Project number IST-25582

CGL

Computational Geometric Learning

Approximate Nearest Neighbor Search for Points on Lower Dimensional Flats

STREP

Information Society Technologies

Period covered:	February 1, 2013–October 31, 2013
Date of preparation:	November 14, 2013
Date of revision:	November 14, 2013
Start date of project:	November 1, 2010
Duration:	3 years
Project coordinator name:	Joachim Giesen (FSU)
Project coordinator organisation:	Friedrich-Schiller-Universität Jena Jena, Germany

Approximate Nearest Neighbor Search for Points on Lower Dimensional Flats

Ioannis Z. Emiris * Ioannis Psarros* Ken Chanseau Saint-Germain†

Abstract

In order to improve efficiency in Approximate Nearest Neighbor search, we exploit the structure of the input data by considering points that are distributed almost uniformly on flats of varying, lower dimension, where these flats are distributed uniformly within a bounding sphere. We exploit an existing mapping that transforms Nearest Flat search to Nearest point search. We achieve expected query time logarithmic in the number of flats; the squared bound holds with high probability, under mild extra assumptions. This work extends [EKKNTF12, Sec.3] which considered collinear points. We also introduce a CGAL-based implementation and corroborate our theoretical results with experimental evidence. Our software is faster than CGAL kd-trees: e.g. for 5K points in 55 dimensions, or 30K points in 20 dimensions, with both sets lying on 2-dimensional flats, our code is about 5 times faster.

1 Introduction

Nearest neighbor searching (NNS), also known as similarity searching, is a fundamental computational problem that has significant applications and is the focus of a lot of current projects, e.g. [AIKN09, AMM09, ML09, BHZM11]. One approach to improve the query performance is to exploit the fact that in practice data exhibits highly nonrandom patterns, such as spatial locality. We consider the problem of approximate nearest neighbor, when the dataset of points lie on some affine spaces of lower dimension. This has applications in many areas, including machine learning, geometric inference, and high-dimensional optimization, as well as in computer vision. For instance, in motion segmentation [VH04], point trajectories associated with the same motion, lie on the same low dimensional affine subspace. The well-studied problem of subspace clustering [Vid11], concerns the problem of recognizing these affine subspaces. In [BHZM11], they study the problem of searching for the nearest affine subspace with respect to a given query and they present several applications where the datasets consist of affine subspaces. It is also observed that points may lie in low doubling dimension. Datasets with this property are investigated in [FDKV07], in the context of manifold learning.

Previous Work. Let us consider a set of n points in $\mathbb{R}^d$ and $\mathrm{dist}(p, p')$ be the Euclidean distance between points p, p'. The problem consists in reporting, given query point q, a point p such that $\mathrm{dist}(p, q) \leq \mathrm{dist}(p', q)$, for all input points p'; p is a nearest neighbor of q. An exact solution in high dimension and sublinear time requires heavy resources. For instance, Voronoi diagrams require $O(n^{\lceil d/2 \rceil})$ space, while [Cla88] presented an algorithm with $O(n^{\lceil d/2 \rceil (1+\epsilon)})$ space

*National and Kapodistrian University of Athens, Department of Informatics and Telecommunications, Athens, Greece, {emiris, i.psarros}@di.uoa.gr

†cole Normale Suprieure, Dpartement d'informatique, Paris, France, {chanseau}@ens.fr

and $O(\log n)$ query time for any fixed ϵ. In [Mei93], point location in hyperplane arrangements takes $O(d^5 \log n)$ time, using $O(n^{d+\delta})$ space, for arbitrary $\delta > 0$.

Thus, many techniques focus on computing the approximate nearest neighbor (ANN). Given parameter ϵ, a $(1+\epsilon)$-ANN to q is input point p such that $\text{dist}(q,p) \leq (1+\epsilon)\text{dist}(q,p')$, for all input points p'. In [AMN+98] they introduced the Balanced Box-Decomposition (BBD) tree with query time $O(c \log n)$ with $c \leq d/2\lceil 1 + 6d/\epsilon\rceil^d$ and space $O(dn)$. It is implemented in `ANN`, a state-of-the-art library. A classic data structure is kd-trees [Ben75]. Even if they are successfully used for ANN, they require linear time query for NN in the worst case. A successful implementation of randomized kd-trees is FLANN (Fast Library for Approximate Nearest Neighbor), but lacks theoretical guarantees [ML09]. Approximate Voronoi Diagrams (AVD) establish a tradeoff between space complexity and query time [AMM09]. With a tradeoff parameter $2 \leq \gamma \leq \frac{1}{\epsilon}$, the query time is $O(\log(n\gamma) + \frac{1}{(\epsilon\gamma)^{\frac{d-1}{2}}})$ and the space is $O(n\gamma^{d-1} \log \frac{1}{\epsilon})$. They were implemented in [EKKNTF12] using a quadtree. Improvements to the space-time tradeoff are obtained in [AdFM11].

A popular algorithm in high dimensional spaces is locality sensitive hashing (LSH). In [AI08] they presented an algorithm that almost matches the lower bound of hashing algorithms proved in [MNP07]. Their algorithm achieves query time $O(dn^{\epsilon^{-2}+o(1)})$ using $O(dn+n^{1+\epsilon^{-2}+o(1)})$ space.

An interesting generalization of the problem arises if we replace the pointset with a set of disjoint polyhedra [KS02], a set of triangles, segments, and points in convex position [Wan08], or a set of parallel segments [EMT10], all in 3 dimensions.

Recently, there have been results for the problem where the queries are lines [AIKN09], or the dataset is allowed to be a set of affine subspaces [BHZM07], or both dataset and queries are affine subspaces [BHZM11]. The latter provide a mapping the affine subspaces in $\mathbb{R}^d$ to points in $\mathbb{R}^{O(d^2)}$ while the distance between any subspace and any query is preserved up to an additive and a multiplicative factor. We detail and use this method, and also correct a mistake in Sec. 2.1. The notion of "flat" is equivalent to the notion of "affine subspace". A "k-flat" refers to an affine subspace of dimension k.

Exploiting Structure. One approach to improve performance is to exploit nonrandom patterns of the queries. In [AMMW07, CDI+08] it was shown how to answer efficiently planar point location queries with temporal locality in optimal expected-case time; in [IL03], they gave spatially adaptive methods. For higher dimensions there are very few results. The most relevant is [DSWS08], where they showed how to answer ANN queries in a distance-sensitive manner.

We exploit characteristics of the dataset. For uniformly distributed pointsets, [BWY80] employs a grid decomposition that achieves $O(1)$ expected search time for exact NNS, in general dimension. However, the hidden constant in query time is exponential in the dimension, and space usage is also exponential, which makes the method impractical.

In [MM01], they show that for points uniformly distributed on a single k-flat (whereas we consider several), a variant of kd-trees has expected query time exponential in k instead of exponential in d [FBF77]. This query time concerns the number of cells accessed and not the time to traverse the tree, which is expected to be $O(\log n)$. During a split their algorithm considers the divided bounding box as it would have been if obtained by midpoint splits. Then, it finds the next splitting hyperplane, according to the midpoint splitting method. If this split creates an empty subregion, the algorithm slides the new hyperplane to the non-empty side until it passes through one point.

In [Mag02] the author extends low distortion embeddings to embeddings that preserve additional properties such as volume, angles and distances between points and affine spaces. He proposes an algorithm for Nearest Affine Neighbor with query time in $O(d^{10} \log m)$ and pre-

processing time and space in $O(m^{d^2})$, where m is the number of affine subspaces. However the method by [BHZM11] reduces the problem to ANN in $O(d^2)$ dimensions and needs additional preprocessing time of $O(mkd^2)$. Combining [Mei93] with [BHZM11] leads to an algorithm with query time $O(d^{10}\log m)$ and space $O(m^{d^2})$, as in [Mag02] .

A significant amount of work has been done for pointsets with low doubling dimension.

Definition 1. *The doubling dimension of a metric space M is the smallest positive integer k such that every set S with diameter D_S can be covered by 2^k sets of diameter $D_S/2$.*

In [HPM05], they provide an algorithm for the ANN problem with expected preprocessing time $O(2^k n \log n)$, space $O(2^k n)$ and query time $O(2^k \log n + \epsilon^{-O(k)})$ for pointsets of doubling dimension k. In [IN07] they provide randomized embeddings that preserve nearest neighbor with constant probability, for points lying on low doubling dimension subspaces. Naturally, such an approach can be easily combined with any known data structure for ANN. BBD-trees, would achieve $O(c \log n)$ query time, where $c \leq k'/2\lceil 1 + 6k'/\epsilon\rceil^{k'}$ and $O(k'n)$ space, where $k' = O(k\log(1/\epsilon)/\epsilon)$. In this case we have better dependence on the dimension, for the space complexity, comparing with [HPM05].

In [DF08] they present a data structure similar to kd-trees which adapts to pointsets of low intrinsic dimension. They call this data structure random projection trees and it is based on splitting the space with respect to a random direction, not necessarily along the coordinates. They show that all descendant cells at least $k\log k$ levels below a given cell C with data of intrinsic dimension k, will have at most half the diameter of C. The efficiency of rp-trees for ANN is not analyzed but its query time is probably exponential in k. In [FDKV07], they provide experimental evidence that rp-trees work efficiently for manifold learning. In our model, points lie in low doubling dimension as they lie on affine subspaces of low dimension. However, we exploit both the low intrinsic dimension property and the "uniformness" of the dataset in order to obtain expected query time logarithmic in the number of flats.

Our contribution. We know that the points in $\mathbb{R}^d$ lie on some flats and assume that flats are of small dimension, so that we can learn them during preprocessing. This is the subspace clustering problem [Vid11]. Some notable algorithms are Random Sampling Consensus [FB81],based on randomly sampling points and fitting a flat to them, and Local Subspace Affinity [YP06], based on the observation that points and their neighbors are more probable to lie on the same flat. These algorithms perform well in practice but lack theoretical guarantees. A relevant work is [HPV02], where they compute m flats which approximately minimize $\max_p \min_{i\leq m} \mathrm{dist}(p, f_i)$ in $O(dn^{(k^6 m/\epsilon^5 \log(1/\epsilon))})$.

Given a query point $q \in \mathbb{R}^d$ and approximation factor ϵ, we seek point v which is almost nearest to q. Let $m \in \mathbb{N}$ be the number of flats and $m_i \in \mathbb{N}$ the number of points on k_i-flat f_i for $i \in \{1, \ldots, m\}$. We generalize [EKKNTF12, Sec.3] which considered only lines.

We assume the following conditions:

1. For all i, m_i points are picked uniformly at random on k_i-flat f_i.
2. The m flats are picked uniformly at random in a bounding sphere B (see Rmk 7).
3. For all i, $m_i = \Omega(m^{k_i})$.
4. Queries are not closer than $\lambda > 0$ to any flat, otherwise an additive error may exist.

Assumptions (1), (2) are used in the expected case analysis. A lower bound on the number of points per flat seems natural, since flats with arbitrarily small number of points remove the structural characteristics. In particular, assumption (3) is crucial in the expected case analysis

and is not very restrictive, especially when m is small; still, we can relax it. Alternatively, we may assume that m correlates with n, e.g. $m = \Theta(\log n)$ yields expected query time $O(\log \log n)$. By assumption (4), if q is closer than λ to a flat, then we cannot guarantee a $1+\epsilon$ approximation factor but, instead, there is an additive error in the computed distance as well.

The input points are generated as follows. A bounding sphere B is chosen and m flats that intersect the sphere are picked uniformly at random as in Rmk 7. The intersection of each k_i-flat f_i with the bounding sphere B, is a hypersphere of dimension k_i. On this hypersphere, m_i points are independently and identically distributed (i.i.d.) according to the uniform distribution.

The rest of the paper is organized as follows. In Sec. 2 we present the mapping from [BHZM11], we make a small correction to that paper, and prove our algorithm returns a $(1+\epsilon)$-ANN. In Sec. 2.2 we establish the complexity of our method; the query is achieved in expected time logarithmic in the number of flats, assuming the ambient dimension is fixed, see Thm 13. With high probability, this bound is at most squared, see Thm 15. In Sec. 4 we discuss our experiments and in the Appendix the implementation.

2 The Algorithm

The algorithm first finds the flats on which the points lie. When a query point is given, it finds the N nearest flats to the query. For each of the N flats, the algorithm finds the nearest point u_i to the query. It returns the nearest point to the query among the $u_1, \ldots, u_N$.
Preprocessing:

P1 Find the flats on which the points lie and for each flat store its points.

P2 Map each flat to a point in d'-dimensional space, $d' > d$, and construct a dynamic ANN-structure.

Query steps:

Q1 Given query $q \in \mathbb{R}^d$, map it to the d'-dimensional space and its ANN.

Q2 Remove this point from the ANN-structure.

Q3 Find the flat in the original space that corresponds to the point.

Q4 Among the points contained in this flat, find the (approximate) nearest neighbor to q and compute its distance, denoted by ρ.

Q5 Repeat the above procedure:

- **a)** If the new distance $< \rho$, update ρ with the new distance and update the current nearest neighbor.
- **b)** If the flat found is at distance $> \rho$, return the current nearest neighbor.

In **P1** we compute the k-flats $\{f_1, \ldots, f_m\}$: Iteratively pick a $(k+1)$-tuple of points and check how many of the other points lie on the affine subspace defined by the $(k+1)$-tuple. We keep the flats that contain at least $\Omega(m^k)$ points. We continue the iteration for each possible value of k. The algorithm constructs an ANN-structure per flat. In **P2** we map the flats to a set of points $S \subset \mathbb{R}^{d'}$ where $d' = O(d^2)$. The $(1+\epsilon_1)$-approximate nearest flat in $\mathbb{R}^d$ is obtained from the $(1+\epsilon_1')$-ANN in $\mathbb{R}^{d'}$. Approximation ϵ_1' must be small enough as specified in Cor. 4, in relation to ϵ_1, λ, which lower bounds the distance between any query and any flat, and constant ν dependent on d, R (see Sec. 2.1).

Given query q in **Q1** the algorithm maps it to $q' \in \mathbb{R}^{d'}$, then finds its $(1+\epsilon_1')$-ANN $s_1 \in S$. In **Q2** we remove s_1 from the ANN-structure: BBD-trees allow the deletion of a point in $O(\log m)$, where m is the number of points in the tree. Point s_1 corresponds to flat f_{i_1} which is a $(1+\epsilon_1)$-approximate nearest flat to q. In **Q3**, we find this flat and compute the distance ρ_1 from q to it. In **Q4**, we find $u_1 \in f_{i_1}$ closest to the projection of q. Although f_{i_1} is almost nearest to q, this is not necessarily the case for u_1. In **Q5**, we find N approximate nearest flats and, for each, we repeat the above procedure. Let ρ_j be the distance between q and f_{i_j}, and let $u_j \in f_{i_j}$ be nearest to q. The algorithm stops when

$$\min_{j}^{N}\{\mathrm{dist}(q, u_j)\} \leq \rho_N.$$

We prove in Sec. 2.2 that the algorithm returns an $(1+\epsilon)$-ANN, $\epsilon = \max\{\epsilon_1, \epsilon_2\}$. We prove in Sec. 3 that $\mathbb{E}(N) = O(1)$. To guarantee correctness, we do not set N a priori but instead we find nearest flats until the $(1+\epsilon)$ approximation factor is obtained.

2.1 Mappings

For completeness, we describe the mapping from [BHZM11] and we correct it in (2). Let f_i an affine subspace of $\mathbb{R}^d$, $(u_i)_{1\leq i\leq k}$ an orthonormalised base of f_i, and $(u_i)_{k+1\leq i\leq d}$ an orthonormalised base of the orthogonal subspace of f_i.

Let $S = [u_1, \ldots, u_k] \in M_{d,k}$ and $Z = [u_{k+1}, \ldots, u_d] \in M_{d,d-k}$. The mapping needs to evaluate $ZZ^T \in M^{d,d}$. But SS^T (resp. ZZ^T) is the matrix of the orthogonal projector on f_i (resp. the orthogonal subspace of f_i), so $ZZ^T = I_d - SS^T$. Since k is relatively small, we calculate ZZ^T with S. Since $(u_i)_{k+1\leq i\leq d}$ is an orthonormal basis, we observe

$$Z^T Z = I_{d-k}. \tag{1}$$

Let $t \in \mathbb{R}^{d-k}$ be the offset vector of f_i in its orthogonal subspace. Let $\hat{Z} = \begin{bmatrix} Z \\ t^T \end{bmatrix}$. Then,

$$\hat{Z}\hat{Z}^T = \begin{bmatrix} Z & Zt \\ t^T Z^T & \|t\|^2, \end{bmatrix}$$

where $\|\cdot\|$ denotes Euclidean norm. Let $d' = \frac{d(d+1)}{2}$ and define map $h : \mathcal{S}^d \to \mathbb{R}^{d'}$ as follows:

$$\forall A = (a_{ij})_{1\leq i,j\leq d} \in S^d, \qquad h(A) = (\frac{a_{11}}{\sqrt{2}}, a_{12}, \cdots, \frac{a_{22}}{\sqrt{2}}, a_{23}, \cdots).$$

h transforms a symmetric matrix into a vector. Then, we introduce M such that the following square root is real, where $\|\cdot\|_F$ denotes Frobenius norm:

$$\hat{c}(A) = \sqrt{\frac{M^4 - \|\hat{Z}\hat{Z}^T\|_F^2}{2}}.$$

There is a mistake in the value of $\hat{c}(A)$ in [BHZM07, BHZM11]:

$$\|\hat{Z}\hat{Z}^T\|_F^2 = \|ZZ^T\|_F^2 + 2\|Zt\|^2 + \|t\|^4 = d - k + 2\|t\|^2 + \|t\|^4, \tag{2}$$

instead of $d - k + 3\|t\|^2$. It suffices that $M > (d - k + 2\|t\|^2 + \|t\|^4)^{\frac{1}{4}}$ for all t, e.g., $M = (d + 2R^2 + R^4)^{\frac{1}{4}}$ with R the radius of the bounding ball.

Finally the mapped point for f_i is $u = -[h(\hat{Z}\hat{Z}^T), \hat{c}(A)]^T \in \mathbb{R}^{d'+1}$. We define the orientation of the offset vector from the origin to the affine subspace, so we use $\hat{q} = [q^T, -1]^T$ instead of $\hat{q} = [q^T, 1]^T$. The image of q is $v = [h(\hat{q}\hat{q}^T), 0] \in \mathbb{R}^{d'+1}$.

Proposition 2. [BHZM11] *Assume that f_i is a k-flat mapped onto a point u_i and the query point q is mapped to v. It holds: $dist^2(u_i, v) = \mu \ \mathrm{dist}^2(f_i, q) + \nu$.*

For our mapping, $\mu = 1$ unlike more general settings in [BHZM11], where they present improvements for NNS in d' dimensions.

Proof. Let $(z_{i,j})_{1\leq i,j\leq d} = ZZ^T$. We have $u^T v =$

$$-\left[h(\hat{Z}\hat{Z}^T), \hat{c}(A)\right] \begin{bmatrix} h(\hat{q}\hat{q}^T) \\ 0 \end{bmatrix} = -\frac{\sum_{1\leq i,j\leq d} z_{ij} q_i q_j}{2} + \sum_{1\leq i\leq d} q_i (Zt)_i + \|Zt\|^2 = \frac{-q^T ZZ^T q - \|Zt\|^2 + 2q^T Zt}{2}$$

Now (1) implies $q^T ZZ^T q = q^T ZZ^T ZZ^T q = \|ZZ^T q\|^2$. Hence,

$$u^T v = -\frac{1}{2}(\|ZZ^T q\|^2 + \|Zt\|^2 - 2(ZZ^T q)^T Zt = -\frac{1}{2}\|ZZ^T(q - Zt)\|^2 = -\frac{1}{2}\,\mathrm{dist}^2(q, f_i).$$

Hence

$$\|u - v\|^2 = \|u\|^2 + \|v\|^2 - 2u^T v = \mathrm{dist}^2(q, f_i) + \frac{1}{2}(M^4 + \|\hat{q}\|^4). \tag{3}$$

□

In (3), it is impossible to have $u = v$ because $M > 0$, so the set of mapped affine subspaces is disjoint from the the set of mapped query points. Furthermore, the additional constant only depends on q and is independent of f_i. Hence, the mapping is valid for affine subspaces of varying dimension.

Corollary 3. *The aforementioned mapping is injective, i.e. the mapped elements are unique.*

Proof. Let $u \in \mathbb{R}^{d'+1}$.

- If $\exists q \in \mathbb{R}^d : [h(\hat{q}\hat{q}^T), 0]^T = u$, the only possibility is that: $\exists q' \in \mathbb{R}^d \ / \ [h(\hat{q'}\hat{q'}^T), 0]^T = u$. h is bijective so $\hat{q}\hat{q}^T = \hat{q'}\hat{q'}^T$, therefore $q = q'$.
- If $\exists \mathcal{A} : -[h(\hat{Z}\hat{Z}^T), \hat{c}(\mathcal{A})]^T = u$, with $Z = [u_{k+1}, \ldots, u_d]$, where $(u_i)_{k+1\leq i\leq n}$ is an orthonormal basis of the linear orthogonal subspace of $\mathcal{A}$, then the only possibility is that there exists an affine subspace $\mathcal{A}'$ such that

$$u = -[h\left(\begin{bmatrix} P & t' \\ t'^T & \|t'\|^2 \end{bmatrix}\right), \sqrt{\frac{M^4 - (d - k + 2\|t'\|^2 + \|t'\|^4)}{2}}]^T,$$

  where P is the orthogonal projector of the linear orthogonal subspace of $\mathcal{A}'$ and t' is the offset vector; h bijective so we obtain $P = ZZ^T$ and t' equals the offset vector of $\mathcal{A}$. An affine subspace is entirely defined by these two elements, so $\mathcal{A} = \mathcal{A}'$.

□

2.2 Correctness

Corollary 4. *Query q and flat f_i are mapped to v, u_i respectively. Let T_1 be the minimum distance between q and a flat, which is lower bounded by $\lambda > 0$, and T'_1 be the distance between v and its exact nearest neighbor in $\mathbb{R}^{d'}$. The approximation errors are ϵ_1, ϵ'_1, namely $\mathrm{dist}(f_i, q) \leq (1 + \epsilon_1)\mathrm{T}_1$, and $\mathrm{dist}(u_i, v) \leq (1 + \epsilon'_1)\mathrm{T}'_1$. It suffices to set $\epsilon'_1 > 0$ such that*

$$\epsilon'_1 \ \leq \ \frac{1}{6}\,\epsilon_1\,\frac{\lambda^2}{\nu}.$$

Recall that $2\nu = M^4 + \|(q,1)\|^4$ so $\nu \leq d/2 + 2R^4$, with R the radius of the bounding ball.

Proof. Prop. 2 implies $\mathrm{T}_1'^2 = \mathrm{T}_1^2 + \nu$ because $\mu = 1$. Then,

$$\mathrm{dist}^2(u_i, v) = \mathrm{dist}^2(f_i, q) + \nu \implies (1+\epsilon_1')^2 \mathrm{T}_1'^2 \geq \mathrm{dist}^2(f_i, q) + \nu \implies$$

$$\implies (1+3\epsilon_1')(\mathrm{T}_1^2 + \nu) \geq \mathrm{dist}^2(f_i, q) + \nu \implies (1+3\epsilon_1')\mathrm{T}_1^2 + 3\epsilon_1'\nu \geq \mathrm{dist}^2(f_i, q) \implies$$

$$\implies (1 + 3\epsilon_1' + 3\epsilon_1' \frac{\nu}{\lambda^2})\mathrm{T}_1^2 \geq \mathrm{dist}^2(f_i, q) \implies (1+\epsilon_1)\mathrm{T}_1 \geq \mathrm{dist}(f_i, q).$$

Hence it suffices to set $\epsilon_1' > 0$ such that

$$1 + \epsilon_1 \geq \sqrt{1 + 3\epsilon_1'(1 + \frac{\nu}{\lambda^2})}.$$

□

Theorem 5. *Our algorithm, with the above notation, returns a $(1+\epsilon)$-approximate nearest neighbor, where $\epsilon = \max\{\epsilon_1, \epsilon_2\}$.*

Proof. Recall we use a dynamic ANN-structure for finding the $(1+\epsilon_1)$-approximate nearest flat, and we compute the $(1+\epsilon_2)$-approximate nearest point per flat; the distance to the latter is denoted by ρ. The algorithm then searches for a nearest point on the next nearest flat and updates ρ accordingly, as long as the next nearest flat lies at distance $> \rho$. If the closest point to q lies at distance T_1 on a flat that has not been examined, then Cor. 4 implies $\rho \leq (1+\epsilon_1)\mathrm{T}_1$. If the closest point p_2 lies at distance T_2 on an examined flat f_i, then $\rho \leq (1+\epsilon_2)\mathrm{T}_2$ because

$$\rho^2 = \mathrm{dist}^2(q, f_i) + (1+\epsilon_2)^2 \,\mathrm{dist}^2(\mathrm{proj}_{f_i}(q), p_2) \implies \rho \leq (1+\epsilon_2)\mathrm{T}_2.$$

The minimum distance to any point is $\mathrm{T} = \min\{\mathrm{T}_1, \mathrm{T}_2\}$, hence $\rho \leq (1+\epsilon)\mathrm{T}$. □

3 Time Complexity

The runtime of our algorithm depends on the ANN-structures we choose. We need an ANN-structure for pointset S so as to find the approximate nearest flat. If we assume that d is small in comparison to n, the best data structures are the BBD-trees [AMN+98] and the AVD [AMM09]. The advantage of the BBD-trees is that they are far more practical and we can remove or insert a point to the structure in $O(\log m)$ time where m is the number of points inserted to the structure. Thus, we will be able to remove the point that corresponds to the approximate nearest flat after we find it. On the other hand the advantage of the AVD is the guaranteed tradeoff between space and time but in order to use it we need a strict bound on the number of iterations. Provided that the flats are picked at random in the bounding sphere there is no such bound. Thus, we will use the BBD-trees data structure which has query time $O(c \log m)$ with $c \leq d'/2\lceil 1 + 6d'/\epsilon_1' \rceil^{d'}$ and space $O(d'm)$ where m is the number of flats i.e. the number of points inserted to the structure and $d' = O(d^2)$. Notice that ϵ_1 and ϵ_1' are not the same according to the discussion in Sec. 2.2. In the preprocessing step finding the set of flats costs $\sum_{j=1}^{\max_i k_i} \binom{n}{j} \cdot O(n)$ time if we do it naively.

Now, regarding the problem of finding the point which is nearest to $\mathrm{proj}_{f_i}(q)$ on any flat f_i, we apply the idea of [BWY80]. We decompose each of our k-flats in cells, requiring $O(n)$ space. Then, finding an exact NN on a given flat requires $O(1)$ expected time if $k_i = O(1), i \in \{1, \ldots, m\}$. Hence, the overall expected query time is $O(N \cdot c \log m)$, where N denotes the number of iterations, which are necessary in order to reassure that there is no potential nearest

neighbor in any unchecked flat. We will see below that in the expected case N is constant and it also behaves well with high probability.

In order to control the worst case, we also keep an ANN-structure with approximation factor ϵ_2 per flat. Assuming that this structure has worst-case query time $\mathcal{Q}$, the overall query time is $O(N \cdot c \log m + \mathcal{Q})$. For instance, choosing BBD-trees for all flats demands $O(kn)$ space, where $k = \max_i k_i$ and the query time costs $O(c' \log n)$, where $c' \leq k/2\lceil 1 + 6k/\epsilon_2 \rceil^k$. In addition, in the worst case $N = m$, which means the overall query time is $O(m \cdot (c \log m + c' \log n))$.

Remark 6. We say a flat lies *inside a hyperannulus* if it intersects with the hyperannulus but not with the inner hypersphere.

Remark 7. We pick a k_i-flat f_i uniformly at random in a bounding sphere B of radius R as follows. We pick uniformly at random a unit length vector $e_1 \in \mathbb{R}^d$ and similarly a scalar $\alpha \in [0, R]$. Let point b lie on the line defined by e_1 at distance α from the origin. Then we pick a random subspace in the perpendicular hyperplane of e_1 through b by picking k_i independent uniform unit vectors perpendicular to each other, so as to define the basis of f_i.

The following are proven in [EKKNTF12, Sec.3].

Lemma 8. [EKKNTF12, Sec.3] *The probability of a flat, uniformly distributed as in Rmk 7, to lie inside a hyperannulus depends only on the width of the hyperannulus (and not on the radii of the spheres which define it).*

Corollary 9. [EKKNTF12, Sec.3] *If a hyperannulus of width r is contained in a bounding sphere of radius R and a flat is picked uniformly at random inside the bounding sphere as in Rmk 7, the probability that the flat lies in the hyperannulus is r/R.*

We examine annuli not co-centric with the bounding ball.

Lemma 10. *The probability of a flat, uniformly distributed as in Rmk 7, to lie in a hyperannulus of width r, centered at any point inside the bounding ball of radius R is less or equal to the probability of the flat to lie in a hyperannulus of width r centered at the origin.*

Proof. Consider two annuli, centered at the origin and at point c, both with the same radius of inner sphere and same width $\mathrm{d}r$, where $\mathrm{d}(\cdot)$ denotes the differential operator. We determine which of the two annuli is more probable to be touched by a random $(d-1)$-flat. Let point p_1 be on the surface of the sphere centered at the origin and the corresponding point p_2 on the surface of the other sphere s.t. $p_2 - c = p_1$. We reduce our question, to the question: which of p_1, p_2 is more probable to be touched by the $(d-1)$-flat?

Let us recall the distribution of flats. At first, we choose a random vector e_1 as direction of the offset. If e_1 has not the same direction as p_1, then the flat cannot touch either p_1 or p_2. So assume e_1 has the same direction as p_1. Then, we choose uniformly $\alpha \in [0, R]$. If $\alpha = |p_1|$, we call this event A and the new flat touches the sphere at p_1.

In order for the $(d-1)$-flat to touch the second hypersphere at p_2, e_1 must have the same direction as p_2. In addition α must be equal to the distance of the tangent space from the origin. We call this event B, and obviously $P(A) = P(B)$ because α is chosen uniformly at random. Consequently, a random $(d-1)$-flat touches an annulus of width $\mathrm{d}r$, centered at c with probability equal with that of touching an annulus of width $\mathrm{d}r$ centered at the origin. With an integration implied, a random $(d-1)$-flat lies into an annulus of width r centered at any point c with probability equal to that of an annulus of width r centered at the origin.

When we consider a k-flat for $k < d-1$, there is the additional step of generating a random basis for this flat. If αe_1 lies inside the hyperannulus centered at the origin, then the flat

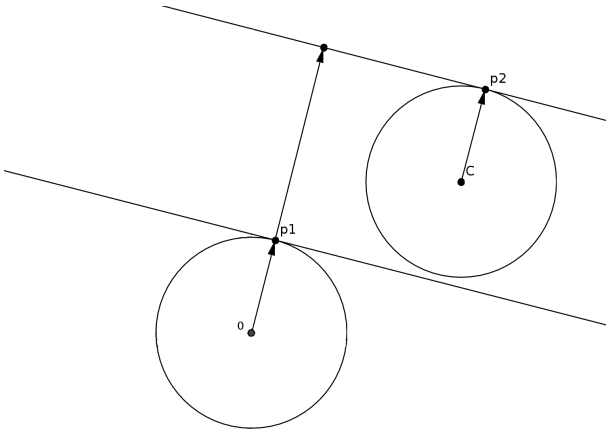

Figure 1: Two parallel $(d-1)$-flats are chosen with the same probability.

certainly intersects the hyperannulus. If αe_1 defines a $(d-1)$-flat, as above, which intersects a hyperannulus centered at c, then the k-flat intersects the hyperannulus depending on c, αe_1 and the choice of the basis for the k-flat. Hence, a random k-flat, for $k < d-1$, lies into an annulus of width r centered at any point c with probability less or equal to that for an annulus of width r centered at the origin. □

We now bound the expected number of flats in an annulus.

Lemma 11. *The expected number of flats, that the algorithm computes, is constant, assuming the projection of q to the first one is at the center of this flat.*

Proof. Let R_i be the radius of hypersphere H_{R_i} defined by the set theoretic intersection $f_i \cap B$, $i \in \{1, \ldots, m\}$; $\mathbb{E}(s_i)$ is the expected distance between point q lying on $f_i \cap B$ and its nearest neighbor on $f_i \cap B$. For any $H \subset \mathbb{R}^k$, let $\mathrm{V}(H)$ be its k-dimensional volume. Let f_u be the k_u-flat such that $u = \arg_i \max \mathbb{E}(s_i/R_i)$. We assume that $\mathrm{V}(H_{R_u}) = 1$ wlog, because we only care about the ratio s_i/R_i which is invariant under scaling.

In [BC08], they show that the expected distance between the center of a hypersphere of unit volume and its nearest neighbor among m_u points uniformly distributed in the hypersphere is

$$\mathbb{E}(s_u) = R_u \cdot \frac{\Gamma(1+\frac{1}{k_u})\Gamma(m_u+1)}{\Gamma(m_u+1+\frac{1}{k_u})}),$$

where R_u is the radius of the unit volume hypersphere. Using Gautschi's inequality [Gau60] and the fact that for $k_u \geq 1$, $\Gamma(1+\frac{1}{k_u}) \leq 1$, we obtain

$$\mathbb{E}(\frac{s_u}{R_u}) = \frac{\Gamma(1+\frac{1}{k_u})\Gamma(m_u+1)}{\Gamma(m_u+1+\frac{1}{k_u})} = \frac{\Gamma(1+\frac{1}{k_u})\Gamma(m_u+1)}{(m_u+\frac{1}{k_u})\Gamma(m_u+\frac{1}{k_u})} < \frac{m_u+1}{(m_u+\frac{1}{k_u})(m_u+1)^{\frac{1}{k_u}}} = O(m_u^{-\frac{1}{k_u}}).$$

Hence,

$$\mathbb{E}(\frac{r_i}{R}) \leq \mathbb{E}(\frac{r_i}{R_i}) \leq \mathbb{E}(\frac{s_i}{R_i}) \leq \mathbb{E}(\frac{s_u}{R_u}) = O(m_u^{-\frac{1}{k_u}}), \quad i \in \{1, \ldots, m\},$$

where $r_i = \min_{p \in f_i} \mathrm{dist}(q,p) - \mathrm{dist}(q, f_i)$ as in Figure 3 and $r_i \leq s_i$ [EKKNTF12, Sec.3].

By Cor. 9 and Lem. 10, this is an upper bound on the probability that a flat lies inside the hyperannulus of width r_i. We have found one flat and consider i.i.d. $m-1$ additional flats, so the expected number of flats we check, using $m_u = \Omega(m^{k_u})$, equals:

$$\mathbb{E}(N) \leq (m-1) \cdot \mathbb{E}(\frac{r_u}{R}) = (m-1) \cdot O(m_u^{-\frac{1}{k_u}}) = O(1). \tag{4}$$

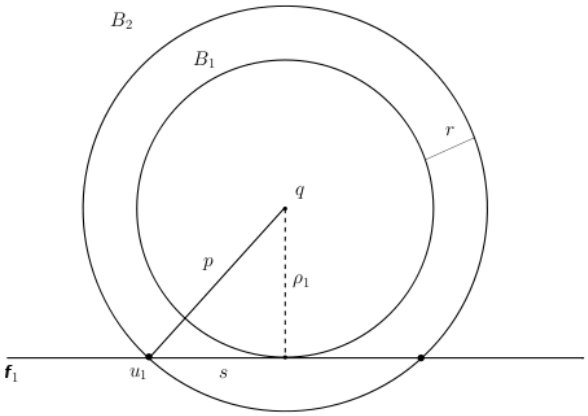

Figure 2: The annulus centered at q and defined by $\min_{p \in f_1} \mathrm{dist}(q,p)$, $\mathrm{dist}(q, f_1)$.

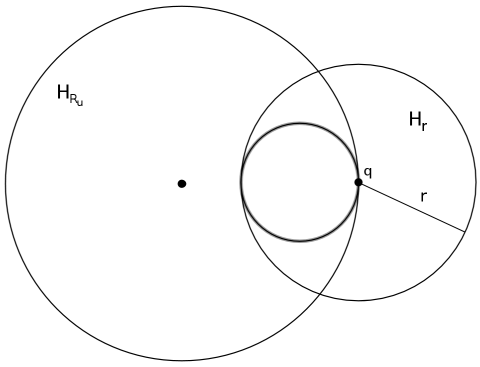

Figure 3: The ball of radius $r/2$ is inside $H_r \cap H_{R_u}$.

□

Lem. 11 holds only when q is the center of hypersphere H_{R_u} or close enough to it. This is an ideal case; let us also investigate the worst case which is q lying on the boundary of H_{R_u}.

Lemma 12. *The expected number of flats that the algorithm computes is constant, assuming the projection of q to the first flat is on the boundary of its intersection with the bounding ball.*

Proof. Since $r = \frac{[\Gamma(\frac{k_u}{2}+1)]^{\frac{1}{k_u}}}{\sqrt{\pi}} \mathrm{V}(H_r)^{\frac{1}{k_u}}$, we have $\mathbb{E}(s_u)$ equal to

$$\int_0^{2R_u} r\,\mathrm{P}(r)\mathrm{d}r = \frac{[\Gamma(\frac{k_u}{2}+1)]^{\frac{1}{k_u}}}{\sqrt{\pi}} m_u \int_0^{\mathrm{V}(H_{2R_u})} (1 - \frac{\mathrm{V}(H_{R_u} \cap H_r)}{\mathrm{V}(H_{R_u})})^{m_u - 1} \frac{\mathrm{V}(H_r)^{\frac{1}{k_u}}}{\mathrm{V}(H_{R_u})} \mathrm{d}(\mathrm{V}(H_{R_u} \cap H_r)),$$

where $\mathrm{P}(r)$ is the probability that the nearest neighbor lies at distance r from $\mathrm{proj}_{f_u}(q)$, and for which it holds $\mathrm{P}(r)\mathrm{d}r = (1 - \frac{\mathrm{V}(H_{R_u} \cap H_r)}{\mathrm{V}(H_{R_u})})^{m_u - 1} \frac{\mathrm{d}(\mathrm{V}(H_{R_u} \cap H_r))}{\mathrm{V}(H_{R_u})}$, where $\mathrm{d}(\mathrm{V}(H_{R_u} \cap H_r))$ is the volume in annulus A between spheres of radii r and $r + \mathrm{d}\,r$ centered at q inside H_{R_u}. We first lower bound $\mathrm{V}(H_{R_u} \cap H_r)$: $\mathrm{V}(H_{R_u} \cap H_r) > \mathrm{V}(H_{\frac{r}{2}})$ since q lies on the boundary. For every r and given k, $\mathrm{V}(H_r) = 2^k\,\mathrm{V}(H_{\frac{r}{2}})$. In addition, $\mathrm{d}(\mathrm{V}(H_{R_u} \cap H_r)) \leq \mathrm{d}(\mathrm{V}(H_r))/2$. Hence,

$$\mathbb{E}(s_u) \leq \frac{[\Gamma(\frac{k_u}{2}+1)]^{\frac{1}{k_u}}}{\sqrt{\pi}} m_u \int_0^{\mathrm{V}(H_{2R_u})} (1 - \frac{\mathrm{V}(H_{\frac{r}{2}})}{\mathrm{V}(H_{R_u})})^{m_u - 1} \frac{\mathrm{V}(H_r)^{\frac{1}{k_u}}}{2 \cdot \mathrm{V}(H_{R_u})} \mathrm{d}(\mathrm{V}(H_r)) =$$

$$= \frac{[\Gamma(\frac{k_u}{2}+1)]^{\frac{1}{k_u}}}{\sqrt{\pi}} m_u \int_0^{\mathrm{V}(H_{2R_u})} (1-\frac{\mathrm{V}(H_r)}{\mathrm{V}(H_{2R_u})})^{m_u-1} \frac{\mathrm{V}(H_r)^{\frac{1}{k}}}{\mathrm{V}(H_{2R_u})} 2^{k-1} \mathrm{d}(\mathrm{V}(H_r)).$$

Instead of assuming that $\mathrm{V}(H_{R_u}) = 1$, we assume $\mathrm{V}(H_{2R_u}) = 1$ and obtain,

$$\mathbb{E}(s_u) \leq 2^k R_u m_u \int_0^1 (1-\mathrm{V}(H_r))^{m_u-1} (\mathrm{V}(H_r))^{\frac{1}{k_u}} \mathrm{d}(\mathrm{V}(H_r)) \implies \mathbb{E}(\frac{s_u}{R_u}) = O(\frac{2^k}{(m_u)^{\frac{1}{k_u}}}).$$

We have assumed that our flats are of low dimension, i.e. $k_u = O(1)$, thus,

$$\mathbb{E}(N) \leq (m-1)\mathbb{E}(\frac{s_u}{R_u}) = O(1).$$

We hope to obtain the same without employing $k_u = O(1)$ by a tighter analysis. □

Thus we arrive at our expected-case complexity.

Theorem 13. *If $m_i = \Omega(m^{k_i})$, $i \in \{1, \ldots, m\}$, our algorithm returns an $(1+\epsilon_1)$-approximate nearest neighbor in expected query time $O(c \log m)$, using space $O(d'm+n)$, where $c \leq d'/2\lceil 1 + 6d'/\epsilon_1' \rceil^{d'}$, ϵ_1' is defined in Cor. 4 and $d' = O(d^2)$.*

In addition, we discuss a complexity bound with high probability, using the theory of Bins and Balls [MU05, Lem.5.1,p.93].

Lemma 14. *With probability $1 - O(\frac{1}{m^2})$ the number of flats our algorithm computes is $O(\log m)$.*

Proof. The difference in our case comparing to the standard setting of Bins and Balls model, is that we only care about one bin, and the probability with which one ball is dropped into this bin is proportional to a random variable. In other words,

$$\text{Prob}\left[\text{at least } \lambda \text{ flats in annulus defined by } q \text{ in } f_u\right] \leq \int_0^{2R_u} \binom{m-1}{\lambda} (\frac{r}{R_u})^{\lambda} \mathrm{P}(r) \mathrm{d}r \; <$$

$$< 2^{k_u} \binom{m-1}{\lambda} m_u \int_0^1 \mathrm{V}(H_r)^{\frac{\lambda}{k_u}} (1-\mathrm{V}(H_r))^{m_u-1} \, \mathrm{d}(\mathrm{V}(H_r)) = 2^{k_u} \binom{m-1}{\lambda} \cdot O(\frac{1}{m_u{}^{\frac{\lambda}{k_u}}}) = O(2^{k_u} \frac{1}{\lambda!}).$$

Now, for $\lambda = \frac{k_u \ln 2 + 3 \ln m}{\ln \ln m}$, we have,

$$2^{k_u} \frac{1}{\lambda!} \leq 2^{k_u} (\frac{\mathrm{e}}{\lambda})^{\lambda} = 2^{k_u} (\frac{\mathrm{e} \ln \ln m}{3 \ln m + k_u \ln 2})^{(k_u \ln 2 + 3 \ln m)/\ln \ln m}$$

$$\leq 2^{k_u} (\frac{\ln \ln m}{\ln m})^{(k_u \ln 2 + 3 \ln m)/\ln \ln m} = 2^{k_u} (\mathrm{e}^{\ln \ln \ln m - \ln \ln m})^{(k_u \ln 2 + 3 \ln m)/\ln \ln m}$$

$$\leq 2^{k_u} (\mathrm{e}^{(\ln \ln \ln m)(3 \ln m)/(\ln \ln m) - 3 \ln m - k_u \ln 2}) = \mathrm{e}^{(\ln \ln \ln m)(3 \ln m)/(\ln \ln m) - 3 \ln m} \leq \frac{1}{m^2}.$$

□

Lem. 14 yields the following:

Theorem 15. *Under the hypotheses and the notation of Thm 13, our algorithm returns an $(1+\epsilon)$-ANN in query time $O(c \log^2 m + \mathcal{Q})$ with probability $1 - O(\frac{1}{m^2})$, where $\mathcal{Q}$ is the query time corresponding to ANN on each flat.*

We conclude by relaxing our assumptions.

Our analysis is based on the assumption that $m_i = \Omega(m^{k_i}), i \in \{1, \ldots, m\}$. Let us consider the expected case after relaxing this assumption. Assume $m_i = \Omega(g(m)^{k_i}), i \in \{1, \ldots, m\}$, where $g(m)$ is a small fraction of m, e.g. $g(m) = m/\log^{O(1)} m$. It follows from (4) that

$$\mathbb{E}(N) = O(\frac{m}{g(m)}).$$

This leads to a generalization of Thm 13.

Theorem 16. *If $m_i = \Omega(g(m)^{k_i})$, $i \in \{1, \ldots, m\}$, our algorithm returns an $(1+\epsilon_1)$-approximate nearest neighbor in expected query time $O(c \cdot \frac{m}{g(m)} \log m)$ and space $O(d'm + n)$ where $c \leq d'/2\lceil 1 + 6d'/\epsilon_1' \rceil^{d'}$, ϵ_1' is defined in Cor. 4 and $d' = O(d^2)$.*

For instance, if $g(m) = m/\log^{O(1)} m$, we achieve expected query time polylogarithmic in the number of flats.

4 Experiments and Future Work

Our experimental results are summarized into two tables. In Table 1, we keep the number of points constant, while in Table 2 we keep the dimension constant. We also allow for points to lie within a small distance from the corresponding flat, which is called *flat thickness*. This relaxation of our model affects the query time.

We explore two cases regarding dimension k. The case where the points lie on lines and where they lie on planes. In our expected case analysis, the expectation comes from the randomness of our dataset but in our experiments we generate a random dataset and we make multiple queries to that dataset. What our analysis implies is that with high probability, we generate a dataset for which the number of flats our algorithm needs to check is $O(\log m)$ for any query.

We can also observe that when k is closer to d, our algorithm tends to check more flats than otherwise. This is reasonable and discussed in the proof of Lem. 10. The higher the dimension of a flat, the more probable to lie inside the annulus defined by q, the distance to its nearest flat and the distance to the nearest point on the nearest flat.

We present query times for the case when we search for the nearest point after projecting points to their corresponding flat and when we do not project so that the points lie on d dimensions. In both cases, our query times are better than those achieved by CGAL standard algorithm.

In our analysis, we assume uniform distribution for the points on flats. We intend to investigate the distribution with bounded density with respect to Lebesgue measure which was proposed in [BWY80]. According to this, there exist constants $0 < c_1 \leq c_2$ such that the probability of a point to lie in any convex subregion C is in the interval $[c_1 \, \mathrm{V}(C), c_2 \, \mathrm{V}(C)]$.

Acknowledgments. Sec. 2.1 and the Appendix essentially reproduce parts of the Internship report by Ken Chanseau Saint-Germain in the Department of Informatics and Telecommunications, National and Kapodistrian University of Athens. We thank Ioannis Kavvouras[1] for contributing an initial implementation.

[1]Department of Informatics and Telecommunications, National and Kapodistrian University of Athens, {i.kabboyras}@di.uoa.gr.

References

[AdFM11] Sunil Arya, Guilherme D. da Fonseca, and David M. Mount. Approximate polytope membership queries. In *Proc. ACM Symposium on Theory of Computing*, pages 579–586, 2011.

[AI08] Alexandr Andoni and Piotr Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. *Commun. ACM*, 51(1):117–122, 2008.

[AIKN09] Alexandr Andoni, Piotr Indyk, Robert Krauthgamer, and Huy L. Nguyen. Approximate line nearest neighbor in high dimensions. In *Proc. 19th annual ACM-SIAM Symposium on Discrete algorithms*, pages 293–301, 2009.

[AMM09] Sunil Arya, Theocharis Malamatos, and David M. Mount. Space-time tradeoffs for approximate nearest neighbor searching. *J. ACM*, 57(1):1:1–1:54, 2009.

[AMMW07] Sunil Arya, Theocharis Malamatos, David M. Mount, and Ka Chun Wong. Optimal expected-case planar point location. *SIAM J. Comput.*, 37(2):584–610, 2007.

[AMN+98] Sunil Arya, David M Mount, Nathan S Netanyahu, Ruth Silverman, and Angela Y Wu. An optimal algorithm for approximate nearest neighbor searching fixed dimensions. *J. ACM*, 45(6):891–923, 1998.

[BC08] Pratip Bhattacharyya and Bikas K. Chakrabarti. The mean distance to the n-th neighbour in a uniform distribution of random points: an application of probability theory. *European J. of Physics*, 29:639–645, 2008.

[Ben75] Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Commun. ACM*, 18(9):509–517, 1975.

[BHZM07] Ronen Basri, Tal Hassner, and Lihi Zelnik-Manor. Approximate nearest subspace search with applications to pattern recognition. In *Proc. IEEE Conf. Computer Vision and Pattern Recognition*, 2007.

[BHZM11] Ronen Basri, Tal Hassner, and Lihi Zelnik-Manor. Approximate nearest subspace search. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 33(2):266–278, 2011.

[BWY80] Jon Louis Bentley, Bruce W. Weide, and Andrew C. Yao. Optimal expected-time algorithms for closest point problems. *ACM Trans. Math. Softw.*, 6(4):563–580, 1980.

[CDI+08] Sébastien Collette, Vida Dujmović, John Iacono, Stefan Langerman, and Pat Morin. Distribution-sensitive point location in convex subdivisions. In *Proc. ACM-SIAM Symposium on Discrete algorithms*, pages 912–921, 2008.

[Cla88] Kenneth L. Clarkson. A randomized algorithm for closest-point queries. *SIAM J. Comput.*, 17(4):830–847, 1988.

[DF08] Sanjoy Dasgupta and Yoav Freund. Random projection trees and low dimensional manifolds. In *Proc. ACM Symposium on Theory of Computing*, pages 537–546. ACM, 2008.

[DSWS08] Jonathan Derryberry, Don Sheehy, Maverick Woo, and Danny Dominic Sleator. Achieving spatial adaptivity while finding approximate nearest neighbors. In *Proc. Canadian Conf. Computational Geometry*, 2008.

[EKKNTF12] Ioannis Z. Emiris, Alexandros Konstantinakis-Karmis, Dimitri Nicolopoulos, and Emmanouil Thanos-Filis. Data structures for approximate nearest neighbor search. *Technical Report CGL-TR-29*, 2012.

[EMT10] Ioannis Z. Emiris, Theocharis Malamatos, and Elias P. Tsigaridas. Approximate nearest neighbor queries among parallel segments. *26th Europ. Workshop Comp. Geom. (EuroCG), Dortmund*, pages 141–144, 2010.

[FB81] Martin A. Fischler and Robert C. Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM*, 24(6):381–395, 1981.

[FBF77] Jerome H. Friedman, Jon Louis Bentley, and Raphael Ari Finkel. An algorithm for finding best matches in logarithmic expected time. *ACM Trans. Math. Softw.*, 3(3):209–226, 1977.

[FDKV07] Yoav Freund, Sanjoy Dasgupta, Mayank Kabra, and Nakul Verma. Learning the structure of manifolds using random projections. In *Proc. 21st Annual Conf. Neural Information Processing Systems*, 2007.

[Gau60] Walter Gautschi. Some elementary inequalities relating to the gamma and incomplete gamma function. *J. Math. and Phys. 38*, pages 77–81, 1959/60.

[HPM05] Sariel Har-Peled and Manor Mendel. Fast construction of nets in low dimensional metrics, and their applications. In *Proc. Symposium on Computational Geometry*, pages 150–158. 2005.

[HPV02] Sariel Har-Peled and Kasturi Varadarajan. Projective clustering in high dimensions using core-sets. In *Proc. Symposium on Computational Geometry*, pages 312–318. 2002.

[IL03] John Iacono and Stefan Langerman. Proximate planar point location. In *Proc. Symposium on Computational Geometry*, pages 220–226. 2003.

[IN07] Piotr Indyk and Assaf Naor. Nearest-neighbor-preserving embeddings. *ACM Trans. Algorithms*, 3(3), 2007.

[KS02] Vladlen Koltun and Micha Sharir. Polyhedral voronoi diagrams of polyhedra in three dimensions. In *Proc. Symposium on Computational Geometry*, pages 227–236. 2002.

[Mag02] Avner Magen. Dimensionality reductions that preserve volumes and distance to affine spaces, and their algorithmic applications. In *Proc. 6th Intern. Workshop Randomization and Approximation Techniques in Computer Science*, pages 239–253, 2002.

[Mei93] Stefan Meiser. Point location in arrangements of hyperplanes. *Information and Computation*, 106(2):286–303, 1993.

[ML09] Marius Muja and David G. Lowe. Fast approximate nearest neighbors with automatic algorithm configuration. In *Proc. Intern. Conf. Computer Vision Theory and Applications*, pages 331–340, 2009.

[MM01] Songrit Maneewongvatana and David M. Mount. On the efficiency of nearest neighbor searching with data clustered in lower dimensions. In *Proc. Intern. Conf. Computational Science*, pages 842–851, 2001.

[MNP07] Rajeev Motwani, Assaf Naor, and Rina Panigrahy. Lower bounds on locality sensitive hashing. *SIAM J. Discret. Math.*, 21(4):930–935, 2007.

[MU05] Michael Mitzenmacher and Eli Upfal. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. Cambridge University Press, 2005.

[VH04] Rene Vidal and Richard Hartley. Motion segmentation with missing data using powerfactorization and GPCA. In *Proc. IEEE Conf. Computer Vision and Pattern Recognition*, pages 310–316, 2004.

[Vid11] René Vidal. Subspace clustering. *IEEE Signal Process. Mag.*, 28(2):52–68, 2011.

[Wan08] Yusu Wang. Approximating nearest neighbor among triangles in convex position. *Inf. Process. Lett.*, 108(6):379–385, 2008.

[YP06] Jingyu Yan and Marc Pollefeys. A general framework for motion segmentation: Independent, articulated, rigid, non-rigid, degenerate and non-degenerate. In *European Conf. Computer Vision*, pages 94–106. 2006.

Appendix

We report on an implementation by K. Chanseau Saint-Germain in C++, using CGAL and EIGEN libraries and GMP rational numbers. The experiments took place in the `geomcomp.di.uoa.gr` server of the University of Athens with an Intel Core i5-2320 CPU@3.00GHz, 4GB RAM. We ran several versions of the program, and the CGAL kd-tree algorithm in order to compare the algorithms. The preprocessing takes a really long time to execute because of the clustering, that should be improved.

Let R the radius of the bounding sphere. To generate n points of artificial data in dimension d, lying on subspaces of dimension k, our algorithm generates $\log n$ subspaces by the following steps:

1. Generate a random point t with a continuous uniform distribution in the ball of dimension d of centre O and of radius $R/2$.

2. Generate k random vectors $(u_i)_{1\leq i\leq k}$ with a continuous uniform distribution in the ball of dimension d and of radius 1.

3. Calculate the orthogonal hyperplane of t and project orthogonally $(u_i)_{1\leq i\leq k}$ on it: we obtain $(u'_i)_{1\leq i\leq k}$ with $\forall i \in [1,k]$, $u'_i \perp t$.

4. Orthonormalise with Gram-Schmidt $(u'_i)_{1\leq i\leq k}$: let $(v_i)_{1\leq i\leq k}$ the orthonormalised basis of the affine subspace.

5. Generate at least $\log n$ random points (p'_i) in the ball of dimension k, of centre O and of radius $R/2$.

6. For all i, transform p'_i in d dimensions: $p_i = [v_1 \cdots v_k]p'_i + t$. These points are the output.

The first step of preprocessing is, given their dimension, to find the affine subspaces. Our clustering algorithm is simple but not efficient. It chooses a set S of $(k+1)\log n$ points, so there is at least one k-dimensional affine subspace defined by these points. For all combinations of $k+1$ points in S, it tests whether one of the points in the complement S^C belongs to the affine subspace they defined. When it is the case, it assigns all the lying points to this affine subspace, and repeats the operation from the beginning with all the remaining points. However, the clustering slows down the experiments and should be improved to help experiments for higher values of k, for instance, which are impossible with the current clustering.

In the implementation, points lying on k-flats are stored in a kd-tree of the CGAL library, of dimension d', and a map structure is used to convert them to d dimensions. We have developed the following classes:

- `Properties` gets the properties given in the command line to initialise the algorithm.
- `AffineSubspace` encapsulates the offset `_origin`, the basis `_base` and dimension `_dim` of an affine subspace, all its points `_points`, and the distance `_thickness` from the flat at which they may lie. It also encapsulates the projector matrix `_ZZT`, the nearest neighbour structure `_nnstr` and a mapping `_dTodp` between cartesian coordinates in dimension d and dimension d', if needed. The constructor calculates the basis and the offset thanks to a Gram-Schmidt process. The method `add(Pointd)` adds the point if it belongs to the affine subspace, and returns true if it is true. The methods `pointsCard()`, `orthBase()`, `dim()`, `offset()`, `ZZT()`, `distance(Pointd)` and `search(Pointd)` construct the data if they do not already exist.
- `Clusters` constructs the affine subspaces from a set of points.
- `Mapping`: its constructor maps a vector of affine subspaces in d'. The method `mapQuery` maps the query points in d'.
- `NNStructure` creates the NN structure of CGAL, by creating a kd-tree with the constructor and encapsulating it. The method `getQuery(Pointd)` returns the ith nearest neighbour, `removePoint()` increments the parameter i, `open()` sets it to 1, and `close()` deletes the set of nearest neighbours.
- `QueryAlgorithm` runs our overall algorithm.

parameters						time (in seconds)						number of queries for ... checked subspaces								
inp.	quer.	d	k	R	r	CG	pre.	dir.	pro.	sub.	s.ad.	1	2	3	4	5	6	7	8	9
5000	10000	5	1	5000	0.01	8	2	11	6	2	2.6	7485	2163	329	23	0	0	0	0	0
5000	10000	10	1	5000	0.01	33	6	29	13	7	7.3	9571	415	14	0	0	0	0	0	0
5000	10000	15	1	5000	0.01	98	3	64	27	15	15.2	9885	114	1	0	0	0	0	0	0
5000	10000	20	1	5000	0.01	83	14	85	44	24	24.6	9749	247	4	0	0	0	0	0	0
5000	10000	25	1	5000	0.01	183	21	124	67	44	44.4	9911	89	0	0	0	0	0	0	0
5000	10000	30	1	5000	0.01	329	9	183	93	51	51.4	9924	75	1	0	0	0	0	0	0
5000	10000	35	1	5000	0.01	395	23	218	128	70	70.9	9878	122	0	0	0	0	0	0	0
5000	10000	40	1	5000	0.01	376	27	267	158	92	93.1	9877	123	0	0	0	0	0	0	0
5000	10000	45	1	5000	0.01	837	11	347	206	113	113.9	9920	80	0	0	0	0	0	0	0
5000	10000	50	1	5000	0.01	797	163	400	248	140	141.7	9879	120	1	0	0	0	0	0	0
5000	10000	55	1	5000	0.01	556	80	441	296	168	171.7	9781	215	4	0	0	0	0	0	0
5000	10000	60	1	5000	0.01	1150	216	614	349	244	245.5	9940	58	2	0	0	0	0	0	0
5000	10000	5	2	5000	0.01	8	20	12	6	3	4.9	5538	3061	1087	284	30	0	0	0	0
5000	10000	10	2	5000	0.01	56	90	39	15	7	8.4	8244	1556	183	17	0	0	0	0	0
5000	10000	15	2	5000	0.01	174	64	91	33	17	18.7	9083	855	60	2	0	0	0	0	0
5000	10000	20	2	5000	0.01	348	461	131	48	28	29.9	9359	618	23	0	0	0	0	0	0
5000	10000	25	2	5000	0.01	452	825	191	78	46	54.4	8354	1483	144	18	1	0	0	0	0
5000	10000	30	2	5000	0.01	776	41	252	94	51	52.8	9659	333	8	0	0	0	0	0	0
5000	10000	35	2	5000	0.01	1015	158	326	132	68	74.2	9144	809	44	3	0	0	0	0	0
5000	10000	40	2	5000	0.01	909	117	398	176	92	104.2	8781	1117	92	10	0	0	0	0	0
5000	10000	45	2	5000	0.01	1341	517	492	228	115	125.7	9124	821	54	1	0	0	0	0	0
5000	10000	50	2	5000	0.01	1429	243	624	272	150	178.6	8322	1468	193	17	0	0	0	0	0
5000	10000	55	2	5000	0.01	1721	3693	635	305	163	177.0	9196	752	50	2	0	0	0	0	0

inp.: number of input points
quer.: number of queries
d: general dimension
k: subspace dimension
R: radius of the bounding sphere
r: flat thickness

CG: CGAL algorithm
pre.: preprocessing
dir.: computes directly the nearest neighbour in a subspace, without projecting points
pro.: computes with projected points
sub.: time with the search of the nearest subspace only
s.ad.: same time than sub., but adjusted with the number of checked subspaces.

Table 1: Experiments with constant number of input points.

parameters						time (in seconds)						number of queries for .. checked subspaces								
inp.	quer.	d	k	R	r	CG	pre.	dir.	pro.	sub.	s.ad.	1	2	3	4	5	6	7	8	9
5000	10000	20	1	5000	0.01	159	6	93	45	24	24.3	9888	111	1	0	0	0	0	0	0
10000	10000	20	1	5000	0.01	183	44	114	48	25	25.3	9886	113	1	0	0	0	0	0	0
15000	10000	20	1	5000	0.01	189	32	151	48	30	30.3	9899	100	1	0	0	0	0	0	0
20000	10000	20	1	5000	0.01	306	34	164	56	34	36.8	9188	800	12	0	0	0	0	0	0
25000	10000	20	1	5000	0.01	255	49	175	50	27	27.3	9901	98	1	0	0	0	0	0	0
30000	10000	20	1	5000	0.01	201	44	155	60	31	31.5	9839	160	1	0	0	0	0	0	0
35000	10000	20	1	5000	0.01	372	61	240	52	29	29.2	9929	70	1	0	0	0	0	0	0
40000	10000	20	1	5000	0.01	261	75	201	63	27	27.2	9913	87	0	0	0	0	0	0	0
5000	10000	20	2	5000	0.01	288	695	166	61	28	32.4	8567	1300	126	7	0	0	0	0	0
10000	10000	20	2	5000	0.01	255	3038	199	50	26	29.7	8707	1179	107	7	0	0	0	0	0
15000	10000	20	2	5000	0.01	710	114	245	61	28	33.9	8169	1573	228	29	1	0	0	0	0
20000	10000	20	2	5000	0.01	366	335	218	60	29	34.8	8222	1566	197	15	0	0	0	0	0
25000	10000	20	2	5000	0.01	477	623	280	59	29	34.5	8389	1440	153	17	1	0	0	0	0
30000	10000	20	2	5000	0.01	346	219	209	58	28	33.9	8132	1649	206	13	0	0	0	0	0

inp.: number of input points
quer.: number of queries
d: general dimension
k: subspace dimension
R: radius of the bounding sphere
r: flat thickness

CG: CGAL algorithm
pre.: preprocessing
dir.: computes directly the nearest neighbour in a subspace, without projecting points
pro.: computes with projected points
sub.: time with the search of the nearest subspace only
s.ad.: same time than sub., but adjusted with the number of checked subspaces.

Table 2: Experiments with constant dimension $d = 20$.