)graph, with one source, called the *root*, and several sinks, called *leaves*. Associated with each non-leaf node v is a set $\mathcal{R}_v$ of *query regions*, satisfying three conditions.

1. $\mathcal{R}_v$ contains at most Δ query regions, for some constant $\Delta \geq 2$.
2. Every query region is a connected subset of $\mathbb{R}^d$.
3. The union of the query regions in $\mathcal{R}_v$ is $\mathbb{R}^d$.

We associate an outgoing edge of v with each query region in $\mathcal{R}_v$. Thus, the outdegree of the graph is at most Δ. The indegree can be arbitrarily large. In addition, every internal node v is labeled either a *primal node* or a *dual node*, depending on whether its query regions $\mathcal{R}_v$ are interpreted as a partition of primal or dual space. The query regions associated with primal (resp. dual) nodes are called primal (resp. dual) query regions.

We do not require the query regions to be disjoint. In the general case, we do not require the query regions to be convex, semialgebraic, simply connected, of constant complexity, or even computable in any sense. However, a few of our results only hold for partition graphs with particular types of query regions. If all the query regions in a partition graph are constant-complexity polyhedra, we call it a *polyhedral* partition graph. If all the query regions are constant-complexity semialgebraic sets (also called *Tarski cells*), we call it a *semialgebraic* partition graph.

Given a partition graph, we preprocess a set P of points for hyperplane queries as follows. We preprocess each point $p \in P$ individually by performing a depth-first search of the partition graph, using the query regions to determine which edges to traverse. Whenever we reach a primal node v, we traverse the edges corresponding to the query regions in $\mathcal{R}_v$ that contain p. Whenever we reach a dual node v, we traverse the edges corresponding to the query regions in $\mathcal{R}_v$ that intersect the dual hyperplane p^*. The same point may enter or leave a node along several different edges, but we only test the query regions at a node once for each point. Thus, each point traverses a given edge at most once. For each leaf ℓ, we maintain a *leaf subset* P_ℓ containing the points that reach ℓ. See Figure 1(a).

To answer a hyperplane query, we use almost exactly the same algorithm as to preprocess a point: a depth-first search of the partition graph, using the query regions to determine which edges to traverse. The only difference is that the behavior at the primal and dual nodes is reversed. See Figure 1(b).

Whenever the query algorithm reaches a leaf ℓ, it *examines* the corresponding leaf subset P_ℓ. The output of the query algorithm is computed from the examined subsets, by assuming that the query hyperplane contains each examined subset. For example, the output of an emptiness query is "yes" if and only if all the examined subsets are empty. (In fact, we can assume that if the algorithm ever examines a nonempty leaf subset, it immediately halts and answers "no".) The output of a counting query is

```
Preprocess(p):

at each primal node v:
    for each query region R ∈ ℛ_v:
        if R contains p:
            traverse the edge corresponding to R
at each dual node v:
    for each query region R ∈ ℛ_v:
        if R intersects p*:
            traverse the edge corresponding to R
at each leaf ℓ:
    add p to P_ℓ
```

(a)

```
Query(h):

at each dual node v:
    for each query region R ∈ ℛ_v:
        if R contains h*:
            traverse the edge corresponding to R
at each primal node v:
    for each query region R ∈ ℛ_v:
        if R intersects h:
            traverse the edge corresponding to R
at each leaf ℓ:
    examine P_ℓ
```

(b)

FIG. 1. *Preprocessing and query algorithms for hyperplane searching.*

the sum of the sizes of the examined subsets; here, examining a leaf subset means adding its size to a running counter. The output of a reporting query is the union of the examined subsets; here, "examine" simply means "output". More generally, if the points are given weights from a (*not* necessarily faithful) semigroup $(S, +)$, the output is the semigroup sum over all examined subsets P_ℓ of the total weight of the points in P_ℓ. (We will not explicitly consider semigroup queries in the rest of the paper.)

By modifying the preprocessing algorithm slightly, we can also use partition graphs to answer halfspace queries. For any hyperplane h, let h^+ denote its closed upper halfspace and h^- its closed lower halfspace.[7] Recall that the standard duality transformation $(a_1, a_2, \ldots, a_d) \longleftrightarrow x_d + a_d = a_1x_1 + a_2x_2 + \cdots + a_{d-1}x_{d-1}$ preserves incidences and relative orientation between points and hyperplanes: If a point p is above (on, below) a hyperplane h, then the dual point h^* is above (on, below) the dual hyperplane p^*.

To support halfspace queries, we associate one or two subsets of P with every query region, called *internal subsets*. With each primal region $R \in \mathcal{R}_v$, we associate a single internal subset P_R, which contains the points that reach v and lie inside R. With each dual region $R \in \mathcal{R}_v$, we associate two internal subsets P_R^+ and P_R^-, which contain the points that reach v and whose dual hyperplanes lie below and above R, respectively. Our modified preprocessing and halfspace query algorithms are shown in Figure 2. Note that the modified preprocessing algorithm can still be used for hyperplane searching.

For purposes of proving lower bounds, the *size* of a partition graph is the number of edges in the graph, the *query time* for a particular hyperplane is the number of edges traversed by the query algorithm, and the *preprocessing time* is the total number of edge traversals during the preprocessing phase. We ignore, for example, the time required in practice to construct the graph, the complexity of the query regions, the time required to determine which query regions intersect a hyperplane or contain a point, the sizes of the subsets P_R, P_R^+, P_R^-, and P_ℓ, the time required to maintain and test these subsets, and the size of the output (in the case of reporting queries).

We emphasize that since we never charge for the construction of the partition graph itself, the graph and its query regions can depend arbitrarily on the input point

[7]We assume throughout the paper that query halfspaces are closed and that no query halfspace is bounded by a vertical hyperplane. Handling open halfspaces involves only trivial modifications to our query algorithm, which have almost no effect on our analysis. Vertical halfspace queries can be handled either through standard perturbation techniques or by using a lower-dimensional data structure.

```
Preprocess(p):
at each primal node v:
    for each query region R ∈ ℛ_v:
        if R contains p:
            add p to P_R
            traverse the edge corresponding to R
at each dual node v:
    for each query region R ∈ ℛ_v:
        if R intersects p*:
            traverse the edge corresponding to R
        else if R is above p*:
            add p to P_R^+
        else if R is below p*:
            add p to P_R^-
at each leaf ℓ:
    add p to P_ℓ
```

(a)

```
Query(h^±):
at each dual node v:
    for each query region R ∈ ℛ_v:
        if R contains h*:
            examine P_R^±
            traverse the edge corresponding to R
at each primal node v:
    for each query region R ∈ ℛ_v:
        if h intersects R:
            traverse the edge corresponding to R
        else if h^± contains R:
            examine P_R
at each leaf ℓ:
    examine P_ℓ
```

(b)

FIG. 2. *Modified preprocessing algorithm and query algorithm for halfspace searching.*

set P and on the types of queries we expect to receive. Our preprocessing algorithm has "time" to construct the *optimal* partition graph for any given input, and even very similar inputs may result in radically different partition graphs.

We will also consider offline range searching problems, where we are given both a set P of points and a set H of ranges (either hyperplanes or halfspaces), and are asked, for example, whether any range contains a point. A *partitioning algorithm* constructs a partition graph, which can depend arbitrarily on the input, preprocesses each point in P, and performs a query for each range in H. The running time of the partitioning algorithm is the sum of the preprocessing and query times. In the original definition [31], the preprocessing and queries were performed concurrently, but this has no effect on the overall running time.[8] Again, since we ignore the time required in practice to construct the partition graph, partitioning algorithms have the full power of nondeterminism.

3.2. Basic Properties. Partition graphs have several properties that are very useful in proving lower bounds.

LEMMA 3.1. *In any partition graph, if a point lies in a query range, it also lies in at least one of the subsets P_R, P_R^+, P_R^-, or P_ℓ examined during a query.*

Proof. First suppose some point p lies on some hyperplane h. Clearly, any primal query region that contains p also intersects h, and any dual query region that contains h^* also intersects p^*. Thus, there is at least one path from the root to a leaf ℓ that is traversed both while preprocessing p (so $p \in P_\ell$) and while querying h (so P_ℓ is examined).

Now suppose some point p lies in some upper halfspace h^+. (The argument for lower halfspaces is symmetric.) Let v be a node farthest from the root that is reached both while preprocessing p and while querying h^+. If v is a leaf, we are done. If v is a primal node, then some query region $R \in \mathcal{R}_v$ contains p but does not intersect h. Since $p \in R$, the subset P_R contains p, and since $p \in h^+$, R must lie above h, so P_R is examined. Finally, if v is a dual node, some query region $R \in \mathcal{R}_v$ contains the dual

[8]However, if we perform all the searches concurrently using the *streaming* technique of Edelsbrunner and Overmars [28], we only need to maintain a single root-to-leaf path in the graph at any time. Thus, the space used by an offline partitioning algorithm is more reasonably modeled by the *depth* of its partition graph. We will not pursue this idea further in this paper.

point h^* but does not intersect the dual hyperplane p^*. Since $h^* \in R$, the subset P_R^+ is examined. Since $p \in h^+$, the dual point h^*, and thus the query region R, lies above the dual hyperplane p^*, so $p \in P_R^+$. □

LEMMA 3.2. *In any partition graph, if an internal subset P_R, P_R^+, or P_R^- is examined during a halfspace query, then the query halfspace contains every point in that set.*

Proof. Suppose we are performing a query for the upper halfspace h^+. (Again, the argument for lower halfspaces is symmetric.) If the primal subset P_R is examined, then $P_R \subset R \subseteq h^+$. If the dual subset P_R^+ is examined, then every point $p \in P_R^+$ is above the hyperplane h, since the dual point $h^* \in R$ is above each dual hyperplane p^*. □

Lemma 3.1 implies that partition graphs are "conservative"—the output of a reporting query contains every point in the query range, the output of a counting query is never smaller than the actual number of points in the query range, and an emptiness query never reports that a nonempty query range is empty. Lemma 3.2 further implies that the output of a query can be incorrect only if the query algorithm examines a leaf subset that contains a point outside the query range.

The following lemma follows immediately from a close examination of the query algorithm.

LEMMA 3.3. *A counting query is correct if and only if the points in the query range are the disjoint union of the subsets examined by the query algorithm. A reporting query is correct if and only if the points in the query range are the (not necessarily disjoint) union of the subsets examined by the query algorithm.*

We say that a partition graph *supports* a particular class of online range queries for a given set of points if, after the points are preprocessed, any query in the class is answered correctly. Even though we have a single preprocessing algorithm, a single partition graph need not support all query types. However, in several cases, support for one type of query automatically implies support for another type of query, with the same (or possibly smaller) worst-case query time. These implications are summarized in the following lemma.

LEMMA 3.4. *The following hold for any partition graph.*

(a) If a counting query is answered correctly, then a reporting query for the same range is also answered correctly.

(b) If a reporting query is answered correctly, then an emptiness query for the same range is also answered correctly.

(c) For any hyperplane h, if counting (resp. reporting) queries for the halfspaces h^+ and h^- are answered correctly, then a counting (resp. reporting) query for h is also answered correctly.

(d) For any hyperplane h, if a reporting query for h is answered correctly, then reporting queries for the halfspaces h^+ and h^- are also answered correctly.

Proof. Parts (a) and (b) follow immediately from Lemma 3.3.

Fix a hyperplane h, and let $P^+ = P \cap h^+$, $P^- = P \cap h^-$, and $P^\circ = P \cap h$. Suppose reporting queries for the halfspaces h^+ and h^- are answered correctly. The reporting query algorithms for h^+, h^-, and h traverse precisely the same set of edges and reach precisely the same set of nodes. Let $\mathcal{L}$ be the set of leaves reached by any of these three queries, and let $P_{\mathcal{L}} = \bigcup_{\ell \in \mathcal{L}} P_\ell$. By Lemma 3.3, both P^+ and P^- are reported as the union of several internal subsets and $P_{\mathcal{L}}$. It follows that $P_{\mathcal{L}} \subseteq P^+ \cap P^- = P^\circ$. Lemma 3.1 implies that every point in P° lies in some leaf subset P_ℓ where $\ell \in \mathcal{L}$, so $P^\circ \subseteq P_{\mathcal{L}}$. Thus, a reporting query for h is also answered correctly. The argument for counting queries is identical, except that the examined subsets are disjoint. This proves part (c).

To prove part (d), suppose a reporting query for h is answered correctly. A reporting query for the halfspace h^+ traverses exactly the same edges and visits exactly the same nodes as the reporting query for h. In particular, the leaf subsets P_ℓ examined by the halfspace query algorithm contain only points on the hyperplane h. Lemma 3.2 implies that every internal subset examined by the reporting query algorithm lies in h^+, and Lemma 3.1 implies that every point in P^+ lies in some examined subset. It follows that precisely the points in P^+ are reported. □

3.3. "Trivial" Lower Bounds. We conclude this section by proving "trivial" lower bounds on the size, preprocessing time, and query time of any partition graph, for points in any dimension.

THEOREM 3.5. *Any partition graph that supports hyperplane emptiness queries has size $\Omega(n)$, preprocessing time $\Omega(n \log n)$, and worst-case query time $\Omega(\log n)$.*

Proof. Let P be an arbitrary set of n points in $\mathbb{R}^d$. Without loss of generality, all the points in P have distinct x_d coordinates; otherwise, rotate the coordinate system slightly. Let H be a set of n hyperplanes normal to the x_d-axis, with each hyperplane just above (farther in the x_d-direction than) one of the points in P. Thus, for all $1 \le i \le n$, there is a hyperplane in H with i points below it and $n-i$ points above it. Any partition graph that correctly answers hyperplanes queries for P must at least detect that every hyperplane in H is empty.

For each point in P, call the hyperplane in H just above it its *partner.* We say that a point is *active* at a node v of the partition graph if both the point and its partner reach v. We say that a node v *deactivates* a point p if both p and its partner h reach v but no edge out of v is traversed by both p and h. Every point in P must be deactivated by some node in the partition graph, since otherwise some active point p and its partner h would reach a common leaf, so a query for h would be answered incorrectly.

Any primal query region R contains at most one active point whose partner does not intersect R. Similarly, for any dual query region R, there is at most one active point whose dual hyperplane misses R and whose partner's dual point lies in R. Thus, any node deactivates at most Δ points. Moreover, since every point in P must be deactivated, the partition graph must have at least n query regions and thus at least n edges.

The *level* of a node is its distance from the root. There are at most Δ^k nodes at level k. At least $n - \sum_{i=0}^{k-1} \Delta^{k+1} \ge n - \Delta^{k+2}$ points are active at some node at level k. In particular, at least $n(1-1/\Delta)$ points are active at level $\lfloor \log_\Delta n - 3 \rfloor$. It follows that at least $n(1-1/\Delta)$ points in P each traverse at least $\lfloor \log_\Delta n - 2 \rfloor$ edges, so the total preprocessing time is at least

$$n(1-1/\Delta)\lfloor \log_\Delta n - 2 \rfloor = \Omega(n \log n).$$

Similarly, at least $n(1-1/\Delta)$ hyperplanes in H each traverse at least $\lfloor \log_\Delta n - 2 \rfloor$ edges, so the worst-case query time is $\Omega(\log n)$. □

Lemma 3.4 implies that the same lower bounds also apply to counting and reporting queries, both for hyperplanes and for halfspaces. Theorem 3.5 also implies that any partitioning algorithm, given n points and k hyperplanes (or halfspaces, if we are not performing emptiness queries), requires at least $\Omega(n \log k + k \log n)$ time in the worst case; this was previously proved in [31], using essentially the same argument. We will prove similar lower bounds for halfspace emptiness queries in Section 8.

4. Space-Time Tradeoffs. We now present our space-time tradeoff lower bounds for hyperplane emptiness and related queries in the partition graph model. All of these bounds are derived from results in the semigroup arithmetic model, which we described in Section 2, using the following theorem.

THEOREM 4.1. *Let P be a set of points. Given a partition graph of size s that supports hyperplane (resp. halfspace) counting or reporting queries for P in time t, we can construct a storage scheme of size $O(s)$ that supports hyperplane (resp. halfspace) queries for P, over any idempotent faithful semigroup, in time $O(t)$.*

Proof. For each query region R and leaf ℓ, define the subsets P_R, P_R^-, P_R^+, and P_ℓ by the preprocessing algorithm in Figure 2. We claim that these subsets of P form the clusters of the required storage scheme. There are at most $3s$ of these clusters: at most two for each of the s query regions, plus one for each of the $\leq s$ leaves. To prove the theorem, it suffices to show that the points in any query range can be expressed as the union of $O(t)$ clusters.

Suppose the partition graph supports hyperplane reporting queries. By Lemma 3.3, the points on any hyperplane h are reported as the union of several leaf subsets P_ℓ. Since the query algorithm reaches at most t leaves, the set $P \cap h$ is the union of at most t clusters.

Similarly, if the partition graph supports halfspace reporting queries, then the points in any halfspace $h^\pm$ are reported as the union of at most t subsets P_R, at most Δt subsets $P_R^\pm$, and at most t subsets P_ℓ. Thus, the set $P \cap h^\pm$ is the union of at most $(2 + \Delta)t = O(t)$ clusters.

The argument for counting queries is identical, except that the $O(t)$ clusters used to answer any query are disjoint. (In fact, the resulting storage scheme works for *any* faithful semigroup.) □

This theorem implies that lower bounds for range queries in the semigroup arithmetic model are also lower bounds for the corresponding counting and reporting queries in the partition graph model. The following results now immediately follow from Theorems 2.1 and 2.5.

COROLLARY 4.2. *Let P be a uniformly generated set of n points in $[0, 1]^d$. With high probability, any partition graph of size s that supports halfspace counting or reporting queries for P in time t satisfies the inequality $st^d = \Omega((n/\log n)^{d-(d-1)/(d+1)})$.*

COROLLARY 4.3. *Any partition graph of size s that supports d-dimensional hyperplane counting or reporting queries in time t satisfies the inequality $st^{d(d+1)/2} = \Omega(n^d)$ in the worst case.*

One way to determine if a hyperplane is empty is by counting or reporting the points in its two halfspaces—the hyperplane is empty if any only if every point in the original point set is counted or reported exactly once. Thus, any halfspace counting or reporting data structure also supports hyperplane emptiness queries. The following result implies that the reverse is *almost* true in our model of computation.

THEOREM 4.4. *Let P be a set of points. Given a partition graph of size s that supports hyperplane emptiness queries for P in time t, we can construct a storage scheme of size $O(s)$ that supports halfspace queries for P, over any idempotent faithful semigroup, in time $O(t)$.*

Proof. Suppose a partition graph G supports hyperplane emptiness queries for the set P in time t. Clearly, G also supports reporting queries for any *empty* hyperplane in time t, since all the examined subsets are empty. Then by Lemma 3.4 (d), G correctly answers any halfspace reporting query in time at most t, provided the boundary of the query halfspace is empty. However, for any halfspace, there is another halfspace with empty boundary that contains precisely the same points. It follows that every set of the form $P \cap h^+$ or $P \cap h^-$ can be expressed as the union of at most $(2+\Delta)t = O(t)$ internal subsets. □

The following lower bound now follows immediately from Theorem 2.1.

COROLLARY 4.5. *Let P be a uniformly generated set of n points in $[0,1]^d$. With high probability, any partition graph of size s that supports hyperplane emptiness queries for P in time t satisfies the inequality $st^d = \Omega((n/\log n)^{d-(d-1)/(d+1)})$.*

Lemma 3.4 implies that the same lower bound applies to hyperplane counting or reporting queries. This improves the lower bound in Corollary 4.3 whenever $s = O(n^{d-1})$ or $t = \Omega(n^{2/d(d+1)})$. Similarly, this lower bound also applies to halfspace counting or reporting queries, giving us a rather roundabout proof of Corollary 4.2. However, none of these results applies immediately to halfspace emptiness queries; we will derive lower bounds for these queries in Section 8.

5. Polyhedral Covers.

5.1. Definitions. In order to derive our improved offline lower bounds and preprocessing-query time tradeoffs, we first need to define a combinatorial object called a *polyhedral cover*. The formal definition is fairly technical, but intuitively, one can think of a polyhedral cover of a set P of points and a set H of hyperplanes as a collection of constant-complexity convex polytopes such that for every point $p \in P$ and hyperplane $h \in H$, some polytope in the collection contains p and does not intersect h. A polyhedral cover of P and H provides a compact representation of the relative orientation of every point in P and every hyperplane in H.

Our combinatorial bounds rely heavily on certain properties of convex polytopes and polyhedra. Many of these properties are more easily proved, and have fewer special cases, if we state and prove them in projective space rather than affine Euclidean space. In particular, developing these properties in projective space allows us to more easily deal with unbounded polyhedra, degenerate polyhedra, and duality transformations. Everything we define in this subsection can be formalized algebraically in the language of polyhedral cones and linear subspaces one dimension higher; we will give a less formal, purely geometric treatment. For more technical details, we refer the reader to the first two chapters of Ziegler's lecture notes [58], or the survey by Henk *et al.*[36].

The d-dimensional real projective space $\mathbb{RP}^d$ can be defined as the set of lines through the origin in the $(d+1)$-dimensional real vector space $\mathbb{R}^{d+1}$. Every k-dimensional linear subspace of $\mathbb{R}^{d+1}$ induces a $(k-1)$-dimensional *flat* f in $\mathbb{RP}^d$, and its orthogonal complement induces the $(d-k-1)$-dimensional *dual flat* f^*. Hyperplanes are $(d-1)$-dimensional flats, points are 0-dimensional flats, and the empty set is the unique (-1)-flat.

Any finite set H of (at least two) hyperplanes in $\mathbb{RP}^d$ defines a regular cell complex called an *arrangement*, each of whose cells is the closure of a maximal connected subset of $\mathbb{RP}^d$ contained in the intersection of a fixed subset of H and not intersecting any other hyperplane in H. The largest cells in the arrangement are the closures of the connected components of $\mathbb{RP}^d \setminus \bigcup H$; the intersection of any pair of cells is another cell of lower dimension.

A *projective polyhedron* is a single cell, not necessarily of full dimension, in an arrangement of hyperplanes in $\mathbb{RP}^d$. A *projective polytope* is a simply-connected polyhedron, or equivalently, a polyhedron that is disjoint from some hyperplane (not necessarily in the polyhedron's defining arrangement). Every projective polyhedron is (the closure of) the image of a convex polyhedron under some projective transformation, and every projective polytope is the projective image of a convex polytope. Every flat is also a projective polyhedron.

The *(projective) span* of a polyhedron $\Pi \subseteq \mathbb{RP}^d$, denoted $\mathrm{span}(\Pi)$, is the flat of minimal dimension that contains it. The *dimension* of a polyhedron is the dimension of its span. The *relative interior* (resp. *relative boundary*) of a polyhedron is its interior (resp. boundary) in the subspace topology of its span. A hyperplane *supports* a polyhedron if it intersects the polyhedron but not its relative interior. In particular, a flat has no supporting hyperplanes.

A *proper face* of a polyhedron is the intersection of the polyhedron and one or more supporting hyperplanes. Every proper face of a polyhedron is a lower-dimensional polyhedron. A *face* of a polyhedron is either a proper face or the entire polyhedron. We write $|\Pi|$ to denote the number of faces of a polyhedron Π, and $\phi \leq \Pi$ to denote that ϕ is a face of Π. The faces of a polyhedron form a graded lattice under inclusion. Every projective polyhedron has a face lattice isomorphic to that of a convex polytope, possibly of lower dimension.

The *dual* of a polyhedron Π, denoted Π^*, is the set of points whose dual hyperplanes intersect Π in one of its faces:

$$\Pi^* \triangleq \{p \mid (p^* \cap \Pi) \leq \Pi\}.$$

In other words, $p \in \Pi^*$ if and only if p^* either contains Π, supports Π, or completely misses Π. This definition generalizes both the polar of a convex polytope containing the origin and the projective dual of a flat. We easily verify that Π^* is a projective polyhedron whose face lattice is the inverse of the face lattice of Π. In particular, Π and Π^* have the same number of faces. See [58, pp. 59–64] and [51, pp. 143–150] for similar definitions.

We say that a polyhedron Π *separates* a set P of points and a set H of hyperplanes if Π contains P and the dual polyhedron Π^* contains the dual points H^*, or equivalently, if any hyperplane in H either contains Π or is disjoint from its relative interior. In particular, if Π is of full dimension, then the hyperplanes in H avoid the interior of Π. Both P and H may intersect the relative boundary of Π, and points in P may lie on hyperplanes in H. See Figure 3. Note that Π separates P and H if and only if Π^* separates H^* and P^*. We say that P and H are *r-separable*, denoted $P \bowtie_r H$, if there is a projective polyhedron with at most r faces that separates them. We write $P \not\bowtie_r H$ if P and H are not r-separable.

Finally, an *r-polyhedral cover* of a set P of points and a set H of hyperplanes is an indexed set of subset pairs $\{(P_i, H_i)\}$ with the following properties.

- $P_i \subseteq P$ and $H_i \subseteq H$ for all i.
- If $p \in P$ and $h \in H$, then $p \in P_i$ and $h \in H_i$ for some i.
- $P_i \bowtie_r H_i$ for all i.

We emphasize that the subsets P_i are not necessarily disjoint, nor are the subsets H_i. We refer to each subset pair (P_i, H_i) in a polyhedral cover as a *r-polyhedral minor.* The *size* of a cover is the sum of the sizes of the subsets P_i and H_i.

Let $\pi_r(P, H)$ denote the size of the smallest r-polyhedral cover of P and H. Let $\pi^\circ_{d,r}(n, k)$ denote the maximum of $\pi_r(P, H)$ over all sets P of n points and H of k hyperplanes in $\mathbb{RP}^d$, such that no point lies on any hyperplane. In all our terminology and notation, whenever the parameter r is omitted, we take it to be a fixed constant. In the remainder of this section, we derive asymptotic lower bounds for $\pi^\circ_d(n, k)$.

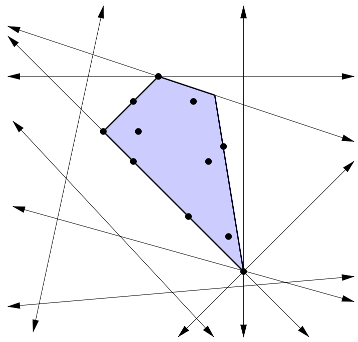

FIG. 3. *A polygon separating a set of points and a set of lines in $\mathbb{RP}^2$.*

5.2. Topological Properties. The lower bound proofs in [31] relied on the following trivial observation: If we perturb a set of points and hyperplanes just enough to remove any point-hyperplane incidences, and every point is above every hyperplane in the perturbed set, then no point was below a hyperplane in the original set. In this section, we establish the corresponding, but no longer trivial, property of separable sets and polyhedral covers. Informally, if a set of points and hyperplanes is not separable, then arbitrarily small perturbations cannot make it separable. Similarly, arbitrarily small perturbations of a set of points and hyperplane cannot decrease its minimum polyhedral cover size.

We start by proving a more obvious property of convex polytopes, namely, that infinitesimally perturbing a set of points can only increase the complexity of its convex hull.

LEMMA 5.1. *For any integers n and r, the set of n-point configurations in $\mathbb{R}^d$ whose convex hulls have at most r faces is topologically closed.*

Proof. Let $A = \{a_1, a_2, \ldots, a_n\}$ and $B = \{b_1, b_2, \ldots, b_n\}$ be two n-point configurations (*i.e.*, indexed sets of n points) in $\mathbb{R}^d$. We say that A *is simpler than* B, written $A \sqsubseteq B$, if for any subset of B contained in a facet of $\operatorname{conv}(B)$, the corresponding subset of A is contained in a facet of $\operatorname{conv}(A)$.[9] Equivalently, $A \sqsubseteq B$ if and only if for any subset of $d+1$ points in B, d of whose vertices lie on a facet of $\operatorname{conv}(B)$, the corresponding simplex in A either has the same orientation or is degenerate. Simpler point sets have less complex convex hulls: if $A \sqsubseteq B$, then $|\operatorname{conv}(A)| \le |\operatorname{conv}(B)|$. If both $A \sqsubseteq B$ and $B \sqsubseteq A$, then the convex hulls of A and B are combinatorially equivalent.

If B is fixed, then the relation $A \sqsubseteq B$ can be encoded as the conjunction of a finite number of algebraic inequalities of the form

$$\begin{vmatrix} 1 & a_{i_0 1} & a_{i_0 2} & \cdots & a_{i_0 d} \\ 1 & a_{i_1 1} & a_{i_1 2} & \cdots & a_{i_1 d} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & a_{i_d 1} & a_{i_d 2} & \cdots & a_{i_d d} \end{vmatrix} \Diamond\, 0,$$

where $\Diamond$ is either $\ge$, $=$, or $\le$, and a_{ij} denotes the jth coordinate of $a_i \in A$. In every such inequality, the corresponding points $b_{i_1}, b_{i_2}, \ldots, b_{i_d} \in B$ lie on a single facet of

[9] Every set of points is simpler than itself. It would be more correct, but also more awkward, to say "A is at least as simple as B".

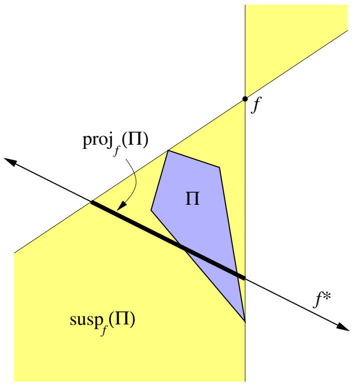

FIG. 4. *Suspension (double wedge) and projection (line segment) of a polygon by a point in* $\mathbb{RP}^2$.

conv(B). For every d-tuple of points in B contained in a facet of conv(B), there are $n-d$ such inequalities, one for every other point in B. (If we replaced the loose inequalities $\leq, \geq$ with strict inequalities $<, >$, the resulting expression would encode the combinatorial equivalence of conv(A) and conv(B).) Thus, for any *fixed* n-point set B, the set $\{A \in (\mathbb{R}^d)^n \mid A \sqsubseteq B\}$ is the intersection of a finite number of closed algebraic halfspaces, and is thus a closed semialgebraic set.

There are only finitely many equivalence classes of convex polytopes with a given number of faces or vertices [34]. Thus, there is a *finite* set $\mathcal{B} = \{B^1, B^2, \ldots\}$ of n-point configurations, one of each possible combinatorial type, such that if conv(A) has at most r faces, then $A \sqsubseteq B^i$ for some configuration $B^i \in \mathcal{B}$. It follows that the set of n-point configurations whose convex hulls have at most r faces is the union of a finite number of closed sets, and is therefore closed. ☐

Before continuing, we need to introduce one more important concept. For any subset $X \subseteq \mathbb{RP}^d$ and any flat f, the *suspension of X by f*, denoted $\text{susp}_f(X)$, is formed by replacing each point in X by the smallest flat containing that point and f:

$$\text{susp}_f(X) \triangleq \bigcup_{p \in X} \text{span}(p \cup f).$$

The suspension of a subset of projective space roughly corresponds to an infinite cylinder over a subset of an affine space, at least when the suspending flat f is on the hyperplane at infinity. The *projection of X by f*, denoted $\text{proj}_f(X)$, is the intersection of the suspension and the dual flat f^*:

$$\text{proj}_f(X) \triangleq \text{susp}_f(X) \cap f^*.$$

In particular, $\text{susp}_f(X)$ is the set of all points in $\mathbb{RP}^d$ whose projection by f is in $\text{proj}_f(X)$. The projection of a subset of projective space corresponds to the orthogonal projection or shadow of a subset of affine space onto a lower-dimensional flat. See Figure 4. For any polyhedron Π and any flat f, $\text{susp}_f(\Pi)$ and $\text{proj}_f(\Pi)$ are also polyhedra. These two polyhedra have the same number of faces (in fact, they have isomorphic face lattices), although in general they have fewer faces than Π. In particular, $\text{susp}_f(f) = f$ and $\text{proj}_f(f) = \varnothing$.

LEMMA 5.2. *Let H be a set of hyperplanes in $\mathbb{RP}^d$. For any integers r and n, the set $\text{Sep}_r(H, n)$ of n-point configurations $P \in (\mathbb{RP}^d)^n$ such that P and H are r-separable is topologically closed.*

Proof. There are two cases to consider: Either the hyperplanes in H do not have a common intersection, or they intersect in a common flat. The proof of second case relies on the first.

Case 1 ($\bigcap H = \varnothing$)*:* Any polyhedron that separates P and H must be completely contained in a closed d-dimensional cell of the arrangement of H. Since there are only finitely many such cells, it suffices to show, for each closed d-cell $\mathcal{C}$, that the set $\mathrm{Sep}_r(H,n) \cap \mathcal{C}^n$ of n-point configurations contained in $\mathcal{C}$ and r-separable from H is topologically closed. We will actually show that $\mathrm{Sep}_r(H,n) \cap \mathcal{C}^n$ is a compact semialgebraic set.

Fix a cell $\mathcal{C}$. Since the hyperplanes in H do not have a common intersection, both $\mathcal{C}$ and any polyhedra it contains must be polytopes. By choosing an appropriate hyperplane "at infinity" that misses $\mathcal{C}$, we can treat $\mathcal{C}$ and any polytopes it contains as *convex* polytopes in $\mathbb{R}^d$. Any separating polytope with r or fewer faces is the (closed) convex hull of some set of r points, all contained in $\mathcal{C}$. Thus, we can write

$$\mathrm{Sep}_r(H,n) \cap \mathcal{C}^n = \left\{ P \in \mathcal{C}^n \;\middle|\; \exists\, A \in \mathcal{C}^r : P \subset \mathrm{conv}(A) \,\wedge\, |\mathrm{conv}(A)| \le r \right\}.$$

Lemma 5.1 implies that the set

$$\left\{ (P,A) \in \mathcal{C}^{n+r} \;\middle|\; |\mathrm{conv}(A)| \le r \right\} = \left\{ A \in \mathcal{C}^r \;\middle|\; |\mathrm{conv}(A)| \le r \right\} \times \mathcal{C}^n$$

is compact (closed and bounded). The set

$$\left\{ (P,A) \in \mathcal{C}^{n+r} \;\middle|\; P \subset \mathrm{conv}(A) \right\}$$

can be rewritten as

$$\left\{ (P,A) \in \mathcal{C}^{n+r} \;\middle|\; \bigwedge_{i=1}^{n} \left(\exists \lambda_i \in [0,1]^r : \sum_{j=1}^{r} \lambda_{ij} a_j = p_i \;\wedge\; \sum_{j=1}^{r} \lambda_{ij} = 1 \right) \right\}.$$

(Here p_i is the ith point in P, a_j is the jth point in A, λ_i is the vector of barycentric coordinates for the point p_i, and λ_{ij} is its jth component.) This set is an orthogonal projection of the compact semialgebraic set

$$\left\{ (P,A,\Lambda) \in \mathcal{C}^{n+r} \times [0,1]^{r\times n} \;\middle|\; \bigwedge_{i=1}^{n} \left(\sum_{j=1}^{r} \lambda_{ij} a_j = p_i \;\wedge\; \sum_{j=1}^{r} \lambda_{ij} = 1 \right) \right\}$$

and is thus also compact. (Here Λ is the $n \times r$ matrix of barycentric coordinates λ_{ij}.) It follows that $\mathrm{Sep}_r(H,n) \cap \mathcal{C}^n$ is an orthogonal projection of the intersection of two compact sets and so must be compact.

Case 2 ($\bigcap H \neq \varnothing$)*:* The previous argument does not work in this case, because the cells in the arrangement of H are not simply connected, and thus are not polytopes. However, they are combinatorially equivalent to polytopes of lower dimension. To prove that $\mathrm{Sep}_r(H,n)$ is closed, we essentially project everything down to a lower-dimensional subspace in which the hyperplanes do not have a common intersection and apply our earlier argument.

We will actually prove that the complement of $\mathrm{Sep}_r(H,n)$ is open. Let P be an arbitrary set of n points in $\mathbb{RP}^d$ such that P and H are *not* r-separable. To prove the lemma, it suffices to show that there is an open set $\mathcal{U} \subseteq (\mathbb{RP}^d)^n$ with $P \in \mathcal{U}$, such that $Q \not\bowtie H$ for all $Q \in \mathcal{U}$.

Let $f = \bigcap H$, and let f^* be the flat dual to f. Without loss of generality, suppose that $P \setminus f = \{p_1, p_2, \ldots, p_m\}$ and $P \cap f = \{p_{m+1}, \ldots, p_n\}$ for some integer m. (Either of these two subsets may be empty.) The lower-dimensional hyperplanes $H \cap f^*$ do not have a common intersection, so by our earlier argument, $\mathrm{Sep}_r(H \cap f^*, m)$ is closed.

If some polyhedron Π separated P and H, then its projection $\mathrm{proj}_f(\Pi) \subseteq f^*$ would separate the projected points $\mathrm{proj}_f(P)$ and the lower-dimensional hyperplanes $H \cap f^*$. Conversely, if any polyhedron $\Pi \subseteq f^*$ separated $\mathrm{proj}_f(P)$ and $H \cap f^*$ then its suspension $\mathrm{susp}_f(\Pi)$ would separate P and H. Thus, $P \bowtie H$ if and only if $\mathrm{proj}_f(P) \bowtie (H \cap f^*)$. Moreover, since $\mathrm{proj}_f(P) = \mathrm{proj}_f(P \setminus f)$, it follows that $P \bowtie H$ if and only if $(P \setminus f) \bowtie H$. (Note that this argument is not valid for arbitrary flats f, but only for $f = \bigcap H$.)

Since by assumption $P \not\bowtie H$, it follows that $\mathrm{proj}_f(P) \not\bowtie (H \cap f^*)$. Thus, by Case 1, there is an open set $\mathcal{U}' \subseteq (f^*)^m$, with $\mathrm{proj}_f(P) \in \mathcal{U}'$, such that $S \not\bowtie H$ for any $S \in \mathcal{U}'$. Let $\mathcal{U}'' \subseteq (\mathbb{RP}^d \setminus f)^m$ be the set of m-point configurations R with $\mathrm{proj}_f(R) \in \mathcal{U}'$. Since $\mathbb{RP}^d \setminus f \cong f^* \times \mathbb{R}^{\dim f}$, we have $\mathcal{U}'' \cong \mathcal{U}' \times (\mathbb{R}^{\dim f})^m$, so $\mathcal{U}''$ is an open neighborhood of $P \setminus f$. For any $R \in \mathcal{U}''$, since $\mathrm{proj}_f(R) \not\bowtie (H \cap f^*)$, we have $R \not\bowtie H$. Finally, let $\mathcal{U} = \mathcal{U}'' \times (\mathbb{RP}^d)^{n-m}$; clearly, $\mathcal{U}$ is an open neighborhood of P. For every configuration $Q \in \mathcal{U}$, there is a subset $R \subseteq Q$ such that $R \not\bowtie H$, so $Q \not\bowtie H$. □

LEMMA 5.3. *Let P be a set of n points and H a set of k hyperplanes in $\mathbb{RP}^d$. For all $Q \in (\mathbb{RP}^d)^n$ in an open neighborhood of P, $\pi_r(Q, H) \geq \pi_r(P, H)$.*

Proof. For any indexed set of objects (points or hyperplanes) $X = \{x_1, x_2, \ldots\}$ and any set of indices $I \subseteq \{1, 2, \ldots, |X|\}$, let X_I denote the subset $\{x_i \mid i \in I\}$.

Fix two sets of indices $I \subseteq \{1, 2, \ldots, n\}$ and $J \subseteq \{1, 2, \ldots, k\}$, and consider the corresponding subsets $P_I \subseteq P$ and $H_J \subseteq H$. If $P_I \bowtie H_J$, define $\mathcal{U}_{I,J} = (\mathbb{RP}^d)^n$. Otherwise, define $\mathcal{U}_{I,J} \subseteq (\mathbb{RP}^d)^n$ to be an open neighborhood of P such that $Q_I \not\bowtie H_J$ for all $Q \in \mathcal{U}_{I,J}$. Lemma 5.2 implies the existence of such an open neighborhood.

Let $\mathcal{U}$ be the intersection of the $2^n 2^k$ open sets $\mathcal{U}_{I,J}$. Since each $U_{I,J}$ is an open neighborhood of P, $\mathcal{U}$ is also an open neighborhood of P. For all $Q \in \mathcal{U}$ and for all index sets I and J, if $Q_I \bowtie H_J$, then $P_I \bowtie H_J$. In other words, every r-polyhedral minor of Q and H corresponds to a r-polyhedral minor of P and H. Thus, for any r-polyhedral cover of Q and H, there is a corresponding r-polyhedral cover of P and H with exactly the same size. □

5.3. Lower Bounds. We are finally in a position to prove our combinatorial lower bounds. As in Section 2, let $I(P, H)$ denote the number of point-hyperplane incidences between P and H.

LEMMA 5.4. *Let P be a set of n points and H a set of k hyperplanes, such that no subset of a hyperplanes contains b points in its intersection. If P and H are r-separable, then $I(P, H) \leq r(a + b)(n + k)$.*

Proof. Let Π be a polyhedron with r faces that separates P and H. For any point $p \in P$ and hyperplane $h \in H$ such that p lies on h, there is some face ϕ of Π that contains p and is contained in h. For each face $\phi \leq \Pi$, let P_ϕ denote the points in P that are contained in ϕ, and let H_ϕ denote the hyperplanes in H that contain ϕ. Every point in P_ϕ lies on every hyperplane in H_ϕ.

Since no set of a hyperplanes can all contain the same b points, it follows that for all ϕ, either $|P_\phi| < b$ or $|H_\phi| < a$. Thus, we can bound $I(P, H)$ as follows.

$$I(P, H) \leq \sum_{\phi \leq \Pi} I(P_\phi, H_\phi) = \sum_{\phi \leq \Pi} \big(|P_\phi| \cdot |H_\phi|\big) \leq (a + b) \sum_{\phi \leq \Pi} \big(|P_\phi| + |H_\phi|\big).$$

Since Π has r faces, the last sum counts each point in P and each hyperplane in H at most r times. □

THEOREM 5.5. $\pi^\circ_d(n,k) = \Omega\left(n^{1-2/d(d+1)}k^{2/(d+1)} + n^{2/(d+1)}k^{1-2/d(d+1)}\right)$.

Proof. Let P be a restricted set of n points and H a set of k hyperplanes in $\mathbb{R}^d$, such that $I(P,H) = \Omega(n^{2/(d+1)}k^{1-2/d(d+1)})$ as described by Lemma 2.4. Consider any subsets $P_i \subseteq P$ and $H_i \subseteq H$ such that $P_i \bowtie_r H_i$. Applying Lemma 5.4 with $s = 2$ and $t = d$, we have $I(P_i, H_i) \leq (2+d)r(|P_i| + |H_i|)$. It follows that any collection of r-polyhedral minors that includes every incidence between P and H must have size at least $I(P,H)/(2+d)r$. Thus, $\pi_r(P,H) = \Omega(n^{2/(d+1)}k^{1-2/d(d+1)})$. Finally, Lemma 5.3 implies that we can perturb P slightly, removing all the incidences, without decreasing the polyhedral cover size.

The symmetric lower bound $\Omega(n^{1-2/d(d+1)}k^{2/(d+1)})$ follows by considering the dual points H^* and the dual hyperplanes P^*. □

When $d \leq 3$, this result follows from earlier bounds on the complexity of monochromatic covers derived in [31]. (In a monochromatic minor, either every point lies above every hyperplane, or every point lies below every hyperplane.)

Our d-dimensional lower bound only improves our $(d-1)$-dimensional lower bound when $k = O(n^{2/(d-1)})$ or $k = \Omega(n^{(d-1)/2})$. We can combine the lower bounds from all dimensions $1 \leq i \leq d$ into a single expression, as in [31, 30]:

$$\pi^\circ_d(n,k) = \Omega\left(\sum_{i=0}^{d}\left(n^{1-2/i(i+1)}k^{2/(i+1)} + n^{2/(i+1)}k^{1-2/i(i+1)}\right)\right).$$

If the relative growth rates of n and k are fixed, this entire sum reduces to a single term. In particular, when $k = n$, the best lower bound we can prove is $\pi^\circ_d(n,n) \geq \pi^\circ_2(n,n) = \Omega(n^{4/3})$, the proof of which requires only the original point-line configuration of Erdős. (See Theorem 2.2.)

We conjecture that $\pi^\circ_d(n,n) = \Theta(n^{2d/(d+1)})$. The lower bound would follow from a construction of n points and n hyperplanes with $\Omega(n^{2d/(d+1)})$ incident pairs, such that no d points lie on the intersection of d hyperplanes, or in other words, such that the bipartite incidence graph of P and H does not have $K_{d,d}$ as a subgraph. (The results of Clarkson *et al.* [19] and of Guibas, Overmars, and Robert [35] imply that this is the smallest forbidden subgraph for which the desired lower bound is possible.) An upper bound of $\pi^\circ_d(n,n) = O(n^{2d/(d+1)}2^{O(\log^* n)})$ follows from the running time of Matoušek's algorithm for Hopcroft's problem [42], using the results in the next section.

6. Better Offline Lower Bounds. Recall that a *partitioning algorithm*, given a set of points and hyperplanes, constructs a partition graph (which may depend arbitrarily on the input, at no cost), preprocesses the points, and queries the hyperplanes, using the algorithms in Figure 1. For a *polyhedral* partitioning algorithm, the partition graph's query regions are all convex (or projective) polyhedra, each with at most r faces, where r is some fixed constant.[10]

In [31], it was shown that the worst-case running time of any partitioning algorithm that solves Hopcroft's point-hyperplane incidence problem, given n points and k hyperplanes as input, requires time $\Omega(n\log k + n^{2/3}k^{2/3} + k\log n)$ when $d = 2$, or

[10]In most actual partitioning algorithms, every query region is either a simplex ($r = 2^{d+1}$) or a combinatorial cube ($r = 3^d + 1$).

$\Omega(n \log k + n^{5/6}k^{1/2} + n^{1/2}k^{5/6} + k \log n)$ for any $d \geq 3$. Here, by restricting our attention to polyhedral partitioning algorithms, we derive (slightly) better lower bounds in arbitrarily high dimensions.

THEOREM 6.1. *Let P be a set of points and H a set of hyperplanes, such that no point lies on any hyperplane. The running time of any polyhedral partitioning algorithm that solves Hopcroft's problem, given P and H as input, is at least $\Omega(\pi(P, H))$.*

Proof. For any partitioning algorithm $\mathcal{A}$, let $T_{\mathcal{A}}(P, H)$ denote its running time given the points P and hyperplanes H as input. Recall that $T_{\mathcal{A}}(P, H)$ is defined as follows:

$$T_{\mathcal{A}}(P,H) \triangleq \sum_{p \in P} \#\text{edges traversed by } p + \sum_{h \in H} \#\text{edges traversed by } h$$
$$= \sum_{\text{edge } e} \big(\#\text{points traversing } e + \#\text{hyperplanes traversing } e\big).$$

We say that a point or hyperplane *misses* an edge from v to w if it reaches v but does not traverse the edge. (It might still reach w by traversing some other edge.) Recall that each node in the partition graph has at most Δ outgoing edges, for some fixed constant Δ. Thus, for every edge that a point or hyperplane traverses, there are at most $\Delta - 1$ edges that it misses.

$$\Delta \cdot T_{\mathcal{A}}(P,H) \geq \sum_{\text{edge } e} \big(\#\text{points traversing } e + \#\text{points missing } e + \#\text{hyperplanes traversing } e + \#\text{hyperplanes missing } e\big).$$

Call any edge that leaves a primal node a primal edge, and any edge that leaves a dual node a dual edge.

$$\Delta \cdot T_{\mathcal{A}}(P,H) \geq \sum_{\substack{\text{primal}\\ \text{edge } e}} \big(\#\text{points traversing } e + \#\text{hyperplanes missing } e\big) + \sum_{\substack{\text{dual}\\ \text{edge } e}} \big(\#\text{hyperplanes traversing } e + \#\text{points missing } e\big)$$

For each primal edge e, let P_e be the set of points that traverse e, and let H_e be the set of hyperplanes that miss e. The edge e is associated with a query region Π, a polyhedron with at most r faces. The polyhedron Π separates P_e and H_e, since every point in P_e is contained in Π, and every hyperplane in H_e is disjoint from Π.

Similarly, for each dual edge e, let H_e be the set of hyperplanes that traverse it, and P_e the points that miss it. The associated polyhedral query region Π separates the dual points H_e^* and the dual hyperplanes P_e^*. By the definitions of separation and dual polyhedra, Π^* separates P_e and H_e.

Now our argument is similar to the proof of Theorem 3.5. Say that a node v *splits* a point p and a hyperplane h if both p and h reach v but no edge out of v is traversed by both p and h. For every point $p \in P$ and hyperplane $h \in H$, some node must split p and h, since otherwise p and h would both reach a leaf, and the output of the algorithm would be incorrect. Thus, for some outgoing edge e of this node, we have $p \in P_e$ and $h \in H_e$.

It follows that the collection of subset pairs $\{(P_e, H_e)\}$ is an r-polyhedral cover of P and H. The size of this cover is at least $\Delta \cdot T_{\mathcal{A}}(P, H)$ and, by definition, at most $\pi_r(P, H)$. $\square$

We emphasize that in order for this lower bound to hold, no point can lie on a hyperplane. If some point lies on a hyperplane, then the trivial partitioning algorithm, whose partition graph consists of a single leaf, correctly "detects" the incident pair at no cost. This is consistent with the intuition that it is trivial to prove that some point lies on some hyperplane, but proving that no point lies on any hyperplane is more difficult.

COROLLARY 6.2. *The worst-case running time of any polyhedral partitioning algorithm that solves Hopcroft's problem, given n points and k hyperplanes in $\mathbb{R}^d$, is*

$$\Omega\left(n^{1-2/d(d+1)}k^{2/(d+1)} + n^{2/(d+1)}k^{1-2/d(d+1)}\right).$$

Again, our d-dimensional bound improves our $(d-1)$-dimensional bound only for certain values of n and k. We can combine the lower bounds for different dimensions into the following single expression:

$$\Omega\left(n\log k + \sum_{i=0}^{d}\left(n^{1-2/i(i+1)}k^{2/(i+1)} + n^{2/(i+1)}k^{1-2/i(i+1)}\right) + k\log n\right).$$

This lower bound was previously known for *arbitrary* partitioning algorithms for counting or reporting versions of Hopcroft's problem—Given a set of points and lines, return the number of point-hyperplane incidences, or a list of incident pairs—as well as for offline *halfspace* counting and reporting problems [31].

7. Preprocessing-Query Time Tradeoffs. Based on the offline results in the previous section, we now establish tradeoff lower bounds between preprocessing and query time for online hyperplane emptiness and related queries. These are the first such lower bounds for any range searching problem in any model of computation; preprocessing time is not even defined in earlier models such as semigroup arithmetic and pointer machines. In some instances, our bounds allow us to improve the space-time tradeoff bounds established in Section 4.

THEOREM 7.1. *Any partition graph that supports line emptiness queries in time t after preprocessing time p satisfies the inequality $pt^2 = \Omega(n^2)$ in the worst case.*

Proof. Suppose $p < n^2$, since otherwise there is nothing to prove. Let $k = cp^{3/2}/n$, where c is a constant to be specified later. Note that $k = O(n^2)$, and since $p = \Omega(n\log n)$ by Theorem 3.5, we also have $k = \Omega(n^{1/2}\log^{1/2} n)$. Thus, there is a set of n points and k lines such that for any partition graph, the total time required to preprocess the n points and correctly answer the k line queries is at least $\alpha n^{2/3}k^{2/3} = \alpha c^{2/3}p$ for some positive constant α [31]. If we choose $c = (2/\alpha)^{3/2}$, the total query time is at least p. Thus, at least one query requires time at least $p/k = \Omega(n/p^{1/2})$. □

This lower bound almost matches the best known upper bound $pt^2 = O(n^2\log^{\varepsilon} n)$, due to Matoušek [42].

The following higher-dimensional bound follows from Corollary 6.2 using precisely the same argument.

THEOREM 7.2. *Any polyhedral partition graph that supports d-dimensional hyperplane emptiness queries in time t after preprocessing time p satisfies the inequalities $pt^{(d+2)(d-1)/2} = \Omega(n^d)$ and $pt^{2/(d-1)} = \Omega(n^{(d+2)/d})$ in the worst case. When $d \le 3$, these bounds apply to arbitrary partition graphs.*

Although in general these bounds are far from optimal, there are two interesting special cases that match known upper bounds [17, 38, 42] up to polylogarithmic factors.

COROLLARY 7.3. *Any polyhedral partition graph that supports hyperplane emptiness queries after $O(n \operatorname{polylog} n)$ preprocessing time requires query time $\Omega(n^{(d-1)/d}/\operatorname{polylog} n)$ in the worst case. When $d \leq 3$, this bound applies to arbitrary partition graphs.*

COROLLARY 7.4. *Any polyhedral partition graph that supports hyperplane emptiness queries in $O(\operatorname{polylog} n)$ time requires preprocessing time $\Omega(n^d/\operatorname{polylog} n)$ in the worst case. When $d \leq 3$, this bound applies to arbitrary partition graphs.*

In any realistic model of computation, the size of a data structure is a lower bound on its preprocessing time. However, partition graphs can have large subgraphs of that are never visited during the preprocessing phase or that cannot be visited by any query. In principle, since we do not charge for the actual construction of a partition graph, its size can be arbitrarily larger than its preprocessing time.

We say that a partition graph is *trim* if every edge that does not point to a leaf is traversed both while preprocessing some point and while answering some query. Given any partition graph, we can easily make it trim (trim it?) without increasing any of its resource bounds. Since $s \geq \Delta \cdot p$ for any trim partition graph, any asymptotic lower bound on the preprocessing time for a trim partition graph is also a lower bound on its size.

COROLLARY 7.5. *Any trim partition graph of size s that supports line emptiness queries in time t satisfies the inequality $st^2 = \Omega(n^2)$ in the worst case.*

This lower bound is optimal, up to constant factors. Chazelle [17] and Matoušek [42] describe a family of line query data structures satisfying the matching upper bound $st^2 = O(n^2)$.

COROLLARY 7.6. *Any trim polyhedral partition graph of size s that supports d-dimensional hyperplane emptiness queries in time t satisfies the inequality $st^{(d+2)(d-1)/2} = \Omega(n^d)$ in the worst case. In particular, if $t = O(\operatorname{polylog} n)$, then $s = \Omega(n^d/\operatorname{polylog} n)$. When $d \leq 3$, these bounds apply to arbitrary trim partition graphs.*

COROLLARY 7.7. *Any trim polyhedral partition graph of size s that supports d-dimensional hyperplane emptiness queries in time t satisfies the inequality $st^{2/(d-1)} = \Omega(n^{(d+2)/d})$ in the worst case. In particular, if $s = O(n \operatorname{polylog} n)$, then $t = \Omega(n^{(d-1)/d}/\operatorname{polylog} n)$. When $d \leq 3$, these bounds apply to arbitrary trim partition graphs.*

Corollary 7.6 is an improvement over Theorem 4.5 for all $s = \Omega(n^{d-1})$ or $t = O(n^{2(d-1)/d^3})$; and Corollary 7.7 is an improvement whenever $s = O(n^{1+2/(d^2+d)})$ or $t = \Omega(n^{1-2/d})$. (These bounds are conservative; the actual breakpoints are much messier.) The lower bounds for near-linear space and polylogarithmic query time are optimal up to polylogarithmic factors.

All of these lower bounds apply to hyperplane and halfspace counting and reporting queries as well, by Lemma 3.4. In fact, the results in [31] imply that for counting and reporting queries, the preprocessing-query tradeoffs apply to *arbitrary* partition graphs, and the space-time tradeoffs to arbitrary trim partition graphs, in all dimensions. Corollary 7.6 is always an improvement (although a small one) over the lower bound in Corollary 4.3.

8. Halfspace Emptiness Queries. The space and time bounds for the best hyperplane (or simplex) emptiness query data structures are only a polylogarithmic factor smaller than the bounds for hyperplane (or simplex) counting queries. The situation is entirely different for halfspace queries. The best halfspace counting data structure known requires roughly $\Omega(n^d)$ space to achieve logarithmic query time [17,

TABLE 2
Best known upper bounds for halfspace emptiness queries.

	Space	Preprocessing	Query Time	Source
$d \leq 3$	$O(n)$	$O(n \log n)$	$O(\log n)$	[21, 3, 26]
$d \geq 4$	$O(n^{\lfloor d/2 \rfloor} / \log^{\lfloor d/2 \rfloor} n)$	$O(n^{\lfloor d/2 \rfloor} / \log^{\lfloor d/2 \rfloor - \varepsilon} n)$	$O(\log n)$	[44]
	$O(n)$	$O(n^{1+\varepsilon})$	$O(n^{1-1/\lfloor d/2 \rfloor} 2^{O(\log^* n)})$	[39]
	$O(n)$	$O(n \log n)$	$O(n^{1-1/\lfloor d/2 \rfloor} \operatorname{polylog} n)$	[44]
	$n \leq s \leq n^{\lfloor d/2 \rfloor}$	$O(s \operatorname{polylog} n)$	$O((n \operatorname{polylog} n)/s^{1/\lfloor d/2 \rfloor})$	[44]

42]; whereas, the same query time can be achieved with $o(n^{\lfloor d/2 \rfloor})$ space if we only want to know whether the halfspace is empty [44].

Table 2 lists the resource bounds for the best known online halfspace emptiness data structures. The fastest offline algorithm, given n points and k halfspaces, requires

$$O\left(n \log k + (nk)^{\lfloor d/2 \rfloor / (\lfloor d/2 \rfloor + 1)} \operatorname{polylog}(n+k) + k \log n\right)$$

time to decide if any point lies in any halfspace [44]. In contrast, the only lower bounds previously known for halfspace emptiness queries are trivial. Linear space and logarithmic query time are required to answer online queries. A simple reduction from the set intersection problem shows that $\Omega(n \log k + k \log n)$ time is required for the offline problem in the algebraic decision tree and algebraic computation tree models [8, 50].

In this section, we derive the first nontrivial lower bounds on the complexity of halfspace emptiness queries. To prove our results, we use a simple reduction argument to transform hyperplane queries into halfspace queries in a higher-dimensional space [31, 29]. A similar transformation is described by Dwyer and Eddy [27].

Define the function $\sigma_d : \mathbb{R}^{d+1} \to \mathbb{R}^{\binom{d+2}{2}}$ as follows:

$$\sigma_d(x_0, x_1, \ldots, x_d) = \left(x_0^2, x_1^2, \ldots, x_d^2, \sqrt{2}\, x_0 x_1, \sqrt{2}\, x_0 x_2, \ldots, \sqrt{2}\, x_{d-1} x_d\right).$$

This map has the property that $\langle \sigma_d(p), \sigma_d(h) \rangle = \langle p, h \rangle^2$ for any vectors $p, h \in \mathbb{R}^{d+1}$, where $\langle \cdot, \cdot \rangle$ denotes the usual inner product. In a more geometric setting, σ_d maps points and hyperplanes in $\mathbb{R}^d$, represented as homogeneous coordinate vectors, to points and hyperplane in $\mathbb{R}^{d(d+3)/2}$, also represented in homogeneous coordinates. For any point p and hyperplane h in $\mathbb{R}^d$, the point $\sigma_d(p)$ is contained in the hyperplane $\sigma_d(h)$ if and only if p is contained in h; otherwise, $\sigma_d(p)$ is strictly above $\sigma_d(h)$. Thus, a hyperplane h intersects a point set P if and only if the closed lower halfspace $\sigma_d(h)^-$ intersects the lifted point set $\sigma_d(P)$. In other words, any (lower) halfspace emptiness data structure for $\sigma_d(P)$ is also a hyperplane emptiness data structure for P.

Unfortunately, this is not quite enough to give us our lower bounds, since the reduction does not preserve the model of computation. Specifically, the query regions in a partition graph used to answer d-dimensional queries must be subsets of $\mathbb{R}^d$. To complete the reduction, we need to show that the $d(d+3)/2$-dimensional partition graph can be "pulled back" to a d-dimensional partition graph.

In order for such a transformation to be possible, we need to restrict the query regions allowed in our partition graphs. A *Tarski cell* is a semialgebraic set defined by a constant number of polynomial equalities and inequalities, each of constant degree. Every Tarski cell has a constant number of connected components, and the intersection

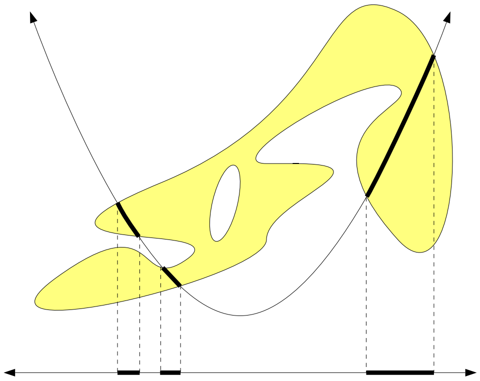

FIG. 5. *Each Tarski cell induces a constant number of lower-dimensional query regions.*

of any two Tarski cells is another Tarski cell (with larger constants). A *semialgebraic* partition graph is a partition graph whose query regions are all Tarski cells.

THEOREM 8.1. *Let P be a set of points in $\mathbb{R}^d$, and let $\hat{P} = \sigma_d(P)$. Given a semialgebraic partition graph $\hat{G}$ that supports $d(d+3)/2$-dimensional halfspace emptiness queries over $\hat{P}$, we can construct a semialgebraic partition graph G that supports d-dimensional hyperplane emptiness queries over P, with the same asymptotic space, preprocessing time, and query time bounds.*

Proof. We actually prove a stronger theorem, by assuming only that $\hat{G}$ supports emptiness queries for hyperplanes of the form $\sigma_d(h)$. Since no point in $\hat{P}$ is ever below such a hyperplane, any partition graph that supports lower halfspace emptiness queries also supports our restricted class of hyperplane emptiness queries. As we noted earlier, these queries are equivalent to hyperplane emptiness queries over the original point set P.

Given $\hat{G}$, we construct G as follows. G has the same set of nodes as $\hat{G}$, but with different query regions. Since each query region $\hat{R}$ in the original partition graph $\hat{G}$ is a Tarski cell, it intersects the algebraic surface $\sigma_d(\mathbb{R}^d)$ in a constant number of connected components $\hat{R}_1, \hat{R}_2, \ldots, \hat{R}_\delta$, where the constant δ depends on the number and degree of the inequalities that define $\hat{R}$. The query regions in G are the preimages $R_i = \sigma_d^{-1}(\hat{R}_i)$ of these components. See Figure 5.

The edge associated with each d-dimensional query region R_i has the same endpoints as the edge associated with the original query region $\hat{R}$. Thus, there may be several edges in G with the same source and target. (Recall that partition graphs are directed acyclic *multi*graphs.) If a query region $\hat{R}$ does not intersect $\sigma_d(\mathbb{R}^d)$, then the corresponding edge in $\hat{G}$ is not represented in G at all, so G may not be a connected graph. Nodes in G that are not connected to the root can be safely discarded. The size, preprocessing time, and query time for G are clearly at most a constant factor more than the corresponding resources for $\hat{G}$.

The leaf subsets P_ℓ in G cannot be larger than the corresponding subsets $\hat{P}_\ell$ in $\hat{G}$. (They might be smaller, but that only helps us.) Similarly, a hyperplane query cannot reach more leaves in G than the corresponding query reaches in $\hat{G}$. It follows that G supports hyperplane emptiness queries: For any hyperplane h, if $\hat{G}$ reports that $\sigma_d(h)$ is empty, G (correctly) reports that h is empty. □

We emphasize that some restriction on the query regions is necessary to prove any nontrivial lower bounds for halfspace emptiness queries. There is a partition graph of constant size, requiring only linear preprocessing, that supports halfspace emptiness queries in constant time. The graph consists of a single primal node with two query regions—the convex hull of the points and its complement—and two leaves. On the other hand, our restriction to Tarski cells is stronger than necessary. It suffices that every query region intersects (some projective transformation of) the surface $\sigma_d(\mathbb{R}^d)$ in a constant number of connected components.

The following corollaries are now immediate consequences of our earlier results.

COROLLARY 8.2. *For any $d \geq 2$, any semialgebraic partition graph that supports d-dimensional halfspace emptiness queries has size $\Omega(n)$, preprocessing time $\Omega(n \log n)$, and worst-case query time $\Omega(\log n)$.*

COROLLARY 8.3. *Any semialgebraic partition graph of size s that supports $d(d+3)/2$-dimensional halfspace emptiness queries in time t satisfies the inequality $st^d = \Omega((n/\log n)^{d-(d-1)/(d+1)})$ in the worst case.*

COROLLARY 8.4. *The worst-case running time of any semialgebraic partitioning algorithm which, given n points and k halfspaces in $\mathbb{R}^d$, decides if any halfspace contains a point, is $\Omega(n \log k + k \log n)$ for all $2 \leq d \leq 4$, $\Omega(n \log k + n^{2/3}k^{2/3} + k \log n)$ for all $5 \leq d \leq 8$, and $\Omega(n \log k + n^{5/6}k^{1/2} + n^{1/2}k^{5/6} + k \log n)$ for all $d \geq 9$.*

COROLLARY 8.5. *Any semialgebraic partition graph that supports 5-dimensional halfspace emptiness queries in time t after preprocessing time p satisfies the inequality $pt^2 = \Omega(n^2)$ in the worst case. Any trim semialgebraic partition graph of size s that supports 5-dimensional halfspace emptiness queries in time t satisfies the inequality $st^2 = \Omega(n^2)$ in the worst case.*

COROLLARY 8.6. *Any semialgebraic partition graph that supports 9-dimensional halfspace emptiness queries in time t after preprocessing time p satisfies the inequalities $pt^5 = \Omega(n^3)$ and $pt = \Omega(n^{5/3})$ in the worst case. Any trim semialgebraic partition graph of size s that supports 9-dimensional halfspace emptiness queries in time t satisfies the inequalities $st^5 = \Omega(n^3)$ and $st = \Omega(n^{5/3})$ in the worst case.*

Corollaries 8.2 and 8.4 are optimal when $d \leq 3$; Corollary 8.4 is also optimal up to polylogarithmic factors when $d = 5$; and Corollary 8.5 is optimal up to polylogarithmic factors.

Theorem 8.1 does not imply better offline lower bounds or preprocessing/query tradeoffs for halfspace emptiness queries in dimensions higher than 9, since the corresponding hyperplane results require polyhedral query regions. Marginally better lower bounds can be obtained directly in dimensions 14 and higher(!) in the *polyhedral* partition graph model by generalizing the arguments in Sections 5 and 6 (as in [30]). However, since these lower bounds are far from optimal, we omit further details.

9. Conclusions. We have presented the first nontrivial lower bounds on the complexity of hyperplane and halfspace emptiness queries. Our lower bounds apply to a broad class of range query data structures based on recursive decomposition of primal and/or dual space.

The lower bounds we developed for counting and reporting queries actually apply to any type of query where the points in the query range are required as the union of several subsets. For example, simplex range searching data structures are typically constructed by composing several levels of halfspace "counting" data structures [42]. To answer a query for the intersection of k halfspaces, the points in the first halfspace are (implicitly) extracted as the disjoint union of several subsets, and a $(k-1)$-halfspace query is recursively performed on each subset.

With a few notable exceptions, our lower bounds are far from the best known upper bounds, and a natural open problem is to close the gap. In particular, we have only "trivial" lower bounds for four-dimensional halfspace emptiness queries. We conjecture that the correct space-time tradeoffs are $st^d = \Theta(n^d)$ for hyperplanes and $st^{\lfloor d/2 \rfloor} = \Theta(n^{\lfloor d/2 \rfloor})$ for halfspaces. Since these bounds are achieved by current algorithms—exactly for hyperplanes [42], within polylogarithmic factors for halfspaces [44]—the only way to prove our conjecture is to improve the lower bounds.

Our space-time tradeoffs derive from lower bounds for halfspace queries in the semigroup arithmetic model [10], and our preprocessing-query tradeoffs follow from lower bounds on the combinatorial complexity of polyhedral covers. Any improvements to these lower bounds would improve our results as well. Both of these results ultimately reduce to bounds on the minimum size of a decomposition of the (weighted) incidence graph of a set of points and a set of halfspaces into complete bipartite subgraphs.

The best known data structures for d-dimensional hyperplane emptiness queries and $2d$- or $(2d+1)$-dimensional halfspace emptiness queries have the same resource bounds. We conjecture that this is also true of *optimal* data structures for these problems. Is there a reduction from hyperplane queries to halfspace queries that only increases the dimension by a constant factor (preferably two)?

Finally, can our techniques be applied to other closely related problems, such as nearest neighbor queries [2], linear programming queries [40, 11] and ray shooting queries [2, 20, 41, 44]?

Acknowledgments. I thank Pankaj Agarwal for suggesting studying the complexity of online emptiness problems.

REFERENCES

[1] P. K. Agarwal and J. Erickson, *Geometric range searching and its relatives*, in Advances in Discrete and Computational Geometry, B. Chazelle, J. E. Goodman, and R. Pollack, eds., vol. 223 of Contemporary Mathematics, AMS Press, 1999, pp. 1–56.

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

[3] A. Aggarwal, M. Hansen, and T. Leighton, *Solving query-retrieval problems by compacting Voronoi diagrams*, in Proc. 22nd Annu. ACM Sympos. Theory Comput., 1990, pp. 331–340.

[4] A. Andersson, *Sublogarithmic searching without multiplications*, in Proc. 36th Annu. IEEE Pympos. Found. Comput. Sci., 1995, pp. 655–663.

[5] ———, *Faster deterministic sorting and searching in linear space*, in Proc. 37th Annu. IEEE Sympos. Found. Comput. Sci., 1996, pp. 135–141.

[6] A. Andersson and K. Swanson, *On the difficulty of range searching*, Comput. Geom. Theory Appl., 8 (1997), pp. 115–122.

[7] S. Arya and D. M. Mount, *Approximate range searching*, in Proc. 11th Annu. ACM Sympos. Comput. Geom., 1995, pp. 172–181.

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

[9] A. Borodin, R. Ostrovsky, and Y. Rabani, *Lower bounds for high dimensional nearest neighbor search and related problems*, in Proc. 31st Annu. ACM Sympos. Theory Comput., 1999, pp. 312–321.

[10] H. Brönnimann, B. Chazelle, and J. Pach, *How hard is halfspace range searching?*, Discrete Comput. Geom., 10 (1993), pp. 143–155.

[11] T. M. Chan, *Fixed-dimensional linear programming queries made easy*, in Proc. 12th Annu. ACM Sympos. Comput. Geom., 1996, pp. 284–290.

[12] ———, *Output-sensitive results on convex hulls, extreme points, and related problems*, Discrete Comput. Geom., 16 (1996), pp. 369–387.

[13] B. Chazelle, *A functional approach to data structures and its use in multidimensional searching*, SIAM J. Comput., 17 (1988), pp. 427–462.

[14] ———, *Lower bounds on the complexity of polytope range searching*, J. Amer. Math. Soc., 2 (1989), pp. 637–666.

[15] ———, *Lower bounds for orthogonal range searching, I: The reporting case*, J. ACM, 37 (1990), pp. 200–212.
[16] ———, *Lower bounds for orthogonal range searching, II: The arithmetic model*, J. ACM, 37 (1990), pp. 439–463.
[17] ———, *Cutting hyperplanes for divide-and-conquer*, Discrete Comput. Geom., 9 (1993), pp. 145–158.
[18] ———, *Lower bounds for off-line range searching*, Discrete Comput. Geom., 17 (1997), pp. 53–66.
[19] B. Chazelle, H. Edelsbrunner, L. J. Guibas, and M. Sharir, *Algorithms for bichromatic line segment problems and polyhedral terrains*, Algorithmica, 11 (1994), pp. 116–132.
[20] B. Chazelle and J. Friedman, *Point location among hyperplanes and unidirectional ray-shooting*, Comput. Geom. Theory Appl., 4 (1994), pp. 53–62.
[21] B. Chazelle, L. J. Guibas, and D. T. Lee, *The power of geometric duality*, BIT, 25 (1985), pp. 76–90.
[22] B. Chazelle and B. Rosenberg, *Simplex range reporting on a pointer machine*, Comput. Geom. Theory Appl., 5 (1996), pp. 237–247.
[23] K. Clarkson, H. Edelsbrunner, L. J. Guibas, M. Sharir, and E. Welzl, *Combinatorial complexity bounds for arrangements of curves and spheres*, Discrete Comput. Geom., 5 (1990), pp. 99–160.
[24] K. L. Clarkson, *An algorithm for approximate closest-point queries*, in Proc. 10th Annu. ACM Sympos. Comput. Geom., 1994, pp. 160–164.
[25] M. de Berg, M. Overmars, and O. Schwarzkopf, *Computing and verifying depth orders*, SIAM J. Comput., 23 (1994), pp. 437–446.
[26] D. P. Dobkin and D. G. Kirkpatrick, *Determining the separation of preprocessed polyhedra – a unified approach*, in Proc. 17th Internat. Colloq. Automata Lang. Program., vol. 443 of Lecture Notes Comput. Sci., Springer-Verlag, 1990, pp. 400–413.
[27] R. Dwyer and W. Eddy, *Maximal empty ellipsoids*, Internat. J. Comput. Geom. Appl., 6 (1996), pp. 169–186.
[28] H. Edelsbrunner and M. H. Overmars, *Batched dynamic solutions to decomposable searching problems*, J. Algorithms, 6 (1985), pp. 515–542.
[29] J. Erickson, *On the relative complexities of some geometric problems*, in Proc. 7th Canad. Conf. Comput. Geom., 1995, pp. 85–90.
[30] ———, *New lower bounds for halfspace emptiness*, in Proc. 37th Annu. IEEE Sympos. Found. Comput. Sci., 1996, pp. 472–481.
[31] ———, *New lower bounds for Hopcroft's problem*, Discrete Comput. Geom., 16 (1996), pp. 389–418.
[32] ———, *Space-time tradeoffs for emptiness queries*, in Proc. 13th Annu. ACM Sympos. Comput. Geom., 1997, pp. 304–313.
[33] M. L. Fredman, *Lower bounds on the complexity of some optimal data structures*, SIAM J. Comput., 10 (1981), pp. 1–10.
[34] J. E. Goodman and R. Pollack, *Allowable sequences and order types in discrete and computational geometry*, in New Trends in Discrete and Computational Geometry, J. Pach, ed., vol. 10 of Algorithms and Combinatorics, Springer-Verlag, 1993, pp. 103–134.
[35] L. Guibas, M. Overmars, and J.-M. Robert, *The exact fitting problem in higher dimensions*, Comput. Geom. Theory Appl., 6 (1996), pp. 215–230.
[36] M. Henk, J. Richter-Gebert, and G. M. Ziegler, *Basic properties of convex polytopes*, in Handbook of Discrete and Computational Geometry, J. E. Goodman and J. O'Rourke, eds., CRC Press LLC, 1997, ch. 13, pp. 243–270.
[37] J. Kleinberg, *Two algorithms for nearest-neighbor search in high dimension*, in Proc. 29th Annu. ACM Sympos. Theory Comput., 1997, pp. 599–608.
[38] J. Matoušek, *Efficient partition trees*, Discrete Comput. Geom., 8 (1992), pp. 315–334.
[39] ———, *Reporting points in halfspaces*, Comput. Geom. Theory Appl., 2 (1992), pp. 169–186.
[40] ———, *Linear optimization queries*, J. Algorithms, 14 (1993), pp. 432–448.
[41] ———, *On vertical ray shooting in arrangements*, Comput. Geom. Theory Appl., 2 (1993), pp. 279–285.
[42] ———, *Range searching with efficient hierarchical cuttings*, Discrete Comput. Geom., 10 (1993), pp. 157–182.
[43] ———, *Geometric range searching*, ACM Comput. Surv., 26 (1994), pp. 421–461.
[44] J. Matoušek and O. Schwarzkopf, *On ray shooting in convex polytopes*, Discrete Comput. Geom., 10 (1993), pp. 215–232.
[45] P. B. Milterson, *Lower bounds for Union-Split-Find related problems on random access machines*, in Proc. 26th Annu. ACM Sympos. Theory Comput., 1994, pp. 625–634.

[46] P. B. Milterson, N. Nisan, S. Safra, and A. Widgerson, *On data structures and asymmetric communication complexity*, J. Comput. System Sci., 57 (1998), pp. 37–49.
[47] M. H. Overmars, *Efficient data structures for range searching on a grid*, J. Algorithms, 9 (1988), pp. 254–275.
[48] J. Pach and P. K. Agarwal, *Combinatorial Geometry*, John Wiley & Sons, New York, NY, 1995.
[49] F. P. Preparata and M. I. Shamos, *Computational Geometry: An Introduction*, Springer-Verlag, New York, NY, 1985.
[50] J. M. Steele and A. C. Yao, *Lower bounds for algebraic decision trees*, J. Algorithms, 3 (1982), pp. 1–8.
[51] J. Stolfi, *Oriented Projective Geometry: A Framework for Geometric Computations*, Academic Press, New York, NY, 1991.
[52] L. Székely, *Crossing numbers and hard Erdős problems in discrete geometry*, Combinatorics, Probability, and Computing, 6 (1997), pp. 353–358.
[53] E. Szemerédi and W. Trotter, Jr., *Extremal problems in discrete geometry*, Combinatorica, 3 (1983), pp. 381–392.
[54] R. E. Tarjan, *A class of algorithms which require nonlinear time to maintain disjoint sets*, J. Comput. Syst. Sci., 18 (1979), pp. 110–127.
[55] D. E. Willard, *Log-logarithmic worst case range queries are possible in space* $\Theta(n)$, Inform. Process. Lett., 17 (1983), pp. 81–89.
[56] A. C. Yao, *On the complexity of maintaining partial sums*, SIAM J. Comput., 14 (1985), pp. 277–288.
[57] A. C.-C. Yao, *Should tables be sorted?*, J. ACM, 28 (1981), pp. 615–628.
[58] G. M. Ziegler, *Lectures on Polytopes*, vol. 152 of Graduate Texts in Mathematics, Springer-Verlag, Heidelberg, 1994.