Approximate Nearest Neighbor Queries Revisited

Timothy M. Chan*

Abstract

This paper proposes new methods to answer approximate nearest neighbor queries on a set of n points in d-dimensional Euclidean space. For any fixed constant d, a data structure with $O(\varepsilon^{(1-d)/2} n \log n)$ preprocessing time and $O(\varepsilon^{(1-d)/2} \log n)$ query time achieves approximation factor $1+\varepsilon$ for any given $0 < \varepsilon < 1$; a variant reduces the ε-dependence by a factor of $\varepsilon^{-1/2}$. For any arbitrary d, a data structure with $O(d^2 n \log n)$ preprocessing time and $O(d^2 \log n)$ query time achieves approximation factor $O(d^{3/2})$.

1 Introduction

Let P be a set of n point sites in d-dimensional space $\mathbb{R}^d$. In the well-known *post office problem*, we want to preprocess P into a data structure so that a site closest to a given query point q (called the *nearest neighbor* of q) can be found efficiently. Distances are measured under the Euclidean metric. The post office problem has many applications within computational geometry and from other areas such as data compression, pattern recognition, databases, and statistics.

For $d = 2$, Voronoi diagrams provide an optimal solution to the problem with $O(n \log n)$ preprocessing time, $O(n)$ space, and $O(\log n)$ query time [13]. Unfortunately, even for $d = 3$, no near-linear preprocessing method is known that achieves near-logarithmic query time; the best methods, based on ray shooting [2, 12], require $O((n/m^{1/\lceil d/2 \rceil}) \operatorname{polylog} n)$ query time for an $O(m)$-space structure ($n < m < n^{\lceil d/2 \rceil}$). To obtain better performance, a number of researchers thus turned to an approximate version of the post office problem: instead of a site with the minimum distance to the query point q, find a site s whose distance to q is within c times the minimum. We call such an s a *c-nearest neighbor of q* and the number $c > 1$ the *approximation factor*.

For any constant d, the approximate post office problem was solved optimally by Arya, *et al.* [5]: an $O(n)$-space structure called the ***balanced box-decomposition (BBD) tree*** can find $(1+\varepsilon)$-nearest neighbors in $O(\log n)$ time for any fixed $\varepsilon > 0$; this structure can be constructed in $O(n \log n)$ time. Despite its optimality, the main drawback of Arya, *et al.*'s method is the "constant" factors hidden in the big-Oh notation. These factors depend on the parameters d and ε. If d is held fixed and ε is allowed to vary, then the actual query time is $O(\varepsilon^{-d} \log n)$ in the worst case. The preprocessing does not depend on ε. Even for small values of d such as 3 and 4, obtaining a 1.001-nearest neighbor may be time-consuming with this method.

To obtain better ε-dependence in the query time, one can use a different approximation method due to Clarkson [10], which is based on an earlier randomized method of Arya and Mount [3]. With high probability Clarkson's query algorithm requires only $O(\varepsilon^{(1-d)/2} \log n)$ time, but preprocessing takes $O((\varepsilon^{1-d} \log(\rho/\varepsilon)) n^2)$ time. The parameter ρ is related to the ratio of the distance between the farthest pair of sites to the distance between the closest pair of sites. In Section 2, we obtain a strict improvement of Clarkson's result: the same $O(\varepsilon^{(1-d)/2} \log n)$ query time is obtained but with only $O(\varepsilon^{(1-d)/2} n \log n)$ preprocessing time. This method uses the BBD trees of Arya, *et al.* in a novel way. Further reduction of the ε-dependence by an $\varepsilon^{-1/2}$ factor is possible at the expense of an extra $\log n$ factor in the query time.

The above method, though efficient in low dimensions, is impractical in high dimensions, because constant factors grow exponentially when d varies. This exponential dependence in d is also inherent in Arya, *et al.*'s method and in traditional methods based on grids (bucketing), quadtrees, and k-d trees; see Arya, *et al.* [6] for an analysis of a grid method. In some applications such as vector quantization, the dimension d may actually be a function of n.

To circumvent the exponential growth problem, one

*Dept. of Math. and Computer Science, University of Miami, Coral Gables, FL 33124, tchan@cs.miami.edu

Computational Geometry 97 Nice France

can demand less and settle for a rough approximation to the post office problem. (In applications where any reasonable metric will do, a rough approximation is just as good.) Bern [7] described an interesting quadtree method to find $O(\sqrt{d})$-nearest neighbors in $O(d2^d \log n)$ time with $O(d8^d n \log n)$ preprocessing time. Alternatively, a randomized method finds $O(d^{3/2})$-nearest neighbors in $O(d \log^2 n)$ query time with high probability, using $O(d4^d n \log^2 n)$ preprocessing time. In Section 3, we improve Bern's technique by removing the exponential factors completely using a deterministic method: we can find $O(d^{3/2})$-nearest neighbors in $O(d^2 \log n)$ query time with $O(d^2 n \log n)$ preprocessing time.

2 Fine Approximation in Low Dimensions

In what follows, $\|\cdot\|$ denotes the Euclidean norm. Given $q \in \mathbb{R}^d$ and $r > 0$, $B(q,r)$ denotes the closed ball $\{p \in \mathbb{R}^d : \|p - q\| \leq r\}$. For any point $p \in \mathbb{R}^d$, we use p_i to denote the i-th coordinate of p, and we use p' and p'' to denote the projected points $(p_1, \ldots, p_{d-1})$ and $(p_1, \ldots, p_{d-2})$ respectively. In this section, we assume that the dimension d is a fixed constant and constants in big-Oh notation may depend on d.

2.1 Preliminaries on BBD trees

Let P be a set of n point sites in $\mathbb{R}^d$. For our purposes, a *BBD tree* for P is a binary tree with $O(n)$ nodes and $O(\log n)$ depth, satisfying the following properties. Each node v of the tree is associated with a cell, $cell(v) \subset \mathbb{R}^d$. Let $P(v) = P \cap cell(v)$. If *root* denotes the root of the tree, then $P(root) = P$. If $left(v)$ and $right(v)$ denote the left and right children of an internal node v, then $cell(left(v))$ and $cell(right(v))$ have disjoint interiors, covering $cell(v)$. For simplicity, we assume that no site lies on the boundary of a cell. The cells of the tree obey the following conditions:

1. each cell has constant complexity, and
2. the number of cells with disjoint interior and diameter at least s, intersecting a set of diameter r, is bounded by $\lceil 1 + Cr/s \rceil^d$ for some fixed constant C.

The first condition follows directly from Arya, *et al.*'s construction of the BBD tree [5]; they used cells that are differences of two axis-aligned boxes. The second condition is a consequence of their *packing lemma* [4, 5]. The construction time is $O(n \log n)$.

Lemma 2.1 *Given $q \in \mathbb{R}^d$ and $r > 0$, one can find $k = O(\log n)$ nodes of the BBD tree, $v_1, \ldots, v_k$, in $O(\log n)$ time, such that* (i) *each site in $B(q,r)$ lies in some $P(v_i)$, and* (ii) *each $P(v_i)$ is contained in $B(q,2r)$.*

Proof: Consider the following algorithm, based on a query algorithm by Arya and Mount for approximate range searching [4]:

Algorithm BDD-Query(v)
1. if $v = nil$ or $cell(v) \cap B(q,r) = \emptyset$ then return $\emptyset$
2. if $cell(v)$ has diameter $< r$ then return $\{v\}$
3. return BDD-Query($left(v)$) $\cup$ BDD-Query($right(v)$)

Let $\{v_1, \ldots, v_k\}$ to be the set of nodes returned by a call to BDD-Query($root$). It is easy to see that (i) holds. To show (ii), observe that for each v_i, $cell(v_i)$ intersects $B(q,r)$ and has diameter less than r; it follows that each point in $cell(v_i)$ lies in $B(q,2r)$.

It remains to bound the running time of BDD-Query(). We say that a node v is *expanded* if $cell(v)$ intersects $B(q,r)$ and the diameter of $cell(v)$ is at least r. The packing lemma (second condition) ensures that the number of expanded nodes with disjoint interior is bounded by a constant. Since cells of nodes on the same level of the BBD tree have disjoint interiors, the total number of expanded nodes is at most a constant times the depth of the tree, i.e., $O(\log n)$. This bounds the running time as well as the number k of nodes returned by BDD-Query(). □

2.2 Two BBD-based data structures

Let P be a set of n point sites in $\mathbb{R}^d$ and let $\varepsilon > 0$ be fixed. We now give a data structure for P such that given a query point $q \in \mathbb{R}^d$, we can quickly report a $(1+\varepsilon)$-nearest neighbor of q, i.e., a site s such that $\|s - q\| \leq (1+\varepsilon)\|p - q\|$ for any $p \in P$. The idea is to consider a number of coordinate systems and build BBD trees under each one. (Kapoor and Smid [11] adopted a similar strategy, using range trees, to obtain dynamic data structures for approximate nearest neighbor queries.) We note that each site lies inside a certain narrow cone under some coordinate system. We show that finding a nearest neighbor restricted to such a cone can be solved using approximate range searching via Lemma 2.1.

Lemma 2.2 *Let $\Delta_1 = \{p \in \mathbb{R}^d : \|p'\| \leq \delta p_d\}$, where $\delta = \sqrt{\varepsilon/8}$. With $O(n \log n)$ preprocessing time and space, one can answer the following query in $O(\log n)$*

time: given $q \in \mathbb{R}^d$ and $r > 0$, return a site s satisfying the inequality

$$\min\{\|s-q\|, r\} \leq (1+\varepsilon)\max\{\|p-q\|, r/2\} \quad (1)$$

for any $p \in P$ with $p - q \in \Delta_1$.

Proof: Construct a BBD tree for the $(d-1)$-dimensional set $P' = \{p' : p \in P\}$ (for simplicity, assume that no two sites have the same projection). For each node v of the tree with cell $cell(v)$, sort the point set $P(v) = \{p \in P : p' \in cell(v)\}$ according to the last coordinate and augment the node v with an array storing this sorted list. The space complexity of this augmented BBD tree is $O(n \log n)$, since the tree is of logarithmic depth. The preprocessing time remains $O(n \log n)$, including the sorting step.

Given point $q \in \mathbb{R}^d$ and $r > 0$, we can find a site s with the desired property as follows. Using algorithm BDD-Query() in Lemma 2.1, find $k = O(\log n)$ nodes of the BBD tree, $v_1, \ldots, v_k$, such that (i) each projected site in $B(q', \delta r)$ lies in some $P'(v_i)$, and (ii) each $P'(v_i)$ is contained in $B(q', 2\delta r)$. Now, define s to be a site in $P(v_1) \cup \ldots \cup P(v_k)$ that minimizes $|s_d - q_d|$.

Clearly, s can be found by performing $k = O(\log n)$ binary searches on the sorted lists at nodes $v_1, \ldots, v_k$ in $O(\log^2 n)$ time. To reduce the running time to $O(\log n)$, we employ a standard technique, attributed to Lueker and Willard [13]: for each internal node v of the BBD tree and each point $p \in P(v)$, keep pointers to the successor and predecessor of p in the sorted lists of $left(v)$ and $right(v)$; this increases preprocessing time and space by at most a constant factor. To answer a query using algorithm BDD-Query(), we only need to perform one binary search at the root; given the position of the query point q in the sorted list at node v, we can deduce the position of q in the sorted list at its children in constant time.

Let p be a site with $p - q \in \Delta_1$. It remains to show that inequality (1) holds. If $\|p-q\| > r$, then we are done. Otherwise, since $p - q \in \Delta_1$, we have $\|p'-q'\| \leq \delta|p_d - q_d| \leq \delta\|p-q\| \leq \delta r$. By (i), p must belong to some $P(v_i)$, so that $|s_d - q_d| \leq |p_d - q_d| \leq \|p-q\|$. Furthermore, by (ii), we have $\|s'-q'\| \leq 2\delta r = (\sqrt{\varepsilon/2})r$. Then

$$\begin{aligned} \|s-q\|^2 &= |s_d - q_d|^2 + \|s'-q'\|^2 \\ &\leq \|p-q\|^2 + \varepsilon r^2/2 \\ &\leq (1+2\varepsilon)\max\{\|p-q\|^2, r^2/4\}. \end{aligned}$$

Taking square roots, we get $\|s-q\| < (1+\varepsilon)\max\{\|p-q\|, r/2\}$, implying (1). □

The set Δ_1 is a (spherical) cone of angular diameter $\phi = 2\arctan\delta = \Theta(\delta)$. It is well known that the space $\mathbb{R}^d$ can be covered by $O(\phi^{1-d})$ cones of angular diameter ϕ; for example, in the plane, such a system of cones can be obtained by rotation over angles of ϕj for $j = 1, \ldots, \lceil 2\pi/\phi \rceil$ (see Yao [16]). Let $\{\Delta_1^{(j)}\}$ be a collection of $O(\phi^{1-d}) = O(\varepsilon^{(1-d)/2})$ rotated copies of Δ_1 covering $\mathbb{R}^d$. We have the following:

Theorem 2.3 *With $O(\varepsilon^{(1-d)/2} n \log n)$ preprocessing time and space, we can find a $(1+\varepsilon)$-nearest neighbor of a query point $q \in \mathbb{R}^d$ in $O(\varepsilon^{(1-d)/2} \log n)$ time.*

Proof: By changing coordinate systems, we can replace Δ_1 with the cone $\Delta_1^{(j)}$ in Lemma 2.2 for each j. In the preprocessing, we construct the data structure for each of the $O(\varepsilon^{(1-d)/2})$ cones. The total preprocessing time is $O(\varepsilon^{(1-d)/2} n \log n)$. To answer the query, we first compute a 2-nearest neighbor t of q in $O(\log n)$ time by Arya, *et al.*'s method [5]. Set $r = \|t-q\|$; observe that $\max\{\|p-q\|, r/2\} = \|p-q\|$ for any $p \in P$. Let $s^{(j)}$ be the site returned by the query algorithm of the lemma for the cone $\Delta_1^{(j)}$. Then the site in $\{s^{(j)}\} \cup \{t\}$ closest to q is a $(1+\varepsilon)$-nearest neighbor of q. The query time is therefore $O(\varepsilon^{(1-d)/2} \log n)$. □

We can slightly improve the ε-dependence in the time bounds by using a different set of cones and known techniques on planar Voronoi diagrams [13].

Lemma 2.4 *Let $\Delta_2 = \{p \in \mathbb{R}^d : \|p''\| \leq \delta p_d\}$, where $\delta = \sqrt{\varepsilon/8}$. With $O(n \log n)$ preprocessing time and space, one can answer the following query in $O(\log^2 n)$ time: given $q \in \mathbb{R}^d$ and $r > 0$, return a site s satisfying inequality (1) for any $p \in P$ with $p - q \in \Delta_2$.*

Proof: Construct a BBD tree for the $(d-2)$-dimensional set $P'' = \{p'' : p \in P\}$. For each node v of the tree with cell $cell(v)$, store the point set $P(v) = \{p \in P : p'' \in cell(v)\}$ and the Voronoi diagram of the two-dimensional point set $\{(p_{d-1}, p_d) : p \in P(v)\}$. The space complexity of this augmented BBD tree is $O(n \log n)$, since the tree is of logarithmic depth. The preprocessing time is $O(n \log^2 n)$ if we use an $O(n \log n)$-time algorithm to construct the Voronoi diagrams. We can reduce the preprocessing time to $O(n \log n)$ if we construct these Voronoi diagrams in a bottom-up fashion, since planar Voronoi diagrams can be transformed into 3-d halfspace intersections, and the intersection of two 3-d convex polyhedra can be be computed in linear time [9].

Given point $q \in \mathbb{R}^d$ and $r > 0$, we follow the proof of Lemma 2.2 and use algorithm BDD-Query() in Lemma 2.1 to find $k = O(\log n)$ nodes of the BBD tree, $v_1, \ldots, v_k$, such that (i) each projected site in $B(q'', \delta r)$

lies in some $P''(v_i)$, and (ii) each $P''(v_i)$ is contained in $B(q'', 2\delta r)$. We then define s to be a site in $P(v_1) \cup \ldots \cup P(v_k)$ that minimizes $|s_{d-1} - q_{d-1}|^2 + |s_d - q_d|^2$. This site s can be found by performing $k = O(\log n)$ point location queries on the Voronoi diagrams at nodes $v_1, \ldots, v_k$. Using an optimal method for planar point location, we can compute s in $O(\log^2 n)$ time. That s satisfies the desired property now follows as in the proof of Lemma 2.2. □

If we ignore the $(d-1)$-st coordinate, then Δ_2 is a cone of angular diameter ϕ in $\mathbb{R}^{d-1}$. We can thus cover $\mathbb{R}^d$ with only $O(\phi^{2-d}) = O(\varepsilon^{1-d/2})$ rotated copies $\{\Delta_2^{(j)}\}$ of Δ_2. Lemma 2.4 then implies the following analogue of Theorem 2.3:

Theorem 2.5 *With $O(\varepsilon^{1-d/2} n \log n)$ preprocessing time and space, we can find a $(1+\varepsilon)$-nearest neighbor of a query point $q \in \mathbb{R}^d$ in $O(\varepsilon^{1-d/2} \log^2 n)$ time.*

2.3 Applications

In certain batched applications of the post office problem, we can eliminate the extra $\log n$ factor in Theorem 2.5, by using a simple grid scheme in place of BBD trees. For example, given n red points and n blue points in $\mathbb{R}^3$, we have an $O(\varepsilon^{-1/2} n \log n)$-time algorithm for computing a pair of red and blue points whose distance is at most $1+\varepsilon$ times the closest red-blue pair. This is faster than the known exact methods [1] if $1/\varepsilon$ is within the order of $n^{2/3}$. Similar results can be obtained for an approximation of the Hausdorff distance of two (static) point sets.

3 Rough Approximation in High Dimensions

In this section, we assume a real-RAM model of computation that supports integer division and base-2 integer logarithm in unit time. Constants in big-Oh notation do not depend on the dimension d.

3.1 Preliminaries on balanced quadtrees

We define a *quadtree box* to be a set $B \subseteq [0,2)^d$ of the form

$$B = [\tfrac{a_1}{2^\ell}, \tfrac{a_1+1}{2^\ell}) \times \ldots \times [\tfrac{a_i}{2^\ell}, \tfrac{a_i+1}{2^\ell}) \times [\tfrac{a_{i+1}}{2^\ell}, \tfrac{a_{i+1}+2}{2^\ell}) \times \ldots \times [\tfrac{a_d}{2^\ell}, \tfrac{a_d+2}{2^\ell})$$

for some $\ell \in \mathbb{N}$, $i \in \{0, 1, \ldots, d-1\}$, and integers $a_1, \ldots, a_d$. The number ℓ is called the *level* of B. We say that B is of *stage* $d\ell + i$. Note that the diameter of B is less than $2^{1-\ell}\sqrt{d}$.

The following lemma, due to Arya, *et al.* [5], when applied recursively, yields a tree of logarithmic depth called the *balanced quadtree*.

Lemma 3.1 *Given an n-point set $P \subset [0,2)^d$, there exists a quadtree box B such that both point sets $P \cap B$ and $P \setminus B$ have cardinality at most $2n/3$. Furthermore, if the points in P have been sorted along each of the d coordinates, then B can be constructed in $O(dn)$ time.*

Proof: Define a sequence of quadtree boxes $B_0, B_1, \ldots$ iteratively as follows. Let $B_0 = [0,2)^d$, which is a quadtree box of stage 0. Given a quadtree box B_{k-1} of stage $k-1$ ($k \geq 1$), write B_{k-1} as the disjoint union of two quadtree boxes of stage k. One of the two boxes contains at least $|P \cap B_{k-1}|/2$ of the points of P. Let B_k be this box.

Now, consider the smallest index k with $|P \cap B_k| \leq 2n/3$; we know such a k exists since the diameters of $B_0, B_1, \ldots$ converge to 0. Then $|P \cap B_k| \geq |P \cap B_{k-1}|/2 > n/3$ and so $|P \setminus B_k| \leq 2n/3$. This proves the existence of the quadtree box B. The proof can be made constructive using integer logarithms to yield the specified time bound, as shown by Arya et al. [5] (they called their algorithm "centroid shrink"). □

For our purposes, a balanced quadtree for the point set P is a binary tree with $O(n)$ nodes and $O(\log n)$ depth, satisfying the following properties. Each node v of the tree stores a quadtree box $B(v)$ and each leaf stores a site. Let $P(v)$ be the set of sites stored in the leaves of the subtree rooted at v. If *root* denotes the root of the tree, then $P(root) = P$. If *left*(v) and *right*(v) denote the left and right children of an internal node v, then $P(left(v)) = P(v) \cap B(v)$ and $P(right(v)) = P(v) \setminus B(v)$. By Lemma 3.1, such a tree exists and can be constructed in $O(dn \log n)$ time after an initial sorting phase of $O(dn \log n)$ time.

3.2 A quadtree-based data structure

Let P be a set of n point sites in $\mathbb{R}^d$. Following an approach of Bern [7], we now show that balanced quadtrees can be used to answer approximate nearest neighbor queries on P. The idea is to consider a certain set of vectors $\{v^{(j)}\}$ and build balanced quadtrees for the translate $P + v^{(j)}$ for each j. Bern showed that using a set of 2^d vectors, one can find an $O(\sqrt{d})$-nearest neighbor of any query point. Alternatively, using a set of $O(t \log n)$ random vectors, one can find $O(d^{3/2})$-nearest neighbors with probability $1 - O(1/n^t)$. We note that a set of

$O(d)$ carefully chosen vectors actually suffices to yield approximation factor $O(d^{3/2})$.

We first need some definitions. Given $x \in \mathbb{R}$ and $r > 0$, let $x \operatorname{div} r = \lfloor x/r \rfloor$ and $x \bmod r = x - \lfloor x/r \rfloor r$. We say that two points $p, q \in \mathbb{R}^d$ *belong to the same r-grid cell* if and only if for each $i = 1, \ldots, d$, we have $p_i \operatorname{div} r = q_i \operatorname{div} r$. We say that a point p is *α-central in its r-grid cell* if and only if for each $i = 1, \ldots, d$, we have $\alpha r \le p_i \bmod r < (1-\alpha)r$, or equivalently, $(p_i + \alpha r) \bmod r \ge 2\alpha r$.

Observation 3.2 *Let $p, q \in \mathbb{R}^d$. If q is α-central in its r-grid cell and $\|p - q\| \le \alpha r$, then p and q belong to the same r-grid cell.* □

Our key lemma is the following:

Lemma 3.3 *Suppose d is even. Let*

$$v^{(j)} = \left(\tfrac{j}{d+1}, \ldots, \tfrac{j}{d+1}\right) \in \mathbb{R}^d.$$

For any point $p \in \mathbb{R}^d$ and $r = 2^{-\ell}$ ($\ell \in \mathbb{N}$), there exists $j \in \{0, 1, \ldots, d\}$ such that $p + v^{(j)}$ is $(\frac{1}{2d+2})$-central in its r-grid cell.

Proof: Suppose, on the contrary, that $p + v^{(j)}$ is not $(\frac{1}{2d+2})$-central for any $j = 0, 1, \ldots, d$. Then, for each j, there is an index $i(j) \in \{1, \ldots, d\}$ with

$$(p_{i(j)} + \tfrac{j}{d+1} + \tfrac{r}{2d+2}) \bmod r < \tfrac{r}{d+1},$$

or equivalently,

$$((d+1)2^{\ell} p_{i(j)} + 2^{\ell} j + \tfrac{1}{2}) \bmod (d+1) < 1.$$

By the pigeonhole principle, there exists two distinct indices $j, j' \in \{0, 1, \ldots, d\}$ with $i(j) = i(j')$. Letting $z = (d+1)2^{\ell} p_{i(j)} + \frac{1}{2}$, we have $(z + 2^{\ell} j) \bmod (d+1) < 1$ as well as $(z + 2^{\ell} j') \bmod (d+1) < 1$. This is possible only if $2^{\ell} j \equiv 2^{\ell} j' \pmod{(d+1)}$. Since 2^{ℓ} and $d+1$ are relatively prime, we must have $j = j'$: a contradiction! □

We now give a data structure for answering nearest neighbor queries with an $O(d^{3/2})$ approximation factor for even d. (For odd d, replace d by $d+1$.) If the L_∞ metric is used instead of the Euclidean metric, the approximation factor reduces to $O(d)$.

Theorem 3.4 *Suppose d is even. Let $c = 4d^{3/2} + 4d^{1/2} + 1$. With $O(d^2 n \log n)$ preprocessing time and $O(d^2 n)$ space, we can find a c-nearest neighbor of a query point $q \in \mathbb{R}^d$ in $O(d^2 \log n)$ time.*

Proof: We may assume that the given n-point set P is contained in $[0, 1)^d$. Let $p^* \in P$ be an exact nearest neighbor of the query point q and let $r = 2^{-\ell}$ ($\ell \in \mathbb{N}$) be such that

$$\tfrac{r}{4d+4} < \|p^* - q\| \le \tfrac{r}{2d+2}.$$

(If $\|p^* - q\| > \frac{1}{2d+2}$, then any site is a c-nearest neighbor.)

Suppose that q is $(\frac{1}{2d+2})$-central in its r-grid cell, and suppose that a balanced quadtree for P is available. Consider the following query algorithm, which runs in $O(d \log n)$ time and returns a set of $O(\log n)$ sites:

Algorithm Quad-Query(v)
1. if v is a leaf then return {the site in v}
2. if $q \in B(v)$ then
3. return Quad-Query(*left*(v))
4. else return Quad-Query(*right*(v)) $\cup$ $\{s(v)\}$
where $s(v)$ is any site from $P($*left*$(v))$

We prove that if $p^* \in P(v)$, then Quad-Query(v) contains a c-nearest neighbor of q. By line 1, the statement is true if v is a leaf. If v is an internal node, we consider four cases:

Case 1: $q \in B(v)$, $p^* \in B(v)$. Then $p^* \in P($*left*$(v)) = P(v) \cap B(v)$, and inductively, we may assume that the set Quad-Query(*left*(v)) from line 3 contains a c-nearest neighbor of q.

Case 2: $q \notin B(v)$, $p^* \notin B(v)$. Then $p^* \in P($*right*$(v)) = P(v) \setminus B(v)$, and inductively, we may assume that the set Quad-Query(*right*(v)) from line 4 contains a c-nearest neighbor of q.

Case 3: $q \in B(v)$, $p^* \notin B(v)$. Suppose that $B(v)$ is of level ℓ'. Then the two points p^* and q do not belong to the same $(2^{-\ell'})$-grid cell. However, by Observation 3.2, they belong to the same r-grid cell with $r = 2^{-\ell}$. We must have $\ell' > \ell$, implying that the diameter of $B(v)$ is less than $2^{1-\ell'}\sqrt{d} \le r\sqrt{d}$. Then any site $s \in P($*left*$(v))$ from the set returned in line 3 is a c-nearest neighbor of q:

$$\|s - q\| < r\sqrt{d} < (4d^{3/2} + 4d^{1/2})\|p^* - q\|.$$

Case 4: $q \notin B(v)$, $p^* \in B(v)$. As in Case 3, we can argue that the diameter of $B(v)$ is at most $r\sqrt{d}$. Then the site $s(v) \in P($*left*$(v))$ found in line 4 is a c-nearest neighbor of q:

$$\begin{aligned} \|s(v) - q\| &\le \|s(v) - p^*\| + \|p^* - q\| \\ &< r\sqrt{d} + \|p^* - q\| \\ &< (4d^{3/2} + 4d^{1/2} + 1)\|p^* - q\|. \end{aligned}$$

We conclude that Quad-Query(*root*) returns a set of $O(\log n)$ sites containing a c-nearest neighbor of q, under the assumption that q is $(\frac{1}{2d+2})$-central in its r-grid cell. This assumption can be removed as follows. By Lemma 3.3, we know that $q + v^{(j)}$ is $(\frac{1}{2d+2})$-central in its r-grid cell for some $j \in \{0, 1, \ldots, d\}$. During preprocessing, build a balanced quadtree for the point set $P + v^{(j)}$ for each j. The total preprocessing time for the $d+1$ trees is $O(d^2 n \log n)$. By answering a query for $q + v^{(j)}$ on the point set $P + v^{(j)}$ for each j, we obtain a set of $O(d \log n)$ sites containing a c-nearest neighbor of q, and we can return the closest point to q in this set. The query time is therefore $O(d^2 \log n)$. □

3.3 Applications

One application of our technique is the design of robust dictionaries for high-dimensional point sets. Let P be a set of n points in $\mathbb{R}^d$ such that every two points are of distance at least δ apart. In a query, we want to find a point in P that is within distance ε from a given query point. Assuming that ε is much smaller than δ, we can obtain $O(d^2 \log n)$ query and update time using a simple grid scheme based on Lemma 3.3. Note that the naive grid scheme requires either $\Omega(2^d n)$ space or $\Omega(2^d \log n)$ worst-case query time.

Another application is the construction of sparse spanner graphs. A *t-spanner* of P is a subgraph of the complete Euclidean graph of P, with the property that the shortest path length between any pair of points $p, q \in P$ is bounded by $t\|p - q\|$. The number $t > 1$ is called the *stretch factor*. We show how to construct a spanner of size $O(dn \log n)$ with an $O(d^{3/2})$ stretch factor by modifying the proof of Theorem 3.4. Previous algorithms (e.g., [8, 14, 15]) have exponential dependence of d in the running time.

Theorem 3.5 *Suppose d is even. Let $t = 8d^{3/2} + 8d^{1/2} + 1$. A t-spanner with $O(dn \log n)$ edges can be constructed in $O(d^2 n \log n)$ time.*

Proof: Given a balanced quadtree for $P \subset [0,1)^d$, consider the following procedure, which runs in $O(n \log n)$ time and returns a graph with $O(n \log n)$ edges:

Algorithm Spanner(v)

1. if v is a leaf then return $\emptyset$
2. return Spanner(*left*(v)) $\cup$ Spanner(*right*(v)) $\cup$ $\{\{s(v), p\} : p \in P(v)\}$ where $s(v)$ is any site from $P(\mathit{left}(v))$

Given a pair $p, q \in P$, let $r = 2^{-\ell}$ ($\ell \in \mathbb{N}$) be such that

$$\tfrac{r}{4d+4} < \|p - q\| \leq \tfrac{r}{2d+2}.$$

Suppose that q is $(\frac{1}{2d+2})$-central in its r-grid cell. We prove that if $p, q \in P(v)$, then the shortest path length between p and q in Spanner(v) is at most $t\|p - q\|$:

Case 1: $p, q \in B(v)$. Then $p, q \in P(\mathit{left}(v))$, and the claim follows by induction.

Case 2: $p, q \notin B(v)$. Then $p, q \in P(\mathit{right}(v))$, and the claim follows by induction.

Case 3: $p \notin B(v)$, $q \in B(v)$. As in the Case 3 of the proof of Theorem 3.4, we can show that the diameter of $B(v)$ is at most $r\sqrt{d}$. Then the path $p, s(v), q$ in Spanner(v) has length

$$\begin{aligned} &\|p - s(v)\| + \|s(v) - q\| \\ &\leq \|p - q\| + 2\|s(v) - q\| \\ &\leq \|p - q\| + 2r\sqrt{d} \\ &\leq (8d^{3/2} + 8d^{1/2} + 1)\|p - q\|. \end{aligned}$$

Case 4: $p \in B(v)$, $q \notin B(v)$. Similar to Case 3.

We conclude that Spanner(*root*) contains a path between p and q of length at most $t\|p - q\|$, under the assumption that q is $(\frac{1}{2d+2})$-central in its r-grid cell. We can remove this assumption by constructing balanced quadtrees for $P + v^{(j)}$ for $j \in \{0, \ldots, d\}$, using Lemma 3.3, and taking the union of the resulting $d+1$ graphs. □

It is well known that a minimum spanning tree of a t-spanner has weight at most t times the weight of the Euclidean minimum spanning tree. Using a Fibonacci-heap implementation of Prim's algorithm on the spanner graph, we get an $O(d^2 n \log n)$-time algorithm that approximates Euclidean minimum spanning trees to within a factor of $O(d^{3/2})$. By well-known reductions, similar results follow for Euclidean traveling salesman tours and minimum Steiner trees.

For the application to closest pairs, we can reduce the $O(d^2)$ factor in the running time to $O(d)$ using a different algorithm. Details can be found in the full paper.

4 Conclusions

We have examined ways of reducing "constant factors" in previous approaches to approximate nearest neighbor queries on a set of n points. For any fixed dimension d and approximation factor $1 + \varepsilon$, we obtain constants that are $O(\varepsilon^{(1-d)/2})$ or $O(\varepsilon^{1-d/2})$. For any dimension and a sufficiently large approximation factor, we obtain constants that are polynomial in the dimension. The

time and space bounds of our methods are optimal or near-optimal in terms of n.

One open problem for low dimensions is to reduce the ϵ-dependence even further, if possible, while keeping $O(n \operatorname{polylog} n)$ space and $O(\operatorname{polylog} n)$ query time. Reducing the space complexity to linear in Theorems 2.3 and 2.5 would be interesting. Another problem is to derive our high-dimensional results in Section 3 using only algebraic operations.

References

[1] P. K. Agarwal, H. Edelsbrunner, O. Schwarzkopf, and E. Welzl. Euclidean minimum spanning trees and bichromatic closest pairs. *Discrete Comput. Geom.*, 6:407–422, 1991.

[2] P. K. Agarwal and J. Matoušek. Ray shooting and parametric search. *SIAM J. Comput.*, 22:764–806, 1993.

[3] S. Arya and D. M. Mount. Approximate nearest neighbor queries in fixed dimensions. In *Proc. 4th ACM-SIAM Sympos. Discrete Algorithms*, pages 271–280, 1993.

[4] S. Arya and D. M. Mount. Approximate range searching. In *Proc. 11th ACM Sympos. Comput. Geom.*, pages 172–181, 1995.

[5] S. Arya, D. M. Mount, N. S. Netanyahu, R. Silverman, and A. Wu. An optimal algorithm for approximate nearest neighbor searching. In *Proc. 5th ACM-SIAM Sympos. Discrete Algorithms*, pages 573–582, 1994.

[6] S. Arya, D. M. Mount, and O. Narayan. Accounting for boundary effects in nearest neighbor searching. *Discrete Comput. Geom.*, 16:155–176, 1996.

[7] M. Bern. Approximate closest-point queries in high dimensions. *Inform. Process. Lett.*, 45:95–99, 1993.

[8] P. B. Callahan and S. R. Kosaraju. Faster algorithms for some geometric graph problems in higher dimensions. In *Proc. 4th ACM-SIAM Sympos. Discrete Algorithms*, pages 291–300, 1993.

[9] B. Chazelle. An optimal algorithm for intersecting three-dimensional convex polyhedra. *SIAM J. Comput.*, 21:671–696, 1992.

[10] K. L. Clarkson. An algorithm for approximate closest-point queries. In *Proc. 10th ACM Sympos. Comput. Geom.*, pages 160–164, 1994.

[11] S. Kapoor and M. Smid. New techniques for exact and approximate dynamic closest-point problems. *SIAM J. Comput.*, 25:775–796, 1996.

[12] J. Matoušek and O. Schwarzkopf. On ray shooting in convex polytopes. *Discrete Comput. Geom.*, 10:215–232, 1993.

[13] F. P. Preparata and M. I. Shamos. *Computational Geometry: An Introduction.* Springer-Verlag, New York, 1985.

[14] J. S. Salowe. Construction of multidimensional spanner graphs, with applications to minimum spanning trees. In *Proc. 7th ACM Sympos. Comput. Geom.*, pages 256–261, 1991.

[15] P. M. Vaidya. A sparse graph almost as good as the complete graph on points in k dimensions. *Discrete Comput. Geom.*, 6:369–381, 1991.

[16] A. C. Yao. On constructing minimum spanning trees in k-dimensional spaces and related problems. *SIAM J. Comput.*, 11:721–736, 1982.