39 NEAREST NEIGHBORS IN HIGH-DIMENSIONAL SPACES

Piotr Indyk

INTRODUCTION

In this chapter we consider the following problem: given a set P of points in a high-dimensional space, construct a data structure which given any *query* point q finds the point in P closest to q. This problem, called ***nearest neighbor search***[1], is of significant importance to several areas of computer science, including pattern recognition, searching in multimedial data, vector compression [GG91], computational statistics [DW82], and data mining. Many of these applications involve data sets which are very large (e.g., a database containing Web documents could contain over one billion documents). Moreover, the dimensionality of the points is usually large as well (e.g., in the order of a few hundred). Therefore, it is crucial to design algorithms which scale well with the database size as well as with the dimension.

The nearest-neighbor problem is an example of a large class of *proximity problems*, which, roughly speaking, are problems whose definitions involve the notion of distance between the input points. Apart from nearest-neighbor search, the class contains problems like closest pair, diameter, minimum spanning tree and variants of clustering problems.

Many of these problems were among the first investigated in the field of computational geometry. As a result of this research effort, many efficient solutions have been discovered for the case when the points lie in a space of *constant* dimension. For example, if the points lie in the plane, the nearest-neighbor problem can be solved with $O(\log n)$ time per query, using only $O(n)$ storage [SH75, LT80]. Similar results can be obtained for other problems as well. Unfortunately, as the dimension grows, the algorithms become less and less efficient. More specifically, their space or time requirements grow *exponentially* in the dimension. In particular, the nearest-neighbor problem has a solution with $O(d^{O(1)} \log n)$ query time, but using roughly $n^{O(d)}$ space [Cla88, Mei93]. Alternatively, if one insists on linear or near-linear storage, the best known running time bound for *random* input is of the form $\min(2^{O(d)}, dn)$, which is essentially linear in n even for moderate d. Worse still, the exponential dependence of space and/or time on the dimension (called the "curse of dimensionality") has been observed in applied settings as well. Specifically, it is known that many popular data structures (using linear or near-linear storage), exhibit query time linear in n when the dimension exceeds certain threshold (usually 10-20, depending on the number of points). See e.g., [WSB98] for more information.

The lack of success in removing the exponential dependence on the dimension led many researchers to conjecture that no efficient solutions exists for these problems when the dimension is sufficiently large (e.g., see [MP69]). At the same time,

[1] Many other names occur in literature, including *best match*, *post office problem* and *nearest neighbor*.

it raised the question: Is it possible to remove the exponential dependence on d, if we allow the answers to be *approximate*. The notion of approximation is best explained for nearest-neighbor search: instead of reporting a point p closest to q, the algorithm is allowed to report *any* point within distance $(1+\epsilon)$ times the distance from q to p. Similar definitions can be naturally applied to other problems. Note that this approach is similar to designing efficient approximation algorithms for NP-hard problems.

During recent years, several researchers have shown that indeed in many cases approximation enables reduction of the dependence on dimension from exponential to polynomial. In this chapter we will survey these results. In addition, we will discuss the issue of *proving* that the curse of dimensionality is inevitable if one insists on exact answers, and survey the known results in this direction.

Although this chapter is devoted almost entirely to approximation algorithms with running times polynomial in the dimension, the notion of approximate nearest neighbor was first formulated in the context of algorithms with exponential query times. Chapter 51, section 7 of this handbook covers those results in more detail.

Before proceeding further, we mention that our treatment of the topic is primarly theoretical. For experimental evaluations and applications of the algorithms described in this chapter, see e.g., [GIM99, CDF$^+$00, HGI00, Shi00, Buh01, BT01, Ya01, Buh02, OMST02, GSM03]. In addition, we focus on algorithms operating in main memory. For external memory algorithms, see e.g., recent proceedings of *SIGMOD* and *VLDB* conferences.

39.1 APPROXIMATE NEAR NEIGHBOR

Almost all algorithms for proximity problems in high-dimensional spaces proceed by reducing the problem to the problem of finding an *approximate near neighbors*, which is the decision version of the approximate near<u>est</u> neighbor problem. Thus, we start from describing the results for the former problem.

For the definitions of metric spaces and normed spaces, see Chapter 8.

GLOSSARY

Approximate Near Neighbor, or (r,c)***-NN:*** Given a set P on n points in a metric space $M=(X,D)$, design a data structure that supports the following operation: For any query $q \in X$, if there exists $p \in P$ such that $D(p,q) \le r$, find a point $p' \in P$ such that $D(q,p') \le cr$

Dynamic problems: Problems which involve designing a data structure for a set of points (e.g., approximate near neighbor) and support insertions and deletions of points. We distinguish dynamic problems from their *static* versions by adding the word “Dynamic” (or letter “D”) in front of their names (or acronyms). E.g., the dynamic version of the approximate near-neighbor problem is denoted by (r,c)-DNN.

Hamming metric: A metric (Σ^d, D) where Σ is a set of *symbols*, and for any $p,q \in \Sigma^d$, $D(p,q)$ is equal to the number of $i \in \{1\dots d\}$ such that $p_i \neq q_i$.

TABLE 39.1.1 Approximate Near Neighbors.

#	APPROX.	QUERY TIME	SPACE	UPDATE TIME
1a	Source: [KOR00] (cf. [HIM03]); Randomness: Monte Carlo			
	$1+\epsilon$	$d\log n/\min(\epsilon^2,1)$	$n^{O(1/\epsilon^2+\log(1+\epsilon)/(1+\epsilon))}$	$n^{O(1/\epsilon^2+\log(1+\epsilon)/(1+\epsilon))}$
1b	Source: [Ind01]; Randomness: Monte Carlo			
	$1+\epsilon$	$n^{O(\frac{1+\log(1+\epsilon)}{1+\epsilon})}$	dn	$d\log^{O(1)} n$
2	Source: [HIM03]; Randomness: Monte Carlo			
	$1+\epsilon$	$dn^{1/(1+\epsilon)}$	$n^{1+1/(1+\epsilon)}+dn$	$dn^{1/(1+\epsilon)}$
3	Source: [Ind00]; Randomness: Las Vegas			
	$1+\epsilon$	$(d\log n/\epsilon)^{O(1)}$	$n^{1/\epsilon^{O(1)}}$	static
4	Source: [Ind00]; Randomness: Deterministic			
	$3+\epsilon$	$(d\log n/\epsilon)^{O(1)}$	$n^{1/\epsilon^{O(1)}}$	static

RANDOM PROJECTION APPROACH

The first algorithms for (r,c)-NN in high dimensions were obtained by using the technique of random projections. This technique is applicable if the underlying metric D is induced by an l_p norm, for $p \in [0,2]$. We first focus on the case where all input and query points are binary vectors from $\{0,1\}^d$, and D is the Hamming distance (or equivalently, the metric is induced by the l_1 norm). The parameters of the algorithms discovered for this case are presented in the following table.

We mention that the idea of using random projections for high-dimensional approximate nearest neighbor first appeared in the paper by Kleinberg [Kle97]. Although his algorithms still suffered from the curse of dimensionality (i.e., used exponential storage or had $\Omega(n)$ query time), his ideas provided inspiration for designing improved algorithms.

Dimensionality reduction. The key technique used to obtain results (1a), (1b), (3), and (4) is *dimensionality reduction* i.e., a randomized procedure which reduces the dimension of Hamming space from d to $k = O(\log n/\epsilon^2)$, while preserving a certain range of distances between the input points and the query up to a factor of $1+\epsilon$ This notion has been introduced earlier in Chapter 8 in the context of Euclidean space. In case of Hamming space, [KOR00] showed the following.

THEOREM 39.1.1

For any given $r \in \{1\dots d\}$, $\epsilon \in (0,1]$ and $P \in (0,1)$, one can construct a distribution over mappings $A : \{0,1\}^d \to \{0,1\}^k$, $k = O(\log(1/P)/\epsilon^2)$, and a "scaling factor" S, so that for any $p,q \in \{0,1\}^d$, if $D(p,q) \in [r,10r]$, then $D(A(p),A(q)) = S\cdot D(p,q)(1\pm\epsilon)$ with probability at least $1-P$.

The factor 10 can be replaced by any constant. As in the case of Euclidean norm, the mapping A is linear. However, unlike in the Euclidean case where the mapping was defined over the set of reals $\mathbb{R}$, the mapping A is defined over $GF(2)$ (i.e., over the set $\{0,1\}$ with addition and multiplication taken modulo 2). The $k\times n$ matrix A is obtained by choosing each entry of A independently at random from the set $\{0,1\}$. The probability that an entry is equal to 1 is roughly r/d.

A different method of generating mapping A was proposed in [Ind00]. The mapping is nonlinear, but somewhat easier to analyze (and derandomize). It is based on "Locality-Sensitive Hashing," described later in this section.

Algorithm (1a) is an immediate application of Theorem 39.1.1. Specifically, it allows us to reduce the $(r, c+\epsilon)$-NN problem in d-dimensional space to (r,c)-NN problem in k-dimensional space. Since the *exact* nearest-neighbor problem in k-dimensional space can be solved by storing the answers to all 2^k queries q, the bound follows. Algorithm (1b) is follows by using a variation of this approach. Algorithms (3) and (4) are obtained by using a deterministic version of Theorem 39.1.1 [Ind00].

We note that one can apply the same approach to solve the near-neighbor problem in the Euclidean space. In particular, it is fairly easy to solve the $(r, 1+\epsilon)$-NN problem in l_2^d using $n(1/\epsilon)^{O(d)}$ space [HIM03]. Applying the Johnson-Lindenstrauss lemma leads to an algorithm with storage bound similar (although slightly worse) to the bound of algorithm (1a) [HIM03].

Locality-Sensitive Hashing. As may have been noticed, the storage bounds for algorithms (1a), (3) and (4) are quite high. On the other hand, the query time of algorithm (1b) is low only for fairly large values of ϵ [Ind01]. In this context, algorithm (2) provides an attractive tradeoff, since even for small values of ϵ (e.g., $\epsilon = 1.0$) its running time is fairly low (e.g., $d\sqrt{n}$). The algorithm is based on the concept of *Locality-Sensitive Hashing*, or *LSH* [HIM03] (see also [KWZ95, Bro98]). A family of hash functions $h : \{0,1\}^d \to U$ is called (r_1, r_2, P_1, P_2)-sensitive (for $r_1 < r_2$ and $P_1 > P_2$) if for any $q, p \in \{0,1\}^d$

- If $D(p,q) \leq r_1$ then $\Pr[h(q) = h(p)] \geq P_1$,
- If $D(p,q) > r_2$ then $\Pr[h(q) = h(p)] \leq P_2$

where $\Pr[\cdot]$ is defined over the random choice of h. We note that the notion of locality-sensitive hashing can be defined for any metric space D in a natural way (see [Cha02] for sufficient and necessary conditions for existence of LSH for D). However, for Hamming space, LSH families are particularly easy: it is sufficient to take all functions h_i, $i = 1 \dots d$, such that $h_i(p) = p_i$, $p \in \{0,1\}^d$. Because $\Pr[h(p) = h(q)] = 1 - D(p,q)/d$, it is immediate that this family is sensitive.

If we are provided with an LSH family with a "large" gap between P_1 and P_2, the $(r_2/r_1, r_1)$-NN problem can solved in the following way. During preprocessing, all input points p are hashed to the bucket $h(p)$. In order to answer the query q, the algorithm retrieves the points in the bucket $h(q)$ and checks if any one of them is close to q. If the gap between P_1 and P_2 is sufficiently large, this approach can be shown to result in sublinear query time. Unfortunately, the P_1/P_2 gap guaranteed by the above LSH family is not large enough. However, the gap can be amplified by concatenating several independently chosen hash functions $h_1 \dots h_l$ (i.e., hashing the points using functions h' such that $h'(p) = (h_1(p), \dots, h_l(p))$. Details can be found in [HIM03].

A somewhat similar hashing-based algorithm (for the closest-pair problem) was earlier proposed in [KWZ95], and also in [Bro98]. Due to different problem formulation and analysis, comparing their performance with the guarantees of the LSH approach seems difficult.

We also mention that the above algorithm can be modified to solve the approximate *nearest*-neighbor problem, within the same time bounds (i.e., without incurring any additional overhead, as is the case for the reductions presented in the next section). Details can be found in [Cha02].

Extensions to l_p norms. The approximate near-neighbor problem under l_p norms, for $p \in [1, 2]$, can reduced to the same problem in Hamming space. The reduction is particularly easy for the l_1^d norm. If we assume that all points of interest p have coordinates in the range $\{1 \dots M\}$, then if we define $U(p) = (U(p_1), \dots, U(p_d))$ where $U(x)$ is a string of x ones followed by $M - x$ zeros, we get $\|p - q\|_1 = D(U(p), U(q))$. In general, M could be quite large, but can be reduced to $d^{O(1)}$ in the context of approximate near neighbor [HIM03]. Thus we can reduce (r, c)-NN under l_1 to (r, c)-NN in Hamming space.

In order to obtain algorithms for l_p norm where $p \in (1, 2]$, we use the fact that l_p^d can be embedded into $l_1^{O(d)}$ with bounded distortion (see Chapter 8). Alternatively, for $p = 2$, one can solve the problem directly in Euclidean space [HIM03], as described earlier.

DIVIDE-AND-CONQUER APPROACH

The dimensionality reduction and locality-sensitive hashing techniques have natural limitations. In particular, they cannot be used for solving the near-neighbor problem under the l_∞ norm. Fortunately, this norm has other nice properties which makes designing approximate nearest-neighbor data structures possible.

The only algorithm known for solving (r, c)-NN under the l_∞^d norm [Ind98] has the following parameters, for any $\rho > 0$:

- Approximation factor: $c = O(4\lfloor \log_{1+\rho} \log 4d \rfloor)$; if $\rho = \log d$ then $c = 3$
- Space: $dn^{1+\rho}$
- Query time: $O(d \log n)$ for the static, or $(d + \log n)^{O(1)}$ for the dynamic case
- Update time: $d^{O(1)} n^\rho$ (described in [Ind01])

The basic idea of the algorithm is to use a divide and conquer approach. In particular, consider hyperplanes H consisting of all points with one (say the ith) coordinate equal to the same value. The algorithm tries to find a hyperplane H having the property that the set of points $P_L \subset P$ which are on the left side of H and within distance $\geq r$ from H, is not "much smaller" than the set P_M of points within distance r from H. Moreover, a similar condition has to be satisfied for an analogously defined set P_R of points on the right side of H. If such H exists, we divide P into $P_1 = P_L \cup P_M$ and set $P_2 = P \setminus P_L$ and build the data structure recursively on P_1 and P_2. It is easy to see that while processing a query q, it suffices to recurse on either P_1 or P_2, depending on the side of H the query q lies on. Also, one can prove that the increase in storage caused by duplicating P_M is moderate. On the other hand, if H does not exist, one can prove that a large subset C of P has $O(r)$ diameter. In such a case we can pick any point from C as its representative, and apply the algorithm recursively on $P \setminus C$.

GLOSSARY

Product metrics: An f-product of metrics $M_1, \ldots M_k$ with distance functions $D_1, \ldots D_k$ is a metric over $M_1 \times \ldots \times M_k$ with distance function D such that $D((p_1, \ldots, p_k), (q_1, \ldots, q_k)) = f(D_1(p_1, q_1), \ldots, D_k(p_k, q_k))$.

Although the l_∞ data structure seems to rely on the geometry of the l_∞ norm, it turns out that it can be used in a much more general setting. In particular, assume that we are given k metrics $M_1 \ldots M_k$ such that for each metric M_i we have a data structure for (a variant of) (r, c)-NN in metric M_i, with $Q(n)$ query time and $S(n)$ space. In this setting, it is possible to construct a data structure solving $(r, O(c \log \log n))$-NN in the max-product metric M of $M_1, \ldots, M_k$ (i.e., an f-product with f computing the maximum of its arguments) [Ind02]. The data structure for M achieves query time roughly $O(Q(n) \log n + k \log n)$ and space $O(kS(n)n^{1+\delta})$, for any constant $\delta > 0$. The data structure could be viewed as an abstract version of the data structure for the l_∞ norm (note that the l_∞^d norm is a max-product of l_p^1 norms). For the particular case of the l_∞^d norm, it is easy to verify that the result of [Ind02] provides a $O(\log \log n)$-approximate algorithm using space polynomial in n. At the same time, the algorithm of [Ind98] has $O(\log \log d)$-approximation guarantee when using the same amount of space. Interestingly, the former data structure gives an approximation bound comparable to the latter one, while being applicable in much more general setting.

EXTENSIONS VIA EMBEDDINGS

Most of the algorithms described so far work only for l_p norms. However, they can be used for other metric spaces M, by using low-distortion embeddings of M into l_p norms. See Chapter 8 for more information.

AVERAGE-CASE ALGORITHMS

The approximate algorithms described so far are designed to work for any (i.e., worst-case) input. However, researchers have also investigated *exact* algorithms for the NN problem, which achieve fast query times for *average* input. Below we describe three such results.

Near-neighbor in Hamming space. Consider the point set P where each point is chosen independently and uniformly at random from the set $\{0, 1\}^d$. In addition, assume that the nearest neighbor p of the query point q is located within distance r from q. In this setting, it was shown in [GPY94] that q can be retrieved in $O(dn^{r/d})$ time, using a data structure which requires $O(dn^{1+r/d})$ space. The basic idea of their approach is similar to the locality-sensitive hashing approach of [HIM03]; however, the set of projected coordinates is chosen in a deterministic fashion, to optimize certain parameters.

Nearest neighbor in the l_2^d norm Consider the "continuous" version of the Hamming distance scenario, such that each point in P is chosen independently and uniformly at random from the set $[-1, 1]^d$. In addition, assume that the nearest neighbor p of the query point q is located within distance $r = 2b\sqrt{d}$ for some (small) constant b. The value of b is always small enough so that r does not exceed the average distance between two random points.

Under these assumptions, it was shown in [Yia00] that the k-d-tree data structure (augmented in a proper way) enjoys $O(dn^{\rho})$ query time, where ρ is a function of b. The analysis in the paper is idealized (i.e., uses approximations not shown to be rigorous).

We note that if d is large enough, then the distance between the query point and any data point is sharply concentrated around its mean (say $2t\sqrt{d}$). In this case, if $r = 2bt\sqrt{d}$, $b \in (0,1)$, then by using locality-sensitive hashing with approximation factor $1/b$, one obtains an algorithm with query time dn^{b}. It appears that this bound outperforms the computational bound given in [Yia00]. However, the k-d-tree data structure used in [Yia00] uses only linear space, unlike the LSH-based approach.

Nearest neighbor in the l_∞^d norm. Consider a point set generated as before, but with the query point generated from the same distribution as the input points (and independently from the latter). In this setting, it was shown [AHL01, HL02] that there is a nearest-neighbor data structure using $O(dn)$ space, with query time $O(n \log d)$. Note that a naive algorithm would suffer from query time of $O(nd)$. The algorithm uses a clever prunning approach to quickly eliminate points that *cannot* be nearest neighbors of the query point.

39.2 REDUCTIONS TO APPROXIMATE NEAR NEIGHBOR

GLOSSARY

We define the following problems, for a given set of points P in a metric space $M = (X, D)$:

Approximate Closest Pair, or c-CP: Find a pair of points $p', q' \in P$ such that $D(p', q') \le c \min_{p,q \in P, p \neq q} D(p, q)$

Approximate Close Pair, or (r, c)-CP: If there exists $p, q \in P, p \neq q$, such that $D(p, q) \le r$, find a pair $p', q' \in P, p' \neq q'$, such that $D(q', p') \le cr$.

Approximate Chromatic Closest Pair, or c-CCP: Assume that each point $p \in P$ is labeled with a color $c(p)$. The goal is to find a pair of points p, q such that $c(p) \neq c(q)$ and $D(p, q)$ is approximately minimal (as in the definition of c-CP).

Approximate Bichromatic Closest Pair, or c-BCP: As above, but $c(p)$ assumes only two values.

Approximate Chromatic/Bichromatic Close Pair, or (r, c)-CCP/(r, c)-BCP: Decision versions of c-CCP or c-BCP (as in the definition of (r, c)-CP).

Approximate Furthest Pair, or Diameter, or c-FP: Find $p, q \in P$ such that $D(p, q) \ge \max_{p', q' \in P} D(p', q')/c$. The decision problem, called ***Approximate Far Pair***, or (r, c)-FP, is defined in the natural way.

Approximate Furthest Neighbor, or c-FN: A maximization version of the Approximate Near Neighbor. The decision problem, called Approximate Far Neighbor or (r, c)-FN, is defined in a natural way.

Approximate Minimum Spanning Tree, or c-MST: Find a tree T spanning all points in P whose weight $w(T) = \sum_{(p,q)\in T} D(p,q)$ is at most c times larger than the weight of any tree spanning P.

Approximate Bottleneck Matching, or c-BM: Assuming $|P|$ is even, find a set of $|P|/2$ non-incident edges E joining points in P (i.e., a matching), such that the following function is minimized (up to factor of c)

$$\max_{\{p,q\}\in E} D(p,q)$$

Approximate Facility Location, or c-FL: Find a set $F \subset P$ such that the following function is minimized (up to factor of c), given the cost function $c : P \to \mathbb{R}^+$

$$\sum_{p\in F} c(p) + \sum_{p\in P} \min_{f\in F} D(p,f)$$

In general, we could have two sets: P_c of *cities* and P_f of *facilities*; in this case we require that $F \subset P_f$ and we are only interested in the cost of P_c.

Spread (of a point set): The ratio between the diameter of the set to the distance between its closest pair of points.

In this section we show that the problems defined above can be efficiently reduced to the approximate near-neighbor problem discussed in the previous section.

First, we observe that any problem from the above list, say $c(1+\delta)$-P for some $\delta > 0$, can be easily reduced to its decision version (say (r,c)-P), if we assume that the spread of $P\cup\{q\}$ is always bounded by some value, say Δ. For simplicity, assume that the minimum distance between the points in P is 1. The reduction proceeds by building (or maintaining) $O(\log_{1+\delta}\Delta)$ data structures for (r,c)-P, where r takes values $(1+\delta)^i/2$ for $i = 0, 1 \ldots$. It is not difficult to see that a query to $c(1+\delta)$-P can be answered by $O(\log\log_{1+\delta}\Delta)$ calls to these structures for (r,c)-P, via binary search.

In general, the spread of P could be unbounded. However, in many cases it is easy to ensure that $\Delta \le n^{O(1)}$. This can be accomplished, for example, by "discretizing" the input to c-MST or c-FL. In those cases, the above reduction is very efficient.

Reductions from other problems are specified in the following table. The bounds for the time and space used by the algorithm in the "To" column are denoted by $T(n)$ and $S(n)$, respectively.

We mention that a few other reductions have been given in [KOR00, BOR99b]. For the problems discussed in this section, they are less efficient than the reductions in the above table. Additionally, [BOR99b] reduces the problems of computing *approximate agglomerative clustering* and *sparse partitions* to $O(n\log^{O(1)} n)$ calls to a dynamic approximate nearest neighbor data structure. See [BOR99b] for the definitions and algorithms.

Also, we mention that a reduction from $(1+\epsilon)$-approximate furthest neighbor to $(1+\epsilon)$-approximate nearest neighbor (for the static case and under the l_2 norm) has been given in [GIV01]. However, a direct (and dynamic) algorithm for the approximate furthest neighbor in l_2^d, achieving a better query and update times of $dn^{1/(1+\epsilon)^2}$, has been recently given in [Ind03]. The former paper also presents an algorithm for computing a $\sqrt{2}+\epsilon$-approximate diameter (for any $\epsilon > 0$) of a given pointset in $dn\log^{O(1)} n$ time.

TABLE 39.2.1 Reductions to Approximate Near Neighbors.

#	FROM	TO	TIME	SPACE
1	Source: [HIM03].			
	$c(1+\delta)$-NN	(r,c)-NN	$T(n)\log^{O(1)} n$	$S(n)\log^{O(1)} n$
2	Source: [Epp95]; amortized time.			
	c-DBCP	c-DNN	$T(n)\log^{O(1)} n$	$S(n)\log^{O(1)} n$
	(r,c)-DBCP	(r,c)-DNN	$T(n)\log^{O(1)} n$	$S(n)\log^{O(1)} n$
3	Source: [HIM03]; via Kruskal alg.			
	$c(1+\delta)$-MST	(r,c)-DBCP	$nT(n)\log^{O(1)} n$	
4	Source: [GIV01, Ind01]; via Primal-Dual			
	$3c^3(1+\delta)$-FL	(r,c)-DBCP	$nT(n)\log^{O(1)} n$	
5	Source: [GIV01, Ind01].			
	$2c$-BM	c-DBCP	$nT(n)\log^{O(1)} n$	

We now describe briefly the main techniques used to achieve the above results.

Nearest neighbor. We start from the reduction of c-NN to (r,c)-NN. As we have seen already, the reduction is easy if the spread of P is small. Otherwise, it is shown that data set can be clustered into $n/2$ clusters, in such a way that:

- If the query point q is "close" to one of the clusters, it must be far away from a constant fraction of points in P; thus, we can ignore these points in the search for an approximate nearest neighbor.
- If the query point q is "far" from a cluster, then all points in the clusters are equally good candidates for the *approximate* nearest neighbor; thus we can replace the cluster by its representative point.

These ideas were originally introduced in [IM98], but their data structure was quite complex and inefficient. In [HP01] Har-Peled presented a considerably simpler data structure, achieving better time and space bounds.

Bichromatic closest pair. A very powerful reduction from various variants of c-DBCP to c-DNN was given in [Epp95]. His algorithm was originally designed for the case $c = 1$, but it can be verified to work also for general $c \geq 1$ [Epp99]. Moreover, as mentioned in the original paper, the reduction does not require the distance function $D()$ be a metric.

The basic idea of the algorithm is to try to maintain a graph that contains an edge connecting the two closest bichromatic points. A natural candidate for such a graph is the graph formed by connecting each point to its nearest neighbor. This, however, does not work, because a vertex in such a graph can have very high degree, leading to high update cost. Another option would be to maintain a single path, such that the ith vertex points to its nearest neighbor of the opposite color, chosen from points not yet included in the path. This graph has low degree, but its rigid structure makes it difficult to update it at each step. So the actual data structure is based on the path idea but allows its structure to degrade in a controlled way, and only rebuilds it when it gets too far degraded, so that the rebuilding work is spread over many updates. Then, however, one needs to keep track of the information from the degraded parts of the path, which can be done using a second shorter path, and so on. The constant factor reduction in the lengths of each successive path means the total number of paths is only logarithmic.

Minimum spanning tree. Many existing algorithms for computing MST (e.g., Kruskal's algorithm) can be expressed as a sequence of operations on a CCP data structure. For example, Kruskal's algorithm repetitively seeks the lightest edge whose endpoints belong to different components, and then merges the components. These operations can be easily expressed as operations on a CCP data structure, where each component has a different color. The contribution of [HIM03] was to show that in case of Kruskal's algorithm, using an *approximate* c-CCP data structure enables one to compute an *approximate* c-MST. Also, note that c-CCP can be implemented by $\log n$ c-BCP data structures [HIM03]. Other reductions from c-MST to c-BCP are given in [BOR99b, IST99].

Minimum bottleneck matching. The main observation behind this algorithm is that a matching is also a spanning forest with the property that any connected component has even cardinality (call it an *even* forest). At the same time, it is possible to convert *any* even forest to a matching, in a way that increases the length of the longest edge by at most a factor of 2. Thus, it suffices to find an even forest with minimum edge length. This can be done by including longer and longer edges to the graph, and stopping at the moment when all components have even cardinality. It is not difficult to implement this procedure as a sequence of c-CCP (or c-BCP) calls.

Other algorithms. The algorithm for the remaining problem (c-FL) is obtained by implementing the primal-dual approximation algorithm [JV99]. Intuitively, the algorithm proceeds by maintaining a set of balls of increasing radii. The latter process can be implemented by resorting to c-CCP. The approximation factor follow from the analysis of the original algorithm.

39.3 LOWER BOUNDS

In the previous sections we presented many algorithms solving approximate versions of proximity problems. The main motivation for designing approximation algorithms was the "curse of dimensionality" conjecture, i.e., the conjecture that finding exact solutions to those problems requires either superpolynomial (in d) query time, or superpolynomial (in n) space. In this section we state the conjecture more rigorously, and describe the progress toward proving it.

We start from the exact near-neighbor problem. For this problem, the curse of dimensionality can be formalized as follows. Assume that $d = n^{o(1)}$, but $d = \omega(\log n)$.

Conjecture 1 *Any data structure for* $(r, 1)$*-NN in Hamming space over* $\{0, 1\}^d$*, with* $d^{O(1)}$ *query time, must use* $n^{\omega(1)}$ *space.*

The conjecture as stated above is probably the weakest version of the "curse of dimensionality" phenomenon for the near-neighbor problem. It is plausible that other (stronger) versions of the conjecture could hold. In particular, at present, we do not know any data structure which simultanously achieves $o(dn)$ query time and $2^{o(d)}$ space for the above range of d. At the same time, achieving $O(dn)$ query time with space dn, or $O(d)$ query time with space 2^d is quite simple (via linear scan or using exhaustive storage).

Also note that if $d = O(\log n)$, achieving $2^{o(d)} = o(n)$ space is impossible via a simple incompressiblity argument.

Below we describe the work toward proving the conjecture. The first result addresses the complexity of a simpler problem, namely the *partial match* problem. This problem is of importance in databases and other areas and has been long investigated (e.g., see [Riv74]). Thus, the lower bounds for this problem are interesting in their own right.

GLOSSARY

Partial match: Given a set P of n vectors from $\{0,1\}^d$, design a data structure that supports the following operation: For any query $q \in \{0,1,*\}^d$, check if there exists $p \in P$ such that for all $i = 1 \ldots d$, if $q_i \neq *$ then $p_i = q_i$.

It is not difficult to see that any data structure solving $(r,1)$-NN in the Hamming metric $\{0,1\}^d$, can be used to solve the partial match problem using essentially the same space and query time. Thus, any lower bound for partial match problem implies a corresponding lower bound for the near neighbor problem. The best currently known lower bound for the partial match has been established in [BOR99a], following earlier work of [MNSW94]. Their lower bound holds in the *cell-probe* model, a very general model of computation, capturing e.g., the standard Random Access Machine model. Specifically, they show that any (possibly randomized) cell-probe algorithm for the partial match problem, in which the algorithm is allowed to retrieve at most $O(n^{1-\epsilon})$ bits from any memory cell in one step for $\epsilon > 0$, must either have $\Omega(\log d)$ query time or use $n^{\Omega(\log d)}$ memory cells.

For the exact near-neighbor problem, an exponentially larger bound was given in [BR00]. They showed that any (possibly randomized) cell-probe algorithm for $(r,1)$-NN in d-dimensional Hamming space, with cell size restriction as above, must either have query time $> t$, or use $2^{\Omega(d/t)}$ space. Thus, if $t = o(d/\log n)$, the space used must be superpolynomial in n.

The two aforementioned lower bounds are proved in a very general model, using the tools of *communication complexity*. As a result, they cannot yield lower bounds of $\omega(d/\log n)$ for the required query time, assuming $n^{\Theta(1)}$ space, as we now explain.

The communication complexity approach interprets the data structure as a communication channel between Alice (holding the query point q) and Bob (holding the database P). The goal of the communication (for Alice) is to learn the nearest neighbor of q. Since the data structure has polynomial size, each access to one of its memory cell is equivalent to Alice sending $O(\log n)$ bits of information to Bob. If we show that Alice needs to send at least b bits to Bob to solve the problem, we obtain $\Omega(b/\log n)$ lower bound for the query time. However, $b \leq d$, since Alice can always choose to transmit the whole input vector q. Thus, $\Omega(d/\log n)$ lower bound is the best result one can achieve using the communication complexity approach. A partial step toward removing this obstacle was made in [BV02], employing the *branching programs* model of computation. In particular, they focused on randomized algorithms that have very small (inversly polynomial in n) probability of error. They showed that any algorithm for the $(r,1)$-NN problem in the Hamming metric over $\{1\dots d^6\}^d$, has either $\Omega(d\log(d\log d/S))$ query time or uses $\Omega(S)$ space. This holds for $n=\Omega(d^6)$. Thus, if the query time is $o(d\log d)$, then the data structure must use $2^{d^{\Omega(1)}}$ space.

This completes the survey of lower bounds for the *exact* near-neighbor search. For the approximate version of this problem a cell-probe-based lower bound was shown in [CCGL99]. Specifically, the authors show that any deterministic data structure for the c-approximate *nearest* neighbor $\{0,1\}^d$ requires either $\Omega(\log\log d/\log\log\log d)$ query time, or use $n^{\omega(1)}$ space. They assume that a memory cell can contain up to $d^{O(1)}$ bits accessible in one step. Moreover, the approximation factor c can be as high as $2^{(\log d)^{1-\epsilon}}$ for any $\epsilon>0$.

For comparison, if randomization is allowed, then by using Theorem 39.1.1 combined with binary search one can get a data structure for the same problem (for any fixed $c>1$), with polynomial size and query time $O(\log\log_c d)$. Note that the assumption $c>1$ is crucial for those algorithms to achieve polynomial space bound.

REDUCTIONS

Despite the recent progress toward resolving the "curse of dimensionality" conjecture and the widespread belief in its validity, proving it seems currently beyond reach. Nevertheless, it is natural to assume the validity of the conjecture (or its variants), and see what conclusions can be derived from this assumption. Below we survey a few results of this type.

In order to describe the results, we need to state another conjecture.

Conjecture 2 *Let $d=n^{o(1)}$ but $d=\log^{\omega(1)} n$. Any data structure for the partial match problem with parameters d and n which provides $d^{O(1)}$ query time must use $2^{d^{\Omega(1)}}$ space.*

Note that, for the same ranges[2] of d, Conjecture 2 is analogous to Conjecture 1, but much stronger: it considers an easier problem, and states stronger bounds. However, since the partial match problem was extensively investigated on its own, and no algorithm with bounds remotely resembling the above have been discovered (cf. [CIP02] for a survey), Conjecture 2 is believed to be true.

Assuming Conjecture 2, it is possible to show lower bounds for some of the approximate nearest-neighbor problems discussed in Section 39.1. In particular, it was shown [Ind98] that any data structure for (r,c)-NN under l_∞^d for $c<3$ can be used to solve the partial match problem with parameter d, using essentially the same query time and storage (the number of points in the database is the same in both cases). Thus, unless Conjecture 2 is false, the 3-approximation algorithm from Section 39.1 is optimal, in the sense that it provides the smallest approximation factor possible while preserving polynomial (in d) query time and subexponential (in d) storage. Note that this result resembles the non-approximability results based on the $\mathrm{P}\neq\mathrm{NP}$ conjecture.

On the other hand, it was shown [CIP02] that the exact near-neighbor problem

[2] For $d=\log^{O(1)} n$, Conjecture 2 is true by a simple incompressibility argument. At the same time, the status of Conjecture 1 for $d\in[\omega(\log n),\log^{O(1)} n]$ is still unresolved.

under the l_∞^k norm can be reduced to solving the partial match problem with the parameter $d = (k + \log n)^{O(1)}$; the number of points n is the same for both problems. In fact, the same holds for a more general problem of ***orthogonal range queries***. Thus, Conjecture 2, and its variant for the $(r, 1)$-NN under l_∞^d (or for orthogonal range queries), are equivalent. This strengthens the belief in validity of Conjecture 2, since the exact nearest neighbor under l_∞ norm and the orthogonal range query problem received additional attention in the Computational Geometry community.

39.4 LOW VS. HIGH DIMENSIONS IN COMPUTATIONAL GEOMETRY

It is apparent that nearest neighbors and related problems in high dimensions enjoy properties quite different from their low-dimensional counterparts (see Chapter 51). Among the main differences are:

- Exact computation seems (and is conjectured to be) intractable in high dimensions; on the other hand, very efficient algorithms exists in low-dimensional cases.
- The core problem that seems to capture the computational difficulty is the near-neighbor problem in Hamming space $\{0, 1\}^d$, a problem trivial for constant dimension.
- Unlike the low-dimenional case, the tools of ***combinatorial geometry*** are rarely used to design or analyse algorithms in high dimensions. This phenomenon seems to reflect the fact that the typical tools (such as complexity of arrangements, or packing bounds) lead to exponential algorithmic complexity. Instead, tools from functional analysis (most notably embeddings) are used.

Nevertheless, there seems to be interesting connections between low and high dimensional scenarios. For example, the key componenent of several reductions given in Section 39.2 is the result of Eppstein [Epp95]. His algorithm was originally developed with low-dimensional applications in mind; however, its framework was sufficiently general to be useful in high-dimensional case as well.

As an example of impact in the other direction, one could mention the nearest-to-near neighbor reduction of [IM98]. When applied in the low-dimensional case, their result gave the first algorithm for $(1 + \epsilon)$-approximate nearest neighbor, with polynomial space and polylogarithmic query time, for dimension d up to $O(\log n)$ (earlier results could provide that bound only for $d = O(\log \log n)$, due to exponential dependence of the query time on the dimension). These results were further refined in the low-dimensional context in [HP01, AM02], yielding an efficient approximate nearest-neighbor data structure for low dimensions.

Finally, we mention an example of a fruitful marriage between low- and high-dimensional techniques. Consider the following problem. For a constant d, assume we are given n $(d-1)$-dimensional flats $H_1 \ldots H_n$ living in $\mathbb{R}^d$, as well as a set P of n points P in $\mathbb{R}^d$. The goal is to compute a tree spanning the points in P, such that the total number of times a tree edge crosses a flat is as small as possible.

In [HPI00], the authors provided a c-approximate algorithm for this problem, with running time $O(n^{2d/(d+1)+\delta} + n^{1+1/c} \log^{O(1)} n)$, for any $\delta > 0$ (the factors polynomial in $1/(c-1)$ are omitted). Note that this time is subquadratic for any constant d and $c > 1$. The main idea of the algorithm is to observe that the number of flats crossed on the way from point p to p' is a metric, and moreover, this metric can be isometrically embedded into n-dimensional Hamming space. This allows one to use the high-dimensional approximate MST algorithms from Section 39.2. To make that algorithm run fast, one needs to perform the dimensionality reduction before computing MST (essentially as in Theorem 39.1.1). However, just computing the n-dimensional representation of each of n points in P requires $\Omega(n^2)$ time. To avoid this bottleneck, the dimensionality reduction performed on "implicit" n-dimensional representations of the points in P, by using partition trees of Matoušek.

RELATED CHAPTERS

Chapter 8: Low-distortion embeddings of discrete metric spaces
Chapter 24: Arrangements
Chapter 36: Range searching
Chapter 51: Pattern Recognition

REFERENCES

[AHL01] H. Alt and L. Heinrich-Litan. Exact l_∞-nearest neighbor search in high dimensions. *Proc. 17th Annu. ACM Sympos. Comput. Geom.*, pages 157–163, 2001.

[AM02] S. Arya and T. Malamatos. Linear-size approximate voronoi diagrams. *Proc. ACM-SIAM Sympos. Discrete Algorithms*, pages 147–155, 2002.

[BOR99a] A. Borodin, R. Ostrovsky, and Y. Rabani. Lower bounds for high dimensional nearest neighbor search and related problems. *Proc. 31st Annu. ACM Sympos. Theory Comput.*, 1999.

[BOR99b] A. Borodin, R. Ostrovsky, and Y. Rabani. Subquadratic approximation algorithms for clustering problems in high dimensional spaces. *Proc. 31st Annu. ACM Sympos. Theory Comput.*, 1999.

[BR00] O. Barkol and Y. Rabani. Tighter bounds for nearest neighbor search and related problems in the cell probe model. *Proc. 32nd Annu. ACM Sympos. Theory Comput.*, 2000.

[Bro98] A. Broder. Filtering near-duplicate documents. *Proc. FUN*, 1998.

[BT01] J. Buhler and M. Tompa. Finding motifs using random projections. *Proc. Annu. Internat. Conf. Computational Molecular Biology (RECOMB01)*, 2001.

[Buh01] J. Buhler. Efficient large-scale sequence comparison by locality-sensitive hashing. *Bioinformatics*, 17:419–428, 2001.

[Buh02] J. Buhler. Provably sensitive indexing strategies for biosequence similarity search. *Proc. Annu. Internat. Conf. Computational Molecular Biology (RECOMB02)*, 2002.

[BV02] P. Beame and E. Vee. Time-space tradeoffs, multiparty communication complexity, and nearest-neighbor problems. *Proc. 34th Annu. ACM Sympos. Theory Comput.*, 2002.

[CCGL99] Amit Chakrabarti, B. Chazelle, Benjamin Gum, and Alexey Lvov. A lower bound on the complexity of approximate nearest-neighbor searching on the hamming cube. *Proc. 31st Annu. ACM Sympos. Theory Comput.*, 1999.

[CDF$^+$00] E. Cohen, M. Datar, S. Fujiwara, A. Gionis, P. Indyk, R. Motwani, J. Ullman, and C. Yang. Finding interesting associations without support prunning. *Proc. 16th Internat. Conf. Data Engineering (ICDE)*, 2000.

[Cha02] M. Charikar. Similarity estimation techniques from rounding. *Proc. 34th Annu. ACM Sympos. Theory Comput.*, 2002.

[CIP02] Moses Charikar, Piotr Indyk, and Rina Panigrahy. New algorithms for subset query, partial match, orthogonal range searching and related problems. *Internat. Colloquium on Automata,Languages, and Programming*, 2002.

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

[DW82] L. Devroye and T.J. Wagner. Nearest neighbor methods in discrimination. *Handbook of Statistics, volume 2, P.R. Krishnaiah and L.N. Kanal, editors, North-Holland*, 1982.

[Epp95] D. Eppstein. Dynamic euclidean minimum spanning trees and extrema of binary functions. *Discrete Comput. Geom.*, 13:111–122, 1995.

[Epp99] D. Eppstein. Personal communication. 1999.

[GG91] A. Gersho and R.M. Gray. *Vector Quantization and Data Compression*. Kluwer, 1991.

[GIM99] A. Gionis, P. Indyk, and R. Motwani. Similarity search in high dimensions via hashing. *Proc. 25th Internat. Conf. Very Large Data Bases (VLDB)*, 1999.

[GIV01] A. Goel, P. Indyk, and K. Varadarajan. Reductions among high-dimensional geometric problems. *Proc. ACM-SIAM Sympos. Discrete Algorithms*, 2001.

[GPY94] D. Greene, M. Parnas, and F. Yao. Multi-index hashing for information retrieval. *Proc. 35th Annu. IEEE Sympos. Foundations of Computer Science*, pages 722–731, 1994.

[GSM03] B. Georgescu, I. Shimshoni, and P. Meer. Mean shift based clustering in high dimensions: A texture classification example. *Proc. 9th International Conference on Computer Vision*, 2003.

[GW97] M.X. Goemans and D.P. Williamson. The primal-dual method for approximation algorithms and its application to network design problems. *Approximation Algorithms*, 1997.

[HGI00] T. Haveliwala, A. Gionis, and P. Indyk. Scalable techniques for clustering the web. *WebDB Workshop*, 2000.

[HIM03] S. Har-Peled, P. Indyk, and R. Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. *Submitted, 2003. Merges [IM98] and [HP01].*

[HL02] L. Heinrich-Litan. Exact l_∞-nearest neighbor search in high dimensions. *Proc. 18th European Workshop on Computational Geometry*, 2002.

[HP01] S. Har-Peled. A replacement for voronoi diagrams of near linear size. *Proc. Sympos. Foundations of Computer Science*, 2001.

[HPI00] S. Har-Peled and P. Indyk. When crossings count - approximating the minimum spanning tree. *Proc. Annu. ACM Sympos. Computational Geometry*, 2000.

[IM98] P. Indyk and R. Motwani. Approximate nearest neighbor: towards removing the curse of dimensionality. *Proc. 30th Annu. ACM Sympos. Theory Comput.*, 1998.

[Ind98] P. Indyk. On approximate nearest neighbors in l_∞ norm. *J. Comput. Syst. Sci., to appear. Preliminary version appeared in Proc. Sympos. Foundations of Computer Science*, 1998.

[Ind00] P. Indyk. Dimensionality reduction techniques for proximity problems. *Proc. 9th ACM-SIAM Sympos. Discrete Algorithms*, 2000.

[Ind01] P. Indyk. *High-dimensional computational geometry.* Dept. of Comput. Sci., Stanford Univ., 2001.

[Ind02] P. Indyk. Approximate nearest neighbor algorithms for frechet metric via product metrics. *Proc. Sympos. Comput. Geom.*, 2002.

[Ind03] P. Indyk. Better algorithms for high-dimensional proximity problems via asymmetric embeddings. *Proc. ACM-SIAM Sympos. Discrete Algorithms*, 2003.

[IST99] P. Indyk, S.E. Schmidt, and M. Thorup. On reducing approximate mst to closest pair problems in high dimensions. *Manuscript*, 1999.

[JV99] K. Jain and V. Vazirani. Primal-dual approximation algorithms for metric facility location and k-median problems. *Proc. Sympos. Foundations of Computer Science*, 1999.

[Kle97] J. Kleinberg. Two algorithms for nearest-neighbor search in high dimensions. *Proc. Twenty-Ninth Annu. ACM Sympos. Theory of Computing*, 1997.

[KOR00] E. Kushilevitz, R. Ostrovsky, and Y. Rabani. Efficient search for approximate nearest neighbor in high dimensional spaces. *SIAM J. Comput.*, 30(2):457-474, 2000.

[KWZ95] R.M. Karp, O. Waarts, and G. Zweig. The bit vector intersection problem. *Proc. 36th Annu. IEEE Sympos. Foundations of Computer Science*, pages 621–630, 1995.

[LT80] R. Lipton and R. Tarjan. Applications of a planar separator theorem. *SIAM J. Comput.*, 9:615–627, 1980.

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

[MNSW94] P.B. Miltersen, N. Nisan, S. Safra, and A. Wigderson. Data structures and asymmetric communication complexity. *Proc. 26th Annu. ACM Sympos. Theory Comput.*, 1994.

[MP69] M. Minsky and S. Papert. *Perceptrons.* MIT Press, Cambridge, 1969.

[OMST02] Z. Ouyang, N. Memon, T. Suel, and D. Trendafilov. Cluster-Based Delta Compression of Collections of Files. *Proc. International Conference on Web Information Systems Engineering (WISE)*, 2002.

[Riv74] R.L. Rivest. *Analysis of Associative Retrieval Algorithms.* Ph.D. thesis, Stanford Univ., 1974.

[SH75] M.I. Shamos and D. Hoey. Closest point problems. *Proc. 16th Annu. IEEE Sympos. Found. Comput. Sci.*, pages 152–162, 1975.

[Shi00] N. Shivakumar. *Detecting digital copyright violations on the Internet (Ph.D. thesis).* Dept. of Comput. Sci., Stanford Univ., 2000.

[WSB98] Roger Weber, H.J. Schek, and Stephen Blott. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. *Proc. 24th Int. Conf. Very Large Data Bases (VLDB)*, 1998.

[Ya01] C. Yang. MACS: Music Audio Characteristic Sequence Indexing for Similarity Retrieval. *Proc. Workshop on Applications of Signal Processing to Audio and Acoustics*, 2001.

[Yia00] P.N. Yiannilos. Locally lifting the curse of dimensionality for nearest neighbor search. *Proc. ACM-SIAM Sympos. Discrete Algorithms*, 2000.