

Space-Time Tradeoffs for Approximate Spherical Range Counting

Sunil Arya*

Theocharis Malamatos†

David M. Mount‡

Abstract

We present space-time tradeoffs for approximate spherical range counting queries. Given a set S of n data points in \mathbb{R}^d along with a positive approximation factor ϵ , the goal is to preprocess the points so that, given any Euclidean ball B , we can return the number of points of any subset of S that contains all the points within a $(1 - \epsilon)$ -factor contraction of B , but contains no points that lie outside a $(1 + \epsilon)$ -factor expansion of B .

In many applications of range searching it is desirable to offer a tradeoff between space and query time. We present here the first such tradeoffs for approximate range counting queries. Given $0 < \epsilon \leq 1/2$ and a parameter γ , where $2 \leq \gamma \leq 1/\epsilon$, we show how to construct a data structure of space $O(n\gamma^d \log(1/\epsilon))$ that allows us to answer ϵ -approximate spherical range counting queries in time $O(\log(n\gamma) + 1/(\epsilon\gamma)^{d-1})$. The data structure can be built in time $O(n\gamma^d \log(n/\epsilon) \log(1/\epsilon))$. Here n , ϵ , and γ are asymptotic quantities, and the dimension d is assumed to be a fixed constant.

At one extreme (low space), this yields a data structure of space $O(n \log(1/\epsilon))$ that can answer approximate range queries in time $O(\log n + (1/\epsilon)^{d-1})$ which, up to a factor of $O(\log 1/\epsilon)$ in space, matches the best known result for approximate spherical range counting queries. At the other extreme (high space), it yields a data structure of space $O((n/\epsilon^d) \log(1/\epsilon))$ that can answer queries in time $O(\log n + \log 1/\epsilon)$. This is the fastest known query time for this problem.

We also show how to adapt these data structures to the problem of computing an ϵ -approximation to the k th nearest neighbor, where k is any integer from 1 to n given at query time. The space bounds are identical to the range searching results, and the query time is larger only by a factor of $O(1/(\epsilon\gamma))$.

Our approach is broadly based on methods developed for approximate Voronoi diagrams (AVDs), but it involves a number of significant extensions from the context of nearest neighbor searching to range searching. These include generalizing AVD node-separation properties from leaves to internal nodes of the tree and constructing efficient generator sets through a radial decomposition of space. We have also developed new arguments to analyze the time and space requirements in this more general setting.

*Department of Computer Science, The Hong Kong University of Science and Technology, Clear Water Bay, Kowloon, Hong Kong. Supported by the Research Grants Council, Hong Kong, China (HKUST6080/01E). Email: arya@cs.ust.hk.

†Max-Planck-Institut für Informatik, Im Stadtwald, D-66123 Saarbrücken, Germany. Email: tmalamat@mpi-sb.mpg.de.

‡Department of Computer Science and Institute for Advanced Computer Studies, University of Maryland, College Park, Maryland 20742. Partially supported by the National Science Foundation under grant CCR-0098151. Email: mount@cs.umd.edu.

1 Introduction

Answering range counting queries is among the most fundamental problems in spatial information retrieval and computational geometry. The objective is to store a finite set of points so that it is possible to quickly count the points lying inside a given query range. Examples of ranges include rectangles, spheres, halfspaces, and simplices. In this paper we consider the weighted, counting version of the problem for spherical ranges. More generally, we assume each point stores an element of a semigroup, and the objective is to compute the semigroup sum of points in the range.

Range searching is a well studied problem in computational geometry, and nearly matching upper and lower bounds exist for many formulations. The most relevant case for us is that of halfspace range counting queries. Matoušek [14] has shown that in dimension d , with space $O(m)$ it is possible to achieve a query time of $O(n/m^{1/d} \log(m/n))$, where $n \leq m \leq n^d$. Nearly matching lower bounds have been given in the semigroup arithmetic model, first for simplex range searching by Chazelle [8] and later for halfspace range searching by Brönnimann, Chazelle and Pach [6].

Spherical range searching involves ranges that are (closed) Euclidean balls. It is well known that by projecting the points onto an appropriate paraboloid, spherical range searching can be reduced to halfspace searching in \mathbb{R}^{d+1} [9]. Since a halfspace can be viewed as a sphere of infinite radius, lower bounds on halfspace range queries apply to spherical range queries as well. Unfortunately, the lower bounds on halfspace range searching destroy any reasonable hope of achieving the ideal of answering multidimensional spherical range queries in logarithmic query times using roughly linear storage space. This suggests the importance of pursuing approximation algorithms. Achieving speed-ups through approximation is reasonable in many applications in engineering and science where the data or ranges are imprecise [11, 13] and also in exact algorithms where approximations are used to obtain density estimates [15].

Let $b(p, r)$ denote a Euclidean ball in \mathbb{R}^d centered at a point p and having radius r . Given $\epsilon > 0$ and the range $b(p, r)$, a set $S' \subseteq S$ is an admissible solution to an ϵ -approximate range query if it contains all the

Table 1: Summary of results for ϵ -approximate spherical range counting (Range) and k th nearest neighbor queries (NN(k)), with low-space ($\gamma = 2$) and high-space ($\gamma = 1/\epsilon$). $O(\log 1/\epsilon)$ factors have been omitted.

Query	Resource	Tradeoff	Low-Space	High-Space
Range	Space	$n\gamma^d$	n	n/ϵ^d
	Query Time	$\log(n\gamma) + 1/(\epsilon\gamma)^{d-1}$	$\log n + 1/\epsilon^{d-1}$	$\log n$
NN(k)	Space	$n\gamma^d$	n	n/ϵ^d
	Query Time	$\log(n\gamma) + 1/(\epsilon\gamma)^d$	$\log n + 1/\epsilon^d$	$\log n$

points of a $(1 - \epsilon)$ -factor contraction of $b(p, r)$ and does not contain any point that lies outside a $(1 + \epsilon)$ -factor expansion of this ball, that is,

$$S \cap b(p, r(1 - \epsilon)) \subseteq S' \subseteq S \cap b(p, r(1 + \epsilon)).$$

An ϵ -approximate range counting query returns the exact number (or semigroup sum) of the points in any such admissible solution. Note that the range is approximated, not the count. Although the error is two-sided, it is an easy matter to modify the values of r and ϵ to generate only one-sided errors. Arya and Mount [3] considered this problem and showed that with $O(n \log n)$ preprocessing time and $O(n)$ space, ϵ -approximate range counting queries can be answered in time $O(\log n + 1/\epsilon^{d-1})$.

In range searching it is often desirable to offer a tradeoff between space and query time. Unfortunately, Arya and Mount's results on approximate range searching do not admit any such tradeoffs. In this paper we remedy this situation by offering space-time tradeoffs for approximate spherical range counting queries. Let S be a set of n points in \mathbb{R}^d , and let $0 < \epsilon \leq 1/2$ be the approximation bound. We take n and ϵ to be asymptotic quantities and assume that d is a constant. Given a parameter γ , where $2 \leq \gamma \leq 1/\epsilon$, we show how to construct a data structure of space $O(n\gamma^d \log(1/\epsilon))$ that can answer ϵ -approximate range queries in time $O(\log(n\gamma) + 1/(\epsilon\gamma)^{d-1})$. The data structure can be built in time $O(n\gamma^d \log(n/\epsilon) \log(1/\epsilon))$. Note that the construction time exceeds the space by a relatively modest factor of $O(\log(n/\epsilon))$.

At one extreme ($\gamma = 2$) this yields a data structure of space $O(n \log(1/\epsilon))$ that answers queries in time $O(\log n + 1/(\epsilon)^{d-1})$, thus matching the results of Arya and Mount [3] up to a factor of $\log(1/\epsilon)$ in the space. At the other extreme ($\gamma = 1/\epsilon$) this yields a data structure of space $O(n/(\epsilon^d) \log(1/\epsilon))$ that can answer queries in time $O(\log n + \log 1/\epsilon)$. To our knowledge, these are the fastest query times for approximate spherical range searching. These results are summarized in Table 1.

We also show how to adapt the data structure to answer approximate k th nearest neighbor queries. For

$1 \leq k \leq n$ and a real parameter $\epsilon > 0$, we say that a point $p \in S$ is an ϵ -approximate k th nearest neighbor of a point $q \in \mathbb{R}^d$, if $(1 - \epsilon)r_k \leq \|pq\| \leq (1 + \epsilon)r_k$, where r_k denotes the true distance from q to its k th closest point in S . As before, we provide a space-time tradeoff. Given γ , where $2 \leq \gamma \leq 1/\epsilon$, we can construct a data structure of space $O(n\gamma^d \log(1/\epsilon))$ that allows us to answer such queries in $O(\log(n\gamma) + 1/(\epsilon\gamma)^d)$ time. Note that the value of k is not assumed to be a constant and can be provided at query time. The data structure can be constructed in time $O(n(\gamma/\epsilon)^{d/2} \log(n/\epsilon) \log(1/\epsilon))$. The space bounds are identical to the range searching results, and the query time is larger only by a factor of $O(1/(\epsilon\gamma))$. To our knowledge, these are the best known results for approximate k th nearest neighbor searching.

Our earlier work on linear space structures for approximate nearest neighbor queries [1] suggested the problem of achieving space-time tradeoffs for spherical range queries. Virtually all range searching structures operate by precomputing a number of *generators*, each of which is a subset of the point set. For counting queries we require that the intersection of any range with the point set can be expressed as a disjoint cover of an (ideally small) set of generators. The number of points (or generally the semigroup sum) for each generator is precomputed along with a data structure for computing the generators needed to answer a query.

The principal challenge in providing space-time tradeoffs in our case is determining a good way of defining generators. A natural approach is to subdivide the range space so that queries that are sufficiently similar (in a metric sense, depending on ϵ) can take advantage of this by using roughly the same generator sets. Our approach is to subdivide space hierarchically into hypercube cells using a quadtree-like decomposition, and each node is responsible for handling queries whose center lies inside the corresponding cell and whose radius is proportional to the cell size.

The most difficult aspect of this approach is achieving good bounds on the space required, particularly for point sets that are not uniformly distributed and

may be highly clustered. Our approach is to adapt recent techniques derived for approximate nearest neighbor searching based on *approximate Voronoi diagram* (AVDs). Har-Peled [12] introduced the AVD of a point set S as a quadtree-like partition of space into cells, such that all the points within each leaf cell have the same approximate nearest neighbor. Later results by Arya *et al.* [1, 2] generalized this by allowing each cell to store a small number of representative points and showed how this can lead to significant space improvements. An important element of their work is the notion of subdividing space hierarchically into hypercubes so that certain *separation properties* hold with respect to the point set. Such properties assert that the region surrounding each leaf cell of the decomposition is simple enough that all the information needed for answering queries can be encoded succinctly.

To achieve our results we have generalized a number of elements of the AVD construction. We show how to extend the separation properties for nearest neighbor searching (which is closely related to approximate range emptiness queries) to apply to arbitrary range counting queries. We have developed new arguments to analyze the total space requirements. Another new element is generator construction. Because ranges are spherical, rather than using quadtree cells themselves to define generators, we have developed a more efficient method based on a radial generalization of a quadtree decomposition of space. This uses a polar representation of the points relative to the center of each cell. One of the appealing features of our overall approach to range searching is that it is based largely on quadtree decompositions and straightforward generalizations thereof. These data structures are easy to implement, and are not subject to numerical issues.

2 Preliminaries

Throughout we assume that the dimension d is a fixed constant, and treat n , ϵ and γ as asymptotic quantities. We assume that the set S of points has been scaled and translated to lie within a ball of radius $\epsilon/2$ placed at the center of the unit hypercube $[0, 1]^d$. Let x and y denote any two points in \mathbb{R}^d . We use $\|xy\|$ to denote the Euclidean distance between x and y and \overline{xy} to denote the segment joining x and y . We denote by $b(x, r)$ a closed ball of radius r centered at x , i.e., $b(x, r) = \{y : \|xy\| \leq r\}$. For a ball b and any positive real γ , we use γb to denote the ball with the same center as b and whose radius is γ times the radius of b , and \overline{b} to denote the set of points that are not in b .

We now briefly review the notion of box-decomposition trees, as they play an important role in our constructions and analyses. Intuitively, a box-

decomposition tree is an enhanced form of the well known quadtree structure and its higher dimensional generalizations. A (multidimensional) *quadtree* is a hierarchical decomposition of space into hypercubes in \mathbb{R}^d . Starting with the unit hypercube $U = [0, 1]^d$, a *quadtree box* is any d -cube that can be obtained by a recursive splitting process that starts with U and generally splits an existing quadtree box by d axis-orthogonal hyperplanes passing through its center into 2^d identical sub-cubes. Such a decomposition naturally defines a 2^d -ary tree, such that each node is associated with a cube, called its *cell*. The *size* of a quadtree box is its side length.

Quadtrees suffer from two shortcomings, which make them inappropriate for worst-case analysis. First, if points are densely clustered in some very small region of space, it is not possible to bound the number of quadtree splits needed to decompose the cluster by a function of n alone. The box-decomposition (BD) tree [4] overcomes this by introducing an additional decomposition operation called shrinking. Second, if the point distribution is not uniform, the tree may not have logarithmic depth. The balanced box-decomposition (BBD) tree [5] extends the BD tree and remedies this problem by employing a balanced shrinking operation.

More formally, a *box-decomposition (BD) tree* of a set S of n points is a 2^d -ary tree¹ that compactly represents a hierarchical decomposition of space [4]. A BD-tree cell is either a quadtree box or the set theoretic difference of two quadtree boxes, an *outer box* and an *inner box*. Cells of the former type are called *box cells* and cells of the latter type are called *doughnut cells*. As with the quadtree, the root node is associated with U . When a cell contains at most one point of S it is declared a leaf. There are two ways that an internal node can be decomposed. A *splitting operation* decomposes space into 2^d subcubes in exactly the same way as the quadtree. A *shrinking operation* decomposes a quadtree box u into two children. The inner child cell is the smallest quadtree box u' that contains $S \cap u$, and outer child cell is doughnut cell $u - u'$, which contains no points and hence is a leaf. The relevant properties of the BD tree are given below. Properties (i) and (iii) are proved in [4], and property (ii) is a simple generalization of (i).

- (i) The BD tree has $O(n)$ nodes and can be constructed in time $O(n \log n)$.
- (ii) A collection \mathcal{C} of n quadtree boxes can be stored in a BD tree with $O(n)$ nodes such that the

¹The tree defined in [4] is a binary version of this tree, but the assumption that all cells are hypercubes simplifies our presentation.

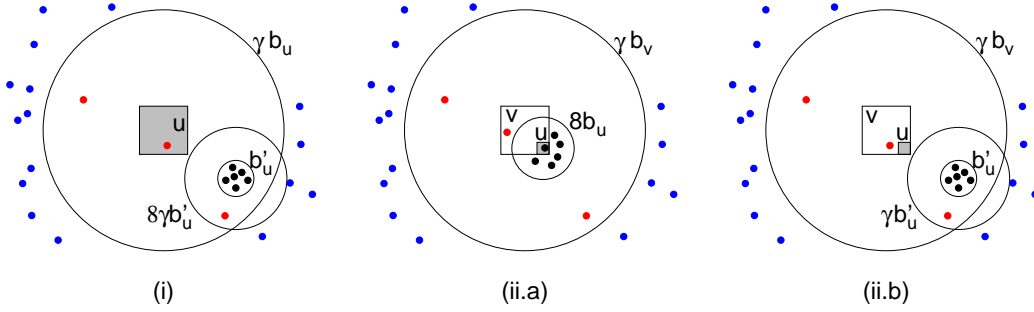


Fig. 1: The separation properties for cells of the BD tree.

subdivision induced by its leaves is a refinement of the subdivision induced by the quadtree boxes in \mathcal{C} . This can be constructed in time $O(n \log n)$.

- (iii) The number of cells of the BD tree with pairwise disjoint interiors, each of size at least s , that intersect a ball of radius r is at most $O((1+r/s)^d)$.

The *balanced box-decomposition (BBD) tree* has many of the properties of a BD tree, but it has $O(\log n)$ depth. As in the BD tree, a cell of a BBD tree is a quadtree box or the difference of two quadtree boxes. Let S be a set of n points in \mathbb{R}^d , and let T denote its BBD tree. A quadtree box (not necessarily in T) is *nonempty* if it contains at least one point of S . We will use the BBD tree for the following construction, called an *annulus cover*. Given two concentric balls b_1 and b_2 , where b_2 is contained within b_1 , and given $s > 0$ find the set \mathcal{Q} of nonempty quadtree boxes of side length s that overlap the annulus $b_1 - b_2$.

The properties of the BBD tree that are relevant to this paper are given below. Property (i) is proved in [5], and property (ii) follows from the analysis in [3]. (The straightforward proof is omitted.)

- (i) T has $O(n)$ nodes and $O(\log n)$ depth and can be constructed in time $O(n \log n)$.
- (ii) It is possible to compute an annulus cover as described above in time $O(\log n + t)$, where t is the number of nonempty quadtree boxes of size s that intersect the larger annulus $2b_1 - b_2/2$. In the same time we can compute $|S \cap z|$ for each box z in the cover. Since quadtree boxes exist only in discrete sizes (powers of 2) it is understood that if s is not of this form this construction is performed for the next smaller power of 2.

For both BD trees and BBD trees, we define the *size* of a cell to be the size of its outer box. Throughout, for a cell u , we will use s_u to denote its size and b_u to denote the ball of radius $s_u d/2$ whose center coincides with the center of u 's outer box (note that $u \subseteq b_u$). For $\gamma > 0$,

we will make frequent use of the ball γb_u , which we call the γ -*expansion* of u .

3 Approximate Range Counting

Let S be a set of n points in \mathbb{R}^d and let $0 < \epsilon \leq 1/2$ and $2 \leq \gamma \leq 1/\epsilon$ be two real parameters. In this section we show how to answer ϵ -approximate spherical range counting queries in time $O(\log(n\gamma) + 1/(\epsilon\gamma)^{d-1})$ using a data structure of space $O(n\gamma^d \log(1/\epsilon))$. We will show that the data structure can be constructed in time $O(n\gamma^d \log(n/\epsilon) \log(1/\epsilon))$.

Recall that S has been scaled and translated to lie within a ball of radius $\epsilon/2$ placed at the center of the unit hypercube U . It follows that queries whose center lies outside of U can be answered trivially. If B contains the center of U , then we can treat all the points of S as lying in the approximate range and return n as the answer; otherwise, we return 0. We turn to the interesting case when the center of B lies within U .

During preprocessing, we first construct a BD tree based on the following lemma, which states that it is possible to construct a BD tree whose nodes enjoy certain separation properties with respect to the points of S . (See Fig. 1.) Intuitively, part (i) says that, for a leaf cell, if there are many points close to it, they are sufficiently well clustered relative to their distance to the cell. As we will see, (i) is useful in answering approximate range queries when a range is sufficiently small. When a range is large, however, we need to use information associated with nodes higher up in the tree. Part (ii) describes the separation properties that aid in this case, and this innovation is needed for range searching (versus nearest neighbor searching). The parameter γ controls the separation and is used to control the space/time tradeoff. The proof of the lemma is omitted due to space limitations.

LEMMA 3.1. (Separation Properties) *Let S be a set of n points in \mathbb{R}^d , and let $\gamma \geq 16$ and $0 < f \leq 1$ be two real parameters. It is possible to construct a BD tree T with*

$O(nf\gamma^d \log \gamma)$ nodes satisfying the following properties. Let u be a cell corresponding to a node of T .

- (i) Suppose that u is a leaf cell. Then there exists a ball b'_u such that $|S \cap (\gamma b_u - b'_u)| = O(1/f)$ and the ball $8\gamma b'_u$ does not overlap u .
- (ii) Suppose that u is a box cell obtained by a shrink operation. Let v denote the parent cell of u . Then either (a) $|S \cap (\gamma b_v - 8b_u)| = O(1/f)$ or (b) there exists a ball b'_u such that $|S \cap (\gamma b_v - b'_u)| = O(1/f)$ and the ball $\gamma b'_u$ does not overlap u .

Moreover, in time $O(n\gamma^d \log(n\gamma) \log \gamma)$, we can construct T with the following information stored at the nodes. For each leaf cell u , we store the ball b'_u and $S \cap (\gamma b_u - b'_u)$. For each box cell u , if it satisfies (ii.a) we store $S \cap (\gamma b_v - 8b_u)$ and $|S \cap 8b_u|$, and if it satisfies (ii.b) we store the ball b'_u and $S \cap (\gamma b_v - b'_u)$.

Note that the cases in the lemma are neither mutually exclusive nor do they apply to all nodes (internal nodes obtained by splitting, in particular). When both conditions apply to a node, both pieces of auxiliary information are stored. In the lemma, the points of size $O(1/f)$ are called *pollutants*, since they do not satisfy the separation properties. They result as the leftovers of a sampling process and are needed to achieve our strongest bounds. Since they are handled by simple brute force in our query processing, the reader may find it easier to simply ignore them on first reading.

We set $f = (\epsilon\gamma)^{d-1}$, construct the BD tree T described in Lemma 3.1, and compute all the quantities mentioned in the last two sentences of the lemma. (If $\gamma < 16$, we set γ to 16 before using the lemma.) This takes time $O(n\gamma^d \log(n\gamma) \log \gamma)$. We then associate certain additional information with the nodes u of T , which will help in answering the queries. Before describing what information is stored with the nodes, it helps to give a brief overview of how queries are processed. Let \mathcal{L}_T denote the leaf cells of T . We distinguish between two kinds of query ranges. Let B be a query ball whose center lies inside a leaf cell $x \in \mathcal{L}_T$. We say that B is a *small* range if it is contained within γb_x ; otherwise we call it a *large* range. If B is a small range, we answer it using information stored at the leaf x . If B is a large range, we first check whether it contains U . In this case, we return n as the answer (recall $S \subseteq U$). Otherwise, let v be the first box cell on the path from leaf x to the root, such that $B \subseteq \gamma b_v$, and let u be v 's child on this path (note that u is a box cell). We answer the query using information associated with u .

Thus a leaf cell x is responsible for handling any query ball B that is centered in x such that $B \subseteq \gamma b_x$.

A box cell u is responsible for handling B centered in u that satisfies $B \not\subseteq \gamma b_u$ and $B \subseteq \gamma b_v$, where v denotes the parent cell of u . We now present details on the information stored with each type of cell and explain how this information helps to answer queries efficiently. To speed up the preprocessing, we assume that we have also constructed the BBD tree T_b for the set S of points.

Leaf cells. Each leaf cell is responsible for handling queries that lie entirely within its γ -expansion. We handle the points in the cluster ball b'_u , by subdividing them into a grid of small disjoint generator subsets, and we handle the relatively small number of pollutants by brute force. During the preprocessing phase, for each leaf cell u , we compute a set $\mathcal{Q}(u)$ of weighted quadtree boxes as follows. Let b'_u be the ball described in Lemma 3.1(i). If b'_u is the empty ball or does not overlap γb_u , then we set $\mathcal{Q}(u) = \emptyset$. Otherwise, we expand the ball b'_u such that $8\gamma b'_u$ just touches u . Henceforth, we will use b'_u to refer to this expanded ball and r_u to denote its radius. (See Fig. 2.) We then find the set $\mathcal{Q}(u)$ of nonempty quadtree boxes of diameter $2\epsilon(8\gamma - 1)r_u/3$ that overlap b'_u . By a straightforward packing argument, $|\mathcal{Q}(u)| = O(1/(\epsilon\gamma)^d)$. For each box $z \in \mathcal{Q}(u)$, we assign it a *weight* equal to $|S \cap z|$. By BBD property (ii), we can compute $\mathcal{Q}(u)$ and assign weights to the boxes in it in time $O(\log n + t_u)$, where t_u is the number of nonempty quadtree boxes of diameter $2\epsilon(8\gamma - 1)r_u/3$ that overlap the ball $2b'_u$.

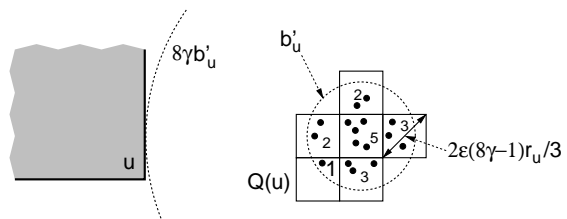


Fig. 2: Processing of leaf cells.

Next, we scan the list of $O(1/f) = O(1/(\epsilon\gamma)^{d-1})$ pollutants in $S \cap (\gamma b_u - b'_u)$ and eliminate those points that are contained in a box in $\mathcal{Q}(u)$; let $\mathcal{P}(u)$ denote the set of points that remain. Assuming the floor function, $\mathcal{P}(u)$ can be easily computed in time $O(1/(\epsilon\gamma)^{d-1})$.

By storing $\mathcal{P}(u)$ and $\mathcal{Q}(u)$ with each leaf u , we can answer queries as follows. Recall that u is responsible for answering the query for a ball B centered in u such that $B \subseteq \gamma b_u$. To answer this query, we do a linear scan of the boxes in $\mathcal{Q}(u)$ and compute the total weight of the boxes that overlap B . To this we add the number of points of $\mathcal{P}(u)$ that lie inside B and return it as the answer. The correctness of this approach is obvious from Lemma 3.1(i) and the fact that if B overlaps a box z in $\mathcal{Q}(u)$ then the diameter of z is

at most ϵ times the radius of B . The query time is $O(|\mathcal{Q}(u)| + |\mathcal{P}(u)|) = O(1/(\epsilon\gamma)^d)$.

By using a slightly more sophisticated method, we can reduce the query time by a factor of $O(1/(\epsilon\gamma))$. The idea is to organize the quadtree boxes in $\mathcal{Q}(u)$ into a BD tree during the preprocessing phase. By BD property (ii), this can be done in time $O(|\mathcal{Q}(u)| \log |\mathcal{Q}(u)|)$. With the help of this BD tree, using standard techniques [3], we can find the total weight of the boxes $z \in \mathcal{Q}(u)$ that overlap the query range B in time proportional to the number of boxes of $\mathcal{Q}(u)$ that overlap ∂B . A straightforward packing argument [3] shows that this quantity is $O(1/(\epsilon\gamma)^{d-1})$. Since the time to scan $\mathcal{P}(u)$ is also $O(1/(\epsilon\gamma)^{d-1})$, this quantity bounds the overall query time.

We now estimate the space requirements for all the leaves. By the bounds on $|\mathcal{P}(u)|$ and $|\mathcal{Q}(u)|$ given above, the space used for a leaf u is $O(1/(\epsilon\gamma)^d)$. Recall that the number of leaves is $O(nf\gamma^d \log \gamma)$, where $f = (\epsilon\gamma)^{d-1}$. Thus, the total space used by all the leaves together is $O((n\gamma^{d-1} \log \gamma)/\epsilon)$. However, this simple bound is based on the assumption that, for every leaf cell, $|\mathcal{Q}(u)|$ achieves its worst case space bound. Lemma 3.2 shows that this cannot happen and improves this bound by a factor of nearly $O(1/(\epsilon\gamma))$ using a charging argument (similar to that used in [2]). Our proof of this lemma makes use of the concept of a well-separated pair decomposition of a point set [7], so we first briefly review this notion.

We say that two sets of points X and Y are *well-separated* if they can be enclosed within two disjoint d -dimensional balls of radius r , such that the distance between the centers of these balls is at least αr , where $\alpha \geq 2$ is a real parameter called the *separation factor*. If we consider joining the centers of these two balls by a line segment, the resulting shape resembles a *dumbbell*. The balls are the *heads* of the dumbbell. A *well-separated pair decomposition* of S is a set $\mathcal{P}_{S,\alpha} = \{(X_1, Y_1), \dots, (X_m, Y_m)\}$ of pairs of subsets of S such that (i) for $1 \leq i \leq m$, X_i and Y_i are well-separated and (ii) for any distinct points $x, y \in S$, there exists a unique pair (X_i, Y_i) such that either $x \in X_i$ and $y \in Y_i$ or $x \in Y_i$ and $y \in X_i$. (We say that the pair (X_i, Y_i) *separates* x and y .) Callahan and Kosaraju [7] have shown that, for any set S of n points, there exists a well-separated pair decomposition containing $O(\alpha^d n)$ pairs.

LEMMA 3.2. *Let T be the BD tree described in Lemma 3.1 for any value of f , $0 < f \leq 1$. Let \mathcal{L}_T denote the leaf cells of T . For any leaf cell $u \in \mathcal{L}_T$, let $\mathcal{Q}(u)$ be as defined above. Then $\sum_{u \in \mathcal{L}_T} |\mathcal{Q}(u)| = O(n\gamma^d \log(1/\epsilon))$.*

Proof:(sketch) Let \mathcal{D} be the set of dumbbells corre-

sponding to the well-separated pair decomposition of S , using separation factor 8. Each dumbbell $P \in \mathcal{D}$ allocates a unit *charge* to the leaf cells in T satisfying certain conditions. Let a and b denote the centers of two heads of P , $\ell = \|ab\|$, and o denote the center of the segment \overline{ab} . Let \mathcal{B}_P denote the set of balls, centered at o , having radius $2^i \ell$, where $\log(c_1 \gamma) \leq i \leq \log(c_2 d/\epsilon)$ for suitable constants c_1 and c_2 . For a ball $b \in \mathcal{B}_P$, let \mathcal{C}_b be the set of leaf cells in T overlapping b that have size at least $c_3 r_b / (\gamma d)$, where r_b denotes the radius of b and c_3 is a suitable constant. The dumbbell P allocates a unit charge to each leaf cell in $\mathcal{C}_P = \cup_{b \in \mathcal{B}_P} \mathcal{C}_b$. By BD property (iii), $|\mathcal{C}_b| = O(\gamma^d)$. Since the number of balls in \mathcal{B}_P is $O(\log(1/(\epsilon\gamma)))$, it follows that P allocates a unit charge to $O(\gamma^d \log(1/(\epsilon\gamma)))$ leaf cells. Thus, the total charge allocated by all the dumbbells together is $O(n\gamma^d \log(1/(\epsilon\gamma)))$.

Next we show that each leaf cell u receives a charge from at least $\Omega(|\mathcal{Q}(u)| - 1)$ dumbbells. Recall that r_u is the radius of the ball b'_u such that the ball $8\gamma b'_u$ just touches u . We now show that there exists a subset $\mathcal{Q}'(u) \subseteq \mathcal{Q}(u)$ such that $|\mathcal{Q}'(u)| = \Omega(|\mathcal{Q}(u)|)$ and the distance between any pair of boxes in $\mathcal{Q}'(u)$ is at least $\Omega(\epsilon\gamma r_u/d)$. We can find $\mathcal{Q}'(u)$ as follows. Initially, we set $\mathcal{Q}'(u) = \emptyset$. We then consider the boxes in $\mathcal{Q}(u)$ one by one. Two boxes in $\mathcal{Q}(u)$ are said to be *neighbors* if they share a $(d-1)$ -facet. At each step, we add a box in $\mathcal{Q}(u)$ to $\mathcal{Q}'(u)$ and then eliminate it and all its neighbors in $\mathcal{Q}(u)$ from further consideration. We continue in this manner until all the boxes in $\mathcal{Q}(u)$ have been pruned. Clearly this process finds a set $\mathcal{Q}'(u)$ with the desired properties.

Let $\mathcal{X}(u)$ be the set of points obtained by picking one point of S from each box in $\mathcal{Q}'(u)$. The distance between any two points in $\mathcal{X}(u)$ is $\Omega(\epsilon\gamma r_u/d)$ and at most $O(r_u)$. It is easy to show that there exist $|\mathcal{X}(u)| - 1$ distinct dumbbells in \mathcal{D} that separate pairs of points in $\mathcal{X}(u)$ (we omit the details). Noting that $s_u = \Omega(r_u/d)$, one can now verify that each of these dumbbells allocates a charge to cell u , for a suitable choice of the constants c_1, c_2 and c_3 . Thus we have shown that each leaf cell u receives a charge of $\Omega(|\mathcal{Q}(u)| - 1)$.

Since the total charge allocated by all the dumbbells together is $O(n\gamma^d \log(1/(\epsilon\gamma)))$ and the number of leaf cells is $O(nf\gamma^d \log \gamma)$, it follows that $\sum_{u \in \mathcal{L}_T} |\mathcal{Q}(u)| = O(n\gamma^d \log(1/(\epsilon\gamma)) + nf\gamma^d \log \gamma) = O(n\gamma^d \log(1/\epsilon))$. This completes the proof. \square

Using this lemma and the bound on $|\mathcal{P}(u)|$ given above, it follows that the space used by all the leaves is $O(n\gamma^d \log(1/\epsilon))$. We conjecture that the $\log(1/\epsilon)$ factor is an artifact and can be eliminated.

We now estimate the preprocessing time for all the leaves. Recall that for each leaf u , $\mathcal{P}(u)$ can

be computed in time $O(1/(\epsilon\gamma)^{d-1})$ and $\mathcal{Q}(u)$ can be computed in time $O(t_u + \log n)$. Here t_u is the number of nonempty quadtree boxes of diameter $2\epsilon(8\gamma - 1)r_u/3$ that overlap the ball $2b'_u$. Applying a similar charging argument as used above to bound $\sum_{u \in \mathcal{L}_T} |\mathcal{Q}(u)|$, it follows that $\sum_{u \in \mathcal{L}_T} t_u = O(n\gamma^d \log(1/\epsilon))$. Thus, in time $O((f\gamma^d \log \gamma)n \log n + (\gamma^d \log(1/\epsilon))n)$, we can compute $\mathcal{P}(u)$ and $\mathcal{Q}(u)$ for all leaves u . Also, recall that it takes $O(|\mathcal{Q}(u)| \log |\mathcal{Q}(u)|)$ time to organize the boxes in $\mathcal{Q}(u)$ into a BD tree. Since $|\mathcal{Q}(u)| = O((1/\epsilon\gamma)^d)$ and $\sum_{u \in \mathcal{L}_T} |\mathcal{Q}(u)| = O(n\gamma^d \log(1/\epsilon))$, it follows that

$$\sum_{u \in \mathcal{L}_T} |\mathcal{Q}(u)| \log |\mathcal{Q}(u)| = O(n\gamma^d \log^2(1/\epsilon)).$$

Thus, the total preprocessing time for all the leaves is $O((f\gamma^d \log \gamma)n \log n + (\gamma^d \log^2(1/\epsilon))n) = O(n\gamma^d \log(n/\epsilon) \log(1/\epsilon))$.

Box cells obtained by shrinking. If u satisfies (ii.b) of Lemma 3.1 then, in view of the similarity of this property with (i), it is clear that we can use the same approach as described above for leaf cells. So suppose that u satisfies property (ii.a). Recall that u is responsible for a query range B centered in u that satisfies $B \not\subseteq \gamma b_u$ and $B \subseteq \gamma b_v$, where v is u 's parent cell. Since $\gamma \geq 16$ and $B \not\subseteq \gamma b_u$, it is easy to show that $B \supseteq 8b_u$. Thus the exact answer to such a query can be computed as the sum of $|S \cap 8b_u|$ and the number of points of $S \cap (\gamma b_v - 8b_u)$ that lie within B . Both $|S \cap 8b_u|$ and $S \cap (\gamma b_v - 8b_u)$ are precomputed and stored with u , so the query can be answered in time $O(1/(\epsilon\gamma)^{d-1})$. The total space and preprocessing time for all the box cells obtained by shrinking is the same as for all the leaves.

Box cells obtained by splitting. By definition, u is one of the 2^d quadtree boxes of equal size into which u 's parent cell v is split. Recall that u is responsible for handling query balls B centered within u such that B is not contained within its γ -expansion but is contained within the γ -expansion of its parent, that is $B \not\subseteq \gamma b_u$ and $B \subseteq \gamma b_v$. Using the triangle inequality, an easy calculation shows that $\gamma' b_u \subseteq B \subseteq \gamma'' b_u$, where $\gamma' = \gamma - 2$ and $\gamma'' = 3\gamma$.

Such a cell does not generally enjoy any separation properties with respect to the point set S . Our approach is to compute a cover of the annulus $\gamma'' b_u - \gamma' b_u$ by a set of disjoint regions such that the subset of points lying within these regions can be used as generators for the query. We describe two approaches for constructing these regions. The first is a simple method based on a grid decomposition of the annulus. The second achieves better query performance and can be thought of as a radial version of a quadtree decomposition taking place in the space of polar coordinates. We provide

both because the space analysis of the second method depends on the space analysis of the first.

During the preprocessing phase, we compute an annulus cover $\mathcal{Q}(u)$ by boxes of size $\epsilon\gamma' s_u/4$ for the annulus $\gamma'' b_u - \gamma' b_u$. For each box $z \in \mathcal{Q}(u)$, we assign it a *weight* equal to $|S \cap z|$. (See Fig. 3(a).) By BBD property (ii) this can be done in time $O(\log n + t_u)$, where t_u is the number of nonempty quadtree boxes of size $\epsilon\gamma' s_u/4$ overlapping the larger annulus $2\gamma'' b_u - (\gamma'/2)b_u$. In the same time, using standard techniques [3], we compute the number of points in $S \cap \gamma' b_u$ that are not contained in any box of $\mathcal{Q}(u)$, and store this information with u .

By a straightforward packing argument, $|\mathcal{Q}(u)| = O(1/\epsilon^d)$. Let \mathcal{A}_T denote the set of box cells obtained by a split operation. Since T has $O(nf\gamma^d \log \gamma)$ nodes, where $f = (\epsilon\gamma)^{d-1}$, it follows that $\sum_{u \in \mathcal{A}_T} |\mathcal{Q}(u)| = O((n\gamma^{2d-1} \log \gamma)/\epsilon)$. This bound can be significantly improved by using a charging argument similar to that used earlier for leaf cells. Applying this argument yields $\sum_{u \in \mathcal{A}_T} |\mathcal{Q}(u)| = O(n\gamma^d \log(1/\epsilon))$. (We omit the details for lack of space.) Similarly, we obtain $\sum_{u \in \mathcal{A}_T} t_u = O(n\gamma^d \log(1/\epsilon))$. Thus, the time to compute $\mathcal{Q}(u)$ for all the cells $u \in \mathcal{A}_T$ is $O((f\gamma^d \log \gamma)n \log n + (\gamma^d \log(1/\epsilon))n)$.

It is possible to answer a query by computing the total weight of the boxes in $\mathcal{Q}(u)$ that overlap the range B . However, since $|\mathcal{Q}(u)|$ can be as large as $\Omega(1/\epsilon^d)$, this would lead to a high query time. We now discuss how this can be reduced to $O(1/(\epsilon\gamma)^{d-1})$ by using a more efficient decomposition of the annulus based on a radial decomposition of space.

First we need to introduce some notation. Let o denote the center of the quadtree box u . Let H denote the axis-parallel hypercube of size $\gamma' s_u d$ centered at o . Consider a regular grid of side length $\epsilon(\gamma')^2 s_u / (16\pi)$ on each of the $2d$ faces of dimension $(d-1)$ of ∂H . Let \mathcal{C} be the set of cones that have their apex at o and whose base is a cell in the grid placed on the faces of ∂H . Extend each cone through this base to infinity. It is easy to see that the angular diameter of each cone in \mathcal{C} is at most $\epsilon\gamma'/16$ and $|\mathcal{C}| = O(1/(\epsilon\gamma)^{d-1})$. Let \mathcal{B} be a set of $O(1/\epsilon)$ spheres between $\gamma' b_u$ and $\gamma'' b_u$, such that the radii of any two successive spheres differs by at most $\epsilon\gamma' s_u d / 16$. The boundaries of both $\gamma' b_u$ and $\gamma'' b_u$ are included in \mathcal{B} . (See Fig. 3(b).) We refer to a cell in the arrangement of $\mathcal{C} \cup \mathcal{B}$ as a *fragment*. Note that $\gamma'' b_u - \gamma' b_u$ is partitioned into $O(1/(\epsilon\gamma)^{d-1}(1/\epsilon))$ fragments; let $\mathcal{F}(u)$ denote this set of fragments.

During the preprocessing phase, for each quadtree box $z \in \mathcal{Q}(u)$, we determine a fragment in $\mathcal{F}(u)$ that it overlaps (assuming the floor function, this takes constant time), and add the weight of z to the weight of

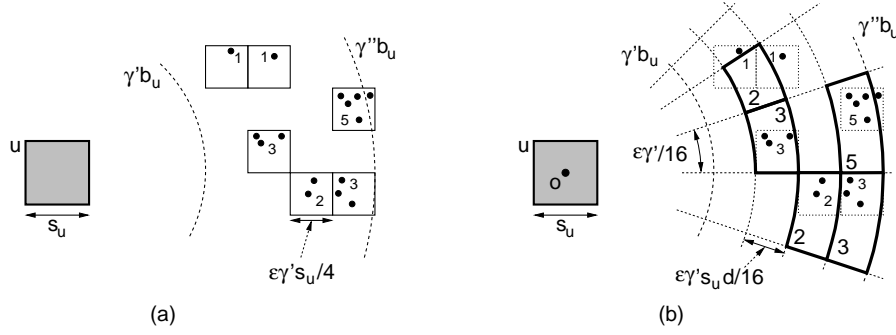


Fig. 3: Processing of cells obtained by splitting: (a) the annulus cover $\mathcal{Q}(u)$ and (b) the nonempty radial fragments $\mathcal{F}'(u)$.

the fragment. Initially we assume that the weight of all the fragments is zero. (The reason for assigning weights from quadtree boxes of the annulus cover, rather than from the point set S directly, is that quadtree boxes share a common coordinate system, which allows us to construct and analyze them globally for all nodes. In contrast, the radial fragments are based on coordinate systems that are local to each node.) At the end of this process, the weight of a fragment is the sum of the weights of all the quadtree boxes in $\mathcal{Q}(u)$ that transfer their weight to it. Let $\mathcal{F}'(u)$ denote the set of fragments that finally have a non-zero weight. (See Fig. 3(b).) Since $\mathcal{F}'(u)$ typically has fewer fragments than $\mathcal{F}(u)$, we use hashing [10, 17] to compute only the fragments of $\mathcal{F}'(u)$ along with their associated weights. This can be done in time $O(|\mathcal{Q}(u)|)$. The total space used to store $\mathcal{F}'(u)$ over all box cells obtained by splitting is $\sum_{u \in \mathcal{A}_T} |\mathcal{F}'(u)| \leq \sum_{u \in \mathcal{A}_T} |\mathcal{Q}(u)| = O(n\gamma^d \log(1/\epsilon))$.

We answer a query by scanning the fragments of $\mathcal{F}'(u)$ and determining the total weight of the fragments that overlap the range B . To this we add the precomputed number of the points of $S \cap \gamma'b_u$ that are not included in any box of $\mathcal{Q}(u)$. The correctness of this method follows from Lemma 3.4. Intuitively, the lemma shows that the algorithm makes errors only with respect to points of S that lie close to the boundary of B . An important ingredient in its proof is Lemma 3.3, which implies that all points in a fragment are at nearly the same distance from any point in cell u . Define the *absolute ratio* between two positive numbers x and y to be the maximum of x/y and y/x . (The proofs of both these lemmas involve a straightforward geometric analysis, and are omitted due to space limitations.)

LEMMA 3.3. *Let $0 < \epsilon \leq 1/2$ and $\hat{\gamma} \geq 2$. Let u be a hypercube of side length s centered at o . Let x_1 and x_2 be two points such that $\|ox_1\| \geq \|ox_2\| \geq \hat{\gamma}sd$, $\|ox_1\| - \|ox_2\| \leq \epsilon\hat{\gamma}sd/8$, and $\angle x_1ox_2 \leq \epsilon\hat{\gamma}/8$. Then for any point o' in u , the absolute ratio of $\|o'x_1\|$ and $\|o'x_2\|$ is at most $1 + \epsilon/3$.*

LEMMA 3.4. *Let u be a box cell obtained by splitting a cell v . Let B be a query ball centered at a point in u such that $B \not\subseteq \gamma'b_u$ and $B \subseteq \gamma'b_v$. Let r denote the radius of B . Let p be a point in S whose distance from ∂B is at least ϵr , and let $z \in \mathcal{Q}(u)$ be the quadtree box that contains p .*

- (i) *If $p \in B$, then any fragment in $\mathcal{F}'(u)$ that overlaps z must also overlap B .*
- (ii) *If $p \notin B$, then no fragment in $\mathcal{F}'(u)$ overlaps both z and B .*

The query time of this method is $O((1/\epsilon\gamma)^{d-1}(1/\epsilon))$, since it is proportional to the number of fragments in $\mathcal{F}'(u)$. While this is already a significant improvement over the query time obtained by searching all the boxes in $\mathcal{Q}(u)$, we can speed it up still further through the use of BD trees. The idea is to organize the fragments of $\mathcal{F}'(u)$ into $2d$ BD trees, one for each of the faces of H . Since the fragments of $\mathcal{F}(u)$ are pieces of a d -dimensional annuli, they are certainly not quadtree boxes. Nonetheless, by applying an appropriate transformation to polar coordinates, it is possible to map the fragments to a d -dimensional grid of hypercubes, from which we can then apply any standard algorithm for constructing a BD tree from a set of quadtree boxes [3]. To see how a point in this “fragment space” can be described as d polar coordinates, the first $d - 1$ polar coordinates arise by shooting a ray from o through this point until it hits a $(d - 1)$ -dimensional face of the hypercube H , and then representing the resulting point using a local coordinate system for this face. Based on these coordinates alone, the fragments form a $(d - 1)$ -dimensional grid of hypercubes of side length $h = \epsilon(\gamma')^2 s_u / (16\pi)$. The last coordinate of the polar representation arises from the distance of the point from the center of H . Along this axis, each fragment has a length of $r = \epsilon\gamma' s_u d / 16$. By scaling this last coordinate by the amount h/r , in polar space the fragments are now mapped to a grid of

d -dimensional hypercubes, as desired.

Now, by applying standard techniques to the resulting BD tree [3], we can determine the total weight of the fragments of $\mathcal{F}'(u)$ that overlap B in time proportional to the number of fragments that intersect ∂B (transformed into polar space). Applying Lemma 3.3, it is not hard to show that ∂B intersects $O(1)$ fragments in each cone of \mathcal{C} , which implies that the query time is $O(1/(\epsilon\gamma)^{d-1})$.

It takes $O(|\mathcal{F}'(u)| \log |\mathcal{F}'(u)|)$ time to construct the BD trees for the fragments in $\mathcal{F}'(u)$. Noting that $\sum_{u \in \mathcal{A}_T} |\mathcal{F}'(u)| = O(n\gamma^d \log(1/\epsilon))$ and $|\mathcal{F}'(u)| = O((1/\epsilon\gamma)^{d-1}(1/\epsilon))$, it follows that

$$\sum_{u \in \mathcal{A}_T} |\mathcal{F}'(u)| \log |\mathcal{F}'(u)| = O(n\gamma^d \log^2(1/\epsilon)).$$

Thus, the preprocessing time for all the box cells in \mathcal{A}_T is $O((f\gamma^d \log \gamma)n \log n + (\gamma^d \log^2(1/\epsilon))n) = O(n\gamma^d \log(n/\epsilon) \log(1/\epsilon))$.

Balancing the tree. The only problem with the approach as described above is that the BD tree may not be balanced. Thus the time to find the node that stores the information needed for answering a query can be very large. To remedy this we can transform the BD tree T into a BBD tree T' . This can be performed in time $O(m \log m)$, where m denotes the number of nodes in T . This does not increase the space asymptotically, but has the desirable effect of reducing the time for tree descent to logarithmic in the total number of nodes, that is, $O(\log(n\gamma))$. (For lack of space we omit further details of these standard ideas.) Thus, the total query time is $O(\log(n\gamma) + 1/(\epsilon\gamma)^{d-1})$. Putting it all together, we have shown the following theorem.

THEOREM 3.1. *Let S be a set of n points in \mathbb{R}^d , and let $0 < \epsilon \leq 1/2$ and $2 \leq \gamma \leq 1/\epsilon$ be two real parameters. Then we can construct a data structure of $O(n\gamma^d \log(1/\epsilon))$ space that allows us to answer ϵ -approximate range queries in time $O(\log(n\gamma) + 1/(\epsilon\gamma)^{d-1})$. The time to construct the data structure is $O(n\gamma^d \log(n/\epsilon) \log(1/\epsilon))$.*

Remark: There is an interesting parallel here with exact range queries. In approximate nearest neighbor searching (analogous to halfspace emptiness) ignoring the log terms, the exponent in the query time as a function of d grows roughly as $(d-1)/2$ [2], whereas we have just shown that for general counting queries it grows roughly as $d-1$. In the exact case, there is a corresponding gap between the exponent on n of $d/2$ for halfspace emptiness [16] and d for halfspace range queries [14]. This suggests that our complexity bounds are close to what would be expected. Nonetheless, our

recent research shows that for idempotent semigroups (for example, min and max queries), where it suffices to cover the range by generators rather than partition it, it is possible to achieve significantly better dependencies on dimension and similar lower bounds can be derived. Interestingly, there is no such distinction in the complexity of idempotent and non-idempotent semigroup range queries in the exact case. These results will be reported in a future paper.

4 Approximate k th Nearest Neighbor Queries

In this section we describe how we can enhance the data structure for approximate range counting to answer approximate k th nearest neighbor queries. The role played by the query point q is analogous to that played earlier by the center of the query range, but now the radius of the range is unknown. Here is a simple approach, which we will improve upon later. Recall that a box cell u in the BD tree that contains q is responsible for handling ranges centered at q such that $B \not\subseteq \gamma b_u$ and $B \subseteq \gamma b_v$, where v is the parent of u . To answer a query, for each node along the search path of the query point we can apply the results of the last section to count the number of points in the approximations of the smallest and largest of these ranges, and then use this information to determine the appropriate node. From the results of the last section this computation takes time $O(1/(\epsilon\gamma)^{d-1})$ for each node visited. There can be $\Omega(n)$ nodes on a path in the BD tree, but, as mentioned above, we can perform this search efficiently in a BBD tree, whose depth is $O(\log(n\gamma))$. Overall it takes $O(\log(n\gamma) \cdot (1/(\epsilon\gamma)^{d-1}))$ time to find a node that can handle a range corresponding to an approximate k th nearest neighbor. After finding such a node, we can perform a final search for the desired radius approximately. The exact nature of the search depends on the type of node, but it is easy to show that $O(\log(1/\epsilon) \cdot (1/(\epsilon\gamma)^{d-1}))$ time suffices. Thus, the total query time for this approach is $O(\log(n/\epsilon) \cdot (1/(\epsilon\gamma)^{d-1}))$. The space and preprocessing time for this data structure are the same as given in Theorem 3.1.

Let us consider how to improve this simple approach. Determining the numbers of points in the largest and smallest ranges at each node depends on the location of q , and so cannot be precomputed. However, for the purposes of driving the search it suffices to compute these quantities for any point that is sufficiently close to q . We will partition each node u into small hypercubes, and precompute counts for a point in each hypercube.

Let u be a box cell in the BD tree T . During the preprocessing phase, we partition u into a regular grid of $O(1/(\epsilon\gamma)^d)$ hypercubes of size $\epsilon\gamma s_u/c$, where c is a

suitable constant. For each hypercube z , we precompute the answer to an $\epsilon/4$ approximate range query for the smallest and largest balls handled by u that are centered at a point inside z . Since the radius of these balls exceeds $(\gamma - 1)s_u d/2$, one can easily show that the answers stored with z are valid for any point $q \in z$ (only the approximation error for the range query increases slightly from $\epsilon/4$ to $\epsilon/2$). At query time, we determine the hypercube z containing q (which takes $O(1)$ time assuming the floor function and $O(\log(1/(\epsilon\gamma)))$ time otherwise) and use the counts stored with z to determine if u is responsible for a range corresponding to the approximate k th nearest neighbor.

The above modification implies that each box cell uses space $O(1/(\epsilon\gamma)^d)$, which increases the space requirements by a factor of nearly $1/(\epsilon\gamma)$. To remedy this, we reduce the value of parameter f used in the construction of the BD tree T described in Lemma 3.1 from $(\epsilon\gamma)^{d-1}$ to $(\epsilon\gamma)^d$. An inspection of the analysis in Section 3 shows that with this change the total space used by the data structure remains $O(n\gamma^d \log(1/\epsilon))$. (We defer details to the full version.)

We now discuss the time it takes to process the last node u visited when answering a query. If u is a leaf cell, then $|\mathcal{P}(u)| = O(1/f) = O(1/(\epsilon\gamma)^d)$. Recall that $|\mathcal{Q}(u)| = O(1/(\epsilon\gamma)^d)$ and each box $z \in \mathcal{Q}(u)$ is associated with a weight $|S \cap z|$. We associate a weight of one with each point in $\mathcal{P}(u)$. By standard results on weighted selection, we can determine the approximate distance to the k th nearest neighbor in time $O(|\mathcal{P}(u)| + |\mathcal{Q}(u)|) = O(1/(\epsilon\gamma)^d)$. As was the case for range count, a similar approach works for box cells obtained by shrinking, because they satisfy essentially the same separation properties as leaves.

So suppose next that u is a box cell obtained by splitting. Recall that if u is responsible for handling a query range B , then $(\gamma - 2)b_u \subseteq B \subseteq 3\gamma b_u$. We use binary search on the radius of the ball to determine the approximate distance to the k th nearest neighbor. Since we need to test $O(\log(1/\epsilon))$ different radii and it takes $O(1/(\epsilon\gamma)^{d-1})$ time to determine the count for a given radius, the time for the binary search is $O((1/(\epsilon\gamma)^{d-1}) \log(1/\epsilon))$, which is dominated by the leaf query time. Summarizing the discussion of this section we have:

THEOREM 4.1. *Let S be a set of n points in \mathbb{R}^d , and let $0 < \epsilon \leq 1/2$ and $2 \leq \gamma \leq 1/\epsilon$ be two real parameters. We can construct a data structure of $O(n\gamma^d \log(1/\epsilon))$ space that allows us to answer ϵ -approximate k th nearest neighbor queries in time $O(\log(n\gamma) + 1/(\epsilon\gamma)^d)$. Here k is an integer between 1 and n that is specified at query time. The data structure can be constructed in time $O(n(\gamma/\epsilon)^{d/2} \log(n/\epsilon) \log(1/\epsilon))$.*

References

- [1] S. Arya and T. Malamatos. Linear-size approximate Voronoi diagrams. In *Proc. 13th ACM-SIAM Sympos. Discrete Algorithms*, pages 147–155, 2002.
- [2] S. Arya, T. Malamatos, and D. M. Mount. Space-efficient approximate Voronoi diagrams. In *Proc. 34th Annual ACM Sympos. Theory Comput.*, pages 721–730, 2002.
- [3] S. Arya and D. M. Mount. Approximate range searching. *Computational Geometry: Theory and Applications*, 17:135–152, 2000.
- [4] S. Arya, D. M. Mount, N. Netanyahu, R. Silverman, and A. Y. Wu. An optimal algorithm for approximate nearest neighbor searching in fixed dimensions. In *Proc. 5th ACM-SIAM Sympos. Discrete Algorithms*, pages 573–582, 1994.
- [5] S. Arya, D. M. Mount, N. Netanyahu, R. Silverman, and A. Y. Wu. An optimal algorithm for approximate nearest neighbor searching in fixed dimensions. *J. ACM*, 45:891–923, 1998.
- [6] H. Brönnimann, B. Chazelle, and J. Pach. How hard is halfspace range searching. *Discrete Comput. Geom.*, 10:143–155, 1993.
- [7] P. B. Callahan and S. R. Kosaraju. A decomposition of multidimensional point sets with applications to k -nearest-neighbors and n -body potential fields. *J. ACM*, 42:67–90, 1995.
- [8] B. Chazelle. Lower bounds on the complexity of polytope range searching. *J. Amer. Math. Soc.*, 2:637–666, 1989.
- [9] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, Berlin, Germany, 2nd edition, 2000.
- [10] M. L. Fredman, J. Komlos, and E. Szemerédi. Storing a sparse table with $O(1)$ worst case access time. *J. ACM*, 31(3):538–544, 1984.
- [11] S. Funke and E. A. Ramos. Smooth-surface reconstruction in near-linear time. In *Proc. 12th ACM-SIAM Symp. Discrete Algorithms*, pages 781–790, 2002.
- [12] S. Har-Peled. A replacement for Voronoi diagrams of near linear size. In *Proc. 42 Annu. IEEE Sympos. Found. Comput. Sci.*, pages 94–103, 2001.
- [13] J. Majhi, R. Janardan, M. Smid, and P. Gupta. On some geometric optimization problems in layered manufacturing. *Comput. Geom. Theory Appl.*, 12:219–239, 1999.
- [14] J. Matoušek. Range searching with efficient hierarchical cuttings. *Discrete Comput. Geom.*, 10(2):157–182, 1993.
- [15] J. Matoušek. On approximate geometric k -clustering. *Discrete and Comput. Geometry*, 24:61–84, 2000.
- [16] J. Matoušek and O. Schwarzkopf. On ray shooting in convex polytopes. *Discrete Comput. Geom.*, 10(2):215–232, 1993.
- [17] R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge University Press, New York, NY, 1995.