Spectral Approaches to Nearest Neighbor Search

Amirali Abdullah
University of Utah
amirali@cs.utah.edu

Alexandr Andoni
Microsoft Research
andoni@microsoft.com

Ravindran Kannan
Microsoft Research
kannan@microsoft.com

Robert Krauthgamer
Weizmann Institute
robert.krauthgamer@weizmann.ac.il

***Abstract*—We study spectral algorithms for the high-dimensional Nearest Neighbor Search problem (NNS). In particular, we consider a semi-random setting where a dataset is chosen arbitrarily from an unknown subspace of low dimension, and then perturbed by full-dimensional Gaussian noise. We design spectral NNS algorithms whose query time depends polynomially on the dimension and logarithmically on the size of the point set. These spectral algorithms use a repeated computation of the top PCA vector/subspace, and are effective even when the random-noise magnitude is *much larger* than the interpoint distances. Our motivation is that in practice, a number of spectral NNS algorithms outperform the random-projection methods that seem otherwise theoretically optimal on worst-case datasets. In this paper we aim to provide theoretical justification for this disparity.**

The full version of this extended abstract is available on arXiv.

I. Introduction

A fundamental tool in high-dimensional computational geometry is the random projection method. Most notably, the Johnson-Lindenstrass Lemma [27] says that projecting onto a uniformly random k-dimensional subspace of $\mathbb{R}^d$ approximately preserves the distance between any (fixed) points $x, y \in \mathbb{R}^d$ (up to scaling), except with probability exponentially small in k. This turns out to be a very powerful approach as it effectively reduces the dimension from d to a usually much smaller k via a computationally cheap projection, and as such has had a tremendous impact on algorithmic questions in high-dimensional geometry.

A classical application of random projections is to the high-dimensional Nearest Neighbor Search (NNS) problem. Here we are given a dataset of n points from $\mathbb{R}^d$, which we preprocess to subsequently find, given a query point $q \in \mathbb{R}^d$, its closest point from the dataset. It is now well-known that exact NNS admits algorithms whose running times have good dependence on n, but exponential dependence on the dimension d [34], [11]; however these are unsatisfactory for moderately large d.

To deal with this "curse of dimensionality", researchers have studied algorithms for *approximate* NNS, and in the high-dimensional regime, most (provable) algorithms rely heavily on the random projection method., where, one projects datapoints into a random lowe dimensional subspace and uses the Johnson-Lindenstruass theorem [27] to argue that distances are approximately preserved in the projection.

All known variants of the successful Locality Sensitive Hashing approach for Euclidean spaces, including [23], [16], [3], [4], involve random projections.[1] For example, the space partitioning algorithm of [16] can be viewed as follows. Project the dataset onto a random k-dimensional subspace, and impose a randomly-shifted uniform grid. For a query point q, just look up the points in the grid cell where q falls into. (see also [38]).

Random projections are one of the two principal methods of reducing the dimension of data. The other method is a "data-aware" projection given by Principal Component Analysis (PCA) based on Singular Value Decomposition (SVD). While random projections work well and have provable guarantees, it is natural to ask whether one can provably improve the performance by using data-aware projections such as PCA. This question is all the more relevant since in practice, PCA-based dimensionality reduction methods — such as PCA tree [43], [33], [46] and its variants (called randomized kd-tree) [42], [36], spectral hashing [48], semantic hashing [41], and WTA hashing [50] — have been quite successful, often outperforming algorithms based on vanilla random projections. In contrast to the random projection method, none of these methods have rigorous proofs of correctness or performance guarantees.

Bridging the gap between random projections and data-aware projections has been recognized as a big open question in Massive Data Analysis, see e.g. a recent National Research Council report [37, Section 5]. The challenge here is that random projections are themselves (theoretically) optimal not only for dimensionality reduction [2], [26], but also for some of its algorithmic applications [25], [49], including NNS in certain settings [5]. We are aware of only one work addressing this question: data-dependent LSH, which was introduced recently [4], provably improves the query time polynomially. However, their *space partitions* are very different from the aforementioned practical heuristics (e.g., they are not spectral-based), and do not explain why data-aware *projections* help at all.

In this work, we address this gap by studying data-aware projections for the nearest neighbor search problem. As argued above, for worst-case inputs we are unlikely to beat the performance of random projections. Instead, we consider

[1]While [23] is designed for the Hamming space, their algorithm is extended to the Euclidean space by an embedding of ℓ_2 into ℓ_1, which itself uses random projections [28].

a semi-random model, where the dataset is formed by first taking an arbitrary (worst-case) set of n points in a k-dimensional subspace of $\mathbb{R}^d$, and then perturbing each point by adding to it Gaussian noise $N_d(0, \sigma^2 I_d)$. The query point is selected using a similar process. Our algorithms are able to find the query's nearest neighbor as long as there is a small gap (1 vs $1+\epsilon$) in the distance to the nearest neighbor versus other points in the *unperturbed* space — this is a much weaker assumption than assuming the same for the perturbed points.

Most importantly, our results hold even when the noise magnitude is *much larger* than the distance to the nearest neighbor. The noise vector has length (about) $\sigma\sqrt{d}$, and so for $\sigma \gg 1/\sqrt{d}$, the noise magnitude exceeds the original distances. In such a case, a random projection to a smaller dimension will not work — the error due to the projection will lose all the information on the nearest neighbor. In contrast, we show that data-aware projections provably guarantee good performance in our model provided $\sigma \in \Omega^*(1/\sqrt[4]{d})$, alongwith some additional polynomial dependence on k and $\log n$ which we assume much smaller than d.

A. Algorithmic Results

We propose two spectral algorithms for nearest neighbor search, which achieve essentially the same performance as NNS algorithms in k and $O(k \log k)$-dimensional space, respectively. These spectral algorithms rely on computing a PCA subspace or vector respectively of the datapoints. Precise results, are stated in Theorems V.1 and VI.1. We focus here on a *qualitative* description.

The first algorithm performs *iterative PCA*. Namely, it employs PCA to extract a subspace of dimension (at most) k, identifies the datapoints captured well by this subspace, and then repeats iteratively on the remaining points. The algorithm performs at most $O(\sqrt{d \log n})$ PCAs in total, and effectively reduces the original NNS problem to $O(\sqrt{d \log n})$ instances of NNS in k dimensions. Each of these NNS instances can be solved by any standard low-dimensional $(1+\epsilon)$-approximate NNS, such as [12], [9], [8], [19], [7], [13], which can give, say, query time $(1/\epsilon)^{O(k)} \log^2 n$. See Section V, and the crucial technical tool it uses in Section IV. As a warmup, we initially introduce a simplified version of the algorithm for a (much) simpler model in Section III.

The second algorithm is a variant of the aforementioned PCA tree, and constructs a tree that represents a recursive space partition. Each tree node finds the top PCA direction, and partitions the dataset into slabs perpendicular to this direction. We recurse on each slab until the tree reaches depth $2k$. The query algorithm searches the tree by following a small number of children (slabs) at each node. This algorithm also requires an additional preprocessing step that ensures that the dataset is not overly "clumped". The overall query time is $(k/\epsilon)^{O(k)} \cdot d^2$. See Section VI.

While the first algorithm is randomized, the second algorithm is deterministic and its failure probability comes only from the semi-random model (randomness in the instance).

B. Related Work

There has been work on understanding how various tree data structures adapt to a *low-dimensional* point set, including [15], [46]. For example, [46] show that PCA trees adapt to a form of "local covariance dimension", a spectrally-inspired notion of dimension, in the sense that a PCA tree halves the "diameter" of the point set after a number of levels dependent on this dimension notion (as opposed to the ambient dimension d). Our work differs in a few respects. First, our data sets do not have a small local covariance dimension. Second, our algorithms have guarantees of performance and correctness for NNS for a worst-case query point (e.g., the true nearest neighbor can be any dataset point). In contrast, [46] prove a guarantee on diameter progress, which does not necessarily imply performance guarantees for NNS, and, in fact, may only hold for average query point (e.g., when the true nearest neighbor is random). Indeed, for algorithms considered in [46], it is easy to exhibit cases where NNS fails.[2]

For our model, it is tempting to design NNS algorithms that find the original k-dimensional subspace and thus "de-noise" the dataset by projecting the data onto it. This approach would require us to solve the ℓ_∞-regression problem with high precision.[3] Unfortunately, this problem is NP-hard in general [18], and the known algorithms are quite expensive, unless k is constant. Har-Peled and Varadarajan [22] present an algorithm for ℓ_∞-regression achieving $(1+\epsilon)$-approximation in time $O(nde^{e^{O(k^2)}\epsilon^{-2k-3}})$, which may be prohibitive when $k \geq \Omega(\sqrt{\log \log n})$. In fact, there is a constant $\delta > 0$ such that it is Quasi-NP-hard, i.e., implies $\mathrm{NP} \subseteq \mathrm{DTIME}(2^{(\log n)^{O(1)}})$, to even find a $(\log n)^\delta$ approximation to the best fitting k-subspace when $k \geq d^\epsilon$ for any fixed $\epsilon > 0$ [44].

We also note that the problem of finding the underlying k-dimensional space is somewhat reminiscent of the learning mixtures of Gaussians problem [14]; see also [6], [45], [35] and references therein. In the latter problem, the input is n samples generated from a mixture of k Gaussian distributions with unknown mean (called centers), and the goal is to identify these centers (the means of the k Gaussians). Our setting can be viewed as having n centers in a k-dimensional subspace of $\mathbb{R}^d$, and the input contains exactly one sample from each of the n centers. Methods known from learning mixtures of Gaussians rely crucially on obtaining multiple

[2]For example, if we consider the top PCA direction of a dataset and the median threshold, we can plant a query–near-neighbor pair on the two sides of the partition. Then, this pair, which won't affect top PCA direction much, will be separated in the PCA tree right from the beginning.

[3]As we explain later, related problems, such as ℓ_2-regression would not be sufficient.

samples from the same center (Gaussian), and thus do not seem applicable here.

Finally, the problem of nearest neighbor search in Euclidean settings with "effective low-dimension" has received a lot of attention in the literature, including [29], [30], [21], [10], [13], [24] among many others. Particularly related is also the work [20], where the authors consider the case when the dataset is high-dimensional, but the query comes from a (predetermined) low-dimensional subspace. These results do not seem readily applicable in our setting because our dataset is really high-dimensional, say in the sense that the doubling dimension is $\Omega(d)$.

C. Techniques and Ideas

We now discuss the main technical ideas behind our two algorithms. The first intuitive line of attack is to compute a k-dimensional PCA space of the datapoints, project them into this k-dimensional subspace, and solve the NNS problem there. This approach fails because the noise is too large, and PCA only optimizes *the sum of distances* (i.e., an average quantity, as opposed to the "worst case" quantity). In particular, suppose most of the points lie along some direction $\vec{u}$ and only a few points lie in the remaining dimensions of our original subspace U (which we call sparse directions). Then, the k-PCA of the dataset will return a top singular vector close to $\vec{u}$, but the remaining singular vectors will be mostly determined by random noise. In particular, the points with high component along sparse directions may be very far away from the subspace returned by our PCA, and hence "poorly-captured" by the PCA space. Then, the NNS data structure on the projected points will fail for some query points. If the difference between a query point q and its nearest neighbor p^* is along $\vec{u}$, whereas the difference between q and a poorly-captured point p' is along the sparse directions, then any such p' will cluster around q in the k-PCA space, at a distance much closer than $\|q - p^*\|$.

Our first algorithm instead runs k-PCAs *iteratively*, while pruning away points "well-captured" by the PCA (i.e., close to the PCA subspace). In particular, this allows us to discover the points in sparse directions in *later* iterations. To ensure correctness of nearest neighbor queries in presence of large noise in a given iteration, we need to take only those singular vectors that are actually close to U (which may be much fewer than k). To detect such directions, we consider only singular vectors corresponding to singular values exceeding some threshold value. Proving that such singular vectors are "good" is non-trivial, starting even with the definition of what it means to be "close" for two subspaces that have different dimensions. For this purpose, we employ the so-called $\sin\theta$ machinery, which was developed by Davis and Kahan [17] and by Wedin [47], to bound the perturbations of the singular *vectors* of a matrix in presence of noise. Notice the difference from the more usual theory of perturbations of the singular *values*. For example, in contrast to singular values, it is not true that the top singular vector is "stable" when we perturb a given matrix.

The actual algorithm has one more important aspect: in each iteration, the PCA space is computed on a *sample* of the (surviving) data points. This modification allows us to control spurious conditioning induced by earlier iterations. In particular, if instead we compute the PCA of the full data, once we argue that a vector $\tilde{p}$ "behaves nicely" in one iteration, we might effectively condition on the direction of its noise, potentially jeopardizing noise concentration bounds in later iterations. (While we do not know if sampling is really necessary for the algorithm to work, we note that practically it is a very reasonable idea to speed up preprocessing nonetheless.) This concludes the description of our first algorithm.

The second algorithm is based on the PCA-tree, which partitions the space recursively, according to the top PCA direction. This can be seen as another (extreme) form of "iterative PCA". At each node, the algorithm extracts the top PCA direction, which is always guaranteed to be close to original space U. We then partition the dataset into a few slabs along this direction, and recurse on the datapoints captured by each slab separately. The performance of the PCA tree depends exponentially on its depth, hence the crux of the argument is to bound the depth. While it seems plausibly easy to show that a partitioning direction should never be repeated, this would give too loose of a bound, as there could be a total of $\approx \exp(k)$ essentially distinct directions in a k-dimensional space. Instead, we perform a mild form of orthonormalization as we progress down the tree, to ensure only $O(k)$ directions are used in total. In the end, the query time is roughly $k^{O(k)}$, i.e., equivalent to a NNS in an $O(k \log k)$-dimensional space.

The algorithm also requires two further ideas. First, one has to use *centered* PCA, i.e., PCA on the data centered at zero: otherwise, every small error in PCA direction may move points a lot for subsequent iterations, misleading a non-centered PCA. Second, from time to time, we need to do "de-clumping" of the data, which essentially means that the data is sparsified if the points are too close to each other. This operation also appears necessary; otherwise, a cluster of points that are close in the original space, might mislead the PCA due to their noise components. Furthermore, in contrast to the first algorithm, we cannot afford $\approx d$ iterations to eliminate "bad" directions one by one.

II. The Model

We assume throughout the dataset is generated as follows.[4] Let U be a k-dimensional subspace of $\mathbb{R}^d$. Let $P = \{p_1, \ldots, p_n\}$ be a set of n points all living in U and having at least unit norm, and let $q \in U$ be a query

[4]An exception is the warm-up Section III, where the noise is small adversarial.

point. We assume that $d = \Omega(\log n)$. The point set P is perturbed to create $\tilde{P} = \{\tilde{p}_1, \ldots, \tilde{p}_n\}$ by adding to each point independent Gaussian noise, and the query point q is perturbed similarly. Formally,

$$\begin{aligned} \tilde{p}_i &= p_i + t_i \text{ where } t_i \sim N_d(0, \sigma I_d), \qquad \forall p_i \in P, \\ \tilde{q} &= q + t_q \text{ where } t_q \sim N_d(0, \sigma I_d). \end{aligned} \tag{II.1}$$

Let us denote the nearest neighbor to q in P by p^* and let $\tilde{p}^*$ be its perturbed version. We shall actually consider the *near-neighbor* problem, by assuming that in the unperturbed space, there is one point $p^* \in P$ within distance 1 from the query, and all other points are at distance at least $1+\epsilon$ from the query, for some known $0 < \epsilon < 1$. Formally,

$$\begin{aligned} &\exists p^* \in P \text{ such that } \|q - p^*\| \leq 1 \text{ and} \\ &\forall p \in P \setminus \{p^*\}, \|q - p\| \geq 1 + \epsilon. \end{aligned} \tag{II.2}$$

We note that even if there is more than one such point p^* so that $\|q - p^*\| \leq 1$, our algorithms will return one of these close p^* correctly. Also our analysis in Section V extends trivially to show that for any x such that $x \geq 1$ and $\|q - p^*\| = x$, our first algorithm the iterative PCA actually returns a $(1+\epsilon)$-approximate nearest neighbor to q. We omit the details of this extended case for ease of exposition.

A. Preliminary Observations

For the problem to be interesting, we need that the perturbation does not change the nearest neighbor, i.e., $\tilde{p}^*$ remains the closest point to $\tilde{q}$. We indeed show this is the case as long as $\sigma \ll \epsilon/\sqrt[4]{d \log n}$. Notice that the total noise magnitude is roughly $\sigma\sqrt{d}$, which can be much larger than 1 (the original distance to the nearest neighbor). Hence after the noise is added, the ratio of the distance to the nearest neighbor and to other (nearby) points becomes very close to 1. This is the main difficulty of the problem, as, for example, it is the case where random dimensionality reduction would lose nearest neighbor information. We recommend to keep in mind the following parameter settings: $k = 20$ and $\epsilon = 0.1$ are constants, while $d = \log^3 n$ and $\sigma = \Theta(1/\log n)$ depend asymptotically on $n = |P|$. In this case, for example, our algorithms actually withstand noise of magnitude $\Theta(\sqrt{\log n}) \gg 1$.

Here and in the rest of the paper, we will repeatedly employ concentration bounds for Gaussian noise, expressed as tail inequalities on χ^2 distribution. The following are standard concentration of measure results, and we defer the routine proofs to the full version.

Theorem II.1. *([32]) Let $X \sim \chi^2_d$. For all $x \geq 0$,*

$$\Pr\left[X \geq d\left(1 + 2\sqrt{\tfrac{x}{d}}\right) + x\right] \leq e^{-x}, \quad \textit{and}$$
$$\Pr\left[X \leq d\left(1 - 2\sqrt{\tfrac{x}{d}}\right)\right] \leq e^{-x}.$$

Corollary II.2. *For $n \geq 1$, let $X \sim \chi^2_d$. Then* $\Pr[|X - d| \geq d + 4\sqrt{d \log n} + 4 \log n] \leq \frac{2}{n^4}$.

We also obtain that after the perturbation of P, q, the nearest neighbor of $\tilde{q}$ will remain $\tilde{p}^*$, w.h.p.

Lemma II.3. *Consider the above model II.1-II.2 for $n > 1$, $\epsilon \in (0,1)$, dimensions $k < d = \Omega(\log n)$, and noise standard deviation $\sigma \leq c\epsilon/\sqrt[4]{d \log n}$, where $c > 0$ is a sufficiently small constant. Then w.h.p. the nearest neighbor of $\tilde{q}$ (in $\tilde{P}$) is $\tilde{p}^*$.*

The problem remains essentially the same if we assume the noise has no component in the space U. Indeed, we can absorb the noise inside U ito the "original" points (P and q). With high probability, this changes the distance from q to every point in P by at most $O(\sigma\sqrt{k \log n}) \ll \epsilon$. Hence, in the rest of the article, we will assume the noise is perpendicular to U.

III. Warmup: Iterative PCA under Small Adversarial Noise

To illustrate the basic ideas in our "iterative PCA" approach, we first study it in an alternative, simpler model that differs from Section II in that the noise is *adversarial* but of *small magnitude*. The complete "iterative PCA" algorithm for the model from Section II will appear in Section V.

In the bounded noise model, for fixed $\epsilon \in (0,1)$, we start with an n-point dataset P and a point q, both lying in a k-dimensional space $U \subset \mathbb{R}^d$, such that

$$\begin{aligned} &\exists p^* \in P \text{ such that } \|q - p^*\| \leq 1 \text{ and} \\ &\forall p \in P \setminus \{p^*\},\ \|q - p\| \geq 1 + \epsilon \end{aligned}$$

The set $\tilde{P}$ consists of points $\tilde{p}_i = p_i + t_i$ for all $p_i \in P$, where the noise t_i is arbitrary, but satisfies $\|t_i\| \leq \epsilon/16$ for all i. Similarly, $\tilde{q} = q + t_q$ with $\|t_q\| \leq \epsilon/16$.

Theorem III.1. *Suppose there is a $(1 + \epsilon/4)$-approximate NNS data structure for n points in a k-dimensional Euclidean space with query time F_{query}, space F_{space}, and preprocessing time F_{prep}. Then for the above adversarial-noise model, there is a data structure that preprocesses $\tilde{P}$, and on query $\tilde{q}$ returns $\tilde{p}^*$. This data structure has query time $O((dk + F_{\text{query}}) \log n)$, space $O(F_{\text{space}})$, and preprocessing time $O(n + d^3 + F_{\text{prep}})$.*

First we show that the nearest neighbor "remains" p^* even after the perturbations (similarly to Lemma II.3. Let $\alpha = \epsilon/16$.

Claim III.2. *The nearest neighbor of $\tilde{q}$ in $\tilde{P}$ is $\tilde{p}^*$.*

Proof: For all i, we have $\|\tilde{p}_i - p_i\| \leq \|t_i\| \leq \alpha$, hence by the triangle inequality, $\|\tilde{q} - \tilde{p}^*\| \leq \|\tilde{q} - q\| + \|q - p^*\| + \|p^* - \tilde{p}^*\| \leq \|q - p^*\| + 2\alpha$. For all $p \neq p^*$, a similar argument gives $\|\tilde{q} - \tilde{p}\| \geq \|q - p^*\| + \epsilon - 2\alpha$. ■

We now describe the algorithm used to prove Theorem III.1. Our algorithm first finds a small collection $\mathcal{U}$ of k-dimensional subspaces, such that every point of $\tilde{P}$ is "captured well" by at least one subspace in $\mathcal{U}$. We find this collection $\mathcal{U}$ by iteratively applying PCA, as follows (see Algorithm III.1). First compute the top (principal) k-dimensional subspace of $\tilde{P}$. It "captures" all points $\tilde{p} \in \tilde{P}$ within distance $\sqrt{2}\alpha$ from the subspace. Then we repeat on the remaining non-captured points, if any are left. In what follows, let $p_{\tilde{U}}$ denote the projection of a point p onto $\tilde{U}$, and define the distance between a point x and a set (possibly a subspace) S as $d(x, S) = \inf_{y \in S} \|x - y\|$.

Algorithm III.1 Iterately locate subspaces

```
$j \leftarrow 0$; $\tilde{P}_0 \leftarrow \tilde{P}$
**while** $\tilde{P}_j \neq \emptyset$ **do**
  $\tilde{U}_j \leftarrow$ the $k$-dimensional PCA subspace of $\tilde{P}_j$
  $M_j \leftarrow \{\tilde{p} \in \tilde{P}_j : d(\tilde{p}, \tilde{U}_j) \leq \sqrt{2}\alpha\}$
  $\tilde{P}_{j+1} \leftarrow \tilde{P}_j \setminus M_j$
  $j \leftarrow j + 1$
**end while**
**return** $\tilde{\mathcal{U}} = \{\tilde{U}_0, \ldots, \tilde{U}_{j-1}\}$ and the associated point sets
$\{M_0, M_1, \ldots, M_{j-1}\}$.
```

The remainder of the preprocessing algorithm just constructs for each subspace $\tilde{U} \in \mathcal{U}$ a data structure for k-dimensional NNS, whose dataset is the points captured by $\tilde{U}$ projected onto this subspace $\tilde{U}$ (treating $\tilde{U}$ as a copy of $\mathbb{R}^k$). Overall, the preprocessing phase comprises of $O(\log n)$ PCA computations and constructing $O(\log n)$ data structures for a k-dimensional NNS.

The query procedure works as follows. Given a query point $\tilde{q}$, project $\tilde{q}$ onto each $\tilde{U} \in \mathcal{U}$ to obtain $\tilde{q}_{\tilde{U}}$, and find in the data structure corresponding to this $\tilde{U}$ a $(1 + \epsilon/4)$-approximate nearest neighbor point $\tilde{p}_{\tilde{U}}$ to $\tilde{q}_{\tilde{U}}$. Then compute the distance between $\tilde{q}$ and each $\tilde{p}$ (original points corresponding to $\tilde{p}_{\tilde{U}}$), and report the the closest one to $\tilde{q}$.

We now proceed to analyze the algorithm.

Claim III.3. *Algorithm III.1 terminates within $O(\log n)$ iterations.*

Proof: Let U be the PCA subspace of P and let $\tilde{U}$ be the PCA subspace of $\tilde{P}$. Since $\tilde{U}$ minimizes (among all k-dimensional subspaces) the sum of squared distances from all $\tilde{p} \in \tilde{P}$ to $\tilde{U}$,

$$\sum_{\tilde{p} \in \tilde{P}} d(\tilde{p}, \tilde{U})^2 \leq \sum_{\tilde{p} \in \tilde{P}} d(\tilde{p}, U)^2 \leq \sum_{\tilde{p} \in \tilde{P}} \|\tilde{p} - p\|^2 \leq \alpha^2 n.$$

Hence, at most half of the points in $\tilde{P}$ may have distance to $\tilde{U}$ which is greater than $\sqrt{2}\alpha$. The current set M will capture the other (at least a half fraction) points, and the algorithm then proceeds on the remaining set. Each subsequent iteration thus decreases the number of points by a constant factor. After $O(\log n)$ iterations all points of $\tilde{P}$ must be captured. ■

Claim III.4. *The data structure for the subspace $\tilde{U}$ that captures $\tilde{p}^*$ always reports this point as the $(1 + \epsilon/4)$-approximate nearest neighbor of $\tilde{q}$ (in $\tilde{U}$).*

The proof is deferred to the full version, and follows from careful application of the triangle inequality and Pythagoras' Theorem, using the bounds on the noise-magnitude bound $\alpha < \epsilon/16$ and on the distance to the subspace $\sqrt{2}\alpha$.

We can now complete the proof of Theorem III.1. By Claim III.4, $\tilde{p}^*$ is always reported by the k-dimensional data structure it is assigned to. But this is the closest point overall, by Claim III.2, and thus our algorithm eventually reports this point $\tilde{p}^*$, which proves the correctness part of Theorem III.1. To argue the time and space guarantees, we just note that computing one PCA on n points takes time $O(n + d^3)$, and there are in total $O(\log n)$ PCAs to compute, and obviously also k-dimensional NNS data structures to query against.

IV. Stability of a Top PCA Subspace

Before continuing to the full iterative-PCA algorithm, we need to address the challenge of controlling the stability of the PCA subspace under random noise. In particular, we will need to show that the PCA subspace $\tilde{U}$ computed from the noisy dataset $\tilde{P}$ is "close" to the original subspace U.

Notation: Throughout, $s_j(M)$ denotes the j-th largest singular value of a real matrix M, and $\|M\| = s_1(M)$ denotes its spectral norm, while $\|M\|_F$ denotes the Frobenius norm of M. All vector norms, i.e. $\|v\|$ for $v \in \mathbb{R}^d$, refer to the ℓ_2-norm.

A. $\sin -\theta$ *Distance between subspaces.*

The $\sin\theta$ *distance* between two subspaces B and A of $\mathbb{R}^d$ is defined as

$$\sin\theta(B, A) = \max_{x \in B, \|x\|=1} \min_{y \in A} \|x - y\|.$$

Observe that the minimum here is just the distance to a subspace $\operatorname{dist}(x, A)$, and it is attained by orthogonal projection. Thus, for all $x' \in B$ (not necessarily of unit length) $\operatorname{dist}(x', A) = \|x'\| \cdot \operatorname{dist}\left(\frac{x'}{\|x'\|}, A\right) \leq \|x'\| \cdot \sin\theta(B, A)$.

Let $X \in \mathbb{R}^{n \times d}$ be the matrix corresponding to our original point set P (of size $n \geq d$) lying in a subspace U of dimension $k \leq d$. Let $T \in \mathbb{R}^{n \times d}$ be a perturbation matrix (noise), and then $\tilde{X} = X + T$ corresponds to our perturbed point set $\tilde{P}$. Our next theorem uses $\|T\|$ directly without assuming anything about its entries, although in our context where the entries of T are drawn from independent Gaussians of magnitude σ, Theorem V.2 implies that w.h.p. $\|T\| \leq O(\sigma\sqrt{n + d})$. In fact, if the matrix T is random, then m (and possibly also γ) should be interpreted as random variables that depend on T.

Theorem IV.1. *Let $\tilde{X} = X + T$ be defined as above, and fix a threshold $\gamma_1 > 0$. If $m \le k$ is such that at least m singular values of $\tilde{X}$ are at least γ_1, then*

$$\sin\theta(SR_m(\tilde{X}), SR_k(X)) \le \frac{\|T\|}{\gamma_1},$$

where $SR_\ell(M)$ denotes the span of the top ℓ right-singular vectors of a matrix M.

We defer the proof of Theorem IV.1 to the full version, but note it is a simple corollary of Wedin's sin-θ Theorem [47].

V. Iterative PCA Algorithm

We now present the iterative PCA algorithm, that solves the NNS problem for the semi-random model from Section II. In particular, the underlying pointset lives in a k-dimensional space, but each point is also added a Gaussian noise $N_d(0, \sigma^2 I_d)$, which has norm potentially much larger than the distance between a query and it nearest neighbor. The algorithm reduces the setting to a classical k-dimensional NNS problem.

Theorem V.1. *Suppose there is a $(1 + \epsilon/8)$-approximate NNS data structure for n points in a k-dimensional space with query time F_{query}, space F_{space}, and preprocessing time F_{prep}. Assume the Gaussian-noise model* (II.1)-(II.2)*, with $\sigma(k^{1.5}\sqrt{\log n} + \sqrt[4]{k^3 d \log n}) < c\epsilon$ for sufficiently small constant $c > 0$.*

Then there is a data structure that preprocesses $\tilde{P}$, and on query $\tilde{q}$ returns $\tilde{p}^*$ with high probability. This data structure has query time $O((dk + F_{\text{query}})\sqrt{d \log n} + d^{O(1)})$, uses space $O(F_{\text{space}}\sqrt{d \log n} + d^{O(1)})$, and preprocessing time $O((nd^2 + d^3 + F_{\text{prep}})\sqrt{d \log n})$.

A. Algorithm Description

The iterative-PCA algorithm computes a collection $\mathcal{U}$ of $O(\sqrt{d \log n})$ subspaces, such that every point in the perturbed dataset $\tilde{P}$ is within squared distance $\Psi = d\sigma^2 + 0.001\epsilon^2$ of some subspace in the collection $\mathcal{U}$. For each such subspace $\tilde{U}_j^s \in \mathcal{U}$, we project onto $\tilde{U}_j^s$ the points captured by this subspace $\tilde{U}_j^s$, and construct on the resulting pointset a k-dimensional NNS data structure. We consider only singular vectors corresponding to sufficiently large singular values, which helps ensure robustness to noise. In particular, this threshold is $\delta(n) \triangleq c\epsilon\sqrt{\frac{n}{k}}$ for small constant $c \le 0.001$. Also, the PCA space is computed on a sample of the current pointset only.

See Algorithm V.1 for a detailed description of computing $\mathcal{U}$.

We now present the overall NNS algorithm in detail. The preprocessing stage runs Algorithm V.1 on the pointset $\tilde{P}$, stores its output, and constructs a k-dimensional NNS data structure for each of the pointsets $M_0, \dots, M_{j-1}$ (here j refers to the final value of this variable). Note that we also have a "left-over" set $R = \tilde{P}_j \bigcup \cup_{l=0}^{j-1} \tilde{P}_l^s$, which includes the points remaining at the end plus the sampled points used to construct the subspaces $\mathcal{U}$.

Algorithm V.1 Iteratively locate subspaces.

```
Define $\Psi \triangleq d\sigma^2 + 0.001\epsilon^2$, $r \triangleq O(d^9 k^3 \frac{\log n}{\epsilon^2\sigma^2})$, and $\delta(n) \triangleq c\epsilon\sqrt{\frac{n}{k}}$ for small constant $c \le 0.001$.
$j \leftarrow 0$, $\tilde{P}_0 \leftarrow \tilde{P}$
while $|\tilde{P}_j| > r$ do
    Sample $r$ points from $\tilde{P}_j$ (with repetition) to form the set/matrix $\tilde{P}_j^s$
    $m \leftarrow$ number of singular values of $\tilde{P}_j^s$ that are at least $\delta(r)$
    $\tilde{U}_j^s \leftarrow$ the subspace spanned by the $m$ top singular vectors of $\tilde{P}_j^s$
    $M_j \leftarrow$ all $\tilde{p} \in \tilde{P}_j \setminus \tilde{P}_j^s$ at distance $\mathrm{dist}(\tilde{p}, \tilde{U}_j^s) \le \sqrt{\Psi}$
    $\tilde{P}_{j+1} \leftarrow \tilde{P}_j \setminus (M_j \cup \tilde{P}_j^s)$
    $j \leftarrow j + 1$
end while
return the subspaces $\tilde{\mathcal{U}} = \{\tilde{U}_0^s, \dots, \tilde{U}_{j-1}^s\}$, their pointsets $\{M_0, M_1, \dots, M_{j-1}\}$, and the remaining set $R = \tilde{P}_j \bigcup \cup_{l=0}^{j-1} \tilde{P}_l^s$.
```

The query stage uses those j data structures to compute a $(1+\epsilon/8)$-approximates NNS of q in each of $M_0, \dots, M_{j-1}$, and additionally finds the NNS of q inside R by exhaustive search. It finally reports the closest point found.

We make henceforth three assumptions that hold without loss of generality. First, we assume that $\|p_i\| \ge 1$, which is without loss of generality as we can always move the pointset away from the origin. Overall, this ensures that $\|P\|_F^2 \ge |P|$.

Second, we assume that all points $\tilde{P}$ have norm at most $L \triangleq d^{3/2}$, which follows by applying a standard transformation of partitioning the dataset by a randomly shifted grid with side-length d. This transformation ensures that the query and the nearest neighbor, at distance $O(\sigma\sqrt{d})$ are in the same grid cell with probability at least $1 - o(1)$ (see, e.g., [1]).

Third, we assume that $\sigma \gg \epsilon/\sqrt{d}$, as otherwise we can apply the algorithm from Section III directly. (The algorithm in the current section works also for small σ, but the exposition becomes simpler if we assume a larger σ.) In the context of our model of Section II and Lemma II.3, this is equivalent to asserting $d \gg \log n$.

Finally, we remark that the algorithm can be changed to not use explicitly the value of σ, by taking only the closest $O\left(\sqrt{\frac{\log n}{d}}\right)$ points to a given space $\tilde{U}_j^s$. We omit the details.

B. Analysis

We now present a high-level overview of the proof. First, we characterize the space $\tilde{U}_j^s$, and in particular show that it is close to (a subspace of) the original space U, using the sine-theta machinery and matrix concentration bounds. Second, we use the closeness of $\tilde{U}_j^s$ to U to argue that: (a) projection of the noise onto $\tilde{U}_j^s$ is small; and (b) the projection of a point $\tilde{p}$ is approximately $\|p\|$, *on average*. Third, we use these bounds to show that the space $\tilde{U}_j^s$ captures a good fraction of points to be put into M_j, thus allowing us to bound the number of iterations. Fourth, we show that, for each point $\tilde{p} = p + t$ that has been "captured" into M_j, its projection into $\tilde{U}_j^s$ is a faithful representations of p, in the sense that, for such a point, the distance to the projection of $\tilde{q}$ onto $\tilde{U}_j^s$ is close to the original distance (before noise). This will suffice to conclude that the k-dimensional NNS for that set M_j shall return the right answer (should it happen to have the nearest neighbor p^*).

Slightly abusing notation, let P represent both the pointset and the corresponding $n \times d$ matrix, and similarly for $\tilde{P}$ or a subset thereof like $\tilde{P}_j$. Let T be the noise matrix, i.e., its rows are the vectors t_i and $\tilde{P} = P + T$. We shall use the following bounds from random matrix theory:

Theorem V.2 ([31], [40]). *Let the matrix $T \in \mathbb{R}^{n\times d}$ have entries drawn independently at random from $N(0,\sigma)$. Then with probability approaching 1 asymptotically as n and d increase, $\|T\| \le 3\sigma \max\{\sqrt{n}, \sqrt{d}\}$.*

Corollary V.3. *With high probability, for every subset A of the rows of T, with $|A| \ge d$, the corresponding sub-matrix T_A of T, has spectral norm $\|T_A\| \le \eta(|A|) = O(\sigma\sqrt{|A| \cdot \log n})$.*

In addition, by Corollary II.2 and the parameters of our model in Theorem V.1, w.h.p.

$$\forall p_i \in P, \qquad \|t_i\|^2 \le \sigma^2 d + 0.0001\epsilon^2. \tag{V.1}$$

We assume in the rest of the proof that these events occur. Since both are high probability events, we may use a union bound and assume they occur over all iterations without any effect of conditioning on the points.

The thrust of our proof below is to analyze one iteration of Algorithm V.1. We henceforth use j to denote an arbitrary iteration (not its final value), and let $n_j = |\tilde{P}_j| > r$ denote the number of points at that iteration.

Define $\tilde{U}_j^s$ and $\tilde{U}_j$ to be the PCA space of $\tilde{P}_j^s$ and $\tilde{P}_j$ respectively, i.e., full current set and sampled set. Stipulate the dimension of $\tilde{U}_j^s$ and $\tilde{U}_j$ to be $m \le k$ and $m \le \ell \le k$ where m is set according to the thresholding step in Algorithm V.1 and ℓ will be specified later. We show that the computed PCA space $\tilde{U}_j^s$ is close to U using the sine-theta machinery established in Section III. We consider the point sets as matrices, and concatenate the following two observations for deriving our result:

- The PCA space of sampled noisy set (scaled) is close to that of the full noisy set, by a standard sampling concentration result by Rudelson and Vershynin [39].
- The PCA space of the noisy set is close to that of the unperturbed set by using Theorem IV.1 with the bound on the spectral norm of random matrices from Lemma V.3.

In the full version, we use the above two items to show:

Lemma V.4.

$$\sin\theta(\tilde{U}_j^s, \tilde{U}_j) \le O\left(\frac{\sigma k^{1.5}\sqrt{\log n}}{\epsilon}\right) \text{ and}$$
$$\sin\theta(\tilde{U}_j, U_j) \le O\left(\frac{\sigma k^{1.5}\sqrt{\log n}}{\epsilon}\right).$$

Since $\sin\theta$ is concave in the right regime, Lemma V.4 now gives us as a simple corollary the following useful result:

$$\sin\theta(\tilde{U}_j^s, U_j) \le O\left(\frac{\sigma k^{1.5}\sqrt{\log n}}{\epsilon}\right). \tag{V.2}$$

1) Analysis: Noise inside the PCA space: We now show that the noise vector t_i of each point $\tilde{p}_i = p_i + t_i$ has a small component inside $\tilde{U}_j^s$. We use the $\sin\theta$ bound in Eqn. (V.2) for this.

Define $V \in \mathbb{R}^{d\times k}$ as the projection matrix onto the space U, so e.g. $t_i V$ is the zero vector, and define analogously $\tilde{V}_j^s \in \mathbb{R}^{d\times m}$ to be the projection matrix onto the m-dimensional space $\tilde{U}_j^s$. Equations (V.2) and Equation (V.1) easily imply now:

Lemma V.5. $\|t_i \tilde{V}_j^s\| \le \|t_i\| \cdot \sin\theta(\tilde{U}_j^s, U)$.

Corollary V.6. *For every point p_i and iteration j, $\|t_i\tilde{V}_j^s\| \le O\left(\frac{1}{\epsilon}\sigma^2 k^{1.5}\sqrt{d\log n}\right)$.*

We defer the routine proof to the full version.

We now show that the component of a data point $\tilde{p}_i$ inside the PCA space $\tilde{U}_j^s$ of some iteration j, typically recovers most of the "signal", i.e., the unperturbed version p_i. More precisely, we compare the length seen inside the PCA space $\|\tilde{p}_i\tilde{V}_j^s\|$ with the original length $\|p_i\|$. While the upper bound is immediate, the lower bound holds only *on average*. We again refer to the full version for proofs of the following lemmas:

Lemma V.7. *W.h.p., for all $\tilde{p}_i \in \tilde{P}_j$, $\|\tilde{p}_i\tilde{V}_j^s\|^2 - \|p_i\|^2 \le d\sigma^2 + 0.0001\epsilon^2$.*

Lemma V.8. $\sum_{\tilde{p}_i\in\tilde{P}_j}(\|\tilde{p}_i\tilde{V}_j^s\|^2 - \|p_i\|^2) \ge -k\delta(n_j)^2$.

We now show that each iteration captures in M_j a good fraction of the remaining points, thereby bounding the number of iterations overall. In particular, we give a lower bound on the number of indexes i such that $\tilde{p}_i$ is close to the m dimensional PCA subspace $\tilde{U}_j^s$. Note that the square of this distance for a point $\tilde{p}_i$ is precisely $\|\tilde{p}_i\|^2 - \|\tilde{p}_i\tilde{V}\|^2$.

Let X and Y be quantities according to Lemmas V.7 and V.8, such that

$$Xn_j \leq \sum_i (\|\tilde{p}_i \tilde{V}_j^s\|^2 - \|p_i\|^2); \tag{V.3}$$

$$\|\tilde{p}_i \tilde{V}_j^s\|^2 - \|p_i\|^2 \leq Y. \tag{V.4}$$

Now let f be the fraction of i's such that $\|\tilde{p}_i \tilde{V}_j^s\|^2 - \|p_i\|^2 \leq -0.0002\epsilon^2$. Then

$$\begin{aligned} Xn_j &\leq \sum_i \|\tilde{p}_i \tilde{V}_j^s\|^2 - \sum_i \|p_i\|^2 \\ &\leq (1-f)n_j Y - 0.0002 f n_j \epsilon^2. \end{aligned}$$

Rearrangement of terms gives us that $f \leq \frac{Y-X}{Y+0.0002\epsilon^2}$. By Lemma V.8 , we have $Xn_j = -k\delta^2 \geq -c^2\epsilon^2 n_j \geq -0.00001\epsilon^2 n_j$ and so $X \geq -0.00001\epsilon^2$. And by Lemma V.7, we have $Y \leq d\sigma^2 + 0.0001\epsilon^2$. Elementary calculations now yield

$$f \leq 1 - \Omega\left(\frac{\epsilon^2}{d\sigma^2}\right) \leq 1 - \Omega\left(\sqrt{\frac{\log n}{d}}.\right)$$

Now for the rest of the $(1-f) \geq \sqrt{\frac{\log n}{d}}$ fraction of the points, the distance to the PCA subspace $\tilde{U}$ is, by Pythagoras' Theorem,

$$\begin{aligned} \|\tilde{p}\|^2 - \|\tilde{p}\tilde{V}_j^s\|^2 &= \|t\|^2 + (\|p\|^2 - \|\tilde{p}\tilde{V}_j^s\|^2) \\ &\leq \|t\|^2 + 0.0002\epsilon^2. \end{aligned}$$

Since $\|t\|^2 \leq d\sigma^2 + 0.0001\epsilon^2$, we get the required inequality that a large fraction of the points is within squared distance $d\sigma^2 + 0.001\epsilon^2 = \Psi$. It follows that the fraction of points captured by $\tilde{U}_j^s$, i.e., in a single iteration, is at least $\Omega\left(\sqrt{\frac{\log n}{d}}\right)$, which immediately implies a bound on the number of iterations as follows.

Lemma V.9. *Algorithm V.1 will terminate in at most* $O\left(\sqrt{\frac{d}{\log n}} \log n\right) = O(\sqrt{d \log n})$ *iterations.*

It now remains to show that the data structure that captures the actual nearest neighbor $\tilde{p}^*$ will still report $\tilde{p}^*$ as the nearest neighbor to $\tilde{q}$ in the k-dimensional data structure. The proof is involved, so we only sketch some intuition here and discuss the details in the full version. Essentially, we use the randomness of the noise to show almost none of it projects down to $\tilde{U}$, and that almost all of the magnitude of each unperturbed point is captured by $\tilde{U}$. When using the randomness, we have to be careful to show that conditioning is not introduced by non selection of points in the same iteration, and here we use that the PCA at each step is only computed on a sample which is then set aside into a separate bucket.

2) Algorithm performance: We now remark on the resulting parameters of the algorithm.

Processing an iteration of the preprocessing stage takes $O(rd^2 + d^3 + ndk) = O(nd^2)$ time for: computing $\tilde{P}_j^s$, the PCA space, and M_j respectively. Hence, over $O(\sqrt{d \log n})$ iterations, together with preprocessing of the k-dimensional NNS data structures, we get preprocessing time $O((nd^2 + d^3 + F_{\text{prep}})\sqrt{d \log n})$. Space requirement is essentially that of $O(\sqrt{d \log n})$ instances of k-dimensional NNS data structure, plus the space to store $O(\sqrt{d \log n})$ spaces $\tilde{U}_j^s$, and the left-over set R.

The query time is composed of: computing the projections into $O(\sqrt{d \log n})$ subspaces, querying the k-dimensional NNS data structures, and computing the distances to left-over points in R. Overall, this comes out to $O(dk \cdot \sqrt{d \log n} + \sqrt{d \log n} \cdot F_{\text{query}} + d|R|)$.

VI. PCA TREE

We now present our second spectral algorithm, which is closely related to the PCA tree [43], [46]. We first give the algorithm and then present its analysis. Overall, we prove the following theorem.

Theorem VI.1. *Consider the Gaussian-noise model* (II.1)-(II.2)*, and assume that its parameters satisfy* $\sigma < \kappa \cdot \min\left\{\frac{\epsilon}{\sqrt{k \log n}}, \frac{\epsilon}{\sqrt{k}\sqrt[4]{d \log n}}\right\}$*, for sufficiently small constant* $\kappa > 0$*. There exists a data structure that preprocesses* $\tilde{P}$*, and then given the query* $\tilde{q}$*, returns the nearest neighbor* $\tilde{p}^*$ *w.h.p.*[5] *And w.h.p. the query time is* $(k/\epsilon)^{O(k)} \cdot d^2$*, the space requirement is* $O(nd)$*, and the preprocessing time is* $O(n^2 d + nd^3)$*.*

The algorithm itself is deterministic.

A. Algorithm description

The algorithm constructs one-dimensional partitioning tree hierarchically, where each tree node is associated with a subset of the pointset $\tilde{P}$. We start with the root of the tree, associated with all n points $\tilde{P}$. Now at each tree node x, we take the pointset associated with x, termed $\tilde{P}_x^{in}$. First, we perform a process called "de-clumping", which just discards part of the dataset, to obtain a set $\tilde{P}_x \subseteq \tilde{P}_x^{in}$. We describe this process at the end.

The main operation at a node is to take the top centered-PCA direction of $\tilde{P}_x$, termed v_x. By centered-PCA we mean subtracting from each vector in $\tilde{P}_x$ their average $a = \frac{1}{|\tilde{P}_x|}\sum_{\tilde{p} \in \tilde{P}_x} \tilde{p}$, and then taking the top PCA direction. Now, let $\theta \triangleq \frac{\epsilon}{1000k^{3/2}}$ and let Θ be the partition of the real line into segments of length θ, namely $\Theta = \{[\theta i, \theta(i+1)) \mid i \in \mathbb{Z}\}$. Then we partition $\tilde{P}_x$ into parts depending on which segment from Θ the projection of a point $\tilde{p} \in \tilde{P}_x$ onto v_x falls into. Then, we orthogonalize with respect to v_x, namely, transform each point $\tilde{p} \in \tilde{P}_x$ into $\tilde{p}' = \tilde{p} - \langle \tilde{p}, v_x \rangle v_x$. For

[5]The probability is over the randomness from the model.

each non-empty segment of Θ we produce a child of x associated with the points that fall into that segment, and repeat recursively on it. We stop once the current tree node has at most d points associated with it.

During a query, we follow the tree into all the buckets (slabs) that intersect a ball of radius $1+\epsilon/2$ around $\tilde{q}$. In each leaf, compute the exact distance from q to all points associated to that leaf. Finally, report the closest point found.

We now describe the de-clumping procedure that is done at each node. We compute the top centered-singular value of $\tilde{P}_x^{in}$. If this value is at least $\lambda_c = \lambda_c(|\tilde{P}_x^{in}|) \triangleq \frac{\epsilon}{16}\sqrt{|\tilde{P}_x^{in}|/k}$, then set $\tilde{P}_x \triangleq \tilde{P}_x^{in}$. Otherwise, find the closest pair of points in $\tilde{P}_x^{in}$, and let δ denote their squared-distance. Remove all the pairs of points in $\tilde{P}_x^{in}$ that have squared-distance at most $\delta+\epsilon^2/2$, to obtain $\tilde{P}_x$. (The removal is iterative, proceeding in arbitrary order.)

The thrust of our analysis is the following Lemma:

Lemma VI.2 (Tree Depth)**.** *The constructed PCA tree has depth at most $2k$.*

At a high level, our proof of Lemma VI.2 proceeds by showing that the top PCA direction v_x at each node x is close to U. We argue this by a careful induction over levels of the tree, and proving that declumping filters out noisy directions from our data. Due to our orthogonalization at each step, and since U is only of dimension k, we cannot discover too many "new" directions in or near U. The exact proof is quite technical, and hence we defer the details to the full version.

Having established an upper bound on the tree depth, we next show that the query algorithm will indeed return the nearest neighbor $\tilde{p}^*$ for the query $\tilde{q}$ (modelled as in Section II). We show this in two steps: first we prove the result, assuming the point $\tilde{p}^*$ was not thrown out during de-clumping. Then we show that the de-clumping indeed does not throw out the point $\tilde{p}^*$:

Lemma VI.3. *The query algorithm returns the point $\tilde{p}^*$, assuming it was not thrown out during the de-clumping process.*

Lemma VI.4. *p^* is never thrown out due to the de-clumping process.*

The proofs of both lemma are deferred to the full version due to space constraints. Now the space and preprocessing bounds follow immediately from the construction. We just need to argue about the query time.

Claim VI.5. *The query time is $(k/\epsilon)^{O(k)}d^2$.*

Proof: At each node of the tree, there are at most $O(1/\theta) = O(k^{3/2}/\epsilon)$ child nodes that are followed. Hence, in total, we reach $O(1/\theta)^{2k} = (k/\epsilon)^{O(k)}$ leaves. The factor of d^2 comes from the fact that each leaf has at most d points to check the distance against. ■

Acknowledgment

Amirali Abdullah conducted much of this work at Microsoft Research Silicon Valley and was also supported in part by the NSF grant #CCF-0953066. Robert Krauthgamer acknowledges work done in part while at Microsoft Research Silicon Valley. His work also supported in part by the Israel Science Foundation grant #897/13, the US-Israel BSF grant #2010418, and by the Citi Foundation.

References

[1] Aiger, D., Kaplan, H., and Sharir, M. Reporting neighbors in high-dimensional euclidean spaces. In *SODA* (2013), pp. 784–803.

[2] Alon, N. Problems and results in extremal combinatorics I. *Discrete Mathematics 273* (2003), 31–53.

[3] Andoni, A., and Indyk, P. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. In *Proceedings of the Symposium on Foundations of Computer Science (FOCS)* (2006), pp. 459–468.

[4] Andoni, A., Indyk, P., Nguyen, H., and Razenshteyn, I. Beyond locality-sensitive hashing. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA)* (2014).

[5] Andoni, A., Indyk, P., and Pătraşcu, M. On the optimality of the dimensionality reduction method. In *Proceedings of the Symposium on Foundations of Computer Science (FOCS)* (2006), pp. 449–458.

[6] Arora, S., and Kannan, R. Learning a mixture of gaussians. *Proceedings of the Symposium on Theory of Computing (STOC)* (2001).

[7] Arya, S., and Malamatos, T. Linear-size approximate voronoi diagrams. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA)* (2002), pp. 147–155.

[8] Arya, S., Malamatos, T., and Mount, D. M. Space-time tradeoffs for approximate nearest neighbor searching. *Journal of the ACM (JACM) 57*, 1 (2009), 1.

[9] Arya, S., Mount, D., Netanyahu, N., Silverman, R., and Wu, A. An optimal algorithm for approximate nearest neighbor searching. *J. ACM 6*, 45 (1998), 891–923. Previously appeared in SODA'94.

[10] Beygelzimer, A., Kakade, S., and Langford, J. Cover trees for nearest neighbor. In *23rd international conference on Machine learning* (2006), ACM, pp. 97–104.

[11] Clarkson, K. A randomized algorithm for closest-point queries. *SIAM Journal on Computing 17* (1988), 830–847.

[12] Clarkson, K. An algorithm for approximate closest-point queries. *Proceedings of the Tenth Annual ACM Symposium on Computational Geometry* (1994), 160–164.

[13] Cole, R., and Gottlieb, L.-A. Searching dynamic point sets in spaces with bounded doubling dimension. In *38th annual ACM symposium on Theory of computing* (2006), ACM, pp. 574–583.

[14] Dasgupta, S. Learning mixtures of gaussians. In *40th Annual Symposium on Foundations of Computer Science* (1999), IEEE Computer Society, pp. 634–.

[15] Dasgupta, S., and Freund, Y. Random projection trees and low dimensional manifolds. In *Proceedings of the 40th*

annual ACM symposium on Theory of computing (2008), ACM, pp. 537–546.

[16] DATAR, M., IMMORLICA, N., INDYK, P., AND MIRROKNI, V. Locality-sensitive hashing scheme based on p-stable distributions. *Proceedings of the ACM Symposium on Computational Geometry (SoCG)* (2004).

[17] DAVIS, C., AND KAHAN, W. M. The rotation of eigenvectors by a perturbation. III. *SIAM J. Numer. Anal. 7* (1970), 1–46.

[18] GRITZMANN, P., AND KLEE, V. Computational complexity of inner and outer j-radii of polytopes in finite-dimensional normed spaces. *Mathematical programming 59*, 1-3 (1993), 163–213.

[19] HAR-PELED, S. A replacement for voronoi diagrams of near linear size. In *Proceedings of the Symposium on Foundations of Computer Science (FOCS)* (2001), pp. 94–103.

[20] HAR-PELED, S., AND KUMAR, N. Approximate nearest neighbor search for low-dimensional queries. *SIAM J. Comput. 42*, 1 (2013), 138–159. Previously in SODA'11.

[21] HAR-PELED, S., AND MENDEL, M. Fast construction of nets in low dimensional metrics, and their applications. In *21st Annual Symposium on Computational Geometry* (2005), ACM, pp. 150–158.

[22] HAR-PELED, S., AND VARADARAJAN, K. R. High-dimensional shape fitting in linear time. *Discrete & Computational Geometry 32*, 2 (2004), 269–288.

[23] INDYK, P., AND MOTWANI, R. Approximate nearest neighbor: towards removing the curse of dimensionality. *Proceedings of the Symposium on Theory of Computing (STOC)* (1998), 604–613.

[24] INDYK, P., AND NAOR, A. Nearest neighbor preserving embeddings. *ACM Transactions on Algorithms* (2007).

[25] INDYK, P., AND WOODRUFF, D. Tight lower bounds for the distinct elements problem. *Proceedings of the Symposium on Foundations of Computer Science (FOCS)* (2003), 283–290.

[26] JAYRAM, T. S., AND WOODRUFF, D. P. Optimal bounds for johnson-lindenstrauss transforms and streaming problems with subconstant error. *ACM Transactions on Algorithms 9*, 3 (2013), 26. Previously in SODA'11.

[27] JOHNSON, W., AND LINDENSTRAUSS, J. Extensions of lipshitz mapping into hilbert space. *Contemporary Mathematics 26* (1984), 189–206.

[28] JOHNSON, W., AND SCHECHTMAN, G. Embedding ℓ_p^m into ℓ_1^n. *Acta Mathematica 149* (1982), 71–85.

[29] KARGER, D., AND RUHL, M. Finding nearest neighbors in growth-restricted metrics. *Proceedings of the Symposium on Theory of Computing (STOC)* (2002).

[30] KRAUTHGAMER, R., AND LEE, J. Navigating nets: Simple algorithms for proximity search. *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA)* (2004).

[31] LATAŁA, R. Some estimates of norms of random matrices. *Proceedings of the American Mathematical Society 133*, 5 (2005), 1273–1282.

[32] LAURENT, B., AND MASSARAT, P. Adaptive estimation of a quadratic functional by model selection. *Annals of Statistics 28* (1998), 1303–1338.

[33] MCNAMES, J. A fast nearest-neighbor algorithm based on a principal axis search tree. *Pattern Analysis and Machine Intelligence, IEEE Transactions on 23*, 9 (2001), 964–976.

[34] MEISER, S. Point location in arrangements of hyperplanes. *Information and Computation 106* (1993), 286–303.

[35] MOITRA, A., AND VALIANT, G. Settling the polynomial learnability of mixtures of gaussians. In *51st Annual IEEE Symposium on Foundations of Computer Science* (2010), IEEE, pp. 93–102.

[36] MUJA, M., AND LOWE, D. G. Fast approximate nearest neighbors with automatic algorithm configuration. In *VISAPP (1)* (2009), pp. 331–340.

[37] *Frontiers in Massive Data Analysis*. The National Academies Press, 2013.

[38] PANIGRAHY, R. Entropy-based nearest neighbor algorithm in high dimensions. *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA)* (2006).

[39] RUDELSON, M., AND VERSHYNIN, R. Sampling from large matrices: An approach through geometric functional analysis. *Journal of the ACM (JACM) 54*, 4 (2007), 21.

[40] RUDELSON, M., AND VERSHYNIN, R. Non-asymptotic theory of random matrices: extreme singular values. In *Proceedings of the International Congress of Mathematicians. Volume III* (New Delhi, 2010), Hindustan Book Agency, pp. 1576–1602.

[41] SALAKHUTDINOV, R., AND HINTON, G. Semantic hashing. *International Journal of Approximate Reasoning 50*, 7 (2009), 969–978.

[42] SILPA-ANAN, C., AND HARTLEY, R. Optimised kd-trees for fast image descriptor matching. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on* (2008), IEEE, pp. 1–8.

[43] SPROULL, R. Refinements to nearest-neighbor searching in k-dimensional trees. *Algorithmica 6* (1991), 579–589.

[44] VARADARAJAN, K. R., VENKATESH, S., AND ZHANG, J. On approximating the radii of point sets in high dimensions. In *43rd Annual IEEE Symposium on Foundations of Computer Science* (2002), IEEE, pp. 561–569.

[45] VEMPALA, S., AND WANG, G. A spectral algorithm for learning mixture models. *Journal of Computer and System Sciences 68*, 4 (2004), 841–860. Previously in FOCS'02.

[46] VERMA, N., KPOTUFE, S., AND DASGUPTA, S. Which spatial partition trees are adaptive to intrinsic dimension? In *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence* (2009), AUAI Press, pp. 565–574.

[47] WEDIN, P.-Å. Perturbation bounds in connection with singular value decomposition. *BIT Numerical Mathematics 12*, 1 (1972), 99–111.

[48] WEISS, Y., TORRALBA, A., AND FERGUS, R. Spectral hashing. In *Advances in neural information processing systems* (2008), pp. 1753–1760.

[49] WOODRUFF, D. Optimal space lower bounds for all frequency moments. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA)* (2004).

[50] YAGNIK, J., STRELOW, D., ROSS, D. A., AND LIN, R.-S. The power of comparative reasoning. In *Computer Vision (ICCV), 2011 IEEE International Conference on* (2011), IEEE, pp. 2431–2438.