Algorithmica (2018) 80:279–299
https://doi.org/10.1007/s00453-016-0254-4

Robust Proximity Search for Balls Using Sublinear Space

Sariel Har-Peled[1] · **Nirman Kumar**[2]

Received: 8 September 2015 / Accepted: 14 November 2016 / Published online: 28 November 2016

Abstract Given a set of n disjoint balls $b_1, \ldots, b_n$ in $\mathbb{R}^d$, we provide a data structure of near linear size that can answer $(1 \pm \varepsilon)$-approximate kth-nearest neighbor queries on the balls in $O(\log n + 1/\varepsilon^d)$ time, where k and ε may be provided at query time. If k and ε are provided in advance, we provide a data structure to answer such queries requiring $O(n/k)$ space; that is, the data structure requires sublinear space if k is sufficiently large.

Keywords Data structures · Approximation algorithms · Proximity search

1 Introduction

The *nearest neighbor* problem is a fundamental problem in computer science [1,18]. Here, one is given a set of points P, and a query point q, and one needs to output the nearest point in P to q. There is a trivial $O(n)$ algorithm for this problem. Typ-

A preliminary version of this paper appeared in FSTTCS 2014 [15].

Work on this paper was partially support by NSF AF awards CCF-0915984, CCF-1421231, and CCF-1217462.

✉ Nirman Kumar
nkumar8@memphis.edu

Sariel Har-Peled
sariel@illinois.edu

[1] Department of Computer Science, University of Illinois, 201 N. Goodwin Avenue, Urbana, IL 61801, USA

[2] Department of Computer Science, University of Memphis, Dunn Hall 375, Memphis, TN 38152, USA

ically the set of data points is fixed, while different queries keep arriving. Thus, one can use preprocessing to facilitate a faster query. There are several applications of nearest neighbor search in computer science including pattern recognition, information retrieval, vector compression, computational statistics, clustering, data mining, learning and many others. See the survey by Clarkson [9].

There is no known way to solve this problem with logarithmic query time for dimension $d > 2$, while using near linear space for the data structure.

Approximate Nearest Neighbor (ANN) In light of the above, major effort has been devoted to developing approximation algorithms for nearest neighbor search [6,9,13,17]. In the $(1+\varepsilon)$-*approximate nearest neighbor* problem, one is additionally given an approximation parameter $\varepsilon > 0$ and one is required to find a point $\mathsf{u} \in \mathsf{P}$ such that $\mathsf{d}(\mathsf{q}, \mathsf{u}) \leq (1+\varepsilon)\mathsf{d}(\mathsf{q}, \mathsf{P})$. In d dimensional Euclidean space, one can answer ANN queries in $O(\log n + 1/\varepsilon^{d-1})$ time using linear space [6,12]. Unfortunately, the constant hidden in the O notation is exponential in the dimension (and this is true for all bounds mentioned in this paper), and specifically because of the $1/\varepsilon^{d-1}$ term in the query time. As such, this approach is only efficient in low dimensions. Interestingly, for this data structure, the approximation parameter ε need not be specified during the construction and one can provide it during the query. An alternative approach is to use Approximate Voronoi Diagrams (AVD), introduced by Har-Peled [11], which is a partition of space into regions of low total complexity, with a representative point for each region that is an ANN for any point in the region. In particular, Har-Peled showed that there is such a decomposition of size $O\big((n/\varepsilon^d)\log^2 n\big)$, see also [13]. This allows ANN queries to be answered in $O(\log n)$ time. Arya and Malamatos [2] showed how to build AVDs of linear complexity – $O(n/\varepsilon^d)$. Their construction uses WSPD (Well-Separated Pairs Decomposition) [7]. Further trade-offs between query time and space usage for AVDs were studied by Arya *et al.* [4].

k-Nearest Neighbors A more general problem is the k-nearest neighbors problem where one is interested in finding the k points in P nearest to the query point q. This is widely used in classification, where the majority label is used to label the query point. A restricted version is to find only the kth-nearest neighbor. This problem and its approximate version have been considered in [3,14]. For this problem Arya *et al.* show in [3] that one can achieve a tradeoff in the space vs query time of the data structure. If ε is known during preprocessing but k is known only during query they can achieve a query time of $O(\log(n/\varepsilon))$ with space requirement of $O(\frac{n}{\varepsilon^d}\log\varepsilon^{-1})$, or a query time of $O(\log n + 1/\varepsilon^d)$ with a space usage of $O(n\log\varepsilon^{-1})$. The latter result is improved in [14] where a data structure is shown that has a query time of $O(\log n + 1/\varepsilon^{d-1})$ and a space usage of $O(n)$ even when both ε and k are only supplied during query time. Moreover, they also show that if ε, k are supplied during preprocessing then the query time can be improved to $O(\log(\frac{n}{k\varepsilon}))$ with a space usage of $O(C_\varepsilon n/k)$ where $C_\varepsilon = \varepsilon^{-d}\log\varepsilon^{-1}$.

Sublinear Space, Summarizing Data and (k,ε)-ANN Recently, the authors [14] showed that one can compute a (k,ε)-AVD that $(1+\varepsilon)$-approximates the distance to the kth nearest neighbor, and surprisingly, requires $O(n/k)$ space; that is, sublinear

space if k is sufficiently large. For example, for the case $k = \Omega(\sqrt{n})$, which is of interest in practice, the space required is only $O(\sqrt{n})$. Such ANN is of interest when one is worried that there is noise in the data, and thus one is interested in the distance to the kth NN which is more robust and noise resistant than the nearest neighbor. Alternatively, one can think about such data structures as enabling one to summarize the data in a way that still facilitates meaningful proximity queries.

This Paper Here, we consider a generalization of the kth-nearest neighbor problem. Specifically, given a set of n disjoint balls in $\mathbb{R}^d$ the task is to preprocess them. Now, given a query point one can find approximately the kth closest ball. The distance of a query point to a ball is defined as the distance to its boundary if the point is outside the ball or 0 otherwise. Clearly, this problem is a generalization of the kth-nearest neighbor problem by viewing points as balls of radius 0. Algorithms for the kth-nearest neighbor for points, do not extend in a straightforward manner to this problem because the distance function is no longer a metric. Indeed, there can be two far off points both close to a single ball, and thus the triangle inequality does not hold. The problem of finding the closest ball can also be modeled as a problem of approximating the minimization diagram of a set of functions. Here, a function would correspond to the distance from one of the given balls. There has been some recent work by the authors on this topic, see [16], where a fairly general class of functions admits a near linear sized data structure permitting a logarithmic time query for the problem of approximating the minimization diagram. However, the problem that we consider here does not fall under the aforementioned framework [16]. The technical assumptions of this framework [16] mandate that the set of points which form the 0-sublevel set of a distance function, i.e., the set of points at which the distance function is 0 is a single point (or an empty set). This is not the case for the problem under consideration. Also, we are interested in the more general kth-nearest neighbor problem, while the previous work [16] considers only the nearest-neighbor problem, i.e., $k = 1$.

Our Results

We first show how to preprocess the set of balls into a data structure requiring space $O(n)$, in $O(n \log n)$ time. For a query point q, a number $1 \leq k \leq n$ and $\varepsilon > 0$, one can compute a $(1 \pm \varepsilon)$-approximate kth closest ball in time $O(\log n + \varepsilon^{-d})$. If both k and ε are available during preprocessing, one can preprocess the balls into a (k, ε)-AVD, using $O(\frac{n}{k\varepsilon^d} \log(1/\varepsilon))$ space. For a query point q, a (k, ε)-ANN ball can be computed, in $O(\log(n/k) + \log(1/\varepsilon))$ time.

Plan of Attack, and Highlights The idea is to try and extend our previous work [14] to the new, more general setup. Since we are dealing with balls instead of points, the task is more challenging, and we do it in stages:

(A) *Approximate range counting on balls.* Given a set of disjoint balls, and a query ball, we want to count the number of input balls that intersect it, while allowing an approximation only for the query ball. This is somewhat more challenging than approximate range counting, as done by Arya and Mount [5], as some of the balls intersecting the query ball might be significantly larger. To this end, we

build a data structure that enables us to quickly count exactly the large input balls that intersect the query ball. This is described in Sect. 3.

(B) *Linear space* (k, ε)-**ANN on balls.** Given a query point, we compute its distance to the ith nearest center, for $i = k - \mathsf{c}_d, \ldots, k$, where c_d is some constant that depends on the dimension. Next, we argue that either one of these distances is the required approximate distance (and this can be verified using the approximate range counting data structure from above), or alternatively, the distance is determined by "huge" balls that have radius significantly larger than the desired distance. As such, we extract the at most c_d large balls that might be relevant, add their distance to the query point to the set of candidate distance, and search these distances. This yields a constant approximation to the kth ANN, and converting it to a $(1+\varepsilon)$-approximation is easy using our tools. This is described in Sect. 4.

(C) *Quorum clustering for balls.* Somewhat oversimplifying things, the basic strategy in the previous work [14], was to find the point achieving the global minimum of the kth ANN distance function, approximate the function correctly in a region around this point, remove the points that define the minimum from the data-set, and repeat. This was facilitated by finding the smallest ball that contains k input points. For balls, it is not clear how to find the smallest ball that intersects k balls, remove these balls, repeat this process, and moreover, do it efficiently. Furthermore, it is no longer true that one can remove these k balls, as some of them might be huge. Instead, conceptually, we remove only the small balls (the exact details of what we do are more involved, and require significantly more care) from these k balls. Furthermore, instead of using the smallest ball intersecting k balls, we use the smallest ball containing $k - \mathsf{c}_d$ centers of the balls, and expand it till it intersects k balls. We then repeat this mining process till all centers are excavated. Surprisingly, since the quorum clustering is done on the centers and not on the balls, we are able to implement this process efficiently, and furthermore, we can argue that it yields a meaningful quorum clustering for the input balls. This is described in Sect. 5.

(D) *Sublinear space* (k, ε)-**AVD on balls.** Now, equipped with the new quorum clustering of the balls, we can build a (k, ε)-AVD for the balls. Surprisingly, the construction now follows [14] in a straightforward fashion. The resulting data structure uses $O(n/k)$ space, and can answer (k, ε)-ANN queries in $O(\log n)$ time.

Paper Organization In Sect. 2 we define the problem, list some assumptions, and introduce notation. In Sect. 3 we set up some basic data structures to answer approximate range counting queries for balls. In Sect. 4 we present the data structure, query algorithm and proof of correctness for our data structure which can compute $(1 \pm \varepsilon)$-approximate kth-nearest neighbors of a query point when k, ε are only provided during query time. In Sect. 5 we present approximate quorum clustering, see [8,14], for a set of disjoint balls. Using this, in 6, we present the (k, ε)-AVD construction. We conclude in Sect. 7.

2 Problem Definition and Notation

We are given a set of disjoint[1] balls $\mathcal{B} = \{b_1, \ldots, b_n\}$, where $b_i = \mathsf{b}(\mathsf{c}_i, \mathsf{r}_i)$, for $i = 1, \ldots, n$. Here $\mathsf{b}(\mathsf{c}, \mathsf{r}) \subseteq \mathbb{R}^d$ denotes the closed ball with center c and radius $\mathsf{r} \geq 0$. Additionally, we are given an approximation parameter $\varepsilon \in (0, 1)$. For a point $\mathsf{q} \in \mathbb{R}^d$, the *distance* of q to a ball $b = \mathsf{b}(\mathsf{c}, \mathsf{r})$ is

$$\mathsf{d}(\mathsf{q}, b) = \max\Big(\|\mathsf{q} - \mathsf{c}\| - \mathsf{r},\ 0 \Big).$$

Observation 2.1 *For two balls $b_1 \subseteq b_2 \subseteq \mathbb{R}^d$, and any point $\mathsf{q} \in \mathbb{R}^d$, we have $\mathsf{d}(\mathsf{q}, b_1) \geq \mathsf{d}(\mathsf{q}, b_2)$.*

The *kth-nearest neighbor distance* of q to $\mathcal{B}$, denoted by $\mathsf{d}_k(\mathsf{q}, \mathcal{B})$, is the kth smallest number in $\mathsf{d}(\mathsf{q}, b_1), \ldots, \mathsf{d}(\mathsf{q}, b_n)$. Similarly, for a given set of points P, $\mathsf{d}_k(\mathsf{q}, \mathsf{P})$ denotes the kth-nearest neighbor distance of q to P.

Problem definition. We aim to build a data structure to answer $(1 \pm \varepsilon)$-approximate kth-nearest neighbor (i.e., (k, ε)-*ANN*) queries, where for any query point $\mathsf{q} \in \mathbb{R}^d$ one needs to output a ball $b \in \mathcal{B}$ such that, $(1 - \varepsilon)\mathsf{d}_k(\mathsf{q}, \mathcal{B}) \leq \mathsf{d}(\mathsf{q}, b) \leq (1 + \varepsilon)\mathsf{d}_k(\mathsf{q}, \mathcal{B})$ .There are different variants depending on whether ε and k are provided with the query or in advance.

Notation. We use *cube* to denote a set of the form $[a_1, a_1 + \ell] \times [a_2, a_2 + \ell] \times \ldots \times [a_d, a_d + \ell] \subseteq \mathbb{R}^d$, where $a_1, \ldots, a_d \in \mathbb{R}$, and $\ell \geq 0$ is the side length of the cube.

Observation 2.2 *For any set of balls $\mathcal{B}$, the function $\mathsf{d}_k(\mathsf{q}, \mathcal{B})$ is a 1-Lipschitz function; that is, for any two points u, v, we have that $\mathsf{d}_k(\mathsf{u}, \mathcal{B}) \leq \mathsf{d}_k(\mathsf{v}, \mathcal{B}) + \|\mathsf{u} - \mathsf{v}\|$.*

Assumption 2.3 We assume that all the input balls are contained inside the cube $\Big[1/2 - \delta, 1/2 + \delta\Big]^d$, which can be ensured by translation and scaling (which preserves order of distances), where $\delta = \varepsilon/4$. As such, we can ignore queries outside the unit cube $[0, 1]^d$, as any input ball is a valid answer in this case.

For a real positive number x and a point $\mathsf{p} = (\mathsf{p}_1, \ldots, \mathsf{p}_d) \in \mathbb{R}^d$, define $\mathsf{G}_x(\mathsf{p})$ to be the grid point $(\lfloor \mathsf{p}_1/x \rfloor\, x, \ldots, \lfloor \mathsf{p}_d/x \rfloor\, x)$. The number x is the *width* or *side length* of the *grid* G_x. The mapping G_x partitions $\mathbb{R}^d$ into cubes that are grid *cells cell*.

Definition 2.4 A cube is a *canonical cube* if it is contained inside the unit cube $U = [0, 1]^d$, it is a cell in a grid G_r, and r is a power of two, i.e., it might correspond to a node in a quadtree having $[0, 1]^d$ as its root cell (see the book [12] for an introduction to quadtrees). Such a grid G_r is a *canonical grid canonical grid*. Note that all the cells corresponding to nodes of a compressed quadtree are canonical.

[1] Our data structure and algorithm work for the more general case where the balls are interior disjoint, where we define the interior of a "point ball", i.e., a ball of radius 0, as the point itself. This is not the usual topological definition.

Definition 2.5 Given a set $b \subseteq \mathbb{R}^d$, and a parameter $\delta > 0$, let $\mathsf{G}_{\approx}(b, \delta)$ denote the set of coarsest canonical grid cells whose diameter is at most $\delta \operatorname{diam}(b)$ and that intersect b, where $\operatorname{diam}(b) = \max_{\mathsf{p},\mathsf{u} \in b} \|\mathsf{p} - \mathsf{u}\|$ denotes the *diameter* of b. Such a cell has side length $2^{\left\lfloor \log_2 \delta \operatorname{diam}(b)/\sqrt{d} \right\rfloor}$. Clearly, the diameter of any grid cell of $\mathsf{G}_{\approx}(b, \delta)$, is at most $\delta \operatorname{diam}(b)$. Let $\mathsf{G}_{\approx}(b) = \mathsf{G}_{\approx}(b, 1)$. It is easy to verify that $|\mathsf{G}_{\approx}(b)| = O(1)$. The set $\mathsf{G}_{\approx}(b)$ is the *grid approximation* to b.

Let $\mathcal{B}$ be a family of balls in $\mathbb{R}^d$. Given a set $X \subseteq \mathbb{R}^d$, let

$$\mathcal{B}(X) = \left\{ b \in \mathcal{B} \;\middle|\; b \cap X \neq \emptyset \right\}$$

denote the set of balls in $\mathcal{B}$ that intersect X.

For two compact sets $X, Y \subseteq \mathbb{R}^d$, $X \preceq Y$ (equivalently, $Y \succeq X$) if and only if $\operatorname{diam}(X) \leq \operatorname{diam}(Y)$. For a set X and a set of balls $\mathcal{B}$, let $\mathcal{B}_{\succeq}(X) = \left\{ b \in \mathcal{B} \;\middle|\; b \cap X \neq \emptyset \text{ and } b \succeq X \right\}$. Let c_d denote the maximum number of pairwise disjoint balls of radius at least r, that may intersect a given ball of radius r in $\mathbb{R}^d$. Clearly, we have $\left|\mathcal{B}_{\succeq}(b)\right| \leq \mathsf{c}_d$ for any ball b. We have the following bounds.

Lemma 2.6 $2 \leq \mathsf{c}_d \leq 3^d$ *for all* d.

Proof This is an easy result that follows from a standard packing argument, and we include a proof for the sake of completeness. Let $b = \mathsf{b}(\mathsf{c}, \mathsf{r})$ be a given ball of radius r. For the lower bound we can take two balls both of radius r which touch b at diametrically opposite points and lie outside b. We now show the upper bound. Let $\mathcal{B}$ be a set of disjoint balls, each having radius at least r and intersecting b. Consider a ball $b' \in \mathcal{B}$. If no point of the boundary of b' intersects b, then clearly b' contains b in its interior and it is easy to see that $|\mathcal{B}| = 1$. As such we assume that all balls in $\mathcal{B}$ have some point of their boundary inside b. Take any point p of the boundary of b' such that p is in b, and consider a ball of radius r that lies completely inside b', is of radius r and is tangent to b' at p. We can find such a ball for each ball in $\mathcal{B}$. Moreover, these balls are all disjoint. Thus we have $|\mathcal{B}|$ disjoint balls of radius exactly r that intersect b. It is easy to see that all such balls are completely inside $\mathsf{b}(\mathsf{c}, 3\mathsf{r})$. By a simple volume packing bound it follows that $|\mathcal{B}| \leq 3^d$. □

3 Approximate Range Counting for Balls

In the following, we use *cell queries*: Given a compressed quadtree $\widehat{T}$ that stores a set P of n points, and a query canonical grid cell $\widehat{\square}$, we would like to find the *single* node $v \in \widehat{T}$, such that $\mathsf{P} \cap \widehat{\square} = \mathsf{P}_v$ (i.e., a single subtree of $\widehat{T}$ containing all the points of P inside $\widehat{\square}$). Such queries can be readily implemented in $O(\log n)$ time, see [12].

Data-structure 3.1 ($\mathbb{D}$) For a given set of disjoint balls $\mathcal{B} = \{b_1, \ldots, b_n\}$ in $\mathbb{R}^d$, we build the following data structure, that is useful in performing several of the tasks at hand.

(A) *Store balls in a (compressed) quadtree.* For $i = 1, 2, \ldots, n$, let $G_i = \mathsf{G}_{\approx}(b_i)$, and let $G = \bigcup_{i=1}^{n} G_i$ denote the union of these cells. Let $\mathcal{T}$ be a compressed quadtree decomposition of $[0, 1]^d$, such that all the cells of G are cells of $\mathcal{T}$. We preprocess $\mathcal{T}$ to answer point location queries for the cells of G. This takes $O(n \log n)$ time, see [12].

(B) *Compute list of "large" balls intersecting each cell.* For each node u of $\mathcal{T}$, there is a list of balls registered with it. Formally, *register* a ball b_i with all the cells of G_i. Clearly, each ball is registered with $O(1)$ cells, and it is easy to see that each cell has $O(1)$ balls registered with it, since the balls are disjoint.
Next, for a cell $\square$ in $\mathcal{T}$ we compute a list storing $\mathcal{B}_{\geq}(\square)$, and these balls are *associated* with this cell. These lists are computed in a top-down manner. To this end, propagate from a node u its list $\mathcal{B}_{\geq}(\square)$ (which we assume is already computed) down to its children. For a node receiving such a list, it scans it, and keeps only the balls that intersect its cell (adding to this list the balls already registered with this cell). For a node $v \in \mathcal{T}$, let $\mathcal{B}_v$ be this list. Since each node only gets a constant sized list from its parent, and there are $O(n)$ nodes, the total time spent in the list propagation is only $O(n)$.

(C) *Build compressed quadtree on centers of balls.* Let $\mathcal{C}$ be the set of centers of the balls of $\mathcal{B}$. Build, in $O(n \log n)$ time, a compressed quadtree $\mathcal{T}_{\mathcal{C}}$ storing $\mathcal{C}$.

(D) ANN *for centers of balls.* Build a data structure $\mathcal{D}$, for answering 2-approximate k-nearest neighbor distances on $\mathcal{C}$, the set of centers of the balls, see [14], where k and ε are provided with the query. The data structure $\mathcal{D}$, returns a distance ρ such that, $\mathsf{d}_k(\mathsf{q}, \mathcal{C}) \leq \rho \leq 2\mathsf{d}_k(\mathsf{q}, \mathcal{C})$ The space usage of this data structure is $O(n)$ and the query time is $O(\log n)$.

(E) *Answering approximate range searching for the centers of balls.* Given a query ball $b_{\mathsf{q}} = \mathsf{b}(\mathsf{q}, x)$ and a parameter $\delta > 0$, one can, using $\mathcal{T}_{\mathcal{C}}$, report (approximately), in $O(\log n + 1/\delta^d)$ time, the points in $b_{\mathsf{q}} \cap \mathcal{C}$. Specifically, the query process computes $O(1/\delta^d)$ sets of points, such that their union X, has the property that $b_{\mathsf{q}} \cap \mathcal{C} \subseteq X \subseteq (1 + \delta)b_{\mathsf{q}} \cap \mathcal{C}$, where $(1 + \delta)b_{\mathsf{q}}$ is the scaling of b_{q} by a factor of $1 + \delta$ around its center. Indeed, compute the set $\mathsf{G}_{\approx}(b_{\mathsf{q}})$, and then using cell queries in $\mathcal{T}_{\mathcal{C}}$ compute the corresponding cells (this takes $O(\log n)$ time). Now, descend to the relevant level of the quadtree to all the cells of the right size, that intersect b_{q}. Clearly, the union of points stored in their subtrees are the desired set. This takes overall $O(\log n + 1/\delta^d)$ time.
A similar data structure for approximate range searching is provided by Arya and Mount [5], and our description above is provided for the sake of completeness.

Overall, it takes $O(n \log n)$ time to build this data structure.

We denote the collection of data structures above by $\mathbb{D}$ and where necessary, specific functionality it provides, say for finding the large balls intersecting a cell, by $\mathbb{D}(\mathrm{B})$.

3.1 Approximate Range Counting Among Balls

We need the ability to answer approximate range counting queries on a set of disjoint balls. Specifically, given a set of disjoint balls $\mathcal{B}$, and a query ball b, the target is to

compute the size of the set $b \cap \mathcal{B} = \left\{ b' \in \mathcal{B} \;\middle|\; b' \cap b \neq \emptyset \right\}$. Let q be the center of b and let its radius be x. Then, the set $b \cap \mathcal{B}$ is precisely $\mathcal{B}(\mathsf{b}(\mathrm{q}, x))$. To make this query computationally fast, we allow an approximation. More precisely, we compute a number N that lies between $\left|\mathcal{B}\left(\mathsf{b}(\mathrm{q}, x)\right)\right|$ and $\left|\mathcal{B}\left(\mathsf{b}(\mathrm{q}, (1+\delta)x)\right)\right|$, i.e, N lies between the number of balls intersecting b and the number of balls intersecting a $(1+\delta)$ expansion of b.

Lemma 3.2 *Given a compressed quadtree $\mathcal{T}$ of size n, a convex set X, and a parameter $\delta > 0$, one can compute the set of nodes in $\mathcal{T}$, that realizes $\mathsf{G}_{\approx}(X, \delta)$ (see Definition 2.5), in $O\left(\log n + 1/\delta^d\right)$ time. Specifically, this outputs a set X_N of nodes, of size $O\left(1/\delta^d\right)$, such that their cells intersect $\mathsf{G}_{\approx}(X, \delta)$, and their parents cell diameter is larger than $\delta \operatorname{diam}(X)$. Note that the cells in X_N might be significantly larger if they are leaves of $\mathcal{T}$.*

Proof Let $\mathsf{G}_{\approx} = \mathsf{G}_{\approx}(X, 1)$ be the grid approximation to X. Using cell queries on the compressed quadtree, one can compute the cells of $\mathcal{T}$ that correspond to these canonical cells. Specifically, for each cube $\square \in \mathsf{G}_{\approx}(X)$, the query either returns a node for which this is its cell, or it returns a compressed edge of the quadtree; that is, two cells (one is a parent of the other), such that $\square$ is contained in of them and contains the other. Such a cell query takes $O(\log n)$ time [12]. This returns $O(1)$ nodes in $\mathcal{T}$ such that their cells cover $\mathsf{G}_{\approx}(X)$.

Now, traverse down the compressed quadtree starting from these nodes and collect all the nodes of the quadtree that are relevant. Clearly, one has to go at most $O(\log 1/\delta)$ levels down the quadtree to get these nodes, and this takes $O(1/\delta^d)$ time overall. □

Lemma 3.3 *Let X be any convex set in $\mathbb{R}^d$, and let $\delta > 0$ be a parameter. Using $\mathbb{D}(B)$, one can compute, in $O\left(\log n + 1/\delta^d\right)$ time, all the balls of $\mathcal{B}$ that intersect X, with diameter $\geq \delta \operatorname{diam}(X)$.*

Proof We compute the cells of the quadtree realizing $\mathsf{G}_{\approx}(X, \delta)$ using Lemma 3.2. Now, from each such cell (and its parent), we extract the list of large balls intersecting it (there are $O(1/\delta^d)$ such nodes, and the size of each such list is $O(1)$). Next we check for each such ball if it intersects X and if its diameter is at least $\delta \operatorname{diam}(X)$. We return the list of all such balls. □

3.2 Answering a Query

Given a query ball $b_{\mathrm{q}} = \mathsf{b}(\mathrm{q}, x)$, and an approximation parameter $\delta > 0$, our purpose is to compute a number N, such that $\left|\mathcal{B}\left(\mathsf{b}(\mathrm{q}, x)\right)\right| \leq N \leq \left|\mathcal{B}\left(\mathsf{b}(\mathrm{q}, (1+\delta)x)\right)\right|$.

The query algorithm works as follows:

(A) Using Lemma 3.3, compute a set X of all the balls that intersect b_{q} and are of radius $\geq \delta x/4$.
(B) Using $\mathbb{D}(\mathrm{C})$, see Data-structure 3.1, compute the $O(1/\delta^d)$ cells of $\mathcal{T}_{\mathcal{C}}$ that correspond to $\mathsf{G}_{\approx}\left(b_{\mathrm{q}}(1+\delta/4), \delta/4\right)$. Let N' be the total number of points in $\mathcal{C}$ stored in these nodes.

(C) The quantity $N' + |X|$ is almost the desired quantity, except that we might be counting some of the balls of X twice. To this end, let N'' be the number of balls in X with centers in $\mathsf{G}_{\approx}\big(b_{\mathtt{q}}(1+\delta/4), \delta/4\big)$

(D) Let $N \leftarrow N' + |X| - N''$. Return N.

We only sketch the proof, as the proof is straightforward. Indeed, the union of the cells of $\mathsf{G}_{\approx}\big(b_{\mathtt{q}}(1+\delta/4), \delta/4\big)$ contains $\mathsf{b}(\mathtt{q}, x(1+\delta/4))$ and is contained in $\mathsf{b}(\mathtt{q}, (1+\delta)x)$. All the balls with radius smaller than $\delta x/4$ and intersecting $\mathsf{b}(\mathtt{q}, x)$ have their centers in cells of $\mathsf{G}_{\approx}\big(b_{\mathtt{q}}(1+\delta/4), \delta/4\big)$, and their number is computed correctly. Similarly, the "large" balls, i.e., those with radius at least $\delta x/4$, are computed correctly. The last stage ensures we do not over-count by 1 each large ball that also has its center in $\mathsf{G}_{\approx}\big(b_{\mathtt{q}}(1+\delta/4), \delta/4\big)$. It is also easy to check that $|\mathcal{B}(\mathsf{b}(\mathtt{q}, x))| \leq N \leq |\mathcal{B}(\mathsf{b}(\mathtt{q}, x(1+\delta)))|$. We now analyze the running time. Computing cells of $\mathsf{G}_{\approx}\big(b_{\mathtt{q}}(1+\delta/4), \delta/4\big)$ takes $O(\log n + 1/\delta^d)$ time. Computing the "large" balls takes $O\big(\log n + 1/\delta^d\big)$ time. Checking for each large ball if it is already counted by the "small" balls takes $O(1/\delta^d)$ time overall by using a grid. We denote the above query algorithm by $\mathrm{rangeCount}(\mathtt{q}, x, \delta)$.

The above implies the following.

Lemma 3.4 *A given n-element set $\mathcal{B}$ of disjoint balls in $\mathbb{R}^d$ can be preprocessed in $O(n \log n)$ time into a data structure of size $O(n)$, such that given a query ball $\mathsf{b}(\mathtt{q}, x)$ and approximation parameter $\delta > 0$, the query algorithm* $\mathrm{rangeCount}(\mathtt{q}, x, \delta)$ *returns, in $O(\log n + 1/\delta^d)$ time, a number N satisfying $|\mathcal{B}(\mathsf{b}(\mathtt{q}, x))| \leq N \leq |\mathcal{B}(\mathsf{b}(\mathtt{q}, (1+\delta)x))|$.*

4 Answering k-ANN Queries Among Balls

Given a set $\mathcal{B}$ of n disjoint balls, a query point $\mathtt{q}$, and integer k with $1 \leq k \leq n$, in this section we will show how to compute a constant factor approximation to $\mathsf{d}_k(\mathtt{q}, \mathcal{B})$. We also show how to refine the approximation factor to be $1 \pm \varepsilon$. While $\mathcal{B}$ is available for preprocessing, both k and ε need only be provided during query time.

4.1 Computing a Constant Factor Approximation to $\mathsf{d}_k(\mathtt{q}, \mathcal{B})$

Lemma 4.1 *Let $\mathcal{B}$ be a set of disjoint balls in $\mathbb{R}^d$, and consider a ball $b = \mathsf{b}(\mathtt{q}, r)$ that intersects at least k balls of $\mathcal{B}$. Then, among the k nearest neighbors of $\mathtt{q}$ from $\mathcal{B}$, there are at least $\max(0, k - \mathsf{c}_d)$ balls of radius at most r. The centers of all these balls are in $\mathsf{b}(\mathtt{q}, 2r)$.*

Proof Consider the k nearest neighbors of $\mathtt{q}$ from $\mathcal{B}$. Any such ball that has its center outside $\mathsf{b}(\mathtt{q}, 2r)$, has radius at least r, since it intersects $b = \mathsf{b}(\mathtt{q}, r)$. Since the number of balls that are of radius at least r and intersect b is bounded by c_d, there must be at least $\max(0, k - \mathsf{c}_d)$ balls among the k nearest neighbors, each having radius less than r. Clearly, the centers of these balls are in $\mathsf{b}(\mathtt{q}, 2r)$. □

Corollary 4.2 *Let* $\gamma = \min(k, \mathsf{c}_d)$. *Then,* $\mathsf{d}_{k-\gamma}(\mathsf{q}, \mathcal{C})/2 \leq \mathsf{d}_k(\mathsf{q}, \mathcal{B})$.

The basic observation is that we only need a rough approximation to the right radius, as using approximate range counting (i.e., Lemma 3.4), one can improve the approximation.

We first provide some intuition how we compute $\mathsf{d}_k(\mathsf{q}, \mathcal{B})$ approximately. For $1 \leq i \leq n$, let x_i denote the distance of q to the ith closest center in $\mathcal{C}$. Let $d_k = \mathsf{d}_k(\mathsf{q}, \mathcal{B})$. Let i be the minimum index such that $d_k \leq x_i$. Since $d_k \leq x_k$, it must be that $i \leq k$. There are several possibilities:

(A) If $i \leq k - \mathsf{c}_d$ (i.e., $d_k \leq x_{k-\mathsf{c}_d}$) then, by Lemma 4.1, the ball $\mathsf{b}(\mathsf{q}, 2d_k)$ contains at least $k - \mathsf{c}_d$ centers. As such, $d_k \leq x_{k-\mathsf{c}_d} \leq 2d_k$, and $x_{k-\mathsf{c}_d}$ is a good approximation to d_k.
(B) If $i > k - \mathsf{c}_d$, and $d_k \leq 4x_{i-1}$, then x_{i-1} is the desired approximation, since we have $x_{i-1} < d_k \leq 4x_{i-1}$.
(C) If $i > k - \mathsf{c}_d$, and $d_k \geq x_i/4$, then x_i is the desired approximation, since we have $x_i/4 \leq d_k \leq x_i$.
(D) Otherwise, it must be that $i > k - \mathsf{c}_d$, and $4x_{i-1} < d_k < x_i/4$. Now, the centers of $(i-1)$ balls lie within distance x_{i-1} to q but no centers lie in the range of distances (x_{i-1}, x_i). Let $b_j = \mathsf{b}(\mathsf{c}_j, \mathsf{r}_j)$ be the jth closest ball to q, for $j = 1, \ldots, k$. All the balls $b_1, \ldots, b_k$ intersect $\mathsf{b}(\mathsf{q}, d_k)$ but on the other hand, only $(i-1)$ have centers within this ball, since $x_i > 4d_k$.
Thus, at least $k - i + 1$ balls have centers further than x_i but they must come as close as d_k to q. In other words, their radius must be at least $3x_i/4$ and they all intersect the ball $\mathsf{b}(\mathsf{q}, x_i/4)$. We can easily compute these at most $\mathsf{c}_d + 1$ big balls using $\mathbb{D}(\mathrm{B})$. The kth closest ball is also such a ball, and it turns out we can then get a good approximation to its distance from the list of balls computed.

In the preprocessing, we build $\mathbb{D}$ in $O(n \log n)$ time.

To describe our algorithm precisely, first we introduce some notation. For $x \geq 0$, let $N(x)$ denote the number of balls in $\mathcal{B}$ that intersect $\mathsf{b}(\mathsf{q}, x)$; that is $N(x) = \left|\left\{b \in \mathcal{B} \;\middle|\; b \cap \mathsf{b}(\mathsf{q}, x) \neq \emptyset\right\}\right|$, and $C(x)$ denote the number of centers in $\mathsf{b}(\mathsf{q}, x)$, i.e., $C(x) = |\mathcal{C} \cap \mathsf{b}(\mathsf{q}, x)|$. Also, let $\#(x)$ denote the 2-approximation to the number of balls of $\mathcal{B}$ intersecting $\mathsf{b}(\mathsf{q}, x)$, as computed by Lemma 3.4; that is $N(x) \leq \#(x) \leq N(2x)$.

We now provide our algorithm to answer a query. We are given a query point $\mathsf{q} \in \mathbb{R}^d$ and a number k.

Using $\mathbb{D}(\mathrm{D})$, compute a 2-approximation for the smallest ball containing $k - i$ centers of $\mathcal{B}$, for $i = 0, \ldots, \gamma$, where $\gamma = \min(k, \mathsf{c}_d)$, and let r_{k-i} be this radius. That is, for $i = 0, \ldots, \gamma$, we have $C(r_{k-i}/2) \leq k - i \leq C(r_{k-i})$. For $i = 0, \ldots, \gamma$, compute $N_{k-i} = \#(r_{k-i})$ (Lemma 3.4).

Let α be the maximum index such that $N_{k-\alpha} \geq k$. Clearly, α is well defined as $N_k \geq k$. The algorithm is executed in the following steps.

(A) If $\alpha = \gamma$ we return $r_{k-\gamma}$.
(B) If $\#(r_{k-\alpha}/8) < k$, we return $r_{k-\alpha}$.
(C) Otherwise, compute all the balls of $\mathcal{B}$ that are of radius at least $r_{k-\alpha}/4$ and intersect the ball $\mathsf{b}(\mathsf{q}, r_{k-\alpha}/4)$, using $\mathbb{D}(\mathrm{B})$ (see Data-structure 3.1). For each

such ball b, compute the distance $\zeta = \mathsf{d}(\mathrm{q}, b)$ of q to it. Find the minimum ζ such that $\#(2\zeta) \geq k$, and return 2ζ.

Lemma 4.3 *Given a set of n disjoint balls $\mathcal{B}$ in $\mathbb{R}^d$, one can preprocess them in $O(n \log n)$ time into a data structure of size $O(n)$, such that given a query point $\mathrm{q} \in \mathbb{R}^d$, and a number k, one can compute, in $O(\log n)$ time, a number x, such that, $x/8 \leq \mathsf{d}_k(\mathrm{q}, \mathcal{B}) \leq 2x$.*

Proof The data structure and query algorithm are described above. We next prove correctness, by showing in order that for each of (A)–(C), if the algorithm returned in that step, its output satisfies the desired properties.

To prove that (A) returns the correct answer observe that under the given assumptions,

$$r_{k-\gamma}/8 \leq \mathsf{d}_{k-\gamma}(\mathrm{q}, \mathcal{C})/4 \leq \mathsf{d}_k(\mathrm{q}, \mathcal{B})/2 \leq r_{k-\gamma},$$

where the first inequality follows as $r_{k-\gamma}$ is a valid 2 approximation to $\mathsf{d}_{k-\gamma}(\mathrm{q}, \mathcal{C})$, the second inequality follows from Corollary 4.2, and the third inequality follows as $N(2r_{k-\gamma}) \geq \#\left(r_{k-\gamma}\right) \geq k$, while $\mathsf{d}_k(\mathrm{q}, \mathcal{B})$ is the smallest number x such that $N(x) \geq k$.

For (B) observe that we have that $N(r_{k-\alpha}/8) \leq \#(r_{k-\alpha}/8) < k$ and as such we have $r_{k-\alpha}/8 < \mathsf{d}_k(\mathrm{q}, \mathcal{B})$. But by assumption, $\#(r_{k-\alpha}) \geq k$. Thus, $N(2r_{k-\alpha}) \geq \#(r_{k-\alpha}) \geq k$, and $\mathsf{d}_k(\mathrm{q}, \mathcal{B}) \leq 2r_{k-\alpha}$.

For (C), first observe that $\alpha < \gamma$ as the algorithm did not return in (A). Since α is the maximum index such that $\#(r_{k-\alpha}) \geq k$. Namely, $N(r_{k-\alpha-1}) \leq \#(r_{k-\alpha-1}) < k$ implying, $r_{k-\alpha-1} < \mathsf{d}_k(\mathrm{q}, \mathcal{B})$. Since the algorithm did not return in (B) we have that, $\#(r_{k-\alpha}/8) \geq k$ and therefore $N(r_{k-\alpha}/4) \geq \#(r_{k-\alpha}/8) \geq k$, implying $\mathsf{d}_k(\mathrm{q}, \mathcal{B}) \leq r_{k-\alpha}/4$. As such, $r_{k-\alpha-1} < r_{k-\alpha}/4$. Now the ball $\mathsf{b}(\mathrm{q}, r_{k-\alpha-1})$ contains at least $k - \alpha - 1$ centers from $\mathcal{C}$, but it does not contain $k - \alpha$ centers. Indeed, otherwise we would have $\mathsf{d}_{k-\alpha}(\mathrm{q}, \mathcal{C}) \leq r_{k-\alpha-1}$ and $r_{k-\alpha} \leq 2\mathsf{d}_{k-\alpha}(\mathrm{q}, \mathcal{C}) \leq 2r_{k-\alpha-1}$. On the other hand $r_{k-\alpha-1} < \mathsf{d}_k(\mathrm{q}, \mathcal{B}) \leq r_{k-\alpha}/4$, which would be a contradiction. Similarly, there is no center of any ball whose distance from q is in the range $(r_{k-\alpha-1}, r_{k-\alpha}/2)$ otherwise we would have that $\mathsf{d}_{k-\alpha}(\mathrm{q}, \mathcal{C}) < r_{k-\alpha}/2$ and this would mean that $r_{k-\alpha} \leq 2\mathsf{d}_{k-\alpha}(\mathrm{q}, \mathcal{C}) < r_{k-\alpha}$, a contradiction. Now, the center of the kth closest ball is clearly more than $r_{k-\alpha-1}$ away from q, since $N(r_{k-\alpha-1}) < k$. As such its distance from q is at least $r_{k-\alpha}/2$. Since $\mathsf{d}_k(\mathrm{q}, \mathcal{B}) \leq r_{k-\alpha}/4$ it follows that the kth closest ball intersects $\mathsf{b}(\mathrm{q}, r_{k-\alpha}/4)$ and moreover, its radius is at least $r_{k-\alpha}/4$. Since we compute all such balls in (C), we do encounter the kth closest ball. Observe that the distance $\mathsf{d}_k(\mathrm{q}, \mathcal{B})$ to this ball satisfies $\#(2\mathsf{d}_k(\mathrm{q}, \mathcal{B})) \geq N(\mathsf{d}_k(\mathrm{q}, \mathcal{B})) = k$. Thus we return 2ζ for some $\zeta \leq \mathsf{d}_k(\mathrm{q}, \mathcal{B})$. Moreover, in this case ζ satisfies, $N(4\zeta) \geq \#(2\zeta) \geq k$. We conclude that $4\zeta \geq \mathsf{d}_k(\mathrm{q}, \mathcal{B})$. This leads to, $(2\zeta)/2 \leq \mathsf{d}_k(\mathrm{q}, \mathcal{B}) \leq 2(2\zeta)$.

As for the running time, notice that we need to use the algorithm of Lemma 3.4 $O(1)$ times, each iteration taking time $O(\log n)$. After this we need another $O(\log n)$ time for the invocation of the algorithm in Lemma 3.3. As such, the total query time is $O(\log n)$. □

We now show how to refine the approximation.

Lemma 4.4 *Given a set $\mathcal{B}$ of n balls in $\mathbb{R}^d$, it can be preprocessed in $O(n \log n)$ time into a data structure of size $O(n)$, such that, given a query point q, numbers k, x, and an approximation parameter $\varepsilon > 0$ with $x/8 \leq \mathsf{d}_k(\mathsf{q}, \mathcal{B}) \leq 2x$, one can find a ball $b \in \mathcal{B}$ satisfying $(1-\varepsilon)\mathsf{d}_k(\mathsf{q}, \mathcal{B}) \leq \mathsf{d}(\mathsf{q}, b) \leq (1+\varepsilon)\mathsf{d}_k(\mathsf{q}, \mathcal{B})$ in $O\left(\log n + 1/\varepsilon^d\right)$ time.*

Proof We are going to use the same data structure as Lemma 3.4, for the query ball $b_{\mathsf{q}} = \mathsf{b}(\mathsf{q}, 2x(1+\varepsilon))$. We compute all large balls of $\mathcal{B}$ that intersect b_{q}. Here a large ball is a ball of radius $> x\varepsilon$, and a ball of radius at most $x\varepsilon$ is considered to be a small ball. Consider the $O(1/\varepsilon^d)$ grid cells of $\mathsf{G}_{\approx}\left(b_{\mathsf{q}}, \varepsilon/16\right)$. In $O(1/\varepsilon^d)$ time we can record the number of centers of large balls inside any such cell. Clearly, any small ball that intersects $\mathsf{b}(\mathsf{q}, 2x)$ has its center in some cell of $\mathsf{G}_{\approx}\left(b_{\mathsf{q}}, \varepsilon/16\right)$. We use the quadtree $\mathcal{T}_{\mathcal{C}}$ to find out exactly the number of centers, $N_{\square}$, of small balls in each cell $\square$ of $\mathsf{G}_{\approx}\left(b_{\mathsf{q}}, \varepsilon/16\right)$, by finding the total number of centers using $\mathcal{T}_{\mathcal{C}}$, and decreasing this by the count of centers of large balls in that cell. This can be done in time $O(\log n + 1/\varepsilon^d)$. We pick an arbitrary point in $\square$, and assign it weight $N_{\square}$, and treat it as representing all the small balls in this grid cell – clearly, this introduces an error of size $\leq \varepsilon x$ in the distance of such a ball from q, and as such we can ignore it in our argument. In the end of this snapping process, we have $O(1/\varepsilon^d)$ weighted points, and $O(1/\varepsilon^d)$ large balls. We know the distance of the query point from each one of these points/balls. This results in $O(1/\varepsilon^d)$ weighted distances, and we want the smallest ℓ, such that the total weight of the distances $\leq \ell$ is at least k. This can be done by weighted median selection in linear time in the number of distances, which is $O(1/\varepsilon^d)$. Once we get the required point, we can output any ball b corresponding to the point. Clearly, b satisfies the required conditions. □

4.2 The Result

Theorem 4.5 *Given a set of n disjoint balls $\mathcal{B}$ in $\mathbb{R}^d$, one can preprocess them in time $O(n \log n)$ into a data structure of size $O(n)$, such that given a query point $\mathsf{q} \in \mathbb{R}^d$, a number k with $1 \leq k \leq n$ and $\varepsilon > 0$, one can find in time $O\left(\log n + \varepsilon^{-d}\right)$ a ball $b \in \mathcal{B}$, such that, $(1-\varepsilon)\mathsf{d}_k(\mathsf{q}, \mathcal{B}) \leq \mathsf{d}(\mathsf{q}, b) \leq (1+\varepsilon)\mathsf{d}_k(\mathsf{q}, \mathcal{B})$.*

5 Quorum Clustering

For an introduction to quorum clustering on points see [8,14]. Here we compute a similar clustering on balls, that we refer to as quorum clustering on balls. Later, in Sect. 6 we use this to get a sublinear space data structure for the (k, ε)-ANN problem. The following provides both a definition and a computational procedure for computing a quorum clustering on balls. We are given a set $\mathcal{B}$ of n disjoint balls in $\mathbb{R}^d$, and we describe how to compute quorum clustering for them quickly. We assume that $n \geq k > 2\mathsf{c}_d$. Let $m = \lceil n/(k - \mathsf{c}_d) \rceil$.

Let ξ be some constant. Let $\mathcal{B}_0 = \emptyset$. For $i = 1, \ldots, m$, let $\mathcal{R}_i = \mathcal{B} \setminus (\bigcup_{j=0}^{i-1} \mathcal{B}_j)$, and let $\Lambda_i = \mathsf{b}(\mathsf{w}_i, \mathsf{x}_i)$ be any ball that satisfies,

(A) Λ_i contains $\min(k - \mathsf{c}_d, |\mathcal{R}_i|)$ balls of $\mathcal{R}_i$ completely inside it,
(B) Λ_i intersects at least k balls of $\mathcal{B}$, and

(C) the radius of Λ_i is at most ξ times the radius of the smallest ball satisfying the above conditions.

Next, we remove any $\min(k - \mathsf{c}_d, |\mathcal{R}_i|)$ balls that are contained in Λ_i from $\mathcal{R}_i$ to get the set $\mathcal{R}_{i+1}$. We call the removed set of balls $\mathcal{B}_i$. We repeat this process till all balls are extracted, thereby taking m steps overall. Notice that at each step i, we only require that Λ_i intersects k balls of $\mathcal{B}$ (and not $\mathcal{R}_i$), but that it must contain $k - \mathsf{c}_d$ balls from $\mathcal{R}_i$. Also, the last quorum ball may contain fewer balls. The balls $\Lambda_1, \ldots, \Lambda_m$, are the resulting ξ-*approximate quorum clustering*.

5.1 Computing an Approximate Quorum Clustering

Definition 5.1 For a set P of n points in $\mathbb{R}^d$, and an integer ℓ, with $1 \leq \ell \leq n$, let $r_{\text{opt}}(\mathsf{P}, \ell)$ denote the radius of the smallest ball which contains at least ℓ points from P, i.e., $r_{\text{opt}}(\mathsf{P}, \ell) = \min_{\mathsf{q} \in \mathbb{R}^d} \mathsf{d}_\ell(\mathsf{q}, \mathsf{P})$.

Similarly, for a set $\mathcal{R}$ of n balls in $\mathbb{R}^d$, and an integer ℓ, with $1 \leq \ell \leq n$, let $R_{\text{opt}}(\mathcal{R}, \ell)$ denote the radius of the smallest ball which completely contains at least ℓ balls from $\mathcal{R}$.

Lemma 5.2 *([14]) Given a set P of n points in $\mathbb{R}^d$ and an integer ℓ, with $1 \leq \ell \leq n$, one can compute, in $O(n \log n)$ time, a sequence of $\lceil n/\ell \rceil$ balls, $\mathsf{o}_1 = \mathsf{b}(\mathsf{u}_1, \psi_1), \ldots, \mathsf{o}_{\lceil n/\ell \rceil} = \mathsf{b}(\mathsf{u}_{\lceil n/\ell \rceil}, \psi_{\lceil n/\ell \rceil})$, such that, for all i, $1 \leq i \leq \lceil n/\ell \rceil$, we have*

(A) For every ball o_i, there is an associated subset P_i of $\min(\ell, |\mathsf{Q}_i|)$ points of $\mathsf{Q}_i = \mathsf{P} \setminus (\mathsf{P}_{i-1} \cup \ldots \cup \mathsf{P}_0)$ that it covers, where $\mathsf{P}_0 = \emptyset$.
(B) The ball $\mathsf{o}_i = \mathsf{b}(\mathsf{u}_i, \psi_i)$ is a 2-approximation to the smallest ball covering $\min(\ell, |\mathsf{Q}_i|)$ points in Q_i. That is $\psi_i/2 \leq r_{\text{opt}}(\mathsf{Q}_i, \min(\ell, |\mathsf{Q}_i|)) \leq \psi_i$.

The algorithm to construct an approximate quorum clustering is as follows. First, we use the algorithm of Lemma 5.2 with the set of points $\mathsf{P} = \mathcal{C}$, and $\ell = k - \mathsf{c}_d$ to get a list of $m = \lceil n/(k - \mathsf{c}_d) \rceil$ balls $\mathsf{o}_1 = \mathsf{b}(\mathsf{u}_1, \psi_1), \ldots, \mathsf{o}_m = \mathsf{b}(\mathsf{u}_m, \psi_m)$, satisfying the conditions of Lemma 5.2. Next we use the algorithm of Theorem 4.5, to compute (k, ε)-ANN distances from the centers $\mathsf{u}_1, \ldots, \mathsf{u}_m$, to the balls of $\mathcal{B}$.

Thus, we get numbers γ_i satisfying, $(1/2)\mathsf{d}_k(\mathsf{u}_i, \mathcal{B}) \leq \gamma_i \leq (3/2)\mathsf{d}_k(\mathsf{u}_i, \mathcal{B})$. Let $\zeta_i = \max(2\gamma_i, 3\psi_i)$, for $i = 1, \ldots, m$. Now, if $(k - \mathsf{c}_d) \mid n$ we sort the numbers $\zeta_1, \ldots, \zeta_m$. Suppose the sorted order is the permutation π of $\{1, \ldots, m\}$. We output the balls $\Lambda_i = \mathsf{b}(\mathsf{u}_{\pi(i)}, \zeta_{\pi(i)})$, for $i = 1, \ldots, m$ as the approximate quorum clustering. On the other hand, if $(k - \mathsf{c}_d) \nmid n$, we sort the numbers $\zeta_1, \ldots, \zeta_{m-1}$, and suppose the sorted order is the permutation π of $\{1, \ldots, m-1\}$. We output the balls $\Lambda_i = \mathsf{b}(\mathsf{u}_{\pi(i)}, \zeta_{\pi(i)})$, for $i = 1, \ldots, m-1$, followed by $\mathsf{b}(\mathsf{u}_m, \zeta_m)$ as the approximate quorum clustering. Thus, for the case $(k - \mathsf{c}_d) \nmid n$, this ensures that except for the last cluster $\mathsf{b}(\mathsf{u}_m, \zeta_m)$, the others each contain at least $k - \mathsf{c}_d$ balls of $\mathcal{B}$ inside them.

5.2 Correctness

Lemma 5.3 *Let $\mathcal{B} = \{b_1, \ldots, b_n\}$ be a set of n disjoint balls, where $b_i = \mathsf{b}(\mathsf{c}_i, \mathsf{r}_i)$, for $i = 1, \ldots, n$. Let $\mathcal{C} = \{\mathsf{c}_1, \ldots, \mathsf{c}_n\}$ be the set of centers of these balls. Let $b = \mathsf{b}(\mathsf{c}, \mathsf{r})$ be any ball that contains at least ℓ centers from $\mathcal{C}$, for some $2 \leq \ell \leq n$. Then $\mathsf{b}(\mathsf{c}, 3\mathsf{r})$ contains the ℓ balls that correspond to those centers.*

Proof Without loss of generality suppose b contains the ℓ centers $\mathsf{c}_1, \ldots, \mathsf{c}_\ell$, from $\mathcal{C}$. Now consider any index i with $1 \leq i \leq \ell$, and consider any $j \neq i$, which exists as $\ell \geq 2$ by assumption. Since $\mathsf{b}(\mathsf{c}, \mathsf{r})$ contains both c_i and c_j, $2\mathsf{r} \geq \|\mathsf{c}_i - \mathsf{c}_j\|$ by the triangle inequality. On the other hand, as the balls b_i and b_j are disjoint we have that $\|\mathsf{c}_i - \mathsf{c}_j\| \geq \mathsf{r}_i + \mathsf{r}_j \geq \mathsf{r}_i$. It follows that $\mathsf{r}_i \leq 2\mathsf{r}$ for all $1 \leq i \leq \ell$. As such the ball $\mathsf{b}(\mathsf{c}, 3\mathsf{r})$ must contain the entire ball b_i, and thus it contains all the ℓ balls $b_1, \ldots, b_\ell$, corresponding to the centers. □

Lemma 5.4 *Let $\mathcal{B} = \{b_1 = \mathsf{b}(\mathsf{c}_1, \mathsf{r}_1), \ldots, b_n = \mathsf{b}(\mathsf{c}_n, \mathsf{r}_n)\}$ be a set of n disjoint balls in $\mathbb{R}^d$. Let $\mathcal{C} = \{\mathsf{c}_1, \ldots, \mathsf{c}_n\}$ be the corresponding set of centers, and let ℓ be an integer with $2 \leq \ell \leq n$. Then, $r_{\mathrm{opt}}(\mathcal{C}, \ell) \leq R_{\mathrm{opt}}(\mathcal{B}, \ell) \leq 3r_{\mathrm{opt}}(\mathcal{C}, \ell)$.*

Proof The first inequality follows since the ball realizing the optimal covering of ℓ balls, clearly contains their centers as well, and therefore ℓ points from $\mathcal{C}$. To see the second inequality, consider the ball $b = \mathsf{b}(\mathsf{c}, \mathsf{r})$ realizing $r_{\mathrm{opt}}(\mathcal{C}, \ell)$, and use Lemma 5.3 on it. This implies $R_{\mathrm{opt}}(\mathcal{B}, \ell) \leq 3r_{\mathrm{opt}}(\mathcal{C}, \ell)$. □

Lemma 5.5 *The balls $\Lambda_1, \ldots \Lambda_m$ computed above are a 12-approximate quorum clustering of $\mathcal{B}$.*

Proof The proof in the case $(k - \mathsf{c}_d) \mid n$ is similar and easier than the proof in the case $(k - \mathsf{c}_d) \nmid n$. As such we only prove this for the case $(k - \mathsf{c}_d) \nmid n$.

Consider the balls $\mathsf{o}_1 = \mathsf{b}(\mathsf{u}_1, \psi_1), \ldots, \mathsf{o}_m = \mathsf{b}(\mathsf{u}_m, \psi_m)$ computed by the algorithm of Lemma 5.2. Suppose $\mathcal{C}_i$, for $i = 1, \ldots, m$, is the set of centers assigned to the balls b_i. That is, $\mathcal{C}_1, \ldots, \mathcal{C}_m$, form a disjoint decomposition of $\mathcal{C}$, where each of them except for the last one, is of size $k - \mathsf{c}_d$.

For $i = 1, \ldots, m$, let $\mathcal{B}_i$ denote the set of balls corresponding to the centers in $\mathcal{C}_i$. Now, while constructing the approximate quorum clusters we are going to assign the set of balls $\mathcal{B}_{\pi(i)}$ for $i = 1, \ldots, m-1$, to Λ_i, and the balls of $\mathcal{B}_m$ to Λ_m.

We define $\pi(0) = 0$ and $\mathcal{B}_0 = \emptyset$. For $i = 1, \ldots, m$, let $\mathcal{R}_i = \mathcal{B} \setminus (\bigcup_{j=0}^{i-1} \mathcal{B}_{\pi(j)})$ denote the set of remaining balls to choose from after $i - 1$ of the quorum clusters have been output. We show that the required guarantees hold for the quorum clusters. Fix an i with $0 \leq i \leq m - 1$, and suppose that the clusters upto i have been output and satisfy the required properties, and now we need to output the $(i + 1)$-th cluster. The balls of $\bigcup_{j=0}^{i} \mathcal{B}_{\pi(j)}$ have been used up, and the balls in $\mathcal{R}_{i+1}$ remain unused so far. Consider an optimal ball, i.e., a ball $b = \mathsf{b}(\mathsf{c}, \mathsf{r})$ that contains completely $\min(k - \mathsf{c}_d, |\mathcal{R}_{i+1}|)$ balls among $\mathcal{R}_{i+1}$, intersects k balls from $\mathcal{B}$, and is the smallest such possible one. Fix some $\min(k - \mathsf{c}_d, |\mathcal{R}_{i+1}|)$ balls from $\mathcal{R}_{i+1}$ that this optimal ball contains. Consider the sets of centers $\mathcal{C}'$ of these balls. The quorum clusters $\mathsf{o}_{\pi(j)}$ for $j = i + 1, \ldots, m - 1$, and o_m, contain all these centers, by construction. Out of these

indices, i.e., out of the indices $\{\pi(i+1), \ldots, \pi(m-1), m\}$, let p be the minimum index such that o_p contains one of these centers. Notice that if $i+1 < m$, then p is one of $\{\pi(i+1), \ldots, \pi(m-1)\}$ because o_m contains fewer than $k - \mathsf{c}_d$ centers. Now, when o_p was constructed, i.e., at the pth iteration of the algorithm of Lemma 5.2, all the centers from $\mathcal{C}'$ were available. Since the optimal ball $b = \mathsf{b}(\mathsf{c}, \mathsf{r})$ contains $\min(k - \mathsf{c}_d, |\mathcal{R}_{i+1}|)$ available centers too, it follows that $\psi_p \leq 2\mathsf{r}$ since Lemma 5.2 guarantees this. Moreover, by the Lipschitz property, see Observation 2.2, it follows that $\mathsf{d}_k(\mathsf{u}_p, \mathcal{B}) \leq \mathsf{d}_k(\mathsf{c}, \mathcal{B}) + \|\mathsf{u}_p - \mathsf{c}\| \leq \mathsf{r} + (\mathsf{r} + \psi_p) \leq 4\mathsf{r}$, where the second last inequality follows as the balls $b = \mathsf{b}(\mathsf{c}, \mathsf{r})$ and the ball $\mathsf{o}_p = \mathsf{b}(\mathsf{u}_p, \psi_p)$ intersect. Therefore, for the index p we have that, $2\gamma_p \leq 3\mathsf{d}_k(\mathsf{u}_p, \mathcal{B}) \leq 12\mathsf{r}$, and also that $3\psi_p \leq 6\mathsf{r}$. As such $\zeta_p = \max(2\gamma_p, 3\psi_p) \leq 12\mathsf{r}$. For $i+1 < m$, the index $\pi(i+1)$ minimizes this quantity among the indices $\{\pi(i+1), \ldots, \pi(m-1)\}$ (as we took the sorted order), as such it follows that $\zeta_{i+1} \leq 12\mathsf{r}$. It is also easy to see that the inequality holds true if $i+1 = m$.

The proof is completed by showing that the ball Λ_{i+1} contains the balls assigned to it. For the remainder of the proof, let p denote $\pi(i+1)$ if $i+1 < m$, and m otherwise. Since $k > 2\mathsf{c}_d$ it follows that $k - \mathsf{c}_d \geq 2$. Thus, if $|\mathcal{R}_{i+1}| \geq 2$ then $\min(k - \mathsf{c}_d, |\mathcal{R}_{i+1}|) \geq 2$ and therefore, by Lemma 5.3, $\mathsf{b}(\mathsf{u}_p, 3\psi_p)$ contains the balls of $\mathcal{B}_p$. On the other hand, suppose $|\mathcal{R}_{i+1}| = 1$. Since $k - \mathsf{c}_d \geq 2$ this can only happen when $i+1 = m$. In this case, $p = m$ as well, and $\psi_m = 0$. However, since $k \geq 2$, the number $\mathsf{d}_k(\mathsf{u}_m, \mathcal{B})$ is at least as large as the radius of the only ball b in $\mathcal{R}_m$ since all balls are disjoint. But then, $\mathsf{d}_k(\mathsf{u}_m, \mathcal{B}) \leq 2\gamma_m$ and so the ball Λ_m of radius $\zeta_m = \max(2\gamma_m, 3\psi_m)$ contains b. □

Lemma 5.6 *Given a set $\mathcal{B}$ of n disjoint balls in $\mathbb{R}^d$, and a number k with $2\mathsf{c}_d < k \leq n$, in $O(n \log n)$ time, one can output a sequence of $m = \lceil n/(k - \mathsf{c}_d) \rceil$ balls $\Lambda_1, \ldots, \Lambda_m$, such that the following is true for all $1 \leq i \leq m$.*

- *(A) For each ball Λ_i, there is an associated subset $\mathcal{B}_i$ of $\min(k - \mathsf{c}_d, |\mathcal{R}_i|)$ balls of $\mathcal{R}_i = \mathcal{B} \setminus (\mathcal{B}_0 \cup \mathcal{B}_1 \cup \ldots \cup \mathcal{B}_{i-1})$, that it completely covers, where $\mathcal{B}_0 = \emptyset$.*
- *(B) The ball Λ_i intersects at least k balls from $\mathcal{B}$.*
- *(C) The radius of the ball Λ_i is at most 12 times that of the smallest ball covering $\min(k - \mathsf{c}_d, |\mathcal{R}_i|)$ balls of $\mathcal{R}_i$ completely, and intersecting k balls of $\mathcal{B}$.*

Proof The correctness was proved in Lemma 5.5. To see the time bound is also easy as the computation time is dominated by the time in Lemma 5.2, which is $O(n \log n)$. □

6 Construction of the Sublinear Space Data Structure for (k, ε)- ANN

In this section we show that if both k and ε are provided at the time of preprocessing, we can compute an approximate Voronoi diagram for approximating the kth-nearest ball, that takes $O(n/k)$ space, thus improving upon the results of Sect. 4. We assume $k > 2\mathsf{c}_d$ without loss of generality, and we let $m = \lceil n/(k - \mathsf{c}_d) \rceil = O(n/k)$. Here k and ε are prespecified in advance.

6.1 Preliminaries

The following notation was introduced in [14]. A ball b of radius r in $\mathbb{R}^d$, centered at a point c, can be interpreted as a point in $\mathbb{R}^{d+1}$, denoted by $b' = (\mathsf{c}, \mathsf{r})$. For a regular point $\mathsf{p} \in \mathbb{R}^d$, its corresponding image under this transformation is the *mapped* point $\mathsf{p}' = (\mathsf{p}, 0) \in \mathbb{R}^{d+1}$. Namely, we view it as a ball of radius 0 and use the mapping defined on balls. Given a point $\mathsf{u} = (\mathsf{u}_1, \ldots, \mathsf{u}_d) \in \mathbb{R}^d$ its Euclidean norm is denoted by $\|\mathsf{u}\|$. A point $\mathsf{u} = (\mathsf{u}_1, \mathsf{u}_2, \ldots, \mathsf{u}_{d+1}) \in \mathbb{R}^{d+1}$ can be interpreted as being in the product metric of $\mathbb{R}^d \times \mathbb{R}$ and endowed with the product metric norm

$$\|\mathsf{u}\|_{\oplus} = \sqrt{\mathsf{u}_1^2 + \cdots + \mathsf{u}_d^2} + |\mathsf{u}_{d+1}| \,.$$

It can be verified that the above defines a norm, and for any $\mathsf{u} \in \mathbb{R}^{d+1}$ we have $\|\mathsf{u}\| \leq \|\mathsf{u}\|_{\oplus} \leq \sqrt{2}\,\|\mathsf{u}\|$.

6.2 Construction

The input is a set $\mathcal{B}$ of n disjoint balls in $\mathbb{R}^d$, and parameters k and ε.

The construction of the data structure is similar to the construction of the kth-nearest neighbor data structure from the authors' paper [14]. We compute, using Lemma 5.6, a ξ-approximate quorum clustering of $\mathcal{B}$ with $m = \lceil n/(k - \mathsf{c}_d) \rceil = O(n/k)$ balls, $\Sigma = \{\Lambda_1 = \mathsf{b}(\mathsf{w}_1, \mathsf{x}_1), \ldots, \Lambda_m = \mathsf{b}(\mathsf{w}_m, \mathsf{x}_m)\}$, where $\xi \leq 12$. The algorithm then continues as follows:

(A) Compute an exponential grid around each quorum cluster. Specifically, let

$$\mathcal{I} = \bigcup_{i=1}^{m} \bigcup_{j=0}^{\lceil \log(32\xi/\varepsilon) \rceil} \mathsf{G}_{\approx}\left(\mathsf{b}(\mathsf{w}_i, 2^j \mathsf{x}_i), \frac{\varepsilon}{\zeta_1}\right) \tag{6.1}$$

be the set of grid cells covering the quorum clusters and their immediate environ, where ζ_1 is a sufficiently large constant (say, $\zeta_1 = 256\xi$).

(B) Intuitively, $\mathcal{I}$ takes care of the region of space immediately next to a quorum cluster[2]. For the other regions of space, we can apply a construction of an approximate Voronoi diagram for the centers of the clusters (the details are somewhat more involved). To this end, lift the quorum clusters into points in $\mathbb{R}^{d+1}$, as follows

$$\Sigma' = \left\{\Lambda'_1, \ldots, \Lambda'_m\right\},$$

where $\Lambda'_i = (\mathsf{w}_i, \mathsf{x}_i) \in \mathbb{R}^{d+1}$, for $i = 1, \ldots, m$. Note that all points in Σ' belong to $U' = [0, 1]^{d+1}$ by Assumption 2.3. Now build a $(1 + \varepsilon/8)$-AVD for Σ' using the algorithm of Arya and Malamatos [2], for distances specified by the $\|\cdot\|_{\oplus}$ norm. The AVD construction provides a list of canonical cubes covering $[0, 1]^{d+1}$ such that in the smallest cube containing the query point, the associated

[2] That is, intuitively, if the query point falls into one of the grid cells of $\mathcal{I}$, we can answer a query in constant time.

point of Σ', is a $(1+\varepsilon/8)$-ANN to the query point. (Note that these cubes are not necessarily disjoint. In particular, the smallest cube containing the query point q is the one that determines the assigned approximate nearest neighbor to q.)
Clip this collection of cubes to the hyperplane $x_{d+1} = 0$ (i.e., throw away cubes that do not have a face on this hyperplane). For a cube $\square$ in this collection, denote by $\mathrm{nn}'(\square)$, the point of Σ' assigned to it. Let $\mathcal{S}$ be this resulting set of canonical d-dimensional cubes.

(C) Let $\mathcal{W}$ be the space decomposition resulting from overlaying the two collection of cubes, i.e. $\mathcal{I}$ and $\mathcal{S}$. Formally, we compute a compressed quadtree $\mathcal{T}$ that has all the canonical cubes of $\mathcal{I}$ and $\mathcal{S}$ as nodes, and $\mathcal{W}$ is the resulting decomposition of space into cells. One can overlay two compressed quadtrees representing the two sets in linear time [10,12]. Here, a cell associated with a leaf is a canonical cube, and a cell associated with a compressed node is the set difference of two canonical cubes. Each node in this compressed quadtree contains two pointers – to the smallest cube of $\mathcal{I}$, and to the smallest cube of $\mathcal{S}$, that contains it. This information can be computed by doing a BFS on the tree.
For each cell $\square \in \mathcal{W}$ we store the following.
  (I) An arbitrary representative point $\square_{\mathsf{rep}} \in \square$.
  (II) The point $\mathrm{nn}'(\square) \in \Sigma'$ that is associated with the smallest cell of $\mathcal{S}$ that contains this cell. We also store an arbitrary ball, $\mathbf{b}(\square) \in \mathcal{B}$, that is one of the balls completely inside the cluster specified by $\mathrm{nn}'(\square)$ – we assume we stored such a ball inside each quorum cluster, when it was computed.
  (III) A number $\beta_k(\square_{\mathsf{rep}})$ that satisfies $\mathsf{d}_k(\square_{\mathsf{rep}}, \mathcal{B}) \leq \beta_k(\square_{\mathsf{rep}}) \leq (1+\varepsilon/4)\mathsf{d}_k(\square_{\mathsf{rep}}, \mathcal{B})$, and a ball $\mathrm{nn}_k(\square_{\mathsf{rep}}) \in \mathcal{B}$ that realizes this distance. In order to compute $\beta_k(\square_{\mathsf{rep}})$ and $\mathrm{nn}_k(\square_{\mathsf{rep}})$ use the data structure of Sect. 4, see Theorem 4.5.

6.3 Answering a Query

Given a query point q, compute the leaf cell (equivalently the smallest cell) in $\mathcal{W}$ that contains q by performing a point-location query in the compressed quadtree $\mathcal{T}$. Let $\square$ be this cell. Let,

$$\lambda^* = \min\Big(\big\|\mathrm{q}' - \mathrm{nn}'(\square)\big\|_{\oplus}, \beta_k(\square_{\mathsf{rep}}) + \big\|\mathrm{q} - \square_{\mathsf{rep}}\big\|\Big). \tag{6.2}$$

If $\mathrm{diam}(\square) \leq (\varepsilon/8)\lambda^*$ we return $\mathrm{nn}_k(\square_{\mathsf{rep}})$ as the approximate kth-nearest neighbor, else we return $\mathbf{b}(\square)$.

6.4 Correctness

Lemma 6.1 *The number* $\lambda^* = \min\Big(\big\|\mathrm{q}' - \mathrm{nn}'(\square)\big\|_{\oplus}, \beta_k(\square_{\mathsf{rep}}) + \big\|\mathrm{q} - \square_{\mathsf{rep}}\big\|\Big)$ *satisfies,* $\mathsf{d}_k(\mathrm{q}, \mathcal{B}) \leq \lambda^*$.

Proof This follows by the Lipschitz property, see Observation 2.2. □

Lemma 6.2 *Let* $\square \in \mathcal{W}$ *be any cell containing* q. *If* $\operatorname{diam}(\square) \leq \varepsilon \mathsf{d}_k(\mathsf{q}, \mathcal{B})/4$, *then* $\operatorname{nn}_k(\square_{\mathsf{rep}})$ *is a valid* $(1 \pm \varepsilon)$*-approximate kth-nearest neighbor of* q.

Proof For the point $\square_{\mathsf{rep}}$, by Observation 2.2, we have that

$$\begin{aligned}\mathsf{d}_k(\square_{\mathsf{rep}}, \mathcal{B}) &\leq \mathsf{d}_k(\mathsf{q}, \mathcal{B}) + \|\mathsf{q} - \square_{\mathsf{rep}}\| \leq \mathsf{d}_k(\mathsf{q}, \mathcal{B}) + \operatorname{diam}(\square) \\ &\leq (1 + \varepsilon/4)\mathsf{d}_k(\mathsf{q}, \mathcal{B}).\end{aligned}$$

Therefore, the ball $\operatorname{nn}_k(\square_{\mathsf{rep}})$ satisfies

$$\begin{aligned}\mathsf{d}(\square_{\mathsf{rep}}, \operatorname{nn}_k(\square_{\mathsf{rep}})) &\leq (1 + \varepsilon/4)\mathsf{d}_k(\square_{\mathsf{rep}}, \mathcal{B}) \leq (1 + \varepsilon/4)^2 \mathsf{d}_k(\mathsf{q}, \mathcal{B}) \\ &\leq (1 + 3\varepsilon/4)\mathsf{d}_k(\mathsf{q}, \mathcal{B}).\end{aligned}$$

As such we have that

$$\begin{aligned}\mathsf{d}(\mathsf{q}, \operatorname{nn}_k(\square_{\mathsf{rep}})) &\leq \mathsf{d}(\square_{\mathsf{rep}}, \operatorname{nn}_k(\square_{\mathsf{rep}})) + \|\mathsf{q} - \square_{\mathsf{rep}}\| \\ &\leq ((1 + 3\varepsilon/4) + \varepsilon/4)\, \mathsf{d}_k(\mathsf{q}, \mathcal{B}) \leq (1 + \varepsilon)\mathsf{d}_k(\mathsf{q}, \mathcal{B}).\end{aligned}$$

Similarly, using the Lipschitz property, we can argue that, $\mathsf{d}(\mathsf{q}, \operatorname{nn}_k(\square_{\mathsf{rep}})) \geq (1 - \varepsilon)\mathsf{d}_k(\mathsf{q}, \mathcal{B})$, and therefore we have, $(1 - \varepsilon)\mathsf{d}_k(\mathsf{q}, \mathcal{B}) \leq \mathsf{d}(\mathsf{q}, \operatorname{nn}_k(\square_{\mathsf{rep}})) \leq (1 + \varepsilon)\mathsf{d}_k(\mathsf{q}, \mathcal{B})$, and the required guarantees are satisfied. □

Lemma 6.3 *For any point* $\mathsf{q} \in \mathbb{R}^d$ *there is a quorum ball* $\Lambda_i = \mathsf{b}(\mathsf{w}_i, \mathsf{x}_i)$ *such that*

(A) Λ_i *intersects* $\mathsf{b}(\mathsf{q}, \mathsf{d}_k(\mathsf{q}, \mathcal{B}))$,
(B) $\mathsf{x}_i \leq 3\xi \mathsf{d}_k(\mathsf{q}, \mathcal{B})$, *and*
(C) $\|\mathsf{q} - \mathsf{w}_i\| \leq 4\xi \mathsf{d}_k(\mathsf{q}, \mathcal{B})$.

Proof By assumption, $k > 2\mathsf{c}_d$, and by Lemma 4.1 among the k nearest neighbor of q, there are $k - \mathsf{c}_d$ balls of radius at most $\mathsf{d}_k(\mathsf{q}, \mathcal{B})$. Let $\mathcal{B}'$ denote the set of these balls. Among the indices $1, \ldots, m$, let i be the minimum index such that one of these $k - \mathsf{c}_d$ balls is completely covered by the quorum cluster $\Lambda_i = \mathsf{b}(\mathsf{w}_i, \mathsf{x}_i)$. Since $\mathsf{b}(\mathsf{q}, \mathsf{d}_k(\mathsf{q}, \mathcal{B}))$ intersects the ball while Λ_i completely contains it, clearly Λ_i intersects $\mathsf{b}(\mathsf{q}, \mathsf{d}_k(\mathsf{q}, \mathcal{B}))$. Now consider the time Λ_i was constructed, i.e, the ith iteration of the quorum clustering algorithm. At this time, by assumption, all of $\mathcal{B}'$ was available, i.e., none of its balls were assigned to earlier quorum clusters. The ball $\mathsf{b}(\mathsf{q}, 3\mathsf{d}_k(\mathsf{q}, \mathcal{B}))$ contains $k - \mathsf{c}_d$ unused balls and touches k balls from $\mathcal{B}$, as such the smallest such ball had radius at most $3\mathsf{d}_k(\mathsf{q}, \mathcal{B})$. By the guarantee on quorum clustering, $\mathsf{x}_i \leq 3\xi \mathsf{d}_k(\mathsf{q}, \mathcal{B})$. As for the last part, as the balls $\mathsf{b}(\mathsf{q}, \mathsf{d}_k(\mathsf{q}, \mathcal{B}))$ and $\Lambda_i = \mathsf{b}(\mathsf{w}_i, \mathsf{x}_i)$ intersect and $\mathsf{x}_i \leq 3\xi \mathsf{d}_k(\mathsf{q}, \mathcal{B})$, we have by the triangle inequality that $\|\mathsf{q} - \mathsf{w}_i\| \leq (1 + 3\xi)\mathsf{d}_k(\mathsf{q}, \mathcal{B}) \leq 4\xi \mathsf{d}_k(\mathsf{q}, \mathcal{B})$, as $\xi \geq 1$. □

Definition 6.4 For a given query point, any quorum cluster that satisfies the conditions of Lemma 6.3 is defined to be an *anchor cluster*. By Lemma 6.3 an anchor cluster always exists.

Lemma 6.5 *Suppose that among the quorum cluster balls $\Lambda_1, \ldots, \Lambda_m$, there is some ball $\Lambda_i = \mathsf{b}(\mathsf{w}_i, \mathsf{x}_i)$ which satisfies that $\|\mathsf{q} - \mathsf{w}_i\| \leq 8\xi \mathsf{d}_k(\mathsf{q}, \mathcal{B})$ and $\varepsilon \mathsf{d}_k(\mathsf{q}, \mathcal{B})/4 \leq \mathsf{x}_i \leq 8\xi \mathsf{d}_k(\mathsf{q}, \mathcal{B})$ then the output of the algorithm is correct.*

Proof We have

$$\frac{32\xi \mathsf{x}_i}{\varepsilon} \geq \frac{32\xi(\varepsilon \mathsf{d}_k(\mathsf{q}, \mathcal{B})/4)}{\varepsilon} = 8\xi \mathsf{d}_k(\mathsf{q}, \mathcal{B}) \geq \|\mathsf{q} - \mathsf{w}_i\|.$$

Thus, by construction, the expanded environ of the quorum cluster $\mathsf{b}(\mathsf{w}_i, \mathsf{x}_i)$ contains the query point, see Eq. $(6.1)_{p17}$. Let j be the smallest non-negative integer such that $2^j \mathsf{x}_i \geq \mathsf{d}(\mathsf{q}, \mathsf{w}_i)$. We have that, $2^j \mathsf{x}_i \leq \max(\mathsf{x}_i, 2\mathsf{d}(\mathsf{q}, \mathsf{w}_i))$. As such, if $\square$ is the smallest cell in $\mathcal{W}$ containing the query point q, then

$$\begin{aligned}\operatorname{diam}(\square) &\leq \frac{\varepsilon}{\zeta_1} 2^{j+1} \mathsf{x}_i \leq \frac{\varepsilon}{\zeta_1} \cdot \max(2\mathsf{x}_i, 4\mathsf{d}(\mathsf{q}, \mathsf{w}_i)) \\ &\leq \frac{\varepsilon}{\zeta_1} \cdot \max\Big(16\xi \mathsf{d}_k(\mathsf{q}, \mathcal{B}), 32\xi \mathsf{d}_k(\mathsf{q}, \mathcal{B})\Big) \leq \frac{\varepsilon}{8} \mathsf{d}_k(\mathsf{q}, \mathcal{B}),\end{aligned}$$

by Eq. $(6.1)_{p17}$, and if $\zeta_1 \geq 256\xi$. Now, by Lemma 6.1 we have that $\lambda^* \geq \mathsf{d}_k(\mathsf{q}, \mathcal{B})$. As such, $\operatorname{diam}(\square) \leq (\varepsilon/8)\lambda^*$. Therefore, the algorithm returns $\mathrm{nn}_k(\square_{\mathsf{rep}})$ as the $(1 \pm \varepsilon)$-approximate kth-nearest neighbor, but then by Lemma 6.2 it is a correct answer. □

Lemma 6.6 *The query algorithm always outputs a correct approximate answer, i.e., the output ball b satisfies $(1 - \varepsilon)\mathsf{d}_k(\mathsf{q}, \mathcal{B}) \leq \mathsf{d}(\mathsf{q}, b) \leq (1 + \varepsilon)\mathsf{d}_k(\mathsf{q}, \mathcal{B})$.*

Proof Suppose that among the quorum cluster balls $\Lambda_1 = \mathsf{b}(\mathsf{w}_1, \mathsf{x}_1), \ldots, \Lambda_m = \mathsf{b}(\mathsf{w}_m, \mathsf{x}_m)$, there is some ball Λ_i such that we have, $\|\mathsf{q} - \mathsf{w}_i\| \leq 8\xi \mathsf{d}_k(\mathsf{q}, \mathcal{B})$ and $(\varepsilon/4)\mathsf{d}_k(\mathsf{q}, \mathcal{B}) \leq \mathsf{x}_i \leq 8\xi \mathsf{d}_k(\mathsf{q}, \mathcal{B})$, then by Lemma 6.5 the algorithm returns a valid approximate answer. Assume this condition is not satisfied. Let the anchor cluster be $\Lambda = \mathsf{b}(\mathsf{w}, \mathsf{x})$. Since the anchor cluster satisfies $\|\mathsf{q} - \mathsf{w}\| \leq 4\xi \mathsf{d}_k(\mathsf{q}, \mathcal{B})$ and $\mathsf{x} \leq 3\xi \mathsf{d}_k(\mathsf{q}, \mathcal{B})$, it must be the case that, $\mathsf{x} < (\varepsilon/4)\mathsf{d}_k(\mathsf{q}, \mathcal{B})$. Since the anchor cluster intersects $\mathsf{b}(\mathsf{q}, \mathsf{d}_k(\mathsf{q}, \mathcal{B}))$, we have that $\|\mathsf{q} - \mathsf{w}\| \leq (1 + \varepsilon/4)\mathsf{d}_k(\mathsf{q}, \mathcal{B})$. Thus, $\|\mathsf{q}' - \Lambda'\|_{\oplus} = \|\mathsf{q} - \mathsf{w}\| + \mathsf{x} \leq (1 + \varepsilon/2)\mathsf{d}_k(\mathsf{q}, \mathcal{B})$. Let $\square$ be the smallest cell in which q is located. Now consider the point $\mathrm{nn}'(\square) \in \Sigma'$. Suppose it corresponds to the cluster Λ_j, i.e., $\Lambda'_j = \mathrm{nn}'(\square)$. Since $\mathrm{nn}'(\square)$ is a $(1 + \varepsilon/8)$-ANN to q among the points of Σ', $\|\mathsf{q}' - \mathrm{nn}'(\square)\|_{\oplus} \leq (1 + \varepsilon/8)\|\mathsf{q}' - \Lambda'\|_{\oplus} \leq (1 + \varepsilon/8)(1 + \varepsilon/2)\mathsf{d}_k(\mathsf{q}, \mathcal{B}) \leq (1 + \varepsilon)\mathsf{d}_k(\mathsf{q}, \mathcal{B}) \leq 2\mathsf{d}_k(\mathsf{q}, \mathcal{B}) \leq 8\xi \mathsf{d}_k(\mathsf{q}, \mathcal{B})$. It follows that, $\|\mathsf{q} - \mathsf{w}_j\| \leq 8\xi \mathsf{d}_k(\mathsf{q}, \mathcal{B})$, and $\mathsf{x}_j \leq 8\xi \mathsf{d}_k(\mathsf{q}, \mathcal{B})$. By our assumption, it must be the case that, $\mathsf{x}_j < (\varepsilon/4)\mathsf{d}_k(\mathsf{q}, \mathcal{B})$. Now, there are two cases. Suppose that, $\operatorname{diam}(\square) \leq (\varepsilon/8)\lambda^*$. Then, since we have $\lambda^* \leq \|\mathsf{q}' - \mathrm{nn}'(\square)\|_{\oplus}$. Namely, $\lambda^* \leq 2\mathsf{d}_k(\mathsf{q}, \mathcal{B})$. As such, $\operatorname{diam}(\square) \leq (\varepsilon/4)\mathsf{d}_k(\mathsf{q}, \mathcal{B})$. In this case we return $\mathrm{nn}_k(\square_{\mathsf{rep}})$ by the algorithm, but the result is correct by Lemma 6.2. On the other hand, if we return $\mathbf{b}(\square)$, it is easy to see that $\mathsf{d}(\mathsf{q}, \mathbf{b}(\square)) \leq \|\mathsf{q} - \mathsf{w}_j\| + \mathsf{x}_j \leq (1 + \varepsilon)\mathsf{d}_k(\mathsf{q}, \mathcal{B})$. Also, as $\mathbf{b}(\square)$ lies completely inside Λ_j it follows by Observation 2.1, that $\mathsf{d}(\mathsf{q}, \mathbf{b}(\square)) \geq \mathsf{d}(\mathsf{q}, \Lambda_j) \geq \|\mathsf{q} - \mathsf{w}_j\| - \mathsf{x}_j \geq (\|\mathsf{q} - \mathsf{w}_j\| + \mathsf{x}_j) - 2\mathsf{x}_j \geq \mathsf{d}_k(\mathsf{q}, \mathcal{B}) - (\varepsilon/2)\mathsf{d}_k(\mathsf{q}, \mathcal{B}) \geq (1 - \varepsilon/2)\mathsf{d}_k(\mathsf{q}, \mathcal{B})$, where the second last inequality follows by Lemma 6.1. □

6.5 The Result

Theorem 6.7 *Given a set $\mathcal{B}$ of n disjoint balls in $\mathbb{R}^d$, a number k, with $1 \leq k \leq n$, and $\varepsilon \in (0,1)$, one can preprocess $\mathcal{B}$, in $O\left(n \log n + \frac{n}{k} C_\varepsilon \log n + \frac{n}{k} C'_\varepsilon\right)$ time, where $C_\varepsilon = O\left(\varepsilon^{-d} \log \varepsilon^{-1}\right)$ and $C'_\varepsilon = O\left(\varepsilon^{-2d} \log \varepsilon^{-1}\right)$. The space used by the data structure is $O(C_\varepsilon n/k)$. Given a query point q, this data structure outputs a ball $b \in \mathcal{B}$ in $O\left(\log \frac{n}{k\varepsilon}\right)$ time, such that $(1-\varepsilon)\mathsf{d}_k(\mathrm{q}, \mathcal{B}) \leq \mathsf{d}(\mathrm{q}, b) \leq (1+\varepsilon)\mathsf{d}_k(\mathrm{q}, \mathcal{B})$.*

Proof If $k \leq 2\mathsf{c}_d$ then Theorem 4.5 provides the desired result. For $k > 2\mathsf{c}_d$, the correctness was proved in Lemma 6.6. We only need to bound the construction time and space as well as the query time. Computing the quorum clustering takes time $O(n \log n)$ by Lemma 5.6. Observe that $|\mathcal{I}| = O\left(\frac{n}{k\varepsilon^d} \log \frac{1}{\varepsilon}\right)$. From the construction of Arya and Malamatos [2], we have $|\mathcal{S}| = O\left(\frac{n}{k\varepsilon^d} \log \frac{1}{\varepsilon}\right)$ (note, that since we clip the construction to a hyperplane, we get $1/\varepsilon^d$ in the bound and not $1/\varepsilon^{d+1}$). A careful implementation of this stage takes time $O\left(n \log n + |\mathcal{W}|\left(\log n + \frac{1}{\varepsilon^d}\right)\right)$. Overlaying the two compressed quadtrees representing them takes linear time in their size, that is $O(|\mathcal{I}| + |\mathcal{S}|)$.

The most expensive step is to perform the $(1 \pm \varepsilon/4)$-approximate kth-nearest neighbor query for each cell in the resulting decomposition of $\mathcal{W}$, see Eq. 6.2 (i.e., computing $\beta_k\left(\square_{\mathsf{rep}}\right)$ for each cell $\square \in \mathcal{W}$). Using the data structure of Sect. 4 (see Theorem 4.5) each query takes $O\left(\log n + 1/\varepsilon^d\right)$ time.

$$O\left(n \log n + |\mathcal{W}|\left(\log n + \frac{1}{\varepsilon^d}\right)\right) = O\left(n \log n + \frac{n}{k\varepsilon^d} \log \frac{1}{\varepsilon} \log n + \frac{n}{k\varepsilon^{2d}} \log \frac{1}{\varepsilon}\right)$$

time, and this bounds the overall construction time.

The query algorithm is a point location query followed by an $O(1)$ time computation and takes time $O\left(\log\left(\frac{n}{k\varepsilon}\right)\right)$. □

Note that the space decomposition generated by Theorem 6.7 can be interpreted as a space decomposition of complexity $O(C_\varepsilon n/k)$, where every cell has two input balls associated with it, which are the candidates to be the desired (k, ε)-ANN. That is, Theorem 6.7 computes a $(k.\varepsilon)$-AVD of the input balls.

7 Conclusions

In this paper, we presented a generalization of the usual $(1 \pm \varepsilon)$-approximate kth-nearest neighbor problem in $\mathbb{R}^d$, where the input are balls of arbitrary radius, while the query is a point. We first presented a data structure that takes $O(n)$ space, and the query time is $O(\log n + \varepsilon^{-d})$. Here, both k and ε could be supplied at query time. Next we presented an (k, ε)-AVD taking $O(n/k)$ space. Thus showing, surprisingly, that the problem can be solved in sublinear space if k is sufficiently large.

References

1. Andoni, A., Indyk, P.: Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. Commun. ACM **51**(1), 117–122 (2008)
2. Arya, S., Malamatos, T.: Linear-size approximate Voronoi diagrams. In: Proceedings of the 13th ACM-SIAM Symposium Discrete Algorithms (SODA), pp. 147–155 (2002)
3. Arya, S., Malamatos, T., Mount, D.M.: Space–time tradeoffs for approximate spherical range counting. In: Proceedings of the 16th ACM-SIAM Symposium Discrete Algs (SODA), pp. 535–544 (2005)
4. Arya, S., Malamatos, T., Mount, D.M.: Space–time tradeoffs for approximate nearest neighbor searching. J. Assoc. Comput. Mach. **57**(1), 1–54 (2009)
5. Arya, S., Mount, D.M.: Approximate range searching. Comput. Geom. Theory Appl. **17**, 135–152 (2000)
6. Arya, S., Mount, D.M., Netanyahu, N.S., Silverman, R., Wu, A.Y.: An optimal algorithm for approximate nearest neighbor searching in fixed dimensions. J. Assoc. Comput. Mach. **45**(6), 891–923 (1998)
7. Callahan, P.B., Kosaraju, S.R.: A decomposition of multidimensional point sets with applications to k-nearest-neighbors and n-body potential fields. J. Assoc. Comput. Mach. **42**, 67–90 (1995)
8. Carmi, P., Dolev, S., Har-Peled, S., Katz, M.J., Segal, M.: Geographic quorum systems approximations. Algorithmica **41**(4), 233–244 (2005)
9. Clarkson, K.L.: Nearest-neighbor searching and metric space dimensions. In: Shakhnarovich, G., Darrell, T., Indyk, P. (eds.) Nearest-Neighbor Methods for Learning and Vision: Theory and Practice, pp. 15–59. MIT Press, Cambridge (2006)
10. de Berg, M., Haverkort, H., Thite, S., Toma, L.: Star-quadtrees and guard-quadtrees: I/O-efficient indexes for fat triangulations and low-density planar subdivisions. Comput. Geom. Theory Appl. **43**, 493–513 (2010)
11. Har-Peled, S.: A replacement for Voronoi diagrams of near linear size. In: Proceedings of the 42nd Annual IEEE Symposium Foundations of Computer Science (FOCS), pp. 94–103 (2001)
12. Har-Peled, S.: Geometric Approximation Algorithms. Mathematical Surveys and Monographs, vol. 173. American Mathematical Society, Boston (2011)
13. Har-Peled, S., Indyk, P., Motwani, R.: Approximate nearest neighbors: towards removing the curse of dimensionality. Theory Comput. **8**, 321–350 (2012). Special issue in honor of Rajeev Motwani
14. Har-Peled, S., Kumar, N.: Down the rabbit hole: robust proximity search in sublinear space. SIAM J. Comput. **43**(4), 1486–1511 (2014)
15. Har-Peled, S., Kumar, N.: Robust proximity search for balls using sublinear space. In: Proceedings of the 34th Conference Foundation of Software Technology and Theoretical Computer Science (FFSTTCS), LIPIcs, vol. 29, pp. 315–326 (2014)
16. Har-Peled, S., Kumar, N.: Approximating minimization diagrams and generalized proximity search. SIAM J. Comput. **44**(4), 944–974 (2015)
17. Indyk, P., Motwani, R.: Approximate nearest neighbors: towards removing the curse of dimensionality. In: Proceedings of the 30th Annual ACM Symposium Theory Computing (STOC), pp. 604–613 (1998)
18. Shakhnarovich, G., Darrell, T., Indyk, P.: Nearest-Neighbor Methods in Learning and Vision: Theory and Practice. Neur. Info. Proc. The MIT Press, Cambridge (2006)