1 ANN on the Hypercube

1.1 Hypercube and Hamming distance

Definition 1.1 The set of points $\mathcal{H}^d = \{0, 1\}^d$ is the $d$-dimensional hypercube. A point $p = (p_1, \ldots, p_d) \in \mathcal{H}^d$ can be interpreted, naturally, as a binary string $p_1p_2 \ldots p_d$. 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 comply with the triangle inequality, and is as such a metric.

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

Definition 1.2 For a set $P$ of points, a data-structure $\text{NNbr}_\approx(P, r, (1 + \varepsilon)r)$ solves the approximate near neighbor problem, if given a query point $q$, the data-structure works as follows.

- If $d(q, P) \leq r$ then $\text{NNbr}_\approx$ outputs a point $p \in P$ such that $d(p, q) \leq (1 + \varepsilon)r$.
- If $d(q, P) \geq (1 + \varepsilon)r$, in this case $\text{NNbr}_\approx$ outputs that “$d(q, P) \geq r$”.
- If $r \leq d(q, P) \leq (1 + \varepsilon)r$, either of the above answers is acceptable.

Given such a data-structure $\text{NNbr}_\approx(P, r, (1 + \varepsilon)r)$, one can construct a data-structure that answers ANN using $O(\log(n/\varepsilon))$ queries.

1.2 Constructing $\text{NNbr}$ for the Hamming cube

Let $P = \{p_1, \ldots, p_n\}$ be a subset of vertices of the hypercube in $d$ dimensions. Let $r, \varepsilon > 0$ be two prespecified parameters. We are interested in building an $\text{NNbr}_\approx$ for balls of radius $r$ in the Hamming distance.

Definition 1.3 Let $U$ be a (small) positive integer. A family $\mathcal{F} = \{h : S \rightarrow [0, U]\}$ of functions, is an $(r, R, \alpha, \beta)$-sensitive if for any $u, q \in S$, we have:

- If $u \in b(q, r)$ then $\Pr[h(u) = h(v)] \geq \alpha$.
- If $u \notin b(q, R)$ then $\Pr[h(u) = h(v)] \leq \beta$.

where $h$ is randomly picked from $\mathcal{F}$, $r < R$, and $\alpha > \beta$. 
Intuitively, if we can construct a \((r, R, \alpha, \beta)\)-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 \(\alpha\) and \(\beta\) might be very close to each other, and we need a way to do amplification.

**Lemma 1.4** For the hypercube \(\mathcal{H}^d = \{0, 1\}^d\), and a point \(b = (b_1, \ldots, b_d) \in \mathcal{H}^d\), let \(\mathcal{F}\) be the set of functions
\[
\left\{ h_i(b) = b_i \mid b = (b_1, \ldots, b_d) \in \mathcal{H}^d, \text{ for } i = 1, \ldots, d \right\}.
\]
Then for any \(r, \epsilon\), the family \(\mathcal{F}\) is \((r, (1 + \epsilon)r, 1 - r/d, 1 - r(1+\epsilon)/d)\)-sensitive.

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

Similarly, if \(d_H(u, v) \geq (1 + \epsilon)r\) then the probability that \(h\) would map into a coordinate that \(u\) and \(v\) agree on is \(\leq 1 - (1 + \epsilon)r/d\).

Let \(k\) be a parameter to be specified shortly. Let
\[
\mathcal{G}(\mathcal{F}) = \left\{ g : \{0, 1\}^d \rightarrow \{0, 1\}^k \mid g(u) = (h^1(u), \ldots, h^k(u)), \text{ for } h^1, \ldots, h^k \in \mathcal{F} \right\}.
\]
Intuitively, \(\mathcal{G}\) is a family that extends \(\mathcal{F}\) by probing into \(k\) coordinates instead of only one coordinate.

### 1.3 Construction the near-neighbor data-structure

Let \(\tau\) be (yet another) parameter to be specified shortly. We pick \(g_1, \ldots, g_\tau\) functions randomly and uniformly from \(\mathcal{G}\). For each point \(u \in P\) compute \(g_1(u), \ldots, g_\tau(u)\). We construct a hash table \(H_i\) to store all the values of \(g_i(p_1), \ldots, g_i(p_n)\), for \(i = 1, \ldots, \tau\).

Given a query point \(q \in \mathcal{H}^d\), we compute \(p_1(q), \ldots, p_\tau(q)\), and retrieve all the points stored in those buckets in the hash tables \(H_1, \ldots, H_\tau\), respectively. For every point retrieved, we compute its distance to \(q\), and if this distance is \(\leq (1 + \epsilon)r\), we return it. If we encounter more than \(4\tau\) points we abort, and return ‘fail’. If no “close” point is encountered, the search returns ‘fail’.

We choose \(k\) and \(\tau\) so that with constant probability (say larger than half) we have the following two properties:

1. If there is a point \(u \in P\), such that \(d_H(u, q) \leq r\), then \(g_j(u) = g_j(q)\) for some \(j\).
2. Otherwise (i.e., \(d_H(u, q) \geq (1 + \epsilon)r\)), the total number of points colliding with \(q\) in the \(\tau\) hash tables, is smaller than \(4\tau\).

Given a query point, \(q \in \mathcal{H}^d\), we need to perform \(\tau\) probes into \(\tau\) hash tables, and retrieve at most \(4\tau\) results. Overall this takes \(O(d^0(1)\tau)\) time.

**Lemma 1.5** If there is a \((r, (1 + \epsilon)r, \alpha, \beta)\)-sensitive family \(\mathcal{F}\) of functions for the hypercube, then there exists a NNbr\(_\approx\)(\(P, r, (1 + \epsilon)r\)) which uses \(O(dn + n^{1+\rho})\) space and \(O(n^{\rho})\) hash probes for each query, where
\[
\rho = \frac{\ln 1/\alpha}{\ln 1/\beta}.
\]
This data-structure succeeds with constant probability.
Proof: It suffices to ensure that properties (1) and (2) holds with probability larger than half.

Set $k = \log_{1/\beta} n = \frac{\ln n}{\ln(1/\beta)}$, then the probability that for a random hash function $g \in \mathcal{G}(\mathcal{F})$, we have $g(p) = g(q)$ for $p \in P \setminus b(q, (1 + \varepsilon)r)$ is at most

$$\Pr[g(p') = g(q)] \leq \beta^k \leq \exp\left(\ln(\beta) \cdot \frac{\ln n}{\ln(1/\beta)}\right) \leq \frac{1}{n}.$$

Thus, the expected number of elements from $P \setminus b(q, (1 + \varepsilon)r)$ colliding with $q$ in the $j$th hash table $H_j$ is bounded by one. In particular, the overall expected number of such collisions in $H_1, \ldots, H_r$ is bounded by $\tau$. By the Markov inequality we have that the probability that the collusions number exceeds $4\tau$ is less than 1/4; therefore the probability that the property (2) holds is $\geq 3/4$.

Next, for a point $p \in b(q, r)$, consider the probability of $g_j(p) = g_j(q)$, for a fixed $j$. Clearly, it is bounded from below by

$$\geq \alpha^k = \alpha^{\log_{1/\beta} n} = n^{-\frac{\ln 1/\alpha}{\ln(1/\beta)}} = n^{-\rho}.$$

Thus the probability that such a $g_j$ exists is at least $1 - (1 - n^{-\rho})^r$. By setting $\tau = 2n^\rho$ we get property (1) holds with probability $\geq 1 - 1/e^2 > 4/5$. The claim follows.

**Claim 1.6** For $x \in [0, 1)$ and $t \geq 1$ such that $1 - tx > 0$ we have $\frac{\ln(1 - x)}{\ln(1 - tx)} \leq \frac{1}{t}$.

Proof: 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 $t \geq 1$. Therefore, $g$ is non-increasing in the region in which we are interested, and so $g(x) \leq 0$ for all values in this interval.

**Lemma 1.7** There exists a $\text{NNbr}_\infty(r, (1 + \varepsilon)r)$ which uses $O(dn + n^{1+1/(1+\varepsilon)})$ space and $O(n^{1/(1+\varepsilon)})$ hash probes for each query. The probability of success (i.e., there is a point $u \in P$ such that $d_H(u, q) \leq r$, and we return a point $v \in P$ such that $|uv| \leq (1 + \varepsilon)r$) is a constant.

Proof: By Lemma 1.4, we have a $(r, (1 + \varepsilon)r, \alpha, \beta)$-sensitive family of hash functions, where $\alpha = 1 - \frac{\varepsilon}{d}$ and $\beta = 1 - \frac{\tau(1+\varepsilon)}{d}$. As such

$$\rho = \frac{\ln 1/\alpha}{\ln 1/\beta} = \frac{\ln \alpha}{\ln \beta} = \frac{\ln \frac{d-r}{d}}{\ln \frac{d-(1+\varepsilon)r}{d}} = \frac{\ln(1 - \frac{\varepsilon}{d})}{\ln(1 - (1 + \varepsilon)\frac{r}{d})} \leq \frac{1}{1 + \varepsilon},$$

by Claim 1.6.

By building $O(\log n)$ structures of Lemma 1.7 we can do probability amplification and get a correct result with High probability.

**Theorem 1.8** Given a set $P$ of $n$ points on the hypercube $\mathbb{H}^d$, parameters $\varepsilon > 0$ and $r > 0$, one can build a $\text{NNbr}_\infty = \text{NNbr}_\infty(P, r, (1 + \varepsilon)r)$, such that given a query point $q$, one can decide if:

- $b(q, r) \cap P \neq \emptyset$, then $\text{NNbr}_\infty$ returns a point $u \in P$, such that $d_H(u, q) \leq (1 + \varepsilon)r$.
- $b(q, (1 + \varepsilon)r) \cap P = \emptyset$ then $\text{NNbr}_\infty$ returns that no point is in distance $\leq r$ from $q$.

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.
Proof: Note, that every point can be stored only once. Any other reference to it in the data-structure can be implemented with a pointer. Thus, the $O(dn)$ requirement on the space. The other term follows by repeating the space requirement of Lemma 1.7 $O(\log n)$ times.

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

**Theorem 1.9** Given a set $P$ of $n$ points on the hypercube $\mathbb{H}^d$, parameters $\varepsilon > 0$ and $r > 0$, one can build ANN data-structure using $O((d+n^{1/(1+\varepsilon)})\varepsilon^{-1} n \log^2 n)$ space, such that given a query point $q$, one can returns an ANN in $P$ (under the Hamming distance) in $O(dn^{1/(1+\varepsilon)} \log (\varepsilon^{-1} \log n))$ time. The result returned is correct with high probability.

## 2 LSH and ANN on Euclidean Space

### 2.1 Preliminaries

**Lemma 2.1** Let $X = (X_1, \ldots, X_d)$ be a vector of $d$ independent variables which have distribution $N(0,1)$, and let $v = (v_1, \ldots, v_d) \in \mathbb{R}^d$. We have that $v \cdot X = \sum_i v_i X_i$ is distributed as $\|v\| Z$, where $Z \sim N(0,1)$.

**Proof:** If $\|v\| = 1$ then this holds by the symmetry of the normal distribution. Indeed, let $e_1 = (1,0,\ldots,0)$. By the symmetry of the $d$-dimensional normal distribution, we have that $v \cdot X \sim e_1 \cdot X = X_1 \sim N(0,1)$.

Otherwise, $v \cdot X/\|v\| \sim N(0,1)$, and as such $v \cdot X \sim N(0,\|v\|^2)$, which is indeed the distribution of $\|v\| Z$.

A $d$-dimensional distribution that has the property of Lemma 2.1 is called a 2-stable distribution.

### 2.2 Locality Sensitive Hashing

Let $p,q$ 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 $N^d(0,1)$ (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 $p \in \mathbb{R}^d$, and consider the random hash function

$$h(p) = \left[ \frac{p \cdot \vec{v} + t}{r} \right].$$

If $p$ and $q$ are in distance $\eta$ from each other, and when we project to $\vec{v}$, the distance between the projection is $t$, then the probability that they get the same hash value is $1 - t/r$, since this is the probability that the random sliding will not separate them. As such, we have that the probability of collusion is

$$\alpha(\eta) = \Pr[h(p) = h(q)] = \int_{t=0}^{r} \Pr[|p \cdot \vec{v} - q \cdot \vec{v}| = t] \left(1 - \frac{t}{r}\right) dt.$$

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

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

In fact, if are going to use this hashing scheme for constructing locality sensitive hashing, like in Lemma 1.5 then we care about the ratio

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

The following is verified using numerical computations on a computer, Lemma 2.2 ([DNIM04]) One can choose $r$, such that $\rho(1 + \varepsilon) \leq \frac{1}{1 + \varepsilon}$.

Lemma 2.2 implies that the hash functions defined by Eq. (1) 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 $\alpha'$ and $\beta'$. As such, we can use this hashing family to construct $\text{NNbr}_\approx$ for the set $P$ of points in $\mathbb{R}^d$. Following the same argumentation of Theorem 1.8 we have the following.

**Theorem 2.3** Given a set $P$ of $n$ points in $\mathbb{R}^d$, parameters $\varepsilon > 0$ and $r > 0$, one can build a $\text{NNbr}_\approx = \text{NNbr}_\approx(P, r, (1 + \varepsilon)r)$, such that given a query point $q$, one can decide if:

- $b(q, r) \cap P \neq \emptyset$, then $\text{NNbr}_\approx$ returns a point $u \in P$, such that $d_H(u, q) \leq (1 + \varepsilon)r$.
- $b(q, (1 + \varepsilon)r) \cap P = \emptyset$ then $\text{NNbr}_\approx$ returns that no point is in distance $\leq r$ from $q$.

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 $O(dn + n^{1/(1 + \varepsilon)} \log n)$. The result returned is correct with high probability.

### 2.3 ANN in High Dimensional Euclidean Space

Unlike the 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 follows from what we had seen in previous lectures).

**Theorem 2.4** Given a set $P$ of $n$ points in $\mathbb{R}^d$, then one can construct data-structures $D$ that answers $(1 + \varepsilon)$-ANN queries, by performing $O(\log(n/\varepsilon))$ $\text{NNbr}_\approx$ queries. The total number of points stored at $\text{NNbr}_\approx$ data-structures of $D$ is $O(ne^{-1} \log(n/\varepsilon))$.

Constructing the data-structure of Theorem 2.4 requires building a low quality HST. Unfortunately, the previous construction seen for HST are exponential in the dimension, or take quadratic time. We next present a faster scheme.

#### 2.3.1 Low quality HST in high dimensional Euclidean space

**Lemma 2.5** Let $P$ be a set of $n$ in $\mathbb{R}^d$. One can compute a $nd$-HST of $P$ in $O(nd \log^2 n)$ time (note, that the constant hidden by the $O$ notation does not depend on $d$).

**Proof:** Our construction is based on a recursive decomposition of the point-set. In each stage, we split the point-set into two subsets. We recursively compute a $nd$-HST for each point-set, and we merge the two trees into a single tree, by creating a new vertex, assigning it an appropriate value, and hung the two subtrees from this node. To carry this out, we try to separate the set into two subsets that are furthest away from each other.
Let $R = R(P)$ be the minimum axis parallel box containing $P$, and let $\nu = l(P) = \sum_{i=1}^{d} \| I_i(R) \|$, where $I_i(R)$ is the projection of $R$ to the $i$th dimension.

Clearly, one can find an axis parallel strip $H$ of width $\nu/((n-1)d)$, such that there is at least one point of $P$ on each of its sides, and there is no points of $P$ inside $H$. Indeed, to find this strip, project the point-set into the $i$th dimension, and find the longest interval between two consecutive points. Repeat this process for $i = 1, \ldots, d$, and use the longest interval encountered. Clearly, the strip $H$ corresponding to this interval is of width $\nu/((n-1)d)$. On the other hand, $\text{diam}(P) \leq \nu$.

Now recursively continue the construction of two trees $T^+, T^-$, for $P^+, P^-$, respectively, where $P^+, P^-$ is the splitting of $P$ into two sets by $H$. We hung $T^+$ and $T^-$ on the root node $v$, and set $\Delta_0 = \nu$. We claim that the resulting tree $T$ is a nd-HST. To this end, observe that $\text{diam}(P) \leq \Delta_0$, and for a point $p \in P^-$ and a point $q \in P^+$, we have $\|pq\| \geq \nu/((n-1)d)$, which implies the claim.

To construct this efficiently, we use an efficient search trees to store the points according to their order in each coordinate. Let $D_1, \ldots, D_d$ be those trees, where $D_i$ store the points of $P$ in ascending order according to the $i$th axis, for $i = 1, \ldots, d$. We modify them, such that for every node $v \in D_i$, we know what is the largest empty interval along the $i$th axis for the points $P_v$ (i.e., the points stored in the subtree of $v$ in $D_i$). Thus, finding the largest strip to split along, can be done in $O(d \log n)$ time. Now, we need to split the $d$ trees into two families of $d$ trees. Assume we split according to the first axis. We can split $D_1$ in $O(\log n)$ time using the splitting operation provided by the search tree (Treaps for example can do this split in $O(\log n)$ time). Let assume that this split $P$ into two sets $L$ and $R$, where $|L| < |R|$.

We still need to split the other $d-1$ search trees. This is going to be done by deleting all the points of $L$ from those trees, and building $d-1$ new search trees for $L$. This takes $O(|L|d \log n)$ time. We charge this work to the points of $L$.

Since in every split, only the points in the smaller portion of the split get charged, it follows that every point can be charged at most $O(\log n)$ time during this construction algorithm. Thus, the overall construction time is $O(dn \log^2 n)$ time.

### 2.3.2 The overall result

Plugging Theorem 2.3 into Theorem 2.4 we have:

**Theorem 2.6** Given a set $P$ of $n$ points in $\mathbb{R}^d$, parameters $\varepsilon > 0$ and $r > 0$, one can build ANN data-structure using

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

space, such that given a query point $q$, one can returns an $(1+\varepsilon)$-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 $O(dn^{1+1/(1+\varepsilon)} e^{-2 \log^3 (n/\varepsilon)})$.

**Proof:** We compute the low quality HST using Lemma 2.5. This takes $O(nd \log^2 n)$ time. Using this HST, we can construct the data-structure $\mathcal{D}$ of Theorem 2.4, where we do not compute the NNbr$_\approx$ data-structures. We next traverse the tree $\mathcal{D}$, and construct the NNbr$_\approx$ data-structures using Theorem 2.3.

We only need to prove the bound on the space. Observe, that we need to store each point only once, since other place can refer to the point by a pointer. Thus, this is the $O(nd)$ space requirement. The other term comes from plugging the bound of Theorem 2.4 into the bound of Theorem 2.3.
3 Bibliographical notes

Section 1 follows the exposition of Indyk and Motwani [IM98]. The fact that one can perform approximate nearest neighbor in high dimensions in time and space polynomial in the dimension is quite surprising. One can reduce the approximate near-neighbor in euclidean space to the same question on the hypercube (we show the details below). This implies together with the reduction from ANN to approximate near-neighbor (seen in previous lectures) that one can answer ANN in high dimensional euclidean space with similar performance. Kushilevitz, Ostrovsky and Rabani [KOR00] offered an alternative data-structure with somewhat inferior performance.

The value of the results showed in this write-up depend to large extent on the reader perspective. Indeed, for 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 low dimension perspective, where $\varepsilon$ is assumed to be small, this result is slightly sublinear. On the other hand, if one is willing to pick $\varepsilon$ 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) $n^{1/11}$.

The idea of doing locality sensitive hashing directly on the Euclidean space, as done in Section 2 is not shocking after seeing the Johnson-Lindenstrauss lemma. It is taken from a recent paper of Datar et al. [DNIM04]. 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. Another open problem is to improve the performance of the LSH scheme.

The low-quality high-dimensional HST construction of Lemma 2.5 is taken from [Har01]. The running time of this lemma can be further improved to $O(dn \log n)$ by more careful and involved implementation, see [CK95] for details.

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 $Q$ the resulting set of points in $\mathbb{R}^{k'}$. We want to solve $\text{NNbr}_r$ 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 $\text{NNbr}_r$ inside this grid cell (which has bounded diameter as a function of $r$), since with small probability that the result would be correct. We amplify the probability by repeating this polylogarithmic number of times.

Thus, we can assume that $P$ is contained inside a cube of side length $\leq k'mr$, and 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 $(010, 101)$ assuming the number of bits for each coordinate is 3.) 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.

This relationship indicates that the ANN on the hypercube is “equivalent” to the ANN in Euclidean space. In particular, making progress on the ANN on the hypercube would probably lead to similar progress on the Euclidean ANN problem.

We had only scratched the surface of proximity problems in high dimensions. The interested reader is referred to the survey by Indyk [Ind04] for more information.
References


