Nearest Neighbor Search in Lower Dimensional Flats

Ioannis Z. Emiris* Ioannis Psarros*

Abstract

In order to improve efficiency in Approximate Nearest Neighbor (ANN) search, we exploit the structure of the input data by considering points that are distributed almost uniformly on flats of varying, fixed dimension. These flats are distributed uniformly within a bounding sphere of radius 1. We employ an existing mapping that transforms Nearest Flat search to Nearest point search. Our algorithm, using linear space in the number of points, returns an $(1+\epsilon)$-approximate neighbor in expected time $O(c \log m)$, where m is the number of flats, $c = O^*((\epsilon\lambda^2)^{-d^2})$, d is the ambient space dimension, λ is the minimum distance between a query point and a flat, and $O^*(F)$ implies we omit polylog factors in F. With high probability, queries take $O(c \log^2 m)$, under mild extra assumptions. If we do not rely on λ, there is also an additive error in the distance of the output point.

1 Introduction

Nearest neighbor searching is a fundamental computational problem. We consider $(1+\epsilon)$-ANN, when the data points lie on affine spaces (flats) of constant dimension. Consider n points in $\mathbb{R}^d$ and let $\text{dist}(p, p')$ be the Euclidean distance between points p, p'. The problem consists in reporting, given query point q, a point p such that $\text{dist}(q, p) \leq (1+\epsilon)\text{dist}(q, p')$, for all input points p'.

Datasets with this structure appear in computer vision and image processing [4]. In motion segmentation, point trajectories associated with the same motion lie on the same low dimensional flat. Images of the same face under varying illumination (approximately lie) on a low dimensional flat. Our model generalizes to points approximately lying on flats and we intend to formalize this notion in the future.

For data structures such as kd-trees, Balanced Box-Decomposition (BBD) trees, and Approximate Voronoi Diagrams, the query time is typically exponential in d and logarithmic in n, and their complexity bounds do not take structure into account. Locality sensitive hashing achieves query time $O(dn^{\epsilon^{-2}+o(1)})$ using $O(dn + n^{1+\epsilon^{-2}+o(1)})$ space [1], but does not exploit structure formally. Only recently, there has been a result for line queries [2].

In [11] are presented randomized embeddings that preserve properties such as distance between points and flats. They also present a reduction from the nearest flat problem to the vertical ray shooting in arrangements of hyperplanes. For point queries, computing the nearest flat is in $O(d^{10} \log m)$, which is faster than ours, but preprocessing time and space are in $O(m^{d^2})$, where m denotes the number of flats.

Structure has been studied via pointsets with low doubling dimension k which generalizes the notion of Euclidean space dimension. In [9], they provide an algorithm for ANN with expected preprocessing in $O(2^k n \log n)$, space $O(2^k n)$, and query in $O(2^k \log n + \epsilon^{-O(k)})$; the latter is proportional to $\log n$ whereas our bound to $\log m$, which can be $\log \log n$. In [10], they provide randomized embeddings that preserve nearest neighbors in dimension $O(k \log(1/\epsilon)/\epsilon)$ with constant probability. The approach is combined with any ANN-structure but, again, structure of the input is not exploited.

In [7] they present random projection trees, which adapt to pointsets of low doubling dimension. In particular, cell diameter decreases as a function of k, but ANN queries seem to be exponential in k. Similar behavior can be achieved in kd-trees [13] by randomly rotating the ambient space as part of preprocessing.

Our model. Since flats are of constant dimension, we assume that we learn them during preprocessing. This is the subspace clustering problem solved, e.g., by RanSaC (Random Sampling Consensus).

Let $m \in \mathbb{N}$ be the number of flats and $n_i \in \mathbb{N}$ the number of points on flat f_i s.t. $\sum_{i=1}^{m} n_i = n$. Let k_i be the dimension of f_i. Moreover, we assume:

1. For $i = 1, \ldots, m$, the n_i points are distributed uniformly at random on f_i; this hypothesis can be relaxed with small extra work.
2. The m flats are distributed uniformly at random in a bounding sphere B (see below).
3. For $i = 1, \ldots, m$, $n_i = \Omega(m^{k_i})$.
4. Queries are not closer than $\lambda > 0$ to any flat, otherwise an additive error exists in the distance of the output point.

For assumption (3), a lower bound on the number

*Department of Informatics and Telecommunications, University of Athens, Greece, {emiris,i.psarros}@di.uoa.gr. This research has been co-financed by the European Union (European Social Fund ESF) and Greek national funds through the Operational Program "Education and Lifelong Learning" of the National Strategic Reference Framework (NSRF) - Research Funding Program: THALIS UOA (MIS 375891).

of points per flat seems natural, since flats with arbitrarily small number of points remove the structural characteristics. The assumption is not very restrictive, especially when m is small; still, we can relax it. In particular, we may assume that m correlates with n, e.g. $m = \Theta(\log n)$ yields expected query time $O(c \log\log n)$.

The input points could be generated as follows. A bounding sphere B is chosen and m flats that intersect the sphere are picked uniformly at random (see 2). The intersection of each k_i-flat f_i with the bounding sphere B is a hypersphere of dimension k_i. In the ball bounded by this hypersphere, n_i points are independently and identically distributed (i.i.d.) according to the uniform distribution.

We pick a k_i-flat f_i uniformly at random in 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 $b = \alpha e_1$. Now, f_i is random in the orthogonal hyperplane of e_1 with offset b, specified by a basis of k_i independent uniform unit vectors: if $k_i < d-1$, we pick $e_2, \ldots, e_{k_i+1}$ s.t. $\forall i \in \{1, \cdots, k_i\}$, e_{i+1} is uniformly distributed in the space orthogonal to the span of $\{b, e_2, \ldots, e_i\}$. Now, $b, e_2, \cdots, e_{k_i+1}$ define f_i.

This extended abstract better reformulates the theoretical results of [8], where some missing proofs can be found. The latter presents an implementation of our algorithm which uses CGAL kd-trees: for $5 \cdot 10^3$ points in 55 dimensions, or $30 \cdot 10^3$ points in 20 dimensions, with both sets lying on 2-dimensional flats, it is about 5 times faster than simply using CGAL kd-trees. We intend to experimentally compare software based on LSH, e.g. LSHKIT or Caltech Large Scale Image Search [1].

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. On 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: Find the flats on which the points lie and for each flat store its points. Map each flat to a point in d'-dimensional space, $d' > d$, and construct a dynamic ANN-structure. Query steps:

Q1 (a) Given query q, map it to the d'-dimensional space, (b) find and remove its ANN from the ANN-structure.

Q2 Find the corresponding flat. Among its points, find the approximate nearest neighbor to q, which lies at distance ρ.

Q3 Repeat starting at **Q1**(b): (a) If the new distance $< \rho$, update ρ with the new distance and update the current nearest neighbor. (b) Otherwise, return the current nearest neighbor.

Preprocessing may compute k-dimensional flats as follows. 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. Starting with $k = 1$, we continue until all points are assigned to flats (having assumed this happens for a constant number of dimensions). The algorithm constructs an ANN-structure per flat. Then, 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 $\mathbb{R}^{d'}$, where ϵ_1' is specified in lem. 2.

In **Q1** the algorithm maps q to $q' \in \mathbb{R}^{d'}$, then finds its $(1+\epsilon_1')$-ANN $s_1 \in S$. We then remove s_1 from the ANN-structure. Point s_1 corresponds to flat f_{i_1} which is a $(1+\epsilon_1)$-approximate nearest flat to q. In **Q2**, we compute the distance ρ_1 from q to f_{i_1}, and find $u_1 \in f_{i_1}$ (almost) 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 **Q3**, we find approximate nearest flats and, for each, we repeat the above procedure. Let N be the number of flats computed. 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 $\min_{1 \le j \le N}\{\text{dist}(q, u_j)\} \le \rho_N$, where N is such that the $(1+\epsilon)$ approximation factor is obtained.

For completeness, we describe the mapping from [4] and we correct it in (1). Let f_i be a k-flat and let Z a $d \times (d-k)$ matrix whose columns form an orthonormal basis of the orthogonal subspace of f_i. Let $t \in \mathbb{R}^{d-k}$ be the offset vector of f_i in its orthogonal subspace and $\hat{Z} = \begin{bmatrix} Z \\ t^T \end{bmatrix}$. Let

$$h(A) = (\frac{a_{11}}{\sqrt{2}}, a_{12}, \cdots, \frac{a_{22}}{\sqrt{2}}, a_{23}, \cdots),$$

map a symmetric $d \times d$ matrix $A = (a_{ij})$ into a d'-dimensional vector, where $d' = \frac{d(d+1)}{2} + 1$. We introduce M such that the following 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 [4], corrected in [6]:

$$\|\hat{Z}\hat{Z}^T\|_F^2 = d - k + 2\|t\|^2 + \|t\|^4. \quad (1)$$

It suffices to pick $M = \sqrt[4]{d + 2R^2 + R^4}$, where R is the bounding ball radius.

[1] en.wikipedia.org/wiki/Locality-sensitive_hashing

Proposition 1 [4] *Flat f_i is mapped to point $u_i = -[h(\hat{Z}\hat{Z}^T), \hat{c}(A)]^T \in \mathbb{R}^{d'}$, and q is mapped to $v = [h(\hat{q}\hat{q}^T), 0] \in \mathbb{R}^{d'}$, where $\hat{q} = [q^T, 1]^T$. Then,*

$$\mathrm{dist}^2(u_i, v) = \mu \ \mathrm{dist}^2(f_i, q) + \nu,$$

where $\mu = 1$, $2\nu = M^4 + \|(q,1)\|^4$, so $\nu \leq d/2 + 2R^4$.

Since ν only depends on q but not on f_i, the mapping is valid for flats of varying dimension. There is also a tradeoff between μ and ν, see [4].

Let T_1 be the minimum distance between q and a flat, where $\mathrm{T}_1 \leq \lambda$, and let T'_1 be the distance between v and its exact nearest neighbor in $\mathbb{R}^{d'}$.

Lemma 2 *The approximation errors ϵ_1, ϵ'_1 are such that $\mathrm{dist}(f_i, q) \leq (1+\epsilon_1)\mathrm{T}_1$, and $\mathrm{dist}(u_i, v) \leq (1+\epsilon'_1)\mathrm{T}'_1$, provided*

$$0 < \epsilon'_1 \ \leq \ \frac{1}{6}\,\epsilon_1\,\frac{\lambda^2}{\nu}.$$

Proof. Prop. 1 implies $\mathrm{T}'^2_1 = \mathrm{T}^2_1 + \nu$ because $\mu = 1$. Then, $\mathrm{dist}^2(u_i, v) = \mathrm{dist}^2(f_i, q) + \nu \implies$

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

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

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

$\implies (1+\epsilon_1)\mathrm{T}_1 \geq \mathrm{dist}(f_i, q)$. Hence it suffices to set $\epsilon'_1 > 0$ such that $1 + \epsilon_1 \geq \sqrt{1 + 3\epsilon'_1(1 + \nu/\lambda^2)}$. □

Theorem 3 *Our algorithm returns a $(1+\epsilon)$-approximate nearest neighbor, $\epsilon = \max\{\epsilon_1, \epsilon_2\}$.*

Proof. The distance to the $(1+\epsilon_2)$-approximate nearest point per flat 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 Lem. 2 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

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

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

If we do not rely on λ, Lem. 2 implies that the additive error in computing the nearest flat is $\leq 3\epsilon'_1\nu$, which also bounds the overall additive error.

Corollary 4 *Our algorithm returns a point p such that $\mathrm{dist}(q,p) \leq (1+\epsilon)\,\mathrm{dist}(q,p') + 3\epsilon'_1(d/2 + 2R^4)$, for all input points p'.*

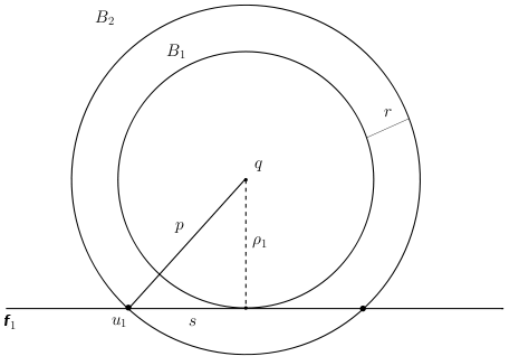

Figure 1: The annulus centered at q and defined by $\min_{p \in f_1} \mathrm{dist}(q,p)$ and $\mathrm{dist}(q, f_1)$.

3 Time Complexity

We use BBD-trees to store pointset $S \subset \mathbb{R}^{d'}$ which have query time $O(c \log m)$, $c = O^*((\epsilon\lambda^2)^{-d^2})$, and take space $O(d^2 m)$, where m is the number of points in the tree [3]. Recall $O^*(F)$ implies we omit polylog factors in F. In preprocessing, finding the flats costs $\sum_{1 \leq j \leq \max_i k_i} \binom{n}{j} \cdot O(n)$ time in a straightforward manner.

To find the point nearest to $\mathrm{proj}_{f_i}(q)$ on any f_i, we apply the idea of [5]. We decompose each of our k-flats in cells, requiring $O(n)$ space. Then, finding an exact nearest neighbor on a flat requires $O(1)$ expected time if $k_i = O(1), i \in \{1, \ldots, m\}$. Hence, the overall expected query time is $O(N \cdot c \log m)$.

The probability that a flat lies in a hyperannulus of width r is r/R. The idea behind the following lemma is that, for each unit vector e_1, the probability that we pick α s.t. a random hyperplane lies inside an annulus of width r is r/R. For flats of lower dimension the probability also depends on the random choice of its basis and hence it cannot be larger.

Lemma 5 *The probability of a flat to lie inside an annulus of width r, centered at any point inside the bounding ball of radius R is $\leq r/R$.*

Lemma 6 *The expected total number of flats that the algorithm finds, denoted $\mathbb{E}[N]$, is constant.*

Proof. We first bound the expected number of flats in an annulus defined by the distance from q to its nearest flat f_1 and the distance from q to its nearest point on f_1 (See Fig. 1). This is a bound on $\mathbb{E}[N]$ because the algorithm stops when the next nearest flat is farther than the current nearest point. We first compute the expected distance between $\mathrm{proj}_{f_1}(q)$ and its nearest neighbor on f_1. This distance is an upper bound on the width of the aforementioned annulus

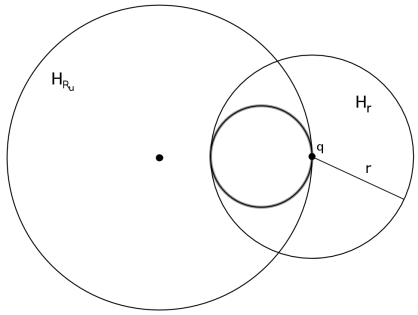

Figure 2: The ball of radius $r/2$ is inside $H_r \cap H_{R_u}$.

and we can relate it with the probability that a flat lies inside the annulus, using Lem. 5.

We focus on the worst case for q, where $\text{proj}_{f_1}(q)$ is on the boundary of $f_1 \cap B$. Let R_i be the radius of hypersphere $H_{R_i} = f_i \cap B$, $i = 1, \ldots, m$; $\mathbb{E}(s_i)$ is the expected distance between point $\text{proj}_{f_i}(q) \in f_i \cap B$ 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 have

$$\mathbb{E}(s_u) = \int_0^{2R_u} r\,\mathrm{P}(r)\mathrm{d}r, \quad \text{s.t.} \quad r = \frac{[\Gamma(\frac{k_u}{2}+1)\,\mathrm{V}(H_r)]^{\frac{1}{k_u}}}{\sqrt{\pi}}$$

and $\mathrm{P}(r)$ is the probability that the nearest neighbor lies at distance r from $\text{proj}_{f_u}(q)$, thus

$$\mathrm{P}(r)\mathrm{d}r = (1 - \frac{\mathrm{V}(H_{R_u} \cap H_r)}{\mathrm{V}(H_{R_u})})^{n_u - 1} \frac{\mathrm{d}(\mathrm{V}(H_{R_u} \cap H_r))}{\mathrm{V}(H_{R_u})},$$

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}. Now $\mathrm{V}(H_{R_u} \cap H_r) > \mathrm{V}(H_{\frac{r}{2}})$ since q lies on the boundary (Fig. 2). For every r and given k, $\mathrm{V}(H_r) = 2^k\,\mathrm{V}(H_{\frac{r}{2}})$. In addition, $\mathrm{d}(\mathrm{V}(H_{R_u} \cap H_r)) \leq \mathrm{d}(\mathrm{V}(H_r))/2$. We also assume wlog $\mathrm{V}(H_{2R_u}) = 1$. Finally, $\mathbb{E}(s_u) \leq$

$$2^{k_u} R_u n_u \int_0^1 (1 - \mathrm{V}(H_r))^{n_u - 1} (\mathrm{V}(H_r))^{\frac{1}{k_u}} \mathrm{d}(\mathrm{V}(H_r)) =$$

$$2^{k_u} R_u \cdot \frac{\Gamma(1 + \frac{1}{k_u})\Gamma(n_u + 1)}{\Gamma(n_u + 1 + \frac{1}{k_u})} \Rightarrow \mathbb{E}(\frac{s_u}{R_u}) = O(2^{k_u} n_u^{-\frac{1}{k_u}}).$$

We have assumed $k_u = O(1)$, thus, $\mathbb{E}[N] \leq (m-1)\mathbb{E}(\frac{s_u}{R_u}) = O(1)$ since we have already found one flat and consider i.i.d. $m - 1$ additional flats. □

Theorem 7 *Our algorithm returns an $(1 + \epsilon_1)$-ANN in expected query time $O(c \log m)$, using space $O(d^2 m + n)$, where $c = O^*((\epsilon\lambda^2)^{-d^2})$.*

To control the worst case, we maintain a $(1 + \epsilon_2)$-ANN structure per flat. Assuming that this structure has worst-case query time $\mathcal{Q}$, the overall query time is $O(N \cdot (c \log m + \mathcal{Q}))$. Employing BBD-trees for all flats requires $O(n \max_i k_i)$ space, allows point deletions in $O(\log \max_i n_i)$, and queries in $O(c' \log(\max_i n_i))$, $c' = O^*((\epsilon_2)^{-k})$. In addition, in the worst case $N = m$.

Applying the theory of Balls and Bins [12, Lem.5.1,p.93], we obtain that, with probability $1 - O(\frac{1}{m^2})$, our algorithm computes $O(\log m)$ flats.

Theorem 8 *Our algorithm, with BBD-trees storing the points in each flat, returns an $(1 + \epsilon)$-ANN in $O(\log m(c \log m + c' \log(\max_i n_i))$, with probability $1 - O(\frac{1}{m^2})$ and, in the worst case, in $O(m \cdot (c \log m + c' \log(\max_i n_i)))$. Space usage is in $O(d^2 m + n)$.*

References

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

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

[3] S. Arya, D.M. Mount, N.S. Netanyahu, R. Silverman, and A.Y. Wu. An optimal algorithm for approximate nearest-neighbor searching in fixed dimensions. *J. ACM*, 45(6):891–923, 1998.

[4] R. Basri, T. Hassner, and L. Zelnik-Manor. Approximate nearest subspace search. *IEEE Trans. on PAMI*, 33(2):266–278, 2011.

[5] J. L. Bentley, B. W. Weide, and A. C. Yao. Optimal expected-time algorithms for closest point problems. *ACM Trans. Math. Softw.*, 6(4):563–580, 1980.

[6] K. Chanseau Saint-Germain. *Internship Report.* Dept. Informatics & Telecommunications, University of Athens, Summer 2013.

[7] S. Dasgupta and Y. Freund. Random projection trees and low dimensional manifolds. In *Proc. ACM STOC*, pp.537–546. 2008.

[8] I.Z. Emiris, I. Psarros, and K. Chanseau Saint-Germain. Approximate nearest neighbor search for points on lower dimensional flats. *Tech. Report CGL-TR-77*, October 2013.

[9] S. Har-Peled and M. Mendel. Fast construction of nets in low dimensional metrics, and their applications. In *Proc. SoCG*, pages 150–158. 2005.

[10] P. Indyk and A. Naor. Nearest-neighbor-preserving embeddings. *ACM Trans. Algorithms*, 3(3), 2007.

[11] A. Magen. Dimensionality reductions that preserve volumes and distance to affine spaces, and their algorithmic applications. In *Proc. 6th Intern. Workshop RANDOM*, pages 239–253, 2002.

[12] M. Mitzenmacher and E. Upfal. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis.* Cambridge University Press, 2005.

[13] S. Vempala. Randomly-oriented k-d Trees Adapt to Intrinsic Dimension. In *Proc. Found. of Soft. Tech. & Theor. CS*, pages 48-57, 2012.