Approximate Range Searching: The Absolute Model*

Guilherme D. da Fonseca
Department of Computer Science
University of Maryland
College Park, Maryland 20742
fonseca@cs.umd.edu

David M. Mount
Department of Computer Science and
Institute for Advanced Computer Studies
University of Maryland
College Park, Maryland 20742
mount@cs.umd.edu

May 7, 2007

Abstract

Range searching is a well known problem in the area of geometric data structures. We consider this problem in the context of approximation, where an approximation parameter $\varepsilon > 0$ is provided. Most prior work on this problem has focused on the case of relative errors, where each range shape R is bounded, and points within distance $\varepsilon \cdot diam(R)$ of the range's boundary may or may not be included. We consider a different approximation model, called the *absolute model*, in which points within distance ε of the range's boundary may or may not be included, regardless of the diameter of the range. We consider range spaces consisting of halfspaces, Euclidean balls, simplices, axis-aligned rectangles, and general convex bodies. We consider a variety of problem formulations, including range searching under general commutative semigroups, idempotent semigroups, groups, and range emptiness. We show how idempotence can be used to improve not only approximate, but also exact halfspace range searching. Our data structures are much simpler than both their exact and relative model counterparts, and so are amenable to efficient implementation.

*The work of the first author has been supported in part by CAPES, Brazil, under grant BEX-1319027, and the work of the second author has been supported in part by the National Science Foundation under grant CCR-0635099.

1 Introduction

The *range searching problem* involves preprocessing a set P of n points in $\mathbb{R}^d$ so that given a region R, that is drawn from a predefined set of shapes $\mathcal{R}$, a predefined function $f(P \cap R)$ can be computed efficiently. The set of possible shapes $\mathcal{R}$ characterizes the *range space*, and the elements of $\mathcal{R}$ are called *ranges*. Examples of range spaces include halfspaces, axis-aligned rectangles, and simplices. We let $q(R) = f(P \cap R)$ denote the result of the *query*. The points $p \in P$ are called *data points*.

Range searching is a well-studied problem in computational geometry. Excellent surveys have been written by Matoušek [16] and Agarwal and Erickson [1]. The two most common examples include *range counting*, where $f(Q) = |Q|$ and *range reporting*, where $f(Q) = Q$.

More generally, we have a commutative semigroup $(\mathbf{S}, +)$, a *weight* function $w : P \to \mathbf{S}$, and want to compute $f(Q) = \sum_{p \in Q} w(p)$. This general version of the problem is called the *semigroup version*. In the *group* version of the problem, $(\mathbf{S}, +)$ is a commutative (Abelian) group. The group version may admit more efficient solutions, because the presence of inverses means that both addition and subtraction may be used to compute the answer to the query. In the *idempotent* version of the problem, $(\mathbf{S}, +)$ is an idempotent group, i.e. $x + x = x$ for all $x \in \mathbf{S}$. One example of idempotent semigroup is $(\mathbb{R}, \max)$. The *emptiness version* can be modeled by the idempotent semigroup $(\{0, 1\}, \vee)$ and assigning $w(p) = 1$ for all $p \in P$.

The relatively high complexity of exact range searching has led researchers to consider the problem in the context of approximation. A natural way to do this is to consider the range shape to be "fuzzy," and allow points that are close to the range boundary to either be counted or not. There are two natural ways to define approximation. In both cases a user-supplied approximation parameter $\varepsilon > 0$ is given, either at preprocessing time or at query time. In the *relative error model* (or simply *relative model* for short) it is assumed that the range shape R is bounded, and points lying within distance $\varepsilon \cdot \mathit{diam}(R)$ of the boundary of the range may or may not be included. In contrast, in the *absolute error model* (or simply *absolute model*) points lying within distance ε of the range's boundary may or may not be included.

Note that, in the absolute model, some type of scaling is needed, for otherwise it would be possible to answer queries with arbitrarily high precision by applying some high scale factor to the point and range coordinates, while keeping the error parameter fixed. Without loss of generality, we assume throughout that the point set P has been transformed (through a uniform scaling and translation) to lie within the unit hypercube $[0, 1]^d$. We assume that the ranges have been similarly transformed, and the parameter ε has been scaled correspondingly.

Approximate range searching in the relative model is studied in [6, 3, 4, 5]. Chazelle, Liu and Magen [11] studied approximate range searching in the absolute model, but considered the problem in spaces of high dimension. They presented a data structure which answers halfspace queries in $O((d/\varepsilon)^2 \log^{O(1)}(d/\varepsilon))$ time with $dn^{O(1/\varepsilon^2)}$ storage. Throughout this paper, we assume that the dimension d is constant.

There are a number of reasons for studying approximate range searching in the absolute model. First, the absolute model admits much simpler solutions. While the most efficient data structures for answering range queries both in the relative model and exact case tend

to be quite complex, our techniques are extremely simple (involving simple structures such as grids and quadtrees) and so are amenable to efficient implementation. In the absolute model, the data structures are not sensitive to the point distribution. Therefore, the absolute model allows us to reason about the range searching problem (for example, the best size and shape of the generators) in a simpler context, and may lead to more efficient and simpler structures for exact range searching.

Second, the absolute model is better suited for several applications, when compared to the relative model. If the coordinates of a point represent an object that exists within some extent in space, or data that is subject to measurement errors or noise, then the approximation quality should be based on the expected error of the point locations, not on the diameter of the query range. A shortcoming of the relative model is that it cannot directly handle unbounded ranges, such as halfspaces and unbounded polyhedra.

Finally, the storage space and query time of the absolute model data structures is independent of n, and hence our results can be adapted to work in the *data stream model* [17, 7]. In the data stream model, the data set is too large to fit in memory, therefore the storage space should be independent of n (sometimes polylogarithmic functions of n are acceptable). Also, the data points are examined one at a time, in a single pass, while queries regarding the points that have already been seen, or that have been seen recently, need to be answered efficiently. Exact range searching clearly requires $\Omega(n)$ storage for all reasonable sets of ranges, and approximate range searching in the relative model requires $\Omega(n)$ storage when the query ranges can be scaled arbitrarily. Suri, Tóth, and Zhou [18] consider approximate range counting in the data stream model, approximating the number of points inside the query region.

Because most of our results make use of bucketing, we assume a model of computation that supports integer division. We use $\widetilde{O}(x)$ to denote $O(x \log^{O(1)} x)$, $\widetilde{\Omega}(x)$ to denote $\Omega(x/\log^{O(1)} x)$, and $\alpha(m, n)$ to denote the inverse Ackermann function [19].

The main results of this paper include the introduction of the absolute model, and approximate data structures for the most common range spaces, including a data structure that benefits from idempotence, as well as the *halfbox quadtree* data structure. Several data structures provide a space-time tradeoff, where the query time can be made arbitrarily low, at the cost of extra storage space. The complexities of our approximate data structures are summarized in Table 1.

We also relate approximate data structures in the absolute model to exact data structures. Assuming uniformly distributed data points, we provide an exact data structure for halfspace range searching, which makes use of idempotence to improve over the most efficient exact data structure previously known. This exact data structure is defined in the semigroup arithmetic model [10, 8, 5], and has $O(n^{1-2/(d+1)})$ expected query time with $O(n)$ space, matching the lower bound proved in [8] up to logarithmic factors. The theoretical importance of the data structure relies on the fact that uniform distribution and the semigroup arithmetic model are also assumed in the lower bound proved in [8]. Therefore, we open some important theoretical and practical questions: Is the average case complexity for uniformly distributed data strictly lower than the worst case complexity? Does the semigroup arithmetic model

Range	Version	Storage space	Query time	Preprocessing	Section
Halfspace	semigroup	$O(1/\varepsilon^d)$	$O(1)$	$\widetilde{O}(n+1/\varepsilon^d)$	3.1
	idempotent	$m \geq 1/\varepsilon^{(d+1)/2}$	$O(1/m\varepsilon^d)$	$\widetilde{O}(n+1/\varepsilon^d)$	3.2
	emptiness	$m \geq 1/\varepsilon^{(d-1)/2}$	$O(1/m\varepsilon^{d-1})$	$O(n+1/\varepsilon^d)$	3.4
Spherical	semigroup	$\widetilde{O}(1/\varepsilon^d)$	$O(1/\varepsilon^{(d-1)/2})$	$\widetilde{O}(n+1/\varepsilon^d)$	4.1
	semigroup	$m \geq 1/\varepsilon^{d+1}$	$O(1/m^{(1/2)-O(1/d)}\varepsilon^{d-O(1)})$	$O(n+m/\varepsilon^{(d-1)/2})$	4.1
Simplex	group	$\widetilde{O}(1/\varepsilon^d)$	$O(1/\varepsilon^{d-2}+\log(1/\varepsilon))$	$\widetilde{O}(n+1/\varepsilon^d)$	4.2
Orthogonal	semigroup	$m \geq 1/\varepsilon^d$	$O(\alpha(m, 1/\varepsilon^d))$	$O(n+m)$	5
	group	$O(1/\varepsilon^d)$	$O(1)$	$O(n+1/\varepsilon^d)$	5
Convex	semigroup	$O(1/\varepsilon^d)$	$O(1/\varepsilon^{d-1})$	$O(n+1/\varepsilon^d)$	5

Table 1: Complexities of several approximate range searching data structures.

allow more efficient idempotent halfspace range searching data structures than the real RAM model? An improved data structure that worked in the real RAM model would be of practical interest, even if it relied on the uniform distribution of the points.

In Section 2, we formalize some definitions. In Section 3, we introduce halfspace range searching data structures for different versions of the problem. In Section 4, we introduce the halfbox quadtree, which answers spherical and simplex queries. In Section 5, we briefly mention approximate data structures for orthogonal and general convex ranges.

2 Preliminaries

In this section, we provide basic definitions and discuss some results that are used throughout the paper. We start by formally defining the absolute model.

Given a range $R \in \mathcal{R}$ and an approximation error $\varepsilon > 0$, we define R^+ as the locus of points x such that $dist(x, R) \leq \varepsilon$. We define R^- as the locus of points x such that $dist(x, \overline{R}) \geq \varepsilon$, where $\overline{R}$ is the complement of R. We say that R_ε *ε-approximates R within B* if $R^- \cap B \subseteq R_\varepsilon \cap B \subseteq R^+ \cap B$. We say that R_ε *ε-approximates R* if R_ε ε-approximates R within $[0,1]^d$. We say $q_\varepsilon(R)$ is an *ε-approximation* of $q(R)$ if there is R_ε such that $q_\varepsilon(R) = q(R_\varepsilon)$ and R_ε approximates R.

We define a computational model, called the *Approximate Semigroup Arithmetic model* (*ASA model* for short), which makes it easier to describe our data structures. The ASA model is similar to the semigroup arithmetic model [10, 8, 5]. We explain how to convert our approximate data structures from the ASA model to the real RAM model (with integer division), preserving the same query time and storage space.

Given a collection of sets $\mathcal{S}$, let $\bigcup(\mathcal{S}) = \bigcup_{S\in\mathcal{S}} S$. In the semigroup version, we say that a set of regions $\mathcal{G}$ and a function $g : \mathcal{R} \to 2^{\mathcal{G}}$ *ε-generates* $\mathcal{R}$ if, for all $R \in \mathcal{R}$, the sets in $g(R)$ are pairwise disjoint, and $\bigcup(g(R))$ ε-approximates R. The elements of $\mathcal{G}$ are called *generators*. In the idempotent version the elements of $g(R)$ do not need to be pairwise disjoint, because $x + x = x$ for all $x \in \mathbf{S}$. In the group version, as the elements of the semigroup $(\mathbf{S}, +)$ have an inverse, the generators can be summed and subtracted in a multiset fashion, as long as

the final result is a set, that is, no point is counted more than once, or counted a negative number of times. For simplicity, we use $\bigcup(g(R))$ to refer to the sums and subtractions of generators in the group version.

Our definition of generators is different than the standard semigroup arithmetic model definition. We define a generator as a region of the space. In the semigroup arithmetic model, a generator is defined to be a linear form of weights of a set of data points. Our definition is more natural for the data structures described in this paper, and can be generalized to handle non-discrete point sets.

In the ASA model, a set $\mathcal{G}$ and a function g that ε-generates $\mathcal{R}$ is called a *data structure* for $\mathcal{R}$. The *storage space* of the data structure is defined as $|\mathcal{G}|$, and the *query time* is defined as $T(\mathcal{G}, \mathcal{R}) = \max_{R \in \mathcal{R}} |g(R)|$. We say that a data structure provides *internal approximation* if $\bigcup(g(R)) \subseteq R$, for all $R \in \mathcal{R}$, and provides *external approximation* if $R \subseteq \bigcup(g(R))$, for all $R \in \mathcal{R}$. Modifying our data structures to provide either internal approximation or external approximation is straightforward.

There would be little use to develop upper bounds using the ASA model, if we could not convert the data structures from the ASA model to a more standard model of computation such as the real RAM model. The ASA model (as well as the semigroup arithmetic model) considers neither preprocessing time nor the time to identify the proper generators for a given range. Identifying the proper generators consists of computing $g(R)$ efficiently, and is a simple task for the data structures we present, with the exception of the exact halfspace data structure from Section 3.3. *Preprocessing* consists of computing $w(G) = \sum_{p \in P \cap G} w(p)$, for all $G \in \mathcal{G}$. We may modify our set of generators $\mathcal{G}$ in order to obtain a set that is faster to preprocess, by using an approximation of each $G \in \mathcal{G}$, instead of G itself, when computing $w(G)$. We provide details on how to preprocess our data structures efficiently throughout the text. Therefore, our data structures work in the real RAM model, with the exception of the exact halfspace data structure from Section 3.3, which works in the semigroup arithmetic model.

3 Halfspace Range Searching

In the halfspace range searching problem, the range space is the set of all halfspaces. Brönnimann, Chazelle, and Pach [8] showed that a data structure with $m \geq n$ storage space takes $\widetilde{\Omega}(n^{1-\frac{d-1}{d(d+1)}}/m^{\frac{1}{d}})$ query time. The lower bound uses the semigroup arithmetic model, and holds on the expected case when the data points are uniformly distributed in the unit hypercube.

The most efficient exact data structure known for the semigroup version is due to Matoušek [15] and has $\widetilde{O}(n/m^{1/d})$ query time with m storage space. For small d, the gap between the best general lower bound and the best upper bound is significant. For example, when $m = O(n)$ and $d = 2$, there is a $\widetilde{\Omega}(n^{1/3})$ lower bound, and a $O(n^{1/2})$ upper bound.

Arya, Malamatos, and Mount [5] studied the importance of the semigroup being idempotent. A semigroup $(\mathbf{S}, +)$ is *idempotent* if $x + x = x$ for all $x \in \mathbf{S}$, and is *integral* if $kx \neq x$ for all $x \in \mathbf{S} \setminus \{0\}$, and $k \in \mathbb{N}^+$. They showed that, when the semigroup is integral, the lower

bound for exact halfspace range searching can be improved to $\widetilde{\Omega}(n/m^{(d+1)/(d^2+1)})$. They also used idempotence to develop more efficient spherical range searching data structures, in the relative model. We show that idempotence can be used not only to improve halfspace range searching in the absolute model, but also in the exact version assuming uniform distribution.

Chazelle, Liu and Magen [11] considered the approximate version of the problem in the absolute model, but not assuming that the dimension d is a constant. They presented a data structure with $O((d/\varepsilon)^2 \log^{O(1)}(d/\varepsilon))$ query time, and $dn^{O(1/\varepsilon^2)}$ storage.

In Section 3.1, we show that the approximate version can be solved in $O(1)$ query time, $O(1/\varepsilon^d)$ space, and $\widetilde{O}(n + 1/\varepsilon^d)$ preprocessing time. This is noteworthy, given the high complexity of the exact version. In Section 3.2, we make use of idempotence to achieve a space-time tradeoff, building a data structure with $m \geq 1/\varepsilon^{(d+1)/2}$ storage space and $O(1/m\varepsilon^d)$ query time. In Section 3.3, we use the approximate idempotent data structure to improve the best known bounds of exact range searching, in the semigroup arithmetic model, when the points are uniformly distributed in the unit cube. In Section 3.4, we improve the idempotent data structure for the case of emptiness queries.

Without loss of generality, we consider the range space $\mathcal{R}$ to be the set of halfspaces of the form $x_d \leq b + a_1 x_1 + \cdots + a_{d-1} x_{d-1}$ with $-1 \leq a_1, \ldots, a_{d-1} \leq 1$. We call the terms $a_1, \ldots, a_{d-1}$ *slopes* and b the x_d*-intercept*. An arbitrary halfspace can be converted into this form through an appropriate rotation. As only $2d$ different rotations are necessary, one data structure can be kept for each rotated set of points, without changing our asymptotic results.

3.1 Approximate Semigroup Version

In this section, we provide a data structure to solve approximate halfspace range searching with $O(1)$ query time, $O(1/\varepsilon^d)$ storage space, and $\widetilde{O}(n + 1/\varepsilon^d)$ preprocessing time. The general idea is to define a sufficiently large set $\mathcal{G}$ of halfspaces, so that any query halfspace is approximated by some halfspace in $\mathcal{G}$. As no two halfspaces in $\mathcal{G}$ are too similar to each other, efficient preprocessing requires building approximate data structures for subdivisions of the unit cube.

We define the set of generators $\mathcal{G}$ as the set that contains $\emptyset$, $[0,1]^d$, and the halfspaces whose boundary intersect the unit hypercube and have the slopes and the x_d-intercept as multiples of a parameter ε' to be specified later. Therefore, $\mathcal{G}$ contains $O(1/\varepsilon'^d)$ halfspaces. Given a halfspace $R \in \mathcal{R}$, the function $g(R)$ is the set containing the single halfspace obtained by rounding all slopes and the x_d-intercept of R to the closest multiple of ε'. If the boundary of R does not intersect the unit hypercube, then $g(R)$ is defined as $\{[0,1]^d\}$ if $[0,1]^d \subseteq R$, and $\{\emptyset\}$ if $R \cap [0,1]^d = \emptyset$.

Lemma 3.1 *$g(R)$ $(d\varepsilon'/2)$-approximates R.*

Proof: The case when the boundary of R does not intersect the unit hypercube is trivial. Let $a_1, \ldots, a_{d-1}, b$ denote the slopes and the x_d-intercept of R, and $a'_1, \ldots, a'_{d-1}, b'$ denote the slopes and the x_d intersect of $g(R)$. With some simple manipulations, we can show that the

maximum distance δ between the boundaries of R and R', along the x_d axis and inside the unit box, is

$$\delta \leq |b - b'| + \sum_{i=1}^{d-1} |a_i - a'_i|.$$

As b' and a'_i are obtained by rounding b and a_i to the closest multiples of ε', we have $|b - b'| \leq \varepsilon'/2$, and $|a_i - a'_i| \leq \varepsilon'/2$. Therefore, $\delta \leq d\varepsilon'/2$. □

To build an ε-approximate data structure, we set $\varepsilon' = 2\varepsilon/d$. Using Lemma 3.1, we have:

Theorem 3.2 *$(\mathcal{G}, g)$ is an ε-approximate halfspace range searching data structure, for the semigroup version, with $O(1/\varepsilon^d)$ storage space, $O(1)$ query time, and $\widetilde{O}(n + 1/\varepsilon^d)$ preprocessing time.*

It is easy to implement the query algorithm in the real RAM model, without changing the storage space or the query time, as long as integer division is available. Preprocessing the data structure efficiently is not trivial, though. We discuss two ways to perform the preprocessing. Both approaches produce an approximation factor greater than ε, therefore the parameter ε needs to be scaled accordingly. The first way is similar to Chan's discrete Voronoi diagram construction [9]:

1. Divide the unit hypercube in $1/\varepsilon$ horizontal slices, each of thickness ε.
2. Project the points from each slice to a $(d-1)$-dimensional horizontal hyperplane that is parallel to the slice boundary and passes through the center of the slice.
3. Recursively compute a $(d-1)$-dimensional approximate halfspace range searching data structure for each slice. Use the 0-dimensional case as a trivial base case.
4. For each generator $G \in \mathcal{G}$ ($|\mathcal{G}| = 1/\varepsilon^d$), and each slice s (out of $1/\varepsilon$ slices), perform a query for the intersection of G and s using the data structure for the slice s. Make $w(G)$ the sum of the results of all queries from generator G.

Disregarding the additive term of $O(n)$, the preprocessing time $T(d)$ for a d-dimensional data structure satisfies $T(0) = O(1)$, and $T(d) = O(1/\varepsilon^{d+1}) + (1/\varepsilon)T(d-1) = O(1/\varepsilon^{d+1})$.

The data structure obtained this way will not be an ε-approximate data structure, as the error accumulates through the d levels of the recursion. Additionally, projecting the points from a slice into a hyperplane adds an error of $\varepsilon/2$ at each level. Nevertheless, the data structure is $O(\varepsilon)$-approximate (more precisely, $(3d/2\varepsilon)$-approximate).

The second way to preprocess the data structure is:

1. Divide the unit hypercube in 2^d identical hypercubes.
2. Recursively compute an approximate halfspace range searching data structure for each subdivision. Use the case when the hypercube has diameter ε as a base case.

3. For each generator $G \in \mathcal{G}$ ($|\mathcal{G}| = 1/\varepsilon^d$), and each subdivision s (out of 2^d subdivisions), perform a query for the intersection of G and s using the data structure for the subdivision s. Make $w(G)$ the sum of the results of all queries from generator G.

Disregarding the additive term of $O(n)$, the preprocessing time $T(\delta)$ for a data structure of diameter δ satisfies $T(\varepsilon) = O(1)$, and $T(\delta) = O(\delta/\varepsilon^d) + 2^d T(\delta/2) = O(\delta \log(\delta/\varepsilon)/\varepsilon^d)$.

We should note that the error accumulates through $O(\log(1/\varepsilon))$ levels. Consequently, the data structure is $O(\varepsilon \log(1/\varepsilon))$-approximate. We can set $\varepsilon = \varepsilon'/\log(1/\varepsilon')$, and obtain an $O(\varepsilon')$-approximate data structure in $O(n + \log^{d+1}(1/\varepsilon')/\varepsilon'^d)$ preprocessing time.

3.2 Approximate Idempotent Version

In this section, we make use of idempotence to achieve a space-time tradeoff, building a data structure with $m \geq 1/\varepsilon^{(d+1)/2}$ storage space and $O(1/m\varepsilon^d)$ query time. The idea is to use a set of properly placed large balls as generators. There are two sources of approximation error: one comes from the fact that we are approximating flat surfaces with balls, and the second one comes from the fact that we may use balls that are not exactly tangent to the surface being approximated. First, we present a scheme involving an infinite number of generators, which addresses the first issue. Then, we reduce this to a finite set of generators.

Let $r > \sqrt{d} + 1$ be a constant, and let $\varepsilon < 1/2$ be an approximation parameter. We define $\mathcal{G}'$ to be the set of balls B of radius r such that B is centered at $(x_1, \ldots, x_d)$ where $x_1, \ldots, x_{d-1}$ are multiples of $\sqrt{r\varepsilon}/d$. Note that $|\mathcal{G}'|$ is infinite. Given a halfspace $R \in \mathcal{R}$, let $g'(R)$ be the subset of balls from $\mathcal{G}'$ that are tangent to the boundary of R and are contained in R.

Lemma 3.3 *$(\mathcal{G}', g')$ $(\varepsilon/2)$-generates $\mathcal{R}$.*

Proof: As $r > \sqrt{d}$, it suffices to show that every point in the boundary of R, and inside the unit hypercube, is within distance at most $\varepsilon/2$ of some ball in $g(R)$.

Look at the point of tangency $t(B)$ between a ball $B \in g(R)$ and the boundary of R. We can prove (using the Pythagorean Theorem) that there is a $(d-1)$-dimensional ball B', on the surface of R, of radius greater than $\sqrt{r\varepsilon}$, and centered at $t(B)$, such that all points inside B' are within distance $\varepsilon/2$ of B. The points $t(B)$, for $B \in g(R)$, form a grid of distance at most $\sqrt{r\varepsilon}/\sqrt{d}$ on the surface of R. As the radius of B' is at least $\sqrt{r\varepsilon}$, every point in the surface of R is contained in at least one ball B'. (Figure 1). □

We now define $\mathcal{G} \subset \mathcal{G}'$ as the set of balls B such that

1. B has radius r;
2. there is a hyperplane h with slopes between -1 and 1, that is tangent to B at point p, with $p \in [-\sqrt{r\varepsilon}/d, 1 + \sqrt{r\varepsilon}/d]^{d-1} \times [0, 1 + \sqrt{d}]$; and
3. B is centered at $(x_1, \ldots, x_d)$ where $x_1, \ldots, x_{d-1}$ are multiples of $\sqrt{r\varepsilon}/d$, and x_d is a multiple of $\varepsilon/2$.

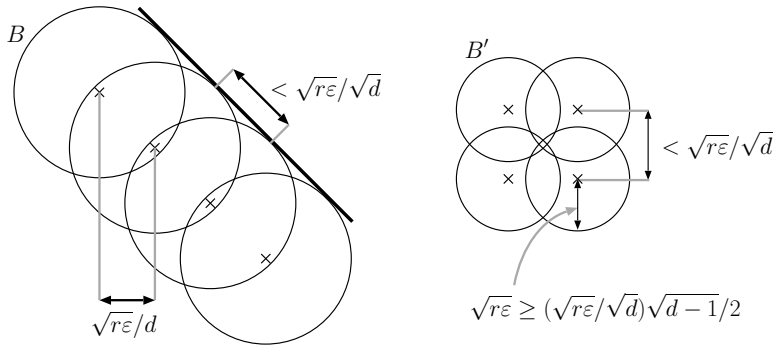

Figure 1: Proof of Lemma 3.3.

We define a *column* of $\mathcal{G}$ as the set of balls with centers having the same $x_1, \ldots, x_{d-1}$ coordinates. The function $g(R)$ is obtained by replacing each ball B' from $g'(R)$ that approximates the boundary of R within $[0,1]^d$, with the closest ball $B \in \mathcal{G}$ such that $B \subset R$. Intuitively, the ball B is the ball immediately below B' in the same column.

Theorem 3.4 *$(\mathcal{G}, g)$ is an ε-approximate halfspace range searching data structure, for the idempotent version, with internal approximation, $m \geq 1/\varepsilon^{(d+1)/2}$ storage space, $O(1/m\varepsilon^d)$ query time, and $O(n + \log^{d+1}(1/\varepsilon)/\varepsilon^d)$ preprocessing time.*

Proof: (sketch) The balls in $\mathcal{G}$ are arranged in $O((r/\varepsilon)^{(d-1)/2})$ columns, and there are $O(1/\varepsilon)$ balls in each column, therefore the storage space is $|\mathcal{G}| = m = O(r^{(d-1)/2}/\varepsilon^{(d+1)/2})$. The query time is $O((1/r\varepsilon)^{(d-1)/2}) = O(1/m\varepsilon^d)$, for $r > 1 + \sqrt{d}$. We defer the discussion on how to preprocess the data structure until Section 4.1.

The set $\mathcal{G}$ has a ball within distance $\varepsilon/2$ from any ball in $\mathcal{G}'$ that can ε-approximate the boundary of a halfspace in $\mathcal{R}$ inside the unit hypercube. From Lemma 3.3, we conclude that the set $g(R)$ ε-approximates R. □

3.3 Exact Idempotent Version

In this section, we show how to use the approximate idempotent data structure to build an exact halfspace range searching data structure in the semigroup arithmetic model. To our knowledge, this is the first data structure to match the $\widetilde{\Omega}(n^{1-2/(d+1)})$ lower bound [8] when the storage space is linear, and the data points are uniformly distributed within the unit cube. We do not provide an efficient way to determine the set of generators for a given query. Consequently, the results hold only for the semigroup arithmetic model.

The general idea of the data structure is to properly set the parameter ε used in the data structure from Section 3.2, in order to make the expected number of points in the fuzzy boundary equal to the query time of the approximate data structure. Generators for individual points can be used to count the points in the fuzzy boundary.

In this section, we assume that the n data points are uniformly distributed in the unit hypercube $[0,1]^d$. Let $m \geq n$ denote the storage space, and $\varepsilon = O((nm)^{-1/(d+1)})$. We apply

Theorem 3.4 to build an ε-approximate data structure $(\mathcal{G}_1, g_1)$ with $O(m)$ storage space and query time

$$O\left(\frac{1}{m\varepsilon^d}\right) = O\left(\frac{n^{1-1/(d+1)}}{m^{1/(d+1)}}\right) = O(n\varepsilon).$$

As the data structure (G_1, g_1) provides internal approximation, no data points outside R are counted, but some data points inside R and within distance ε from the boundary may not be counted. To answer the query exactly, we set $\mathcal{G}_2 = \{\{p\} : p \in P\}$ and define $g_2(R)$ as the set of generators $\{p\} \in \mathcal{G}_2$ such that $p \in R$ and p is within distance ε from the boundary of R. As the data points are uniformly distributed within the unit hypercube, for any fixed $R \in \mathcal{R}$, the expected number of generators in $g_2(R)$ is $E(|g_2(R)|) = O(n\varepsilon)$.

Let $\mathcal{G}' = \mathcal{G}_1 \cup \mathcal{G}_2$, and $g'(R) = g_1(R) \cup g_2(R)$. To use the standard semigroup arithmetic model definition of generators, let $\mathcal{G}$ denote the set of linear forms $\sum_{p \in P \cap G'} w(p)$ for all $G' \in \mathcal{G}'$.

Theorem 3.5 *$\mathcal{G}$ is an exact halfspace range searching data structure in the semigroup arithmetic model with $m \geq n$ storage space, and $O(n^{1-1/(d+1)}/m^{1/(d+1)})$ expected query time assuming that the points are uniformly distributed in a hypercube. When $m = n$, the query time is $O(n^{1-2/(d+1)})$.*

Theorem 3.5 matches the lower bound of $\widetilde{\Omega}(n^{1-(d-1)/d(d+1)}/m^{1/d})$ [8] for the case of $m = n$ (up to logarithmic factors). The lower bound is also in the semigroup arithmetic model, and also assumes that the points are uniformly distributed inside the unit hypercube. Therefore, improving the lower bound for the case of $m = n$, if at all possible, would require either a different model of computation or a different set of data points.

3.4 Approximate Emptiness Version

In this section, we show how to reduce the storage space of the idempotent data structure for the special case of range emptiness queries, that is, for the semigroup $(\{0, 1\}, \vee)$. Our approach is to modify the data structure from Section 3.2 to obtain some kind of monotonicity, and then apply compression to reduce the storage space.

For a set of points B, let $\tau(B) = \{(x_1, \ldots, x_d) : \exists (x_1, \ldots, x_{d-1}, x'_d) \in B \text{ with } x'_d > x_d\}$. If B is a ball, then $\tau(B)$ is a bullet-shaped object formed by the ball and a semifinite cylinder extending downwards. We modify the set $\mathcal{G}$ from Section 3.2 into a set $\mathcal{G}_{emp} = \{\tau(B) : B \in \mathcal{G}\}$. It is not hard to see that Theorem 3.4 still holds if we replace $\mathcal{G}$ with $\mathcal{G}_{emp}$.

The set $\mathcal{G}_{emp}$ can be compressed, because generators $G \in \mathcal{G}_{emp}$ with the same $x_1, \ldots, x_{d-1}$ coordinates can be ordered by their x_d coordinates, and the value of $w(G)$ can only change once. Therefore, the storage space requirement is reduced by a factor of $O(1/\varepsilon)$. We omit the details on how to preprocess the data structure.

Theorem 3.6 *There is an ε-approximate halfspace range searching data structure for the emptiness version with $m \geq 1/\varepsilon^{(d-1)/2}$ storage space, $O(1/m\varepsilon^{d-1})$ query time, and $O(n+1/\varepsilon^d)$ preprocessing time.*

Another approach to this problem consists of computing an ε-kernel [2, 9] containing $O(1/\varepsilon^{\frac{d-1}{2}})$ points, and then using an exact halfspace emptiness data structure with the ε-kernel as the set of points. The latter approach attains lower query times for the case of $O(1/\varepsilon^{(d-1)/2})$ space, but involves complex data structures from [14] for $d > 3$.

4 Halfbox Quadtree

In this section, we introduce a data structure called the *halfbox quadtree*, and present some applications. In Section 4.1, we analyze the query time of the halfbox quadtree for spherical ranges. We also show how an additional set of generators can be used to provide a space-time tradeoff. In Section 4.2, we analyze the query time of the halfbox quadtree for simplex ranges, in the group version.

A *quadtree box* is a box that can be obtained by recursively dividing the unit hypercube into 2^d identical hypercubes. We define a *half quadtree box* (*halfbox* for short) as the intersection of a quadtree box and a halfspace. The *size* of a halfbox is the diameter of the corresponding quadtree box. A *halfbox quadtree* is formed by associating each quadtree box Q of diameter at least ε with an ε-approximate halfspace range searching data structure for the bounding box Q. If we use the halfspace data structure from Theorem 3.2, the resulting halfbox quadtree has $O(\log(1/\varepsilon)/\varepsilon^d)$ storage space, $O(n + \log^{d+1}(1/\varepsilon)/\varepsilon^d)$ preprocessing time, $O(1)$ query time for halfbox ranges.

4.1 Approximate Spherical Range Searching

In spherical range searching, the ranges are Euclidean balls. The exact version of the problem can be reduced to halfspace range searching by projecting the points onto an appropriate $(d+1)$-dimensional paraboloid [13]. Spherical range queries in P are equivalent to halfspace range queries in P'. Arya, Malamatos and Mount [3, 5] present approximate data structures, in the relative model, with $m \geq n\log(1/\varepsilon)$ space and $\widetilde{O}(n^{1-1/d}/m^{1/d}\varepsilon^{d-1})$ query time, for general semigroups, and $\widetilde{O}(n^{1/2-1/2d}/m^{1/2d}\varepsilon^{(d-1)/2})$ query time for idempotent semigroups.

In this section, we show that the halfbox quadtree answers spherical range queries in $O(1/\varepsilon^{\frac{d-1}{2}})$ time. The data structure works for general semigroups. We also show how to reduce the query time by increasing the space, adding some extra generators to the data structure. Figure 2 illustrates how a small number of halfboxes can approximate a ball significantly better than a larger number of quadtree boxes. Lemma 4.1 formalizes this result.

Lemma 4.1 *A ball B of radius r can be ε-approximated by a set of disjoint half quadtree boxes, where each half quadtree box has size $\Omega(\sqrt{r\varepsilon})$.*

Proof: Consider a quadtree box Q of diameter δ such that no corresponding halfbox ε-approximates B. Consider a halfspace h that contains B and whose boundary is tangent to B in some point inside Q. As $h \cap Q$ does not approximate B, there must be a line segment

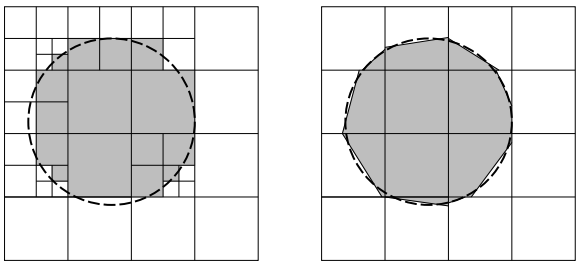

Figure 2: Approximating a ball using quadtree boxes (left) and half quadtree boxes (right).

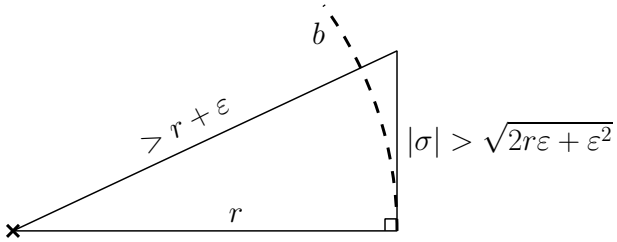

Figure 3: Triangle used in the proof of Lemma 4.1.

$\sigma \subset (h \cap Q)$ of length $|\sigma|$ that contains a point within distance greater than ε from B. Using the Pythagorean Theorem with the triangle from Figure 3, we have $|\sigma| > \sqrt{2r\varepsilon + \varepsilon^2}$. As $|\sigma| < diam(B)$, the theorem holds. □

The following packing lemma allows us to bound the query time as a function of the size of the quadtree boxes (this follows from Lemma 3 in [6]).

Lemma 4.2 *If $\mathcal{Q}$ is a set of pairwise disjoint quadtree boxes, each of diameter at least δ, that intersect the boundary of a convex range of diameter $\Delta \geq \delta$, then $|\mathcal{Q}| = O((\Delta/\delta)^{d-1})$.*

Theorem 4.3 *The halfbox quadtree is an ε-approximate spherical range searching data structure with $O(\log(1/\varepsilon)/\varepsilon^d)$ storage space, $O(1/\varepsilon^{(d-1)/2})$ query time, and $O(n + \log^{d+1}(1/\varepsilon)/\varepsilon^d)$ preprocessing time. If the query ball B has radius r, then the query time is $O((\min(r,1)/(r+1)\varepsilon)^{(d-1)/2})$.*

Proof: If we start with Q as the unit hypercube, and recursively subdivide Q until we can approximate B by a halfbox associated with Q, we notice that a recursive call is only performed when Q intersects the boundary of B. Let $\delta = O(1/2^i)$ denote the diameter of Q at level i in the recursion tree, and $\Delta = O(\min(r,1))$ be the diameter of the intersection of B and the unit box. Using Lemmas 4.1, and 4.2, and summing the number of recursive calls, we have $O((\min(r,1)/(r+1)\varepsilon)^{(d-1)/2})$ query time. □

The halfbox quadtree can be used to preprocess the data structure from Theorem 3.4. The total preprocessing time for a data structure of size $m = r^{(d-1)/2}/\varepsilon^{(d+1)/2}$, with $1 < r < 1/\varepsilon$, is

$$O\left(n + \frac{\log^{d+1}(1/\varepsilon)}{\varepsilon^d} + m\left(\frac{1}{r\varepsilon}\right)^{\frac{d-1}{2}}\right) = O\left(n + \frac{\log^{d+1}(1/\varepsilon)}{\varepsilon^d}\right) = \widetilde{O}\left(n + \frac{1}{\varepsilon^d}\right).$$

The spherical range searching data structure from Theorem 4.3 has minimum storage space, except for a logarithmic factor (because we could place $1/\varepsilon^d$ points in a grid, and retrieve each individual weight using ranges of radius ε). It is natural to ask how we could improve the query time by increasing the storage space. Let $r > 1$ be a parameter, and consider the set of generators $\mathcal{G}$ formed by $\emptyset$, $[0,1]^d$, and the balls B such that

1. B has radius at most r,
2. the radius of B is a multiple of $\varepsilon/2$,
3. the boundary of B intersects $[0,1]^d$, and
4. B is centered at $(x_1, \ldots, x_d)$ where $x_1, \ldots, x_d$ are multiples of $\varepsilon/2\sqrt{d}$.

Theorem 4.4 *The halfbox quadtree, together with $(\mathcal{G}, g)$, is an ε-approximate spherical range searching data structure with $m = r^d/\varepsilon^{d+1} > 1/\varepsilon^{d+1}$ storage space, $O(1/m^{\frac{1}{2}-\frac{1}{2d}}\varepsilon^{d-\frac{1}{2}-\frac{1}{2d}})$ query time, and $O(n + m/\varepsilon^{(d-1)/2})$ preprocessing time.*

Proof: The storage space is dominated by $|\mathcal{G}| = O(r^d/\varepsilon^{d+1}) = m$. Any ball $R \in \mathcal{R}$ of radius at most $r > 1$ is ε-approximated by a ball $g(R) \in \mathcal{G}$. Therefore, queries with radius at most r can be answered in $O(1)$ time. Using the halfbox quadtree, queries with radius greater than r can be answered in

$$O\left(\left(\frac{1}{r\varepsilon}\right)^{\frac{d-1}{2}}\right) = O\left(\frac{1}{m^{\frac{1}{2}-\frac{1}{2d}}\varepsilon^{d-\frac{1}{2}-\frac{1}{2d}}}\right) \text{ time.}$$

□

4.2 Approximate Simplex Range Searching

In simplex range searching, the ranges are d-dimensional simplices. Chazelle [10] proved that, if m units of storage are allowed, then the query time is $\Omega(n/\sqrt{m})$ in the plane, and $\Omega((n/\log n)/m^{1/d})$ in d-dimensional space. In the exact version, the most efficient linear size data structure is due to Matoušek [15] and has $O(n^{1-1/d})$ query time.

We show how to answer ε-approximate simplex queries in $O(\log(1/\varepsilon))$ time for $d = 2$ and $O(1/\varepsilon^{d-2})$ time for $d \geq 3$, using the halfbox quadtree. Our query algorithm requires the use of a subtraction operation, and so applies only in the group setting.

We recursively answer a query $q(Q, R)$ with a simplex range R in the quadtree box Q, starting with Q as the unit hypercube. If $Q \cap R = \emptyset$, then we return 0. If $Q \cap R = B$, then we return the precomputed $w(Q) = \sum_{p \in P \cap Q} w(p)$. If Q does not contain any $(d-2)$-faces of R, then there is a set H of at most $d+1$ halfspaces that form the complement of R, and the halfspaces in H are pairwise disjoint. Then, we can return $w(Q) - \sum_{h \in H} q_\varepsilon(h \cap Q)$, where $h \cap B$ is a halfbox, and $q_\varepsilon(\cdot)$ is the result of the approximate query using the halfbox quadtree. If the diameter of Q is less than ε, then we verify whether R contains the center of

Q and answer accordingly. Otherwise, we return the sum of $q(Q', R)$ for all 2^d subdivisions Q' of Q.

We can use the following packing lemma to bound the query time.

Lemma 4.5 *If C is a set of pairwise disjoint quadtree boxes, each of diameter at least δ, that intersect a $(d-2)$-dimensional convex polytope of constant diameter, then $|C| = O(1 + 1/\delta^{d-2})$.*

To analyze the query time, we look at the recursion tree of the query algorithm. The diameter of the quadtree boxes at level ℓ is $\Theta(2^{-\ell})$. The query algorithm only makes a recursive call when Q intersects the a $(d-2)$-face of R. As R is the intersection of a constant number of halfspaces, the number of $(d-2)$-faces of R is also constant. As R is convex, each $(d-2)$-face of R inside the unit box is a convex polytope of constant diameter. It follows from Lemma 4.5 that the number of recursive calls at level ℓ is $\Theta(2^{\ell(d-2)})$. Summing the number of recursive calls for all $\log(\Theta(1/\varepsilon))$ levels we conclude:

Theorem 4.6 *The halfbox quadtree is an ε-approximate range searching data structure for simplex ranges, in the group version, with $O(1/\varepsilon^d \log(1/\varepsilon))$ storage space, $O(\log(1/\varepsilon))$ query time for $d = 2$, $O(1/\varepsilon^{d-2})$ query time for $d \geq 3$, and $O(n + \log^{d+1}(1/\varepsilon)/\varepsilon^d)$ preprocessing time.*

5 Other Results

An important aspect of this work is the observation that the absolute model allows efficient approximate range searching with very simple methods. As further evidence of this, in this section, we briefly examine approximate range searching data structures for two additional problems: orthogonal ranges and convex ranges. The set of *orthogonal ranges* is the set of all axis-aligned hyper-rectangles. The set of *convex ranges* is the set of all d-dimensional convex shapes.

For the case of orthogonal ranges, we create a set P' of $O(1/\varepsilon^d)$ grid aligned points where, for any point $p \in P$, there is a point $a(p) \in P'$ with $dist(p, a(p)) \leq \varepsilon$. We define the weight $w(p')$, for $p' \in P'$, as the sum of $w(p)$ for the points p with $p' = a(p)$. Then, we build a d-dimensional array A, ranging from 1 to $O(1/\varepsilon)$ in each dimension, using the weights of the points in P'. A partial sum query in A is equivalent to an ε-approximate orthogonal range query for P. If subtraction is allowed, partial sum queries can be answered in $O(1)$ time with $O(1/\varepsilon^d)$ storage space. Without subtraction, partial sum queries take $O(\alpha(m, 1/\varepsilon^d))$ time with $m \geq 1/\varepsilon^d$ storage space [20, 12].

For the case of convex ranges, we define $\mathcal{G}$ as the set of quadtree boxes of diameter at least 2ε. Note that $|\mathcal{G}| = O(1/\varepsilon^d)$. The function $g(R)$ is defined as the set of quadtree boxes from $\mathcal{G}$ whose centers are contained in $\mathcal{R}$. The data structure $(\mathcal{G}, g)$ answers convex range queries in $O(1/\varepsilon^{d-1})$ time. We can efficiently compute $g(R)$ in the real RAM model if we assume that we can determine whether a range intersects a quadtree box in $O(1)$ time (as in [6]). A variation of this data structure uses a compressed quadtree to store arbitrarily

small quadtree boxes. In this variation, the parameter ε only needs to be provided at query time, and the storage space is $O(n)$.

References

[1] P. K. Agarwal and J. Erickson. Geometric range searching and its relatives. In B. Chazelle, J. Goodman, and R. Pollack, editors, *Advances in Discrete and Computational Geometry*, pages 1–56. American Mathematical Society, 1998.

[2] P. K. Agarwal, S. Har-Peled, and K. R. Varadarajan. Geometric approximation via coresets. In J. E. Goodman, J. Pach, and E. Welzl, editors, *Combinatorial and Computational Geometry*, MSRI Publications. Cambridge Univ. Press, 2005.

[3] S. Arya, T. Malamatos, and D. M. Mount. Space-time tradeoffs for approximate spherical range counting. In *16th Ann. ACM-SIAM Symposium on Discrete Algorithms, (SODA'05)*, pages 535–544, 2005.

[4] S. Arya, T. Malamatos, and D. M. Mount. The effect of corners on the complexity of approximate range searching. In *Proceedings of the 22nd ACM Symp. on Computational Geometry (SoCG'06)*, pages 11–20, 2006.

[5] S. Arya, T. Malamatos, and D. M. Mount. On the importance of idempotence. In *Proc. 38th ACM Symp. on Theory of Computing (STOC'06)*, pages 564–573, 2006.

[6] S. Arya and D. M. Mount. Approximate range searching. *Comput. Geom. Theory Appl.*, 17(3-4):135–152, 2000.

[7] B. Babcock, S. Babu, M. Data, R. Motwani, and J. Widom. Models and issues in data stream systems. In *Proc. ACM Princ. Database Systems*, pages 1–16, 2002.

[8] H. Brönnimann, B. Chazelle, and J. Pach. How hard is half-space range searching. *Discrete & Computational Geometry*, 10:143–155, 1993.

[9] T. M. Chan. Faster core-set constructions and data stream algorithms in fixed dimensions. In *Proceedings of the 20th ACM Symp. on Computational Geometry (SoCG'04)*, pages 152–159, 2004.

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

[11] B. Chazelle, D. Liu, and A. Magen. Approximate range searching in higher dimension. In *Proceedings of the 16th Canadian Conference on Computational Geometry (CCCG'04)*, pages 154–157, 2004.

[12] B. Chazelle and B. Rosenberg. Computing partial sums in multidimensional arrays. In *Proc. 5th ACM Symp. Comput. Geom. (SoCG'89)*, pages 131–139, 1989.

[13] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwartzkopf. *Computational Geometry: Algorithms and Applications*. Springer, 2000.

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

[15] J. Matoušek. Range searching with efficient hiearchical cuttings. *Discrete & Computational Geometry*, 10:157–182, 1993.

[16] J. Matoušek. Geometric range searching. *ACM Computing Surveys*, 26(4):421–461, 1994.

[17] S. Muthukrishnan. *Data streams: Algorithms and applications*. Now Publishers, 2005.

[18] S. Suri, C. D. Tóth, and Y. Zhou. Range counting over multidimensional data streams. *Discrete & Computational Geometry*, 36(4):633–655, 2006.

[19] R. E. Tarjan. Efficiency of a good but not linear set union algorithm. *J. ACM*, 22(2):215–225, 1975.

[20] A. C. Yao. Space-time tradeoff for answering range queries. In *Proc. 14th ACM Symp. on Theory of Computing (STOC'82)*, pages 128–136, 1982.