Approximate *k*-flat Nearest Neighbor Search

Wolfgang Mulzer[*]
Freie Universität Berlin
mulzer@inf.fu-berlin.de

Huy L. Nguyễn
Simons Institute
hlnguyen@cs.princeton.edu

Paul Seiferth[*]
Freie Universität Berlin
pseiferth@inf.fu-berlin.de

Yannik Stein[†]
Freie Universität Berlin
yannikstein@inf.fu-berlin.de

ABSTRACT

Let $k \geq 0$ be an integer. In the *approximate k-flat nearest neighbor* (k-ANN) problem, we are given a set $P \subset \mathbb{R}^d$ of n points in d-dimensional space and a fixed approximation factor $c > 1$. Our goal is to preprocess P so that we can efficiently answer *approximate k-flat nearest neighbor queries*: given a k-flat F, find a point in P whose distance to F is within a factor c of the distance between F and the closest point in P. The case $k = 0$ corresponds to the well-studied approximate nearest neighbor problem, for which a plethora of results are known, both in low and high dimensions. The case $k = 1$ is called *approximate line nearest neighbor*. In this case, we are aware of only one provably efficient data structure, due to Andoni, Indyk, Krauthgamer, and Nguyễn (AIKN) [2]. For $k \geq 2$, we know of no previous results.

We present the first efficient data structure that can handle approximate nearest neighbor queries for arbitrary k. We use a data structure for 0-ANN-queries as a black box, and the performance depends on the parameters of the 0-ANN solution: suppose we have a 0-ANN structure with query time $O(n^{\rho})$ and space requirement $O(n^{1+\sigma})$, for $\rho, \sigma > 0$. Then we can answer k-ANN queries in time $O(n^{k/(k+1-\rho)+t})$ and space $O(n^{1+\sigma k/(k+1-\rho)} + n \log^{O(1/t)} n)$. Here, $t > 0$ is an arbitrary constant and the O-notation hides exponential factors in k, $1/t$, and c and polynomials in d.

Our approach generalizes the techniques of AIKN for 1-ANN: we partition P into *clusters* of increasing radius, and we build a low-dimensional data structure for a random projection of P. Given a query flat F, the query can be answered directly in clusters whose radius is "small" compared to $d(F, P)$ using a grid. For the remaining points, the low dimensional approximation turns out to be precise enough. Our new data structures also give an improvement in the space requirement over the previous result for 1-ANN: we can achieve near-linear space and sublinear query time, a further step towards practical applications where space constitutes the bottleneck.

[*]Partially supported by DFG Grants MU 3501/1 & MU 3501/2.

[†]Supported by the DFG within the RTG 'Methods for Discrete Structures' (GRK 1408).

Categories and Subject Descriptors

F.2.2 [**Nonnumerical Algorithms and Problems**]: Geometrical problems and computations

General Terms

Algorithms, Theory

Keywords

approximate nearest neighbors, k-flat, high dimensions

1. INTRODUCTION

Nearest neighbor search is a fundamental problem in computational geometry, with a countless number of applications in databases, information retrieval, computer vision, machine learning, signal processing, etc. [10]. Given a set $P \subset \mathbb{R}^d$ of n points in d-dimensional space, we would like to preprocess P so that for any query point $q \in \mathbb{R}^d$, we can quickly find the point in P that is closest to q.

When the dimension d is "small", there are efficient solutions [7, 17]. As d increases, these algorithms quickly become inefficient: either the query time approaches linear or the space grows exponentially with d. This phenomenon is usually called the "curse of dimensionality". Nonetheless, if one is satisfied with just an *approximate* nearest neighbor whose distance to the query point q lies within some factor $c = 1 + \varepsilon$, $\varepsilon > 0$, of the distance between q and the actual nearest neighbor, there are efficient solutions even for high dimensions. Several methods are known, offering trade-offs between the approximation factor, the space requirement, and the query time (see, e.g., [1, 3] and its references).

From a practical perspective, it is important to keep both the query time and the space small. Ideally, we would like algorithms with almost linear (or at least sub-quadratic) space requirement and sub-linear query time. Fortunately, there are solutions with these guarantees. These methods include *locality sensitive hashing* (LSH) [11, 12] and a more recent approach that improves upon LSH [3]. Specifically, the latter algorithm achieves query time $n^{7/(8c^2)+O(1/c^3)}$ and space $n^{1+7/(8c^2)+O(1/c^3)}$, where c is the approximation factor.

Often, however, the query object is more complex than a single point. Here, the complexity of the problem is much less understood. Perhaps the simplest such scenario occurs when the query object is a k-dimensional flat, for some small constant k. This is called the *approximate k-flat nearest neighbor* problem [2]. It constitutes a natural generalization of approximate nearest neighbors, which corresponds to $k = 0$. In practice, low-dimensional flats are used to model data subject to linear variations. For example, one could capture the appearance of a physical object under different lighting conditions or under different viewpoints [4].

So far, the only known algorithm with worst-case guarantees is for $k = 1$, the *approximate line nearest neighbor* problem. For this case, Andoni, Indyk, Krauthgamer, and Nguyễn (AIKN) achieve sub-linear query time $d^{O(1)}n^{1/2+t}$ and space $d^{O(1)}n^{O(1/\varepsilon^2+1/t^2)}$, for arbitrarily small $t > 0$. For the "dual" version of the problem, where the query is a point but the data set consists of k-flats, three results are known [4,14,15]. The first algorithm is essentially a heuristic with some control of the quality of approximation [4]. The second algorithm provides provable guarantees and a very fast query time of $(d + \log n + 1/\varepsilon)^{O(1)}$ [14]. The third result, due to Mahabadi, is very recent and improves the space requirement of Magen's result [15]. Unfortunately, these algorithms suffer from high space requirements, thus limiting their applicability in practice. In fact, even the basic LSH approach for $k = 0$ is already too expensive for large datasets and additional theoretical work and heuristics are required to reduce the memory usage and make LSH suitable for this setting [13, 18]. For $k \geq 2$, we know of no previous results.

Our results. We present the first efficient data structure for general approximate k-flat nearest neighbor search. Suppose we have a data structure for $(1-1/\log n)c$-approximate *point* nearest neighbors with query time $O(n^\rho + d\log n)$ and space $O(n^{1+\sigma} + d\log n)$, for some constants $\rho, \sigma > 0$. Then our algorithm achieves query time $O(d^{O(1)}n^{k/(k+1-\rho)+t})$ and space $O(d^{O(1)}n^{1+\sigma k/(k+1-\rho)} + n\log^{O(1/t)} n)$, where $t > 0$ can be made arbitrarily small. The constant factors for the query time depend on k, c, and $1/t$. Our main result is as follows.

Theorem 1.1. *Fix $k \in \mathbb{N}$ and an approximation factor $c > 1$. Suppose we have a data structure for $(1-1/\log n)c$-approximate* point *nearest neighbors with query time $O(n^\rho + d\log n)$ and space $O(n^{1+\sigma} + d\log n)$, for constants $\rho, \sigma > 0$. Let $P \subset \mathbb{R}^d$ be a d-dimensional n-point set. For any $t > 0$, we can construct a randomized data structure with space $O(d^{O(1)}n^{1+k\sigma/(k+1-\rho)} + n\log^{O(1/t)} n)$ that answers the following queries in $O(d^{O(1)}n^{k/(k+1-\rho)+t})$ expected time: given a k-flat $F \subset \mathbb{R}^d$, find a point $p \in P$ with $d(p, F) \leq cd(P, F)$.*

Table 1 gives an overview of some approximate point nearest neighbor structures that can be used in Theorem 1.1. The result by AINR gives the current best query performance for large enough values of c. For smaller c, an approach using locality sensitive hashing (LSH1) might be preferable. With another variant of locality sensitive hashing (LSH2), the space can be made almost linear, at the expense of a slightly higher query time. The last result (and related results, e.g., [13]) is of particular interest as the memory consumption is a major bottleneck in practice. It also improves the previous algorithm by AIKN for line queries.

Data Structure	ρ	σ
AINR [3]	$\frac{7}{8}c^{-2} + O(c^{-3})$	$\frac{7}{8}c^{-2} + O(c^{-3})$
LSH1 [1, Thm. 3.2.1]	$1/c^2$	$1/c^2$
LSH2 [1, Thm 3.4.1]	$O(1/c^2)$	0

Table 1: Overview of relevant ANN structures.

Along the way towards Theorem 1.1, we present a novel data structure for k-flat *near* neighbor reporting when the dimension d is constant. The space requirement in this case is $O_d(n\log^{O(d)} n)$ and the time needed for a query is $O_d(n^{k/(k+1)}\log^{d-k-1} n + |R|)$, where R is the answer set. This data structure may be of independent interest and may lead to further applications. Our results provide a vast generalization of the result in AIKN and shows for the first time that it is possible to achieve provably efficient nearest neighbor search for higher-dimensional query objects.

Our techniques. Our general strategy is similar to the approach by AIKN. The data structure consists of two main structures: the *projection structure* and the *clusters*. The projection structure works by projecting the point set to a space of constant dimension and by answering the nearest neighbor query in that space. As we will see, this suffices to obtain a rough estimate for the distance, and it can be used to obtain an exact answer if the point set is "spread out".

Unfortunately, this does not need to be the case. Therefore, we partition the point set into a sequence of *clusters*. A cluster consists of m points and a k-flat K such that all points in the cluster are "close" to K, where m is a parameter to be optimized. Using a rough estimate from the projection structure, we can classify the clusters as *small* and *large* (depending on their radius). The points in the large clusters are spread out and can be handled through projection. The points in the small clusters are well behaved and can be handled directly in high dimensions using grids and discretization.

Organization. In order to provide the curious reader with quick gratification, we will give the main data structure together with the properties of the cluster and the projection structure in Section 2. Considering these structures as black boxes, this already proves Theorem 1.1.

In the remainder of the paper, we describe the details of the helper structures. The necessary tools are introduced in Section 3. Section 4 gives the approximate nearest neighbor algorithm for small clusters. In Section 5, we consider approximate near neighbor reporting for k-flats in constant dimension. This data structure is then used for the projection structures in Section 6.

2. MAIN DATA STRUCTURE AND ALGORITHM OVERVIEW

We describe our main data structure for approximate k-flat nearest neighbor search. It relies on various substructures that will be described in the following sections. We let P denote a d-dimensional n-point set, and $c > 1$ is the desired approximation factor.

Let K be a k-flat in d dimensions. The *flat-cluster* C (or cluster for short) of K with radius α is the set of all points with distance at most α to K, i.e., $C = \{p \in \mathbb{R}^d \mid d(p, K) \leq \alpha\}$. A cluster is *full* if it contains at least m points from P, where m is a parameter to be determined. We call P *α-cluster-free* if there is no full cluster with radius α. Let $t > 0$ be an arbitrarily small parameter. Our data structure requires the following three subqueries.

Q1: Given a query flat F, find a point $p \in P$ with $d(p, F) \leq n^t d(P, F)$.

Q2: Assume P is contained in a flat-cluster with radius α. Given a query flat F with $d(P, F) \geq \alpha/n^{2t}$, return a point $p \in P$ with $d(p, F) \leq cd(P, F)$.

Q3: Assume P is $\alpha n^{2t}/(2k+1)$-cluster free. Given a query flat F with $d(P, F) \leq \alpha$, find the nearest neighbor $p^* \in P$ to F.

Briefly, our strategy is as follows: during the preprocessing phase, we partition the point set into a set of full clusters of increasing radii. To answer a query F, we first perform a query of type **Q1** to obtain an n^t-approximate estimate $\widetilde{r}$ for $d(P, F)$. Using $\widetilde{r}$, we identify the "small" clusters (by radius). These clusters can be processed using a query of type **Q2**. The remaining point set contains no "small" full cluster, so we can process it with a query of type **Q3**.

We will now describe the properties of the subqueries and the organization of the data structure in more detail. The data structure for **Q2**-queries is called the *cluster structure*. It is described in Section 4, and it has following properties.

THEOREM 2.1. *Let Q be a d-dimensional m-point set that is contained in a flat-cluster of radius α. Let $c > 1$ be an approximation factor. Using space $O_c(m^{1+\sigma} + d \log^2 m)$, we can construct a data structure with the following property. Given a query k-flat F with $d(P, F) \geq \alpha/n^{2t}$ and an estimate $\widetilde{r}$ with $d(P, F) \in [\widetilde{r}/n^t, \widetilde{r}]$, we can find a c-approximate nearest neighbor for F in Q in $O_c((n^{2t}k^2)^{k+1}(m^{1-1/k+\rho/k} + (d/k)\log m))$ total time.*

The data structures for **Q1** and **Q3** are very similar, and we cover them in Section 6. They are called *projection structures*, since they are based on projecting P into a low dimensional subspace. In the projected space, we use a data structure for approximate k-flat *near* neighbor search to be described in Section 5. The projection structures have the following properties.

THEOREM 2.2. *Let P be a d-dimensional n-point set, and let $t > 0$ be a small enough constant. Using space and time $O(n \log^{O(1/t)} n)$, we can obtain a data structure for the following query: given a k-flat F, find a point $p \in P$ with $d(p, F) \leq n^t d(P, F)$. A query needs $O(n^{k/(k+1)} \log^{O(1/t)} n)$ time, and the answer is correct with high probability.*

THEOREM 2.3. *Let $P \subset \mathbb{R}^d$ be an n-point set, and let $t > 0$ be a small enough constant. Using space and time $O(n \log^{O(1/t)} n)$, we can obtain a data structure for the following query: given a k-flat F and $\alpha > 0$ such that $d(F, P) \leq \alpha$ and such that P is $\alpha n^t/(2k+1)$-cluster-free, find an exact nearest neighbor for F in P. A query requires time $O(n^{k/(k+1)} \log^{O(1/t)} n + m)$, and the answer is correct with high probability. Here, m denotes the size of a full cluster.*

2.1 Constructing the Data Structure

First, we build a projection structure for **Q1** queries on P. This needs $O(n \log^{O(1/t)} n)$ space, by Theorem 2.2. Then, we repeatedly find the full flat-cluster C with smallest radius. The m points in C are removed from P, and we build a cluster structure for **Q2** queries on this set. By Theorem 2.1, this needs $O_c(m^{1+\sigma} + d \log^2 m)$ space. To find C, we check all flats K spanned by $k+1$ distinct points of P. In Lemma 3.2 below, we prove that this provides a good enough approximation. In the end, we have n/m point sets $Q_1, \ldots, Q_{n/m}$ ordered by decreasing radius, i.e., the cluster for Q_1 has the largest radius. The total space occupied by the cluster structures is $O(nm^\sigma + (n/m)d \log^2 n)$.

Finally, we build a perfect binary tree T with n/m leaves labeled $Q_1, \ldots, Q_{n/m}$, from left to right. For a node $v \in T$ let Q_v be the union of all Q_i assigned to leaves below v. For each $v \in T$ we build a data structure for Q_v to answer **Q3** queries. Since each point is contained in $O(\log n)$ data structures, the total size is $O(n \log^{O(1/t)} n)$, by Theorem 2.3. For pseudocode, see Algorithm 1.

Algorithm 1: Preprocessing algorithm. Compared with AIKN [2], we organize the projection structure in a tree to save space.

```
Input: point set $P \subset \mathbb{R}^d$, approximation factor $c$,
       parameter $t > 0$
1 $Q \leftarrow P$
2 for $i \leftarrow n/m$ downto 1 do
3     For each $V \in \binom{Q}{k+1}$, consider the $k$-flat $K_V$ defined
      by $V$. Let $\alpha_V$ be the radius of the smallest
      flat-cluster of $K_V$ with exactly $m$ points of $Q$.
4     Choose the flat $K = K_V$ that minimizes $\alpha_V$ and set
      $\alpha_i = \alpha_V$.
5     Remove from $Q$ the set $Q_i$ of $m$ points in $Q$ within
      distance $\alpha_i$ from $K$.
6     Construct cluster structure $C_i$ for cluster $(K, Q_i)$.
7 Build a perfect binary tree $T$ with $n/m$ leaves, labeled
  $Q_1, \ldots, Q_{n/m}$ from left to right.
8 foreach node $v \in T$ do
9     Build data structure for Q3 queries as in Thm. 2.3
      for the set $Q_v$ corresponding to the leaves below $v$.
```

2.2 Performing a Query

Suppose we are given a k-flat F. To find an approximate nearest neighbor for F we proceed similarly as AIKN [2]. We use **Q2** queries on "small" clusters and **Q3** queries on the remaining points; for pseudocode, see Algorithm 2.

First, we perform a query of type **Q1** to obtain a n^t-approximate nearest neighbor p_1 for F. This needs time $O(n^{k/(k+1)} \log^{O(1/t)} n)$. Let $\widetilde{r} = d(p_1, F)$. We use $\widetilde{r}$ as an estimate to distinguish between "small" and "large" clusters. Let $i^* \in \{1, \ldots, n/m\}$ be the largest integer such that the cluster assigned with Q_{i^*} has radius $\alpha_{i^*} > \widetilde{r} n^t$. For $i = i^* + 1, \ldots, n/m$, we use $\widetilde{r}$ as an estimate for a **Q2** query on Q_i. Since $|Q_i| = m$ and by Theorem 2.1, this needs total time $O(n^{2t(k+1)+1} m^{-1/k+\rho/k} + (n/m)d \log^2 m)$.

It remains to deal with points in "large" clusters. The goal is to perform a type **Q3** query on $\bigcup_{1 \leq i \leq i^*} Q_i$. For this, we start at the leaf of T labeled Q_{i^*} and walk up to the root. Each time we encounter a new node v from its

Algorithm 2: Algorithm for finding approximate nearest neighbor in high dimensions.

```
  Input  : query flat $F$
  Output: a $c$-approximate nearest neighbor for $F$ in $P$
1 Query the root of $T$ for a $n^t$-approximate nearest
  neighbor $p_1$ to $F$.                          /* type Q1 */
2 $\widetilde{r} \leftarrow d(p_1, F)$
3 $i^* \leftarrow$ maximum $i \in \{1, \dots, n/m\}$ with $\alpha_i > \widetilde{r}n^t$, or 0 if
  no such $i$ exists
4 for $i \leftarrow i^* + 1$ to $n/m$ do
      /* type Q2; we have $d(Q_i, F) \geq \widetilde{r}/n^t \geq \alpha_i/n^{2t}$ */
5     Query cluster structure $C_i$ with estimate $\widetilde{r}$.
  /* type Q3                                           */
6 Query projection structure for a $\widetilde{r}$-thresholded nearest
  neighbor of $F$ in $Q = \bigcup_{i=1}^{j^*} U_i$. return closest point to
  $F$ among query results.
```

right child, we perform a **Q3** query on Q_u, where u denotes the left child of v. Let L be all the left children we find in this way. Then clearly we have $|L| = O(\log n)$ and $\bigcup_{u \in L} Q_u = \bigcup_{1 \leq i \leq i^*} Q_i$. Moreover, by construction, there is no full cluster with radius less than $\widetilde{r}n^t$ defined by $k+1$ vertices of Q_u for any $u \in L$. We will see that this implies every Q_u to be $\widetilde{r}n^t/(2k+1)$-cluster-free, so Theorem 2.3 guarantees a total query time of $O(n^{k/(k+1)} \log^{O(1/t)} n + m)$ for this step. Among all the points we obtained during the queries, we return the one that is closest to F. A good trade-off point is achieved for $m = nm^{-1/k+\rho/k}$, i.e., for $m = n^{k/(k+1-\rho)}$. This gives the bounds claimed in Theorem 1.1.

Correctness. Let p^* be a point with $d(p^*, F) = d(P, F)$. First, suppose that $p^* \in Q_i$, for some $i > i^*$. Then, we have $d(p^*, F) \geq \widetilde{r}/n^t \geq \alpha_i/n^{2t}$, where α_i is the radius of the cluster assigned to Q_i. Since $\widetilde{r}$ is a valid n^t-approximate estimate for $d(F, Q_i)$, a query of type **Q2** on Q_i gives a c-approximate nearest neighbor, by Theorem 2.1. Now, suppose that $p^* \in Q_i$ for $1 \leq i \leq i^*$. Let u be the node of L with $p^* \in Q_u$. Then Theorem 2.3 guarantees that we will find p^* when doing a **Q3** query on Q_u.

3. PRELIMINARIES

Partition Trees. Fix an integer constant $r > 0$, and let $P \subset \mathbb{R}^d$ be a d-dimensional n-point set. A *simplicial r-partition* Ξ for P is a sequence $\Xi = (P_1, \Delta_1), \dots, (P_m, \Delta_m)$ of pairs such that (i) the sets $P_1, \dots, P_m$ form a partition of P with $n/r \leq |P_i| \leq \lceil 2n/r \rceil$, for $i = 1, \dots, m$; (ii) each Δ_i is a relatively open simplex with $P_i \subset \Delta_i$, for $i = 1, \dots, m$; and (iii) every hyperplane h in $\mathbb{R}^d$ *crosses* $O(r^{1-1/d})$ simplices Δ_i in Ξ. Here, a hyperplane h *crosses* a simplex Δ if h intersects Δ, but does not contain it. In a classic result, Matoušek showed that such a simplicial partition always exists and that it can be computed efficiently [6, 16].

By repeatedly computing a simplicial partition, one can construct a *partition tree* for P. A partition tree $\mathcal{T}$ is a rooted tree in which each node is associated with a pair (Q, Δ), such that Q is a subset of P and Δ is a relatively open simplex that contains Q. If $|Q| \geq 2r$, the children of (Q, Δ) constitute a simplicial r-partition of Q. Otherwise, the node (Q, Δ) has $|Q|$ children where each child corresponds to a point in Q. A partition tree has constant degree, linear size, and logarithmic depth.

Given a hyperplane h, there is a straightforward query algorithm to find the highest nodes in $\mathcal{T}$ whose associated simplex does not cross h: start at the root and recurse on all children whose associated simplex crosses h; repeat until there are no more crossings or until a leaf is reached. The children of the traversed nodes whose simplices do not cross h constitute the desired answer. In 2012, Chan [5] gave an optimal construction for partition trees.

Theorem 3.1 (Opt. Partition Trees [5]). *Let $P \subset \mathbb{R}^d$ be an n-point set. For any large enough constant r, there is a* partition tree *$\mathcal{T}$ with the following properties: (i) the tree $\mathcal{T}$ has degree $O(r)$ and depth $\log_r n$; (ii) each node is of the form (Q, Δ), where Q is a subset of P and Δ a relatively open simplex that contains Q; (iii) for each node (Q, Δ), the simplices of the children of Q are contained in Δ and are pairwise disjoint; (iv) the point set associated with a node of depth ℓ has size at most n/r^ℓ; (v) for any hyperplane h in $\mathbb{R}^d$, the number m_ℓ of simplices in $\mathcal{T}$ that h intersects at level ℓ obeys the recurrence*

$$m_\ell = O(r^{\ell(d-1)/d} + r^{(d-2)/(d-1)} m_{\ell-1} + r\ell \log r \log n).$$

Thus, h intersects $O(n^{1-1/d})$ simplices in total. The tree $\mathcal{T}$ can be built in expected time $O(n \log n)$.

k-flat Discretization. For our cluster structure we must find k-flats that are close to many points. The following lemma shows that it suffices to check "few" k-flats for this. The proof generalizes the proof of Lemma 2.3 by AIKN [2] and it can be found in the full version.

Lemma 3.2. *Let $P \subset \mathbb{R}^d$ be a finite point set with $|P| \geq k+1$, and let $F \subset \mathbb{R}^d$ be a k-flat. There is a k-flat F' such that F' is the affine hull of $k+1$ points in P and $\delta_{F'}(P) \leq (2k+1)\delta_F(P)$, where $\delta_{F'}(P) = \max_{p \in P} \mathrm{d}(p, F')$ and $\delta_F(P) = \max_{p \in P} \mathrm{d}(p, F)$.* □

4. CLUSTER STRUCTURE

A *k-flat cluster structure* consists of a k-flat K and a set Q of m points with $d(q, K) \leq \alpha$, for all $q \in Q$. Let $K : u \mapsto A'u + a$ be a parametrization of K, with $A' \in \mathbb{R}^{d \times k}$ and $a \in \mathbb{R}^d$ such that the columns of A' constitute an orthonormal basis for K and such that a is orthogonal to K. We are also given an approximation parameter $c > 1$. The cluster structure uses a data structure for approximate point nearest neighbor search as a black box. We assume that we have such a structure available that can answer c-approximate point nearest neighbor queries in d dimensions with query time $O_c(n^\rho + d \log n)$ and space requirement $O_c(n^{1+\sigma} + d \log n)$ for some constants $\rho, \sigma > 0$. As mentioned in the introduction, the literature offers several data structures for us to choose from.

The cluster structure distinguishes two cases: if the query flat F is close to K, we approximate F by few "patches" parallel to K, s.t. a good nearest neighbor for the patches is also good for K. Since the patches are parallel to K, they can be handled through 0-ANN queries in the orthogonal space $K^\perp$ and low-dimensional queries inside K. If the query flat is far from K, we approximate Q by its projection onto K and handle the query with a low-dimensional data structure.

4.1 Preprocessing

Let $K^\perp$ be the linear subspace of $\mathbb{R}^d$ orthogonal to K. Let Q_a be the projection of Q onto K, and let Q_b be the projection of Q onto $K^\perp$. We compute a k-dimensional partition tree $\mathcal{T}$ for Q_a. As stated in Theorem 3.1, the tree $\mathcal{T}$ has $O(m)$ nodes, and it can be computed in time $O(m \log m)$.

For each node (S_a, Δ) of $\mathcal{T}$, we do the following: we determine the set $S \subseteq Q$ whose projection onto K gives S_a, and we take the projection S_b of S onto $K^\perp$. Then, we build a $d-k$ dimensional c'-ANN data structure for S_b, as given by the assumption, where $c' = (1 - 1/\log n)c$. See Algorithm 3.

Algorithm 3: CreateClusterStructure

```
   Input: k-flat $K \subseteq \mathbb{R}^d$, point set $Q \subset \mathbb{R}^d$ with
          $d(q, K) \leq \alpha$ for all $q \in Q$, approximation
          parameter $c > 1$
1  $Q_a \leftarrow$ projection of $Q$ onto $K$
2  $Q_b \leftarrow$ projection of $Q$ onto $K^\perp$
3  Build a $k$-dimensional partition tree $\mathcal{T}$ for $Q_a$ as in
   Theorem 3.1.
4  $c' \leftarrow (1 - 1/\log n)c$
5  foreach node $(S_a, \Delta) \in \mathcal{T}$ do
6      $S_b \leftarrow$ projection of the points in $Q$ corresponding to
       $S_a$ onto $K^\perp$
7      Build a $(d-k)$-dimensional $c'$-ANN structure for $S_b$
       as given by the assumption.
```

LEMMA 4.1. *The cluster structure can be constructed in total time $O_c(m^{2+\rho} + md\log^2 m)$, and it requires $O_c(m^{1+\sigma} + d\log^2 m)$ space.*

PROOF. By Theorem 3.1, the partition tree can be built in $O(m \log m)$ time. Thus, the preprocessing time is dominated by the time to construct the c'-ANN data structures at the nodes of the partition tree $\mathcal{T}$. Since the sets on each level of $\mathcal{T}$ constitute a partition of Q, and since the sizes of the sets decrease geometrically, the bounds on the preprocessing time and space requirement follow directly from our assumption. Note that by our choice of $c' = (1 - 1/\log n)c$, the space requirement and query time for the ANN data structure change only by a constant factor. $\square$

4.2 Processing a Query

We set $\varepsilon = 1/100 \log n$. Let F be the query k-flat, given as $F : v \mapsto B'v + b$, with $B' \in \mathbb{R}^{d\times k}$ and $b \in \mathbb{R}^d$ such that the columns of B' are an orthonormal basis for F and b is orthogonal to F. Our first task is to find bases for the flats K and F that provide us with information about the relative position of K and F. For this, we take the matrix $M = A'^T B' \in \mathbb{R}^{k\times k}$, and we compute a *singular value decomposition* $M = U\Sigma V^T$ of M [9, Chapter 7.3]. Recall that U and V are orthogonal $k \times k$ matrices and that $\Sigma = \mathrm{diag}(\sigma_1, \dots, \sigma_k)$ is a $k \times k$ diagonal matrix with $\sigma_1 \geq \dots \geq \sigma_k \geq 0$. We call $\sigma_1, \dots, \sigma_k$ the *singular values* of M. The following lemma summarizes the properties of the SVD that are relevant to us.

LEMMA 4.2. *Let $M = A'^T B'$, and let $M = U\Sigma V^T$ be a singular value decomposition for M. Let $u_1, \dots, u_k$ be the columns of U and $v_1, \dots, v_k$ be the columns of V. Then, (i) $u_1, \dots, u_k$ is an orthonormal basis for K (in the coordinate system induced by A'); (ii) $v_1, \dots, v_k$ is an orthonormal basis for F (in the coordinate system induced by B'): and (iii) for $i = 1, \dots, k$, the projection of v_i onto K is $\sigma_i u_i$ and the projection of u_i onto F is $\sigma_i v_i$ (in the coordinate systems induced by A' and B'). In particular, we have $\sigma_1 \leq 1$.* $\square$

We reparametrize K according to U and F according to V. More precisely, we set $A = A'U$ and $B = B'V$, and we write $K : u \mapsto Au + a$ and $F : v \mapsto Bv + b$. The new coordinate system provides a simple representation for the distances between F and K. We begin with a technical lemma that is a simple corollary of Lemma 4.2.

LEMMA 4.3. *Let $a_1, \dots, a_k$ be the columns of the matrix A; let $a_1^\parallel, \dots, a_k^\parallel$ be the columns of the matrix $BB^T A$, and $a_1^\perp, \dots, a_k^\perp$ the columns of the matrix $A - BB^T A$. Then, (i) for $i = 1, \dots, k$, the vector $a_i^\parallel$ is the projection of a_i onto F and the vector $a_i^\perp$ is the projection of a_i onto $F^\perp$; (ii) for $i = 1, \dots, k$, we have $\|a_i^\parallel\| = \sigma_i$ and $\|a_i^\perp\| = \sqrt{1 - \sigma_i}$; and (iii) the vectors $a_1^\parallel, \dots, a_k^\parallel, a_1^\perp, \dots, a_k^\perp$ are pairwise orthogonal. An analogous statement holds for the matrices B, $AA^T B$, and $B - AA^T B$.*

The next lemma shows how our choice of bases gives a convenient representation of the distances between F and K. Its proof can be found in the full version.

LEMMA 4.4. *Take two points $x_F \in K$ and $y_K \in F$ such that $d(F, K) = d(y_K, x_F)$. Write $x_F = Au_F + a$ and $y_K = Bv_K + b$. Then, for any point $x \in K$ with $x = Au + a$, we have*

$$d(F, x)^2 = \sum_{i=1}^{k} \left(1 - \sigma_i^2\right) (u - u_F)_i^2 + d(F, K)^2,$$

and for any point $y \in F$ with $y = Bv + b$, we have

$$d(y, K)^2 = \sum_{i=1}^{k} \left(1 - \sigma_i^2\right) (v - v_K)_i^2 + d(F, K)^2. \quad \square$$

Algorithm 4: QueryClusterStructure

```
    Input: query k-flat $F \subseteq \mathbb{R}^d$; an estimate $\widetilde{r}$ with
           $d(F, Q) \in [\widetilde{r}/n^t, \widetilde{r}]$.
1   $M \leftarrow A'^T B'$.
2   Compute an SVD $M = U\Sigma V^T$ of $M$ with singular
    values $1 \geq \sigma_1 \geq \dots \geq \sigma_k \geq 0$.
3   if $\sigma_k = 1$ then
4       $f \leftarrow$projection of $F$ onto $K^\perp$ ;    /* $F$ and $K$ are
        parallel; $f$ is a point */
5       return $c'$-ANN for $f$ in $Q_b$
6   Reparametrize $K$ according to $U$ and $F$ according to $V$.
    /* Near case                                    */
7   $\mathcal{G} \leftarrow$ set of approximate patches obtained by combining
    Lemma 4.6 and 4.7
8   $R \leftarrow \emptyset$
9   foreach $G \leftarrow \mathcal{G}$ do
10      $R \leftarrow R \cup$ result of ANN query for $G$ as in Lem. 4.8
    /* Far case                                     */
11  $R \leftarrow R \cup$ result of ANN query for $G$ as in Lem. 4.11
12  return point in $R$ that minimizes the distance to $F$
```

We now give a overview of the query algorithm (see Algorithm 4). First, we check if F and K are parallel, i.e., if $\sigma_1 = \dots = \sigma_k = 1$. In this case, we need to perform only a single c'-ANN query in Q_b to obtain the desired result. If F and K are not parallel, we distinguish two cases: if F is far from Q, we approximate Q by its projection Q_a onto K.

Thus, we take the closest point x_F in K to F, and we return an approximate nearest neighbor for x_F in Q_a according to an appropriate metric derived from Lemma 4.4. See Section 4.2.2 for details. If F is close to Q, we use Lemma 4.4 to discretize the relevant part of F into *patches*, such that each patch is parallel to K and such that the best nearest neighbor in Q for the patches provides an ANN for F. Each patch can then be handled essentially by an appropriate nearest neighbor query in $K^\perp$. Details follow in Section 4.2.1. We say F and Q are *close* if $d(F,Q) \le \alpha/\varepsilon$, and *far* if $d(F,Q) > \alpha/\varepsilon$. Recall that we chose $\varepsilon = 1/100 \log n$.

4.2.1 Near: $d(F,Q) \le \alpha/\varepsilon$

We use our reparametrization of F and K to split the coordinates as follows: recall that $1 \ge \sigma_1 \ge \cdots \ge \sigma_k \ge 0$ are the singular values of $M = A'^T B'$. Pick $l \in \{0,\dots,k\}$ such that $1 \ge \sigma_i \ge \sqrt{1-\varepsilon}$, for $i = 1,\dots,l$, and $\sqrt{1-\varepsilon} > \sigma_i \ge 0$, for $i = l+1,\dots,k$. For a $d \times k$ matrix X, let $X_{[i]}$ denote the $d \times i$ matrix with the first i columns of X, and $X_{-[i]}$ the $d \times (k-i)$ matrix with the remaining $k-i$ columns of X. Similarly, for a vector $v \in \mathbb{R}^k$, let $v_{[i]}$ be the vector in $\mathbb{R}^i$ with the first i coordinates of v, and $v_{-[i]}$ the vector in $\mathbb{R}^{k-i}$ with the remaining $k-i$ coordinates of v.

The next lemma is a direct consequence of Lemma 4.4. It says that we can partition the directions in F into those that are almost parallel to K and those that are almost orthogonal to K. Along the orthogonal directions, we discretize F into few lower-dimensional flats that are almost parallel to K. After that, we approximate these flats by few patches that are actually parallel to K. These patches are then used to perform the query. The proof is in the full version.

LEMMA 4.5. *Let $y \in F$ be a point and $y_K \in F$ with $d(F,K) = d(y_K,K)$. Write $y_K = Bv_K + b$ and $y = Bv + b$. Then, $\left\|(v - v_K)_{-[l]}\right\| \le d(y,K)/\sqrt{\varepsilon}$.* □

Using Lemma 4.5, we can discretize the query F into a set of l-flats that are almost parallel to the cluster flat K, as stated in the next lemma. Again, the proof can be found in the full version.

LEMMA 4.6. *There is a set $\mathcal{L}$ of $O((n^{2t}k^{1/2}\varepsilon^{-5/2})^{k-l})$ l-flats such that the following holds: (i) for every $L \in \mathcal{L}$, we have $L \subseteq F$; (ii) for every $L \in \mathcal{L}$ and for every unit vector $u \in L$, the projection of u onto K has length at least $\sqrt{1-\varepsilon}$; and (iii) if $d(F,Q) \in [\alpha/n^{2t}, \alpha/\varepsilon]$, then there is an l-flat $L \in \mathcal{L}$ with $d(L,Q) \le (1+\varepsilon)d(F,Q)$.* □

From now on, we focus on an approximate query l-flat $L : w \mapsto B_1 w + b_1$ with $B_1 = B_{[l]}$. Our next goal is to approximate L by a set of patches such that each is parallel to K.

LEMMA 4.7. *There is a set $\mathcal{G}$ of $O((n^{2t}k^{1/2}\varepsilon^{-2})^l)$ patches such that the following holds: (i) every $G \in \mathcal{G}$ is an l-dimensional polytope, given by $O(l)$ inequalities; (ii) for every $G \in \mathcal{G}$, the affine hull of G is parallel to K; (iii) if $d(L,Q) \in [\alpha/n^{2t}, 2\alpha/\varepsilon]$, then there exists a $G \in \mathcal{G}$ with $d(G,Q) \le (1+\varepsilon)d(L,Q)$; (iv) for all $G \in \mathcal{G}$ and for all $q \in Q$, we have $d(G,q) \ge (1-\varepsilon)d(L,q)$.*

PROOF. Let $C = AA^T B_1$ be the $d \times l$ matrix whose columns $b_1^{\|}, \dots, b_l^{\|}$ constitute the projections of the columns of B onto K. By Lemma 4.3, the vectors $b_i^{\|}$ are orthogonal with $\|b_i^{\|}\| = \sigma_i$, for $i = 1,\dots,l$, and the columns $b_1^{\perp}, \dots, b_l^{\perp}$ of the matrix $B_1 - C$ also constitute an orthogonal set, with $\|b_i^{\perp}\|^2 = 1 - \sigma_i^2$, for $i = 1,\dots,l$. Let z_K be a point in L that minimizes the distance to K, and write $z_K = B_1 w_K + b_1$. Furthermore, for $i = 1,\dots,l$ let $\tau_i = \frac{\alpha\varepsilon}{n^{2t}\sqrt{l(1-\sigma_i^2)}}$, and $o_\tau = \left\lceil \frac{2n^{2t}\sqrt{l}}{\varepsilon^2} \right\rceil$. We use the τ_i and o_τ to define a set I of *index vectors* as $I = \prod_{i=1}^{l} \{-o_\tau \tau_i, (-o_\tau + 1)\tau_i, \dots, o_\tau \tau_i\}$. We have $|I| = O(o_\tau^l) = O((n^{2t}k^{1/2}\varepsilon^{-2})^l)$. For each index vector $i \in I$, we define the patch G_i as

$$G_i : w \mapsto Cw + B_1(w_K + i) + b_1, \text{ subject to } w \in \prod_{i=1}^{l} [0, \tau_i].$$

Our desired set of approximate query patches is now $\mathcal{G} = \{G_i \mid i \in I\}$. The set $\mathcal{G}$ fulfills properties (i) and (ii) by construction, so it remains to check (iii). Fix a point $z \in L$. Since $L \subseteq F$, we can write $z = B_1 w + b_1 = Bv + b$, where the vector w represents the coordinates of z in L and the vector v represents the coordinates of z in F. By Lemma 4.4,

$$d(z,K)^2 = \sum_{i=1}^{k} (1-\sigma_i^2)(v - v_K)_i^2 + d(F,K)^2,$$

where the vector v_K represents the coordinates of a point in F that is closest to K. By definition of L, the last $k-l$ coordinates $v_{-[l]}$ in F are the same for all points $z \in L$, so we can conclude that the coordinates for a closest point to K in L are given by $w_K = (v_K)_{[l]}$ and that

$$d(z,K)^2 = \sum_{i=1}^{l} (1-\sigma_i^2)(w - w_K)_i^2 + d(L,K)^2. \tag{1}$$

Now take a point z_Q in L with $d(z_Q, Q) = d(L,Q)$ and write $z_Q = B_1 w_Q + b_1$. Since we assumed $d(L,Q) \le 2\alpha/\varepsilon$, (1) implies that for $i = 1,\dots,l$, we have $|(w_Q - w_K)_i| \le 2\alpha/\left(\varepsilon\sqrt{1+\sigma_i^2}\right)$. Thus, if for $i = 1,\dots,l$, we round $(w_Q - w_K)_i$ down to the next multiple of τ_i, we obtain an index vector $i_Q \in I$ with $(w_Q - w_K) - i_Q \in \prod_{i=1}^{l} [0, \tau_i]$. We set $s_Q = (w_Q - w_K) - i_Q$. Considering the point $Cs_Q + B_1(u_K + i_Q) + b_1$ in G_{i_Q}, we see that

$$\begin{aligned} d(G_{i_Q}, z_Q)^2 &\le \|Cs_Q + B_1(w_K + i_Q) + b_1 - B_1 w_Q - b_1\|^2 \\ &= \|Cs_Q - B_1((w_Q - w_K) - i_Q)\|^2 \\ &= \|(C - B_1)s_Q\|^2 = \sum_{i=1}^{l} (1-\sigma_i^2)(s_Q)_i^2 \\ &\le \sum_{i=1}^{l} (1-\sigma_i^2)\tau_i^2 = \varepsilon^2\alpha^2/n^{4t}, \end{aligned}$$

using the properties of the matrix $B_1 - C$ stated above. It follows that

$$\begin{aligned} d(G_{i_Q}, Q) &\le d(G_{i_Q}, z_Q) + d(z_Q, Q) \\ &\le \varepsilon\alpha/n^{2t} + d(L,Q) \le (1+\varepsilon)d(L,Q), \end{aligned}$$

since we assumed $d(L,Q) \ge \alpha/n^{2t}$. This proves (iii). Property (iv) is obtained similarly. Let $G_i \in G$, $q \in Q$ and let z be a point in G_i. Write $z = Cw + B_1(w_K + i) + b_1$, where $w \in \prod_{i=1}^{t} [0, \sigma_i]$. Considering the point $z_x = B_1(w + w_K + i) + b_1$ in L, we see that

$$d(G_i, r_x)^2 \le \|z - z_x\|^2 = \|(C - B_1)w\| \le \varepsilon^2\alpha^2/n^{4t}.$$

Thus,

$$\begin{aligned} d(G_i, q) &\geq d(z_x, q) - d(G_i, z_x) \\ &\geq d(L, q) - \varepsilon\alpha/n^{2t} \geq (1-\varepsilon)d(L, q). \quad \square \end{aligned}$$

Finally, we have a patch $G : w \mapsto Cw + b_2$, and we are looking for an approximate nearest neighbor for G in Q. The next lemma states how this can be done.

LEMMA 4.8. *Suppose that $d(G,Q) \in [\alpha/2n^{2t}, 3\alpha/\varepsilon]$. We can find a point $\widetilde{q} \in Q$ with $d(G,\widetilde{q}) \leq (1-1/2\log n)cd(G,Q)$ in total time $O_c((k^2n^{2t}/\varepsilon^2)(m^{1-1/k+\rho/k} + (d/k)\log m))$.*

PROOF. Let G_a be the projection of G onto K, and let g be the projection of G onto $K^\perp$. Since G and K are parallel, g is a point, and G_a is of the form $G_a : w \mapsto Cw + a_2$, with $a_2 \in K$ and $w \in \prod_{i=1}^{t}[0, \tau_i]$. Let $G_a^+ = \{x \in K \mid d_\infty(x, G_a) \leq 3\alpha\sqrt{k}/\varepsilon\}$, where $d_\infty(\cdot,\cdot)$ denotes the ℓ_∞-distance with respect to the coordinate system induced by A. We subdivide the set $G_a^+ \setminus G_a$, into a collection $\mathcal{C}$ of axis-parallel cubes, each with diameter $\varepsilon\alpha/2n^{2t}$. The cubes in $\mathcal{C}$ have side length $\varepsilon\alpha/2n^{2t}\sqrt{k}$, the total number of cubes is $O((kn^{2t}/\varepsilon^2)^k)$, and the boundaries of the cubes lie on $O(k^2n^{2t}/\varepsilon^2)$ hyperplanes.

We now search the partition tree $\mathcal{T}$ to find the highest nodes (Δ, Q) in $\mathcal{T}$ whose simplices Δ are completely contained in a single cube of $\mathcal{C}$. This is done as follows: we begin at the root of $\mathcal{T}$, and we check for all children (Δ, Q) and for all boundary hyperplanes h of $\mathcal{C}$ whether the simplex Δ crosses the boundary h. If a child (Δ, Q) crosses no hyperplane, we label it with the corresponding cube in $\mathcal{C}$ (or with G_a). Otherwise, we recurse on (Δ, Q) with all the boundary hyperplanes that it crosses.

In the end, we have obtained a set $\mathcal{D}$ of simplices such that each simplex in $\mathcal{D}$ is completely contained in a cube of $\mathcal{C}$. The total number of simplices in the set $\mathcal{D}$ is $s = O((k^2n^{2t}/\varepsilon^2)m^{1-1/k})$, by Theorem 3.1. For each simplex in $\mathcal{D}$, we query the corresponding c'-ANN structure. Let $R \subseteq Q_b$ be the set of the query results. For each point $q_b \in R$, we take the corresponding point $q \in Q$, and we compute the distance $d(q, G)$. We return a point $\widetilde{q}$ that minimizes $d(q, G)$. The query time is dominated by the time for the ANN queries. For each $\Delta \in \mathcal{D}$, let m_Δ be the number of points in the corresponding ANN structure. By assumption, an ANN-query takes time $O_c(m_\Delta^\rho + d\log m_\Delta)$, so the total query time is proportional to $\sum_{\Delta\in\mathcal{D}} m_\Delta^\rho + d\log m_\Delta$ which is at most

$$\begin{aligned} s\left(\sum_{\Delta\in\mathcal{D}} m_\Delta/s\right)^\rho + sd\log\left(\sum_{\Delta\in\mathcal{D}} m_\Delta/s\right) \\ \leq O_c\left((k^2n^{2t}/\varepsilon^2)(m^{1-1/k+\rho/k} + (d/k)\log m)\right), \end{aligned}$$

using the fact that $m \mapsto m^\rho + d\log m$ is concave and that $\sum_{\Delta\in\mathcal{D}} m_\Delta \leq m$.

It remains to prove that approximation bound. Take a point q^* in Q with $d(q^*, Q) = d(Q, G)$. Since we assumed that $d(Q, G) \leq 3\alpha/\varepsilon$, the projection q_a^* of q^* onto K lies in G_a^+. Let Δ^* be the simplex in $\mathcal{D}$ with $q_a^* \in \Delta^*$. Suppose that the ANN-query for Δ^* returns a point $\widehat{q} \in Q$. Thus, in $K^\perp$, we have $d(\widehat{q_b}, g) \leq c'd(Q_{b\Delta^*}, g) \leq c'd(q_b^*, g)$, where $\widehat{q_b}$ and q_b^* are the projections of $\widehat{q}$ and q^* onto $K^\perp$ and $Q_{b\Delta^*}$ is the point set stored in the ANN-structure of Δ^*. By the definition of $\mathcal{C}$, in K, we have $d(\widehat{q_a}, G_a) \leq d(q_a^*, G_a) + \varepsilon\alpha/2n^{2t} \leq d(q_a^*, G_a) + \varepsilon d(q^*, G)$, where $\widehat{q_a}$ is the projection of $\widehat{q}$ onto K. By Pythagoras,

$$\begin{aligned} d(\widehat{q}, G)^2 &= d(\widehat{q_b}, g)^2 + d(\widehat{q_a}, G_a)^2 \\ &\leq c'^2 d(q_b^*, g)^2 + (d(q_a^*, G_a) + \varepsilon d(q^*, G))^2 \\ &\leq c'^2 d(q_b^*, g)^2 + d(q_a^*, G_a)^2 + (2\varepsilon + \varepsilon^2)d(q^*, G)^2 \\ &\leq (c'^2 + 3\varepsilon)(q^*, G)^2 \\ &\leq \left((1-1/\log n)^2c^2 + 3/100\log n\right)(q^*, G)^2 \\ &\leq (1-1/2\log n)^2c^2(q^*, G)^2, \end{aligned}$$

recalling that $c' = (1-1/\log n)c$ and $\varepsilon = 1/100\log n$. Since $d(\widetilde{q}, G) \leq d(\widehat{q}, G)$, the result follows. $\square$

Of all the candidate points obtained through querying patches, we return the one closest to F. The following lemma summarizes the properties of the query algorithm.

LEMMA 4.9. *Suppose that $d(F,Q) \in [\alpha/n^{2t}, \alpha/\varepsilon]$. Then the query procedure returns a point $\widetilde{q} \in Q$ with $d(F,\widetilde{q}) \leq cd(F,Q)$ in $O_c((k^2n^{2t}\varepsilon^{-5/2})^{k+1}(m^{1-1/k+\rho/k} + (d/k)\log m))$ total time.*

PROOF. By Lemmas 4.6 and 4.7, there exists a patch G with $d(G,Q) \leq (1+\varepsilon)^2 d(F,Q)$. For this patch, the algorithm from Lemma 4.8 returns a point $\widehat{q}$ with $d(\widehat{q}, G) \leq (1+1/2\log n)cd(G,Q)$. Thus, using Lemma 4.7(iv), we have

$$(1-\varepsilon)d(\widehat{q}, L) \leq d(\widehat{q}, G) \leq (1-1/2\log n)c(1+\varepsilon)^2 d(F,Q)$$

and by our choice of $\varepsilon = 1/100\log n$, we get

$$\begin{aligned} (1-1/2\log n)(1+\varepsilon)^2/(1-\varepsilon) &\leq (1-1/2\log n)(1+3\varepsilon)(1+2\varepsilon) \\ &\leq (1-1/2\log n)(1+6/100\log n) \leq 1. \end{aligned}$$

$\square$

4.2.2 *Far:* $d(F,Q) \geq \alpha/\varepsilon$

If $d(F,Q) \geq \alpha/\varepsilon$, we can approximate Q by its projection Q_a onto K without losing too much. Thus, we can perform the whole algorithm in K. This is done by a procedure similar to Lemma 4.8.

LEMMA 4.10. *Suppose we are given an estimate $\widetilde{r}$ with $d(F,Q_a) \in [\widetilde{r}/2n^{2t}, 2\widetilde{r}]$. Then, we can find a point $\widetilde{q} \in Q_a$ with $d(F,\widetilde{q}) \leq (1+\varepsilon)d(F,Q_a)$ in time $O((k^{3/2}n^t/\varepsilon)m^{1-1/k})$.*

PROOF. Let x_F be a point in K with $d(F,K) = d(F,x_F)$. Write $x_F = Au_F + a$. Define

$$C = \prod_{i=1}^{k}\left((u_F)_i + \left[0, 2\widetilde{r}/\sqrt{1-\sigma_i^2}\right]\right)$$

If we take a point $x \in K$ with $d(x,F) \in [\widetilde{r}/2n^t, 2\widetilde{r}]$ and write $x = Au + a$, then Lemma 4.4 gives

$$d(F,x)^2 = \sum_{i=1}^{k}(1-\sigma_i^2)(u-u_F)_i^2 + d(F,K)^2,$$

so $u \in C$. We subdivide C into copies of the hyperrectangle $\prod_{i=1}^{k}[0, \varepsilon\widetilde{r}/2n^t\sqrt{k(1-\sigma_i^2)}]$. Let $\mathcal{C}$ be the resulting set of hyperrectangles. The boundaries of the hyperrectangles in $\mathcal{C}$ lie on $O(k^{3/2}n^t/\varepsilon)$ hyperplanes. We now search the partition tree $\mathcal{T}$ in order to find the highest nodes (Δ, Q) in $\mathcal{T}$ whose simplices Δ are completely contained in a single hyperrectangle of $\mathcal{C}$. This is done as in Lemma 4.8.

This gives a set $\mathcal{D}$ of simplices such that each simplex in $\mathcal{D}$ is completely contained in a hyperrectangle of $\mathcal{C}$. The total number of simplices in $\mathcal{D}$ is $O((k^{3/2}n^t/\varepsilon)m^{1-1/k})$, by Theorem 3.1. For each simplex $\Delta \in \mathcal{D}$, we pick an arbitrary point $q \in Q_a$ that lies in Δ, and we compute $d(F, q)$. We return the point $\widetilde{q} \in Q_a$ that minimizes the distance to F. The total query time is $O((k^{3/2}n^t/\varepsilon)m^{1-1/k})$.

Now let q^* be a point in Q_a with $d(F, Q_a) = d(F, q^*)$, and let Δ^* be the simplex $\mathcal{D}$ that contains q^*. Furthermore, let $\widehat{q} \in Q_a$ be the point that the algorithm examines in Δ^*. Write $q^* = Au^* + a$ and $\widehat{q} = A\widehat{u} + a$. Since q^* and $\widehat{q}$ lie in the same hyperrectangle and by Lemma 4.4,

$$\begin{aligned} d(F,\widehat{q})^2 &= \sum_{i=1}^{k}(1-\sigma_i^2)(\widehat{u}-u_F)_i^2 + d(F,K)^2 \\ &\le \sum_{i=1}^{k}(1-\sigma_i^2)(u^*-u_F)_i^2 + \varepsilon^2\widetilde{r}^2/4n^{2t} + d(F,K)^2 \\ &\le (1+\varepsilon)^2 d(F,q^*)^2. \end{aligned}$$

Since $d(F, \widetilde{q}) \le d(F, \widehat{q})$, the result follows. $\square$

LEMMA 4.11. *Suppose we are given an estimate $\widetilde{r}$ with $d(F, Q) \in [\widetilde{r}/n^t, \widetilde{r}]$. Suppose further that $d(F, Q) \ge \alpha/\varepsilon$. Then we can find a $\widetilde{q} \in Q$ with $d(F, \widetilde{q}) \le cd(F, Q)$ in time $O((k^{3/2}n^{2t}/\varepsilon)m^{1-1/k})$.*

PROOF. For any point $q \in Q$, let $q_a \in Q$ be its projection onto K. Then, $d(q_a, q) \le \alpha \le \varepsilon d(F, Q)$. Thus, $d(F, Q_a) \in [(1-\varepsilon)d(F, Q), (1+\varepsilon)d(F, Q)]$, and we can apply Lemma 4.10. Let $\widetilde{q_a} \in Q_a$ be the result of this query, and let $\widetilde{q}$ be the corresponding point in Q. We have

$$\begin{aligned} d(F,\widetilde{q}) &\le d(\widetilde{q},\widetilde{q_a}) + d(F,\widetilde{q_a}) \le \varepsilon d(F,Q) + (1+\varepsilon)d(F,Q_a) \\ &\le \varepsilon d(F,Q) + (1+\varepsilon)^2 d(F,Q) \le (1+4\varepsilon)d(F,Q) \le cd(F,Q), \end{aligned}$$

by our choice of ε. $\square$

Combining Lemmas 4.1, 4.9, and 4.11 yields Theorem 2.1.

5. APPROXIMATE K-FLAT RANGE REPORTING IN LOW DIMENSIONS

In this section, we present a data structure for low dimensional k-flat approximate near neighbor reporting. In Section 6, we will use it as a foundation for our projection structures. The details of the structure are summarized in Theorem 5.1. Throughout this section, we will think of d as a constant, and we will suppress factors depending on d in the O-notation.

THEOREM 5.1. *Let $P \subset \mathbb{R}^d$ be an n-point set. We can preprocess P into an $O(n \log^{d-k-1} n)$ space data structure for approximate k-flat near neighbor queries: given a k-flat F and a $\alpha > 0$, find a set $R \subseteq P$ that contains all $p \in P$ with $d(p, F) \le \alpha$ and no $p \in P$ with $d(p, F) > ((4k+3)(d-k-1)+\sqrt{k+1})\alpha$. The query time is $O(n^{k/(k+1)} \log^{d-k-1} n + |R|)$.*

5.1 Preprocessing

Let $E \subset \mathbb{R}^d$ be the $(k+1)$-dimensional subspace of $\mathbb{R}^d$ spanned by the first $k+1$ coordinates, and let Q be the projection of P onto E.[1] We build a $(k+1)$-dimensional partition tree $\mathcal{T}$ for Q, as in Theorem 3.1. If $d > k+1$, we also build a *slab structure* for each node of $\mathcal{T}$. Let v be such a node, and let Ξ be the simplicial partition for the children of v. Let $w > 0$. A *w-slab* S is a closed region in E that is bounded by two parallel hyperplanes of distance w. The *median hyperplane* $\widehat{h}$ of S is the hyperplane inside S that is parallel to the two boundary hyperplanes and has distance $w/2$ from both. A w-slab S is *full* if there are at least $r^{2/3}$ simplices Δ in Ξ with $\Delta \subset S$.

[1]We assume general position: any two distinct points in P have distinct projections in Q.

Algorithm 5: CreateSearchStructure

```
   Input: point set $P \subset \mathbb{R}^d$
1  if $|P| = O(1)$ then
2      Store $P$ in a list and return.
3  $Q \leftarrow$ projection of $P$ onto the subspace $E$ spanned by
   the first $k+1$ coordinates.
4  $\mathcal{T} \leftarrow (k+1)$-dimensional partition tree for $Q$ as in
   Theorem 3.1.
5  if $d > k+1$ then
6      foreach node $v \in \mathcal{T}$ do
7          $\Xi_1 \leftarrow$ simplicial partition for the children of $v$
8          for $j \leftarrow 1$ to $\lfloor r^{1/3} \rfloor$ do
9              $D_j \leftarrow$ CreateSlabStructure($\Xi_j$)
10             $\Xi_{j+1} \leftarrow \Xi_j$ without all simplices inside the
               slab for $D_j$
```

Algorithm 6: CreateSlabStructure

```
   Input: $\Xi_j = (Q_1, \Delta_1), \dots, (Q_{r'}, \Delta_{r'})$
1  $V_j \leftarrow$ vertices of the simplices in $\Xi_j$
2  For each $(k+1)$-subset $V \subset V_j$, find the smallest
   $w_V > 0$ such that the $w_V$-slab with median hyperplane
   $\text{aff}(V)$ is full.
3  Let $w_j$ be the smallest $w_V$; let $S_j$ be the corresponding
   full $w_j$-slab and $\widehat{h}_j = \text{aff}(V)$ its median hyperplane.
4  Find the set $\mathcal{D}_j$ of $r^{2/3}$ simplices in $S_j$; let
   $\mathcal{Q}_j \leftarrow \bigcup_{\Delta_i \in \mathcal{D}_j} Q_i$ and let $\mathcal{P}_j$ be the $d$-dimensional point
   set corresponding to $\mathcal{Q}_j$.
5  $h_j \leftarrow$ the hyperplane orthogonal to $E$ through $\widehat{h}_j$
6  $P' \leftarrow$ projection of $\mathcal{P}_j$ onto $h_j$
7  CreateSearchStructure($P'$)
```

The slab structure for v is constructed in several iterations. In iteration j, we have a current subset $\Xi_j \subseteq \Xi$ of pairs in the simplicial partition. For each $(k+1)$-set $v_0, \dots, v_k$ of vertices of simplices in Ξ_j, we determine the smallest width of a full slab whose median hyperplane is spanned by $v_0, \dots, v_k$. Let S_j be the smallest among those slabs, and let $\widehat{h}_j$ be its median hyperplane. Let $\mathcal{D}_j$ be the $r^{2/3}$ simplices that lie completely in S_j. We remove $\mathcal{D}_j$ and the corresponding point set $\mathcal{Q}_j = \bigcup_{\Delta_i \in \mathcal{D}_j} Q_i$ from Ξ_j to obtain Ξ_{j+1}. Let $\mathcal{P}_j \subseteq P$ be the d-dimensional point set corresponding to $\mathcal{Q}_j$. We project $\mathcal{P}_j$ onto the d-dimensional hyperplane h_j that is orthogonal to E and goes through $\widehat{h}_j$. We recursively build a search structure for the $(d-1)$-dimensional projected point set. The jth slab structure D_j at v consists of this search structure, the hyperplane h_j, and the width w_j. This process is repeated until less than $r^{2/3}$ simplices remain; see Algorithms 5 and 6 for details.

Denote by $S(n,d)$ the space for a d-dimensional search structure with n points. The partition tree $\mathcal{T}$ has $O(n)$ nodes, so the overhead for storing the slabs and partitions is linear. Thus, $S(n,d) = O(n) + \sum_D S(n_D, d-1)$, where the sum is over all slab structures D and where n_D is the number of points in the slab structure D. Since every point appears in $O(\log n)$ slab structures, and since the recursion stops for $d = k+1$, we get

Lemma 5.2. *The search structure for n points in d dimensions needs space $O(n \log^{d-k-1} n)$.* □

5.2 Processing a Query

For a query, we are given a distance threshold $\alpha > 0$ and a k-flat F. For the recursion, we will need to query the search structure with a k-dimensional polytope. We obtain the initial query polytope by intersecting the flat F with the bounding box of P extended by α in each direction. With slight abuse of notation, we still call this polytope F.

A query for F and α is processed by using the slab structures for small enough slabs and by recursing in the partition tree for the remaining points. Details follow.

Suppose we are at some node v of the partition tree, and let j^* be the largest integer with $w_{j^*} \leq (4k+2)\alpha$. For $j = 1, \dots, j^*$, we recursively query each slab structure D_j as follows: let $\widetilde{F} \subseteq F$ be the polytope containing the points in F with distance at most $\alpha + w_j/2$ from h_j, and let F_h be the projection of $\widetilde{F}$ onto h_j. We query the search structure in D_j with F_h and α. Next, we project F onto the subspace E spanned by the first $k+1$ coordinates. Let $\mathcal{D}$ be the simplices in Ξ_{j^*+1} with distance at most α from the projection. For each simplex in $\mathcal{D}$, we recursively query the corresponding child in the partition tree. Upon reaching the bottom of the recursion (i.e., $|P| = O(1)$), we collect all points within distance α from F in the set R.

Algorithm 7: Find a superset R of all points in P with distance less than α from a query polytope F.

```
   Input  : polytope $F$, distance threshold $\alpha > 0$
   Output: point set $R \subseteq P$
1  $R \leftarrow \emptyset$
2  if $|P| = O(1)$ then
3      $R \leftarrow \{p \in P \mid d(p,F) \leq \alpha\}$
4  else if $d = k+1$ then
5      Compute polytope $F_\diamond$ as described.
6      $R \leftarrow R \cup$ all points of $P$ inside $F_\diamond$
7  else
8      $j^* \leftarrow$ the largest integer with $w_{j^*} \leq (4k+2)\alpha$
9      for $j \leftarrow 1$ to $j^*$ do
10         $F_h \leftarrow$ projection of $\widetilde{F}$ onto $h_j$ as described
11         $R \leftarrow R \cup D_j$.query($F_h$, $\alpha$)
12     $\widehat{F} \leftarrow$ projection of $F$ onto the subspace $E$ spanned
       by the first $k+1$ coordinates
13     $\mathcal{D} \leftarrow$ simplices in $\Xi_{j^*+1}$
14     $\mathcal{D}' \leftarrow \{\Delta \in \mathcal{D} \mid d(\Delta, \widehat{F}) \leq \alpha\}$
15     foreach $\Delta \in \mathcal{D}'$ do
16         $R \leftarrow R \cup$ result of recursive query to partition
           tree node for $\Delta$.
17 return $R$
```

If $d = k+1$, we approximate the region of interest by the polytope $F_\diamond = \{x \in \mathbb{R}^d \mid d_1(x,F) \leq \alpha\}$, where $d_1(\cdot,\cdot)$ denotes the ℓ_1-metric in $\mathbb{R}^d$. Then, we query the partition tree $\mathcal{T}$ to find all points of P that lie inside $F_\diamond$. We prove in Lemma 5.4 that $F_\diamond$ is a polytope with $O(d^{O(k^2)})$ facets; see Algorithm 7 for details. The following two lemmas analyze the correctness and query time of the algorithm. We refer to the full version for the proofs.

Lemma 5.3. *The set R contains all $p \in P$ with $d(p,F) \leq \alpha$ and no $p \in P$ with $d(p,F) > \kappa\alpha$, where $\kappa = (4k+3)(d-k-1) + \sqrt{k+1}$.* □

Lemma 5.4. *The query time is $O(n^{k/(k+1)} \log^{d-k-1} n + |R|)$.* □

Theorem 5.1 follows from Lemmas 5.2, 5.3, and 5.4.

5.3 Approximate k-Flat NN Queries

Now, we extend our data structure from Section 5.1 for approximate k-flat nearest neighbor queries with multiplicative error $(4k+3)(d-k-1) + \sqrt{k+1}$. That is, given an n-point set $P \subset \mathbb{R}^d$, we want to find for any given query flat $F \subset \mathbb{R}^d$ a point $p \in P$ with $d(p,F) \leq ((4k+3)(d-k-1) + \sqrt{k+1})d(P,F)$. We reduce this problem to a near neighbor query by choosing an appropriate threshold α that ensures $|R| = O(\sqrt{n})$, using random sampling. For preprocessing we build the data structure D from Theorem 5.1 for P.

Let a query flat F be given. The *F-rank* of a point $p \in P$ is the number of points in P that are closer to F than p. Let $X \subseteq P$ be a random sample obtained by taking each point in P independently with probability $1/\sqrt{n}$. The expected size of X is $\sqrt{n}$, and if $x \in X$ is the closest point to F in X, then the expected F-rank of x is $\sqrt{n}$. Set $\alpha = d(x,F)/((4k+3)(d-k-1) + \sqrt{k+1})$. We query D with F and α to obtain a set R. If $d(P,F) \leq \alpha$, then R contains the nearest neighbor. Otherwise, x is a $((4k+3)(d-k-1) + \sqrt{k+1})$-approximate nearest neighbor for F. Thus, it suffices to return the nearest neighbor in $R \cup \{x\}$. Since with high probability all points in R have F-rank at most $O(\sqrt{n}\log n)$, we have $|R| = O(\sqrt{n}\log n)$, and the query time is $O(n^{k/(k+1)} \log^{d-k-1} n)$. This establishes the following corollary of Theorem 5.1.

Corollary 5.5. *Let $P \subset \mathbb{R}^d$ be an n-point set. We can preprocess P into an $O(n \log^{d-k-1} n)$ space data structure for approximate k-flat nearest neighbor queries: given a flat F, find a point $p \in P$ with $d(p,F) \leq ((4k+3)(d-k-1) + \sqrt{k+1})d(P,F)$. The query needs expected time $O(n^{k/(k+1)} \log^{d-k-1} n)$.*

6. PROJECTION STRUCTURES

We now describe how to answer queries of type **Q1** and **Q3** efficiently. Our approach is to project the points into random subspace of constant dimension and to solve the problem there using our data structures from Theorem 5.1 and Corollary 5.5. For this, we need a Johnson-Lindenstrauss-type lemma that bounds the distortion.

Let $0 < t \leq 2/(2+40k)$ be a parameter and let $P \subset R^d$ be a high dimensional n-point set. Set $d' = 2/t + 2$ and let $M \in \mathbb{R}^{d' \times d}$ be a random projection from $\mathbb{R}^d$ to $\mathbb{R}^{d'}$, scaled by $\sqrt{d/4d'}$. We obtain $\bar{P} \subset \mathbb{R}^{d'}$ by projecting P using M. We build for $\bar{P}$ the data structure D_1 from Corollary 5.5 to answer **Q1** queries and D_2 from Theorem 5.1 to answer **Q3** queries. This needs $O(n \log^{O(d')} n) = O(n \log^{O(1/t)} n)$

space. For each $p \in P$ we write $\bar{p}$ for the d'-dimensional point Mp and $\bar{F}$ for the projected flat MF.

Dimension Reduction. The following lemma is a generalization of a variant of the Johnson-Lindenstrauss-Lemma by Dasgupta and Gupta [8, Lemma 2.2] for k-flats. Its proof is an extension of the proof of Lemma 4.1 by AIKN and can be found in the full version.

LEMMA 6.1. *Let $p \in \mathbb{R}^d$ be a point and let $F \subset \mathbb{R}^d$ be a k-flat. For $d' \in \{40k, \dots, d-1\}$, let $M \in \mathbb{R}^{d' \times d}$ be the projection matrix into a random d'-dimensional subspace, scaled by $\sqrt{d/4d'}$. Let $\bar{p} = Mp$ and $\bar{F} = MF$ be the projections of p and of F, respectively. Then, for any $\beta \geq 40k$, (i) $\Pr[\mathrm{d}(\bar{p}, \bar{F}) \leq \mathrm{d}(p, F)] \geq 1 - e^{-d'/2}$; and (ii) $\Pr[\mathrm{d}(\bar{p}, \bar{F}) \geq \mathrm{d}(p, F)/\beta] \geq 1 - \beta^{-d'/2}$.* □

6.1 Queries of Type Q1

Let a query flat F be given. To answer **Q1** queries, we compute $\bar{F}$ and query D_1 with $\bar{F}$ to obtain a $((4k+3)(d'-k-1)+\sqrt{k+1})$-nearest neighbor $\bar{p}$. We return the original point p. To obtain Theorem 2.2, we argue that if $\bar{p}$ is a $((4k+3)(d-k-1)+\sqrt{k+1})$-nearest neighbor for $\bar{F}$, then p is a n^t-nearest neighbor for F with high probability.

Let $p^* \in P$ be a point with $d(p^*, F) = d(P, F)$. Set $\delta_{p^*} = d(p^*, F)$ and $\bar{\delta}_{p^*} = d(\bar{p^*}, \bar{F})$. Denote by A_1 the event that $\bar{\delta}_{p^*} \leq \delta_{p^*}$. By Lemma 6.1, $\Pr[A_1] \geq 1 - e^{-d'/2} = 1 - e^{-1/t-1}$. Let A_2 be the event that for all points $p \in P$ with $\delta_p = d(p, F) > n^t \delta_{p^*}$ we have $\bar{\delta}_p = d(\bar{p}, \bar{F}) > ((4k+3)(d'-k-1)+\sqrt{k+1})\delta_{p^*}$. For a fixed $p \in P$, by setting $\beta = n^t/((4k+3)(d'-k-1)+\sqrt{k+1})$ in Lemma 6.1, this probability is

$$\begin{aligned}
&\Pr[\bar{\delta}_p > ((4k+3)(d'-k-1)+\sqrt{k+1})\delta_{p^*}] \\
&\geq 1 - (n^t/((4k+3)(d'-k-1)+\sqrt{k+1}))^{-d'/2} \\
&= 1 - n^{-1-t}((4k+3)(2t+1-k)\sqrt{k+1})^{1/t+1} \\
&\geq 1 - n^{-1-t/2},
\end{aligned}$$

for n large enough. By the union bound, we get $\Pr[A_2] \geq 1 - n^{-t/2}$, so the event $A_1 \cap A_2$ occurs with constant probability. Then, p is a n^t-approximate nearest neighbor for F.

6.2 Queries of Type Q3

To answer a query of type **Q3**, we compute the projection $\bar{F}$ and query D_2 with parameter α. We obtain a set $\bar{R} \subset \bar{P}$ in time $O(n^{k/(k+1)} \log^{O(1/t)} n + |\bar{R}|)$. Let $R \subset P$ be the corresponding d-dimensional set. We return a point $p \in R$ that minimizes $d(p, F)$. If $\delta_{p*} \leq \alpha$, the event A_1 from above implies that $\bar{p^*} \in \bar{R}$, and we correctly return p^*.

To bound the size of $|\bar{R}|$, and thus the running time, we use that P is $\alpha n^t/(2k+1)$-cluster-free. Let A_3 be the event that for all $p \in P$ with $d(p, F) > \alpha n^t/(2k+1)$, we have $d(\bar{p}, \bar{F}) > ((4k+3)(d'-k-1)+\sqrt{k+1})\alpha$. By the definition of cluster-freeness and the guarantee of Theorem 5.1, we have $|\bar{R}| = m$ in the case of A_3. Using $\beta = n^t/((2k+1)((4k+3)(d-k-1)+\sqrt{k+1}))$ in Lemma 6.1 and doing a similar calculation as above yields again $\Pr[A_3] \geq 1 - n^{-t/2}$. Thus, we can answer queries of type **Q3** successfully in time $O(n^{k/(k+1)} \log^{O(1/t)} n + m)$ with constant probability, as claimed in Theorem 2.3.

Acknowledgments

This work was initiated while WM, PS, and YS were visiting the Courant Institute of Mathematical Sciences. We would like to thank our host Esther Ezra for her hospitality and many enlightening discussions.

7. REFERENCES

[1] A. Andoni. *Nearest Neighbor Search: the Old, the New, and the Impossible.* PhD thesis, MIT, 2009.

[2] A. Andoni, P. Indyk, R. Krauthgamer, and H. L. Nguyen. Approximate line nearest neighbor in high dimensions. In *Proc. 20th SODA*, pages 293–301, 2009.

[3] A. Andoni, P. Indyk, H. L. Nguyen, and I. Razenshteyn. Beyond locality-sensitive hashing. In *Proc. 24th SODA*, pages 1018–1028, 2014.

[4] R. Basri, T. Hassner, and L. Zelnik-Manor. Approximate nearest subspace search with applications to pattern recognition. In *Proc. CVPR*, pages 1–8, 2007.

[5] T. M. Chan. Optimal partition trees. *Discrete Comput. Geom.*, 47(4):661–690, 2012.

[6] B. Chazelle. *The discrepancy method. Randomness and complexity.* Cambridge University Press, 2000.

[7] K. L. Clarkson. A randomized algorithm for closest-point queries. *SICOMP*, 17(4):830–847, 1988.

[8] S. Dasgupta and A. Gupta. An elementary proof of a theorem of Johnson and Lindenstrauss. *Random Structures Algorithms*, 22(1):60–65, 2003.

[9] R. A. Horn and C. R. Johnson. *Matrix analysis.* Cambridge University Press, second edition, 2013.

[10] P. Indyk. Nearest neighbors in high-dimensional spaces. In J. E. Goodman and J. O'Rourke, editors, *Handbook of Discrete and Computational Geometry*, chapter 39. CRC Press, 2nd edition, 2004.

[11] P. Indyk and R. Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proc. 30th STOC*, pages 604–613, 1998.

[12] E. Kushilevitz, R. Ostrovsky, and Y. Rabani. Efficient search for approximate nearest neighbor in high dimensional spaces. In *Proc. 30th STOC*, pages 614–623, 1998.

[13] Q. Lv, W. Josephson, Z. Wang, M. Charikar, and K. Li. Multi-probe LSH: Efficient indexing for high-dimensional similarity search. In *Proc. 33rd VLDB*, pages 950–961, 2007.

[14] A. Magen. Dimensionality reductions in ℓ_2 that preserve volumes and distance to affine spaces. *Discrete Comput. Geom.*, 38(1):139–153, 2007.

[15] S. Mahabadi. Approximate nearest line search in high dimensions. In *Proc. 26th SODA*, pages 337–354, 2015.

[16] J. Matoušek. Efficient partition trees. *Discrete Comput. Geom.*, 8(3):315–334, 1992.

[17] S. Meiser. Point location in arrangements of hyperplanes. *Inform. and Comput.*, 106(2):286–303, 1993.

[18] R. Panigrahy. Entropy based nearest neighbor search in high dimensions. In *Proc. 17th SODA*, pages 1186–1195, 2006.