7

High-Dimensional Computational Geometry

Alexandr Andoni

CONTENTS

7.1 Introduction ....................................................... 105
7.2 Dimension Reduction ............................................. 106
7.3 Space Partitions .................................................. 107
  7.3.1 From LSH to NNS ........................................ 108
  7.3.2 LSH Families ............................................... 109
    7.3.2.1 Bit Sampling (For Hamming Distance) ............ 110
    7.3.2.2 Grid Partitioning (For $\ell_1$ Distance) .......... 110
    7.3.2.3 Random Projection (For Euclidean and $\ell_p$ Distances) .... 110
    7.3.2.4 Ball Carving (For Euclidean Distance) .......... 110
    7.3.2.5 Min-Hash and Sim-Hash ............................ 111
  7.3.3 Optimality of LSH ......................................... 111
  7.3.4 Data-Dependent Space Partitions ....................... 111
  7.3.5 The $\ell_\infty$ Norm .................................... 112
7.4 Embeddings ..................................................... 112
  7.4.1 Embeddings of Norms ................................... 113
  7.4.2 Embeddings of Specialized Metrics .................... 113
7.5 Sketching ...................................................... 115
  7.5.1 Sketch Constructions ................................... 116
7.6 Small Dimension ............................................... 117
7.7 Conclusion ..................................................... 117
References ........................................................ 118

7.1 Introduction

Consider the following common similarity search scenario. We are given a new image, and we need to determine whether it is similar to some image in the existing database, preprocessed beforehand. How do we capture the notion of similar? The standard approach is to use a suitable distance metric. For example, we can represent a $20 \times 20$-pixel image as a $400$-dimensional vector, one coordinate per pixel. Then we can measure the (dis)similarity using the Euclidean distance in $\mathbb{R}^{400}$.

This problem, called nearest neighbor search (NNS), admits a straightforward solution—scanning the entire database—but it is prohibitive for databases of modern sizes. Instead, it is imperative to obtain a solution with run-time that is sublinear in the database size. The bad news is that this problem suffers from the curse of dimensionality: all solutions degrade exponentially fast with the dimension. In fact, the problem is not believed to have such
sublinear solutions. The good news is that vast improvements are possible in both theory and practice, once we settle for approximate answers. In practice, approximate answers prove to be sufficient even if one looks for exact answers: for many natural datasets there are only few entries that are close to being an optimal answer (false positives), and thus we can efficiently filter them.

The above is just an example of how high-dimensional geometry emerges in big data questions. In general, the geometry is not necessarily Euclidean, and indeed many interesting questions appear for distances such as Hamming, earthmover distance, and edit distance (Levenshtein distance), to name just a few. This geometric perspective confers a number of benefits. First, it brings in our geometric intuition to argue about sets of objects as sets of points. Second, geometric perspective allows us to bring in some powerful tools from the areas such as metric embeddings and functional analysis.

In this chapter, we survey the techniques of the high-dimensional computational geometry through the prism of the NNS problem. From a broader perspective, NNS is just one example of the questions arising when dealing with massive datasets. Yet these questions often benefit from common solution concepts that already surface when considering the NNS problem. In fact, oftentimes, NNS itself serves as a building block for the solutions of the other questions. Hence, we focus on NNS to have a consistent storyline.

We will start by looking at the aforementioned setting: high-dimensional Euclidean space, denoted by $\mathbb{R}^d$ where $d$ is the dimension. This is perhaps the most basic high-dimensional setting. In general, we expect to see a trade-off between the complexity of the geometric structure and the algorithmic efficiency. We now give the exact definition of the approximate nearest neighbor problem.

**Definition 7.1 c-approximate nearest neighbor** Given a set $P$ of $n$ points in a $d$-dimensional space $\mathbb{R}^d$, and approximation factor $c > 1$, construct a data structure such that, given any query point $q$, the data structure returns some point $p$ such that $d(q,p) \leq c \cdot \min_{p^* \in P} d(q,p^*)$.

### 7.2 Dimension Reduction

If high dimensionality presents a problem, why not reduce it? Indeed, the high-dimensional geometry 101 technique is to map the points into a space of smaller dimension $k$, while preserving the distances. The most classic result is the Johnson–Lindenstrauss (JL) lemma [66]. It says that a projection onto a random $k$-dimensional subspace suffices, provided $k$ is large enough.

**Lemma 7.1 JL [66]** Let $A$ be the projection onto a random $k$-dimensional subspace of $\mathbb{R}^d$, scaled by $\sqrt{1/k}$. Then, for any fixed $\epsilon > 0$, and points $x, y \in \mathbb{R}^d$, we have that

$$\Pr_A \left[ \frac{\|Ax - Ay\|}{\|x - y\|} \in (1 - \epsilon, 1 + \epsilon) \right] \geq 1 - e^{-\Omega(\epsilon^2 k)}.$$

An important consequence is that we can argue about distances among a set of $n$ points. In particular, setting $k = \Theta(C\epsilon^{-2}\log n)$, we obtain the probability of preserving a fixed distance as high as $1 - 1/n^C$, for arbitrary large constant $C$. By union bound over the $n^2$ vectors, pairwise differences of the $n$ points are preserved with probability at least $1 - n^{-C+2}$. In fact, this is the most common usage of the lemma.

Another crucial aspect of the above map is that it is oblivious, that is, independent of the set of points. The alternative, a nonoblivious map, would have to first look at the $n$ points before constructing the actual map $A$. The advantage of an oblivious map is that we can apply it to a new point (sometimes called out-of-sample extension).
These properties are already enough to speed up NNS. Indeed, we can pick a projection $A$ as above, and then store points $A p$, where $p \in P$ for a given dataset $P$. On query $q$, we compute the projection $A q$, and then select the point $p$ that minimizes $||A q - A p||$. The correctness of the algorithm is guaranteed by the JL lemma (applied to the point-set $P \cup \{q\}$). The run-time of the query procedure is $O(nk) + O(dk)$ for $k = O(\epsilon^{-2} \log n)$. Note that the second term corresponds to computing the projection $A q$. The query time is an improvement over the naive $O(nd)$ bound for large $d$.

Following the original JL lemma, many natural extensions arose. Below are a few that have drawn a particular attention in the theoretical computer science literature:

- Can we improve the target dimension $k$?
  In general, the answer is no. Jayram and Woodruff [63] show any oblivious map must use dimension $k = \Omega(\epsilon^{-2} \log n)$. Even for nonoblivious maps, Alon [5] shows that $k = \Omega \left( \frac{(1/\epsilon^2)}{(\log 1/\epsilon)} \log n \right)$ is required. See also [77]. However, such dimension reduction may be possible for a point-set with additional structure. For example, if the point-set lies on a line (in $\mathbb{R}^d$), a much smaller $k$ will suffice. We give an example in Section 7.6.

- Are there dimension reductions for other geometric concepts besides distances?
  It turns out that the JL lemma may be used for other concepts, such as preserving $l$-flats (for $l \ll k$) or volumes of $l$-tuples of points [79]. In general, such applications require a higher target dimension $k$.

- Are there dimension reductions for other spaces besides Euclidean?
  In general, the answer is no. For example, even for the Manhattan $\ell_1$ space, it has been proved that dimension reduction for $n$ points, preserving distances up to factor $c > 1$, must use $n^{\Omega(1/c^2)}$ dimensions [25]. More generally, Johnson and Naor [68] show that any space satisfying JL-like dimension reduction must be close to an Euclidean space. However, there is an alternative concept for dimension reduction that can be applied to spaces such as $\ell_1$, as we will see in Section 7.5.

- Can we choose a more efficient map? For example, can we compute $A p$ faster than in $O(dk)$ time?
  Early results show alternative ways to choose $A$. For example, we can take $A$ to have each entry to be from a Gaussian distribution independent, identically distributed (i.i.d.) [42,60], or even random $\pm 1$ [2]. More surprisingly, we can pick $A$’s such that the computation of the map $A p$ takes time $\ll kd$. Such techniques, called fast JL transform, are often based on a (fast) Hadamard transform. In particular, the first such construction from [3] shows that we can set $A = P HD$, where $D$ is a (random) diagonal matrix, $H$ is the Hadamard transform, and $P$ is a (random) sparse projection matrix. Another approach is also to use a carefully designed sparse matrix $A$ [40]. See also [4,69,72,85].

### 7.3 Space Partitions

Back to the NNS problem, how can we obtain a sublinear query time? Note that even with the dimension reduction, the query time of the NNS algorithm from above is linear in $n$. The next important concept in high-dimensional geometry is space partitions, represented via a
hash function \( h : \mathbb{R}^d \rightarrow U \) for some discrete set \( U \). These lead to sublinear time solutions as we will see next.

What properties would we need from such hash functions? We would like that close pairs of points collide (are in the same part), whereas far pairs of points do not collide (are in different parts). Whereas such a strong guarantee is impossible (think of the points near one of the perimeters of a part), it becomes attainable in a randomized sense. This leads to the notion of locality-sensitive hash (LSH) functions, defined as follows.

**Definition 7.2 LSH** For a positive real \( r > 0 \) and \( c > 1 \), a distribution \( \mathcal{H} \) over hash functions \( g : \mathbb{R}^d \rightarrow U \) is called \((r, cr, P_1, P_2)\)-sensitive if for any two points \( p, q \in \mathbb{R}^d \), we have the following:

- If \( \|p - q\| \leq r \), then \( \Pr_{h \in \mathcal{H}}[h(q) = h(p)] \geq P_1 \).
- If \( \|p - q\| \geq cr \), then \( \Pr_{h \in \mathcal{H}}[h(q) = h(p)] \leq P_2 \).

In order for an LSH family to be useful, it has to satisfy \( P_1 > P_2 \).

Note that the definition is designed for a fixed scale of distances, that is, we care only about distances \( \leq r \) (close) versus distances \( \geq cr \) (far). We can use LSH to solve a threshold version of the nearest neighbor problem, called \( r \)-near neighbor problem, and defined as follows:

**Definition 7.3 \( c \)-approximate \( r \)-near neighbor** Fix some threshold \( r > 0 \) and approximation factor \( c > 1 \). Construct a data structure on a point-set \( P \) of \( n \) points in a \( d \)-dimensional space \( \mathbb{R}^d \), such that, given any query point \( q \), if \( P \) contains a some point \( p^* \) within distance \( r \) of \( q \), the data structure reports some point \( p \in P \) at distance at most \( cr \) from \( q \), with probability at least 0.9.

We can use an algorithm for the near neighbor problem to solve the nearest neighbor problem as well [53]. Hence we will focus on the former, the near neighbor problem, in the rest of the section.

### 7.3.1 From LSH to NNS

An LSH family \( \mathcal{H} \) can be used to design an efficient algorithm for approximate near neighbor search. The algorithm essentially is one (or more) hash table(s). To build one hash table, we choose a hash function \( h \in \mathcal{H} \) and hash each point \( p \in P \) into the bucket \( h(p) \). Because the total number of buckets may be large, we retain only the nonempty buckets by resorting to (standard) hashing\(^*\) of the values \( h(p) \).

To process a query \( q \), we look up the bucket \( h(q) \) and report the first \( cr \)-near neighbor found there. Note that, if there exists some \( r \)-near neighbor \( p \), then with probability at least \( P_1 \), it is present in the bucket \( h(q) \), and hence some point is reported. In general, the bucket contains other points as well.

Thus, one hash table achieves a probability of success of \( P_1 \), while using \( O(n) \) space. We can increase the probability of success to, say, 0.9 by using \( L = O(1/P_1) \) independent hash tables, each with its own hash function chosen i.i.d. from \( \mathcal{H} \).

Note that, for one hash table, the query run-time depends on the number of far points that collide with the query \( q \). There are at most \( P_2 n \) such points in expectation. The overall query time is composed of two terms: (1) \( L = O(1/P_1) \) computations of a hash function

\(^*\)See [35] for more details on hashing.
$h \in \mathcal{H}$ and (2) $O(P_2 n \cdot L)$ distance computations (in expectation). Hence, it is enough to obtain $P_2 = 1/n$. We will usually think of a hash function computation as taking $O(d)$ time, comparable to the time for distance computation.

Thus, the main LSH performance question becomes: what is the highest $P_1$ achievable for $P_2 \leq 1/n$? Usually, we obtain such LSH family by amplifying a base family. In particular, given a base $(r, cr, P_1, P_2)$-sensitive family $\mathcal{H}$, we can construct a new family $\mathcal{G}$ of hash functions $g = (h_1, \ldots, h_k)$, where $h_i \in \mathcal{H}$ independently. The new family $\mathcal{G}$ is $(r, cr, P_1, P_2)$ sensitive, where $P_1 = p_1^k$ and $P_2 = p_2^k$. For $k = \lceil \log_{1/p_2} n \rceil$, we obtain $P_2 = p_2^k \leq 1/n$ as desired. At the same time, $P_1 = p_1^k \geq p_2^\log_{1/p_2} n + 1 = n^{-\rho} \cdot p_1$, where $\rho = (\log 1/p_1)/((\log 1/p_2)$.

Note that $\rho$ is invariant to the amplification process, that is, this LSH parameter of the amplified family $\mathcal{G}$ is equal to the one of the base family $\mathcal{H}$.

The parameter $\rho = (\log 1/P_1)/((\log 1/P_2)$ is thus the main measure of quality of an LSH family, or, equivalently a (randomized) space partitioning. We summarize the above in the following theorem, proved in the work of Indyk and Motwani [60] who introduced the LSH scheme. We also sketch the entire algorithm in Figure 7.1.

Theorem 7.1 Fix $d, n \geq 1$, threshold $r \geq 0$, and approximation $c \geq 1$. Suppose there exist some $(r, cr, P_1, P_2)$-sensitive LSH family. Then there exists an algorithm for the $c$-approximate $r$-near neighbor problem using $O(n^{1+\rho}/P_1 + nd)$ storage and preprocessing time. The query time is composed of $O(n^\rho/P_1 \cdot \log_{1/P_2} n)$ computations of a hash function and $O(n^\rho/P_1)$ distance computations.

Because the query time includes the time to evaluate a hash function from the LSH family, it depends on the actual choice of the LSH family. Furthermore, the stated space bound does not include the space to store the description of the hash functions (which is typically smaller than the other terms).

7.3.2 LSH Families

To complete the algorithm, we need to construct some actual LSH families. There is a number of various LSH families proposed over years. As mentioned above, the main quality parameter is $\rho$ (as a function of $c$). The second important parameter is the run-time to evaluate a hash function. We present a few important families below, and refer to [12] for a more comprehensive survey of the LSH families.

**Preprocessing:**

1. Choose $L$ functions $g_i$, $i = 1, \ldots, L$, by setting $g_i = (h_{i,1}, h_{i,2}, \ldots, h_{i,k})$, where $h_{i,1}, \ldots, h_{i,k}$ are chosen at random from the LSH family $\mathcal{H}$.

2. Construct $L$ hash tables, where, for each $j = 1, \ldots, L$, the $j$th hash table contains the dataset points hashed using the function $g_j$.

**Query algorithm for a query point $q$:**

For each $i = 1, 2, \ldots, L$:

1. Retrieve the points from the bucket $g_i(q)$ in the $i$th hash table.

2. For each of the retrieved point, compute the distance from $q$ to it. If the point is a correct answer, report it and stop.

**FIGURE 7.1**

Preprocessing and query algorithms based on LSH. For an $(r, cr, P_1, P_2)$-sensitive family $\mathcal{H}$, we set $k = \lceil \log_{1/P_2} n \rceil$ and $L = O(n^\rho/P_1)$, where $\rho = (\log 1/p_1)/(\log 1/p_2)$. 


7.3.2.1 Bit Sampling (For Hamming Distance)

This is the first (and simplest) LSH family that was introduced in [60]. In this case, the family $\mathcal{H}$ contains all projections of the input point on one of the coordinates, that is, all functions $h_i$ from $\{0,1\}^d$ to $\{0,1\}$ such that $h_i(p) = p_i$, for $i = 1 \ldots d$. A random hash function $h \in \mathcal{H}$ returns a random coordinate of $p$ (note that different applications of $h$ return the same coordinate of the argument). The amplified hash function $g$ will return $k$ random coordinates of the argument.

To see that the family $\mathcal{H}$ is locality sensitive with nontrivial parameters, observe that the probability $Pr_h[h(p) = h(q)]$ is equal to the fraction of coordinates on which $p$ and $q$ agree. Therefore, $P_1 = 1 - r/d$, whereas $P_2 = 1 - cr/d$. One then obtains $\rho = 1/c$ in this case.

7.3.2.2 Grid Partitioning (For $\ell_1$ Distance)

First pick a random vector $w \in \mathbb{R}^d$ by choosing each coordinate from a Gamma distribution with expectation $r$, namely the distribution is $\frac{x}{r} e^{-x/r}$ [10,88]. Then pick a vector $s \in \mathbb{R}^d$ where each coordinate $s_i$ is random from $[0,w_i]$. The hash function is $h(x) = ([x_1 - s_1]/w_1), \ldots, ([x_d - s_d]/w_d))$. The space partition is obtained by shifting the space randomly and then imposing a regular grid with sidelengths $w_1, \ldots, w_d$. This family yields $\rho = 1/c$ as well. (One can also use a grid with equal side lengths, achieving a slightly worse $\rho$.)

7.3.2.3 Random Projection (For Euclidean and $\ell_p$ Distances)

Pick a random projection of $\mathbb{R}^d$ onto a one-dimensional line and chop the line into segments of length $w$, shifted by a random value $s \in [0,w)$ [43]. Formally, $h(x) = ([r \cdot x + s]/w]$, where $r \in \mathbb{R}^d$ is the projection vector with each coordinate $r_i$ drawn from the Gaussian distribution. The exponent $\rho$ drops strictly below $1/c$ for some (carefully chosen) finite value of $w$. This LSH family is very useful in practice (see, e.g., [7]).

A generalization of this approach to $\ell_p$ norms for any $p \in [0,2)$ is possible as well; this is done by picking the vector $r$ from the p-stable distribution [43].

7.3.2.4 Ball Carving (For Euclidean Distance)

This LSH family achieves the best possible $\rho$ for the Euclidean space [11,12]. It can be viewed as a higher dimensional version of the above method.

We construct a hash function $h$ as follows: project the input point $x$ onto a random $t$-dimensional subspace $S$. We would now like to partition $\mathbb{R}^t$ nicely. It would be natural to partition $\mathbb{R}^t$ using a grid, but it does not achieve a better $\rho$. This is because this process roughly corresponds to hashing using concatenation of several one-dimensional functions (as above). Because the LSH algorithms perform such concatenation (amplification) anyway, grid partitioning does not result in any improvement.

Instead we partition $\mathbb{R}^t$ into balls, instead of cubes as before. Because this is impossible for $t \geq 2$, we approximate such a partition by carving out balls (see also [29]). Namely, we create a sequence of balls $B_1, B_2, \ldots$, each of radius $w$, with centers chosen independently at random. Each ball $B_i$ then defines a cell, containing points $B_i \setminus \bigcup_{j \leq i} B_j$.

To complete the construction, we need to take care of the following issue: locating a cell containing a given point could require enumeration of all balls, which would take an unbounded amount of time. Instead, one can simulate the above procedure by replacing each ball by a grid of balls. It is not difficult to observe that a finite (albeit exponential in $t$) number of such grids suffices to cover all points in $\mathbb{R}^t$.

The parameter $t$ is determined by a trade-off between the value of $\rho = 1/c^2 + O[(log t)/(\sqrt{t})]$ and the time to compute the hash function, which is $t^{O(t)}$. 
7.3.2.5 Min-Hash and Sim-Hash

Finally, we present two classic constructions of hash functions that were designed for a similarity measure, rather than a distance.

Min-hash is a family sensitive with respect to the Jaccard coefficient [26,27]. Jaccard coefficient between two sets $A, B \subset U$ is defined as $s(A, B) = \frac{|A \cap B|}{|A \cup B|}$. Here, we pick a random permutation $\pi$ on the ground universe $U$ and define $h_\pi(A) = \min\{\pi(a) \mid a \in A\}$. It is not hard to prove that the probability of collision is $\Pr[h_\pi(A) = h_\pi(B)] = s(A, B)$.

Sim-hash is a family sensitive with respect to the angle between two vectors $x, y \in \mathbb{R}^d$, namely, $\theta(x, y) = \arccos \left[ \frac{x \cdot y}{\|x\| \|y\|} \right]$ [28]. Similar to [49], we pick a random unit-length vector $u \in \mathbb{R}^d$ and define $h_u(x) = \text{sign}(u \cdot x)$. The hash function can also be viewed as partitioning the space into two half-spaces by a randomly chosen hyperplane. The probability of collision of two vectors is $\Pr_u[h_u(x) = h_u(y)] = 1 - \theta(x, y)/\pi$.

7.3.3 Optimality of LSH

Earlier, we saw that we can obtain exponent $\rho = 1/c$ for the Hamming and Manhattan spaces, and $\rho = 1/c^2$ for the Euclidean space. Can we achieve even better LSH families?

It turns out that these values are the best possible LSH exponents $\rho$. Definition 7.2 is purely geometric, and hence, one can use functional analytic tools to prove lower bounds on the exponent $\rho$. This is precisely what [82,86] proved. Below we reproduce the tight bound from [86].

**Theorem 7.2** [86] Fix dimension $d \geq 1$, and approximation $c > 1$. Suppose $\mathcal{H}$ is an $(r, cr, P_1, P_2)$-sensitive family for the Hamming space, where $P_2 > 2^{-\Omega(d)}$. Then $\rho \geq 1/c - o(1)$.

Similarly, any LSH family for the Euclidean space has $\rho \geq 1/c^2 - o(1)$.

Note that we assume that $P_2$ is sufficiently large. Such a condition seems necessary. Otherwise, we can construct a ball carving LSH family with $w = cr/2$, achieving $P_2 = 0$, and hence $\rho = 0$. However, we then obtain a prohibitively low $P_1 < 2^{-\Omega(d)}$, implying a query run-time of $2^{\Omega(d)}$. The restriction on $P_2$ precludes such infeasible families.

7.3.4 Data-Dependent Space Partitions

The above LSH lower bound suggests that the story line of space partitions should be by and large complete: we know space partitions with the best possible quality $\rho$. Yet, it turns out that we can obtain space partitions with better values of $\rho$, by stepping outside the LSH framework (Definition 7.2).

In particular, we can get qualitatively better partitions if the partitions are data dependent [15,19]. Data-dependent hashing is a family $\mathcal{H}$ that chooses a random hash function $h$ dependent on the given dataset $P$. As before, such a family yields a near neighbor data structure with the query time dominated by $n^p$ hash function evaluations, for the similarly defined exponent $\rho$. As before, we want a hash function which we can efficiently evaluate on a new (query) point $q$. Otherwise, we could take $h$ to be the Voronoi diagram: $h(q)$ returns the identity of the closest point $p \in D$. Such a function has the ideal $P_1 = 1$ and $P_2 = 0$, but it takes $\Omega(n)$ time to evaluate and is thus useless.

In [19], the data-dependent hashing for the Euclidean space achieves

$$\rho = \frac{1}{2^{\rho - 1}} + o(1)$$

for which hash function evaluation takes $2^{o(d)}$ time. This improves over the best possible LSH exponent $\rho = 1/c^2 + o(1)$ (ball carving LSH from above). Similarly, for the Hamming
space, the new approach yields \( \rho = [1/(2c - 1)] + o(1) \), improving over the best possible LSH exponent \( \rho = 1/c \) (bit sampling LSH).

It is important to note that this improvement is for worst-case datasets. To put this into perspective, if one were to assume that the dataset has some special structure, it would be more natural to expect speedups with data-dependent hashing (e.g., by adapting to the special structure, perhaps implicitly). However, in the worst-case setting, there is no assumed structure to adapt to.

Nonetheless one can wonder whether we can obtain even better hashing assuming some additional structure in the dataset. Indeed, in practice, there is a number of NNS algorithms that design hash codes that depend on the dataset, but most have no guarantees: correctness or performance. A formidable open question is to try to understand the disparity of the data-independent methods (which are often theoretically optimal) and the data-dependent methods (often with better performance); see [39, page 77]. Some further work along these lines was done in [1,41,93].

Finally, let us observe that data-dependent space partitions come with a price tag: they are not dynamic. In particular, oftentimes, we need to insert and delete points from the dataset. Standard (data-independent) hashing handles this effortlessly: we can easily add and remove points from the hash table. However, if the hash function depends on the dataset, it may become inadequate once we remove/insert a large part of the dataset. In particular, it may loose its LSH properties. Hence, further work is necessary to make such data structures dynamic.

### 7.3.5 The \( \ell_\infty \) Norm

The \( \ell_\infty \) norm admits a particularly intriguing NNS solution, also based on space partitions. Although it has no LSH, the algorithm of [55] achieves an approximation of \( O(\log \log d) \) for \( d \)-dimensional \( \ell_\infty \), with polynomial space. The algorithm can be seen as a data-dependent space partition, organized as a tree. In contrast to the previous hashing approaches, this algorithm has a couple intriguing differences. First, it contains just one tree, constructed deterministically. Second, some dataset points are stored in a few buckets: essentially, a point \( p \) is stored in each bucket \( B \) for which there exists a (query) point \( q \) which is at distance \( \leq \) from \( p \) and hashes to \( B \). The query algorithm just hashes the query point once to the corresponding bucket and retrieves a near neighbor from that bucket.

### 7.4 Embeddings

So far we have discussed about techniques for the \( \ell_j \) distances, such as the Manhattan or Euclidean space. How about other, more complex distances? A natural approach is to reduce the new distances to the ones we know how to handle, such as the above distances. Embeddings provide such a generic reduction, mapping one distance space into another. This purely geometric (functional analytic) concept is useful for many computational problems.

**Definition 7.4** Consider some metric \( (M,d_M) \) and a host metric \( (H,d_H) \). The map \( \phi : M \to H \) is called an embedding of \( M \) into \( H \) if there exist constants \( \alpha, D > 0 \) such that for any \( x,y \in M \) we have that:

\[
d_M(x,y) \leq \alpha \cdot d_H[\phi(x),\phi(y)] \leq D \cdot d_M(x,y).
\]

The factor \( D \) is called the distortion (approximation). A randomized embedding is a randomized map \( \phi \) where the above inequality holds with probability \( 1 - \delta \) for every fixed \( x,y \), and some \( \delta > 0 \).
To see why this is useful, suppose we want to solve NNS under some new metric $M$ (we give some examples below). If we manage to map $M$ into, say, $\ell_1$ with distortion $D$, then we can use the NNS under $\ell_1$ to solve NNS under $M$ with approximation, say, $2D$. We need to ensure also that the map is efficiently computable, but this is often a secondary concern. The main question is: what is the best possible distortion $D$ we can achieve?

The JL lemma 7.1 is a classic example of a (randomized) embedding: it maps a high-dimensional Euclidean $\mathbb{R}^d$ space into the lower dimensional $\mathbb{R}^k$, the host.

We present a few other classic and illustrative embeddings in Sections 7.4.1 and 7.4.2. A more complete (if only slightly outdated) survey can be also found in [56,59]. Also there is an influential list of open problems in the area in [81]. We also remark that there are other important types of embedding. A very important one is one where we want to embed a finite set of points $P \subset M$. This is usually not as useful for the NNS application because we do not have an out-of-sample extension: that is, we do not know how to compute the embedding of the query point $q \notin P$, which is given after the embedding on $P$ is constructed. (See also the discussion in Section 7.7.) Another type is nearest neighbor preserving embedding, where the map merely preserves the distance to the (approximate) nearest neighbor within a dataset. Examples of such mapping were given in [54,58].

### 7.4.1 Embeddings of Norms

Some classic embeddings are those concerning embedding of an $\ell_p$ norm into some other $\ell_q$. We list a few relevant ones as follows.

- $\ell_2$ embeds into $\ell_1$ isometrically [44,48] (i.e., with distortion $D = 1$). In fact, a $(1 + \epsilon)$ distortion map follows from a variant of the JL embedding from Section 7.2: $\phi(x) = \frac{1}{m}Ax$, where $A$ is an $m \times n$ matrix with entries drawn from the Gaussian distribution. This is the standard way to reduce problems under $\ell_2$ to problems under $\ell_1$ (e.g., in [60] for NNS).

- Hamming space $\{0, 1\}^d$ embeds into the square of $\ell_2$ isometrically [78]. The latter is the space $\mathbb{R}^d$ where we compute the distances as $d(x, y) = \sum_{i=1}^{d}(x_i - y_i)^2$. Although this is not a metric, it is useful nonetheless: for example, we can use the Euclidean NNS for it. This is a standard way to reduce the Hamming space to $\ell_2$ and is the one explaining the tight connection between $p$’s for the Hamming and $\ell_2$ spaces from Section 7.3.

- $\ell_1^d$ isometrically embeds into $\ell_\infty^m$ for $m = 2^d$ (see also [80]). The map $\phi(x)$ has a coordinate for each bit vector $b \in \{0, 1\}^d$ and is defined as $\phi(x)_b = \sum_i(-1)^b_i x_i$. This may be used, for example, for computing the diameter of an $n$-point set in $\ell_1^d$. Although the naïve algorithm takes $\Omega(n^2)$ time to compute the diameter of a point-set, this time is only linear for a point-set in $\ell_\infty^m$. Specifically, the diameter of a point-set $P$ is just $\max_{i \in [m]}(\max_{p \in P} p_i - \min_{p \in P} p_i)$, a linear-time computation. Thus, this embedding allows us to solve the $\ell_1^d$ diameter problem in $O(n2^d)$ time.

- Any $n$-point metric $X$ embeds into $\ell_\infty$ of dimension $d = n$. The map $\phi$ has a coordinate for each $x \in Y$ and is defined as $\phi(y)_x = d_X(x, y)$. Thus, $\ell_\infty$ is universal in that it contains any other metric, given a sufficiently high dimension.

### 7.4.2 Embeddings of Specialized Metrics

Many applications often give rise to metrics that are more complex than the norms from above. We look to embed them into simpler metrics, for which we have known algorithms.
The latter metrics include $\ell_1, \ell_2,$ and $\ell_\infty$ (for which we have, say, NNS algorithms by the results from Section 7.3). It turns out that $\ell_1$ is a particularly useful host for the following reasons: First, because $\ell_2$ embeds into $\ell_1$, the latter can accommodate more metrics. Second, many important metrics in fact already have some $\ell_1$ structure in them.

A powerful aspect is that we can also prove impossibility results, termed non embeddability, like we did for LSH. A classic example is the proof that any embedding of the Hamming cube $\{0,1\}^d$ into $\ell_2$ (of any dimension) requires $\Omega(\sqrt{d})$ distortion [45].

Below we define a few notorious examples of metrics amenable to the embeddings approach. Table 7.1 presents the positive and negative results on embedding these metrics into $\ell_1$.

- **Edit distance** (Levenstein distance) is defined for two strings of length $d$ as the minimum number of insertions/deletions/substitutions to transform one string into the other. Edit distance is a basic notion for measuring the degree of misalignment in various structures, and thus plays a central role in several fields, such as bioinformatics or natural language processing sequences. Edit distance on nonrepetitive strings is called Ulam distance.

It is not hard to note that both the edit and Ulam distances contain a copy of Hamming cube of dimension $d^\Omega(1)$ (i.e., we can embed the latter into the former with constant distortion). Hence, it is natural to look for embeddings into $\ell_1$. Note that asking how well does edit distance embed into the Hamming metric is philosophically equivalent to the question: how much complexity do the insertions/deletions add to the Hamming distance, where we are allowed substitutions only.

- **Earthmover distance** (EMD) is defined on sets of points in some metric $(X,d_X)$. For two sets $A,B \subset X$ of the same size, the earthmover distance is the minimum cost matching between $A$ and $B$ [where the cost of matching $a$ to $b$ is $d_X(a,b)$]. Most common examples of the base metric $X$ are the plane $[d]^2$ and the high-dimensional cube $\{0,1\}^d$. Both have applications in the image vision literature; see, for example, [90] and [50,51].

Again, because we can embed $\ell_1$ into EMD (even over a line), it is natural to target $\ell_1$ as the host norm for EMD.

- **Hausdorff distance** is also defined over subsets of a base metric $(X,d_X)$. For two sets $A,B \subset X$, the Hausdorff distance is the minimum between $\max_{a \in A} \min_{b \in B} d_X(a,b)$ and its symmetric version $\max_{b \in B} \min_{a \in A} d_X(a,b)$. This distance is useful in image vision for comparing geometric shapes of points.

### Table 7.1

Distortion for embedding of various metrics into $\ell_1$

<table>
<thead>
<tr>
<th>Metric</th>
<th>Upper Bound</th>
<th>Lower Bound</th>
</tr>
</thead>
<tbody>
<tr>
<td>Edit distance on ${0,1}^d$</td>
<td>$2^{O(\sqrt{\log d \log \log d})}$</td>
<td>$\Omega(\log d)$  [8,16,71,74]</td>
</tr>
<tr>
<td>Ulam distance</td>
<td>$O(\log d)$</td>
<td>$\Omega(\log d / \log \log d)$ [16]</td>
</tr>
<tr>
<td>Block edit distance, edit distance with moves</td>
<td>$O(\log d \log^* d)$ [36–38,83]</td>
<td>—</td>
</tr>
<tr>
<td>EMD over $[d]^2$</td>
<td>$O(\log d)$</td>
<td>$\Omega(\sqrt{\log d})$ [84]</td>
</tr>
<tr>
<td>EMD over ${0,1}^t$ (for sets of size $s$)</td>
<td>$O(\log s \log t)$ [13]</td>
<td>$\Omega(\log s)$ [71]</td>
</tr>
</tbody>
</table>
The Hausdorff metric contains a copy of $\ell_\infty$, and hence, it is a natural host space. A few such embeddings into $\ell_\infty$, for different types of datasets $X$ is shown in [47].

Are there even better host spaces? $\ell_\infty$ is another natural target, especially that it contains any other metric. However, as a host, $\ell_\infty$ often requires prohibitively high dimension: even embedding the Hamming cube $\{0,1\}^d$ into $\ell_\infty$ requires dimension exponential in $d$ (see arguments from [67, Chapter 1, Section 8]). Hence, new candidates are needed.

It turns out that the mixture of norms is a qualitatively better host. Let us define what we mean by mixture first. The norm $\ell_p^d(\ell_q^k)$ is the space of $d \times k$ matrices, where we compute the norm by first taking $\ell_q$ norm of each row, and then take the $\ell_p$ norm of the resulting $d$-dimensional vector. The strength of such mixed norms has been shown in [14], who showed that the aforementioned Ulam metric overlength-dimension vector. The strength of such mixed norms has been shown in [14], who showed that the aforementioned Ulam metric overlength-

### 7.5 Sketching

A generalization of the embedding is the notion of sketching. Sketching can be thought of as a very weak embedding into a computational space, where the host distance is an arbitrary computation (e.g., not restricted to be a metric). The main parameter is now the size of the sketch, which can be thought of as host dimension. Sketches address the most basic, decision version of the distance estimation problem, termed the distance threshold estimation problem (DTEP) [91]. The goal of DTEP is to distinguish close pairs versus far pairs with some probability (similar to the Definition 7.3 of near neighbor problem).

**Definition 7.5** Fix some metric $(M,d_M)$, approximation $D \geq 1$, threshold $r > 0$, failure probability $\delta \in [0,1/3]$, and size $s \in \mathbb{N}$. The map $\phi : M \to \{0,1\}^s$ is called a sketch of $M$ if there exists a referee algorithm $\mathcal{R} : \{0,1\}^s \times \{0,1\}^s \to \{0,1\}$, such that for any $x,y \in M$, the algorithm $\mathcal{R}$ can distinguish between $d_M(x,y) \leq r$ (close pair) versus $d_M(x,y) > Dr$ (far pair), with probability $1-\delta$.

Sketching is useful in many applications, including the NNS problem. In particular, we can use a sketch of size $s$ (for, say, $\delta = 0.1$) to construct an NNS data structure with space $n^{O(s)}$. The query time is $O(\log n)$ times the time to evaluate $\phi$ on the query point. Indeed, from the basic sketch, we can also construct an amplified sketch, which has $O(s \log n)$ size and failure probability at most $1 - 1/n^2$: keep $k = O(\log n)$ independent copies of the basic sketch, with the referee algorithm taking the majority vote of those for the $k$ sketches. For a given query point $q$, this amplified sketch, termed $\phi^k(q)$, is sufficient to determine the approximate near neighbor in an $n$-point dataset $P$ with probability at least $1 - 1/n$: run the referee algorithm on $\langle \phi^k(q), \phi^k(p) \rangle$ for each $p \in P$. Because this procedure uses only the sketch $\phi^k(q)$, of size $ks$, we can construct an index for each possible $ks$-bit input, storing the solution. Overall, we obtain a data structure with $2^{ks} = n^{O(s)}$ space.

Sketches have many applications beyond NNS, including in data-streaming algorithms and compressed sensing, in part by providing an alternative notion of dimension reduction (as we will see later). More recently, it has found uses to speed up algorithms, for example, in numerical linear algebra [95].
Some of these applications use another important variant of sketches, where $M$ is a norm, and we define $\phi : M \to \mathbb{R}^s$ to be a linear map. In this case, we think of $s$ as the number of measurements, as in the compressed sensing literature. Such a linear sketch has the advantage of being easy to update: because $\phi(x + a) = \phi(x) + \phi(a)$, where $x$ is the current vector, and $a$ is an update vector. As such, it has many applications in streaming.

### 7.5.1 Sketch Constructions

The first and most useful constructions are for the Euclidean and Hamming spaces. Again, the JL dimension reduction for $\ell_2$ already gives a sketch. In particular, dimension reduction is a linear sketch into $s = O\left(\frac{1}{\epsilon^2} \log \frac{1}{\delta}\right)$ dimensions for $D = 1 + \epsilon$ approximation (see also [6]).

Most interestingly, $\ell_1$ admits a similar linear sketch, to give us a weaker notion of dimension reduction for $\ell_1$. The sketch is $\phi(p) = \frac{1}{k}Ap$, in which $A$ is a $k \times d$ matrix in which each entry is drawn i.i.d. from the Cauchy distribution [58]. The referee algorithm, on inputs $\phi(p)$, $\phi(q)$, then computes the median value of absolute values of the coordinates of $\phi(p) - \phi(q)$. Note that this is different from aforementioned dimension reduction, which can be thought of a sketch with the referee algorithm that computes $\|\phi(p) - \phi(q)\|$. The median operation is what makes this host space nonmetric.

LSH is also a sketch, with a special property that the referee algorithm just checks for the equality of the two arguments. Although Definition 7.2 of LSH does not immediately imply a size bound, one can easily notice that an LSH function leads to a sketch of size dependent on inputs $\phi(p)$, $\phi(q)$, then computes the median operation is what makes this host space nonmetric.

For the Hamming space, Indyk and Motwani [60] and Kushilevitz et al. [76] show a sketch achieving a size of $s = O\left(\frac{1}{\epsilon^2} \log \frac{1}{\delta}\right)$ bits, for approximation $D = 1 + \epsilon$. The sketch is a modification of the bit sampling LSH from Section 7.3.2. For fixed threshold $r \leq d/2$, the coordinate $i \in [s]$ of the map $\phi(p)$ is simply defined as the exclusive-or of $k = \left\lceil \frac{d}{r} \right\rceil$ randomly chosen bits from $p$. The referee algorithm for inputs $\phi(p)$, $\phi(q)$ just computes the Hamming distance between $\phi(p)$ and $\phi(q)$: the output is close pair iff this fraction is less than a certain threshold.

This sketch plays a very central role for a few reasons. First, it also implies a similar sketch for $\ell_2$ of constant size. Second, all sketches of constant size for other metrics $M$ are essentially obtained via a two-step process: first embed $M$ into the Hamming space, and then apply the above sketch. Third, recent results suggest that, at least for norms, this approach is the only one in town for obtaining constant size sketches [18].

Nonetheless, there exist other methods to obtain sketches with a nontrivial size, namely, sublinear in the dimension of the input. For example, for the Ulam metric, one can obtain a sketch of polylog $d$ size and constant approximation [14]. For EMD metric, the best sketch achieves size $d^\epsilon$ for $O(1/\epsilon)$ approximation [9]. There are also methods for obtaining sketches for arbitrary mixed norms [17,65].

Finally, an important aspect of the sketch definition is that we can prove impossibility results for them. Sketching lower bounds usually follow via communication complexity arguments [75] for the following communication game. Two players, called Alice and Bob, each have an input point $x$ and $y$, respectively, and they need to solve the DTEP problem. For this, Alice and Bob each send a message of length $s$ to the referee, who is to decide on the problem. It is easy to see that lower bounds on communication complexity of this game imply sketching lower bounds.
For example, one can prove that the Hamming sketch from above is optimal [62,64,94]. Many other lower bounds have been proven. Interestingly, in light of the sketch for $\ell_1$, lower bounds on sketching complexity often imply nonembeddability statements as well (see, e.g., [16]).

### 7.6 Small Dimension

Yet another approach to high-dimensional questions is to assume that the dataset has an additional structure, for example, is *intrinsically low dimensional*. Such assumptions are motivated by the fact that, in many datasets, the data are really explained by a few relevant parameters, and the other parameters are just derivates of those. The simplest such example is when a $d$-dimensional dataset lies inside a low-dimensional subspace $S \subset \mathbb{R}^d$, of dimension $k \ll d$. In such cases, we would like to obtain solutions with a performance as if the dimension is effectively $k$. If $k$ is small, we may be able to afford solutions that have exponential dependence on $k$ (e.g., $(1/\epsilon)^{O(k)}$ for $1+\epsilon$ approximation).

Although the above subspace assumption may be too naive, there are other, more realistic definitions. Particular attention has been drawn to the notion of *doubling dimension*, defined for a dataset $P$ as follows: Let $\lambda$ be the smallest integer such that for any $p \in P$ and radius $r$, the set of points that are within $r$ of $p$ can be covered by at most $\lambda$ balls of radius $r/2$. The doubling dimension is then $\log \lambda$. It is easy to check that $d$-dimensional $\ell_2$ and $\ell_1$ spaces have doubling dimension $O(d)$. The notion, introduced in [32,52], has been inspired by the notion of Assouad constant [22]. Some NNS algorithms designed with this notion in mind include [24,73] and others. For example, Indyk and Naor [54] showed that, given a point-set with a doubling dimension $k$, one can use JL lemma to project it into dimension $m = O(k/\epsilon^2 \cdot \log 1/\epsilon)$ only, sufficient for preserving the approximate nearest neighbor.

Other notions of *intrinsic dimension* include Karger–Ruhl dimension [70], smooth manifolds [23,34], and others [1,31,33,41,46,54].

### 7.7 Conclusion

As mentioned in the beginning of the chapter, the focus is on techniques for high-dimensional geometry, in particular as applied to the NNS problem.

Many of the same techniques apply in other contexts. For example, some other computational problems in the realm of high-dimensional geometry include the closest pair problem, or variants of clustering. A number of such problems admit efficient solutions based on efficient NNS problem, or the aforementioned techniques directly (see, [57] for examples). For instance, consider the closest pair problem: given a set of $n$ points, we are to find the closest pair inside it. We can solve the problem using NNS: Just construct an NNS data structure, and then query each point to find its closest match. In fact, until recently, this has been the fastest way to solve the (approximate) closest pair problem. Valiant [92] showed how to obtain faster solutions for a random dataset, exploiting faster matrix multiplication methods.
Still, there are other important high-dimensional geometry topics left uncovered here. The most prominent such topic is that of embedding finite metrics into $\ell_1$ or other simple host spaces. Introduced to theoretical computer science in the seminal paper of [78], such embeddings have had a big impact on the design of approximation algorithms such as the sparsest cut problem [20,21,89]. We refer to the survey [59] for more details.

References


