Approximate Halfspace Range Counting*

Boris Aronov† Micha Sharir‡

September 9, 2009

Abstract

We present a simple scheme extending the shallow partitioning data structures of Matoušek, that supports efficient approximate halfspace range-counting queries in $\mathbb{R}^d$ with *relative* error ε. Specifically, the problem is, given a set P of n points in $\mathbb{R}^d$, to preprocess them into a data structure that returns, for a query halfspace h, a number t so that $(1-\varepsilon)|h\cap P| \leq t \leq (1+\varepsilon)|h\cap P|$. One of our data structures requires linear storage and $O(n^{1+\delta})$ preprocessing time, for any $\delta > 0$, and answers a query in time $O\left(\varepsilon^{-\gamma} n^{1-1/\lfloor d/2\rfloor} 2^{b\log^* n}\right)$, for any $\gamma > 2/\lfloor d/2\rfloor$; the choice of γ and δ affects b and the implied constants. Several variants and extensions are also discussed. As presented, the construction of the structure is mostly deterministic, except for one critical randomized step, and so are the query, storage, and preprocessing costs. The quality of approximation, for every query, is guaranteed with high probability. The construction can also be fully derandomized, at the expense of increasing preprocessing time.

*Work on this paper has been supported by a joint grant No. 2006/194 from the U.S.-Israeli Binational Science Foundation. Work by Boris Aronov has also been supported by NSF ITR Grant CCR-00-81964, NSF Grant CCF-08-30691, and NSA MSP Grant H98230-06-1-0016. Work by Micha Sharir has also been supported by NSF Grants CCF-05-14079 and CCF-08-30272, by Grants 155/05 and 338/09 from the Israel Science Fund, and by the Hermann Minkowski–MINERVA Center for Geometry at Tel Aviv University. A preliminary version of this work appeared as part of B. Aronov, S. Har-Peled, and M. Sharir, "On approximate halfspace range counting and relative epsilon-approximations," *Proceedings of the Twenty-Third Annual Symposium on Computational Geometry*, pp. 327–336, Gyeongju, South Korea, 2007.

†Department of Computer Science and Engineering, Polytechnic Institute of NYU, Brooklyn, NY 11201-3840, USA; *http://cis.poly.edu/~aronov*.

‡School of Computer Science, Tel Aviv University, Tel Aviv 69978 Israel and Courant Institute of Mathematical Sciences, New York University, New York, NY 10012, USA; *michas@post.tau.ac.il*.

1 Introduction

The problem studied in this paper is *approximate range counting*. In abstract terms, we are given a *range space* $(X, \mathcal{R})$, where X is a set of n *objects* and $\mathcal{R}$ is a collection of subsets of X, called *ranges*. In a typical geometric setting, X is a finite subset of some infinite ground set U (e.g., $\mathbb{R}^d$), and $\mathcal{R} = \{R \cap X \mid R \in \mathcal{R}_U\}$, where $\mathcal{R}_U$ is a collection of subsets (*ranges*) of U of some simple shape (such as halfspaces). To simplify the notation, we do not distinguish between $\mathcal{R}$ and $\mathcal{R}_U$. The goal is to preprocess X into a data structure that supports efficient queries of the form: Given $R \in \mathcal{R}_U$, compute a number t such that

$$(1-\varepsilon)|X \cap R| \le t \le (1+\varepsilon)|X \cap R|.$$

Here the *relative error* ε, $0 < \varepsilon < 1$, is either fixed and available during preprocessing, or not known in advance, but specified as part of the query. We refer to such an estimate t as an *ε-approximate count* of $X \cap R$.

Notice that the problem becomes more challenging as $|X \cap R|$ decreases. At the extreme, when $|X \cap R| < 1/\varepsilon$, we must produce the count *exactly*. In particular, we need to be able to detect without any error the *empty ranges*, i.e., those satisfying $X \cap R = \varnothing$. Thus approximate range counting, in the above sense, is at least as hard as *range emptiness* detection.

We make the standard assumption that the range space $(X, \mathcal{R})$ (or, in fact, $(U, \mathcal{R}_U)$) has *finite* (i.e., independent of n) *VC-dimension* δ which is indeed the case in many geometric applications; see [10, 19, 25, 27] for definitions and more details.

Epsilon-approximations. A standard and general technique for tackling the approximate range counting problem is to use ε-approximations. An *(absolute-error) ε-approximation* for $(X, \mathcal{R})$ is a subset $B \subset X$ such that, for each $R \in \mathcal{R}$,

$$\left| \frac{|B \cap R|}{|B|} - \frac{|X \cap R|}{|X|} \right| < \varepsilon. \tag{1}$$

As shown by Vapnik and Chervonenkis [31] (see also [10, 25, 27]), there always exist absolute-error ε-approximations of size $\frac{c\delta}{\varepsilon^2} \log \frac{\delta}{\varepsilon}$, where c is an absolute constant. As a matter of fact, any random sample of these many elements of X is an ε-approximation with constant probability. More precisely, such a sample of size $\frac{c\delta}{\varepsilon^2} \log \frac{\delta}{\varepsilon} + \frac{c}{\varepsilon^2} \log \frac{1}{q}$, for a sufficiently large absolute constant c, is an ε-approximation with probability at least $1 - q$. See [10, 11, 19, 26] for more details. This bound was later improved to $\frac{c\delta}{\varepsilon^2} + \frac{c}{\varepsilon^2} \log \frac{1}{q}$, by Talagrand [30] and by Li *et al.* [22]; see also [16].

Absolute-error ε-approximations are not exactly what is needed for approximate range counting. That is, suppose we are given such an approximation B. For a range $R \in \mathcal{R}$, we can compute (say, by brute force) $|B \cap R|$, and return $|B \cap R| \cdot |X|/|B|$ as an estimate of $|X \cap R|$. By (1), we have $\big||B \cap R| \cdot |X|/|B| - |X \cap R|\big| < \varepsilon |X|$, but we want the error to be at most $\varepsilon |X \cap R|$. If $|X \cap R|$ is large, say, at least $|X|/2$, we can replace ε by $\varepsilon/2$, and guarantee the desired relative error. But if $|X \cap R|$ is small, the error is much larger than what we want.

For this reason, we would like to construct a *relative-error ε-approximation* set for $(X, \mathcal{R})$, which should be a subset $A \subset X$ satisfying, for each $R \in \mathcal{R}$,

$$(1-\varepsilon)\frac{|X \cap R|}{|X|} \le \frac{|A \cap R|}{|A|} \le (1+\varepsilon)\frac{|X \cap R|}{|X|}. \tag{2}$$

However, this "definition" suffers from the same syndrome as the definition of approximate range counting; that is, as $|X \cap R|$ shrinks, the absolute precision of the approximation has to increase. At the extreme, when $A \cap R = \varnothing$, $X \cap R$ must also be empty (assuming $\varepsilon < 1$); in general, we cannot guarantee this property, unless we take $A = X$, which defeats the whole purpose of using small ε-approximations to speed up approximate counting.

For this reason, we refine the definition as follows: a *relative-error* (p, ε)*-approximation* (or a *relative* (p, ε)*-approximation*, for short) is a subset $A \subset X$ that satisfies (2) for each $R \in \mathcal{R}$ with $|R| \geq pn$, where $0 < p < 1$ is another fixed parameter. As noted by Har-Peled [16] (see also [17]), it follows from the result of Li *et al.* [22] (see also [14, 18, 28]) that there exist subsets with this property of size $\frac{c\delta}{\varepsilon^2 p} \log \frac{1}{p}$, where c is an absolute constant. As a matter of fact, any random sample of $\frac{c\delta}{\varepsilon^2 p}(\log \frac{1}{p} + \log \frac{1}{q})$ elements of X is a relative (p, ε)-approximation with probability at least $1 - q$ [16, 22]. The construction can also be derandomized, using the recent technique of Har-Peled [16]; see Section 3.1 for more details.

To appreciate the above bound on the size of relative (p, ε)-approximations, it is instructive to observe that, for a given parameter p, any absolute-error (εp)-approximation A will approximate "large" ranges (of size at least pn) to within *relative* error ε, as is easily checked, so it is a relative (p, ε)-approximation. However, the Vapnik-Chervonenkis bound on the size of A in this case, namely, $\frac{c\delta}{\varepsilon^2 p^2} \log \frac{\delta}{\varepsilon p}$, or even the improved bound of Li *et al.* and others [16, 22, 30], namely $\frac{c\delta}{\varepsilon^2 p^2}$, is larger by roughly a factor of $1/p$ than the bound of [22] stated above.

Another related notion, introduced by Brönnimann *et al.* [9], is that of *sensitive* ε*-approximation*. Specifically, given a range space $(X, \mathcal{R})$ of finite VC-dimension δ, a subset $A \subseteq X$ is a sensitive ε-approximation if for every range $R \in \mathcal{R}$ we have

$$\left| \frac{|X \cap R|}{|X|} - \frac{|A \cap R|}{|A|} \right| \leq \frac{\varepsilon}{2} \left(\left(\frac{|X \cap R|}{|X|} \right)^{1/2} + \varepsilon \right).$$

As shown in [16, Lemma 6.2.8] and [17, Theorem 2.12], (i) a sensitive $\varepsilon\sqrt{p}$-approximation is also a relative (p, ε)-approximation, and (ii) a random sample of $O\left(\frac{\delta}{(\varepsilon')^2} \log \frac{1}{\varepsilon'}\right)$ elements of X is a sensitive ε'-approximation with constant positive probability. Combining (i) and (ii) gives a roundabout way of obtaining relative (p, ε)-approximations with almost the same bound on their size as in [16, 17] (with $\log \frac{1}{p}$ replaced by $\log \frac{1}{\varepsilon p}$). The analysis in [9] also gives an efficient (albeit complicated) *deterministic* algorithm for computing sensitive approximations.

The existence of a relative (p, ε)-approximation A provides a simple mechanism for approximate range counting, in the manner outlined above; that is, for a range R, count $A \cap R$ exactly, say, by brute force in $O(|A|)$ time, and output $|A \cap R| \cdot |X|/|A|$ as an ε-approximate count of $X \cap R$. However, this will work only for ranges of size at least pn. The main contribution of this paper is to show that an appropriate incorporation of relative (p, ε)-approximations into standard range searching data structures yields a procedure for approximate range counting that works, quite efficiently, for ranges *of any size*.

Exact range counting. The motivation for seeking approximate range counting techniques is that exact range counting is (more) expensive. For instance, consider the classical *halfspace range counting* problem [24], which is the exact analog of the main specific problem studied in this paper. Here, for a point set of size n in $\mathbb{R}^d$, for $d > 3$, the best known algorithm for exact range counting

Problem	Storage	Preprocessing	Query time	Source
Reporting	$n \log\log n$	$n \log n$	$n^{1-1/\lfloor d/2 \rfloor} + k$	[23]
Emptiness	n	$n^{1+\delta}$	$n^{1-1/\lfloor d/2 \rfloor} 2^{O(\log^* n)}$	[23]
Approx. counting	$\varepsilon^{-2} n \log n$	$\varepsilon^{-2} n^{1+\delta}$	$\varepsilon^{-2} n^{1-1/\lfloor d/2 \rfloor} 2^{c' \log^* n} \log n$	[4]
	n	$n^{1+\delta}$ / $n \log n$	$\varepsilon^{-\gamma} n^{1-1/\lfloor d/2 \rfloor} \log^{\beta} n$	Theorem 3.1
	n	$n^{1+\delta}$	$\varepsilon^{-\gamma} n^{1-1/\lfloor d/2 \rfloor} 2^{b \log^* n}$	Theorem 3.2

Table 1: Halfspace range searching results for $d > 3$; big-Oh symbol omitted; k is the number of points reported in the reporting query; $\delta > 0$ is an arbitrary but fixed constant; γ is an arbitrary constant in the range $(2/\lfloor d/2 \rfloor, 2)$; β and b depend on d and γ. We include only data structures with near-linear space and preprocessing costs.

with near-linear storage guarantees $O(n^{1-1/d})$ query time [24]. In contrast, our results (reviewed in detail below) reduces the query cost to roughly $O(n^{1-1/\lfloor d/2 \rfloor})$ (ignoring the dependence on ε), about the same cost as that of answering *halfspace range emptiness* queries [23]. Recall that we cannot do better than that, since halfspace range emptiness is a special case of our problem.

For completeness, we recall the results of [23] for halfspace range reporting and emptiness queries, for a set P of n points in $\mathbb{R}^d$, for $d \geq 4$ (refer also to Table 1):

Halfspace range reporting. P can be preprocessed in deterministic $O(n \log n)$ time, into a data structure of size $O(n \log\log n)$, so that a halfspace range reporting query can be answered in time $O(n^{1-1/\lfloor d/2 \rfloor} \log^c n + k)$, where k is the output size, and c is a parameter, depending on the dimension.

Halfspace range emptiness detection. P can be preprocessed in deterministic $O(n^{1+\delta})$ time, for any $\delta > 0$, into a data structure of linear size, so that a halfspace range emptiness query can be answered in time $O(n^{1-1/\lfloor d/2 \rfloor} 2^{c' \log^* n})$, where c' is a parameter, depending on the dimension.

Note that the best known *lower* bounds on the exact halfspace range searching (counting, reporting, or emptiness), as summarized in [2], do not match the best known algorithms, as listed above, so it is better to compare our approximation results to those of existing algorithms rather than to lower bounds.

Other notions of approximate range searching. In our approach, the goal is to approximate the count of points in a range (halfspace, that is) up to some relative error, but we are not allowed to approximate the range itself. In some recent work by Arya *et al.* [6–8], da Fonseca [15], and Chazelle *et al.* [12], approximate range counting is interpreted differently, in that one seeks an exact count in a range that closely approximates the input range, according to some geometric error measure. The more recent studies also address the case where the ranges to be approximated are halfspaces.

Alternative recent solutions. Two recent papers address the approximate range counting problem, and achieve improvements similar to ours. The first result is due to Aronov and Har-Peled [4], who reduce this problem to range emptiness, by performing binary search on the size $|X \cap R|$ for the given range R, until the desired relative error is attained. Each decision step in the search is made by accessing $O\left(\frac{1}{\varepsilon^2} \log n\right)$ different range emptiness structures on certain random samples of X. This technique is a general reduction from approximate range searching to range emptiness testing.

In the revised version [4], the algorithm answers a query in time $O\left(\frac{1}{\varepsilon^2}\log n\right) Q_{\text{empty}}(n)$, where $Q_{\text{empty}}(n)$ is the time to answer a range emptiness query. The storage is $O\left(\frac{1}{\varepsilon^2}\log n\right) S_{\text{empty}}(n)$, and the preprocessing is $O\left(\frac{1}{\varepsilon^2}\log n\right) T_{\text{empty}}(n)$, where $S_{\text{empty}}(n)$ and $T_{\text{empty}}(n)$ are the storage requirements and preprocessing time for the range emptiness data structure, respectively. All bounds apply with high probability. See [4, Theorem 5.6] for details.

Matoušek's data structure [23] mentioned above, combined with the technique of [4] yields (see also Table 1) $Q(n) = O(\varepsilon^{-2} n^{1-1/\lfloor d/2\rfloor} 2^{c'\log^* n}\log n)$, $S(n) = O(\varepsilon^{-2} n \log n)$, and $T(n) = O(\varepsilon^{-2} n^{1+\delta})$, for any $\delta > 0$.

Another approach is presented by Kaplan and Sharir [21], who exploit a general technique of Cohen [13] for estimating the number of data objects in a range R of a larger set X. In this approach, one assigns to each data object of X, independently, a random *weight*, drawn from an exponential distribution with density e^{-x}, sorts the objects by their weights into a random permutation, and then finds the *minimum rank* in that permutation of the objects in the query range R. As in the technique of Aronov and Har-Peled, one then repeats this experiment $O(\frac{1}{\varepsilon^2}\log n)$ times,[1] computes the average μ of the *weights* of the minimum elements, and approximates $|R|$ by $1/\mu$.

To apply this machinery to approximate halfspace range counting, one needs a data structure that preprocesses the given set X of points, and a given random permutation thereof, into a data structure that can answer *halfspace-minimum range queries* efficiently: Given a query halfspace h, find the point of X of minimum rank among those contained in h. Kaplan and Sharir present such structures for halfspaces in $\mathbb{R}^3$ (a revised version [20] extends it to any dimension). The solutions of [20, 21] cater to the situation where we want fast (say, logarithmic or polylogarithmic) query time, allowing the storage and preprocessing costs to grow. Specifically, their algorithm requires $O(n^{\lfloor d/2\rfloor}\log^{2-\lfloor d/2\rfloor})$ storage and $O(n^{\lfloor d/2\rfloor}\log^{2-\lfloor d/2\rfloor})$ expected preprocessing time, and answers an approximate halfspace counting query in $O(\varepsilon^{-2}\log^2 n)$ expected time.

Finally, we mention the recent works of Afshani and Chan [1], and of Har-Peled and Sharir [17], which give efficient algorithms for approximate halfspace range counting in two and three dimensions.

Our results. In this paper, we present an alternative, comparably efficient, and somewhat improved solution for the approximate range counting problem, focusing mainly on halfspace ranges in $\mathbb{R}^d$, $d \geq 4$. Whereas the algorithm of Aronov and Har-Peled uses a range emptiness procedure as a black box, we examine the inner workings of such a procedure (or, more precisely, of a *shallow range reporting procedure*, which has comparable performance), and turn it into an approximate counting procedure. Informally, the range emptiness/reporting data structures of Matoušek [23] consist of a *partition tree*, whose nodes store certain canonical subsets of X, and which has the property that a query with a range that is *shallow* at a node v (i.e., one that contains only a few points of the subset stored at v; see below for a more precise definition) visits only a small number of children of v. When the procedure realizes that the query visits too many children, it stops and reports that the range cannot be shallow. For emptiness queries, this immediately implies that the range is not empty. For reporting queries, one can then afford to perform the reporting by brute force, knowing that the output size is large enough and thus commensurable with the size of the subset of X stored at v.

In contrast, our solution exploits the fact that the range is *deep* (that is, not shallow), to invoke an auxiliary mechanism that approximates its size. Our main auxiliary mechanism is to use a

[1] In both techniques, this is a consequence of using Chernoff bounds to guarantee high probability of success.

relative approximation, as discussed earlier. In this manner, we derive two variants of our general approach. The first algorithm uses $O(n)$ storage, $O(n^{1+\delta})$ preprocessing time, for any $\delta > 0$ (which reduces to $O(n \log n)$ for certain choices of parameters), and answers an approximate halfspace range counting query in $\mathbb{R}^d$ in time $O\left(\varepsilon^{-\gamma} n^{1-1/\lfloor d/2 \rfloor} \operatorname{polylog} n\right)$, where γ can be chosen arbitrarily from the interval $(2/\lfloor d/2 \rfloor, 2)$; the choice of γ and δ affects the implied constants and the power of the logarithm in the query time. An important feature of this implementation is that the storage and preprocessing costs are *independent of* ε, and the dependence of the query time on ε is improved over those in the previous approaches.

A slight potential weakness of this solution is that the query time bound, ignoring its dependence on ε, involves a polylogarithmic factor and is thus comparable with the overhead term in the bound for Matoušek's halfspace range *reporting* query algorithm [23], whereas the query time in the solution of Aronov and Har-Peled [4] is expressed in terms of the cost of Matoušek's halfspace range *emptiness* query algorithm, which is $O\left(n^{1-1/\lfloor d/2 \rfloor} \cdot 2^{O(\log^* n)}\right)$ [23]. See Table 1. On one hand, this replaces the polylogarithmic factor in our time bound by the smaller factor $2^{O(\log^* n)}$, but, on the other hand, in the algorithm of [4] one has to multiply this bound by $O\left(\frac{1}{\varepsilon^2} \log n\right)$, making the dependence on ε worse.

Nevertheless, our second implementation demonstrates that the fine-tuning done in [23] to achieve the improved bound for emptiness queries can also be carried out in our context, leading to an algorithm that uses linear storage and $O(n^{1+\delta})$ preprocessing time, for any $\delta > 0$, and answers a query in time $O\left(\varepsilon^{-\gamma} n^{1-1/\lfloor d/2 \rfloor} \cdot 2^{O(\log^* n)}\right)$, where γ can be chosen anywhere in the same interval as above. This bound compares favorably with the one in [4], both in terms of the dependence on ε and the logarithmic or sublogarithmic factors. Moreover, the storage used by both solutions is $O(n)$, *independent* of ε, which is a significant improvement over the previous results.

The general technique that we propose is sufficiently modular, so as to support various extensions and variants. One interesting variant is a data structure that answers efficiently halfspace *range minimum* queries, with respect to a random permutation of the input set, of the sort that is needed for the technique of [13, 20, 21] described above (see Section 3.5). Another variant is a data structure where ε need not be pre-specified, and can be part of the query; in contrast, the "competing" structures mentioned above have to be built with the prior knowledge of the value of ε. Another recent application of our general approach was recently described by Sharir and Shaul [29].

The analysis of our implementations involves somewhat tedious calculations. In the hope of making the paper more readable, we delegate these calculations to the appendix.

2 The General Technique

In this section we describe, in somewhat more detailed but still high-level terms, our general technique. In the next section we present two concrete implementations of this machinery, resulting in two algorithms with similar, albeit slightly different, performance bounds, as stated above. As already mentioned, hereafter we focus on the case of halfspace ranges in $\mathbb{R}^d$, $d \geq 4$, rather than more general range spaces. To conform with the notation in the existing range-searching literature, we use henceforth P rather than X to denote the input point set.

A note on the randomized behavior of our algorithms. Before beginning the description of our technique, it is important to clarify a critical issue concerning the randomization that we use. The approach is essentially to construct a *shallow partition tree* on P, in the style of Matoušek [23]

(see below), and then attach to each node v of the tree a relative (p_v, ε_v)-approximation of the set P_v associated with v, for appropriate parameters p_v, ε_v. Constructing these sets deterministically is feasible, using the procedure in [9], but this construction is fairly complicated, and causes some degradation in the preprocessing cost. We therefore prefer to construct these approximations as random samples, of the appropriate size, from the respective sets P_v. Verifying that such a sample is indeed a relative (p_v, ε_v)-approximation is a non-trivial task which we do not know how to perform efficiently. So we adopt an approach in which we just sample the subsets and trust each of them to be an approximation of the right kind. While this will be the case on average, some of these samples may fail to satisfy this property, and we cannot guarantee high probability for overall success (i.e., that all samples are approximations of the desired kind), unless we increase the sample sizes, again causing some degradation in the algorithm performance.

Nevertheless, as we show later, in Section 3.3, things work well, with high probability, even in our lax and "careless" approach. Specifically, as we will show, even though some of the samples may not produce the desired level of accuracy, the excessive errors that they produce tend to cancel each other, and result in an overall good accuracy, and this holds, with high probability, for every query halfspace.

The details of this analysis are technical and somewhat involved, so we prefer to present the algorithm under the assumption that all our samples are indeed relative approximations with the desired parameters, and only later resolve the potential discrepancy between this ideal situation and our actual implementation. Of course, this becomes a non-issue if one is willing to invest a bit more in preprocessing, storage, and query costs, so as to ensure, either deterministically or with high probability, that all the samples are indeed relative approximations with the correct parameters.

We now present the high-level version of our algorithm. We recall the following result of Matoušek, where a *t-shallow halfspace* is one that contains fewer than t points of the set P.

Theorem 2.1 (Partition Theorem for Shallow Halfspaces [23])**.** *For any positive integer parameter $r < n$, there exists a partition of P into $r/2 \leq k \leq r$ subsets $P_1, P_2, \ldots, P_k$, where, for each i, $n/r \leq |P_i| \leq 2n/r$, and P_i is enclosed in a simplex Δ_i, such that any hyperplane that bounds an (n/r)-shallow halfspace crosses at most $\mu(r) = O(r^{1-1/\lfloor d/2 \rfloor})$ simplices Δ_i, in four and higher dimensions, and at most $\mu(r) = O(\log r)$ simplices, in the plane and in 3-space. Such a partition can be constructed in time $O(n^{1+\delta})$, for any $\delta > 0$. Moreover, there exists a constant $\xi = \xi(d) > 0$, so that if $r \leq n^{\xi}$, then the partition and the bounding simplices can be constructed in $O(n \log r)$ time.*

For a fixed set P and a choice of parameter r at every interior node (which may vary from node to node), Theorem 2.1 induces, in a natural way, a tree $T = T(P)$, called a *shallow partition tree* of P, which is constructed as follows. Its root stores the entire P, and some bounding simplex Δ of P. The root has k children, each storing one of the sets P_i and its bounding simplex Δ_i. Then the theorem is used recursively on each child, possibly with different values of r, to construct the grandchildren of the root, and so on, stopping when we reach nodes whose associated sets have size smaller than some specific threshold.

Denote by P_v the subset of P stored at a node v of T, and by r_v the parameter r used when applying Theorem 2.1 to P_v. Our proposed approximate range counting data structure is, effectively, an augmented shallow partition tree. We store some additional information at each node v; in the main implementations that we present, this is a relative $(1/r_v, \varepsilon/2)$-approximation A_v of P_v.

Querying with a halfspace h proceeds as follows: When visiting a node v, if the boundary ∂h of h meets many (more than $\mu(r_v)$) simplices of the set $S_v := \{\Delta_i\}$ of the partition at v, or if it fully contains one of these simplices, it cannot be $(|P_v|/r_v)$-shallow with respect to P_v; that is, $|h \cap P_v| \geq |P_v|/r_v$. This makes it easier to answer an approximate range counting query for P_v, e.g., by counting $h \cap A_v$ instead. Otherwise, we recursively obtain an ε-approximate count at all the children of v whose simplex is crossed by ∂h, and return the sum of the answers, which is easily seen to be an ε-approximate count of $|h \cap P_v|$. See Algorithm 1 for the pseudocode.

Algorithm 1 Pseudocode of our main query algorithm

```
1:  function ApxCount(halfspace h, node v of an augmented shallow partition tree, ε)
2:      if v is a leaf node then return LeafNodeApxCount(h, v, ε).
    ▷ S = S_v is the set of simplices associated with children of v.
    ▷ r = r_v is the partition parameter at v.
3:      if ∂h crosses at most μ(r) simplices of S and no simplex is fully contained in h
4:      then                                          ▷ Shallow halfspace, recurse.
5:          answer ← 0.
6:          for all children ξ of v whose bounding simplex is crossed by ∂h do
7:              Add ApxCount(h, ξ, ε) to answer.
8:          return answer.
9:      else                                          ▷ Deep halfspace, answer locally.
10:         return DeepApxCount(h, v, ε).
```

It remains to specify, for each node v, the parameter r_v used at v, the threshold $n_0(\varepsilon)$ for the size of P_v, below which v becomes a leaf, and three subroutines:

1. the implicit subroutine (that we call SearchSim(h, v)) used in lines 3 and 6 of the algorithm to determine how many, and which, of the simplices of S_v are met by the hyperplane ∂h and whether any of the simplices of S_v are contained in h,
2. procedure LeafNodeApxCount(h, v, ε) that directly estimates the count for h at a leaf v of the tree, and
3. procedure DeepApxCount(h, v, ε) that estimates the count of a deep range h at an interior node v, using the relative approximation set or any other appropriate auxiliary structure.

Let $Q_{\text{sim}}(k)$, $Q_{\text{leaf}}(n, \varepsilon)$, and $Q_{\text{deep}}(n, \varepsilon)$ be upper bounds on the running times of these three respective operations, where k (with $r/2 \leq k \leq r$) is the number of simplices to test against, n is the size of the point set associated with the current node, and ε is the approximation parameter. We obtain the following recurrences for the query time $Q(n, \varepsilon)$, storage $S(n, \varepsilon)$, and preprocessing time $T(n, \varepsilon)$ of our data structure. The parameter $r = r_v$ is the one used at the current node of the tree; in our implementations it is a function of n (and possibly ε). For simplicity, we use $k = r$ in the recurrences, for the maximum possible number of children of a node (but our analysis also applies to any valid value of k). We use S_{sim}, T_{sim} (S_{leaf}, T_{leaf} and S_{deep}, T_{deep}) to denote the storage and preprocessing time required by SearchSim (LeafNodeApxCount and DeepApxCount,

respectively). T_{part} is the time needed to construct the partition at v.

$$Q(n,\varepsilon) \leq \begin{cases} Q_{\text{sim}}(r) + \max\{Q_{\text{deep}}(n,\varepsilon),\ \mu(r)Q(2n/r,\varepsilon)\}, & \text{if } n > n_0(\varepsilon), \\ Q_{\text{leaf}}(n,\varepsilon), & \text{otherwise,} \end{cases} \tag{3}$$

$$S(n,\varepsilon) \leq \begin{cases} S_{\text{sim}}(r) + S_{\text{deep}}(n,\varepsilon) + \sum_{i=1}^{r} S(n_i,\varepsilon), & \text{if } n > n_0(\varepsilon), \\ S_{\text{leaf}}(n,\varepsilon), & \text{otherwise,} \end{cases} \tag{4}$$

and

$$T(n,\varepsilon) \leq \begin{cases} T_{\text{part}}(n,r) + T_{\text{sim}}(r) + T_{\text{deep}}(n,\varepsilon) + \sum_{i=1}^{r} T(n_i,\varepsilon), & \text{if } n > n_0(\varepsilon), \\ T_{\text{leaf}}(n,\varepsilon), & \text{otherwise,} \end{cases} \tag{5}$$

where $n_i \leq 2n/r$ for each i, and $\sum_{i=1}^{r} n_i = n$. Note that, since T_{sim}, S_{sim}, and Q_{sim} are the respective preprocessing time and storage of, and query time in, a set of $O(r)$ simplices, they are functions of r only.

3 Two Concrete Implementations

To recap, in order to obtain concrete implementations of the data structure, we have to supply specific choices of the parameters r_v and $n_0(\varepsilon)$, as well as of the three routines SearchSim, LeafNodeApxCount, and DeepApxCount.

There are a number of ways to choose the parameters and to implement these procedures. We focus on two of them; the first is simpler, more naive, and has slightly poorer performance, while the second is more sophisticated with slightly better performance. Roughly speaking, the first implementation has performance comparable with that of the halfspace range *reporting* procedure of [23], whereas the second implementation has performance comparable with that of the halfspace range *emptiness* procedure of [23].

3.1 First implementation

Recall that we are implementing approximate range counting queries for halfspaces in $\mathbb{R}^d$, where $d > 3$. For each node v, put $n_v := |P_v|$. Here we choose $r_v := n_v^{\alpha'}$, for some $0 < \alpha' < \alpha$, whose concrete choice will be discussed below; $\alpha := 1 - 1/\lfloor d/2 \rfloor$ denotes the exponent appearing in the definition of $\mu(r)$ in Theorem 2.1. We store at v a relative $(1/r_v, \varepsilon/2)$-approximation A_v, of size $\frac{cr_v}{\varepsilon^2} \log r_v$, for some constant $c = c(d) > 0$, which we obtain by taking a random sample of these many points from P_v. As follows from [16, 22], such a sample is a relative $(1/r_v, \varepsilon/2)$-approximation of the desired kind with probability at least $1 - 1/r_v^b$, where $b = b(c)$ is a linearly increasing function of c. In Section 3.3 we will see how to boost up the overall success probability, making the failure probability polynomially small in n. For now, we simply assume that A_v is indeed an approximation of the required type and size.

In this implementation, we use brute force for two of the three subroutines. We implement SearchSim by simply iterating over all simplices, and selecting those that ∂h crosses, stopping after collecting more than $\mu(r_v)$ of them, or after encountering a simplex that is fully contained in h. The cost is $O(r_v) = O(n_v^{\alpha'})$. We implement LeafNodeApxCount by iterating over P_v and counting $h \cap P_v$ explicitly, at the cost of $O(n_v)$.

We implement DeepApxCount recursively, by calling ApxCount itself, on an auxiliary data structure constructed for A_v as the input set, with error parameter $\varepsilon/3$. If we were to count $h \cap A_v$ exactly, we would have returned $|h \cap A_v| \cdot n_v/|A_v|$ as an $(\varepsilon/2)$-approximate count of $h \cap P_v$. The recursion, though, only returns a relative $(\varepsilon/3)$-approximation of that latter count, which is thus an ε-approximation of the desired count,[2] for $\varepsilon < 1$, as is easy to verify (cf. (2)).

In order for this implementation to work efficiently, we need to impose some restrictions on the choice of parameters. Specifically, we first insist, for technical reasons that arise in the analysis of the storage requirements (see the appendix), that

$$|A_v| = \frac{cr_v}{\varepsilon^2} \log r_v \leq \frac{n_v}{k \log^3 \log n_v},$$

for some constant k. Intuitively, this requires that the size of A_v be small enough (albeit not drastically smaller) compared to that of P_v. By the choice of r_v, this is equivalent to $n_v^{1-\alpha'} \geq \frac{kc}{\varepsilon^2} \log n_v^{\alpha'} \log^3 \log n_v$, which holds if we choose $n_v \geq (\frac{c'}{\varepsilon^2} \log \frac{1}{\varepsilon} \log^3 \log \frac{1}{\varepsilon})^{1/(1-\alpha')}$, for an appropriate multiple c' of c. We thus set

$$n_0(\varepsilon) := \left(\frac{c'}{\varepsilon^2} \log \frac{1}{\varepsilon} \log^3 \log \frac{1}{\varepsilon} \right)^{1/(1-\alpha')}. \tag{6}$$

Our goal is to make the query time satisfy

$$Q(n, \varepsilon) \leq F(\varepsilon) n^{\alpha} \log^{\beta} n, \tag{7}$$

for some parameter β and function $F(\varepsilon)$ whose specific choices will be discussed shortly. In particular, we want $Q_{\text{leaf}}(n, \varepsilon)$ to satisfy this bound (the bound on Q_{deep} will follow from the recursion—see below). That is, we require

$$n_0(\varepsilon) \leq c'' F(\varepsilon) n_0(\varepsilon)^{\alpha} \log^{\beta} n_0(\varepsilon),$$

where $c'' \geq 1$ is some constant. This will hold by simply requiring $n_0(\varepsilon) = F(\varepsilon) n_0(\varepsilon)^{\alpha}$; that is

$$F(\varepsilon) := n_0(\varepsilon)^{1-\alpha} = \left(\frac{c'}{\varepsilon^2} \log \frac{1}{\varepsilon} \log^3 \log \frac{1}{\varepsilon} \right)^{(1-\alpha)/(1-\alpha')}. \tag{8}$$

Recall that $0 < \alpha' < \alpha = 1 - 1/\lfloor d/2 \rfloor$. Hence $F(\varepsilon)$ is approximately of the form $1/\varepsilon^{\gamma}$, where $\gamma := 2(1-\alpha)/(1-\alpha')$ satisfies $2/\lfloor d/2 \rfloor < \gamma < 2$. Note that γ approaches its upper (resp., lower) bound as α' approaches α (resp., 0).

Query time. Once α' is fixed, the recurrence for $Q(n, \varepsilon)$ becomes

$$Q(n, \varepsilon) \leq \begin{cases} O(n^{\alpha'}) + \max \left\{ Q\left(\frac{n}{k \log^3 \log n}, \frac{\varepsilon}{3} \right),\ \mu(n^{\alpha'}) Q(2n^{1-\alpha'}, \varepsilon) \right\}, & \text{if } n > n_0(\varepsilon), \\ O(n), & \text{otherwise.} \end{cases}$$

We show in the appendix that the recurrence solves to the bound in (7), for an appropriate choice of $\beta = \beta(\alpha')$.

[2] Recall our convention of assuming that all our samples are indeed relative approximations of the desired kind. We will continue to highlight other steps of the algorithm and its analysis which make this assumption; they all work correctly, with high probability, in view of the analysis in Section 3.3.

Storage. The storage bound $S(n, \varepsilon)$ satisfies the recurrence[3]

$$S(n,\varepsilon) \leq \begin{cases} O(n^{\alpha'}) + S\left(\dfrac{n}{k\log^3\log n}, \dfrac{\varepsilon}{3}\right) + \displaystyle\sum_{i=1}^{n^{\alpha'}} S(n_i,\varepsilon), & \text{if } n > n_0(\varepsilon), \\ O(n), & \text{otherwise,} \end{cases} \tag{9}$$

where $n_i \leq 2n^{1-\alpha'}$ for each i, and $\sum_{i=1}^{n^{\alpha'}} n_i = n$. We show in the appendix that the solution of the recurrence is $S(n) = O(n)$, with the implied constant independent of ε. (As will follow from the analysis in the appendix, the factor $\log^3\log n$ can be replaced by any factor of the form $\log^{2+\delta}\log n$, for any positive constant δ.)

Preprocessing. In our implementation, $T_{\text{sim}}(k) = O(k)$ and $T_{\text{leaf}}(n, \varepsilon) = O(n)$. For $T_{\text{part}}(n, r)$, we use the bounds in Theorem 2.1, which depend on how small α' is. In general, $T_{\text{part}}(n, r) = O(n^{1+\delta})$, for any $\delta > 0$, and, if α' is smaller than some threshold α'_0, implied by Theorem 2.1, $T_{\text{part}}(n, r) = O(n \log r) = O(n \log n)$. The resulting recurrence for $T(n, \varepsilon)$ is thus

$$T(n,\varepsilon) \leq \begin{cases} \left\{ \begin{matrix} O(n^{1+\delta}) \\ O(n\log n) \end{matrix} \right\} + T\left(\dfrac{n}{k\log^3\log n}, \dfrac{\varepsilon}{3}\right) + \displaystyle\sum_{i=1}^{n^{\alpha'}} T(n_i,\varepsilon), & \text{if } n > n_0(\varepsilon), \\ O(n), & \text{otherwise,} \end{cases}$$

where $n_i \leq 2n^{1-\alpha'}$ for each i, and $\sum_{i=1}^{n^{\alpha'}} n_i = n$, and we choose the $O(n^{1+\delta})$ version if $\alpha' > \alpha'_0$ and the $O(n \log n)$ version otherwise. It is straightforward to verify that the solution of this recurrence is

$$T(n,\varepsilon) = \begin{cases} O(n^{1+\delta}), & \text{if } \alpha' > \alpha'_0, \\ O(n\log n), & \text{otherwise,} \end{cases}$$

with implied constants of proportionality independent of ε.

We thus have our first main result, with all the ingredients in place (and with the details supplied in the appendix), except for the high probability assertion, for which see Section 3.3.

Before stating the theorem, we emphasize the role of the "hidden parameter" α'. It can be chosen within the range $0 < \alpha' < \alpha = 1 - 1/\lfloor d/2 \rfloor$. As α' decreases, γ decreases (thus making the factor depending on ε smaller), while β increases (thus making the polylogarithmic factor larger). We therefore obtain a tradeoff between these two factors. In addition, bringing α below the threshold α'_0 reduces the preprocessing time bound from $O(n^{1+\delta})$ to $O(n \log n)$.

Another point to emphasize is that, in the statement of Theorems 3.1 and 3.2, the bounds on the query cost, storage, and preprocessing, and on the quality of the approximation, are "almost" (a) deterministic, and (b) worst-case. The bounds hold with high probability. The only randomization used in our algorithm is the sampling of the various relative approximation sets.

Theorem 3.1. *We can preprocess a set P of n points in $\mathbb{R}^d$, with a pre-specified error parameter ε, $0 < \varepsilon < 1$, into a data structure of size $O(n)$, independent of ε, so that, with high probability, for any query halfspace h, we can obtain a relative ε-approximate count of $h \cap P$, in time $O\left(\varepsilon^{-\gamma} n^{\alpha} \log^{\beta} n\right)$, where $\alpha = 1 - 1/\lfloor d/2 \rfloor$, where γ can be chosen anywhere in $(2/\lfloor d/2 \rfloor, 2)$, and where β is a constant that depends on γ, and increases when γ decreases.*

[3] Again, recall that summations are written, for simplicity, with the maximum possible number of child nodes.

The data structure is constructed deterministically, except for the random samplings that produce the various relative approximations. The (worst-case) preprocessing cost is $O(n^{1+\delta})$, for any $\delta > 0$. It reduces to $O(n \log n)$ when γ is chosen sufficiently small, with an appropriate calibration of parameters, thereby increasing β. These bounds are also independent of ε.

Remarks. (1) We note that the construction of the relative approximation sets can also be derandomized, as mentioned in the introduction. We give here only a brief sketch of such a construction, since, in our opinion, the randomized construction, combined with the high-probability analysis of Section 3.3, is much simpler and cleaner. The idea of the derandomized construction is to use the deterministic algorithm of Brönnimann *et al.* [9] to construct a *sensitive* $(\varepsilon\sqrt{p})$-approximation A, and then argue, as in [16, Lemma 6.2.8] and [17, Theorem 2.12], that the resulting set is also a relative (p, ε)-approximation. Such a construction takes time linear in n and polynomial in $\frac{1}{\varepsilon\sqrt{p}}$.

(2) We note that, although the storage and preprocessing time bounds asserted in Theorem 3.1 are independent of ε, there is nevertheless some implicit dependence on ε, in that n has to be sufficiently large, as a function of ε, to make these bounds hold. This remark also applies to our second implementation; see Theorem 3.2 below.

3.2 Second implementation

We present the second data structure ignoring the issue of high success probability again, which is handled in Section 3.3.

This implementation follows the approach of Matoušek [23] for answering halfspace *emptiness* queries. We choose $r_v = n_v / \log^\rho n_v$, for some parameter $\rho > 2$ whose value will be fixed below. The partition tree has depth $O(\log^* n)$. However, efficient implementation of the three subroutines is now more challenging, because r_v is almost as large as n_v.

Our second implementation proceeds as follows. First, the $(1/r_v, \varepsilon/2)$-approximation A_v at a node v is now of size

$$\frac{c_1 r_v}{\varepsilon^2} \log r_v = \frac{c_1 n_v}{\varepsilon^2 \log^\rho n_v} \log \frac{n_v}{\log^\rho n_v},$$

for some absolute constant c_1. We now want this size to be at most $n_v / \log n_v$. That is, we require that, for $n \geq n_0(\varepsilon)$,

$$\frac{c_1 n}{\varepsilon^2 \log^\rho n} \log \frac{n}{\log^\rho n} \leq \frac{n}{\log n},$$

or $\log^\rho n \geq \frac{c_1}{\varepsilon^2} \log^2 n$, which holds if we choose

$$n_0(\varepsilon) := 2^{\left(\frac{c'}{\varepsilon^2}\right)^{1/(\rho-2)}}, \tag{10}$$

for an appropriate constant $c' > 0$. (Note that the dependence of $n_0(\varepsilon)$ on $1/\varepsilon$ can be "controlled" by increasing the value of ρ. However, as we will shortly notice, there is a limit on how large we can take ρ to be.)

We also note that the construction can continue only as long as $\log^\rho n < n$. That is, we need to ensure that

$$\log^\rho n_0(\varepsilon) \leq n_0(\varepsilon),$$

or that

$$\left(\frac{c'}{\varepsilon^2}\right)^{\rho/(\rho-2)} < 2^{\left(\frac{c'}{\varepsilon^2}\right)^{1/(\rho-2)}}.$$

For $x := \left(\frac{c'}{\varepsilon^2}\right)^{1/(\rho-2)}$, the above requirement is that $x^\rho < 2^x$, and it holds if $x = \Omega(\rho \log \rho)$ (with an appropriate constant of proportionality), or

$$\frac{c'}{\varepsilon^2} = \Omega((\rho \log \rho)^{\rho-2}).$$

That is, it is sufficient to ensure that $\rho \log \rho = O(\log \frac{1}{\varepsilon})$, or that

$$\rho \leq \frac{\lambda \log \frac{1}{\varepsilon}}{\log \log \frac{1}{\varepsilon}}, \tag{11}$$

for an appropriate constant λ. It is easily checked that, with λ appropriately calibrated, setting ρ to this maximum value yields $n_0(\varepsilon) = 2^{O(\text{polylog } \frac{1}{\varepsilon})}$.

We now proceed to describe our implementation of the three subroutines.

SearchSim: We take the set V_v of $O(r_v)$ vertices of the simplices of the partition at v, and preprocess it for halfspace range *reporting*, as in [23]. This takes $O(r_v \log \log r_v)$ storage, and a query with a halfspace h takes $O(r_v^\alpha \log^\beta r_v + k)$ time, where $k = |h \cap V_v|$ is the output size. (We may assume, without loss of generality, that β is the same as in our first implementation.) In fact, one can easily fine-tune the algorithm, so that it never reports more than some threshold number k_0 of vertices, which we set to $k_0 := (d+1)\mu(r_v) = O(r_v^\alpha)$. With an appropriate choice of ρ and this choice of k_0, the reporting time is at most $O(n_v^\alpha \log^\beta n_v / \log^{\alpha\rho} n_v) = O(n_v^\alpha / \log^{\alpha\rho-\beta} n_v)$. By choosing ρ sufficiently large, we may assume that this is at most $O(n_v^\alpha)$.

To execute SearchSim(v), we call the reporting structure just defined with the query halfspace h. If it reports all vertices of a single simplex, or if it reports more than k_0 vertices, we stop—h is then deep at v. Otherwise, ∂h crosses at most $(d+1)\mu(r_v)$ simplices. We count the actual number of simplices that it crosses, recurse at the appropriate children of v if this number is at most $\mu(r_v)$, or declare h to be deep otherwise. If h is deep, we proceed to call DeepApxCount(h, v, ε).

LeafNodeApxCount, at a leaf v, is implemented by querying our first data structure, constructed for P_v, with h. The query time is at most $F(\varepsilon)n_0(\varepsilon)^\alpha \log^\beta n_0(\varepsilon)$. We want to upper bound it by a bound of the form $G(\varepsilon)n_0(\varepsilon)^\alpha$, which thus requires that $F(\varepsilon) \log^\beta n_0(\varepsilon) \leq G(\varepsilon)$. Using (10), this constraint will hold if we set

$$G(\varepsilon) = F(\varepsilon) \cdot \left(\frac{c'}{\varepsilon^2}\right)^{\beta/(\rho-2)}. \tag{12}$$

That is, if we take ρ to be sufficiently large, say, its maximum allowed value, given by (11), we can make $G(\varepsilon)$ "close enough" to $F(\varepsilon)$ (when ρ is given its maximum value, $G(\varepsilon)$ is larger than $F(\varepsilon)$ by a factor of the form polylog $\frac{1}{\varepsilon}$), and can still bound it by $\varepsilon^{-\gamma}$, where γ can be chosen anywhere in the interval $(2/\lfloor d/2 \rfloor,\ 2)$.

DeepApxCount is implemented recursively, by calling the auxiliary data structure constructed for A_v with error parameter $\varepsilon/3$, similar to what is done in the first implementation.

Analysis. The analysis of the query time, storage, and preprocessing leads to corresponding recurrence formulas, similar to the ones that arise in the first implementation, whose precise form and solution are given in the appendix. It implies the following second main result.

Theorem 3.2. *We can preprocess a set P of n points in $\mathbb{R}^d$, with a pre-specified error parameter $0 < \varepsilon < 1$, into a data structure of size $O(n)$, independent of ε, so that, with high probability, for any query halfspace h, we can obtain an ε-approximate count of $h \cap P$, in time $O(\varepsilon^{-\gamma} n^{\alpha} \cdot 2^{b \log^* n})$, where $\alpha = 1 - 1/\lfloor d/2 \rfloor$, γ can be chosen anywhere in $(2/\lfloor d/2 \rfloor, 2)$, and b depends on the choice of γ, increasing as γ increases. The preprocessing cost of the algorithm is $O(n^{1+\delta})$, for any $\delta > 0$, and is also independent of ε.*

Again, the bounds on the quality of the approximation and the query time, preprocessing time, and storage hold with high probability.

3.3 Ensuring high probability

So far we have presented the data structures under the assumption that at each node v we have, or can efficiently construct, a relative $(1/r_v, \varepsilon/2)$-approximation of the required size. We can achieve this either by a fairly complicated preprocessing step that constructs these sets deterministically based on the technique in [9], as already mentioned, or by drawing them at random and *verifying* that they are indeed relative approximations with the appropriate parameters, which does not appear to be trivial. In this subsection we argue that neither of these steps is necessary, and that simply drawing these sets at random without any verification still ensures high success probability.

A priori, though, drawing, at each node v, a random sample of size $\frac{cr_v}{\varepsilon^2} \log r_v$ as the respective approximation set (without any verification), as proposed in both implementations, makes it difficult to guarantee high probability. Indeed, the failure probability of such a sample at a node v is only $O(1/r_v^b)$, for some constant b that depends on c. For nodes v that are deep in the tree, r_v is small, and the success probability becomes smaller, approaching constant probability as we get closer to the leaves. Since the number of distinct ranges is polynomial in n, bounding the overall failure probability via a naive probability union bound does not imply a small overall failure probability.

Alternatively, increasing the sample size by a factor of $\log n$ would guarantee low failure probability, but this might (slightly) affect the algorithm's performance (recall that a similar phenomenon occurs in the "competing" algorithms [4, 20, 21]).

We argue that, nevertheless, using such a random sampling approach, with some additional simple mechanisms, does guarantee high success probability. The intuition is that we can think of the elements of all the relative approximation sets at the nodes that a query halfspace reaches as a sequence of independent Bernoulli trials, so that an appropriate weighted sum of their corresponding indicator variables is the approximate count that the algorithm produces. This suggests that the errors that the individual relative approximation sets incur tend to cancel each other out, leading to an overall error that is much smaller than the sum of the individual errors.

In more detail, the analysis proceeds as follows. We consider a fixed halfspace h, and bound the probability that the query with h fails to produce the desired ε-approximate count. Consider the set $V = V^{(i)}$ of all nodes v that satisfy (i) $n/2^i < n_v \leq n/2^{i-1}$, for some fixed "level" $i \geq 1$, (ii) v is reached by the query with h, and (iii) h is found to be deep at v, so $|h \cap P_v|$ is approximated using A_v. (By construction, no node of $V^{(i)}$ is a descendant of another such node.) For each of these

nodes v, we can think of the relative-error approximation A_v as a random sample, where each point of P_v is chosen independently with probability[4]

$$p_v = \frac{\frac{c_v r_v}{\varepsilon^2} \log r_v}{n_v},$$

where c_v is the constant in the bound of [16, 22], which we adjust at v, multiplying it by at most some absolute constant factor, so as to ensure that the probabilities p_v are all equal for nodes v at the same level; we denote this common value by $p^{(i)}$, for nodes at level i. At each such node v, we count $|h \cap A_v|$ (approximately, in both of our implementations, but let us pretend, for the sake of analysis only, and without affecting the resulting asymptotic bounds, that we get the exact count[5] and add $|h \cap A_v| \cdot n_v/|A_v|$ to the global count. To fit into the new model, we slightly modify this step, and add instead to the global count

$$\frac{|h \cap A_v| \cdot n_v}{\mathbf{E}\left\{|A_v|\right\}} = \frac{|h \cap A_v|}{p_v} = \frac{|h \cap A_v|}{p^{(i)}}.$$

Hence, the overall count that the approximations yield at a fixed level i is

$$\frac{1}{p^{(i)}} \cdot \sum_{v \in V} |h \cap A_v| = \frac{1}{p^{(i)}} \cdot |h \cap A|,$$

where $A = A^{(i)} := \bigcup_{v \in V} A_v$. Under the above assumptions, we can treat $|h \cap A|$ as the sum of independent indicator variables I_x, for $x \in h \cap \bigcup_{v \in V} P_v$, where $I_x = 1$ if x is chosen in the respective A_v. The expected value of $\sum_{x \in h \cap \bigcup_{v \in V} P_v} I_x$ is

$$\mu := \mathbf{E}\left\{|h \cap A|\right\} = p^{(i)} \cdot \sum_{v \in V} |h \cap P_v| = \sum_{v \in V} \frac{\frac{c_v r_v}{\varepsilon^2} \log r_v}{n_v} |h \cap P_v|.$$

Note that since h is deep at each of the nodes v, by assumption, we have $|h \cap P_v| \geq n_v/r_v$ for each $v \in V$. Hence

$$\mu \geq \sum_{v \in V} \frac{\frac{c_v r_v}{\varepsilon^2} \log r_v}{n_v} \cdot \frac{n_v}{r_v} \geq |V| \cdot \frac{c_{\min}}{\varepsilon^2} \log r_{\min}, \tag{13}$$

where $c_{\min} = \min\{c_v \mid v \in V\}$ and $r_{\min} = \min\{r_v \mid v \in V\}$. By Corollary A.14 of [3], we have

$$\mathbf{Pr}\left\{\left||h \cap A| - \mu\right| > \varepsilon\mu\right\} < 2e^{-c(\varepsilon)\mu},$$

where

$$c(\varepsilon) = \min\left\{(1+\varepsilon)\ln(1+\varepsilon) - \varepsilon,\ \frac{1}{2}\varepsilon^2\right\} > \frac{1}{4}\varepsilon^2,$$

for $0 < \varepsilon < 1$. Hence, using (13), we have

$$\mathbf{Pr}\left\{\left||h \cap A| - \mu\right| > \varepsilon\mu\right\} < 2e^{-\frac{c_{\min}}{4}|V| \log r_{\min}}.$$

[4]Note that the sampling model used in [31] and the majority of subsequent papers picks a random subset of pre-specified size, with all such subsets chosen with equal probability. Most of the results also hold in the model we use here, where each point is sampled independently, with a fixed probability. We have elected to use the latter model to be able to apply Chernoff bounds.

[5]Recall that the actual approximate count that we get is within a factor of $1 \pm \varepsilon/2$ of the exact count. This is another instance where we rely on the high-probability analysis spelled out here. Technically, we apply it recursively to A_v and guarantee the estimate with high overall probability.

In other words, the failure probability (within the present fixed level i) depends on $|V|$. Specifically, putting $a := c_{\min}/4$, the above probability is smaller than

$$2e^{-a|V|\log r_{\min}} = \frac{2}{r_{\min}^{a|V|}}.$$

Recall that r_v is chosen to be either a fixed fractional power of n_v, or $n_v/\operatorname{polylog} n_v$; in either case, since $n_v \approx n/2^i$ at our fixed level i, the failure probability is at most

$$2\left(\frac{2^i}{n}\right)^{a'|V|}, \tag{14}$$

where a' is an appropriate multiple of a.

Informally, if either $n^{(i)} := n/2^i$ is large or $|V|$ is large, this bound is *exponentially* or at least *polynomially small* in n (recall that $n^{(i)}$ is at least $n_0(\varepsilon)$, so it is never close to 1). However, if $|V|$ is small (in the extreme case, we could even have $|V| = 1$), and if $n^{(i)}$ is small too (we are in a deep level), then the failure probability in (14) might be quite large (in the worst case, it depends only on ε (via the dependence on $n_0(\varepsilon)$), and not on n).

To rectify this problem, we simply note that when $|V|$ is small, we can afford more time for the query at the nodes of V, and can even afford to process the query at these nodes by brute force, testing every member of P_v for belonging to h. To make this argument more precise, we fix some fraction $\nu < \alpha$, and consider situations where $n^{(i)} < n^{\nu/2}$, and $|V| < n^\nu/n^{(i)}$. (It is easily checked that in any other situation, the bound in (14) is at most polynomially small in n, with an exponent that can be chosen arbitrarily large, by calibrating the constants c_v and $c_{\min}$.) In such cases, the brute-force cost of processing all the nodes of V is $O\left(\sum_{v\in V} n_v\right) = O(n^{(i)}|V|) = O(n^\nu)$, and summing this over all levels, we get an overall cost of $O(n^\nu \log n) = O(n^\alpha)$, well within our target bound (note that this part of the procedure is purely deterministic).

To complete the analysis, we apply a union bound to estimate the probability that the approximate count yielded by the above procedure will fail for at least one level for at least one query halfspace h. Since the number of levels is only logarithmic, and the number of (combinatorially different) halfspaces is only $O(n^d)$, it follows that this overall failure probability (namely, that for at least one halfspace the count is not accurate enough) is at most polynomially small in n, with an exponent that can be chosen arbitrarily large.

To implement the modified procedure, we first collect, during the processing of the query with h, the sets $V^{(i)}$ of all the nodes v at any fixed level i, where h is deep in P_v. If, for any level i, $n^{(i)}$ and $|V^{(i)}|$ are both small, in the precise sense defined above, we count $\sum_{v\in V} |h \cap P_v|$ by brute force. Otherwise, we obtain this count using the approximations stored at these nodes, as prescribed earlier.

We have to be careful with the storage requirement, especially in the second implementation, because the above technique requires us to store each of the sets P_v, so that we can search any of these sets by brute force during a query, if necessary. Storing these sets explicitly would require $\Theta(n \log\log n)$ storage in the first implementation, and $\Theta(n \log^* n)$ storage in the second implementation. However, we can reduce the storage to linear, if we maintain just one master list (or array) of all the points in P, so that, for any node v, P_v is a *contiguous* sublist, which

can be specified by two pointers to its first and last elements. We omit the further easy details of constructing this list and these pointers during preprocessing.

The preceding discussion implies that, with high probability, the resulting data structure yields an ε-approximate count of $h \cap P$, for *any* halfspace h. The storage, preprocessing time, and query time all remain asymptotically the same, but the query procedure is slightly modified, as explained above. (Recall that these bounds are worst-case deterministic, except for the sampling of the approximation sets.)

With this analysis, the proofs of Theorems 3.1 and 3.2 are now complete.

3.4 Discussion

We conclude the presentation of the basic technique with a few comments.

(1) Notice that there is a sharp discontinuity in the performance of a query in the first implementation, as we reach the leaves of the partition tree. At internal nodes, we effectively ensure that the cost of the approximate counting via the $(1/r_v, \varepsilon/2)$-approximation stored at a node v is roughly n_v^α. In contrast, when we reach a leaf, the cost goes up to $\Theta(n_v)$. Quite likely, smoothly interpolating between these two scenarios should refine the dependence of the performance bounds on ε. We leave this as an open problem for further research.

(2) Our technique can be modified to produce a data structure where ε is not known in advance. Of course, we cannot let ε become arbitrarily small, or else the size of the structure will grow out of control. We therefore specify the smallest value $\varepsilon_{\min}$ of ε that we want the structure to handle and maintain, at each node v of the tree, many relative approximations. Specifically, we store at v a relative $(1/r_v, \varepsilon_i)$-approximation, for each ε_i in an appropriate geometric sequence, stopping at the larger of $\varepsilon_{\min}$ and the smallest ε_i for which the size of the approximation is still below the threshold $n_v/(k \log^3 \log n_v)$, say. Since the sizes of these approximations also form a geometric sequence, their overall size is still within the allowed storage. Note that a node v may become a leaf for certain values of ε_i, and remain an internal node for smaller values of ε_i. This can easily be handled by comparing n_v with $n_0(\varepsilon_i)$, at each node v. Overall, assuming n is sufficiently large, we get a data structure whose storage and preprocessing costs are as stated in Theorem 3.1 or Theorem 3.2, and whose query performance is again bounded as in these theorems, but in terms of the actual $\varepsilon \geq \varepsilon_{\min}$ specified in the query.

(3) An interesting issue not addressed in this paper is that of designing a range of data structures exhibiting the query-time–storage-space trade-off; such trade-offs have been studies extensively for other range searching problems [2].

3.5 Range-minimum queries

We can apply our technique to design an efficient algorithm for answering range-minimum queries for halfspaces, with respect to a given random permutation π of the input points, of the type needed in the approach of Cohen [13] and of Kaplan *et al.* [20, 21], as described in Section 1. The only difference is that at each node v of the partition tree, we restrict π to P_v, and store the prefix of the first $cr_v \log n$ elements of the resulting restricted permutation π_v, for a sufficiently large constant c. Intuitively, if a halfspace h is deep at v (i.e., $|h \cap P_v| \geq n_v/r_v$), the point p of $h \cap P_v$ of minimum rank should appear, with high probability, among the first $cr_v \log n$ elements of π_v, so we can find

p by examining each of these elements. This allows us to execute a query in much the same way as above. It is also fairly easy to show that the procedure has high overall success probability. Omitting all further details, we obtain (the theorem parallels our first implementation; extending the second implementation can also be done):

Theorem 3.3. *One can preprocess a set P of n points in $\mathbb{R}^d$, and a random permutation π of P, into a linear-size data structure, such that, with high probability, the element of P with minimum rank in π in a query halfspace can be computed in time $O(n^{1-1/\lfloor d/2 \rfloor} \log^\beta n)$, for an appropriate constant $\beta = \beta(d)$. The preprocessing cost is $O(n^{1+\delta})$, for any $\delta > 0$, and it improves to $O(n \log n)$ if β is chosen sufficiently large. The bounds on the preprocessing, storage, and query costs are all worst-case deterministic, and the correctness of the queries is guaranteed with high probability, for all queries.*

3.6 Approximate range counting with semi-algebraic sets

Our techniques can be extended to yield efficient data structures for approximate range counting, where the ranges are semi-algebraic sets of constant description complexity. This has recently been carried out by Sharir and Shaul [29], who first extended Matoušek's range reporting and emptiness data structures [23] to semi-algebraic ranges, and then plugged the machinery of the current paper into those extended structures. See [29] for details.

4 Open Problems

As mentioned above, we have presented a general framework of using shallow partition trees for approximate range counting. We examined in some detail two specific instances of such a data structure and several extensions. It remains to explore other uses of this structure and other combinations of building blocks that may yield more efficient, or less complicated variants. In particular, we do not feel we have completely exhausted the entire "bag of tricks" to reduce the dependence of the query time on the approximation parameter ε.

References

[1] P. Afshani and T. M. Chan, On approximate range counting and depth, *Discrete Comput. Geom.* 41 (2009), 3–21.

[2] P. K. Agarwal and J. Erickson, Geometric Range Searching and Its Relatives, *Advances in Discrete and Computational Geometry*, 1–56, 1999, American Mathematical Society.

[3] N. Alon and J. Spencer, *The Probabilistic Method*, Wiley-Interscience, New York, 1992.

[4] B. Aronov and S. Har-Peled, On approximating the depth and related problems, *SIAM J. Comput.* 38 (2008), 899–921.

[5] B. Aronov, S. Har-Peled and M. Sharir, On approximate halfspace range counting and relative epsilon approximations. *Proc. 23rd ACM Symp. on Computational Geometry* (2007), pp. 327–336.

[6] S. Arya, G. D. da Fonseca, and D. Mount, Tradeoffs in approximate range searching made simpler, *Proc. 21st Brazilian Sympos. Computer Graphics and Image Processing*, 2008, pp. 237–243.

[7] S. Arya, T. Malamatos, and D. Mount, Space-time tradeoffs for approximate spherical range counting, *Proc. 16th Annu. ACM-SIAM Sympos. Discrete Algo.*, 2005, pp. 535–544. Full version available as Dept. of Computer Science Technical Report CS-TR-4842, Univ. of Maryland, November 2006.

[8] S. Arya, T. Malamatos, and D. Mount, The effect of corners on the complexity of approximate range searching, *Discrete and Computational Geometry*, 41 (2009), 398–443.

[9] H. Brönnimann, B. Chazelle, and J. Matoušek, Product range spaces, sensitive sampling, and derandomization, *SIAM J. Comput.* 28 (1999), 1552–1575.

[10] B. Chazelle, *The Discrepancy Method*, Cambridge University Press, Cambridge, UK, 2001.

[11] B. Chazelle, The discrepancy method in computational geometry, chapter 44, in *Handbook of Discrete and Computational Geometry*, 2nd Edition, J. E. Goodman and J. O'Rourke, Eds., CRC Press, Boca Raton, 2004, 983–996.

[12] B. Chazelle, D. Liu, and A. Magen, Approximate range searching in higher dimension, *Comput. Geom. Theory Appls.* 39 (2008), 24–29.

[13] E. Cohen, Size-estimation framework with applications to transitive closure and reachability, *J. Comput. Syst. Sci.* 55 (1997), 441–453.

[14] E. Cohen, H. Kaplan, Y. Mansour, and M. Sharir, Approximations with relative errors in range spaces of finite VC dimension, manuscript, 2006.

[15] G. D. da Fonseca, Approximate range searching: The absolute model, *Proc. Workshop on Algorithms and Data Structures*, 2007, Springer LNCS 4619, pp. 2–14.

[16] S. Har-Peled, Carnival of samplings: nets, approximations, relative and sensitive, manuscript, 2008; http://arxiv.org/abs/0908.3716v1. Revised version appears as Chapter 6, "Yet even more on sampling" in Har-Peled's class notes.

[17] S. Har-Peled and M. Sharir, Relative (p, ε)-approximations in geometry, manuscript, 2009; http://arxiv.org/abs/0909.0717.

[18] D. Haussler, Decision theoretic generalizations of the PAC model for neural nets and other learning applications, *Inf. Comput.* 100 (1992), 78–150.

[19] D. Haussler and E. Welzl, Epsilon nets and simplex range queries, *Discrete Comput. Geom.* 2 (1987), 127–151.

[20] H. Kaplan, E. Ramos, and M. Sharir, Range minima queries with respect to a random permutation, and approximate range counting, *Discrete Comput. Geom.*, accepted.

[21] H. Kaplan and M. Sharir, Randomized incremental construction of three-dimensional convex hulls and planar Voronoi diagrams, and approximate range counting, *Proc. 17th ACM-SIAM Sympos. Discrete Algorithms* (2006), pp. 484–493.

[22] Y. Li, P. M. Long, and A. Srinivasan, Improved bounds on the sample complexity of learning, *J. Comput. Syst. Sci.* 62 (2001), 516–527.

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

[24] J. Matoušek, Efficient partition trees, *Discrete Comput. Geom.* 8 (1992), 315–334.

[25] J. Matoušek, *Geometric Discrepancy*, Algorithms and Combinatorics, Vol. 18, Springer Verlag, Heidelberg, 1999.

[26] J. Matoušek, E. Welzl and L. Wernisch, Discrepancy and approximations for bounded VC-dimension, *Combinatorica* 13 (1993), 455–466.

[27] J. Pach and P. K. Agarwal, *Combinatorial Geometry*, Wiley Interscience, New York, 1995.

[28] D. Pollard, Rates of uniform almost-sure convergence for empirical processes indexed by unbounded classes of functions, manuscript, 1986.

[29] H. Shaul and M. Sharir, Semi-algebraic range reporting and emptiness searching with applications, *SIAM J. Comput.*, submitted; http://arxiv.org/abs/0908.4061v2. An earlier version appeared as "Ray shooting amid balls, farthest point from a line, and range emptiness searching," *Proc. 16th ACM-SIAM Sympos. Discrete Algorithms* (2005), pp. 525–534.

[30] M. Talagrand, Sharper bounds for Gaussian and empirical processes, *Annals of Probability* 22 (1994), 28–76.

[31] V. N. Vapnik and A. Ya. Chervonenkis, On the uniform convergence of relative frequencies of events to their probabilities, *Theory of Probability and its Applications* 16 (1971), 264–280.

Appendix

First implementation.

Analysis of query time. We want to show that the solution of the recurrence

$$Q(n,\varepsilon) \leq \begin{cases} O(n^{\alpha'}) + \max\left\{ Q\left(\frac{n}{k\log^3\log n}, \frac{\varepsilon}{3}\right),\ \mu(n^{\alpha'})Q(2n^{1-\alpha'},\varepsilon)\right\}, & \text{if } n > n_0(\varepsilon), \\ O(n), & \text{otherwise} \end{cases}$$

is given by (7), with $F(\varepsilon)$ given by (8). We proceed by induction on n. Eq. (7) clearly holds for $n \leq n_0(\varepsilon)$, by the choice of $n_0(\varepsilon)$. For larger values of n, to carry out the induction step, we need to show that

$$\begin{aligned} c_1 n^{\alpha'} + \max\left\{ F\left(\frac{\varepsilon}{3}\right)\left(\frac{n}{k\log^3\log n}\right)^{\alpha}\log^{\beta}\left(\frac{n}{k\log^3\log n}\right),\ c_2 n^{\alpha'\alpha} F(\varepsilon) n^{(1-\alpha')\alpha}\log^{\beta} n^{1-\alpha'}\right\} \\ \leq F(\varepsilon) n^{\alpha}\log^{\beta} n, \end{aligned}$$

for appropriate constants c_1, c_2. If we calibrate the constant in the definition of $F(\varepsilon)$, so that $F(\varepsilon)$ is always at least $2c_1$, then $c_1 n^{\alpha'} < c_1 n^{\alpha} \leq \frac{1}{2}F(\varepsilon)n^{\alpha}\log^{\beta} n$. Hence, it is enough to show that

$$\frac{1}{2} + \max\left\{ \frac{F\left(\frac{\varepsilon}{3}\right)}{F(\varepsilon)k^{\alpha}\log^{3\alpha}\log n},\ c_2(1-\alpha')^{\beta}\right\} \leq 1,$$

which holds, by the way $F(\varepsilon)$ is defined in (8), if β and k are chosen large enough.

Remark: Note the tradeoff between β and $F(\varepsilon)$: We can make $F(\varepsilon)$ asymptotically smaller (i.e., decrease γ) if we decrease α' and consequently increase β. (The bound on the cost of a query can thus be optimized, for given values of n and ε, if so desired.)

Analysis of storage. We want to show that the solution of the recurrence

$$S(n,\varepsilon) \leq \begin{cases} O(n^{\alpha'}) + S\left(|A|, \frac{\varepsilon}{3}\right) + \sum_{i=1}^{n^{\alpha'}} S(n_i, \varepsilon), & \text{if } n > n_0(\varepsilon), \\ O(n), & \text{otherwise} \end{cases} \tag{9}$$

is linear in n, where, for an appropriate constant c_1,

$$|A| = \frac{c_1 n^{\alpha'}}{\varepsilon^2} \log(n^{\alpha'}) \leq \frac{n}{k \log^3 \log n}$$

is the size of the relative $(1/r, \varepsilon/2)$-approximation stored at the current node, $n_i \leq 2n^{1-\alpha'}$ for each i, and $\sum_{i=1}^{n^{\alpha'}} n_i = n$.

We first show that $S(n) \leq Dn \log\log n$, where D is a constant that does not depend on ε. Again, we prove this by induction on n, noting that it holds for $n \leq n_0(\varepsilon)$, for an appropriate choice of D (independent of ε). For larger values of n, to carry out the inductive argument, it suffices to show that

$$c_2 n^{\alpha'} + \frac{Dn}{k \log^3 \log n} \log\log\left(\frac{n}{k \log^3 \log n}\right) + \sum_{i=1}^{n^{\alpha'}} Dn_i \log\log(2n^{1-\alpha'}) \leq Dn \log\log n,$$

for an appropriate constant c_2. We use n instead of $n^{\alpha'}$ in the first term, and ensure that $n^{\alpha'/2} > 2$ for $n \geq n_0(\varepsilon)$, which we enforce, independently of ε, by choosing the constant c' in (6) large enough. Then it is sufficient to guarantee that

$$c_2 + \frac{D}{k \log^2 \log n} + D(\log(1 - \alpha'/2) + \log\log n) \leq D \log\log n,$$

which holds if we choose, say, $\alpha' > 2\left(1 - \frac{1}{2^{2/k}}\right)$, and $D > 2c_2 k$. (Again, we have a tradeoff: to reduce $F(\varepsilon)$ asymptotically, we have to choose smaller α', so k has to be chosen larger, which causes the constant of proportionality in the storage bound to increase.)

This finishes the argument that $S(n,\varepsilon) \leq Dn \log\log n$. Armed with this bound, we apply it to $S\left(|A|, \frac{\varepsilon}{3}\right)$, to obtain

$$S\left(|A|, \frac{\varepsilon}{3}\right) \leq \frac{Dn \log\log n}{k \log^3 \log n} = O\left(\frac{n}{\log^2 \log n}\right).$$

Since this dominates the first term $O(n^{\alpha'})$ in the recursion (9), we get a new recurrence of the form (in which we drop the unnecessary dependence on ε)

$$S(n) \leq \begin{cases} \frac{D'n}{\log^2 \log n} + \sum_{i=1}^{n^{\alpha'}} S(n_i), & \text{if } n > n_0(\varepsilon), \\ D'n, & \text{otherwise,} \end{cases}$$

for some absolute constant D', with the usual constraints on the numbers n_i.

We claim that the solution of this recurrence is $S(n) = O(n)$, where the constant of proportionality clearly does not depend on ε. To show this, we unwind the recurrence, as follows. When we expand a node v of the tree and form its children, each child w satisfies, by construction, $n_v^{1-\alpha'} \leq n_w \leq 2n_v^{1-\alpha'} \leq n_v^{1-\alpha'/2}$. We say that a node v *lies at a level* j if

$$n^{(1-\alpha'/2)^{j+1}} < n_v \leq n^{(1-\alpha'/2)^j}.$$

Thus the root lies at level 0, and the maximum level is at most $c_3 \log\log n$, for some constant c_3, depending on ε and α'. Also, no two nodes on a common path have the same level, so the sum of the sizes n_v, over all nodes v of a fixed level, is at most n. Hence, the sum of the overhead terms of all the nodes at level j is at most

$$\frac{D'n}{\log^2 \log n^{(1-\alpha'/2)^{j+1}}} = \frac{D'n}{\left(\log\log n - \xi(j+1)\right)^2},$$

where $\xi = \log \frac{1}{1-\alpha'/2} > 0$. The sum of the amounts of storage $S(n_w)$ at the leaves w of the tree is clearly at most $D'n$. Hence, the overall storage requirement is at most

$$D'n + \sum_{0 \le j \le c_3 \log\log n} \frac{D'n}{\left(\log\log n - \xi(j+1)\right)^2}.$$

The smallest value of any denominator is attained at the parents of the leaves, and is at least $(\log\log n_0(\varepsilon))^2$. This implies (e.g., by replacing the sum by an integral) that the sum can be bounded by $O\left(n/\log\log n_0(\varepsilon)\right)$, which is $O(n)$ (with the constant of proportionality independent of ε)

Second implementation.

Analysis of query time. The query time $Q(n,\varepsilon)$ satisfies the following recurrence.

$$Q(n,\varepsilon) \le \begin{cases} O(n^\alpha) + \max\left\{ Q\left(\frac{n}{\log n}, \frac{\varepsilon}{3}\right),\ \mu\left(\frac{n}{\log^\rho n}\right) \cdot Q(2\log^\rho n, \varepsilon) \right\}, & \text{if } n > n_0(\varepsilon), \\ G(\varepsilon) n^\alpha, & \text{otherwise,} \end{cases}$$

where $G(\varepsilon)$ is defined as in (12).

We claim that $Q(n,\varepsilon) \le G(\varepsilon) n^\alpha \cdot 2^{\phi(n)}$, for an appropriate choice of b and of the constant of proportionality of G, where $\phi(n)$ is the following recursively defined function

$$\phi(n) = \begin{cases} \phi(2\log^\rho n) + 1, & \text{if } n > n_0^*, \\ 1, & \text{if } n \le n_0^*, \end{cases}$$

where n_0^* is the smallest integer satisfying $n \ge 2\log^\rho n$. It is easily checked that $\phi(n) = \Theta(\log^* n)$, so for the sake of simplifying the presentation, we abuse the notation, and refer to ϕ also as $\log^* n$. We are thus set to prove that

$$Q(n,\varepsilon) \le G(\varepsilon) n^\alpha \cdot 2^{b\phi(n)}. \tag{15}$$

We prove (15) by induction on n. It clearly holds when $n \le n_0(\varepsilon)$—this follows from the way G was defined in (12). For larger values of n, using the induction hypothesis, it is sufficient to show that

$$an^\alpha + \max\left\{ G(\varepsilon/3)\left(\frac{n}{\log n}\right)^\alpha 2^{b\log^*(n/(\log n))}, \right. \\ \left. A\left(\frac{n}{\log^\rho n}\right)^\alpha \cdot G(\varepsilon)\left(\log^\rho n\right)^\alpha \cdot 2^{b\log^*(2\log^\rho n)} \right\} \le G(\varepsilon) n^\alpha \cdot 2^{b\log^* n},$$

for appropriate constants a, A. Calibrate the constant of proportionality in the expression for $G(\varepsilon)$ so that $G(\varepsilon) > 2a$. It then suffices to show that

$$\frac{1}{2} + \max\left\{ \frac{G(\varepsilon/3)}{G(\varepsilon)} \cdot \frac{1}{\log^\alpha n},\ \frac{A \cdot 2^{b\log^*(2\log^\rho n)}}{2^{b\log^* n}} \right\} \le 1,$$

which follows easily (i) from the definition of $G(\varepsilon)$, and (ii) from our redefinition of the $\log^*(\cdot)$ function, provided that b is chosen large enough to satisfy $2^b \ge 2A$.

Analysis of storage. The storage bound $S(n,\varepsilon)$ satisfies the following recurrence, in which $n = n_v$, the storage needed for SearchSim is $O(r_v \log\log r_v)$, and the storage for leaf nodes is $O(n_v) = O(n)$, a bound independent of ε, since we use our first data structure at each leaf.

$$S(n,\varepsilon) \le \begin{cases} O\left(\dfrac{n\log\log n}{\log^\rho n}\right) + S\left(\dfrac{n}{\log n}, \dfrac{\varepsilon}{3}\right) + \displaystyle\sum_{i=1}^{n/\log^\rho n} S(n_i,\varepsilon), & \text{if } n > n_0(\varepsilon), \\ O(n), & \text{otherwise,} \end{cases} \tag{16}$$

where $n_i \le 2\log^\rho n$ and $\sum_{i=1}^{n/\log^\rho n} n_i = n$.

We bound $S(n)$ in two stages. We first claim that the solution of the recurrence (16) is $S(n) \le Dn \cdot 2^{b' \log^* n}$, for appropriate constants D, b' independent of ε. We prove this by induction, noting that it trivially holds for $n \le n_0(\varepsilon)$, with an appropriate choice of D. For larger values, to carry out the induction step, it suffices to show that

$$\frac{c'n\log\log n}{\log^\rho n} + \frac{Dn}{\log n} \cdot 2^{b'\log^* n} + \sum_{i=1}^{n/\log^\rho n} Dn_i \cdot 2^{b'\log^*(2\log^\rho n)} \le Dn \cdot 2^{b'\log^* n},$$

for an appropriate constant c'. This is easily seen to hold if D and b' are chosen sufficiently large (independently of ε).

Armed with this intermediate bound, we use it to get the upper bound

$$S\left(\frac{n}{\log n}, \frac{\varepsilon}{3}\right) \le \frac{Dn \cdot 2^{b'\log^* n}}{\log n} \le \frac{D'n}{\log^{1/2} n},$$

for some constant D'. This is also an upper bound on the first term in (16), so the recurrence becomes

$$S(n,\varepsilon) \le \begin{cases} \dfrac{D''n}{\log^{1/2} n} + \displaystyle\sum_{i=1}^{n/\log^\rho n} S(n_i,\varepsilon), & \text{if } n > n_0(\varepsilon), \\ O(n), & \text{otherwise,} \end{cases}$$

for yet another constant D'', where $n_i \le 2\log^\rho n$ and $\sum_{i=1}^{n/\log^\rho n} n_i = n$. We claim that the solution of this recurrence is $S(n,\varepsilon) = O(n)$, with the constant or proportionality independent of ε, and show it by unwinding the recurrence, using the fact that there are only $O(\log^* n)$ levels. Specifically, we proceed similarly to the analysis of the first implementation. We define a sequence $n_0 = n$, $n_{j+1} = 2\log^\rho n_j$, $j \ge 0$, and say that a node v is at level j if $n_{j+1} < n_v \le n_j$. The sum of the sizes n_v, over all nodes v at any fixed level j, is at most n (again, no two nodes on a common path can have the same level), and the sum of their overhead terms is at most $D''n/\log^{1/2} n_{j+1}$. It is now an easy exercise to show that $\sum_{j\ge 0} 1/\log^{1/2} n_j = O(1)$. This is because the n_j's form a sequence of exponential towers, so the terms $1/\log^{1/2} n_j$ increase very rapidly, and their sum is essentially equal to the last term $1/\log^{1/2}\log n_{j_{\max}}$, which is at most a constant (independent of ε), since $n_j \ge n_0(\varepsilon)$ for all j.

This completes the proof that $S(n) = O(n)$, with a constant of proportionality independent of ε.

Analysis of preprocessing. Here we use the bound $T_{\text{part}}(n,r) = O(n^{1+\delta})$, for any $\delta > 0$ where this bound is independent of ε, and note that this bound also applies to T_{sim} and T_{leaf}. We thus get

the following recurrence.

$$T(n,\varepsilon) \leq \begin{cases} O(n^{1+\delta}) + T\left(\frac{n}{\log n}, \frac{\varepsilon}{3}\right) + \sum_{i=1}^{2n/\log^{\rho} n} T(n_i, \varepsilon), & \text{if } n > n_0(\varepsilon), \\ O(n^{1+\delta}), & \text{otherwise,} \end{cases}$$

for any $\delta > 0$, with the usual constraints on the n_i's. Using techniques similar to (and somewhat simpler than) those in the previous analyses, one can easily verify that $T(n,\varepsilon) = O(n^{1+\delta'})$, for any $\delta' > \delta > 0$ with a constant of proportionality independent of ε.