Tradeoffs in Approximate Range Searching Made Simpler

Sunil Arya *
Department of Computer Science
The Hong Kong University of
Science and Technology
Clear Water Bay, Kowloon, Hong Kong
arya@cs.ust.hk

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

Abstract

Range searching *is a fundamental problem in computational geometry. The problem involves preprocessing a set of n points in $\mathbb{R}^d$ into a data structure, so that it is possible to determine the subset of points lying within a given* query range*. In* approximate range searching*, a parameter $\varepsilon > 0$ is given, and for a given query range R the points lying within distance $\varepsilon \cdot diam(R)$ of the range's boundary may be counted or not. In this paper we present three results related to the issue of tradeoffs in approximate range searching. First, we introduce the* range sketching problem*. Next, we present a space-time tradeoff for smooth convex ranges, which generalize spherical ranges. Finally, we show how to modify the previous data structure to obtain a space-time tradeoff for simplex ranges. In contrast to existing results, which are based on relatively complex data structures, all three of our results are based on simple, practical data structures.*

*Research supported by the Research Grants Council of Hong Kong, China under project number 610106.

†Research supported by CAPES and CNPq, Brazil, respectively under grants BEX-1319027 and PDJ-151194/2007-6.

‡Research supported by the National Science Foundation under grant CCR-0635099.

1 Introduction

The *range searching problem* is among the fundamental problems in computational geometry. The problem consists of preprocessing a set P of n points in $\mathbb{R}^d$ into a data structure, so that it is possible to count or enumerate the points lying within a given query range R. The query range is chosen from a predetermined *set of ranges*, for example, the set of halfspaces, simplices, or Euclidean balls. Excellent surveys have been written by Matoušek [12] and Agarwal and Erickson [1].

The relatively high complexity of exact range searching has led researchers to consider the problem in the context of approximation. We consider the range shape to be "fuzzy," and allow points that are close to the range's boundary to either be counted or not. There are two natural ways to define geometric approximation for range searching. In both cases a user-supplied approximation parameter $\varepsilon > 0$ is given. In the *relative model* [3–6] it is assumed that the query range R is bounded and points lying within distance $\varepsilon \cdot diam(R)$ of the range boundary may or may not be included. In the *absolute model* [9], points lying within distance ε of the boundary of the range may or may not be included, regardless of the diameter of the range. In this paper we consider the relative model, but some of our data structures have been derived from recent work in the absolute model [9].

1530-1834/08 $25.00 © 2008 IEEE
DOI 10.1109/SIBGRAPI.2008.24

This paper presents three new results, all of which involve tradeoffs of various forms in the context of approximate range searching. Tradeoffs are important in many retrieval problems, since they provide users the ability to improve access times depending on the amount of space at hand. Our first result is a handy utility, called a *range sketching query*, which intuitively provides a limited resolution summary of the points lying within or near a query range. The other two results involve space-time tradeoffs for approximate range searching, one for *smooth convex ranges* and the other for *simplex ranges*. These two results are based on variants of a novel and simple data structure, called the *(relative) halfbox quadtree*. Throughout, we assume that the dimension $d \geq 2$ is a constant and that the model of computation is the real RAM with integer division.

1.1 Range Sketching

The concept of *sketching* arises frequently in approximation algorithms. In the context of geometric computation it refers to computing a limited resolution approximation to some quantity of interest. For example, sketching has been employed in streaming computations, where huge volumes of data and limited memories forbid exact computations [11]. We introduce the concept of a *range sketching query*. To motivate this concept, it is useful to consider two well known extreme cases, range counting and range reporting. In the former, the contents of a range are described in the most concise terms possible (as a count or weight), but no geometric information regarding the locations of the points is given. In the latter, the locations of the points in the range are given, but the output size and query time may be unbounded.

In many applications what is really desired is a compromise between these two extremes, that is, a concise summary of the distribution of points that lie within the range, albeit with limited resolution. Our approach is to summarize the result of the range query as a disjoint union of weighted hypercubes whose sizes are controlled by a user-supplied parameter and where the weight of each box indicates the number of points lying within it. Range sketching can be seen as query version of the notion of coresets [2]. While a core set concisely describes the structure of an entire set of points, a range sketching query provides similar information within a given region of space.

To make this more formal, let s be a *resolution parameter* that is specified at query time. Given a query range R, the result of a *sketching query* $q_s(R)$ is a set of pairwise disjoint hypercubes $\{Q_1, \ldots, Q_k\}$ that cover $P \cap R$ and, for $1 \leq i \leq k$: (i) $\frac{s}{2} < diam(Q_i) \leq s$, (ii) $Q_i \cap R \neq \emptyset$, (iii) $Q_i \cap P \neq \emptyset$, and (iv) Q_i is associated with a weight $w(Q_i) = \sum_{p \in P \cap Q_i} w(p)$ (see the Figure 1).

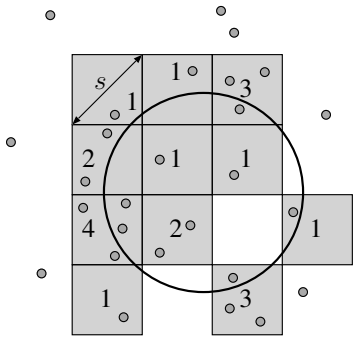

Figure 1. Range sketching.

The output size k is proportional to the number of non-empty quadtree boxes (see Section 2.1) of diameter roughly s that intersect R. Thus, the output size can be controlled through s. Let $c > 0$ be an arbitrary constant, and R^+ denote the locus of the points within distance at most $c \cdot diam(R)$ from R. Let k' be the smallest number of non-empty quadtree boxes of diameter at most s that intersect R^+. We show how to answer range sketching queries in $O(\log n + k')$ time with $O(n)$ storage space and $O(n \log n)$ preprocessing time, for arbitrary ranges. The data structure consists of a balanced variant of the well known quadtree data structure [7, 10]. Our results assume the same unit-cost test assumption as in [6], that is, given a range R and quadtree box Q, we can determine in $O(1)$ time whether $Q \subseteq R$, $R \cap Q = \emptyset$, or neither. Note that answering a range sketching query requires $\Omega(\log n + k)$ time in the decision-tree model and both k and k' are $O(1 + (diam(R)/s)^d)$.

1.2 Smooth Convex Ranges

Space-time tradeoffs have emerged as an important topic in many geometric retrieval problems, such as range searching, in which it is difficult to achieve efficient query times using roughly linear space. In order to better understand the issue in the context of approximate range searching, it is illustrative to consider Figure 2. It presents a plot of query times for approximate spherical range searching for 100,000 uniformly distributed points in $\mathbb{R}^{10}$ when run on the kd-tree structure, as implemented in the ANN library. The library implements a practical version of the approximate range searching algorithm described in [6]. The asymptotic query time of this structure is $O\left(\log n + 1/\varepsilon^{d-1}\right)$, which ignoring the

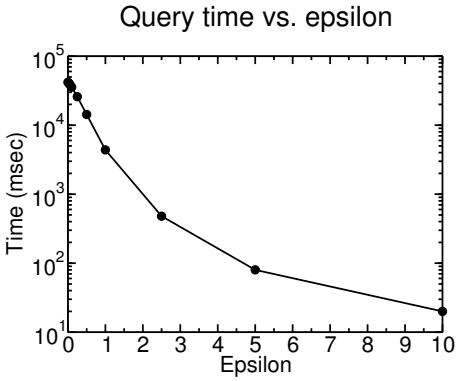

Figure 2. Importance of ε-dependencies.

ε dependencies, is essentially optimal. Observe, however, that as the value of ε varies from 10 down to 0.001, the query times grow from around 20 milliseconds to over $40,000$. Thus, in the case of small ε, 99.95% of the query time is due to the ε dependencies.

Given the burden of such high query times it is natural to wonder whether it is possible to achieve better query times at the price of more space. We will consider this problem from a more general perspective than simple spherical range searching, by considering both smooth and simplex ranges. Current work based on AVDs (see, e.g., [3]) has had considerable theoretical success, but the results rely on methods whose technical complexity renders them inappropriate for implementation. Here we consider approaches based on simple, well known data structures, therefore amenable for efficient implementation.

In a general range searching formulation, we have a commutative semigroup $(\mathbf{S}, +)$, a *weight* function $w : P \rightarrow \mathbf{S}$, and want to compute $\sum_{p \in P \cap R} w(p)$. This is called the *semigroup version*. In the more restrictive *group version* $(\mathbf{S}, +)$ 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. Another restriction is the *idempotent version* in which $(\mathbf{S}, +)$ is an idempotent semigroup, i.e., $x + x = x$ for all $x \in \mathbf{S}$. A special case of the semigroup version is the *reporting version*, where the semigroup is $(2^P, \cup)$ and $w(p) = \{p\}$. Range reporting deserves special attention because we cannot store an arbitrary set of points in $O(1)$ space.

For $\kappa \geq 1$, we say that a range R is *κ-smooth* if, for any point p on the boundary of R, there exists some Euclidean ball completely inside R that touches p and has radius at least $diam(R)/(2\kappa)$. (Observe that a Euclidean ball is 1-smooth, and a convex polytope is not κ-smooth for any finite κ.) A range is said to be *smooth* if it is κ-smooth for some constant κ.

The best upper bounds for approximate range searching for smooth convex ranges over arbitrary semigroups were given by Arya and Mount [6]. They showed that queries could be answered in space $O(n)$ and time $O(\log n + 1/\varepsilon^{d-1})$ but provided no space-time tradeoffs. Approximate smooth range searching was later studied in [4]. It was shown there that, irrespective of space, there is a lower bound of $\Omega\left(\log n + 1/\varepsilon^{(d-1)/2}\right)$ on the query time. They presented a data structure of linear space for the special case of idempotent semigroups, which nearly matches this lower bound, but they left open the question of space-time tradeoffs for more general semigroups.

Here we fill this gap by providing the first space-time tradeoffs for smooth convex ranges and general semigroups. We introduce a data structure called the *(relative) halfbox quadtree*. It is a variant of the halfbox quadtree introduced by Fonseca [9] for the absolute model. This simple data structure involves extending the aforementioned balanced quadtree with an auxiliary (flat) data structure for answering approximate halfspace queries in the absolute model. We show that for $1 \leq \gamma \leq 1/\sqrt{\varepsilon}$, given space $O(n\gamma^d)$ this data structure answers queries in time $O(\log n + 1/(\varepsilon\gamma)^{d-1})$. The data structure can be preprocessed in optimal $O(n \log n + n\gamma^d)$ time.

At the low-space extreme ($\gamma = 1$) this matches the earlier results of Arya and Mount [6]. At the high-space extreme ($\gamma = 1/\sqrt{\varepsilon}$) our upper bound of $O\left(\log n + 1/\varepsilon^{(d-1)/2}\right)$ query time matches the lower bound. Moreover, the tradeoffs are nearly tight throughout the entire range. In particular, Arya, Malamatos, and Mount [5] showed that for general semigroups, given $O(n\gamma^d)$ space, there is a lower bound of $\Omega\left(\log n + 1/(\varepsilon\gamma)^{d-O(1)}\right)$ on the query time for spherical range searching. Since spheres are smooth, this lower bound applies to smooth ranges as well. Indeed, our data structure can be used for spherical range searching, and for the given range of γ values, our data structure is both simpler, and more efficient (by logarithmic factors in the preprocessing and query times and storage space) than the AVD-based data structure for approximate spherical range searching given by Arya, Malamatos, and Mount [3].

1.3 Simplex Ranges

Although simplex range searching is arguably the most well studied problem in exact range searching,

there is little previous work on approximate simplex range searching in the relative model. The best result to date is the solution given in [6], which answers queries for arbitrary convex ranges in $O(\log n + 1/\varepsilon^{d-1})$ time with $O(n)$ space. It provides no space-time tradeoffs. Later, Arya *et al.* [4] presented a lower bound for answering approximate range queries for rotated unit hypercubes over integral semigroups. They showed that for $0 \leq k \leq d-1$, given n/ε^{k^2} space, $\Omega(\log n + 1/\varepsilon^{d-2-2k})$ query time is required. This lower bound applies to simplex range searching as well, but we should note that the lower bound does not allow for the use of subtraction.

We show that for any integer k, $0 \leq k \leq d-1$, we can extend the relative halfbox quadtree to a k-level data structure (where each additional level handles an additional hyperplane). We obtain a data structure that can answer approximate simplex range queries in the group version. Our data structure uses $O\left(n/\varepsilon^{k^2 d/(d-1)}\right)$ storage space and answers queries in $O\left(\log n + \log(1/\varepsilon) + 1/\varepsilon^{d-1-k}\right)$ time. Although simplex ranges are significantly more general than rotated unit hypercubes — the latter are rigid transformations of a single fixed object — our upper bound essentially matches the lower bound for rotated unit hypercubes. Given the same query time, both the lower and upper bound on space are $\Theta\left(n/\varepsilon^{\Theta(k^2)}\right)$.

The remainder of the paper is organized as follows. In Section 2, we explain background results and provide some definitions. In Section 3, we consider the range sketching problem. In Section 4, we introduce the (relative) halfbox quadtree and show how to answer approximate range queries for smooth ranges. Finally, in Section 5, we introduce the multilevel halfbox quadtree and show how to use it to answer approximate simplex range queries.

2 Preliminaries

2.1 Quadtrees

A *quadtree* is a hierarchical partition of the space into d-dimensional hypercubes. The root of the quadtree corresponds to a bounding hypercube for the set of data points. An internal node has 2^d children corresponding to the disjoint subdivisions of the parent hypercube. A leaf is a node which contains a single data point. A *quadtree box* is defined recursively as the original bounding hypercube or the hypercubes obtained by evenly dividing a quadtree box. The quadtree box associated with node v is denoted by $v_\square$.

A *compressed quadtree*, is obtained by replacing all maximal chains of nodes that have a single non-empty child by a single node associated with the coordinates of the smallest quadtree box containing the data points. A compressed quadtree storing n points can be built in $O(n \log n)$ time [7, 10]. The size of a compressed quadtree is $O(n)$, but the height of the tree can be as high as $\Theta(n)$. To overcome this deficiency, Arya and Mount [7] introduced the BBD-tree, which combines quadtree splits with a centroid decomposition construction. In this paper, instead of using a BBD-tree, we use a related data structure by Har-Peled [10] that consists of a compressed quadtree that is threaded with a separate finger tree of height $O(\log n)$.

A *separator* for a tree T with n nodes is a node v such that removing v from T produces a forest F where every tree in F has at most $n/2$ nodes. Every tree has a separator, and it can be computed in $O(n)$ time by following the largest subtrees starting from the root, until the separator is reached. We build the *finger tree* T' by setting the root of T' to be a separator v of T, and the children of v to be the root of the finger trees of the trees obtained by removing v from T. The finger tree T' has $O(n)$ size, $O(\log n)$ height and can be built in $O(n \log n)$ time [10].

An important type of query that can be answered in $O(\log n)$ time with the use of a finger tree is called a cell query [10]. Let T be a compressed quadtree for the set P of data points. Given a query quadtree box Q, a *cell query* consists of finding the only quadtree box Q' in T such that $P \cap Q = P \cap Q'$, if it exists. The quadtree box Q' is unique because T is compressed and Q' exists if and only if $P \cap Q \neq \emptyset$.

2.2 Geometric Approximation Models

Given a range R and a real value $\alpha > 0$, we define $R^{+\alpha}$ as the locus of points within distance at most α from R and we define $R^{-\alpha}$ as the locus of points within distance at least α from the complement of R. We say that a region R_α *α-approximates* R if $R^{-\alpha} \subseteq R_\alpha \subseteq R^{+\alpha}$. We say that a region R_α *α-approximates R within* a region Q if

$$R^{-\alpha} \cap Q \subseteq R_\alpha \cap Q \subseteq R^{+\alpha} \cap Q.$$

The result of an exact range query is defined as $q(R) = \sum_{p \in P \cap R} w(p)$. We define $q_\alpha(R) = \sum_{p \in P \cap R_\alpha} w(p)$ for some region R_α that α-approximates R. In the *absolute model*, answering an ε-approximate query consists of computing $q_\varepsilon(R)$. In the

relative model, answering an ε-approximate query for a range R of diameter Δ consists of computing $q_{\Delta\varepsilon}(R)$. The data structures described in this paper work in the relative model, but make use of the absolute model data structure from [9] described below for completeness.

Let ε be the approximation parameter and δ be the diameter of the set of n data points in d-dimensional space. Given the assumption of a model of computation that supports integer division, halfspace range searching in the absolute model can be answered in $O(1)$ query time with $O((\delta/\varepsilon)^d)$ storage space and $O(n + (\delta/\varepsilon)^d \log^{d+1}(\delta/\varepsilon))$ preprocessing time [9]. (Without this assumption the query time increases to $O(\log(\delta/\varepsilon))$.)

Without loss of generality, we assume that the query halfspaces are of the form $x_d \leq b + a_1x_1 + \cdots + a_{d-1}x_{d-1}$ 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 involving the permutation of the coordinate axes. As only $2d$ different rotations are necessary, one data structure can be kept for each rotated set of points, without changing our asymptotic results. We also assume that the boundary of the query halfspace intersects the bounding hypercube, because otherwise the problem is trivial.

The ε-approximate halfspace range searching data structure consists of a lookup table where the results of halfspace range queries for a predetermined set of halfspaces are stored. The set of halfspaces stored in the lookup table is the set of halfspaces whose boundary intersect the bounding hypercube and have the slopes and the x_d-intercept as multiples of $2\varepsilon/d\delta$. The size of the lookup table is $O((\delta/\varepsilon)^d)$. To answer an approximate halfspace range query, we round the slopes and x_d-intercept of the query halfspace to the nearest multiple of $2\varepsilon/d\delta$, and return the value stored in corresponding table entry. The query time is $O(1)$.

2.3 Packing Lemmas

A *packing lemma* limits the number of disjoint *objects* that can intersect some *region*, as a function of the size of the objects and the region. In our case, the objects are always quadtree boxes. We present a set of packing lemmas, which are important to analyze the query time for our data structures. The following packing lemma follows from Lemma 2 in [6].

Lemma 2.1. *If $\mathcal{Q}$ is a set of pairwise disjoint quadtree boxes, each of diameter at least δ, that intersect a region R of diameter $\Delta \geq \delta$, then $|\mathcal{Q}| = O((\Delta/\delta)^d)$.*

We can improve the previous result when the region consists of the $(d-1)$-dimensional boundary of a convex d-dimensional region. The following packing lemma follows from Lemma 3 in [6].

Lemma 2.2. *If $\mathcal{Q}$ is a set of pairwise disjoint quadtree boxes, each of diameter at least δ, that intersect the boundary of a convex region R of diameter $\Delta \geq \delta$, then $|\mathcal{Q}| = O((\Delta/\delta)^{d-1})$.*

A $(d-k)$-face is *flat* if it is a subset of a $(d-k)$-dimensional hyperplane. If the region consists of flat $(d-k)$-faces, then we can obtain an improved bound. The simple proof has been omitted.

Lemma 2.3. *Given an integer constant k, where $0 \leq k \leq d-1$, if $\mathcal{Q}$ is a set of pairwise disjoint quadtree boxes, each of diameter at least δ, that intersect a flat $(d-1-k)$-face of diameter $\Delta \geq \delta$, then $|\mathcal{Q}| = O((\Delta/\delta)^{d-1-k})$.*

3 Range Sketching

Let P be a set of n points in $\mathbb{R}^d$ and s be a parameter specified at query time. Recall that, given a range R, the result of a *range sketching query* $q_s(R)$ is a set of pairwise disjoint hypercubes $Q_1, \ldots, Q_k$ satisfying the properties described in Section 1. The output size k is proportional to the smallest number of non-empty quadtree boxes of diameter at most s that intersect R. Let $c > 0$ be an arbitrary constant, and R^+ denote the locus of the points within distance at most $c \cdot diam(R)$ from R. Let k' be the smallest number of non-empty quadtree boxes of diameter at most s that intersect R^+. In this Section, we show how to answer range sketching queries in $O(\log n + k')$ time with $O(n)$ storage space and $O(n \log n)$ preprocessing time, for arbitrary ranges. The algorithm is quite simple and is similar in structure to the approximate range search algorithm given by Arya and Mount [6]. A straightforward extension of their algorithm would result in a running time of $O(k' \log n)$, however.

Let $c > 0$ be a constant. Let a be a valid diameter for a quadtree box between $c \cdot diam(R)$ and $c \cdot diam(R)/2$. To answer a range sketching query, we define the set A of quadtree boxes of diameter a that intersect R. Let B be the set formed by the result of cell queries for each element of A. Using Lemma 2.1, we have $|B| = |A| = O(1)$. Also, B can be computed in $O(\log n)$ time.

We answer the range sketching query by applying the following query algorithm for each $v \in B$:

1. If $v_\square \cap R = \emptyset$, then return $\emptyset$.

2. If v is a leaf node, then check whether the point stored in v is contained in R. If yes, return the quadtree box of diameter between $s/2$ and s that contains the point. Otherwise return $\emptyset$.

3. If $diam(v_\square) \leq s$, then return $\{v\}$.

4. Otherwise, return the union of the recursive calls of the procedure for each child of v.

To analyze the time complexity, we look at the set of recursion trees of the query algorithm above for each node $v \in B$. By construction, the subtrees rooted at the nodes of B are disjoint and only contain points within distance c from R. Because T is a compressed quadtree, every non-leaf node v has at least 2 children, and contains more than one data point. Therefore, the number of leaves and also the number of internal nodes in the recursion trees is at most k'. The following theorem summarizes this result.

Theorem 3.1. *There exists a linear space data structure which answers a range sketching query in $O(\log n + k')$ time, where k' is the smallest number of nonempty quadtree boxes of diameter at most s that are within distance $c \cdot diam(R)$ from the query range, and $c > 0$ is an arbitrary constant.*

4 Smooth Convex Ranges

Let $\gamma \geq 1$ be a parameter. We define a *relative halfbox quadtree* as a compressed quadtree T, threaded with a finger tree T', where each quadtree box $v_\square$ of diameter δ stored in T is associated with an absolute model (δ/γ)-approximate halfspace range searching data structure.

A (δ/γ)-approximate halfspace range searching data structure for a quadtree box of diameter δ takes $O(\gamma^d)$ storage space. Since T has $O(n)$ nodes, the total storage space for the relative halfbox quadtree is $O(n\gamma^d)$.

To preprocess T, we start by building a compressed quadtree and a finger tree in $O(n \log n)$ time, using the method described in [7, 10]. The leaves of the compressed quadtree contain a single data point. Therefore, we can build a halfspace range searching data structure associated with the leaf nodes in $O(\gamma^d)$ time for each leaf node. The data structure for an internal node v can be built in $O(\gamma^d)$ time by performing at most 2^d queries to the children of v for each entry in the lookup table. Note that the quadtree boxes of the children of a node v have at most half the size of the quadtree box of v, therefore the total error that accumulates in the data structure associated with the root of the tree is bounded by a factor of 2. The preprocessing algorithm takes $O(n \log n + n\gamma^d)$ time, which is optimal.

Recall the definition of κ-smooth ranges given in the introduction. The following lemma is the a central utility for answering smooth range queries efficiently. The proof is straightforward and has been omitted.

Lemma 4.1. *Let R be an κ-smooth range of diameter Δ, ε an approximation parameter, and Q a quadtree box of diameter δ. If $\delta \leq \Delta\sqrt{\varepsilon/\kappa}$, then any halfspace h whose bounding hyperplane is tangent to R at a point within Q $\Delta\varepsilon$-approximates R within Q.*

Instead of restricting ourselves to smooth ranges of bounded complexity, and specifying how the smooth range is stored, we assume that a certain operation can be performed in $O(1)$ time. This approach is similar to the one used in [4, 6], but our assumption is slightly stronger. We assume that, given a smooth convex range R and a quadtree box Q, in constant time we can determine whether $Q \subseteq R$, $Q \cap R = \emptyset$, or neither and, in the latter case, we can also find in constant time a hyperplane that is tangent to R at some point $x \in Q$. Note that this assumption is easily satisfied for spherical ranges.

Let $c > 0$ be a constant, and R be a query range of diameter Δ. Let a be a valid diameter for a quadtree box, with a between $c\Delta$ and $c\Delta/2$. To answer a range query $q_{\Delta\varepsilon}(R)$, we first determine the set A of quadtree boxes of diameter a that intersect R. By Lemma 2.1, we have $|A| = O(1)$. Let V be the set formed by the result of cell queries for each element of A. Note that $|V| = |A| = O(1)$, and V can be computed in $O(\log n)$ time. The query is answered by performing the following procedure for each $v \in V$, and summing the results.

1. If $v_\square \cap R = \emptyset$, then return 0.

2. If $v_\square \subset R$, then return the precomputed $w(v) = \sum_{p \in P \cap v_\square} w(p)$.

3. If v is a leaf, then verify whether the point stored in v is contained in R and return the weight of the point or 0, accordingly.

4. If $diam(v_\square)/\gamma \leq \Delta\varepsilon/2$ and $diam(v_\square) \leq \Delta\sqrt{\varepsilon/(2\kappa)}$, then determine a halfspace h that is tangent to R inside $v_\square$, and return the precomputed $q_{diam(v_\square)/\gamma}(h \cap v_\square)$.

5. Otherwise, return the sum of the recursive calls of the procedure for each child of v.

We only need to show the correctness of Step 4, since the other steps are clearly correct. To show that Step 4

is correct, we note that, by Lemma 4.1, the halfspace h $(\Delta\varepsilon/2)$-approximates R within $v_\square$. Also, v is associated with a $(\Delta\varepsilon/2)$-approximate halfspace data structure, because $diam(v_\square)/\gamma \leq \Delta\varepsilon/2$. Adding both approximation errors, we obtain a $(\Delta\varepsilon)$-approximation.

To analyze the query time when κ is a constant, we note that a recursive call is only performed when $v_\square$ intersects the boundary of R and

$$diam(v_\square) = \Omega(\Delta\varepsilon\gamma + \Delta\sqrt{\varepsilon}).$$

Using Lemma 2.2, and summing the number of recursive calls, we conclude that the query takes $O(1/(\varepsilon\gamma)^{d-1} + 1/\varepsilon^{(d-1)/2})$ time. Note that there is no advantage in setting $\gamma > 1/\sqrt{\varepsilon}$. The following theorem summarizes the result.

Theorem 4.2. *The relative halfbox quadtree with $1 \leq \gamma \leq 1/\sqrt{\varepsilon}$ is an ε-approximate range searching data structure for smooth ranges, in the relative model, with $O(n\gamma^d)$ storage space, $O(\log n + 1/(\varepsilon\gamma)^{d-1})$ query time, and $O(n \log n + n\gamma^d)$ preprocessing time.*

5 Simplex Ranges

The relative halfbox quadtree can also be used to answer simplex range queries, as long as subtraction is allowed. In order to obtain a space-time tradeoff, we build a multi-level data structure using k levels of the approximate halfspace range searching data structure for each node in the quadtree, where k is an integer parameter between 0 and $d-1$.

Let v be a node in the compressed quadtree and δ be the diameter of $v_\square$. The top level (δ/γ)-approximate absolute model halfspace range searching data structure associated with v remains exactly the same as before. For each halfspace h stored in the lookup table for a quadtree box $v_\square$, we store another (δ/γ)-approximate halfspace range searching data structure for the point set $P \cap v_\square \cap h$. We repeat this for a total of k levels. This structure enables us to compute $q_{\delta/\gamma}(h_1 \cap \ldots \cap h_k \cap v_\square)$ in $O(1)$ time, by successively querying the k levels of the data structure. The storage space of the portion of the data structure corresponding to any given node in the quadtree is $O(\gamma^{dk})$, and the total storage space for the data structure is $O(n\gamma^{dk})$. Using the same approach from Section 4, the preprocessing time is $O(n \log n + n\gamma^{dk})$.

Let R be a query simplex of diameter Δ. We first determine a set of $O(1)$ nodes V, in $O(\log n)$ time, in the same way as for smooth ranges. Then the range query is answered by performing the following procedure for each $v \in V$, and summing the results.

1. If $v_\square \cap R = \emptyset$, then return 0.
2. If $v_\square \subset R$ or $diam(v_\square) \leq \Delta\varepsilon$, then return the precomputed $w(v) = \sum_{p \in P \cap v_\square} w(p)$.
3. If v is a leaf, then check whether the point stored in v is contained in R and return the weight of the point or 0, accordingly.
4. If $diam(v_\square)/\gamma \leq \Delta\varepsilon$ and $v_\square$ does not contain any $(d-1-k)$-faces of R, then there is a set H of at most $d+1$ halfspaces that form the complement of R. Since $v_\square$ does not contain $(d-1-k)$-faces, the intersection of $v_\square$ and a subset of more than k halfspaces from H is equal to the intersection of $v_\square$ and at most k halfspaces from H. Using the inclusion-exclusion principle, approximate the sum s of the weights of the points in the union of the halfspaces of H. Then, return $w(v) - s$.
5. Otherwise, return the sum of the recursive calls of the procedure for each child of v.

To analyze the query time, we note that a recursive call is only performed when either (i) $v_\square$ intersects the boundary of R and $diam(v_\square) > \Delta\varepsilon\gamma$, or (ii) $v_\square$ intersects a $(d-1-k)$-face of R and $diam(v_\square) > \Delta\varepsilon$. Using Lemmas 2.2 and 2.3, it follows that the number of recursive calls is $O(\log(1/\varepsilon) + 1/(\varepsilon\gamma)^{d-1} + 1/\varepsilon^{d-1-k})$.

Including the time for finding the initial set of nodes V, the query time is, therefore, $O(\log n + \log(1/\varepsilon) + 1/(\varepsilon\gamma)^{d-1} + 1/\varepsilon^{d-1-k})$. We set $\gamma = 1/\varepsilon^{\frac{k}{d-1}}$ to balance the contributions of the last two terms. The following theorem summarizes the result.

Theorem 5.1. *For $k \in \{0, \ldots, d-1\}$, there is an ε-approximate range searching data structure for simplex ranges, in the relative model and group version, with $O\left(n/\varepsilon^{\frac{k^2 d}{d-1}}\right)$ storage space, $O(\log n + \log(1/\varepsilon) + 1/\varepsilon^{d-1-k})$ query time, and $O\left(n \log n + n/\varepsilon^{\frac{k^2 d}{d-1}}\right)$ preprocessing time.*

A simplex is *fat* if the angle between any pair of bounding hyperplanes is at least a constant. If the query simplex is fat, then Theorem 5.1 holds for the semigroup version. We omit the details due to space limitations, but a similar construction is presented in [8].

6 Conclusion

In this paper, we provided data structures for three problems related to range searching: range sketching

and approximate smooth range searching, and approximate simplex range searching. All data structures are based on the compressed quadtree together with a finger tree, which is somewhat simpler than the BBD-tree (used in [3–7]).

The relative halfbox quadtree combines a compressed quadtree with a data structure for halfspace range searching in the absolute model [9]. The result is a data structure that allows the arbitrary cutting angles provided by halfspaces and the varying box sizes provided by the quadtree. The data structure provides space-time tradeoffs for smooth, spherical and simplex ranges.

The relative halfbox quadtree was presented for the semigroup and group versions of the problem. In the semigroup version, the halfbox quadtree answers smooth, spherical and fat simplex queries. A naive implementation of the data structure for the reporting version would consist of independently storing the sets of points for each position in the lookup tables. This naive version would require storage space quadratic in n. To obtain a more efficient reporting version of the data structure, we should store links that simulate the behavior of the preprocessing algorithm, and apply compression in the same way as the range reporting data structure for the absolute model presented in [8]. Then, we obtain a data structure for approximate range reporting with the same complexities as the semigroup version except for the additional time required to output the set of points in the range.

The following two weaknesses of the relative halfbox quadtree are important topics for future research. First, the space-time tradeoff for spherical range searching is limited to query times that are not faster than $\Theta(\log n + 1/\varepsilon^{(d-1)/2})$. While this query time is optimal for arbitrary smooth ranges [4], it can be improved for spherical ranges, at the cost of additional storage space [3]. Second, we do not know how to improve the space-time tradeoff of the relative halfbox quadtree for the case of idempotent semigroups. Significantly faster data structures for approximate idempotent spherical range searching exist [5]. The data structure for approximate idempotent halfspace range searching in the absolute model [9] could possibly be used to obtain an idempotent version of the halfbox quadtree.

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. AMS, 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] S. Arya, D.M. Mount, N.S. Netanyahu, R. Silverman, and A.Y. Wu. An optimal algorithm for approximate nearest neighbor searching fixed dimensions. *J. ACM*, 45(6):891–923, 1998.

[8] G. D. da Fonseca. *Approximate Range Searching in the Absolute Error Model*. PhD thesis, University of Maryland, College Park, 2007.

[9] G.D. da Fonseca. Approximate range searching: The absolute model. In *Proceedings of the 10th International Workshop on Algorithms and Data Structures (WADS'07)*, pages 2–14, 2007.

[10] S. Har-Peled. Geometric approximation algorithms. `http://valis.cs.uiuc.edu/~sariel/teach/notes/aprx/`.

[11] Q. Lv, M. Charikar, and K. Li. Image similarity search with compact data structures. In *30th ACM int. conf. on Information and knowledge management (CIKM'04)*, pages 208–217, New York, NY, USA, 2004. ACM.

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