Space-Time Tradeoffs for Emptiness Queries[*]
(Extended abstract)

Jeff Erickson

Center for Geometric Computing
Computer Science Department
Duke University
Durham, NC 27708–0129
jeffe@cs.duke.edu
http://www.cs.duke.edu/~jeffe

Abstract

We present the first nontrivial space-time tradeoff lower bounds for 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 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 must satisfy the inequality $st^d = \Omega((n/\log n)^{d-(d-1)/(d+1)})$. Using different techniques, we show that $\Omega(n^d/\operatorname{polylog} n)$ preprocessing time is required to achieve polylogarithmic query time, and that $\Omega(n^{(d-1)/d}/\operatorname{polylog} n)$ query time is required if only $O(n \operatorname{polylog} n)$ preprocessing time is used. These two lower bounds are optimal up to polylogarithmic factors. For two-dimensional queries, we obtain an optimal continuous tradeoff between these two extremes. Finally, using a reduction argument, we show that the same lower bounds hold for halfspace emptiness queries in $\mathbb{R}^{d(d+3)/2}$ on a restricted class of partition graphs.

1 Introduction

We present the first nontrivial lower bounds on the complexity of data structures that support hyperplane and halfspace emptiness queries. Emptiness query data structures are used to solve several geometric problems, including point location [12], ray shooting [2, 12, 24, 27], nearest and farthest neighbor queries [2], linear programming queries [23, 7], depth ordering [5], collision detection [11], and output-sensitive convex hull construction [23, 8].

Most previous range searching lower bounds are presented in the so-called *semigroup arithmetic* model, originally introduced by Fredman [20] and later refined by Yao [31]. In this model, the points are given weights from an additive 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 number of semigroup additions performed on these partial sums to obtain the answer. (More formal definitions are given in Section 2.) Lower bounds have been established in this model for several types of query ranges [9, 6], in many cases matching the complexities of the corresponding data structures.

Unfortunately, the semigroup model cannot be used to study the complexity of emptiness queries. If the query range is empty, we perform no additions; conversely, if we perform even a single addition, the query range cannot be empty. Similar arguments apply to Tarjan's *pointer machine* model [30], which has been used to derive output-sensitive lower bounds for range *reporting* problems [14]. In fact, the only lower bounds previously known for emptiness queries are trivial. The size of any range searching data structure must be $\Omega(n)$, since it must store each of the points, and the query time must be at least $\Omega(\log n)$ in any reasonable model of computation (such as algebraic computation trees [4] or real RAMs [29]).

[*]Portions of this research were done while the author was a student in the Computer Science Division, U. C. Berkeley, with the support of a GAANN Fellowship. This work was also supported by the National Science Foundation under grant DMS-9627683 and by the U. S. Army Research Office under grant DAAH04-96-1-0013.

Space	Preprocessing	Query Time	Source
$O(n^d/\log^d n)$	$O(n^d/\log^{d-\varepsilon} n)$	$O(\log n)$	[10, 25]
$O(n)$	$O(n^{1+\varepsilon})$	$O(n^{1-1/d})$	[25]
$O(n)$	$O(n \log n)$	$O(n^{1-1/d} \operatorname{polylog} n)$	[21]
$n \le s \le n^d/\log^d n$	$O(n^{1+\varepsilon} + s \log^\varepsilon n)$	$O(n/s^{1/d})$	[10, 25]

Table 1. Best known upper bounds for hyperplane emptiness queries.

Our new 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. Our model is powerful enough to describe most, if not all[1], known data structures for these problems. Partition graphs have been previously used to study offline range searching problems such as Hopcroft's point-line incidence problem [19, 18].

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 review of range searching techniques and results, see the surveys by Matoušek [26] and Agarwal [1].)

- $s = \Omega(n)$, $p = \Omega(n \log n)$, and $t = \Omega(\log n)$.
- 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 [6] 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.
- Lower bounds on the complexity of Hopcroft's point-hyperplane incidence problem [19, 18] imply the worst case bounds $pt^{(d+2)(d-1)/2} = \Omega(n^d)$ 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 $t = O(\operatorname{polylog} n)$ [21, 25]. These results require restrictions on the partition graphs when $d \ge 4$.

All of the previous lower bounds also apply to hyperplane and halfspace counting queries in $\mathbb{R}^d$, and to halfspace emptiness queries in $\mathbb{R}^{d(d+3)/2}$ under some restrictions on the partition graphs.

Lower bounds in the semigroup arithmetic model imply lower bounds for counting queries in the partition graph model. Thus, any partition graph that supports halfspace counting queries satisfies $st^d = \Omega((n/\log n)^{d-(d-1)/(d+1)})$. We also derive the lower bound $st^{d(d+1)/2} = \Omega(n^d)$ for hyperplane queries in the semigroup model, and thus for hyperplane counting queries in the partition graph model as well. This improves the lower bound inherited from hyperplane emptiness queries whenever $s = \Omega(n^{d-1})$ or $t = O(n^{2/d(d+1)})$.

The rest of the paper is organized as follows. Section 2 reviews the definition of the semigroup arithmetic model and states some useful results. In Section 3, we define partition graphs, describe how they are used to answer hyperplane and halfspace queries, and state a few of their basic properties. We prove our new time-space and preprocessing-query tradeoffs for hyperplane emptiness queries in Section 4. In Section 5, we use a reduction argument to derive lower bounds for halfspace emptiness queries. Finally, in Section 6, we offer our conclusions.

2 Semigroup Arithmetic

An *additive 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 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. For example, $(\mathbb{Z}, +)$ and $(\{\text{true}, \text{false}\}, \vee)$ are faithful semigroups, but $(\{0, 1\}, + \bmod 2)$ is not faithful.

Let P be a set of n points in $\mathbb{R}^d$, let $(S, +)$ be a faithful commutative additive 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. (Since S need not have an additive identity, we assign a special value *nil* to the empty sum.) 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.

[1]Difficulties in modeling range searching data structures as partition graphs are discussed in [19, Section 3.5].

Let $x_1, x_2, \dots, x_n$ be a set of n variables over S. A *generator* $g(x_1, \dots, x_n)$ 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, a *storage scheme* for (P, Q, S) is a collection of generators $\{g_1, g_2, \dots, g_s\}$ with the following property: For any query range $q \in Q$, there is an set of indices $I_q \subseteq \{1, 2, \dots, s\}$ and a set of labeled nonnegative integers $\{\beta_i \mid i \in I_q\}$ such that

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

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 $p \in P$ [9]. For the same reason, even though the semigroup $(\{\text{true}, \text{false}\}, \vee)$ is faithful, the semigroup model cannot be used to prove lower bounds for emptiness queries. Emptiness queries can also be formulated as queries over the one-element semigroup $(\{*\}, * + * = *)$, but this semigroup is not faithful.

For any linear form $\sum_{1=1}^{n} \alpha_i x_i$, call the set of points $\{p_i | \alpha_i \neq 0\}$ 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 $P \cap q$:

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

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[2]) 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 in [6, 9] to prove lower bounds.

Some of our lower bounds derive from the following result of Brönnimann, Chazelle, and Pach [6].

[2]Whether or not the clusters used to answer a query must be disjoint depends on the semigroup. For example, if the semigroup is $(\mathbb{Z}, +)$, then the clusters must be disjoint; on the other hand, if the semigroup is $(\mathbb{Z}, \max)$, then the clusters can overlap arbitrarily.

Theorem 2.1 (Brönnimann, Chazelle, Pach). *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 in P in time t must satisfy the inequality $st^d = \Omega((n/\log n)^{d-(d-1)/(d+1)})$.*

Although lower bounds are known for offline hyperplane searching in the semigroup model [15, 19], we are unaware of any previous results for online hyperplane queries. In particular, Chazelle's lower bounds for simplex range searching [9] do not apply when the ranges are hyperplanes. We easily observe that for a set of points in general position, the smallest possible storage scheme, consisting of n singleton sets, allows hyperplane queries to be answered in constant "time". Much better lower bounds can be obtained by considering degenerate point sets.

Theorem 2.2. *Any storage scheme of size s that supports d-dimensional hyperplane queries in time t must satisfy the inequality $st^{d(d+1)/2} = \Omega(n^d)$ in the worst case.*

Proof (sketch): In two dimensions, we easily observe that the clusters in an optimal storage scheme must consist of maximal colinear subsets of the set of points. The two-dimensional lower bound follows from a construction of n points and $s+1$ lines with $\Omega(n^{2/3}s^{2/3})$ incidences discovered by Erdős; see [20] or [28, p. 177]. The higher-dimensional results follow from a natural generalization of the Erdős construction to higher dimensions [19, Lemmas 3.5 and 3.10]. We omit further details from this extended abstract. □

3 Partition Graphs

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. We put no constraints on the indegree. In addition, every non-leaf note v is 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.

```
Preprocess(p):
at each primal node $\nu$:
    for each query region $R \in \mathcal{R}_\nu$:
        if $R$ contains $p$:
            traverse the edge corresponding to $R$
at each dual node $\nu$:
    for each query region $R \in \mathcal{R}_\nu$:
        if $R$ intersects $p^*$:
            traverse the edge corresponding to $R$
at each leaf $\ell$:
    add $p$ to $P_\ell$
```
(a)

```
Query(h):
at each dual node $\nu$:
    for each query region $R \in \mathcal{R}_\nu$:
        if $R$ contains $h^*$:
            traverse the edge corresponding to $R$
at each primal node $\nu$:
    for each query region $R \in \mathcal{R}_\nu$:
        if $R$ intersects $h$:
            traverse the edge corresponding to $R$
at each leaf $\ell$:
    test/count $P_\ell$
```
(b)

Figure 1. Preprocessing and query algorithms for hyperplane searching.

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, some of our results require the query regions to be constant-complexity semialgebraic sets or constant-complexity polyhedra.

Given a partition graph, we preprocess a set P of points for hyperplane emptiness or counting 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 ν, we traverse the edges corresponding to the query regions in $\mathcal{R}_\nu$ that contain p. Whenever we reach a dual node ν, we traverse the edges corresponding to the query regions in $\mathcal{R}_\nu$ that intersect the dual hyperplane p^*. Note that 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. For each leaf ℓ, we maintain the set P_ℓ of the points that reach ℓ.[3] See Figure 1(a).

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

At each leaf ℓ, the query algorithm does something to the corresponding subset of points P_ℓ. If we are answering a counting query, then the algorithm adds the size of the P_ℓ to a running counter, which is output at the end of the search. If we are answering an emptiness query and P_ℓ is nonempty, then the algorithm immediately halts and reports that h contains at least one point; if the query algorithm reaches only leaves with P_ℓ empty, the algorithm reports that the hyperplane is empty.

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.[4] 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 more subsets of P with every query region. With each primal region $R \in \mathcal{R}_\nu$, we associate a subset P_R, which contains all the points that reach ν and lie inside R. With each dual region $R \in \mathcal{R}_\nu$, we associate two subsets P_R^+ and P_R^-, which contain all the points that reach ν 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 the hyperplane traverses, and the *preprocessing time* is the total number of edges traversed in the preprocessing phase. We do not consider the complexity of the query regions, the time required in practice to determine which query regions intersect a hyperplane or contain a point, the sizes of the subsets P_R, P_R^+, P_R^-,

[3]These sets are just a convenience for our analysis. In practice, it suffices to record just the size of each set (for counting queries) or whether or not each set is empty (for emptiness queries).

[4]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 algorithms, which have almost no effect on our analysis. Vertical halfspace queries can be handled either by standard perturbation techniques or by using a lower-dimensional data structure.

```
Preprocess(p):
at each primal node $\nu$:
    for each query region $R \in \mathcal{R}_\nu$:
        if $R$ contains $p$:
            add $p$ to $P_R$
            traverse the edge corresponding to $R$
at each dual node $\nu$:
    for each query region $R \in \mathcal{R}_\nu$:
        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 $\ell$:
    add $p$ to $P_\ell$
```

(a)

```
Query($h^\pm$):
at each dual node $\nu$:
    for each query region $R \in \mathcal{R}_\nu$:
        if $R$ contains $h^*$:
            test/count $P_R^\pm$
            traverse the edge corresponding to $R$
at each primal node $\nu$:
    for each query region $R \in \mathcal{R}_\nu$:
        if $h$ intersects $R$:
            traverse the edge corresponding to $R$
        else if $h^\pm$ contains $R$:
            test/count $P_R$
at each leaf $\ell$:
    test/count $P_\ell$
```

(b)

Figure 2. Modified preprocessing algorithm and query algorithm for halfspaces.

and P_ℓ, or the time required to maintain and test these subsets.

Lemma 3.1. *Every partition graph has the following properties.*

(a) If a set P_R, P_R^+, or P_R^- is tested during an emptiness query or counted during a counting query, then the query halfspace contains every point in that set.

(b) If a point lies in a query range, it also lies in one of the sets P_R, P_R^+, P_R^-, or P_ℓ counted during a counting query. Thus, the output of a counting query is never smaller than the number of points in the query range.

(c) An emptiness query never reports that a non-empty query range is empty.

(d) Exactly the same edges are traversed during an emptiness query for an empty hyperplane, a counting query for the same hyperplane, and a counting query for either of its halfspaces.

(e) Exactly the same edges are traversed during an emptiness query for an empty halfspace and a counting query for the same halfspace.

Proof: These properties either follow directly from definitions or are easily verified by induction. We omit details from this extended abstract. □

We say that a partition graph *supports* a particular type of range query for a given set of points if, after the points are preprocessed, the appropriate query algorithm is always correct. Even though we have a single preprocessing algorithm, a single partition graph need not support all types of queries. However, Lemma 3.1 implies that a partition graph that support one type of query may automatically support other types of queries, with the same or smaller worst-case query time. For example, if a partition graph supports hyperplane or halfspace counting queries, then it also supports hyperplane emptiness queries.

4 Hyperplane Emptiness Queries

4.1 "Trivial" Lower Bounds

Theorem 4.1. *Any partition graph that supports hyperplane emptiness, hyperplane counting, or halfspace counting queries has size $\Omega(n)$, preprocessing time $\Omega(n \log n)$, and worst-case query time $\Omega(\log n)$.*

Proof: It suffices to consider hyperplane emptiness queries, since Lemma 3.1 implies that the same lower bounds for the other two types of queries.

Let P be a set of n points all lying on a vertical line ℓ, and let H be a set of n hyperplanes normal to ℓ, with each hyperplane just above one of the points in P. Any partition graph that correctly answers hyperplanes queries for P must at least report 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 particular node ν of the partition graph if both the point and its partner reach ν. We say that a node ν *splits* a point p and a hyperplane h if p and h both reach ν but no edge out of ν is traversed by both p and h.

For any primal node ν and any query region $R \in \mathcal{R}_\nu$, there is at most one active point that lies in R whose partner does not intersect R. For any dual node ν and any query region $R \in \mathcal{R}_v$, there is at most one active point whose dual hyperplane misses R and whose partner's dual point lies in R. Thus, any node splits at most Δ points from their partners. Moreover, since every point in P must be split from its partner, there must be at least n query regions, so the partition graph must have 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 $n - \sum_{i=0}^{k-1} \Delta^{k+1} \geq n - \Delta^{k+2}$ 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

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

Moreover, 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 $\Omega(\log n)$. □

4.2 Space/Time Tradeoffs

The next result relates the complexity of range queries in the Fredman/Yao semigroup arithmetic model and counting queries in the partition graph model. Recall that a storage scheme is a "data structure" in the semigroup model.

Theorem 4.2. *Let* P *be a set of* n *points in* $\mathbb{R}^d$. *Given a partition graph of size* s *that supports hyperplane (resp. halfspace) counting queries in time* t, *we can construct a storage scheme of size* $O(s)$ *that supports hyperplane (resp. halfspace) queries 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. Each of these subsets defines a generator in the semigroup model. There are at most 3s of these generators: at most two for each of the s query regions, plus one for each of the $\leq s$ leaves.

Suppose the partition graph supports hyperplane counting queries. Lemma 3.1(b) implies that for any hyperplane h, the set of points $P \cap h$ is counted as the disjoint union of several sets P_ℓ, one for every leaf reached by the query algorithm. Since the query algorithm reaches at most t leaves, he set $P \cap h$ is the union of at most t clusters.

If the partition graph supports halfspace counting queries, then for any halfspace $h^\pm$, the points $P \cap h^\pm$ are counted as the disjoint 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 $(2+\Delta)t$ clusters. □

This theorem implies that lower bounds for range queries in the semigroup arithmetic model are also lower bounds for the corresponding counting queries in the partition graph model. Theorems 2.1 and 2.2 now immediately imply the following lower bounds.

Corollary 4.3. *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 queries for* P *in time* t *satisfies the inequality* $st^d = \Omega((n/\log n)^{d-(d-1)/(d+1)})$.

Corollary 4.4. *Any partition graph of size* s *that supports* d*-dimensional hyperplane counting queries in time* t *must satisfy 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 the points in its two halfspaces. Thus, any halfspace counting data structure also supports hyperplane emptiness queries. The following result implies that, in our model of computation, the reverse is almost true as well: Any partition graph that supports hyperplane emptiness queries also supports halfspace counting queries, at least over certain semigroups.

Theorem 4.5. *Let* P *be a set of* n *points in* $\mathbb{R}^d$. *Given a partition graph of size* s *that supports hyperplane emptiness queries in time* t, *we can construct a storage scheme of size* $O(s)$ *that supports halfspace queries in time* $O(t)$.

Proof: Fix a partition graph that supports hyperplane emptiness queries for a set P of n points. For each query region R, define the subsets P_R, P_R^-, and P_R^+ by the preprocessing algorithm in Figure 2.

Each of the subsets P_R, P_R^+, and P_R^- defines a cluster. There are at most 2s of these clusters, where s is the size of the partition graph. We claim that these clusters define a storage scheme that supports halfspace queries in time at most Δt, where t is the worst-case time for a hyperplane emptiness query. We emphasize that the partition graph is *not* required to support halfspace queries.

Let h be an empty hyperplane. If we run the counting query algorithm for the upper halfspace h^+, the output is the sum of the sizes of several clusters P_R and P_R^+. There are at most Δt such subsets, where t is the counting query time for the halfspace h^+, which by Lemma 3.1(d) is also the emptiness query time for the hyperplane h.

Even though the output of the counting query may be incorrect, we claim that the union of these Δt clusters is exactly $P \cap h^+$. Lemma 3.1(a) implies that every point in these clusters is in h^+. Lemma 3.1(b) implies that if some point $p \in P \cap h^+$ is not in one of these clusters, then there must be a leaf that is reached while preprocessing p and also reached while querying h^+. In that case, Lemma 3.1(d) implies that the same leaf is reached while querying h, which means an emptiness query for h would incorrectly report that h is non-empty.

Lower halfspaces are handled symmetrically. We do not need to consider halfspaces bounded by non-empty hyperplanes, since there is always a halfspace with empty boundary that contains the same points. $\square$

Note that the clusters that form $P \cap h^+$ are not necessarily disjoint. Thus, unlike the storage scheme constructed in Theorem 4.2, the storage scheme we construct here is not correct for all faithful semigroups. It is correct for the semigroups $(\mathbb{Z}, \max)$ and $(\{\text{true}, \text{false}\}, \vee)$, or any other semigroup in which the equation $x + x = x$ always holds.

The following lower bound now follows immediately from Theorem 2.1.

Corollary 4.6. *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 $st^d = \Omega((n/\log n)^{d-(d-1)/(d+1)})$.*

Again, Lemma 3.1 implies that the same lower bound applies to hyperplane counting queries. This improves the lower bound in Corollary 4.4 whenever $s = O(n^{d-1})$ or $t = \Omega(n^{2/d(d+1)})$. Lemma 3.1 also implies that the lower bound applies to halfspace counting queries, giving us an roundabout proof of Corollary 4.3.

4.3 Preprocessing/Query Tradeoffs

In a few cases, we can establish tighter tradeoffs by considering preprocessing time instead of space. These tradeoffs follow from offline lower bounds for Hopcroft's problem and its generalizations in an offline model of computation based on partition graphs [19, 18]. A *partitioning algorithm*, given a set of points and hyperplanes as input, constructs a partition graph, preprocesses the points, and queries all the hyperplanes (or equivalently, preprocesses the hyperplanes and queries the points).

Theorem 4.7. *Any partition graph that supports line queries in time t after preprocessing time $p = O(n^2)$ 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 $m = cp^{3/2}/n$, where c is a constant to be specified later. Note that $m = O(n^2)$, and since $p = \Omega(n \log n)$, we also have $m = \Omega(n^{1/2} \log^{1/2} n)$. There is a set of n points and m lines such that for any partition graph, the total time required to preprocess the n points and correctly answer the m line queries is at least $\alpha n^{2/3} m^{2/3} = \alpha c^{2/3} p$ for some positive constant α [19]. 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 $p/m = \Omega(n/p^{1/2})$. $\square$

Using similar techniques, we can improve this lower bound slightly in three or more dimensions.

Theorem 4.8. *Any partition graph, each of whose query regions is a constant complexity polyhedron, that supports d-dimensional plane queries in time t after preprocessing time $p = O(n^d)$ 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 $d = 3$, these lower bounds hold for arbitrary partition graphs.*

Proof: The tradeoffs follow from the offline lower bound $\Omega(n^{1-2/d(d+1)} m^{2/(d+1)} + n^{2/(d+1)} m^{1-2/d(d+1)})$, established in [19] for $d = 3$ and in [18] for $d > 3$, using exactly the same argument as Theorem 4.7. $\square$

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

Corollary 4.9. *Any partition graph, each of whose query regions is a constant-complexity polyhedron, that supports d-dimensional 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.*

Corollary 4.10. *Any partition graph, each of whose query regions is a constant-complexity polyhedron, that supports d-dimensional hyperplane emptiness queries in time $O(\operatorname{polylog} n)$, requires preprocessing time $\Omega(n^d/\operatorname{polylog} n)$ in the worst case.*

5 Halfspace Emptiness Queries

The space and time bounds for the best known 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 known halfspace counting data structure requires roughly $\Omega(n^d)$ space to achieve logarithmic query time [10, 25]; whereas, the same query time can be achieved

	Space	Preprocessing	Query Time	Source
$d = 2, 3$	$O(n)$	$O(n \log n)$	$O(\log n)$	[13, 3, 16]
$d \geq 4$	$O(n^{\lfloor d/2 \rfloor} / \log^{\lfloor d/2 \rfloor} n)$	$O(n^{\lfloor d/2 \rfloor} / \log^{\lfloor d/2 \rfloor - \varepsilon} n)$	$O(\log n)$	[27]
	$O(n)$	$O(n^{1+\varepsilon})$	$O(n^{1-1/\lfloor d/2 \rfloor} 2^{O(\log^* n)})$	[22]
	$O(n)$	$O(n \log n)$	$O(n^{1-1/\lfloor d/2 \rfloor} \operatorname{polylog} n)$	[27]
	$n \leq s \leq n^{\lfloor d/2 \rfloor}$	$O(s \operatorname{polylog} n)$	$O((n \operatorname{polylog} n)/s^{1/\lfloor d/2 \rfloor})$	[27]

Table 2. Best known upper bounds for halfspace emptiness queries.

with $o(n^{\lfloor d/2 \rfloor})$ space if we only want to know whether the halfspace is empty [27]. Table 2 lists the resource requirements for the best known halfspace emptiness data structures.

In this section, we derive lower bounds on the complexity of halfspace emptiness queries. To prove our results, we use the following reduction argument to transform hyperplane queries into halfspace queries in a higher-dimensional space. (A similar transformation is described in [17].)

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

$$\sigma_d(x_0, x_1, \ldots, x_d) = \\ (x_0^2, x_1^2, \ldots, x_d^2, \sqrt{2}x_0x_1, \sqrt{2}x_0x_2, \ldots, \sqrt{2}x_{d-1}x_d)$$

This map has the property that $\langle \sigma_d(p), \sigma_d(h) \rangle = \langle p, h \rangle^2$ for any $p, h \in \mathbb{R}^{d+1}$, 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 in homogeneous coordinates, 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 constant degree polynomial equalities and inequalities. Every Tarski cell has a constant number of connected components, and the intersection 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 5.1. *Let P be a set of points in $\mathbb{R}^d$, and let $\hat{P} = \sigma_d(P)$ be the lifted set of points in $\mathbb{R}^{d(d+3)/2}$. Given a semialgebraic partition graph $\hat{G}$ that supports halfspace emptiness queries over $\hat{P}$, we can construct a semialgebraic partition graph G that supports hyperplane emptiness queries over P, with (asymptotically) the same 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 noted above, the queries we consider 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 3.

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, but this is perfectly legal. If a query region $\hat{R}$ doesn't 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,

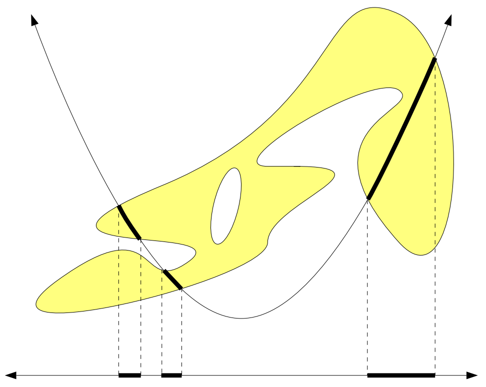

Figure 3. Each Tarski cell induces a constant number of lower-dimensional query regions.

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. $\square$

We emphasize that some restriction on the query regions is necessary to prove any nontrivial lower bounds. 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.

Corollary 5.2. *Any semialgebraic partition graph that supports* d*-dimensional halfspace emptiness queries, for any* $d \geq 2$, *has size* $\Omega(n)$, *preprocessing time* $\Omega(n \log n)$, *and worst-case query time* $\Omega(\log n)$.

Corollary 5.3. *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 5.4. *Any semialgebraic partition graph that supports* 5*-dimensional halfspace emptiness queries in time* t *after preprocessing time* $p = O(n^2)$ *satisfies the inequality* $pt^2 = \Omega(n^2)$ *in the worst case.*

Corollary 5.5. *Any semialgebraic partition graph that supports* 9*-dimensional halfspace emptiness queries in time* t *after preprocessing time* $p = O(n^3)$ *satisfies the inequalities* $pt^5 = \Omega(n^3)$ *and* $pt = \Omega(n^{5/3})$ *in the worst case.*

Theorem 5.1 does not imply better preprocessing/query tradeoffs for halfspace emptiness queries in higher dimensions, since the corresponding hyperplane results require polyhedral query regions. Marginally better lower bounds can be obtained in dimensions 14(!) and higher, using the same arguments as in Theorem 4.8, using the lower bounds for offline halfspace emptiness established in [18]. However, since these lower bounds are far from optimal, we omit further details.

The lower bounds in Corollary 5.2 are optimal when $d \leq 3$, and the bounds in Corollary 5.4 are optimal up to polylogarithmic factors.

6 Conclusions

We have presented 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 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 [25]. To answer a query for the intersection of k halfspaces, the points in the first halfspace are extracted as the (typically disjoint) union of several subsets, and then a $(k-1)$-halfspace query is performed on each subset.

With a few (important!) 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 [25], within polylogarithmic factors for halfspaces [27]—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 [6], and our preprocessing-query tradeoffs follow from lower bounds for offline range searching problems such as Hopcroft's problem [19, 18]. Any improvements to those lower bounds would improve our results as well.

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 linear programming queries [23, 7] and ray shooting queries [2, 12, 24, 27]?

Acknowledgments

I thank Pankaj Agarwal for suggesting this problem for study.

References

[1] P. K. Agarwal. Range searching. Report CS-1996-05, Dept. Comput. Sci., Duke Univ., 1996. http://www.cs.duke.edu/~pankaj/papers/range-survey.ps.gz. To appear in *The CRC Handbook of Discrete and Computational Geometry*, J. E. Goodman and J. O'Rourke, editors, CRC Press.

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

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

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

[5] M. de Berg, M. Overmars, and O. Schwarzkopf. Computing and verifying depth orders. *SIAM J. Comput.* 23:437–446, 1994.

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

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

[8] T. M. Chan. Output-sensitive results on convex hulls, extreme points, and related problems. *Discrete Comput. Geom.* 16:369–387, 1996.

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

[10] B. Chazelle. Cutting hyperplanes for divide-and-conquer. *Discrete Comput. Geom.* 9(2):145–158, 1993.

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

[12] B. Chazelle and J. Friedman. Point location among hyperplanes and unidirectional ray-shooting. *Comput. Geom. Theory Appl.* 4:53–62, 1994.

[13] B. Chazelle, L. J. Guibas, and D. T. Lee. The power of geometric duality. *BIT* 25:76–90, 1985.

[14] B. Chazelle and B. Rosenberg. Simplex range reporting on a pointer machine. *Comput. Geom. Theory Appl.* 5:237–247, 1996.

[15] B. Chazelle. Lower bounds for off-line range searching. *Discrete Comput. Geom.* 17(1):53–66, 1997.

[16] D. P. Dobkin and D. G. Kirkpatrick. Determining the separation of preprocessed polyhedra – a unified approach. *Proc. 17th Internat. Colloq. Automata Lang. Program.*, pp. 400–413. Springer-Verlag, Lecture Notes in Computer Science 443, 1990.

[17] R. Dwyer and W. Eddy. Maximal empty ellipsoids. *Internat. J. Comput. Geom. Appl.* 6:169–186, 1996.

[18] J. Erickson. New lower bounds for halfspace emptiness. *Proc. 37th Annu. IEEE Sympos. Found. Comput. Sci.*, pp. 472–481, 1996.

[19] J. Erickson. New lower bounds for Hopcroft's problem. *Discrete Comput. Geom.* 16:389–418, 1996.

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

[21] J. Matoušek. Efficient partition trees. *Discrete Comput. Geom.* 8:315–334, 1992.

[22] J. Matoušek. Reporting points in halfspaces. *Comput. Geom. Theory Appl.* 2(3):169–186, 1992.

[23] J. Matoušek. Linear optimization queries. *J. Algorithms* 14:432–448, 1993.

[24] J. Matoušek. On vertical ray shooting in arrangements. *Comput. Geom. Theory Appl.* 2(5):279–285, Mar. 1993.

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

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

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

[28] J. Pach and P. K. Agarwal. *Combinatorial Geometry.* John Wiley & Sons, New York, NY, 1995.

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

[30] R. E. Tarjan. A class of algorithms which require nonlinear time to maintain disjoint sets. *J. Comput. Syst. Sci.* 18:110–127, 1979.

[31] A. C. Yao. On the complexity of maintaining partial sums. *SIAM J. Comput.* 14:277–288, 1985.