

Chapter 25

Approximate Nearest Neighbor (ANN) Search in High Dimensions

By Sarel Har-Peled, January 12, 2022^①

Possession of anything new or expensive only reflected a person's lack of theology and geometry; it could even cast doubts upon one's soul.

A confederacy of Dunces, John Kennedy
Toole

25.1. ANN on the hypercube

25.1.1. ANN for the hypercube and the Hamming distance

Definition 25.1.1. The set of points $\mathcal{H}^d = \{0,1\}^d$ is the ***d-dimensional hypercube***. A point $p = (p_1, \dots, p_d) \in \mathcal{H}^d$ can be interpreted, naturally, as a binary string $p_1 p_2 \dots p_d$. The ***Hamming distance*** $d_H(p, q)$ between $p, q \in \mathcal{H}^d$ is the number of coordinates where p and q disagree.

It is easy to verify that the Hamming distance, being the L_1 -norm, complies with the triangle inequality and is thus a metric.

As we saw previously, to solve the $(1 + \varepsilon)$ -ANN problem efficiently, it is sufficient to solve the ***approximate near neighbor*** problem. Namely, given a set P of n points in \mathcal{H}^d , a radius $r > 0$, and parameter $\varepsilon > 0$, we want to decide for a query point q whether $d_H(q, P) \leq r$ or $d_H(q, P) \geq (1 + \varepsilon)r$, where $d_H(q, P) = \min_{p \in P} d_H(q, p)$.

Definition 25.1.2. For a set P of points, a data-structure $\mathcal{D} = \mathcal{D}_{\approx\text{Near}}(P, r, (1 + \varepsilon)r)$ solves the ***approximate near neighbor*** problem if, given a query point q , the data-structure works as follows.

- NEAR: If $d_H(q, P) \leq r$, then \mathcal{D} outputs a point $p \in P$ such that $d_H(p, q) \leq (1 + \varepsilon)r$.
- FAR: If $d_H(q, P) \geq (1 + \varepsilon)r$, then \mathcal{D} outputs “ $d_H(q, P) \geq r$ ”.
- DON'T CARE: If $r \leq d(q, P) \leq (1 + \varepsilon)r$, then \mathcal{D} can return either of the above answers.

Given such a data-structure, one can construct a data-structure that answers the approximate nearest neighbor query using $O(\log(\varepsilon^{-1} \log d))$ queries using an approximate near neighbor data-structure. Indeed, the desired distance $d_H(q, P)$ is an integer number in the range $0, 1, \dots, d$. We can build a $\mathcal{D}_{\approx\text{Near}}$ data-structure for distances $(1 + \varepsilon)^i$, for $i = 1, \dots, M$, where $M = O(\varepsilon^{-1} \log d)$. Performing a binary search over these distances would resolve the approximate nearest neighbor query and requires $O(\log M)$ queries.

As such, in the following, we concentrate on constructing the approximate near neighbor data-structure (i.e., $\mathcal{D}_{\approx\text{Near}}$).

^①This work is licensed under the Creative Commons Attribution-Noncommercial 3.0 License. To view a copy of this license, visit <http://creativecommons.org/licenses/by-nc/3.0/> or send a letter to Creative Commons, 171 Second Street, Suite 300, San Francisco, California, 94105, USA.

25.1.2. Preliminaries

Definition 25.1.3. Consider a sequence m of k , not necessarily distinct, integers $i_1, i_2, \dots, i_k \in \llbracket d \rrbracket$, where $\llbracket d \rrbracket = \{1, \dots, d\}$. For a point $p = (p_1, \dots, p_d) \in \mathbb{R}^d$, its **projection** by m , denoted by mp is the point $(p_{i_1}, \dots, p_{i_k}) \in \mathbb{R}^k$. Similarly, the *projection* of a point set $P \subseteq \mathbb{R}^d$ by m is the point set $mP = \{mp \mid p \in P\}$.

Given two sequences $m = i_1, \dots, i_k$ and $u = j_1, \dots, j_{k'}$, let $m|u$ denote the *concatenated* sequence $m|u = i_1, \dots, i_k, j_1, \dots, j_{k'}$. Given a probability φ , a natural way to create such a projection, is to include the i th coordinate, for $i = 1, \dots, d$, with probability φ . Let \mathcal{D}_φ denote the distribution of such sequences of indices.

Definition 25.1.4. Let \mathcal{D}_φ^T denote the distribution resulting from concatenating T independent sequences sampled from \mathcal{D}_φ . The length of a sampled sequence is dT .

Observe that for a point $p \in \{0, 1\}^d$, and $M \in \mathcal{D}_\varphi^T$, the projection Mp might be higher dimensional than the original point p , as it might contain repeated coordinates of the original point.

25.1.2.1. Algorithm

Input. The input is a set P of n points in the hypercube $\{0, 1\}^d$, and parameters r and ε .

Preprocessing. We set parameters as follows:

$$\beta = \frac{1}{1 + \varepsilon} \in (0, 1), \quad \varphi = 1 - \exp\left(-\frac{1}{r}\right) \approx \frac{1}{r}, \quad T = \beta \ln n, \quad \text{and} \quad L = O(n^\beta \log n).$$

We randomly and independently pick L sequences $M_1, \dots, M_L \in \mathcal{D}_\varphi^T$. Next, the algorithm computes the point sets $Q_i = M_i P_i$, for $i = 1, \dots, L$, and stores them each in a hash table, denoted by D_i , for $i = 1, \dots, L$.

Answering a query. Given a query point $q \in \{0, 1\}^d$, the algorithm computes $q_i = M_i q$, for $i = 1, \dots, L$. From each D_i , the algorithm retrieves a list ℓ_i of all the points that collide with q_i . The algorithm scans the points in the lists ℓ_1, \dots, ℓ_L . If any of these points is in Hamming distance smaller than $(1 + \varepsilon)r$, the algorithm returns it as the desired near-neighbor (and stops). Otherwise, the algorithm returns that all the points in P are in distance at least r from q .

25.1.2.2. Analysis

Lemma 25.1.5. *Let K be a set of r marked/forbidden coordinates. The probability that a sequence $M = (m_1, \dots, m_T)$ sampled from \mathcal{D}_φ^T does not sample any of the coordinates of K is $1/n^\beta$. This probability increases if K contains fewer coordinates.*

Proof: For any i , the probability that m_i does not contain any of these coordinates is $(1 - \varphi)^r = \left(e^{-1/r}\right)^r = 1/e$. Since this experiment is repeated T times, the probability is $e^{-T} = e^{-\beta \ln n} = n^{-\beta}$. ■

Lemma 25.1.6. *We have the following:*

- (A) *Let p be the nearest-neighbor to q in P . If $d_H(q, p) \leq r$ then, with high probability, the data-structure returns a point that is in distance $\leq (1 + \varepsilon)r$ from q .*

(B) In expectation, the total number of points in ℓ_1, \dots, ℓ_L that are in distance $\geq (1 + \varepsilon)r$ from \mathbf{q} is $\leq L$.

Proof: (A) The good event here is that p and \mathbf{q} collide under one of the sequences of M_1, \dots, M_L . However, the probability that $M_i p = M_i \mathbf{q}$ is at least $1/n^\beta$, by [Lemma 25.1.5](#), as this is the probability that M_i does not sample any of the (at most r) coordinates where p and \mathbf{q} disagree. As such, the probability that all L data-structures fail (i.e., none of the lists ℓ_1, \dots, ℓ_L contains p), is at most $(1 - 1/n^\beta)^L < 1/n^{O(1)}$, as $L = O(n^\beta \log n)$.

(B) Let P_\geq be the set of points in P that are in distance $\geq (1 + \varepsilon)r$ from \mathbf{q} . For a point $q \in P_\geq$, with $\Delta = d_H(q, \mathbf{q})$, we have that the probability for $M \in \mathcal{D}_\varphi^T$ misses all the Δ coordinates, where q and \mathbf{q} differ, is

$$(1 - \varphi)^\Delta \leq (1 - \varphi)^{(1+\varepsilon)rT} = \left(e^{-1/r}\right)^{(1+\varepsilon)rT} = \exp(-(1 + \varepsilon)\beta \ln n) = \frac{1}{n},$$

as $\varphi = 1 - e^{-1/r}$, $T = \beta \ln n$, and $\beta = 1/(1 + \varepsilon)$. But then, for any i , we have

$$\mathbb{E}[|\ell_i|] = \sum_{p \in P_\geq} \Pr_{M_i} [M_i p = M_i \mathbf{q}] \leq |P_\geq| \frac{1}{n} \leq 1.$$

As such, the total number of far points in the lists is at most $L \cdot 1 = L$, implying the claim. \blacksquare

25.1.2.3. Running time

For each i , the query computes $M_i \mathbf{q}$ and that takes $O(dT) = O(d \log n)$ time. Repeated L times, this takes $O(Ld \log n)$ time overall. Let X be the random variable that is the number of points in the extracted lists that are in distance $\geq (1 + \varepsilon)r$ from the query point. The time to scan the lists is $O(d(X + 1))$, since the algorithm stops as soon as it finds a near point. As such, by [Lemma 25.1.6](#) (B), the expected query time is $O(Ld \log n + Ld) = O\left(dn^{1/(1+\varepsilon)} \log^2 n\right)$.

Improving the performance (a bit). Observe that for $M \in \mathcal{D}_\varphi^T$, and any two points $p, q \in \{0, 1\}^d$, all the algorithm cares about is whether $Mp = Mq$. As such, if a coordinate is probed many times by M , we might as well probe this coordinate only once. In particular, for a sequence $M \in \mathcal{D}_\varphi^T$, let $M' = \text{uniq}(M)$ be the projection sequence resulting from removing replications in M . Significantly, M' is only of length $\leq d$, and as such, computing $M'p$, for a point p , takes only $O(d)$ time. It is not hard to verify that one can also sample directly $\text{uniq}(M)$, for $M \in \mathcal{D}_\varphi^T$, in $O(d)$ time. This improves the query and processing by a logarithmic factor.

We thus get the following result.

Theorem 25.1.7. *Given a set P of n points in $\{0, 1\}^d$, and parameters r, ε , one can preprocess P in $O(dn^{1+1/(1+\varepsilon)} \log n)$ time and space, such that given a query point \mathbf{q} , the algorithm returns, in expected $O(dn^{1/(1+\varepsilon)} \log n)$ time, one of the following:*

- (A) a point $p \in P$ such that $d_H(\mathbf{q}, p) \leq (1 + \varepsilon)r$, or
- (B) the distance of \mathbf{q} from P is larger than r .

The algorithm may return either result if the distance of \mathbf{q} from P is in the range $[r, (1 + \varepsilon)r]$. The algorithm succeeds with high probability (per query).

One can also get a high-probability guarantee on the query time. For a parameter $\delta > 0$, create $O(\log \delta^{-1})$ LSH data-structures as above. Perform the query as above, except that when the query time exceeds (say) twice the expected time, move on to redo the query in the next LSH data-structure. The probability that the query had failed on one of these LSH data-structures is $\leq 1/2$, by Markov's inequality. As such, overall, the query time becomes $O(dn^{1/(1+\varepsilon)} \log n \log \delta^{-1})$, with probability $\geq 1 - \delta$.

25.2. LSH for the hypercube: An elaborate construction

We next present a similar scheme in a more systematic fashion – this would provide some intuition how we came up with the above construction.

25.2.0.1. On sense and sensitivity

Let $P = \{p_1, \dots, p_n\}$ be a subset of vertices of the hypercube in d dimensions. In the following we assume that $d = n^{O(1)}$. Let $r, \varepsilon > 0$ be two prespecified parameters. We are interested in building an approximate near neighbor data-structure (i.e., $\mathcal{D}_{\approx \text{Near}}$) for balls of radius r in the Hamming distance.

Definition 25.2.1. A family \mathcal{F} of functions (defined over \mathcal{H}^d) is $(r, R, \widehat{\alpha}, \widehat{\beta})$ -sensitive if for any $q, s \in \mathcal{H}^d$, we have the following

(A) If $q \in \mathbf{b}(s, r)$, then $\mathbb{P}[f(q) = f(s)] \geq \widehat{\alpha}$.

(B) If $q \notin \mathbf{b}(s, R)$, then $\mathbb{P}[f(q) = f(s)] \leq \widehat{\beta}$.

In (A) and (B), f is a randomly picked function from \mathcal{F} , $r < R$, and $\widehat{\alpha} > \widehat{\beta}$.

Intuitively, if we can construct an (r, R, α, β) -sensitive family, then we can distinguish between two points which are close together and two points which are far away from each other. Of course, the probabilities α and β might be very close to each other, and we need a way to do amplification.

A simple sensitive family. A priori it is not even clear such a sensitive family exists, but it turns out that the family randomly exposing one coordinate is sensitive.

Lemma 25.2.2. Let $f_i(p)$ denote the function that returns the i th coordinate of p , for $i = 1, \dots, d$. Consider the family of functions $\mathcal{F} = \{f_1, \dots, f_d\}$. Then, for any $r > 0$ and ε , the family \mathcal{F} is $(r, (1 + \varepsilon)r, \alpha, \beta)$ -sensitive, where $\alpha = 1 - r/d$ and $\beta = 1 - r(1 + \varepsilon)/d$.

Proof: If $q, s \in \{0, 1\}^d$ are within distance smaller than r from each other (under the Hamming distance), then they differ in at most r coordinates. The probability that a random $h \in \mathcal{F}$ would project into a coordinate that q and s agree on is $\geq 1 - r/d$.

Similarly, if $d_H(q, s) \geq (1 + \varepsilon)r$, then the probability that a random $h \in \mathcal{F}$ would map into a coordinate that q and s agree on is $\leq 1 - (1 + \varepsilon)r/d$. ■

A family with a large sensitivity gap. Let k be a parameter to be specified shortly, and consider the family of functions \mathcal{G} that concatenates k of the given functions. Formally, let

$$\mathcal{G} = \text{combine}(\mathcal{F}, k) = \left\{ g \mid g(p) = \left(f^1(p), \dots, f^k(p) \right), \text{ for } f^1, \dots, f^k \in \mathcal{F} \right\}$$

be the set of all such functions.

Lemma 25.2.3. For a (r, R, α, β) -sensitive family \mathcal{F} , the family $\mathcal{G} = \text{combine}(\mathcal{F}, k)$ is $(r, R, \alpha^k, \beta^k)$ -sensitive.

Proof: For two fixed points $q, s \in \mathcal{H}^d$ such that $d_H(q, s) \leq r$, we have that for a random $h \in \mathcal{F}$, we have $\mathbb{P}[h(q) = h(s)] \geq \alpha$. As such, for a random $g \in \mathcal{G}$, we have that

$$\begin{aligned} \mathbb{P}[g(q) = g(s)] &= \mathbb{P}[f^1(q) = f^1(s) \text{ and } f^2(q) = f^2(s) \text{ and } \dots \text{ and } f^k(q) = f^k(s)] \\ &= \prod_{i=1}^k \mathbb{P}[f^i(q) = f^i(s)] \geq \alpha^k. \end{aligned}$$

Similarly, if $d_H(q, s) > R$, then $\mathbb{P}[g(q) = g(s)] = \prod_{i=1}^k \mathbb{P}[f^i(q) = f^i(s)] \leq \beta^k$. ■

The above lemma implies that we can build a family that has a gap between the lower and upper sensitivities; namely, $\alpha^k/\beta^k = (\alpha/\beta)^k$ is arbitrarily large. The problem is that if α^k is too small, then we will have to use too many functions to detect whether or not there is a point close to the query point.

Nevertheless, consider the task of building a data-structure that finds all the points of $P = \{p_1, \dots, p_n\}$ that are equal, under a given function $g \in \mathcal{G} = \text{combine}(\mathcal{F}, k)$, to a query point. To this end, we compute the strings $g(p_1), \dots, g(p_n)$ and store them (together with their associated point) in a hash table (or a prefix tree). Now, given a query point q , we compute $g(q)$ and fetch from this data-structure all the strings equal to it that are stored in it. Clearly, this is a simple and efficient data-structure. All the points colliding with q would be the natural candidates to be the nearest neighbor to q .

By not storing the points explicitly, but using a pointer to the original input set, we get the following easy result.

Lemma 25.2.4. Given a function $g \in \mathcal{G} = \text{combine}(\mathcal{F}, k)$ (see [Lemma 25.2.3](#)) and a set $P \subseteq \mathcal{H}^d$ of n points, one can construct a data-structure, in $O(nk)$ time and using $O(nk)$ additional space, such that given a query point q , one can report all the points in $X = \{p \in P \mid g(p) = g(q)\}$ in $O(k + |X|)$ time.

Amplifying sensitivity. Our task is now to amplify the sensitive family we currently have. To this end, for two τ -dimensional points x and y , let $x \approx y$ be the Boolean function that returns *true* if there exists an index i such that $x_i = y_i$ and *false* otherwise. Now, the regular “=” operator requires vectors to be equal in all coordinates (i.e., it is equal to $\bigcap_i (x_i = y_i)$) while $x \approx y$ is $\bigcup_i (x_i = y_i)$. The previous construction of [Lemma 25.2.3](#) using this alternative equal operator provides us with the required amplification.

Lemma 25.2.5. Given an $(r, R, \alpha^k, \beta^k)$ -sensitive family \mathcal{G} , the family $\mathcal{H}_\approx = \text{combine}(\mathcal{G}, \tau)$ if one uses the \approx operator to check for equality is $(r, R, 1 - (1 - \alpha^k)^\tau, 1 - (1 - \beta^k)^\tau)$ -sensitive.

Proof: For two fixed points $q, s \in \mathcal{H}^d$ such that $d_H(q, s) \leq r$, we have, for a random $g \in \mathcal{G}$, that $\mathbb{P}[g(q) = g(s)] \geq \alpha^k$. As such, for a random $h \in \mathcal{H}_\approx$, we have that

$$\begin{aligned} \mathbb{P}[h(q) \approx h(s)] &= \mathbb{P}[g^1(q) = g^1(s) \text{ or } g^2(q) = g^2(s) \text{ or } \dots \text{ or } g^\tau(q) = g^\tau(s)] \\ &= 1 - \prod_{i=1}^{\tau} \mathbb{P}[g^i(q) \neq g^i(s)] \geq 1 - (1 - \alpha^k)^\tau. \end{aligned}$$

Similarly, if $d_H(q, s) > R$, then

$$\mathbb{P}[h(q) \approx h(s)] = 1 - \prod_{i=1}^{\tau} \mathbb{P}[g^i(q) \neq g^i(s)] \leq 1 - (1 - \beta^k)^\tau. \quad \blacksquare$$

To see the effect of [Lemma 25.2.5](#), it is useful to play with a concrete example. Consider an $(r, R, \alpha^k, \beta^k)$ -sensitive family where $\beta^k = \alpha^k/2$ and yet α^k is very small. Setting $\tau = 1/\alpha^k$, the resulting family is (roughly) $(r, R, 1 - 1/e, 1 - 1/\sqrt{e})$ -sensitive. Namely, the gap shrank, but the threshold sensitivity is considerably higher. In particular, it is now a constant, and the gap is also a constant.

Using [Lemma 25.2.5](#) as a data-structure to store P is more involved than before. Indeed, for a random function $h = (g^1, \dots, g^\tau) \in \mathcal{H}_\infty = \text{combine}(\mathcal{G}, \tau)$ building the associated data-structure requires us to build τ data-structures for each one of the functions g^1, \dots, g^τ , using [Lemma 25.2.4](#). Now, given a query point, we retrieve all the points of P that collide with each one of these functions, by querying each of these data-structures.

Lemma 25.2.6. *Given a function $h \in \mathcal{H}_\infty = \text{combine}(\mathcal{G}, \tau)$ (see [Lemma 25.2.5](#)) and a set $P \subseteq \mathcal{H}^d$ of n points, one can construct a data-structure, in $O(nk\tau)$ time and using $O(nk\tau)$ additional space, such that given a query point \mathbf{q} , one can report all the points in $X = \{p \in P \mid h(p) \approx h(\mathbf{q})\}$ in $O(k\tau + |X|)$ time.*

25.2.0.2. The near neighbor data-structure and handling a query

We construct the data-structure \mathcal{D} of [Lemma 25.2.6](#) with parameters k and τ to be determined shortly, for a random function $h \in \mathcal{H}_\infty$. Given a query point \mathbf{q} , we retrieve all the points that collide with h and compute their distance to the query point. Next, scan these points one by one and compute their distance to \mathbf{q} . As soon as encountering a point $s \in P$ such that $d_H(\mathbf{q}, s) \leq R$, the data-structures returns *true* together with s .

Let's assume that we know that the expected number of points of $P \setminus \mathbf{b}(\mathbf{q}, R)$ (i.e., $R = (1 + \varepsilon)r$) that will collide with \mathbf{q} in \mathcal{D} is in expectation L (we will figure out the value of L below). To ensure the worst case query time, the query would abort after checking $4L + 1$ points and would return *false*. Naturally, the data-structure would also return false if all points encountered have distance larger than R from \mathbf{q} .

Clearly, the query time of this data-structure is $O(k\tau + dL)$.

We are left with the task of fine-tuning the parameters τ and k to get the fastest possible query time, while the data-structure has reasonable probability to succeed. Figuring the right values is technically tedious, and we do it next.

25.2.0.3. Setting the parameters

If there exists $p \in P$ such that $d_H(\mathbf{q}, p) \leq r$, then the probability of this point to collide with \mathbf{q} under the function h is $\phi \geq 1 - (1 - \alpha^k)^\tau$. Let us demand that this data-structure succeeds with probability $\geq 3/4$. To this end, we set

$$\tau = 4 \lceil 1/\alpha^k \rceil \implies \phi \geq 1 - (1 - \alpha^k)^\tau \geq 1 - \exp(-\alpha^k \tau) \geq 1 - \exp(-4) \geq 3/4, \quad (25.1)$$

since $1 - x \leq \exp(-x)$, for $x \geq 0$.

Lemma 25.2.7. *The expected number of points of $P \setminus \mathbf{b}(\mathbf{q}, R)$ colliding with the query point is $L = O(n(\beta/\alpha)^k)$, where $R = (1 + \varepsilon)r$.*

Proof: Consider the points in $P \setminus \mathbf{b}(\mathbf{q}, R)$. We would like to bound the number of points of this set that collide with the query point. Observe that in this case, the probability of a point $p \in P \setminus \mathbf{b}(\mathbf{q}, R)$ to

collide with the query point is

$$\begin{aligned} \leq \psi &= 1 - (1 - \beta^k)^\tau = (1 - (1 - \beta^k)) \left(1 + (1 - \beta^k) + (1 - \beta^k)^2 + \dots + (1 - \beta^k)^{\tau-1} \right) \\ &\leq \beta^k \tau \leq 8 \left(\frac{\beta}{\alpha} \right)^k, \end{aligned}$$

as $\tau = 4 \lceil 1/\alpha^k \rceil$ and $\alpha, \beta \in (0, 1)$. Namely, the expected number of points of $P \setminus \mathbf{b}(\mathbf{q}, R)$ colliding with the query point is $\leq \psi n$. \blacksquare

By [Lemma 25.2.6](#), extracting the $O(L)$ points takes $O(k\tau + L)$ time. Computing the distance of the query time for each one of these points takes $O(k\tau + Ld)$ time. As such, by [Lemma 25.2.7](#), the query time is

$$O(k\tau + Ld) = O\left(k\tau + nd(\beta/\alpha)^k\right).$$

To minimize this query time, we “approximately” solve the equation requiring the two terms, in the above bound, to be equal (we ignore d since, intuitively, it should be small compared to n). We get that

$$k\tau = n(\beta/\alpha)^k \rightsquigarrow \frac{k}{\alpha^k} \approx n \frac{\beta^k}{\alpha^k} \implies k \approx n\beta^k \rightsquigarrow 1/\beta^k \approx n \implies k \approx \ln_{1/\beta} n.$$

Thus, setting $k = \ln_{1/\beta} n$, we have that $\beta^k = 1/n$ and, by [Eq. \(25.1\)](#), that

$$\tau = 4 \lceil 1/\alpha^k \rceil = \exp\left(\frac{\ln n}{\ln 1/\beta} \ln 1/\alpha\right) = O(n^\rho), \quad \text{for } \rho = \frac{\ln 1/\alpha}{\ln 1/\beta}. \quad (25.2)$$

As such, to minimize the query time, we need to minimize ρ .

Lemma 25.2.8. (A) For $x \in [0, 1)$ and $t \geq 1$ such that $1 - tx > 0$ we have $\frac{\ln(1-x)}{\ln(1-tx)} \leq \frac{1}{t}$.

(B) For $\alpha = 1 - r/d$ and $\beta = 1 - r(1 + \varepsilon)/d$, we have that $\rho = \frac{\ln 1/\alpha}{\ln 1/\beta} \leq \frac{1}{1 + \varepsilon}$.

Proof: (A) Since $\ln(1-tx) < 0$, it follows that the claim is equivalent to $t \ln(1-x) \geq \ln(1-tx)$. This in turn is equivalent to

$$g(x) \equiv (1-tx) - (1-x)^t \leq 0.$$

This is trivially true for $x = 0$. Furthermore, taking the derivative, we see $g'(x) = -t + t(1-x)^{t-1}$, which is non-positive for $x \in [0, 1)$ and $t > 0$. Therefore, g is non-increasing in the interval of interest, and so $g(x) \leq 0$ for all values in this interval.

(B) Indeed $\rho = \frac{\ln 1/\alpha}{\ln 1/\beta} = \frac{\ln \alpha}{\ln \beta} = \frac{\ln \frac{d-r}{d}}{\ln \frac{d-(1+\varepsilon)r}{d}} = \frac{\ln(1 - \frac{r}{d})}{\ln(1 - (1 + \varepsilon)\frac{r}{d})} \leq \frac{1}{1 + \varepsilon}$, by part (A). \blacksquare

In the following, it would be convenient to consider d to be considerably larger than r . This can be ensured by (conceptually) padding the points with fake coordinates that are all zero. It is easy to verify that this “hack” would not affect the algorithm’s performance in any way and it is just a trick to make our analysis simpler. In particular, we assume that $d > 2(1 + \varepsilon)r$.

Lemma 25.2.9. For $\alpha = 1 - r/d$, $\beta = 1 - r(1 + \varepsilon)/d$, n and d as above, we have that (i) $\tau = O\left(n^{1/(1+\varepsilon)}\right)$, (ii) $k = O(\ln n)$, and (iii) $L = O\left(n^{1/(1+\varepsilon)}\right)$.

Proof: By Eq. (25.1), $\tau = 4\lceil 1/\alpha^k \rceil = O(n^\rho) = O\left(n^{1/(1+\varepsilon)}\right)$, by Lemma 25.2.8(B).

Now, $\beta = 1 - r(1 + \varepsilon)/d \leq 1/2$, since we assumed that $d > 2(1 + \varepsilon)r$. As such, we have $k = \ln_{1/\beta} n = \frac{\ln n}{\ln 1/\beta} = O(\ln n)$.

By Lemma 25.2.7, $L = O\left(n(\beta/\alpha)^k\right)$. Now $\beta^k = 1/n$ and as such $L = O(1/\alpha^k) = O(\tau) = O\left(n^{1/(1+\varepsilon)}\right)$. ■

25.2.0.4. The result

Theorem 25.2.10. Given a set P of n points on the hypercube \mathcal{H}^d and parameters $\varepsilon > 0$ and $r > 0$, one can build a data-structure $\mathcal{D} = \mathcal{D}_{\approx \text{Near}}(P, r, (1 + \varepsilon)r)$ that solves the approximate near neighbor problem (see Definition 25.1.2). The data-structure answers a query successfully with high probability. In addition we have the following:

(A) The query time is $O\left(dn^{1/(1+\varepsilon)} \log n\right)$.

(B) The preprocessing time to build this data-structure is $O\left(n^{1+1/(1+\varepsilon)} \log^2 n\right)$.

(C) The space required to store this data-structure is $O\left(nd + n^{1+1/(1+\varepsilon)} \log^2 n\right)$.

Proof: Our building block is the data-structure described above. By Markov's inequality, the probability that the algorithm has to abort because of too many collisions with points of $P \setminus \mathbf{b}(\mathbf{q}, (1 + \varepsilon)r)$ is bounded by $1/4$ (since the algorithm tries $4L+1$ points). Also, if there is a point inside $\mathbf{b}(\mathbf{q}, r)$, the algorithm would find it with probability $\geq 3/4$, by Eq. (25.1). As such, with probability at least $1/2$ this data-structure returns the correct answer in this case. By Lemma 25.2.6, the query time is $O(k\tau + Ld)$.

This data-structure succeeds only with constant probability. To achieve high probability, we construct $O(\log n)$ such data-structures and perform the near neighbor query in each one of them. As such, the query time is

$$O((k\tau + Ld) \log n) = O\left(n^{1/(1+\varepsilon)} \log^2 n + dn^{1/(1+\varepsilon)} \log n\right) = O\left(dn^{1/(1+\varepsilon)} \log n\right),$$

by Lemma 25.2.9 and since $d = \Omega(\lg n)$ if P contains n distinct points of \mathcal{H}^d .

As for the preprocessing time, by Lemma 25.2.6 and Lemma 25.2.9, we have

$$O(nk\tau \log n) = O\left(n^{1+1/(1+\varepsilon)} \log^2 n\right).$$

Finally, this data-structure requires $O(dn)$ space to store the input points. Specifically, by Lemma 25.2.6, we need an additional $O(nk\tau \log n) = O\left(n^{1+1/(1+\varepsilon)} \log^2 n\right)$ space. ■

In the hypercube case, when $d = n^{O(1)}$, we can build $M = O(\log_{1+\varepsilon} d) = O(\varepsilon^{-1} \log d)$ such data-structures such that $(1 + \varepsilon)$ -ANN can be answered using binary search on those data-structures which correspond to radii r_1, \dots, r_M , where $r_i = (1 + \varepsilon)^i$, for $i = 1, \dots, M$.

Theorem 25.2.11. Given a set P of n points on the hypercube \mathcal{H}^d (where $d = n^{O(1)}$) and a parameter $\varepsilon > 0$, one can build a data-structure to answer approximate nearest neighbor queries (under the Hamming distance) using $O\left(dn + n^{1/(1+\varepsilon)} \varepsilon^{-1} \log^2 n \log d\right)$ space, such that given a query point \mathbf{q} , one can return a $(1 + \varepsilon)$ -ANN in P (under the Hamming distance) in $O\left(dn^{1/(1+\varepsilon)} \log n \log(\varepsilon^{-1} \log d)\right)$ time. The result returned is correct with high probability.

Remark 25.2.12. The result of [Theorem 25.2.11](#) needs to be oblivious to the queries used. Indeed, for any instantiation of the data-structure of [Theorem 25.2.11](#) there exist query points for which it would fail.

In particular, formally, if we perform a sequence of ANN queries using such a data-structure, where the queries depend on earlier returned answers, then the guarantee of a high probability of success is no longer implied by the above analysis (it might hold because of some other reasons, naturally).

25.3. LSH and ANN in Euclidean space

25.3.1. Preliminaries

Lemma 25.3.1. *Let $X = (X_1, \dots, X_d)$ be a vector of d independent variables which have normal distribution \mathbf{N} , and let $v = (v_1, \dots, v_d) \in \mathbb{R}^d$. We have that $\langle v, X \rangle = \sum_i v_i X_i$ is distributed as $\|v\| Z$, where $Z \sim \mathbf{N}$.*

Proof: By [Lemma 25.5.1_{p355}](#) the point X has multi-dimensional normal distribution \mathbf{N}^d . As such, if $\|v\| = 1$, then this holds by the symmetry of the normal distribution. Indeed, let $e_1 = (1, 0, \dots, 0)$. By the symmetry of the d -dimensional normal distribution, we have that $\langle v, X \rangle \sim \langle e_1, X \rangle = X_1 \sim \mathbf{N}$.

Otherwise, $\langle v, X \rangle / \|v\| \sim \mathbf{N}$, and as such $\langle v, X \rangle \sim N(0, \|v\|^2)$, which is indeed the distribution of $\|v\| Z$. ■

Definition 25.3.2. A distribution \mathcal{D} over \mathbb{R} is called *p -stable* if there exists $p \geq 0$ such that for any n real numbers v_1, \dots, v_n and n independent variables X_1, \dots, X_n with distribution \mathcal{D} , the random variable $\sum_i v_i X_i$ has the same distribution as the variable $(\sum_i |v_i|^p)^{1/p} X$, where X is a random variable with distribution \mathcal{D} .

By [Lemma 25.3.1](#), the normal distribution is a *2-stable distribution*.

25.3.2. Locality sensitive hashing (LSH)

Let p and q be two points in \mathbb{R}^d . We want to perform an experiment to decide if $\|p - q\| \leq 1$ or $\|p - q\| \geq \eta$, where $\eta = 1 + \varepsilon$. We will randomly choose a vector \vec{v} from the d -dimensional normal distribution \mathbf{N}^d (which is 2-stable). Next, let r be a parameter, and let t be a random number chosen uniformly from the interval $[0, r]$. For $p \in \mathbb{R}^d$, consider the random hash function

$$h(p) = \left\lfloor \frac{\langle p, \vec{v} \rangle + t}{r} \right\rfloor. \quad (25.3)$$

Assume that the distance between p and q is η and the distance between the projection of the two points to the direction \vec{v} is β . Then, the probability that p and q get the same hash value is $\max(1 - \beta/r, 0)$, since this is the probability that the random sliding will not separate them. Indeed, consider the line through \vec{v} to be the x -axis, and assume q is projected to r and p is projected to $r - \beta$ (assuming $r \geq \beta$). Clearly, q and p get mapped to the same value by $h(\cdot)$ if and only if $t \in [0, r - \beta]$, as claimed.

As such, we have that the probability of collusion is

$$\alpha(\eta, r) = \mathbb{P}[h(p) = h(q)] = \int_{\beta=0}^r \mathbb{P}[|\langle p, \vec{v} \rangle - \langle q, \vec{v} \rangle| = \beta] \left(1 - \frac{\beta}{r}\right) d\beta.$$

However, since \vec{v} is chosen from a 2-stable distribution, we have that $Z = \langle p, \vec{v} \rangle - \langle q, \vec{v} \rangle = \langle p - q, \vec{v} \rangle \sim \mathbf{N}\left(0, \|p - q\|^2\right)$. Since we are considering the absolute value of the variable, we need to multiply this by two. Thus, we have

$$\alpha(\eta, r) = \int_{\beta=0}^r \frac{2}{\sqrt{2\pi\eta}} \exp\left(-\frac{\beta^2}{2\eta^2}\right) \left(1 - \frac{\beta}{r}\right) d\beta,$$

by plugging in the density of the normal distribution for Z . Intuitively, we care about the difference $\alpha(1 + \varepsilon, r) - \alpha(1, r)$, and we would like to maximize it as much as possible (by choosing the right value of r). Unfortunately, this integral is unfriendly, and we have to resort to numerical computation.

Now, we are going to use this hashing scheme for constructing locality sensitive hashing, as in the hypercube case, and as such we care about the ratio

$$\rho(1 + \varepsilon) = \min_r \frac{\log(1/\alpha(1, r))}{\log(1/\alpha(1 + \varepsilon, r))};$$

see Eq. (25.2)_{p350}.

The following is verified using numerical computations (using a computer).

Lemma 25.3.3 ([DIIM04]). *One can choose r , such that $\rho(1 + \varepsilon) \leq \frac{1}{1 + \varepsilon}$.*

Lemma 25.3.3 implies that the hash functions defined by Eq. (25.3) are $(1, 1 + \varepsilon, \alpha', \beta')$ -sensitive and, furthermore, $\rho = \frac{\log(1/\alpha')}{\log(1/\beta')} \leq \frac{1}{1 + \varepsilon}$, for some values of α' and β' . As such, we can use this hashing family to construct an approximate near neighbor data-structure $\mathcal{D}_{\approx \text{Near}}(P, r, (1 + \varepsilon)r)$ for the set P of points in \mathbb{R}^d . Following the same argumentation of **Theorem 25.2.10**, we have the following.

Theorem 25.3.4. *Given a set P of n points in \mathbb{R}^d and parameters $\varepsilon > 0$ and $r > 0$, one can build a $\mathcal{D}_{\approx \text{Near}} = \mathcal{D}_{\approx \text{Near}}(P, r, (1 + \varepsilon)r)$, such that given a query point \mathbf{q} , one can decide:*

(A) *If $\mathbf{b}(\mathbf{q}, r) \cap P \neq \emptyset$, then $\mathcal{D}_{\approx \text{Near}}$ returns a point $u \in P$, such that $d_H(u, \mathbf{q}) \leq (1 + \varepsilon)r$.*

(B) *If $\mathbf{b}(\mathbf{q}, (1 + \varepsilon)r) \cap P = \emptyset$, then $\mathcal{D}_{\approx \text{Near}}$ returns the result that no point is within distance $\leq r$ from \mathbf{q} .*

In any other case, any of the answers is correct. The query time is $O(dn^{1/(1+\varepsilon)} \log n)$ and the space used is $O(dn + n^{1+1/(1+\varepsilon)} \log n)$. The result returned is correct with high probability.

25.3.3. ANN in high-dimensional euclidean space

Unlike the binary hypercube case, where we could just do direct binary search on the distances, here we need to use the reduction from ANN to near neighbor queries. We will need the following result which is implied by **Theorem 25.5.2**_{p355} and **Theorem 25.5.3**_{p355}.

Corollary 25.3.5. *Given a set P of n points in \mathbb{R}^d , one can construct a data-structure \mathcal{D} that answers $(1 + \varepsilon)$ -ANN queries, by performing $O(\log(n/\varepsilon))$ $(1 + \varepsilon)$ -approximate near neighbor queries. The total number of points stored at these approximate near neighbor data-structure of \mathcal{D} is $O(n\varepsilon^{-1} \log(n/\varepsilon))$.*

Constructing the data-structure of **Corollary 25.3.5** requires building a low quality HST; to this end we can use the fast HST construction of **Theorem 25.5.4**_{p355} that builds $2\sqrt{dn}^5$ -approximate HST of n points in \mathbb{R}^d in $O(dn \log n)$ expected time.

25.3.3.1. The overall result

Plugging [Theorem 25.3.4](#) into [Corollary 25.3.5](#), we have the following:

Theorem 25.3.6. *Given a set P of n points in \mathbb{R}^d and parameters $\varepsilon > 0$ and $r > 0$, one can build an ANN data-structure using*

$$O\left(dn + n^{1+1/(1+\varepsilon)}\varepsilon^{-2}\log^3(n/\varepsilon)\right)$$

space, such that given a query point q , one can return a $(1 + \varepsilon)$ -ANN in P in

$$O\left(dn^{1/(1+\varepsilon)}(\log n)\log\frac{n}{\varepsilon}\right)$$

time. The result returned is correct with high probability.

The construction time is $O\left(dn^{1+1/(1+\varepsilon)}\varepsilon^{-2}\log^3(n/\varepsilon)\right)$.

Proof: We compute, in $O(nd\log n)$ expected time, a low quality HST using [Theorem 25.5.4_{p355}](#). Using this HST, we can construct the data-structure \mathcal{D} of [Corollary 25.3.5](#), where we do not compute the $\mathcal{D}_{\approx\text{Near}}$ data-structures. We next traverse the tree \mathcal{D} and construct the $\mathcal{D}_{\approx\text{Near}}$ data-structures using [Theorem 25.3.4](#).

We only need to prove the bound on the space. Observe that we need to store each point only once, since other places can refer to the point by a pointer. Thus, this is the $O(dn)$ space requirement. The other term comes from plugging the bound of [Corollary 25.3.5](#) into the bound of [Theorem 25.3.4](#). ■

25.4. Bibliographical notes

The construction in [Section 25.2](#) is from a work of Har-Peled and Mahabadi [[HM17](#)] – it is a simplification of what follows later on in the chapter. In particular, [Section 25.2](#) follows the exposition of Indyk and Motwani [[IM98](#)]. Kushilevitz et al. [[KOR00](#)] offered an alternative data-structure with somewhat inferior performance. It is quite surprising that one can perform approximate nearest neighbor queries in high dimensions in time and space polynomial in the dimension (which is sublinear in the number of points). One can reduce the approximate near neighbor in Euclidean space to the same question on the hypercube “directly” (we show the details below). However, doing the LSH directly on the Euclidean space is more efficient.

The value of the results shown in this chapter depends to a large extent on the reader’s perspective. Indeed, for a small value of $\varepsilon > 0$, the query time $O(dn^{1/(1+\varepsilon)})$ is very close to linear dependency on n and is almost equivalent to just scanning the points. Thus, from the low-dimensional perspective, where ε is assumed to be small, this result is slightly sublinear. On the other hand, if one is willing to pick ε to be large (say 10), then the result is clearly better than the naive algorithm, suggesting running time for an ANN query which takes (roughly) $O(n^{1/11})$ time.

The idea of doing locality sensitive hashing directly on the Euclidean space, as done in [Section 25.3](#), is not shocking after seeing the Johnson-Lindenstrauss lemma. Our description follows the paper of Datar et al. [[DIIM04](#)]. In particular, the current analysis which relies on computerized estimates is far from being satisfactory. It would be nice to have a simpler and more elegant scheme for this case. This is an open problem for further research.

Currently, the best LSH construction in \mathbb{R}^d is due to Andoni and Indyk [[AI06](#)]. Its space usage is bounded by $O\left(dn + n^{1+1/(1+\varepsilon)^2+o(1)}\right)$ and its query time is bounded by $O\left(dn^{1/(1+\varepsilon)^2+o(1)}\right)$. This (almost) matches the lower bound of Motwani et al. [[MNP06](#)]. For a nice survey on LSH see [[AI08](#)].

From approximate near neighbor in \mathbb{R}^d to approximate near neighbor on the hypercube.

The reduction is quite involved, and we only sketch the details. Let P be a set of n points in \mathbb{R}^d . We first reduce the dimension to $k = O(\varepsilon^{-2} \log n)$ using the Johnson-Lindenstrauss lemma. Next, we embed this space into $\ell_1^{k'}$ (this is the space \mathbb{R}^k , where distances are the L_1 metric instead of the regular L_2 metric), where $k' = O(k/\varepsilon^2)$. This can be done with distortion $(1 + \varepsilon)$.

Let Q' be the resulting set of points in $\mathbb{R}^{k'}$. We want to solve approximate near neighbor queries on this set of points, for radius r . As a first step, we partition the space into cells by taking a grid with sidelength (say) $k'r$ and randomly translating it, clipping the points inside each grid cell. It is now sufficient to solve the approximate near neighbor problem inside this grid cell (which has bounded diameter as a function of r), since with small probability the result would be correct. We amplify the probability by repeating this a polylogarithmic number of times.

Thus, we can assume that P is contained inside a cube of sidelength $\leq k'nr$, it is in $\mathbb{R}^{k'}$, and the distance metric is the L_1 metric. We next snap the points of P to a grid of sidelength (say) $\varepsilon r/k'$. Thus, every point of P now has an integer coordinate, which is bounded by a polynomial in $\log n$ and $1/\varepsilon$. Next, we write the coordinates of the points of P using unary notation. (Thus, a point $(2, 5)$ would be written as $(00011, 11111)$ assuming the number of bits for each coordinates is 5.) It is now easy to verify that the Hamming distance on the resulting strings is equivalent to the L_1 distance between the points.

Thus, we can solve the near neighbor problem for points in \mathbb{R}^d by solving it on the hypercube under the Hamming distance.

See Indyk and Motwani [IM98] for more details.

We have only scratched the surface of proximity problems in high dimensions. The interested reader is referred to the survey by Indyk [Aga04] for more information.

25.5. From previous lectures

Lemma 25.5.1. (A) *The multi-dimensional normal distribution is symmetric; that is, for any two points $p, q \in \mathbb{R}^d$ such that $\|p\| = \|q\|$ we have that $g(p) = g(q)$, where $g(\cdot)$ is the density function of the multi-dimensional normal distribution \mathbf{N}^d .*

(B) *The projection of the normal distribution on any direction is a one-dimensional normal distribution.*

(C) *Picking d variables X_1, \dots, X_d using the one-dimensional normal distribution \mathbf{N} results in a point (X_1, \dots, X_d) that has multi-dimensional normal distribution \mathbf{N}^d .*

Theorem 25.5.2. *Given a set P of n points in \mathbb{R}^d , one can compute a set \mathcal{B} of $O(n\varepsilon^{-1} \log n)$ balls, such that answering $(1 + \varepsilon)$ -ANN queries on P can be done by performing a single target query on \mathcal{B} . Computing this set of balls takes $O(dn \log n + n\varepsilon^{-1} \log n)$ time.*

Furthermore, one can $(1 + \varepsilon/16)$ -approximate each ball of \mathcal{B} and answer $(1 + \varepsilon)$ -ANN queries by performing a single target query on the resulting set of approximate balls.

Alternatively, one can answer ANN queries using $O(\log(n/\varepsilon))$ approximate near neighbor queries.

Theorem 25.5.3. *Given a set P of n points in a metric space \mathcal{M} and an HST \mathcal{H} of P that t -approximates \mathcal{M} , for $t = n^{O(1)}$, one can construct a data-structure \mathcal{D} that answers $(1 + \varepsilon)$ -ANN queries, by performing $O(\log(n/\varepsilon))$ near neighbor queries. The data-structure \mathcal{D} is realized by a set \mathcal{B} of $O(n\varepsilon^{-1} \log^2 n)$ balls, and answering $(1 + \varepsilon)$ -ANN queries on P can be done by performing a target query on \mathcal{B} .*

Theorem 25.5.4. *Given a set P of n points in \mathbb{R}^d , for $d \leq n$, one can compute a $2\sqrt{d}n^5$ -approximate HST of P in $O(dn \log n)$ expected time.*

Bibliography

- [Aga04] P. K. Agarwal. *Range searching*. *Handbook of Discrete and Computational Geometry*. Ed. by J. E. Goodman and J. O'Rourke. 2nd. Boca Raton, FL, USA: CRC Press LLC, 2004. Chap. 36, pp. 809–838.
- [AI06] A. Andoni and P. Indyk. *Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions*. *Proc. 47th Annu. IEEE Sympos. Found. Comput. Sci. (FOCS)*, 459–468, 2006.
- [AI08] A. Andoni and P. Indyk. *Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions*. *Commun. ACM*, 51(1): 117–122, 2008.
- [DIIM04] M. Datar, N. Immorlica, P. Indyk, and V. S. Mirrokni. *Locality-sensitive hashing scheme based on p -stable distributions*. *Proc. 20th Annu. Sympos. Comput. Geom. (SoCG)*, 253–262, 2004.
- [HM17] S. Har-Peled and S. Mahabadi. *LSH on the Hypercube Revisited*. *ArXiv e-prints*, 2017. arXiv: [1704.02546](https://arxiv.org/abs/1704.02546) [cs.CG].
- [IM98] P. Indyk and R. Motwani. *Approximate nearest neighbors: Towards removing the curse of dimensionality*. *Proc. 30th Annu. ACM Sympos. Theory Comput. (STOC)*, 604–613, 1998.
- [KOR00] E. Kushilevitz, R. Ostrovsky, and Y. Rabani. *Efficient search for approximate nearest neighbor in high dimensional spaces*. *SIAM J. Comput.*, 2(30): 457–474, 2000.
- [MNP06] R. Motwani, A. Naor, and R. Panigrahi. *Lower bounds on locality sensitive hashing*. *Proc. 22nd Annu. Sympos. Comput. Geom. (SoCG)*, 154–157, 2006.