HIGH-DIMENSIONAL COMPUTATIONAL GEOMETRY

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF COMPUTER SCIENCE
AND THE COMMITTEE ON GRADUATE STUDIES
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Piotr Indyk
September 2000

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

Rajeev Motwani
(Principal Adviser)

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

Leonidas Guibas

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

Serge Plotkin

Approved for the University Committee on Graduate Studies:

Abstract

Consider the following problem: given a database of points in a multidimensional space, construct a data structure which given any *query* point finds the database point(s) closest to it. This problem, called *nearest neighbor search* has been of great importance to several areas of computer science, including pattern recognition, databases, vector compression, computational statistics and data mining. Many of the above applications involve data sets whose size and dimensionality are very large. 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*. Apart from the nearest neighbor, the class contains problems like closest pair, diameter (or furthest pair), minimum spanning tree and clustering problems. In the latter case the goal is to find a partition of points into k clusters, in order to minimize a certain function. Example functions are: the sum of the distances from each point to its nearest cluster representative (this problem is called k-median), the maximum such distance (k-center), the sum of all distances between points in same clusters (k-clustering), etc.

Since these problems are ubiquitous, they have been investigated in computer science for a long while (e.g., in 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 *small dimension*. Unfortunately, their running time grows *exponentially* with the dimension.

In this thesis we show that it is in fact possible to obtain efficient algorithms

for the aforementioned problems, if we are satisfied with answers which are *approximate*. The running time of our algorithms for the aforementioned problems has only *polynomial* dependence on the dimension, and sublinear (for the nearest neighbor problem) or subquadratic (for the closest pair, minimum spanning tree, clustering etc.) dependence on the number of input points.

The highlights of the results include:

1. Several algorithms for the $(1+\epsilon)$-approximate *near neighbor* problem, where the goal is to find a database point within a pre-specified distance from the query point. In particular, for the Hamming metric, we give an algorithm with $O(dn^{1/(1+\epsilon)})$ time[1] per query/insertion/deletion, as well as an algorithm with $O(d \log n/\epsilon^2)$ query time but large storage space. For the maximum (i.e., l_∞) norm, we give an algorithm which achieves $O(\log_{1+\delta} \log d)$-approximation for any $\delta > 0$, with $O(d \log n)$ query time and $O(dn^\delta)$ update time. This in particular gives a 3-approximate data structure using slightly superpolynomial storage of $O(n^{\log d+1})$.

2. Efficient reductions (with $\log^{O(1)} n$ overhead) to dynamic approximate near neighbor from a variety of problems, including nearest neighbor, furthest neighbor, closest pair, minimum spanning tree, facility location and bottleneck matching. In particular, the sublinear-time algorithms for the near neighbor described above, together with the reductions, yield sublinear-time algorithms for nearest and furthest neighbor, as well as subquadratic-time algorithms for the remaining problems.

3. Efficient embeddings to Hamming metric and l_∞ norm from other metrics, including Euclidean and Manhattan norms and Hausdorff metrics. This allows us to extend the aforementioned results to these metrics as well.

4. Applications of the techniques used to obtain the above results to give efficient algorithms for problems outside the scope of "high-dimensional computational geometry". These problems include: computing statistics for a stream of data

[1]We use n to denote the number of data points and d to denote the dimensionality.

using small space, finding spanning trees crossing few hyperplanes in low dimensions, and approximate pattern matching.

5. Subquadratic-time approximation algorithms for certain problems defined over general metrics, including 1-median, k-median, and 2-clustering. E.g., the algorithm for k-median works in time (roughly) $O(nk)$.

Acknowledgments

Many people helped me to develop, stimulate and direct my interest in science and mathematics. I would like to use this opportunity to thank them all, for their enthusiasm, encouragement and support, without which this thesis would never happen.

I will start by thanking my primary school math teacher, Jadwiga Pietrzak. The time we spent together solving mathematical problems and puzzles was invaluable in developing my interest in mathematics. She also introduced me to the then-novel concept of reading mathematical books, which proved very useful in the years to come.

My good luck for teachers continued in my high school. I am indebted to Kazimierz Napiórkowski, for his enthusiasm about every single crazy computer idea I ever had, and for never-ending discussions about various mathematical concepts. I also benefited a lot from few but important conversations with Andrzej Góralski and Wojciech Tomalczyk, who helped me to decide what I want to do after graduation.

During my years in college, I received lots of help from my undergraduate advisors, Andrzej Skowron and Bogdan Chlebus; I would like to thank them for their attention, collaboration and for helping me write my first papers. I am also indebted to Krzysztof Diks for his incredibly enthusiastic, Hitchcockian-style lectures on graph theory and algorithms, which attracted me to research in that area.

After I moved to Stanford, I benefited a lot from people around here. First of all, I would like to express my gratitude to my advisor, Rajeev Motwani. During the early years of my Ph.D. studies he helped me to focus on the more interesting and important problems, and he was always ready to listen to my questions and ideas, no matter how good (or bad) they were. I am also indebted to Serge Plotkin, whose lectures on network algorithms gave me a significant insight into that topic. I would also like to thank Leo Guibas and Julien Basch, from whom I learned computational geometry by "osmosis" (or at least I think I did).

My Stanford experience would not be the same without my friends, colleagues and office-mates (categories not mutually exclusive). I would like to thank Vassilis, Suresh, Ashish, Moses, Jeffrey, Shiva, Julien, Aris, Sudipto, Lynn, Adam, Glen, Liadan, Mayur, Kamesh, An, Jeremy, John & friends and many others, for making my life fun and enjoyable. I would also like to thank my foosball-mates: Shiva, Kinshuk, Dave, Sean, Lucas, Aris, Sudipto and Drago, for helping me to avoid doing work every now and then.

During the summer interships, I benefited a lot from the guidance and support of my mentors: Prabhakar Raghavan, Byron Dom and Soumen Chakrabarti in IBM Almaden, Muthu Muthukrishnan and Martin Farach-Colton in Bell Labs, and Muthu Muthukrishnan, Joan Feigenbaum and Martin Strauss in AT&T. I would especially like to thank Muthu, for collaboration, support and sponsoring some of the research done in this thesis by lending me money for living (in addition to my gratitude, I also paid him the money back, though).

I learned a lot from my co-authors, whose ideas and suggestion were invaluable in discovering of many of the algorithms presented in this thesis. I would also like to thank those who were too modest to acknowledge that their contribution deserved a co-authorship, which it did. In particular, I would like to thank C. Sinan Güntürk for the analytic proof of Lemma 15, and to Claire Kenyon, who suggested the current algorithm for 2-clustering in Chapter 6.

In addition, I would like to thank people who provided me with comments about many of the results contained in this thesis, as well as about their presentation. In particular, I would like to thank Sariel Har-Peled for several discussions on "computational geometry and the meaning of life" and related topics.

Finally, I would like to thank those whom I owe the most: my parents, Wiesław and Helena, for their love and support, whether it was time for hard work or for taking it easy.

Piotr Indyk
Stanford, California
September, 2000

I gratefully acknowledge the fellowships and grants that supported my research at Stanford. I was supported by the Batory Foundation Grant, Rambus Fellowship, ARO MURI Grant DAAH04-96-1-0007, NSF Grant IIS-9811904, and by NSF Award CCR-9357849, with matching funds from IBM, Mitsubishi, Schlumberger Foundation, Shell Foundation, and Xerox corporation.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Consider the following problem: given a database of points in a multidimensional space, construct a data structure which given any *query* point q finds the database point closest to q. This problem, called *nearest neighbor search*[1], has been of great importance to several areas of computer science since its beginning. One of the earliest applications of nearest neighbor is to the field of pattern recognition [FH51, CH67]. In this case the points are *feature vectors* describing certain objects (e.g., hospital patients). Each point is assigned a *label* (e.g., the success or failure of an operation). Given a new point q with unknown label, the nearest neighbor method finds the database point closest to q and assigns its label to q. This method and a number of its variants, is fairly old, but its simplicity and generality continue to make it a very attractive classification tool.

Another application for which nearest neighbor algorithms are of crucial importance is management of multimedia data. The basic idea is to represent database objects (e.g., text documents), by feature vectors, and search for similar documents by finding close feature vectors. For example, the approach of the *vector space model* [SWY75] for text retrieval (which is the standard approach used in information retrieval [Sal89, SM83]) is to map text documents into vectors by counting for each word the number of their occurrences in the document; the (weighted) vectors are

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

then compared using, say, Euclidean distance. Similar ideas are used for images, video and audio sequences, genetic sequences any other data. Other applications of nearest neighbor search include vector compression [GG91], computational statistics [DW82] and data mining.

Not surprisingly, many of the above 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 (at least 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 fairly large class of *proximity problems*, which, roughly speaking, consists of problems whose definitions involve the notion of distance between the input points. Apart from the nearest neighbor, the class contains problems like:

- closest pair: find the pair of points such that the distance between them is the smallest among all pairs. A variant of this problem, which we call close pairs, requires to report all pairs with distance smaller than a prescribed threshold.

- diameter (or furthest pair): the maximization version of the closest pair

- minimum spanning tree: find a tree spanning all points whose sum of edge lengths is minimized

- clustering problems: find a partition of points into some number of clusters (say k), in order to minimize a certain function. Example functions are: the sum of the distances from each point to its nearest cluster representative (this problem is called k-median), the maximum such distance (k-center), the sum of all distances between points in same clusters (k-clustering), etc.

Since these problems are ubiquitous, they have been investigated in computer science for a long while. In particular, they have been among the first problems investigated in the field of computational geometry when it emerged in mid seventies. As a result of this research effort, many efficient solutions have been discovered for the case when the points lie in a space of *small dimension*. For example, if the points lie in the plane, then:

- nearest neighbor problem can be solved with $O(\log n)$ time per query, using only $O(n)$ storage [SH75, LT80]. The preprocessing needed to build the data structure takes time $O(n \log n)$
- closest pair problem can be solved in $O(n)$ time [Rab76]
- diameter and minimum spanning tree can be computed in $O(n \log n)$ time [Sha75]

These algorithm are very efficient both in theory an practice. Unfortunately, as the dimension grows, the algorithms become less and less efficient. More specifically, their running time grows *exponentially* with the dimension. In particular:

- the nearest neighbor problem can be solved with $O(\log n)$ time per query, but using at least $n^{d/2}$ space [Cla88, Mei93]
- the closest pair algorithms require at least $2^d n$ time [Rab76]
- the minimum spanning tree algorithms require roughly $n \log^d n$ time [Yao82]
- the diameter algorithms require roughly $n^{2-1/d}$ time [AM92]

Therefore, even for moderate dimension, the nearest neighbor data structure uses an exorbitant amount of storage! The other algorithms become inefficient as well; e.g., if $d = \Omega(\log n)$, then the closest pair is best solved by computing the distances between all pairs and reporting the smallest one. This naive algorithm takes time quadratic in the size of the database, which is prohibitive for databases containing, say, more than one hundred thousand points. This led many researchers to conjecture that no efficient solutions exists for these problems when the dimension is large enough [MP69]; this difficulty has been referred to as the "curse of dimensionality".

In this thesis we show that it is in fact possible to obtain efficient algorithms for the aforementioned problems, if we are satisfied with answers which are *approximate*. The notion of approximation is best explained for the nearest neighbor: instead of reporting the point p closest to q, we can report *any* point within distance $(1 +$

ϵ) times the distance from q to p. Similar definitions can be naturally applied to all of the above problems. We mention that the fact that approximation helps to decrease the complexity of algorithms for high-dimensional spaces has been known for a while (see Tables 1.1 and 1.2), although most of those algorithms still suffered from exponential dependence on the dimension (or linear/quadratic dependence on the input size), with the exception of the earlier results of [KWZ95] and [GPY94] (described in detail at the end of this Chapter) and the independently obtained results of [KOR98] and [BOR99b] (see Tables 1.2 and 2.2).

As a result of approximation, many of the high-dimensional problems can be solved much more efficiently than before. For example, the closest pair can be solved essentially in time $O(dn^{1+1/(1+\epsilon)})$. This means that setting ϵ to (say) 1 allows us to compute approximate closest pair in time $dn^{1.5}$, that is, much more efficiently than before. Similar savings can be obtained for other problems. Therefore, our algorithms provide an accuracy/speed trade-off, which enables a software designer to make a choice best suited for the application in hand.

1.1 The overview

The basic structure of our results is shown in Figure 1.1. The main part of the thesis

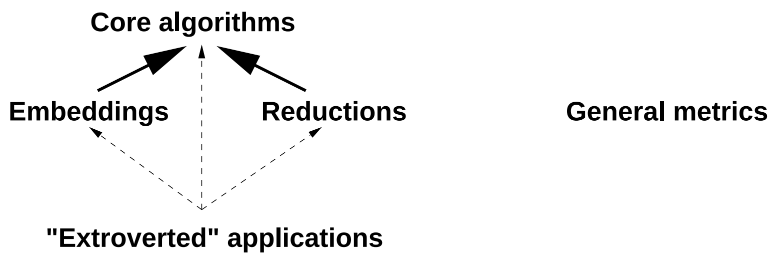

Figure 1.1: The structure of the results.

is formed by three components: *core algorithms*, *reductions* and *embeddings*. In the following we explain each of these components in more detail.

1. "Core algorithms": contains algorithms for the basic high-dimensional problem in basic metrics. The basic problem is the decision version the approximate nearest neighbor, which we call *approximate point location in equal balls*[2], or c-PLEB, where $c = 1 + \epsilon \geq 1$ is the approximation factor. Formally, given a query q, we want to check if there is any database point within a pre-specified distance r from q; if such a point exists, we want the data structure to output any point within distance $c \cdot r$ from q. Moreover, we allow the data set to be *dynamic*, i.e., the data structure should enable insertions and deletions of points at small cost.

   We will design such algorithms for two metrics: the Hamming metric and the l_∞ norm (see Preliminaries for definitions). For Hamming metric we give two algorithms: one with slow update but very small query time and one with fairly small update and query times. All of the algorithms provide storage/update time/query time/accuracy trade-offs; see Table 2.2 for details.

   For the l_∞ norm we give only one algorithm. Again, it provides a space/accuracy trade-off: on one side it guarantees a 3-approximation using slightly superpolynomial storage; on the other side, it offers only slightly superlinear storage but with approximation factor $O(\log \log d)$, where d is the dimension. Interestingly enough, there is evidence that $c = 3$ is the best we can do using subexponential storage (see Chapter 4 for details).

   All of the c-PLEB algorithms are dynamic, although the algorithms which use large space also have large update time, so they are of interest primarily in the static enviroment.

2. "Reductions": contains algorithmic reductions from a large number of problems to dynamic c-PLEB; the list of problems includes all of the aforementioned proximity problems. The reduction tree is shown in Figure 1.2.

   Almost all reductions work for any *metric*, i.e., only require the distance function to be symmetric and satisfy the triangle inequality; the only exception is the

[2] We also call it *near neighbor*.

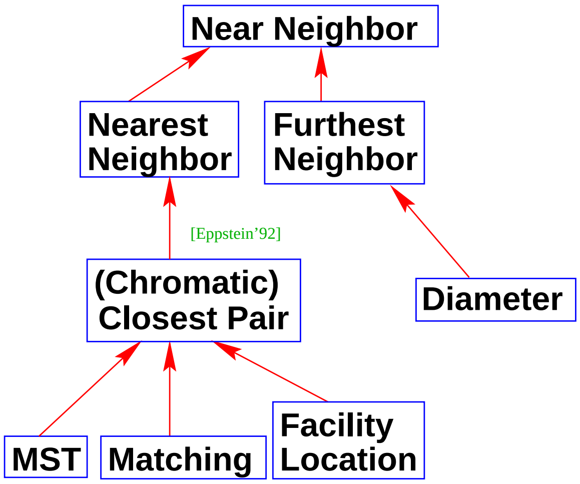

Figure 1.2: The reduction tree.

reduction from furthest neighbor, which works only for the Euclidean metric. All reductions use the c-PLEB algorithm as a black-box, or in other words, they are expressed as a sequence of dynamic c-PLEB operations in a given metric. The time overheads incurred by the reductions are small (polylogarithmic in the data size). Thus, if we have a dynamic c-PLEB with sublinear query/update time and subquadratic preprocessing time, we can compute c-approximate closest pair, diameter, minimum spanning tree, etc., in subquadratic time as well.

3. "Embeddings": contains distance-preserving mappings into the Hamming metric and the l_∞ norm from more complex metrics, like Euclidean norm l_2, *Manhattan* norm l_1 or *Hausdorff metrics* (see Preliminaries for definitions). These mappings allow us to extend the results from part (1) to the new metrics. Since almost all results from part (2) work for arbitrary metrics, we extend those results to new metrics as well. Note that the total number of resulting algorithms is equal to the number of reductions *times* the number of embeddings.

   Apart from the embeddings between different metrics, we also show several embeddings which preserve the metrics but reduce their dimensionalities, for l_1

and l_2 norms. Such embeddings allow us to achieve faster algorithms, usually replacing the dimension by a factor polylogarithmic in the number of points.

Finally, we also investigate *asymmetric embeddings*. These are embeddings which can be used to embed non-metric structures (e.g., partially ordered sets) into metric spaces; see Chapter 4 for definitions. Since searching in such structures is conjectured to be hard (e.g., the *subset query* problem, see Preliminaries), we can conclude that searching in certain metrics is probably hard as well.

The remaining two components from Figure 1.2 are "Extroverted applications" and "General metrics". In the former part we present several results for problems which have the following two properties in common: they are not "high-dimensional problems" as such, and they can be solved using the "high-dimensional tools" developed earlier. The problems include: computing statistics for a stream of data using small space, finding spanning trees crossing few hyperplanes in low dimensions, and approximate pattern matching. The solutions to these problems usually come by observing that they can be posed in "high-dimensional terms" and therefore the high-dimensional tools can be used.

The last part addresses the problems defined over *general* metric spaces. Unfortunately, in this abstract setting, certain problems provably do not have efficient algorithms. For example, the closest pair problem cannot be solved (even approximately) in time subquadratic in the number of points. We will show however, that other problems can be approximately solved in subquadratic time even in general metric spaces, e.g., the k-median problem can be solved roughly in time $O(nk)$, which is much smaller than n^2 if the number of cluster k is small; see Table 6.2 for other results. Note that we need at least $\binom{n}{2}$ bits in order to specify fully a metric over n points; therefore, our algorithms in fact run in time *sublinear* in the input size.

"High-dimensional computational geometry". In this thesis we often use the term "high-dimensional computational geometry" to describe the class of problems addressed here. This term seems appropriate, since many papers on related topics (e.g., [Ber93, Kle97, KOR98, BOR99a, BOR99b, FP99, GIV99]) present algorithmic solutions to problems traditionally attributed to computational geometry in spaces which are "high-dimensional". However, we should mention that our interpretation of the meaning of this term is much more narrow than "geometric computing in high-dimensional spaces". In particular, in this thesis we do not address problems such as linear and convex programming, computing volume of geometric objects, computational problems on lattices, etc. (although we use some results on convex programming in Chapter 3). Since even the term "computational geometry" seems to lack a clear and precise definition, we restrain ourselves from providing a precise definition of "high-dimensional computational geometry".

1.2 Preliminaries

In this section we introduce the problems and notation which we will use in the rest of the thesis.

Metrics and norms. A *metric* is a pair $M = (X, D)$, where X is arbitrary set and $D : X \times X \rightarrow \Re$ is a *distance function*. In order for M to be a metric, D has to satisfy the following three axioms:

1. reflexivity: $D(p, q) = 0$ iff $p = q$, for any $p, q \in X$
2. symmetry: $D(p, q) = D(q, p)$, for any $p, q \in X$
3. triangle inequality: $D(p, q) + D(q, u) \leq D(p, u)$, for any $p, q, u \in X$

For any $p \in X$ and $r > 0$ we use $B(p, r)$ to denote the *ball* of radius r centered at p, i.e., the set $\{q \in X : D(q, p) \leq r\}$. We also use $R(p, r_1, r_2)$ to denote the *ring* centered at p, i.e., the set $\{q \in X : r_1 \leq D(q, p) \leq r_2\}$.

For any set $S \subset X$, we define $\Delta(S) = \max_{p,q \in S} D(p, q)$ to be the *diameter* of S. Also, for any $p \in X$ and $S' \subset X$, we define $D(p, S) = \min_{q \in S} D(p, q)$ and $D(S, S') = \min_{p \in S} D(p, S')$. Note that $D(S, S')$ is *not* a metric.

Most of the metrics considered in this thesis are induced by a *norm*. A norm is a unary function (usually denoted by $|\cdot|$) from $\Re^d$ to $\Re$ satisfying the following axioms:

1. $|x| = 0$ iff $x = 0$, for any $x \in \Re^d$
2. $|c \cdot x| = |c| \cdot |x|$, for any $c \in \Re$ and $x \in \Re^d$
3. $|x + y| \leq |x| + |y|$, for any $x, y \in \Re^d$

It is easy to see that for $p, q \in \Re^d$ the function $D(p, q) = |p - q|$ is a metric. In the rest of the paper we will be primarily interested in *Minkowski norms* $|\cdot|_p$, for $p \in [1, \infty)$, defined as

$$|x|_p = (\sum_{i=1}^{d} |x_i|^p)^{1/p}$$

where x_i denotes the i-th coordinate of x. We use the notation l_p^d to denote the space $\Re^d$ with the $|\cdot|_p$ norm; we will skip d if the dimension of the space is irrelevant.

Let $V_p^d(r)$ denote the *volume* of a ball of radius r in l_p^d. The following fact is standard [Pis89, page 11].

Fact 1 *Let $\Gamma(.)$ denote the gamma function. Then* $V_p^d(r) = \frac{(2\Gamma(1 + 1/p))^d}{\Gamma(1 + n/p)} r^d$ *and* $V_2^d(r) = \frac{2\pi^{d/2}}{d\,\Gamma(d/2)} r^d$.

We also define *Hausdorff metrics.* This is a general class of metric defined over *subsets* of any metric $M = (X, D)$. The Hausdorff distance D_M between any two subsets $A, B \subset X$ is defined as

$$D_M(A, B) = \max(\max_{p \in A} D(p, B), \max_{p \in B} D(p, A))$$

We denote the Hausdorff space over M by $H(M)$. Additionally, the Hausdorff space over all s-subsets of M is denoted by $H^s(M)$.

Inequalities. For all $t, n \in \Re$ such that $|t| \leq n$ we have [MR95]

$$e^t(1 - t^2/n) \leq (1 + t/n)^n \leq e^t$$

The following power series expansions sometimes allow us to obtain useful inequalities.

$$e^x = 1 + x + x^2/2! + x^3/3! \ldots$$

$$\ln(1+x) = x - x^2/2 + x^3/3 \dots$$

Probability and randomness. Let $X_1 \dots X_n$ be random variables with values in $\{0,1\}$ such that $\Pr[X_i = 0] = p$. We say that $X = \sum_i X_i$ has a *Bernoulli distribution*. The *Chernoff bound* implies that [MR95]

$$\Pr[|X - np| \geq \epsilon np] \leq 2\exp(-\epsilon^2 np/3)$$

Let X be a random variable taking values in $\Re^d$. The *median* of X is the smallest number r such that

$$Pr[X \leq r] \geq 1/2$$

If X has Bernoulli distribution, then it is known that the median of X is equal to $\lceil np \rceil$.

Many of the algorithms described in this thesis are randomized. There are two basic variants of randomized algorithms, called *Monte Carlo* and *Las Vegas*. An algorithm of the first type can, with small probability, behave incorrectly. An algorithm of the second type is always correct, but can sometimes run longer than the bounds indicate; in such cases the bounds refer to the *expected running time* or to the running time which guarantees some large enough probability of correctness. Most of the algorithms in this thesis have constant probability of correctness. The probability can be amplified to $1-\delta$ for any $\delta > 0$ by using standard repetition techniques. Namely, we can run the algorithm $k = O(\log 1/\delta)$ times, each time using independent random bits.

We say that an event (e.g., termination of an algorithm) occurs with *very high probability*, if it occurs with probability $1 - 1/n^c$, where n is the size of the input and c can be made arbitrarily large by modifying the parameters of the algorithm by a constant factor. Note that by setting k to $O(\log n)$ we boost the chances of correctness of the algorithm from constant to very high probability.

Superimposed codes. A sequence of sets $S_1 \dots S_n$, $S_i \subset U$, is called a (k,n)-superimposed code [KS64] (or a (k,n)-code), if for any $I \subset \{1 \dots n\}$ such that $|I| \leq k$

and $i \notin I$, we have

$$S_i \not\subset \cup_{j \in I} S_j$$

An interesting problem is to find a (k,n)-code with minimum $|U|$. It is known that there exist families with universe size $O(k^2 \log_k n)$ [KS97] (or $O(k^2 \log n)$ via probabilistic construction) and that for $k = O(n^{1/3})$ the size is at least $\Omega(k^2 \log_k n)$ [CR96]. We will use superimposed codes to achieve both upper and lower bounds in Chapter 4. Here we mention the following generic construction of (k,u)-superimposed codes. Let $\mathcal{H} = \{h_1 \dots h_t\}$ be a family of hash functions $h_i : U \to \{0 \dots v-1\}$ such that

$$\Pr_{h \in \mathcal{H}}[h(x) = h(y)] \leq \frac{1}{2k}$$

Then it is easy to see that the family of sets $S_{h,z} = h^{-1}(z)$ for $h \in \mathcal{H}$ and $z \in \{0 \dots v-1\}$ is a $(k,|U|)$-superimposed code. To see that, take any set $A \subset U$, $|A| \leq k$ and $a \in U - A$. If we choose h at random, then

$$Pr[h(a) \in h(A)] \leq k\frac{1}{2k} = 1/2$$

Therefore, there exists h such that $h(a) \notin h(A)$. In this case we have $a \in S_{h,h(a)}$ and $A \cap S_{h,h(a)} = \emptyset$. Note that this construction gives a code of size $k|\mathcal{H}|$.

One can construct such $\mathcal{H}$ of size $O(k^2 \log^2 |U|)$ from certain error correcting codes, e.g. Reed-Solomon codes [MS77]. The latter codes are defined over an alphabet $\{0 \dots q-1\}$, have length q and for any $a + b = q$ they have q^a codewords with the smallest distance at least b. We can set $q = 2k \log |U|$ and $a = \log |U|$, then the number of codewords is at least $q^a \geq |U|$. For $i = 0 \dots q-1$, we define hash functions $h_i : U \to \{0 \dots q-1\}$ such that $h_i(x)$ is equal to the i-th symbol of the codeword number x. Since $b = 2k \log |U|$, we have that for any two $x \neq y$ we have

$$\Pr_{h \in \mathcal{H}}[h(x) = h(y)] \leq \frac{1}{2k}$$

We mention that computing $h_i(x)$ can be done in time polynomial in q.

Problems. In the following, we will list all the problems we consider in this thesis. We divide them into two classes: *on-line* and *off-line*. In the first case the goal is to design data structures which after some preprocessing can efficiently answer certain queries, e.g., find a nearest neighbor of a given point. Ideally, the data structure should be *dynamic*, i.e., enable fast updates of the data set; otherwise, we call them *static*. On the other hand, the goal of off-line problems is to find certain structure in the data set; there is no separation between preprocessing and query times. The off-line problems can be also dynamic, if the input set P changes.

The definitions below describe the c-approximate versions of the problems, for $c = 1 + \epsilon > 1$. The problems will be defined over a set P of points from a metric $M = (X, D)$.

The approximate on-line problems are listed below. We start from problems where both the query q and the elements of the data set P are points from a metric space.

- point location in equal balls ((r, c)-PLEB): given a query q, find a point $p' \in P$ such that $D(q, p') \leq cr$, if there exists $p \in P$ such that $D(p, q) \leq r$. If the value of r does not matter (e.g., when we can set r to 1 by scaling) we call this problem c-PLEB

- enumerative PLEB ((r, c)-EPLEB): given a query q, report *all* points $p \in P$ such that $D(p, q) \leq r$. The data structure can also report points $p' \in P$ such that $D(p, q) \leq cr$. We mention that (dynamic) c-EPLEB can be easily implemented using dynamic c-PLEB, since if we get any answer p' to query q from c-PLEB, we delete p' from P and repeat the process until there is no point left; at the end, we add all retrieved points back to the data structure. The running time of this procedure is bounded by the number of retrieved points times (c-PLEB query time + c-PLEB update time).

- counting PLEB ((r, c, α)-EPLEB): given a query q, report a number $N(q)$ such that if $N_R(q)$ is the number of points within distance R from q, then

$$(1 - \alpha)N_r(q) \leq N(q) \leq (1 + \alpha)N_{cr}(q)$$

- nearest neighbor search (c-NNS or c-NN): given a query q, find a point $p' \in P$ such that $D(q, p') \leq cD(q, P)$

- furthest neighbor search (c-FNS or c-FN): given a query q, find a point $p' \in P$ such that $D(q, p') \geq 1/c \cdot \max_{p \in P} D(p, q)$

In addition, we consider the following two problems. Let P be a set of n subsets of $\{1 \dots d\}$. The *subset query problem* (SQ) is defined as follows: given a query set $Q \subset \{1 \dots d\}$, find a set $S \in P$ such that $Q \subset S$. This problem has been investigated for the last few decades (e.g., see [Riv74] and [Knu73], p. 557), but despite the research effort no algorithm with $2^{o(d)}$ space and $o(n)$ time has been found, at least for $d = \omega(\log n)$. Therefore, it is now widely conjectured (e.g., see [BOR99a]) that the problem is hard. In this thesis we will use the conjectured hardness of this problem as a basis for other hardness results.

The subset query problem can be generalized to the *orthogonal range query problem* (ORQ), which is defined as follows. Given a set of points $P \subset \{1 \dots u\}^d$, preprocess it to support *range* queries: is the intersection of P and a box $B = \{a_1 \dots b_1\} \times \{a_2 \dots b_2\} \times \dots \times \{a_d \dots b_d\}$ nonempty ? It is easy to see that SQ is a special case of ORQ when $u = 2$; thus the problem is at least as hard as SQ. In Chapter 4 we will show that in fact this problems are essentially equivalent.

The approximate off-line problems are defined as follows.

- closest pair (c-CP): find a pair of points $p', q' \in P$ such that $D(p', q') \leq c \min_{p,q \in X, p \neq q} D(p, q)$

- close pairs: find *all* pairs of points $p, q \in P$ such that $D(p, q) \leq r$, where r is a pre-specified distance. The algorithm can also output pairs $p', q' \in P$ such that $D(p', q') \leq cr$

- chromatic closest pair (c-CCP): 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

- bichromatic closest pair (c-BCP): as before, except that $c(p)$ assumes only two values

- diameter (or furthest pair): find $p, q \in P$ such that

$$D(p,q) \geq \max_{p',q' \in P} D(p',q')/c$$

- discrete minimum enclosing ball: find $p \in P$ such that

$$\max_{q \in P} \leq c \min_{p' \in P} \max_{q \in P} D(p',q)$$

- continuous minimum enclosing ball (CMEB, or MEB): as before, but $p \in X$

- minimum spanning tree (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 the weight of any tree spanning P

- (bottleneck) matching: assuming $|P|$ is even, find a set of $|P|/2$ non-incident edges joining points in P, such that the maximum length of an edge edge is minimized

- facility location: find a set $F \subset P$ such that the following function is minimized, given the cost function $c : P \rightarrow \Re^+$

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

  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

- k-median: find a set F of cardinality k minimizing the above function

- k-center: find a set C of cardinality k minimizing

$$\max_{p \in P} \min_{c \in C} D(p,c)$$

- k-clustering: partion P into $S_1 \ldots S_k$ to minimize

$$\sum_i \sum_{p,q \in S_i} D(p,q)$$

Query	Storage	Paper	Comments
$d^2 \log n$	$d^2 n \log n$	[Ber93, Cha97]	any norm, but $\epsilon \geq d$
$\epsilon^{-(d-1)/2} \log n$	$\epsilon^{-(d-1)/2} n \log n$	[AM93] [Cla94, Cha97]	Euclidean norm
$d^d \epsilon^{-d}$	dn	[AMN+94]	any norm
n	dn	[CL97]	for dot product
$d^2 \log^2 d$ n	n^{2d} $dn \log^2 n \log^2 d$	 [Kle97]	Euclidean norm

Table 1.1: Previous results for the approximate nearest neighbor problem.

Amortization. Let DS be a data structure supporting certain operations, say OP. We say that DS runs in *amortized* $T(n)$ time per operation, if any sequence of n operations takes at most $nT(n)$ steps. Note that each individual step could take much more than $T(n)$ steps.

Other notation. We will use $P(X)$ to denote the set containing all subsets of X. Also, we will use $P_s(X)$ to denote all subsets of X of cardinality at most s.

We use the notation $f(n) = \tilde{O}(g(n))$ to denote the fact that $f(n) = g(n) \log^{O(1)} n$.

We use $[n]$ to denote the set $\{0 \ldots n-1\}$.

1.3 Previous work

In this section we summarize previous work done on the problems we consider here in the paper. In order to present them in a concise way, we show them in tables. In particular, Table 1.1 contains earlier results for approximate NNS and Table 1.2 contains earlier results for off-line problems. We primarily focus on approximate problems, since many of the earlier results for the exact problems have already been mentioned in the introduction. The algorithms work for l_2^d unless stated otherwise.

In addition we mention a few other papers which contain results related to the topic of this thesis. In particular, Greene, Parnas, and Yao [GPY94] presented a scheme which, for binary data chosen *uniformly at random*, retrieves all points within Hamming distance r of q in time $O(dn^{r/d})$, using $O(dn^{1+r/d})$ preprocessing. Also,

Problem	Ref	Approx.	Time	Comments
diameter	folklore	$1+\epsilon$	$(1/\epsilon)^{2d}+dn$	
	[EK89]	$\sqrt{3}$	$O(dn)$	
	[FP99]	$1+\epsilon$	$O(dn\log n+n^2)$	
	[BOR99b]	$1+\epsilon$	$\tilde{O}(n^{2-O(\epsilon^2)}+dn)$	
minimum enclosing ball	[Gar95]	1	$\exp(\sqrt{d})+d^{O(1)}n$	
	many	$1+1/n^{\Theta(1)}$	$d^{O(1)}n$	via linear programming
MST	[Vai84]	$1+\epsilon$	$(1/\epsilon)^d n\log n$	
	[BOR99b]	$1+\epsilon$	$\tilde{O}(n^{2-O(\epsilon^2)}+dn)$	
facility location	[JV99]	3	$\tilde{O}(n^2)$	any metric
	[CG99]	2.4	$\tilde{O}(n^2)$	any metric
k-center	[AP00]	$1+\epsilon$	$n\log k+k/\epsilon^{dk}$	
	[FG88]	2	$\exp(d)n\log k$	
	[HS86]	2	nk	any metric
k-median	[KR99]	$1+\epsilon$	$\exp(1/\epsilon^d)n\log^2 n$	
	[JV99]	6	n^2	any metric
	[CG99]	4	n^3	any metric
	[Mat99]	$1+\epsilon$	$\epsilon^{-k^2 d}n\log^k n$	sum of *squares* of distances
min matching	[GT]	1	$\tilde{O}(n^{2.5})$	any metric
	[GW97]	2	$\tilde{O}(n^2)$	any metric
bottleneck matching	[EK75]	1	$\tilde{O}(n^{2.5})$	any metric

Table 1.2: Previous results for the off-line problems.

Karp, Waarts and Zweig [KWZ95] gave an efficient approximation algorithm for so-called *bit vector intersection* problem. The problem is defined as follows: given a set of n binary vectors $p_1 \dots p_n$ of length d, find all the pairs of vectors which have at least t ones in common. The input set has to satisfy roughly the following conditions. Let n_i be the number of ones in p_i, and let n_{ij} be the number of ones that p_i an p_j have in common.

1. $n_i \leq dP(1+\mu)$, for $P, \mu > 0$
2. there are only $O(n)$ pairs of vectors i, j such that $n_{ij} \geq (1+\mu)Pn_i$
3. $t = d \cdot P^s$, where $1 \leq s \leq 2 - \log_{1/P}(1+\mu)^2$

Their algorithm runs in $O(n^{s+\log_{1/P}(1+\mu)^2})$ time.

Both aforementioned papers use techniques somewhat similar to locality sensitive hashing from Section 2.2.2, although their analysis seems insufficient to give the results we obtain in that section.

Chapter 2

Core algorithms

In this chapter we focus on the basic building tool for geometric algorithms in high dimensions, namely the data structure for performing approximate point location in equal balls (c-PLEB). Recall that the problem is parametrized by $c = 1 + \epsilon$ for $\epsilon > 0$ and $r > 0$. For any query q it reports a point p' such that $D(p', q) \leq cr$ if there exists any point p within distance r from q. Notice that we could assume as well that if no such p exists then the structure returns an arbitrary point from P (instead of just "nothing"). This is due to the fact that it is easy to check if the returned point p is within distance cr from q; therefore, we can easily detect which of the cases occurred.

It is important to remember that both r and ϵ are fixed in advance and cannot be changed during the computation (we show how to relax some of these restrictions in Chapter 3). We refer to this problem as (r, c)-PLEB if the value of r is important[1].

The two particular metrics which we focus on in this chapter are Hamming metric and l_∞ norm; see Table 2.2 and the introduction to Section 2.1 for the detailed statement of the results. Interestingly enough, the algorithmic techniques for dealing with Hamming space are very different from those developed for l_∞. Example differences are:

[1]If the metric M induced by a norm, we can always assume $r = 1$, since otherwise we can initially scale all dimensions by a factor of $1/r$. This approach does not work in general, though (e.g., for Hamming metric).

- randomization: all algorithms for Hamming space use randomization in a natural way (although some of them can be derandomized). In particular, all algorithms exploit projections of a vector onto a random subspace. On the other hand, the algorithm for l_∞ norm is deterministic and based on the divide and conquer approach; there does not seem to be any natural randomized technique which could improve the bound.

- approximation quality: the algorithms for the Hamming space accept any approximation factor $c > 1$; the algorithm for l_∞ works only for c large enough (although still constant). In Chapter 4 we present an evidence that this discrepancy could indeed provably exist.

We will start the description from the algorithm for the l_∞ norm, since it uses techniques which are somewhat more "algorithmic" than those used for the Hamming metric, namely the divide-and-conquer approach. Then we proceed with Hamming metric.

2.1 l_∞ norm

In this section we present an algorithm for the c-PLEB in l_∞ norm. Firstly, we describe a *static* algorithm (i.e., no updates) from [Ind98] and then show how to modify the data structure to enable insertions/deletions in little time.

The static algorithm offers the following tradeoff: for any parameter $\rho > 0$ it uses $O(dn^{1+\rho}\log n)$-size storage and solves the $(r, c(d,\rho))$-PLEB problem with query time $O(d \log n)$ time for $c(d,\rho) = 4\lceil\log_{1+\rho}\log 4d\rceil + 1$. The preprocessing time is linear in the size of the data structure. Thus, by setting $\rho = \log 4d$ we guarantee to solve 5-PLEB using mildly superpolynomial storage; on the other hand we can guarantee a $O(\log\log d)$-nearest neighbor using a storage arbitrarily close to linear. After that, we show that by modifying the data structure slightly one can in fact solve 3-PLEB using $n^{\log d+1}$ space. Later, in Chapter 4 we show that solving c-PLEB for any $c < 3$ implies a solution to the subset query problem with essentially the same time/space bounds. The latter problem is conjectured to require exponential (in d) space if sublinear query time is desired. Thus, assuming the conjecture is true, our algorithm achieves best possible approximation factor using subexponential space.

2.1.1 Static algorithm

The algorithm is based on the notion of a hyperplane separator, defined as follows.

Definition 1 *For $\alpha, \beta, \gamma > 0$ such that $\alpha + \beta + \gamma = 1$, a hyperplane H is a (α, β, γ)-separator for a point set P if:*

- *the set*

$$P_L = \{p \in P | B(p,1) \cap H = \emptyset \text{ and } p \text{ lies on the left side of } H\}$$

  has cardinality $\alpha|P|$,

- *the set*

$$P_M = \{p \in P | B(p,1) \cap H \neq \emptyset\}$$

  has cardinality $\beta|P|$,

- *the set*

$$P_R = \{p \in P | B(p,1) \cap H = \emptyset \text{ and } p \text{ lies on the right side of } H\}$$

  has cardinality $\gamma|P|$.

The basic idea of the algorithm is as follows. We use hyperplanes $H_i(t)$ consisting of all points with the ith coordinate equal to t. We try to find hyperplane separator H having the property that $|P_R|$ is not "much smaller" than $|P_M|$. If H exists, we divide P into $P_1 = P_L \cup P_M$ and $P_2 = P_R \cup P_M$ and apply algorithm recursively on P_1 and P_2; we prove that (due to the fact that $|P_R|$ is comparable to $|P_M|$) the increase in storage caused by duplicating P_M is moderate. If H does not exist, we prove that a large subset C of P has a small diameter. In such a case we pick any point from C as its representative and apply the algorithm recursively on $P - C$.

For any (α, β, γ)-separator, define the quantity

$$L(\beta, \gamma) = \log_{1/(\gamma+\beta)} \frac{\beta+\gamma}{\gamma} = \log_{1/(\gamma+\beta)} \frac{1}{\gamma} - 1.$$

The quantity $L(\beta, \gamma)$ will upper bound the exponent ρ in the space bound $n^{1+\rho}$; therefore, it is of interest to keep it low. In order to do this, that it is of crucial importance to keep $1 - (\gamma + \beta) = \alpha$ fairly large (greater than a constant), since otherwise $L(\beta, \gamma) = \Omega(1/\alpha)$. On the other hand, large $1/\gamma$ is not really that much of a problem, since the dependence of L on $1/\gamma$ is only logarithmic. We will exploit this asymmetry below.

Also, note that $L(\beta, \gamma) > 0$ for any $\beta, \gamma > 0$ such that $\beta + \gamma < 1$. The following Lemma forms a basis of our algorithm.

Lemma 1 *For any set P and $\rho > 0$*

1. *Either there exists an (α, β, γ)-separator $H_j(t)$ for P such that $L(\beta, \gamma) \leq \rho$ and $\alpha, \gamma > \frac{1}{4d}$, or*

2. *There is a set $C \subset P$ of cardinality $|P|/2$ having diameter at most $4\lceil \log_{1+\rho} \log 4d \rceil$.*

Proof: We assume that for every $H_j(t)$ with $|P_R| > \frac{1}{4d}|P|$ the value of $L(\beta, \gamma)$ is at least ρ and show that it implies the existence of C as above. For any point $p \in P$ let $p_{|j}$ denote the jth coordinate of p; for simplicity in the following we assume $j = 1$. For an integer i define $S_i = \{p \in P | p_{|1} \in [2i, 2i+2)\}$ and $s_i = |S_i|/n$. Note that the sets S_i are disjoint and therefore $\sum_i s_i = 1$. Define $t_i = \sum_{j \geq i} s_i$ and $t'_i = \sum_{j \leq i} s_i$. By translating the origin we can make sure that $t_0 = 1/2$ (note, that by our assumption, no two database points have the same value of jth coordinates).

Claim 1 *For any $i \geq 0$ such that $t_{i+1} > \frac{1}{4d}$ we have $t_{i+1} < t_i^{1+\rho}$. Thus, $t_i \leq \frac{1}{2^{(1+\rho)^i}}$ for such i.*

Proof: Consider a plane $H_1(2i+1)$. Note that $t'_{i-1} \geq t_{i+1}$ for $i \geq 0$. We know that as long as $t_{i+1} > \frac{1}{4d}$ we have

$$L(s_i, t_{i+1}) > \rho.$$

or equivalently

$$\log_{\frac{1}{s_i+t_{i+1}}} \frac{s_i + t_{i+1}}{t_{i+1}} > \rho$$

or

$$\frac{s_i + t_{i+1}}{t_{i+1}} > \frac{1}{(s_i + t_{i+1})^\rho}.$$

As $t_{i+1} + s_i = t_i$ the above inequality implies

$$\frac{t_i}{t_{i+1}} > \frac{1}{t_i^\rho}$$

which implies the first part of the claim. The second part follows from the fact that $t_0 = \frac{1}{2}$ and the first part. ■

We now use Claim 1 to prove the theorem. Take $i = \lceil \log_{1+\rho} \log 4d \rceil$; from Claim 1 and monotonicity of t_i it follows that $t_i \le \frac{1}{4d}$. By a symmetric argument, $t'_{-(i+1)} \le \frac{1}{4d}$. Therefore, we have $\sum_{-i \le j \le i-1} s_i \ge 1 - \frac{1}{2d}$. Thus if we delete from P all points from S_j for $i \notin [-i, i-1]$ we decrease the size of P by at most a factor of $1 - \frac{1}{2d}$. The difference in the first coordinate for any pair of remaining points is at most $4i$. By repeating the above argument for each dimension we obtain a set (call it C) of size at least $\frac{n}{2}$ such that any pair of points differ on each coordinate by at most $R = 4i = 4\lceil \log_{1+\rho} \log 4d \rceil$; thus its diameter is bounded by R as well. ■

Lemma 1 facilitates the following algorithm.

Preprocessing. For a given set P we build a tree data structure in the following way. The root of the tree corresponds to the set P; its subtrees are created as follows. If we find a separator with $L(\beta, \gamma) \le \rho$ then we create two point sets $P_1 = P_L \cup P_M$ and $P_2 = P_M \cup P_R$ and build the data structures recursively on each of them. We also store the parameters of the separator at the root node; we call such a node a *separator* node. Otherwise, we know that P contains C of small diameter such that $|C| \ge n/2$. In this case we store the center of the box defining C, a point from C (representing C) and build recursively a data structure for $P - C$. We call such a node a *box* node. Thus each non-leaf node has at most two children.

Searching. Given a query point q searching is performed recursively as follows. If the root is a separator node, we check if q is on the left or on the right side of the separating hyperplane; in the first case we move to node P_1, otherwise to P_2. On the other hand, if the root is a box node, we compute the distance t between q and the box. If the distance is less than 1, we output any point from the box (which is within at most distance $R+1$ from q). If $t>1$, we know that the point does not belong to a ball around any point from C, thus we move to the node $P-C$.

The correctness of the above procedures can be easily seen. It is thus sufficient to analyze the complexity of the procedure.

Lemma 2 *The search time is $O(d \log n)$.*

Proof: Consider the path from the root to the leaf traversed by the searching algorithm for given q. The number of box nodes on that path can be at most $\log n$, as a child of such a node contains at most half of the points of its parent. The time spent per each such node is $O(d)$, therefore the total number of operations performed at box nodes is $O(d \log n)$. On the other hand, the number of separator nodes can be at most $O(d \log n)$, as a child of such a node contains at most a fraction of $(1-\frac{1}{4d})$ of the points of its parent. The time spent per separator is $O(1)$, thus the total incurred cost is $O(d \log n)$. ∎

In order to analyze the size of the data structure and the construction time, we will use the following lemma. Let T_P be a tree representing the data structure constructed for a set P. We define T'_P to be a (conceptual) binary tree obtained from T_P by modifying the box nodes in a following way. Instead of removing the points which fall inside the box, we create a balanced binary tree with the number of leaves equal to the number of inside points and attach it as the second child of the box node. The resulting data structure has still depth $O(d \log n)$ (since it could only increase by $\log n$); moreover the size of T_P is clearly upper bounded by the size of T'_P. In the following lemma we give a bound on the size of T'_P and then we use it to bound the space/time requirements of the preprocessing stage.

Lemma 3 *The tree T'_P has at most $O(n^{1+\rho})$ nodes.*

Proof: Let $N(n)$ be the maximum number of leaf nodes of T'_P for $|P|=n$. We will show inductively that $N(n) \le n^{1+L}$ where $L = L(\beta, \gamma) \le \rho$. Assume that $N(m) \le m^{1+L}$ for all $m < n$. We analyze only separator nodes, since the inductive step holds trivially for box nodes. For each separator node we can write the following inequality

$$N(n) \leq N((\alpha+\beta)n) + N((\beta+\gamma)n).$$

By the inductive assumption we get

$$\begin{aligned} N(n) &\leq ((\gamma+\beta)n)^{1+L} + ((\beta+\alpha)n)^{1+L} \\ &= n^{1+L}((\gamma+\beta)(\gamma+\beta)^{L} + (1-\gamma)^{1+L}) \end{aligned}$$

Since $L(\beta,\gamma) = L$ we can bound the latter quantity by

$$\begin{aligned} &\leq n^{1+L}((\gamma+\beta)\frac{\gamma}{\gamma+\beta} + (1-\gamma)) \\ &\leq n^{1+L}(\gamma + (1-\gamma)) \\ &= n^{1+L} \end{aligned}$$

■

Corollary 1 *The data structure T_P is of size at most $O(dn^{1+\rho})$ and can be constructed in time at most $O(d^2 n^{1+\rho}\log n)$.*

Proof: The first part follow from the upper bound for the size of T_P via T'_P and the fact that both separator and box nodes have description size $O(d)$. In order to bound the running time notice that the processing of a node (say v) with m_v points (i.e. computing s_i's and deciding whether the node is a separator node or a box node) can be done in dm_v time. Since T'_P is a binary tree, m_v can be bounded by the number of leaves of v in T'_P. Since the tree has depth $O(d\log n)$, each leaf node can be "charged" in this way only $O(d\log n)$ times. Therefore, the sum of all m_v's can be bounded by $O(d\log n \cdot n^{1+\rho})$, which gives the promised bound on the preprocessing time. ■

In this way we proved the following Theorem.

Theorem 1 *For any $\rho > 0$, there exists a deterministic algorithm for the $(4\lceil\log_{1+\rho}\log 4d\rceil + 1)$-PLEB using $O(dn^{1+\rho})$ storage with query time $O(d\log n)$. The data structure can be constructed in $O(d^2 n^{1+\rho}\log n)$ time.*

2.1.2 Achieving 3-approximation

The above result provides a 5-approximation algorithm with space $n^{\log d+1}$. Unfortunately the factor 5 is tight for the above algorithm, if we require $n^{O(\log d)}$ storage. To see this, consider a set of $2d$ points such that for each coordinate i data set with the following property for each coordinate i and value $v \in \{-1, 1\}$ there is exactly one point p such that $p_{|i} = -2v$. By replicating this data set $\frac{n}{2d}$ times we obtain n points such that:

- the fraction of points p s.t. $p_{|i} = -2$ is $\frac{1}{2d}$
- the fraction of points p s.t. $p_{|i} = 2$ is $\frac{1}{2d}$
- the fraction of points p s.t. $p_{|i} = 0$ is $1 - 1/d$

Now, one can verify that for this data set no good separator exists. However, the diameter of the data set is equal to 4. Therefore the approximation factor is equal to 5.[2]

In the following we present a modified version of the algorithm which achieves factor 3 within the space bound $n^{\log d+1}$. The basic idea is to enhance the search procedure for the the clusters of diameter 4 contained in $[-2, 2]^d$ (call it C). Firstly, note that we can assume $q \in [-3, 3]^d$. Moreover, observe that if the query point q belongs to $[-1, 1]^d$, then any point from C is within distance 3 from q, so we are done. Therefore, we only have to take care of the situation when there is i such that $|q_{|i}| \in (1, 3]$. Note that in such a case the only points p from P which could be within distance 1 from q are those s.t. $p_{|i} > 0$; call this set P_i. If we knew i during preprocessing, we could build a separate search structure for P_i. Since $|P_i| \leq n/2$, this approach would be very efficient. Of course, i is not known during the preprocessing. However, it turns out that we can afford to build the structures for *all* $i = 1 \ldots d$ and still keep the $n^{O(\log d)}$ space bound. Let $L = \log d + 1$. We need to show that the

[2]One could observe that the algorithm is not completely doomed, since the data set in fact *does* contain a subset of size $n/2$ of diameter 2 (the set of points with all coordinates non-negative). It is possible, however, to construct a similar data set for which no small subset has diameter less than 5, while keeping the fraction of coordinates with values in $\{-2, 2\}$ small. Since this counterexample does not seem to be particularly informative, we omit the description.

$N(n) \leq n^{1+L}$, with the notation as in the proof of Theorem 1. To see this, observe that the number of nodes in the data structures built for a cluster C of size n is bounded by

$$d \cdot N(n/2) \leq d \cdot n^{1+L}/2^{1+L} = n^{1+L}/4$$

and thus $N(n) \leq n^{1+L}$. The search time can be verified to remain $O(d \log n)$.

2.1.3 Dynamization

In this section we present a fairly general theorem which, in particular, allows us to convert the static data structure described in the previous section into a dynamic one, with only a small penalty in the query time bound. We will focus our attention on enabling deletions, since enabling insertions can be done using standard tools [Ove83].

In order for the theorem to be applicable, the data structure and the problem it solves has to satisfy several conditions. Firstly (mainly for simplicity) we will be interested in problems where the elements of the input set, the queries and the answers are just points from $\Re^d$. Moreover, the problem has to be *decomposable*: i.e., for any partition $P_1 \ldots P_l$ of the data set P and any query q, if a_i, $i = 1 \ldots l$ are the answers to q for the sets P_i, then it should be possible to compute an answer a to the query q for the set P from $a_1 \ldots a_l$; to keep things simple, we assume the computation time is $O(dl)$. Note that the c-PLEB problem is trivially decomposable.

The remaining assumptions refer to the type of the data structure solving the static problem. We assume that the data structure is represented by a tree T. Each node u_i of the tree is associated with a set $P_i \subset P$ of points. In particular, for the root node u_1 we have $P_1 = P$ and for each leaf node u_i we assume $|P_i| = 1$. In addition, each internal node contains a data structure D_i. Given the query q, the structure D_i decides (in time $Q_D(n)$) which child of u_i should be examined in order to find the answer to q. We assume that D_i is dynamic, with the update time $U_D(n)$. Note that if $h(n)$ is an upper bound for the height of T, then the search time is bounded by $h(n) \cdot Q_D(n)$.

We assume that the node data structures use space $S_D(n)$. We will measure the space used by the whole data structure by the quantity $S(T) = \sum_i |P_i|$; obviously, the actual space is $\sum_i S_D(|P_i|)$. We assume $S(T)$ is upper-bounded by $S(n) = \Omega(n)$.

For any point $p \in P$ define the multiplicity $m(p)$ of p as the number of sets P_i containing p. Obviously, the average multiplicity of points in P is at most $S(n)/n$. Thus, deleting a *random* point $p \in P$ would cost us at most $S(n)U_D(n)/n$ on the average, which is as good as we can get. Unfortunately, some points could be more costly than other. However, from Markov inequality, we know that the fraction of points with multiplicity greater than $MS(n)/n$ is at most $1/M$. This observation forms the basis of our algorithm, which is as follows. Initially, we build the tree T_1 using the static data structure algorithm. Then, we compute the multiplicity of each point p. Define the set P_1 to be the set of points with multiplicity less than $2S(n)/n$ (there are at least $n/2$ of them). We delete all points in $P - P_1$ from T_1 (which takes $nU_D(n)$ time) and apply recursively the same procedure for $P - P_1$ forming trees $T_2, T_3 \ldots T_l$ for sets $P_2, P_3 \ldots P_l$, $l = \log n$. Note that the total construction time is bounded by order of the original construction time $C(n)$ plus $S(n)U_D(n)$.

In order to perform search in the resulting data structure, we perform search in all structures $T_1 \ldots T_l$ and compose the answers; this takes at most $\sum_{i=1}^{l} h(n/2^i) \cdot Q_D(n/2^i) = O((h(n) \cdot Q_D(n) + d) \log n)$ time. In order to delete a point p, we delete it from T_i such that $p \in P_i$; this takes at most $O(S(n)/n \cdot U_D(n))$ time.

Theorem 2 *The resulting data structure has the following performance:*

- *construction time* $C'(n) = O(C(n) + nU_D(n))$
- *query time* $Q'(n) = O((h(n) \cdot Q_D(n) + d) \log n)$
- *deletion time* $D'(n) = O(S(n)/n \cdot U_D(n))$

By plugging in the parameters of our c-PLEB data structure we obtain $C'(n) = d^2 n^{1+\rho} \log n$, $Q'(n) = d^2 \log^2 n$ and $D'(n) = dn^\rho$. By applying the theorem of [Ove83], p. 104, we obtain a data structure with query time $O(d^2 \log^3 n)$, insertion time $O(d^2 n^\rho \log n)$ and deletion time $O(dn^\rho)$.

#	Paper	Query	Space	Update
1	[KOR98] (cf. [IM98])	$d \log n / \min(\epsilon^2, 1)$ $n^{O(\frac{1+\log(1+\epsilon)}{1+\epsilon})}$	$n^{O(1/\epsilon^2+\log(1+\epsilon)/(1+\epsilon))}$ $O(dn)$	$n^{O(1/\epsilon^2+\log(1+\epsilon)/(1+\epsilon)))}$ $\tilde{O}(d)$
2	[IM98]	$dn^{1/(1+\epsilon)}$	$n^{1+1/(1+\epsilon)} + dn$	$dn^{1/(1+\epsilon)}$
3	[Ind00a]	$(d \log n/\epsilon)^{O(1)}$	$n^{1/\epsilon^{O(1)}}$	static

Table 2.1: The algorithms for Hamming space. All algorithms are Monte Carlo, except for the algorithm in row #3 which is Las Vegas for any $\epsilon > 0$ and deterministic for $\epsilon > 2$.

2.2 Hamming space (Monte Carlo algorithms)

In this section we describe the algorithms for the Hamming space. We present two algorithms: one with very fast query time but huge space requirements (unless the approximation factor is large) and one with sublinear query time and moderate storage. The specific parameters of the algorithms are depicted on Figure 2.1. All of the algorithms are naturally dynamic, i.e., enable point insertions or deletions essentially without any modifications.

The two algorithms are randomized Monte Carlo, with constant probability of correctness; we describe the modifications needed to make them Las Vegas (or even deterministic) in Section 2.3. Since these data structures solve an *on-line problem*, the definition of correctness is somewhat nontrivial. Here we adopt the weakest definition, which is as follows: any operation performed by the algorithm is correct with a certain probability (say P), where the probabilistic space is defined over all configurations of random bits chosen by the algorithm until the operation has been requested. To illustrate this definition, consider a static data structure which uses random bits only during the preprocessing stage (all static randomized data structures in this Chapter are of this form). Then, whenever the data structure answers a query q, it can err with a probability C, but the probabilities for different queries are *not* independent, e.g., each time we ask the same query we get the same answer (although after performing a new preprocessing, the answers can change).

This definition is obviously somewhat unsatisfying, especially if the value of C is not close to 1. However, the probability of correctness can be amplified to $1-\delta$ for arbitrarily small $\delta > 0$ by using standard repetition techniques. Namely, we can build/maintain $k = O(\log 1/\delta/C)$ data structures, each using independent random bits; in order to answer a query we access all of them, reporting the one of the valid (i.e., within distance cr from q) points found by the structures, if any. Clearly, the probability of error is upper bounded by $(1-C)^k$ which is less than δ for proper k. Therefore, in the following we usually do not mention the exact value of correctness probability.

By setting $\delta = C'/2^d$ for $C' < 1$, we are making sure that the probability that the algorithm is incorrect for at least *one* of the 2^d possible queries is at most $\delta \cdot 2^d = C' < 1$[3]. Therefore, we are guaranteed that the algorithm is correct for any query, if the preprocessing phase was succesful (which happens with probability C'). Unfortunately, usually there is no easy way to verify the latter fact.

2.2.1 Fast queries, large storage

In this section we present a data structure for the $(1+\epsilon, r)$-PLEB problem using $n^{O(1/\epsilon^2 + \log(1+\epsilon)/(1+\epsilon))}$ space with query time $O(d \log n / \min(1, \epsilon^2))$. These "strange" bounds are due to the fact that the algorithm behaves quite differently when $\epsilon < 1$ than when $\epsilon \geq 1$ (although the transition is continuous); in particular, the storage overhead becomes *sublinear* in n for large enough ϵ. We present here a variant of the

[3]This reasoning works only in the setting when the database does not change.

data structure by Kushilevitz, Ostrovski and Rabani [KOR98], with certain modifications which allow us to prove the bounds for large ϵ. In Section 2.4 we will describe an alternative algorithm (due to Indyk and Motwani [IM98]) with slightly larger storage requirements of $n^{O(\log(1/\epsilon)/\epsilon^2+1)}$. The latter algorithm is more natural to describe for the l_2 norm rather than the Hamming space (although can be easily applied also in this context) and therefore is postponed to the end of the chapter.

The main tool of both algorithms is a *randomized dimensionality reduction*. We will first present this idea for the case when $\epsilon < 1$. Suppose that the dimensionality d is "low" (say $\log n$). In this case we can easily achieve $O(d)$ lookup time by hashing answers for all possible query points and perform a simple look-up to answer a query. Note that the data structure uses 2^d space, which by our assumption is "not-so-much" (say $n^{O(1)}$). Unfortunately, we are mainly concerned with scenarios in which d is fairly large. It turns out however, that one can always reduce the dimensionality from d to $d' = O(\log n/\epsilon^2)$ without changing the distances between the query point and the data set points by more than a factor of $(1+\epsilon)$. Formally, the following holds:

Lemma 4 *Let $d' = O(\log(1/\beta)/\epsilon^2)$, for $0 < \beta, \epsilon < 1$ and $r \in \{1 \dots d\}$. There exists an integer r' and a probability distribution over mappings $g : \{0,1\}^d \to \{0,1\}^{d'}$ such that for any pair of points p, q we have*

1. *If $D(p,q) \le r$ then $\Pr[D(g(p), g(q)) \le r'] \ge \frac{1}{2}$*
2. *If $D(p,q) > (1+\epsilon)r$ then $\Pr[D(g(p), g(q)) \le r'] \le \beta$*

Moreover, for each p we can compute $g(p)$ in time $O(dd')$.

Before we prove this lemma, we first show how it implies the algorithm working within the stated bounds. Set $\beta = \frac{1}{3n}$. In this way d' becomes $O(\log n/\epsilon^2)$. We map all the points to the d'-dimensional space and apply the $2^{d'}$-storage algorithm with radius r'; note that $2^{d'} = n^{O(\log n/\epsilon^2)}$. In order to answer the query q, we perform a look-up for $g(q)$. Clearly the query time is $O(d \log n/\epsilon^2)$. For the correctness, assume that there is a point p within distance r from q. Observe that:

- the probability that $D(g(p), g(q)) > r'$ is at most $\frac{1}{2}$

- the probability that for a fixed point p' such that $D(p', q) > (1+\epsilon)r$ we have $D(g(p'), g(q)) \leq r'$ is at most $\frac{1}{3n}$. Since there are less than n such points, the probability that such p' *exists* is at most $n \cdot \frac{1}{3n} = \frac{1}{3}$.

Therefore, the probability that either $D(g(p), g(q)) > r'$ or $D(g(p'), g(q)) \leq r'$ for any p' as above is at most $\frac{1}{2} + \frac{1}{3} = \frac{5}{6}$. Therefore, there is a constant probability that none of this events occur, in which case the algorithm correctly solves the $(1+\epsilon)$-PLEB problem.

Now we proceed with the proof of Lemma 4.

Proof: Let $k = \alpha d/r$. For this proof we will actually assume $\alpha = 1$, but the general case will become useful later. The function g is obtained by concatenating d' functions $g_1 \ldots g_{d'}$, such that each g_i maps a d-dimensional point into $\{0, 1\}$. Each g_i is parametrized by a multiset I_i and a function $f_i : \{0,1\}^{|I_i|} \rightarrow \{0, 1\}$. For any p the value of $g_i(p)$ is defined as

$$g_i(p) = f_i(p_{|I_i})$$

where $p_{|I}$ denotes the binary vector of length of length $|I|$ which contains the values of p for all coordinates in $i \in I$. In other words, $p_{|I}$ is a projection of p on coordinates from the set I.

We choose each g_i by choosing the k elements of I_i independently and uniformly at random from $\{1 \ldots d\}$ (note that the elements can repeat) and choosing f_i to be a completely random function. The latter can be done as follows. Clearly, if f is a random function, then the value of $f(p)$ for each p is a random variable independent from other variables $f(p')$, $p' \neq p$. Therefore, we can maintain random f by keeping the set S of arguments p on which f has been evaluated so far, together with the values of $f(p)$. When $f(p)$ is to be evaluated, we check if p belongs to S. If not, we choose $f(p)$ at random and store the pair $(p, f(p))$. Otherwise, we report the stored value of $f(p)$. Note that by using hashing, these operations can be performed in negligible time.

To prove the lemma, we start from estimating the probability $P(\delta)$ that for two points p, q such that $D(p, q) = \delta r$ we have $g_i(p) = g_i(q)$. Clearly, this happens either if $p_{|I_i} = q_{|I_i}$, or when $p_{|I_i} \neq q_{|I_i}$ but $f_i(p_{|I_i}) = f_i(q_{|I_i})$. The probability of the former event is equal to

$$Q(\delta) = (1 - r/d)^k = \left(1 - \frac{\alpha\delta}{\alpha\delta d/r}\right)^{\alpha\delta d/r}$$

We know that

$$e^{-\alpha\delta}\left(1 - \frac{(\alpha\delta)^2}{\alpha\delta d/r}\right) \leq \left(1 - \frac{\alpha\delta}{\alpha\delta d/r}\right)^{\alpha\delta d/r} \leq e^{-\alpha\delta}$$

In other words, $Q(\delta) \approx e^{-\alpha\delta}$ modulo the factor $1 - \frac{(\alpha\delta)^2}{\alpha\delta d/r} = 1 - \alpha\delta r/d$. To make $\alpha\delta r/d$ negligible (say smaller than η), we will simply assume that $\alpha\delta r/\eta < d$. This can be easily satisfied (conceptually) by padding all binary vectors with 0's to achieve a desired vector length. Then, while choosing elements of I_i's, we simply discard any coordinate which we "added" during the above process.

By setting $\alpha = 1$ we obtain that $Q(\delta) \geq (1-\eta)e^{-\delta}$. Since f_i is a random function, we get

$$1/2(1 + (1-\eta)e^{-\delta}) \leq P(\delta) \leq 1/2(1 + e^{-\delta})$$

We will set $r' = \lceil (1 - P(1))d' \rceil$. Notice, that the distance $D(g(p), g(q))$ is the sum of the individual components $|g_i(p) - g_i(q)|$, each of which is an independent binary variable with probability of success equal to $1 - P(1)$. Therefore, with probability at least 1/2 we have $D(g(p), g(q)) \leq (1 - P(1))d' = r'$, and so we are done with the first part.

To prove the second part, observe that $D(g(p'), g(q))$ is again a sum of random binary variables with probability of success equal to $1 - P(c)$, where $c = 1 + \epsilon$. By Chernoff bound

$$\Pr[D(g(p'), g(q)) \leq (1 - P(1))d'] \leq e^{-\gamma^2(1-P(c))d'/2}$$

where

$$\gamma = 1 - \frac{1 - P(1)}{1 - P(c)} \geq 1 - \frac{1/2(1 - (1-\eta)e^{-1})}{1/2(1 - e^{-c})} = 1 - \frac{1 - (1-\eta)e^{-1}}{1 - e^{-c}}$$

Observe that by Taylor expansion $e^{-c} \leq e^{-1}(1 - \epsilon + \epsilon^2/2)$. Therefore

$$\gamma \geq 1 - \frac{1 - (1-\eta)e^{-1}}{1 - e^{-1}(1 - \epsilon + \epsilon^2/2)}$$

If we set $\eta = (\epsilon - \epsilon^2/2)/2$, then

$$\gamma \geq 1 - \frac{1-(1-\eta)e^{-1}}{1-(1-2\eta)e^{-1}} = 1 - \frac{e-1+\eta}{e-1+2\eta} = \frac{\eta}{e-1+2\eta} \geq \eta/4$$

for $\eta < 1$. Thus setting $d' = O(\frac{2}{\gamma^2(1-P(c))} \ln 1/\beta) = O(\ln(1/\beta)/\epsilon^2)$ results in

$$\Pr[D(g(p'), g(q)) \leq r'] \leq \beta.$$

■

This completes the case when $\epsilon < 1$. Unfortunately, the algorithm as described above does not achieve a sublinear storage overhead for large ϵ, due to the fact that $d' = \Theta(\log n)$ even for large ϵ. To achieve a sublinear behavior for $\epsilon \geq 1$ we modify the algorithms slightly: instead of storing answers for *all queries*, we only store the answers for queries with *positive* answer (i.e. for which there exists a close database point). It is easy to see that there are at most $\binom{d'}{r'}$ such queries. To minimize this number we set $k = \frac{d}{cr}$ and keep $r' = (1-P(1))d'$. In this way the new value of $Q(\delta)$ is bounded by

$$e^{-\delta/c} \geq Q(\delta) \geq (1-\eta)e^{-\delta/c}$$

Therefore, $Q(c)$ is a constant close to e^{-1}, which by previous arguments implies $d' = O(\log n)$ is enough to make sure points p' for which $D(p', q) \geq cr$ are not mapped within distance r' from $f(q)$ (note that $c \geq 2$, so the ratio $Q(1)/Q(c)$ is lower bounded by a constant greater than 1). It is sufficient now to estimate the value of r'. Observe that

$$1 - P(1) \leq 1/2(1 - e^{-1/c}(1-\eta)) \leq 1/2(1-(1-1/c)(1-\eta)) = 1/2(\eta + (1-\eta)/c)$$

If we set $\eta = 1/c$ we upper bound $1 - P(1)$ by $1/c$. In this case we have

$$\binom{d'}{r'} \leq \binom{d'}{d'/c} \leq (ec)^{d'/c} \leq e^{O(\log c/c)}$$

This concludes the asymptotic analysis of the algorithm. However, there is no evidence that our choices of the parameters of the algorithm (specifically, the value of k) are optimal. To provide a better understanding of the actual behavior of the

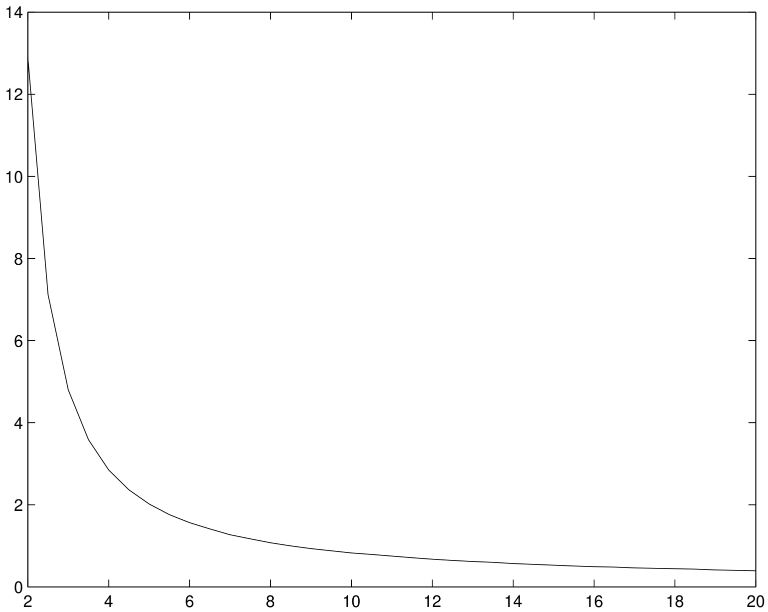

Figure 2.1: The exponent of the storage overhead as a function of the error $c \in [2, 20]$.

algorithm, we estimated the exponent in the storage overhead by performing exhaustive search over the space of parameters for $c = 2 \ldots 20$. Specifically, for each $c \in \{2, 2.5, 3, \ldots 20\}$ we computed the exponent for the algorithm with parameter $k = \alpha d/r$, for $\alpha = 0.1, 0.2, 0.3, \ldots, 100$ and chose the smallest one. The "optimal" values of exponents for each c are presented on Figure 2.1.

Finally, we observe that we can "push" the work done by the algorithm from the preprocessing to the query time. In other words, instead of storing all points close to p, we can just store p's only and search for p''s close to the query q during the query processing. This results in an algorithm with linear space and $n^{O(\log(1+c)/c)}$ query time; the values of the exponent for small c can be again read from Figure 2.1. Unfortunately, the exponent goes below 1 only for fairly large c (around 9), thus for small c the algorithm is useless. This can be improved by using the following idea[4]. Suppose that instead of setting the probability β to $O(1/n)$, we keep it slightly larger. In this case it can happen that some points p' which are far from q are mapped to a point within the distance of r' from $g(q)$. However, their expected number is at most

[4]This idea has been suggested to the author by an anonymous reviewer.

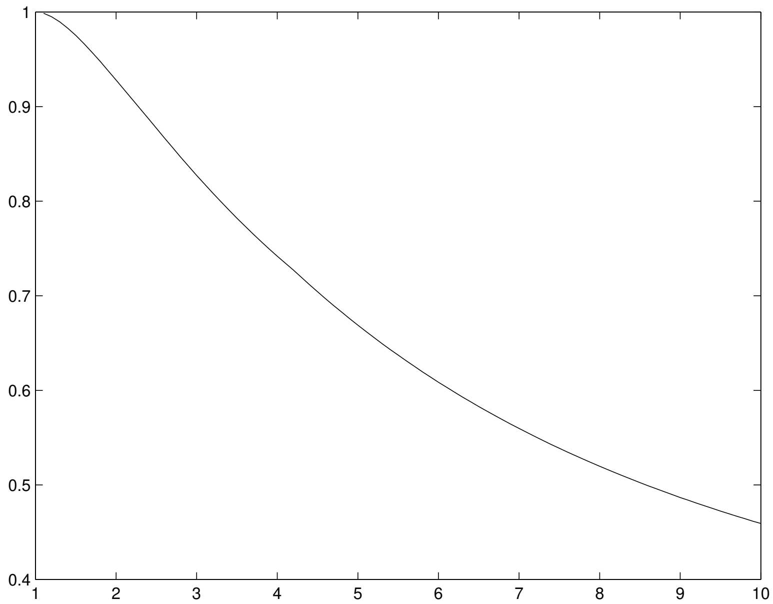

Figure 2.2: The exponent in the search time of the modifed algorithm.

$n\beta$, so if β is not too large, we can just check and discard them all. The benefit of this approach is that smaller $1/\beta$ results is smaller d' and therefore the cost of searching the neighborhood of $g(q)$ is decreased. Specifically, the expected cost of the search now becomes bounded by $n\beta + \binom{d'}{r'}$. By (computationally) optimizing the parameters we obtain a graph as on Figure 2.2

Theorem 3 *There is a Monte Carlo algorithm for c-PLEB in Hamming space with parameters as in row #1 of Figure 2.1.*

2.2.2 Sublinear query/update

In this section we describe the second algorithm, which achieves sublinear bound for both query and update time. The algorithm is based on the concept of the *locality-sensitive hashing (LSH)*, introduced in [IM98]. The basic idea here is to hash the data and query points using hash functions such that the probability of collision is much higher for points which are close to each other than for those which are far apart. Formally, we require the following.

Definition 2 *A family $\mathcal{H} = \{h : S \to U\}$ is called (r_1, r_2, p_1, p_2)-*sensitive *for D if for any $q, p \in S$*

- *if $p \in B(q, r_1)$ then $\Pr_{\mathcal{H}}[h(q) = h(p)] \geq p_1$,*
- *if $p \notin B(q, r_2)$ then $\Pr_{\mathcal{H}}[h(q) = h(p)] \leq p_2$.*

In order for a locality-sensitive family to be useful, it has to satisfy inequalities $p_1 > p_2$ and $r_1 < r_2$.

Now we can describe the algorithm for (r, c)-PLEB as follows. We choose $r_1 = r$ and $r_2 = c \cdot r$. Given a family $\mathcal{H}$ of such hash functions for these parameters, we amplify the gap between the "high" probability p_1 and "low" probability p_2 by concatenating several functions. In particular, for k specified later, define a function family $\mathcal{G} = \{g : S \to U^k\}$ such that $g(p) = (h_1(p), \ldots, h_k(p))$, where $h_i \in \mathcal{H}$. For an integer L we choose L functions $g_1, \ldots, g_L$ from $\mathcal{G}$ independently and uniformly at random. During preprocessing, we store each $p \in P$ in the bucket $g_j(p)$, for $j = 1, \ldots, L$. Since the total number of buckets may be large, we retain only the non-empty buckets by resorting to hashing [FKS84, MR95]. To process a query q, we search all buckets $g_1(q), \ldots, g_L(q)$; as it is possible (though unlikely) that the total number of points stored in those bucket is large, we interrupt search after finding first $3L$ points (including duplicates). Let $p_1, \ldots, p_t$ be the points encountered therein. For each p_j, if $p_j \in B(q, r_2)$ then we return YES and p_j, else we return NO.

The parameters k and L are chosen so as to ensure that with a constant probability the following two properties hold:

1. If there exists $p^* \in B(q, r_1)$ then $g_j(p^*) = g_j(q)$ for some $j = 1 \ldots L$, and
2. The total number of collisions of q with points from $P - B(q, r_2)$ is less than $3L$, i.e.
$$\sum_{j=1}^{L} |(P - B(q, r_2)) \cap g_j^{-1}(g_j(q))| < 3L.$$

Observe that if (1) and (2) hold, then the algorithm is correct.

Theorem 4 *Suppose there is a (r_1, r_2, p_1, p_2)-sensitive family $\mathcal{H}$ for D. Then there exists an algorithm for (r, c)-PLEB under measure D which uses $O(dn + n^{1+\rho})$ space and $O(n^\rho)$ evaluations of the hash function for each query, where $\rho = \frac{\ln 1/p_1}{\ln 1/p_2}$.*

Proof: It suffices to ensure that (1) holds with probability P_1 and (2) holds with probability P_2 such that both P_1 and P_2 are strictly greater than half. Let $p^* \in B(q, r_1)$. Set $k = \log_{1/p_2} n$, then the probability that $g(p') = g(q)$ for $p \in P - B(q, r_2)$ is at most $p_2^k = \frac{1}{n}$. Thus the expected number of elements from $P - B(q, r_2)$ colliding with q under fixed g_j is at most 1; the expected number of such collisions with *any* g_j is at most l. Thus by Markov inequality the probability that this number exceeds $3l$ is less than $1/3$; therefore the probability that the property (2) holds is $P_2 \geq 2/3$.

Consider now the probability of $g_j(p^*) = g_j(q)$. Clearly, it is bounded from below by

$$p_1^k = p_1^{\log_{1/p_2} n} = n^{-\frac{\log 1/p_1}{\log 1/p_2}} = n^{-\rho}.$$

Thus the probability that such a g_j exists is at least $P_1 = 1 - (1 - n^{-\rho})^l$. By setting $L = n^\rho$ we get $P_1 > 1 - 1/e > 1/2$. The theorem follows. ■

In order to use Theorem 4 for the Hamming metric, we need to specify the proper family of hash functions. To this end we exploit the family of all projections of the bit vector on one of its coordinates. Such a family has ben used earlier in [AMRT96] to perform fast hashing with AC^0 operations. Here we note that this family is good even for Hamming space over non-binary alphabet, i.e., say, over Σ^d where Σ is arbitrary alphabet.

Proposition 1 ([AMRT96]) *Let $D(p, q)$ be the Hamming metric for $p, q \in \Sigma^d$. Then for any $r, \epsilon > 0$, the family $\mathcal{H} = \{h_i : h_i((b_1, \ldots b_d)) = b_i,\ i = 1 \ldots n\}$ is $\left(r, r(1+\epsilon), 1 - \frac{r}{d}, 1 - \frac{r(1+\epsilon)}{d}\right)$-sensitive.*

Corollary 2 *For any $\epsilon > 0$, there exists an algorithm for ϵ-PLEB in Hamming metric over Σ^d using $O(dn + n^{1+1/(1+\epsilon)})$ space and $O(n^{1/(1+\epsilon)})$ hash function evaluations for each query. The hash function can be evaluated using $O(d)$ operations.*

Proof: We use Proposition 1 and Theorem 4. First, we need to estimate the value of $\rho = \frac{\ln 1/p_1}{\ln 1/p_2}$, where $p_1 = 1 - \frac{r}{d}$ and $p_2 = 1 - \frac{r(1+\epsilon)}{d}$. Without loss of generality, we assume that $r < \frac{d}{\ln n}$, since we can increase dimensionality by adding a sufficiently long string of 0s at the end of each point, as mentioned earlier. Observe that

$$\rho = \frac{\ln 1/p_1}{\ln 1/p_2} < \frac{\ln \frac{1}{1-r/d}}{\ln \frac{1}{1-(1+\epsilon)r/d}} = \frac{\ln(1-r/d)}{\ln(1-(1+\epsilon)r/d)}$$

Multiplying both the numerator and the denominator by $\frac{d}{r}$ we obtain that:

$$\rho = \frac{\frac{d}{r}\ln(1-r/d)}{\frac{d}{r}\ln(1-(1+\epsilon)r/d)} = \frac{\ln(1-r/d)^{d/r}}{\ln(1-(1+\epsilon)r/d)^{d/r}} = \frac{N}{D}.$$

In order to upper bound ρ, we need to bound N from below and D from above; note that both N and D are negative. To this end we use the following inequalities:

$$(1-(1+\epsilon)r/d)^{d/r} < e^{-(1+\epsilon)} \quad \text{and} \quad (1-r/d)^{d/r} > e^{-1}(1-\frac{1}{d/r}).$$

Therefore,

$$\begin{aligned}
\frac{N}{D} &< \frac{\ln(e^{-1}(1-\frac{1}{d/r}))}{\ln e^{-(1+\epsilon)}} \\
&= \frac{-1+\ln(1-\frac{1}{d/r})}{-(1+\epsilon)} \\
&= 1/(1+\epsilon) - \frac{\ln(1-\frac{1}{d/r})}{1+\epsilon} \\
&< 1/(1+\epsilon) - \ln(1-1/\ln n)
\end{aligned}$$

where the last step uses the assumptions that $\epsilon > 0$ and $r < \frac{d}{\ln n}$. We conclude that

$$n^{\rho} < n^{1/(1+\epsilon)} n^{-\ln(1-1/\ln n)} = n^{1/(1+\epsilon)}(1-1/\ln n)^{-\ln n} = O(n^{1/(1+\epsilon)}).$$

■

We mention that the hash function evaluation can be made faster than $O(d)$ for sparse data, i.e., when the number of non-zero coordinates of a query point is small. It suffices to sample the bits from the non-zero entries of the vectors.

2.3 Hamming space (Las Vegas and deterministic algorithms)

In this section we describe how to modify the algorithms from the previous sections to make them randomized Las Vegas or even completely deterministic. We will start from the algorithm described in Section 2.2.1 and show how to modify it to get an algorithm with $(d \log n/\epsilon)^{O(1)}$ query time using $n^{O(1/\epsilon^3)}$ space, with the following guarantee:

- given a query q, the algorithm returns either a point or a special character "?"
- for a fixed q, the probability (with respect to the random bits used during the preprocessing) of returning "?" is at most $1/n^c$, for any fixed constant c
- whenever the algorithm returns a point, it is *always* a correct $(1+\epsilon)$-approximate answer

In short, whenever we return an answer (which happens with high probability), it is guaranteed to be correct. Then we show how to make this algorithm deterministic; however the approximation factor will increase to $3 + \epsilon$.

2.3.1 Fast queries, large storage

In this section we describe Las Vegas (and then deterministic) algorithm for the $(r, 1+\epsilon)$-PLEB problem in Hamming space with $(d \log n/\epsilon)^{O(1)}$ query time and polynomial time preprocessing, using $n^{O(1/\epsilon^3)}$ space. To make things simpler, we will in fact show how to solve (r, c)-PLEB, for $c = 1 + O(\epsilon)$.

Let $R = cr$. Our solution consists of the following steps.

1. Map the space $\{0,1\}^d$ into $\Sigma^{O(R)}$ for some (large) alphabet Σ using perfect hash functions
2. Encode each symbol from Σ using a binary error-correcting code (with codeword length $O(\log n)$) obtaining a mapping into $\{0,1\}^{O(R \log n)}$

3. Divide the (binary) coordinates into groups of size $O(\log n)$ and solve the Nearest Neighbor problem within each group. During the query processing return the best (closest to the query) answer among all answers obtained from the subproblems.

We mention that one can think about the above steps as a somewhat different implementation of the approach from Section 2.2.1. Recall, that in that section we mapped each point p to the vector $(f_1(p_{|I_1}), \ldots, f_{d'}(p_{|I_{d'}}))$, for $d' = O(\log n)$. Here, the projection of p on I_j is performed in Step 1 while the mapping of $p_{|I_i}$ to $\{0,1\}$ is done here using the error-correcting codes in Step 2. The Step 3 is present in the Monte Carlo algorithm in a somewhat trivial form: since in Section 2.2.1 we generated only one group of coordinates, we did not have to make a choice how to split the coordinates into different groups.

Below we describe each step in detail.

Hashing. The goal of hashing is to obtain a mapping $\pi : \{0,1\}^d \to \Sigma^D$, such that

- π is *non-expansive*, i.e., for any $x, y \in \{0,1\}^d$ we have $D(\pi(x), \pi(y)) \leq SD(x, y)$, where S is a scaling factor
- π is (ϵ, R)-*contractive*, i.e., for any $x, y \in \{0,1\}^d$ such that $D(x, y) \geq R$ we have $D(\pi(x), \pi(y)) \geq SR(1 - \epsilon)$

Moreover, the value of D should be close to SR.

The mapping is obtained as follows. Let $\mathcal{H}$ be a family of hash functions $h : [d] \to [v]$ such that for any $x, y \in [d]$ we have

$$\Pr[h(x) = h(y)] \leq \frac{1}{R/\epsilon}$$

We can assume that $Q = R/\epsilon$ is a prime number (as otherwise we can always decrease ϵ by a constant factor so that this fact becomes true). In this case a family of functions $\mathcal{H}_Q$ consisting of functions $h(x) = ax \bmod Q$, $a \in [Q]$, is known to satisfy the above condition. Having $\mathcal{H}$ we compute $\pi(x)$ as follows. For each $h \in \mathcal{H}$ we construct a mapping π_h which for any $x \in \{0,1\}^d$:

- maps each bit x_i into "bucket" $h(i)$

- sorts all bits x_i within each bucket in ascending order of i's
- concatenates all bits within each bucket into one symbol (empty buckets are represented by an additional special symbol) and outputs the resulting Q-dimensional vector

The value of π is now defined as concatenation of all $\pi_h(x)$ for $h \in \mathcal{H}_Q$.

If we set the scaling factor S to $|\mathcal{H}| = Q$, then it is easy to see that the mapping π is both non-expansive and ϵ-contractive. The first property follows from the fact that each difference bit between x and y creates at most S difference symbols between $\pi(x)$ and $\pi(y)$. On the other hand, the low probability of collision guaranteed by $\mathcal{H}$ implies the second property.

Coding. In this stage each element a from Σ (constructed implicitly by mapping the whole pointset using π) is replaced by a binary word $C(a)$ of length λ. The words have the property that for each pair of different symbols $a, b \in \Sigma$ we have $D(C(a), C(b)) \in [(1/2(1-\epsilon))\lambda, 1/2 \cdot \lambda]$. From [NN93] we know that there exists (efficiently constructible) code with $\lambda = O(\log |\Sigma|(1/\epsilon)^\eta)$ having this property, for a constant $\eta \leq 5$. Since the size of Σ is bounded by $n\text{poly}(d, 1/\epsilon)$, it follows that $\lambda = O((\log n + \log d + \log 1/\epsilon)/\epsilon^\eta)$. By using π and C, the resulting mapping (say g) is clearly non-expansive and $(1-(1-\epsilon)^2)$-contractive, with scaling factor $S' = Q \cdot \lambda/2$. Notice that the dimension D' of the range of g is $P^2 \cdot \lambda$. In particular it implies that for a pair x, y s.t. $D(x, y) \geq R$ we have $D(g(x), g(y)) \geq R' = S'(1-\epsilon)^2 R$, which is $\Theta(D')$.

Divide and conquer. During this stage we partition the set $[D']$ of coordinates of g's range into sets $S_1 \dots S_k$ of size $s = O(\log n)$. We would like this partition to have the property that the function $g_{|S_i}$ is non-expansive and β-contractive for some small β, at least for one S_i. We achieve it with high probability as follows. Set $s = \frac{D'}{R'} t$ for $t = A/\epsilon^2 \log n$. In this case, for sufficient constant A, a random partition $S_1 \dots S_k$ can be easily seen to guarantee that for any fixed query q and $p' \in P$ such that $D(q, p') \geq R$ we have $D(g(q)_{|S_i}, g(p')_{|S_i}) \geq t(1-\epsilon)$ for all $i = 1, \dots, k$ with high probability. At the same time, by pigeonhole principle, for any $p \in P$ such that

$D(q,p) \leq r$ there exists at least one i such that

$$\begin{aligned} D(g(q)_{|S_i}, g(p)_{|S_i}) &\leq rS'/k \\ &= R/c \cdot S' \cdot s/D' \\ &= R/c \cdot S' \cdot t/R' \\ &= t \cdot R/c \cdot S' \cdot 1/(S'(1-\epsilon)^2 R) \\ &= \frac{t}{c(1-\epsilon)^2} \end{aligned}$$

Thus, for (say) $c = 1 + 4\epsilon$ we have (with high probability) a gap between one $D(g(q)_{|S_i}, g(p)_{|S_i})$ and all $D(g(q)_{|S_i}, g(p')_{|S_i})$ and thus projecting all points on S_i and returning the closest point gives a c-approximate answer. On the other hand, it can happen that $D(g(q)_{|S_i}, g(p')_{|S_i})$ is much smaller than $t(1-\epsilon)$, in which case we can return p' instead of p, which is not correct. However, we can test if such situation has occurred simply by testing if $D(g(q)_{|S_i}, g(p')_{|S_i}) \geq t(1-\epsilon)$ for all $i = 1 \ldots k$. If this condition is violated (which happens with low probability), we return "?"; otherwise we can safely claim that there is no point within distance r from q.

Deterministic algorithm. In order to obtain fully deterministic algorithm, we only need to modify Step 3, since other steps are already deterministic. Unfortunately, it is not difficult to see that it is impossible to obtain a fixed partition $S_1 \ldots S_k$ such that $D(g(q)_{|S_i}, g(p')_{|S_i}) \geq t(1-\epsilon)$ if $D(q,p') \geq R$ for all $q \in \{0,1\}^{D'}$, simply because the latter space of queries is too big. Therefore, we will only preserve this property for all pairs q, p' of points from the pointset P and show that this is sufficient. Actually, for technical reasons, we will need to preserve a more general property, namely $D(g(q)_{|S_i}, g(p')_{|S_i}) \geq t(1-\epsilon)D(q,p')/R$ if $R \leq D(q,p') \leq 4R$. If S_i satisfies this property for all $p', q \in P$, we say S_i is "almost non-contractive".

By the earlier argument we know that a random partition is almost non-contractive for all S_i's with high probability, which in particular implies that a good partition exists. Since we need to construct such partition deterministically, we derandomize the random choice of S_i's using the method of conditional probabilities [AS00]. Assume that the randomized algorithm first chooses s elements of S_1, then s elements for S_2 and so on. For each pair $x, y \in P$ let $E_{x,y}$ denote the event

"$D(g(x)_{|S_i}, g(y)_{|S_i}) < t(1-\epsilon)$". It is not difficult to see that the probability of $E_{x,y}$ conditioned on a partial choice of a partition has hypergeometric distribution, and therefore is easily computable. This implies that the method of conditional probabilities yields polynomial time deterministic algorithm for constructing the required partition.

Having the partition, we proceed as follows. Let $P_i = g(P)_{|S_i}$; notice that P_i contains points from $\{0,1\}^s$. For each i we build a data structure D_i solving a nearest neighbor problem for P_i. This is an easy task, since the dimensionality of the space is $s = O(\log n)$, and therefore we can precompute answers to all 2^s possible queries using polynomial (in n) space. The solution to the PLEB problem is now computed by querying all D_i's and checking if the closest point found is close enough to the query point.

It remains to be shown that the algorithm returns a $3+O(\epsilon)$-approximate answer. This follows from the following Lemma.

Lemma 5 *Let F be a mapping from metric $M_1 = (X_1, D_1)$ into $M_2 = (X_2, D_2)$ such that for $q, p, p' \in X$ we have*

1. $D_2(F(p), F(q')) \geq (1-\alpha) D_1(p, p')$ *(i.e., F is almost non-contractive on p, p')*
2. $D_2(F(p), F(q)) > D_2(F(p'), F(q))$ *(i.e., after the mapping, p' is closer to q than p is)*
3. $D_2(F(p), F(q)) \leq D_1(p, q)$ *(i.e., F is non-expansive on p, q)*

Then $D_1(p', q) \leq (\frac{2}{1-\alpha} + 1) D_1(p, q)$ (i.e., p' is an approximate nearest neighbor of q)

Proof: By triangle inequality, $D_1(p, p')$ is at least as big as $D_1(p', q) - D_1(p, q)$. Therefore (by assumption 1) we have $D_2(F(p), F(p')) \geq (1-\alpha)(D_1(p', q) - D_1(p, q))$. However, $D_2(F(p), F(p')) \leq D_2(F(p'), F(q)) + D_2(F(q), F(p))$. By assumptions 2 and 3 we know that $D_2(F(p), F(q)) \leq D_1(p, q)$ and $D_2(F(p'), F(q)) \leq D_2(F(p), F(q)) \leq D_1(p, q)$, thus $D_2(F(p), F(p')) \leq 2D_1(p, q)$. Therefore, we have

$$(1-\alpha)(D_1(p', q) - D_1(p, q)) \leq D_2(F(p), F(p')) \leq 2D_1(p, q)$$

which proves the lemma. ∎

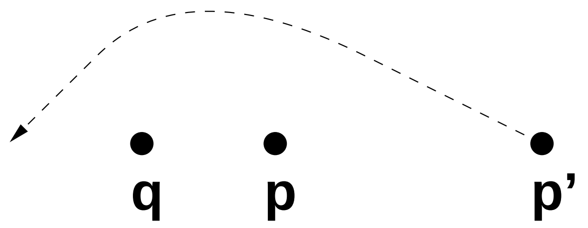

Figure 2.3: Scenario in which the factor 3 is tight.

We mention that the approximation factor in the above lemma is tight. To see this, assume (for the simplicity) that $\alpha = 0$ and take $M = M'$ to the metric induced by l_1^1. Set $q = 0$, $p = 1$, $p' = 3 - \delta$ for arbitrary $\delta > 0$. Let F be a mapping such that $F(q) = q$, $F(p) = p$ and $F(p') = -1 + \delta$. It is easy to see that F satisfies the assumptions of the lemma, while $|p' - q| = 3 - \delta = 3 - \delta|p - q|$. See Figure 2.3 for more details.

Theorem 5 *There is an algorithm for c-PLEB in Hamming space with parameters as in row #3 of Figure 2.1, with preprocessing time polynomial in n, d and $1/\epsilon$.*

2.4 Arbitrary norm

In this section we show how to solve the c-PLEB problem for any norm l_p, for any $c = 1 + \epsilon > 1$, $\epsilon < 1$. The query time is very fast, i.e., $O(d)$; unfortunately, the algorithm uses $O(1/\epsilon)^d n$ space, which is exponential in d. However, in Section 4.1 we show that for $p = 2$ one can reduce d to $O(\log n/\epsilon^2)$ and thus make the space bound polynomial in n. Thus the algorithm provides an alternative to the algorithm introduced in Section 2.2.1.

Assume for now that $p = 2$. Impose a uniform grid of spacing $s = \epsilon/\sqrt{d}$ on $\Re^d$. Clearly, the distance between any two points belonging to one grid cuboid is at most ϵ. For each ball B_i, define $\overline{B}_i$ to be the set of grid cells intersecting B_i. Store all elements from $\cup_i \overline{B}_i$ in a hash table, together with the information about the corresponding ball(s). After preprocessing, to answer a query q it suffices to compute the cell which contains q and check if it is stored in the table.

We claim that for $0 < \epsilon < 1$, $|\overline{B}| = O(1/\epsilon)^d$. To see this, observe that $|\overline{B}|$ is bounded by the volume of a d-dimensional ball of radius $r = 2/\epsilon\sqrt{d}$, which by Fact 1 is $2^{O(d)} r^d / d^{d/2} \leq O(1/\epsilon)^d$. Hence, the total space required is $O(n) \times O(1/\epsilon)^d$. The query time is the time to compute the hash function. For general l_p norms, we modify s to $\epsilon/d^{1/p}$. The bound on $|\overline{B}|$ applies unchanged.

Theorem 6 *For $0 < \epsilon < 1$, there is an algorithm for $(1+\epsilon)$-PLEB in l_p^d using $O(n) \times O(1/\epsilon)^d$ preprocessing and $O(d)$ query time.*

Chapter 3

Reductions

In this chapter we show how to use the basic c-PLEB algorithms from the previous chapter as building blocks for approximate algorithms solving other problems, such as nearest neighbor, various variants of closest pair, furthest neighbor, minimum spanning tree etc. In other words, we construct algorithms for the above problems using the algorithms for c-PLEB as black-boxes. Almost all reductions work for *any* metric space, so we can apply them for either l_∞ norm or Hamming space (or other norms considered in Chapter 4). The only exception is the furthest neighbor reduction, which works only for the Euclidean or Hamming space. Note, however, that this problem is trivial for the l_∞ norm, so the Euclidean norm is the only interesting case anyway.

The scheme of reductions is shown in Figure 3.1. It is somewhat more detailed than the analogous figure in the Introduction. Firstly, the reductions which work in the *amortized* sense are depicted using dashed lines. Secondly, we distinguish static problems (with name starting with "S.") from dynamic ones (with name starting with "D."). The type of problems is important in order to describe the complexity of reductions. Since any reduction is of the form "Problem A" $\rightarrow$ "Problem B", we have four different types of reductions, depending of the types of Problems A and B. The reduction from dynamic closest pair to dynamic nearest neighbor was proved by Eppstein [Epp95]. We mention that a few other (although less efficient) reductions have been given in [KOR98, BOR99b].

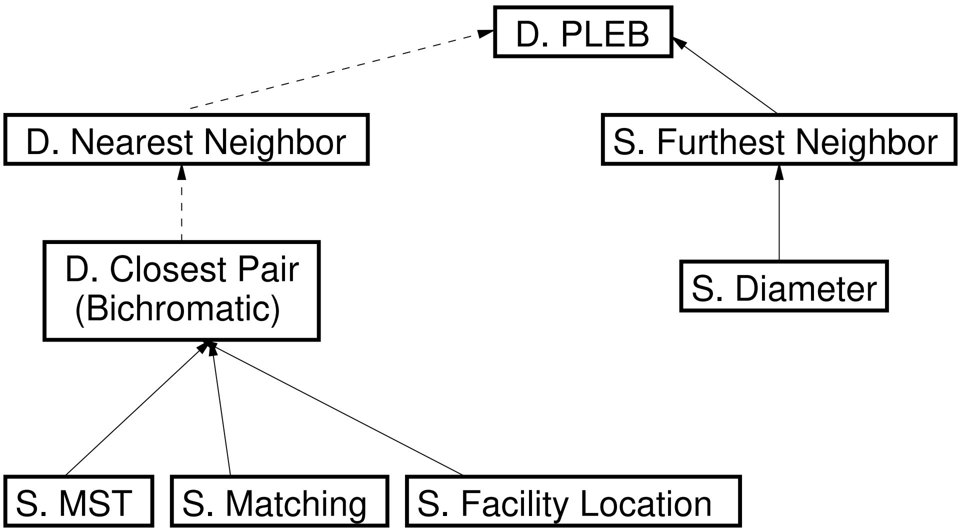

Figure 3.1: The reduction tree.

The complexity of reductions is as follows:

- dynamic $\rightarrow$ dynamic: the cost of any operation on the Problem A data structure is bounded by the cost of all operations on Problem B data structure times $\text{poly}(\log n, 1/\epsilon)$.

- static $\rightarrow$ dynamic: the total cost of the algorithm for Problem A is bounded by the cost of $n \cdot \text{poly}(\log n, 1/\epsilon)$ operations on the data structure for Problem B

- static $\rightarrow$ static: the total cost of the algorithm for Problem A is bounded by the cost of an algorithm for Problem B times $\text{poly}(\log n, 1/\epsilon)$

- dynamic $\rightarrow$ static: does not occur

We also need to specify how do the reductions preserve the accuracy. If we have a reduction from Problem A with accuracy $c' = 1 + \epsilon'$ to Problem B with accuracy $c = 1 + \epsilon$, then depending on the Problem A the we have the following cases:

- for facility location, $c' \leq 3 \cdot c^3$. This is due to the fact that our algorithm have been obtained by representing an existing algorithm of [JV99] as a sequence of c-PLEB operations. That algorithm already achieves a constant factor approximation, so our reduction suffers from the same problem as well. Note that in fact getting arbitrary $c' > 1$ for facility location is very unlikely, since this problem is hard to approximate below certain constant $B > 1$.

- for bottleneck matching, $c' \leq 2c$.
- for nearest neighbor and furthest neighbor problems, $\epsilon' \leq \epsilon + \delta$, where $\delta > 0$ can be made arbitrarily small; as long as δ remains a constant, the asymptotic running time of the resulting algorithm does not change.
- for other problems, $\epsilon' = \epsilon$.

The detailed bounds for the running times and accuracy of the algorithms are given in the respective sections.

3.1 Nearest neighbor

In this section we show, for any $\delta > 0$ and $c > 1$, a reduction from $(c + \delta)$-NNS to c-PLEB, with only a small (i.e., polylogarithmic in n, $1/\epsilon$ and $1/\delta$) overhead in preprocessing and query costs. We start from the static algorithm (first given in [IM98]) and then show how to make the algorithm dynamic.

This static algorithm relies on a data structure called a *ring-cover tree*. This structure exploits the fact that for any point set P, we can either find a *ring-separator* or a *cover*. Either construct allows us to decompose P into smaller sets $S_1, \ldots, S_l$ such that for all i, $|S_i| \leq c|P|$ for some $c < 1$, and $\sum_i |S_i| \leq b|P|$ for $b < 1 + 1/\log^2 n$. This decomposition has the property that while searching P it is possible to quickly restrict the search to one of the sets S_i.

Before we describe the tree data structure, we mention that there is a simpler but much weaker reduction from c-NN to c-PLEB. Let R be the ratio of the smallest and the largest inter-point distances in P. For each $r \in \{1 + \delta)^0, (1 + \delta)^1, \ldots, R\}$, generate a sequence of balls $B^r = \{B_1^r, \ldots, B_n^r\}$ of radius r centered at $p_1, \ldots, p_n$. Each sequence B^r forms an instance of PLEB. Then, given query q, we find via binary search the minimal r for which there exists an i such that $q \in B_i^r$ and return p_i as an approximate nearest neighbor. The overall reduction parameters are: query time overhead factor $O(\log\log_{1+\delta} R)$ and space overhead factor $O(\log_{1+\delta} R)$. The simplicity of this reduction is potentially very useful in practice. On the other hand, the $O(\log_{1+\delta} R)$ space overhead is unacceptable when R is large; in general, R may be unbounded.

We start the description of the ring-cover tree from a few definitions.

Definition 3 *A ring $R(p, r_1, r_2)$ is an $(\alpha_1, \alpha_2, \beta)$-**ring separator** for P if $|P \cap B(p, r_1)| \geq \alpha_1 |P|$ and $|P \setminus B(p, r_2)| \geq \alpha_2 |P|$, where $r_2/r_1 = \beta > 1$, $\alpha_1, \alpha_2 > 0$.*

Intuitively, a ring separator allows us to separate the points inside it from the points outside. For our purpose the separator is best if α_1, α_2 and β are large. An example pointset with a "nice" separator is shown in Figure 3.2. Note that in the extreme case $\alpha_1 = \alpha_2 = 1/2$ the middle part of the ring is empty.

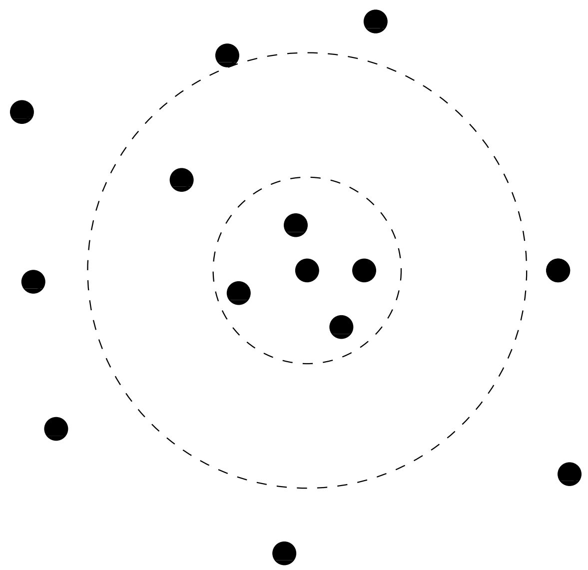

Figure 3.2: Ring separator.

Definition 4 *A set $S \subset P$ is a (γ, δ)-***independent set** *for P if for every $p \in S$, $|P \cap B(p, \gamma\Delta(S))| \leq \delta|P|$, where $0 \leq \gamma, \delta \leq 1$.*

In other words, a set of points forms an independent set if for every point p from the set a ball of reasonably large radius around p contains only a small fraction of points. In the extreme case $\delta = 0$ one can observe that the set of points forms an "error-correcting code", i.e., the distance between any two points is large.

In the following we show that *any* set of points in a metric space either has a "nice" ring separator or contains a "large" independent set. Later we show that both cases are algorithmically tractable.

Theorem 7 *For any P, $0 < \alpha < 1$, and $\beta > 1$, one of the following two properties must hold:*

1. *P has an (α, α, β)-ring separator, or*
2. *P contains a $(\frac{1}{2\beta}, \alpha)$-independent set of size at least $(1 - 2\alpha)|P|$.*

Proof: First note that for $\alpha > 1/2$, property (1) must be false but then property (2) is trivially true. In general, assume that (1) does not hold. Then, for any point p and radius r define:

- $f_p^\infty(r) = |P - B(p, \beta r)|$,
- $f_p^0(r) = |P \cap B(p, r)|$.

Clearly, $f_p^\infty(0) = n$, $f_p^\infty(\infty) = 0$, $f_p^0(0) = 0$, and $f_p^0(\infty) = n$. Also, notice that $f_p^\infty(r)$ is monotonically decreasing and $f_p^0(r)$ is monotonically increasing. Moreover, by standard perturbation argument, we can assume that that both function "jump" at any point only by 1, and that the jumps for $f_p^\infty(r)$ and $f_p^0(r)$ occur at different times. It follows that there must exist a choice of r (say r_p) such that $f_p^\infty(r_p) = f_p^0(r_p)$. Since (1) does not hold, for any value of r we must have $\min(f_p^\infty(r), f_p^0(r)) \leq \alpha n$, which implies that $f_p^\infty(r_p) = f_p^0(r_p) \leq \alpha n$.

Let q be a point such that r_q is minimal. Define $S = P \cap R(q, r_q, \beta r_q)$; it follows that $|S| \geq (1 - 2\alpha)n$. Also, notice that for any $s, s' \in S$, $D(s, s') \leq 2\beta r_q$, implying that $\Delta(S) \leq 2\beta r_q$. Finally, for any $s \in S$, $|P \cap B(s, r_q)| \leq |P \cap B(s, r_s)| \leq \alpha n$. ■

As we mentioned before, our algorithms makes use of either a ring separator or an independent set. In order to exploit the latter structure, however, we will need to show that its existence implies existence of another structure, which we call a *cover*. The formal definition of the cover is as follows.

Definition 5 *A sequence $A_1, \ldots, A_l$ of sets $A_i \subset P$ is called a (b, c, d)-**cover** for $S \subset P$, if there exists $r \geq d\Delta(A)$ for $A = \cup_i A_i$ such that $S \subset A$ and for $i = 1, \ldots, l$,*

- $|P \cap (\cup_{p \in A_i} B(p, r))| \leq b|A_i|$,
- $|A_i| \leq c|P|$.

where $b > 1$, $0 < c < 1$, $d > 0$.

Note that every set of points has a trivial cover, since we can just set r to be half of the minimum interpoint distance. However, for our purpose we will need r to be much larger, e.g., greater than $\Delta(S)/\mathrm{polylog}(n)$. The following theorem shows that the existence of a "nice" independent set implies existence of a "nice" cover.

Theorem 8 *Let S be a (γ, δ)-independent set for P. Then for any b, there is an algorithm which produces a sequence of sets $A_1, \ldots, A_k \subset P$ constituting a $(b, \delta, \frac{\gamma}{(1+\gamma)\log_b n})$-cover for S.*

Proof: The proof is algorithmic. The algorithm works in a greedy fashion. Each new A_i is created by taking an arbitrary uncovered point from P together with all points within a small distance r from it. If the set C_i of points from P within distance r from A_i is not much larger than A_i itself (i.e., its cardinality is less than $b|A_i|$), then the set A_i is added to the cover. Otherwise, the points from C_i are added to A_i and the step is repeated. The number of steps per A_i cannot exceed $O(\log n)$, as each step increases the size of A_i by a constant multiplicative factor.

The formal description of the procedure computing $A_1 \ldots A_k$ for S is as follows.

Algorithm Cover:
$S = P \cap R(q, r_q, \beta r_q)$;
$r \leftarrow \frac{\gamma\Delta(S)}{\log_b n}$; $j \leftarrow 0$;
repeat
    $j \leftarrow j + 1$;
    choose some $p_j \in S$; $B_j^1 \leftarrow \{p_j\}$;
    $i \leftarrow 1$;
    while $|P \cap \cup_{q \in B_j^i} B(q, r)| > b|B_j^i|$ **do**
        $B_j^{i+1} \leftarrow P \cap \cup_{q \in B_j^i} B(q, r)$;
        $i \leftarrow i + 1$
    endwhile;
    $A_j \leftarrow B_j^i$; $S \leftarrow S - A_j$; $P \leftarrow P - A_j$
until $S = \phi$;
$k \leftarrow j$.

The first 3 stages of the algorithm are depicted in Figure 3.3. Ultimately, the upper-left cluster of points will become the first set in the cover, as well as the lower-right one.

In order to prove the correctness of the algorithm, it suffices to make the following four claims.

- $S \subset A = \cup_j A_j$ — Follows from the termination condition of the outer loop.
- for all $j \in \{1, \ldots, k\}$ and any $p \in S$, $|P \cap \cup_{q \in A_j} B(p, r)| \leq b|A_j|$ — Follows from the termination condition of the inner loop.
- for all $j \in \{1, \ldots, k\}$, $|A_j| \leq \delta|P|$ — Clearly, for any j, the inner loop is repeated at most $\log_b n$ times. Hence, $\max_{q \in A_j} D(p_j, q) \leq r \log_b n \leq \gamma\Delta(S)$. Since S is a (γ, δ)-independent set, we have that $|B(p_j, \gamma\Delta(S)) \cap P| \leq \delta|P|$. Hence, $|A_j| \leq \delta|P|$.
- $r \geq \frac{\gamma\Delta(A)}{(1+\gamma)\log_b n}$ — Since $\Delta(A) \leq \Delta(S) + r \log_b n = \Delta(S) + \gamma\Delta(S) = (1+\gamma)\Delta(S)$.

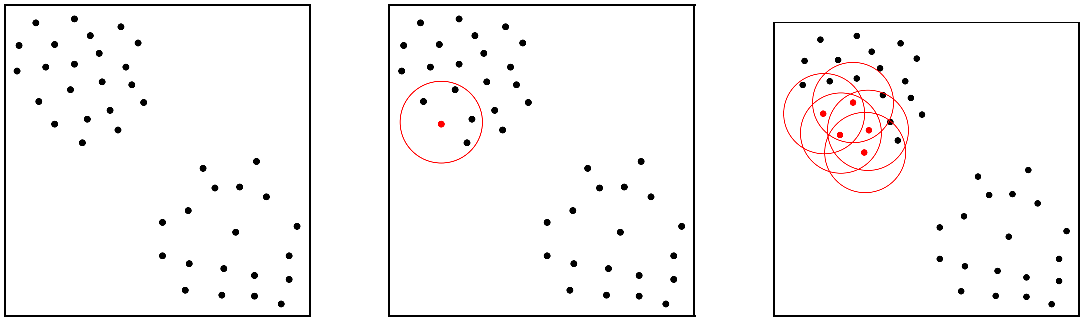

Figure 3.3: First 3 stages of the algorithm Cover.

■

Corollary 3 *For any P, $0 < \alpha < 1$, $\beta > 1$, $b > 1$, one of the following properties must hold:*

1. *P has an (α, α, β)-ring separator $R(p, r, \beta r)$, or*

2. *There is a (b, α, d)-cover for some $S \subset P$ such that $|S| \geq (1 - 2\alpha)n$ and $d = \frac{1}{(2\beta+1)\log_b n}$.*

3.1.1 Constructing ring-cover trees

The construction of a ring-cover tree is recursive. For any given P at the root, we use properties (1) and (2) in Corollary 3 to decompose P into some smaller sets $S_1, \ldots, S_l$; these sets are assigned to the children of the node for P. Note the base case case is when P is sufficiently small and we omit that in this abstract. We also store some additional information at the node for P which enables us to restrict the nearest neighbor search to one of the children of P, by using distance computations or point location queries. There are two cases depending on which of the two properties (1) and (2) holds. Let $\beta = 2(1 + \frac{1}{\delta})$, $b = 1 + \frac{1}{\log^2 n}$, and $\alpha = \frac{1-1/\log n}{2}$.

Case 1. In this case, we will call P a **ring node**. We define its children to be $S_1 = P \cap B(p, \beta r)$ and $S_2 = P - B(p, r)$. Also, we store the information about the ring separator R at the node P.

Case 2. Here, we call P a **cover node**. We define $S_i = P \cap \cup_{p \in A_i} B(p, r)$ and $S_0 = S - A$. The information stored at P is as follows. Let $r_0 = (1 + 1/\delta)\Delta(A)$ and let $r_i = r_0/(1+\delta)^i$ for $i \in \{1, \ldots, k\}$, where $k = \log_{1+\delta} \frac{(1+1/\delta)\log_b n}{\gamma} + 2c$. Notice that $r_k = \frac{\gamma \Delta(A)}{\log_b n 2c} \leq \frac{r}{2c}$. For each r_i, generate an instance of c-PLEB with balls $B(p, r_i)$ for $p \in A$; all instances are stored at P.

Theorem 9 *The ring-cover tree can be constructed in deterministic $\tilde{O}(n^2)$ time.*

Proof: The construction proceeds as follows. On each level, we determine if a node is a ring or cover node; then we compute the ring or the cover. As the number of levels is $\tilde{O}(1)$, it is sufficient to consider only the first (root) node P, which we construct as follows. Firstly, for each $p \in P$ we construct a list L_p containing all other points $q \in P$ sorted in increasing order of $D(p, q)$. This takes $O(n^2 \log n)$ time. Then, we apply the method from the proof of Theorem 1. More specifically, we compute the functions f_p^∞ and f_p^0 for each $p \in P$. Having the lists L_p it can be easily implemented in $O(n^2)$ time. Then we try to find a ring; if one is found we are done. In the opposite case we find an independent set and then apply the algorithm COVER to find a cover. It can be verified that this algorithm runs in time $\tilde{O}(n)$ time assuming the lists L_p are given. The c-PLEB generation adds only another $\tilde{O}(1)$ factor. Thus the total required time is $\tilde{O}(n^2)$. ■

We now describe how to efficiently search a ring-cover tree. It suffices to show that for any node P we can restrict the search to one of its children using a small number of tests. Let $\min_q(p, p')$ denote the point out of p and p' that is closer to q. The search procedure is as follows; we omit the obvious base case.

Procedure Search:

1. **if** P is a ring node with an (α, α, β)-ring separator $R(p, r, \beta r)$ **then**:

   (a) **if** $q \in B(p, r(1 + 1/\delta))$ **then** return **Search**(q, S_1);

(b) **else** compute $p' = \mathbf{Search}(q, S_2)$; return $\min_q(p, p')$.

2. **if** P is a cover node with a (b, c, d)-cover $A_1, \ldots, A_l$ of radius r for $S \subset P$ **then**:

   (a) **if** $q \notin B(a, r_0)$ **then** compute $p = \mathbf{Search}(q, P - A)$, choose any $a \in A$, and return $\min_q(p, a)$;

   (b) **else if** $q \in B(a, r_0)$ for some $a \in A$ but $q \notin B(a', r_k)$ for all $a' \in A$ **then** using binary search on r_is, find a $c(1+\delta)$-NN p of q in A, compute $p' = \mathbf{Search}(q, P - A)$, and return $\min_q(p, p')$;

   (c) **else if** $q \in B(a, r_k)$ for some $a \in A_i$ **then** return $\mathbf{Search}(q, S_i)$.

3.1.2 Analysis of ring-cover trees

We begin the analysis of the ring-cover tree construction by establishing the validity of the search procedure.

Lemma 6 *Procedure Search(q, P) produces a $c(1+\delta)$-nearest neighbor for q in P.*

Proof: Consider the two cases:

1. P is a ring node.

   (a) Consider any $s \in P - S_1$. Then $D(s, p) \leq D(s, q) + D(q, p)$, implying that $D(s, q) \geq D(s, p) - D(q, p)$. Since $s \notin S_1$, we know that $D(s, p) \geq \beta r = 2(1 + 1/\delta)r$, while $D(p, q) \leq r(1 + 1/\delta)$. Then, $D(s, q) \geq (1 + 1/\delta)r \geq D(q, p)$.

   (b) For any $s \in B(p, r)$, $D(q, p) \leq D(q, s) + D(s, p)$, implying that $D(q, s) \geq D(q, p) - D(s, p) \geq D(q, p) - r$. It follows that $\frac{D(q,p)}{D(q,s)} \leq \frac{D(q,p)}{D(q,p)-r} = 1 + \frac{r}{D(q,p)-r} \leq 1 + \delta$.

2. P is a cover node.

   (a) Similar to Case 1(b).

   (b) Obvious.

(c) For any $p \in P - S_i$, $D(p,a) \geq r$. Since $q \in B(a, cr_k)$, we have $D(q,a) \leq cr_k$. Thus

$$D(p,q) \geq D(p,a) - D(a,q) = r - cr_k \geq cr_k \geq D(q,a)$$

Therefore, we can restrict the search to points in S_i.

■

The proofs of Lemmas 7 and 8 are omitted.

Lemma 7 *The depth of a ring-cover tree is* $O(\log_{1/2\alpha} n) = O(\log^2 n)$.

Lemma 8 *Procedure Search requires* $O(\log^2 n \cdot \log k)$ *distance computations or c-PLEB queries.*

Lemma 9 *A ring-cover tree requires space at most* $O(knb^{\log_{1/2\alpha} n}(1+2(1-2\alpha))^{\log_n}) = O(n \operatorname{polylog} n)$ *not counting the additional non-data storage used by algorithms implementing c-PLEBs.*

Proof: Let $S(n)$ be an upper bound on the space requirement for a ring-cover tree for point-set P of size n. Then for a cover node:

$$\begin{aligned} S(n) \leq\ & \max_{l} \max_{A_1 \ldots A_l,\ A_i \text{ disjoint},\ |A_i| \leq \alpha n,\ |A| \geq (1-2\alpha)n} \\ & [\sum_{i=1}^{l} S(b|A_i|)] + S(n - |A|) + |A|bk \end{aligned}$$

For a ring node:

$$S(n) \leq 2S\left(\frac{n}{2}(1 + 2(1-2\alpha))\right) + 1$$

The bound follows by solving this recurrence. ■

Corollary 4 *Given a data structure for c-PLEB which uses* $f(n)$ *space on an instance of size* n *where* $f(n)$ *is convex, a ring-cover tree for an* n*-point set* P *requires total space* $O(f(n \operatorname{polylog} n))$.

The following is a useful observation, which intuitively means that we can represents the inputs points in the c-PLEB instances generated by the above process using "low resolution", without incurring much error.

Fact 2 *For any c-PLEB instance (C, r) generated by a ring-cover tree,* $\dfrac{\Delta(C)}{r} = O\left(\dfrac{1+\epsilon}{\gamma}\log_b n\right)$.

3.1.3 More efficient construction

We do not know if the quadratic construction time can be improved for general metrics. However, if we are given a black-box *dynamic* data structure for c-PLEB and are allowed to use randomization, we can build the data structure much faster.

Theorem 10 *The ring-cover tree can be constructed using randomization in time $\tilde{O}(n) \cdot P(n)$, where $P(n)$ is the time to perform one $\log n$-PLEB query or update.*

Proof: We need to perform efficiently the following two tasks: finding a ring separator (if any exists) and computing the cover. We start from the separator. Recall that we look for a (α, α, β)-separator, where $\alpha = \frac{1-1/\log n}{2}$. Actually, it is easy to verify that in fact $(\alpha', \alpha', \beta)$-separator, where $\alpha' = \frac{1-3/\log n}{2}$, is good enough. Thus, it is sufficient to provide a procedure which outputs an $(\alpha', \alpha', \beta)$-separator if a (α, α, β)-separator exists, and does nothing otherwise. This can be easily achieved using random sampling in the following way. Let $\alpha'' = \frac{1-2/\log n}{2}$; note that $\alpha' < \alpha'' < \alpha$. We randomly sample a set U of $u = C \cdot \log^2 n$ points from P, for a constant C. For C large enough by Chernoff bound we have that with high probability

- a (α, α, β)-separator for P is a $(\alpha'', \alpha'', \beta)$-separator for U
- a $(\alpha'', \alpha'', \beta)$-separator for U is a $(\alpha', \alpha', \beta)$-separator for P.

Therefore, we just need find a $(\alpha'', \alpha'', \beta)$-separator for U. This can be easily done in $u \log u \cdot n = \tilde{O}(n)$ time (note that we have to consider all points in P, not only in U, as potential separator centers).

We now address the problem of computing the cover. To this end observe that the algorithm as can be naturally stated in terms of a sequence of $\tilde{O}(n)$ PLEB operations (queries and updates). In order to use c'-PLEB instead of PLEB procedure, it is sufficient to replace r by r/c' in order to make sure that the cover elements output by the greedy procedure have small cardinality. As long as $c' = O(\log n)$, this does not change the parameters of the algorithm, at least asymptotically. ■

Since we pushed the construction time to $\tilde{O}(n) \cdot P(n)$, we can now attempt to make the data structure *dynamic*. To this end, it is sufficient to notice that the ring-cover tree satisfies the assumptions of Theorem 2. Therefore, we obtain the following theorem.

Theorem 11 *The c-NNS problem can be solved dynamically with update cost $\tilde{O}(U_c(n)+Q_{c'}(n))$ and query time $\tilde{O}(Q_c(n))$, where $U_c(n)$ is the update cost and $Q_c(n)$ is the query cost for the c-PLEB problem and $c \leq c' = O(\log n)$.*

3.2 Bichromatic closest-pair

In this section we address the problem of maintaining the approximate *chromatic closest pair (CCP)*. Note that the static problem can be trivially solved by performing $O(n)$ approximate nearest neighbor queries and updates. The dynamic problem is more challenging. For example, it is possible to reduce dynamic c-NNS to dynamic c-CCP with essentially no increase in running time. This can be done by "painting" all data points using the same color (say blue). Now, when we want to perform the c-NNS query q, we add q to the c-CCP data structure painted in red, compute the bichromatic closest pair (which has to be of the form (q, p)), return p and remove q from the data set.

Nevertheless, Eppstein [Epp95] showed that, in the amortized sense, c-CCP can be reduced to c-NNS with only polylogarithmic overhead. More accurately, he showed the reduction for the *bichromatic* problem, i.e., when the number of colors is equal to 2. The exact parameters of his reduction are as follows.

Fact 3 *[Epp95] Let $T(n)$ be the upper bound of any operation (insertion, deletion* *or query) for a data structure solving the dynamic c-NNS problem. Then one can solve the dynamic c-BCP problem in $O(T(n) \log n)$ amortized time per insertion and $O(T(n) \log^2 n)$ amortized time per deletion.*

His result is actually very general and applies to *any* distance function (i.e., not necessarily a metric). This, for example, means that it also allows us to reduce the decision version of c-BCP to the decision version of c-NNS (i.e., c-PLEB), which saves some polylogarithmic factors in case we only wanted to solve the decision problem.

The restriction of the number of colors to 2 is not crucial, as shown in the following Lemma.

Lemma 10 *The c-CCP with up to k colors can be maintained in time $O(T(n) \log k)$ per update, where $T(n)$ is the update cost of a c-BCP data structure.*

Proof: Let P be the set of point to be maintained and let $c : P \to \{1 \dots k\}$ be the color function. We maintain $\log k$ different bi-colorings of P using $\log k$ data structures. The color a point $p \in P$ in the ith data structure is equal to the ith bit of $c(p)$. Clearly, the $\log k$ sets can be maintained in time $O(T(n) \log k)$ per update. Now observe that any multichromatic pair (p, p') is also bichromatic in one of the $\log k$ colorings and vice versa. Thus the minimum bichromatic pair over all $\log k$ data structures is also the minimum multichromatic pair. ∎

3.3 Minimum spanning tree

In this section we show how to compute c-MST by making $O(n \log n)$ operations on approximate CCP data structure. We show two reductions: one based on Kruskal's algorithm and another one based on Prim's algorithm. The first one is easier to describe, but the second one reduces c-MST directly to c-BCP (instead of c-CCP), and therefore saves a $\log^2$ factor. The results in this section first appeared in [IM98, IST99].

3.3.1 Kruskal's algorithm

The Kruskal's algorithm can be implemented as follows. We maintain a c-CCP data structure for the input set of points P. We call the set of points of a given color a *cluster*. Initially, all points have different color; thus all clusters are singletons. The algorithm performs $n-1$ steps. In each step, it finds the (approximately) smallest edge between two different clusters, say C_1 and C_2, and then merges C_1 and C_2 into one cluster. The first operation clearly corresponds to reporting the (approximately) smallest multichromatic pair. The second operation can be performed by changing the colors of all points in the smaller cluster (say C_1) to the color of C_2. In the worst-case, it could require $\Omega(n)$ insertions and deletions. However, since we only modify the smaller cluster, each point can be recolored at most $\log n$ times. Thus overall the algorithm performs $n-1$ c-CCP queries and at most $n \log n$ updates of the c-CCP structure.

Unfortunately, we are not done yet, since it is a-priori not clear if the algorithm indeed computes a c-approximation to MST. Obviously, if $c = 0$, the correctness follows from the correctness of Kruskal's algorithm. Below, we show that the above algorithm returns a c-approximation to the cost of whatever the Kruskal algorithm reports (and therefore with respect to the cost of minimum spanning tree).

Lemma 11 *The above algorithm reports a c-approximate MST.*

Proof: Let A_c be the above algorithm and recall that A_0 finds an optimal solution. Let $e_1, e_2 \ldots$ be the edges picked by A_c in steps $1, 2$ and so on, and let $e'_1, e'_2 \ldots$ be the edges picked by A_0. It is immediate that the costs of $e'_1, e'_2 \ldots$ are increasing. Consider the edge e_i. It has been added only if for any inter-cluster edge e (before step i) the cost of e'_i is at most c times the cost of e. However, at least *one* of the edges $e'_1 \ldots e'_i$ is an inter-cluster edge, because otherwise at the end of step $i-1$ we would have at most $n-i$ clusters, while the actual number of clusters is $n-i-1$. Thus the cost of e_i is at most c times the cost of e'_i. ∎

3.3.2 Prim's algorithm

Prim's MST algorithm starts by choosing an arbitrary source node s and initial tree containing only s. In each step, we augment T with the smallest edge leaving T until T spans the graph. We can apply dynamic approximate BCP as follows. Initially, s is the only red node while all other nodes are blue. In the general step, BCP picks a min-cost red-blue edge whose blue end-point we color red. Clearly, the whole procedure performs $O(n)$ updates and closest pair queries. We will prove approximation guarantees in a much more general setting.

In fact, both Prim's and Kruskal's algorithm are example of what we call a *generic* MST algorithm. In the standard generic MST algorithm [CLR90] we are building a minimum spanning tree T for a weighted graph $G = (V, E, W)$ on n vertices by adding one edge at the time. In each step, an arbitrary cut C is chosen that is not crossed by any edge from the current T, and then we add a minimum edge crossing C to T. In Prim's algorithm T consists of a "source" tree containing the source node. All nodes outside of the source tree are singletons, and C is the cut separating the source tree from the singletons. We will show that the generic MST works approximately:

Theorem 12 *If in each step of the generic MST algorithm, we pick an edge crossing C which is at most a factor $(1+\epsilon)$ from being a minimal such edge, the resulting tree is at most a factor $(1+\epsilon)$ from minimality.*

Proof: As usual, the (approximate) generic MST algorithm also works for disconnected graphs, producing an approximate MST for each component. In our proof however, for simplicity, we assume the graph is connected.

Lemma 12 *For any graph G', if e crosses some cut C and x is the minimum cost of an edge crossing C, there is a spanning tree T containing e whose cost is at most $W(e) - x$ from being minimal.*

Proof: Let T_{min} be a minimum spanning tree. If T_{min} contains e, we are done. Otherwise, let P be the unique path in T_{min} between the end-points of e. Then P contains an edge f crossing C. Set $T = T_{min} \cup \{e\} \setminus \{f\}$. Then T contains e and

$$W(T) = W(T_{min}) + W(e) - W(f) \geq W(T_{min}) + W(e) - x$$

■

Generally, for any graph G', let $MST(G')$ be the minimum cost of a spanning tree of G'. If A is a subset of the edges of G', $G'_{/A}$ denotes G' with all edges in A contracted. Define $\delta = \epsilon/(1+\epsilon)$. Then $y \leq (1+\epsilon)x \iff x \geq (1-\delta)y$.

For $i = 1, ..., n-1$, let e_i and C_i be the edge and cut of the ith step of our approximate generic MST algorithm. We will apply Lemma 12 with $G' = G_{/\{e_1 \ldots e_{i-1}\}}$. The minimum cost of a spanning tree in G' containing e_i is $W(e_i) + MST(G'_{/\{e_i\}})$. Further e_i is at most $\delta(W(e_i))$ from being a minimal edge crossing C_i, so by Lemma 12, $W(e_i) + MST(G'_{/\{e_i\}}) \leq MST(G') + \delta(W(e_i))$. However, $G'_{/\{e_i\}} = G_{/\{e_1 \ldots e_i\}}$, so $(1-\delta)W(e_i) \leq MST(G_{/\{e_1 \ldots e_{i-1}\}}) - MST(G_{/\{e_1 \ldots e_i\}})$. It follows by telescoping that $\sum_{i=1}^{n-1}(1-\delta)W(e_i) \leq MST(G) - MST(G_{/\{e_1 \ldots e_{n-1}\}})$, but $G_{/\{e_1 \ldots e_{n-1}\}}$ is a singleton node so $MST(G_{/\{e_1 \ldots e_{n-1}\}}) = 0$. Thus we have proved $(1-\delta)\sum_{i=1}^{n-1} W(e_i) \leq MST(G)$, hence that $\sum_{i=1}^{n-1} W(e_i) \leq (1+\epsilon)MST(G)$. This completes the proof of Theorem. ■

Corollary 5 *The $(1+\epsilon)$-approximate MST can be solved using $O(n)$ calls to $(1+\epsilon)$-approximate dynamic BCP.*

3.4 Furthest neighbor

In this section we focus on the "dual" version of the nearest neighbor, namely on the the furthest neighbor problem. We will focus on the *static* version of the problem (no updates); we will see later in Section 4.7.1 how to obtain a reduction from the *dynamic* furthest neighbor, although the accuracy of that reduction will be somewhat lower. We perform it in two steps. In the first step, we show how to get a $\sqrt{2}$-approximate solution to the problem with $\tilde{O}(d)$ query time and $\text{poly}(d, \log n)n$ preprocessing. Then we show how to extend the algorithm to solve c-FNS for any $c > 1$ with query time $\tilde{O}(T(n))$ and $\tilde{O}(P(n) + \text{poly}(d)n)$ where $T(n)$ and $P(n)$ are (respectively) the query time and preprocessing time for static c-NNS algorithm. The material in this section is taken from [GIV99].

3.4.1 A $\sqrt{2}$-approximation for furthest neighbor problems

In this section we develop a data structure for the c-furthest neighbor search problem (c-FNS) by using fast algorithms for c'-MEB, where $c' = 1/L$ and $c = \sqrt{2} + 1/L$, where $L = n^{\Theta(1)}$, with the exponent being an arbitrarily large constant. The preprocessing time of the algorithm is bounded by the running time of c'-MEB; the query time is $O(d^2 \log n)$. Notice, that this immediately implies $\sqrt{2}$-approximations for the diameter and discrete center problem in c'-MEB time.

The basic idea of our algorithm is as follows. Let P the n-point subset of $\Re^d$. Suppose that we could in fact compute the exact minimum enclosing ball $B(O^*, r^*)$ of P. Then there is a subset $R \subseteq P$ of at most $d+2$ points such that each point of R lies on the sphere $S(O^*, r^*)$, and the center O^* is in the convex hull of R (see Figure 3.4). Our data structure consists simply of the set R. For a query point q that is not O^*, (if q is O^*, any point in R is a furthest neighbor of q) we consider the hyperplane h_q passing through O^* and perpendicular to the line through O^* and q. Since O^* is in the convex hull of R, there must be at least one point in R on the side of h_q that does not contain q. We return any such point as our approximate furthest neighbor of q. It is easy to see that the answer is a $\sqrt{2}$-approximation, and that the query can be answered in $O(d^2)$ time.

Since we can only afford to compute an approximate minimum enclosing ball $B(O, r)$ for some r sufficiently close to r^*, we modify the algorithm as follows.

- We now choose our subset $R \subseteq P$ of points to be the set of all points of P within $\delta > 0$ of the sphere $S(O, r)$, that is, $R = P \cap S^\delta(O, r)$. We show that this subset satisfies nearly the same properties as in the exact case.

- However, the set R chosen this way may be quite large, in fact of size $\Omega(n)$. So we first apply a small random perturbation to the point set P (before finding the approximate enclosing ball and R) so that with positive probability the set R will have $O(d \log n)$ points.

Perturbation. Assume for simplicity that $[0,1]^d$ is the boundng box for P. Define a new points set P' by the following random process: for each point $p_i \in P$, choose

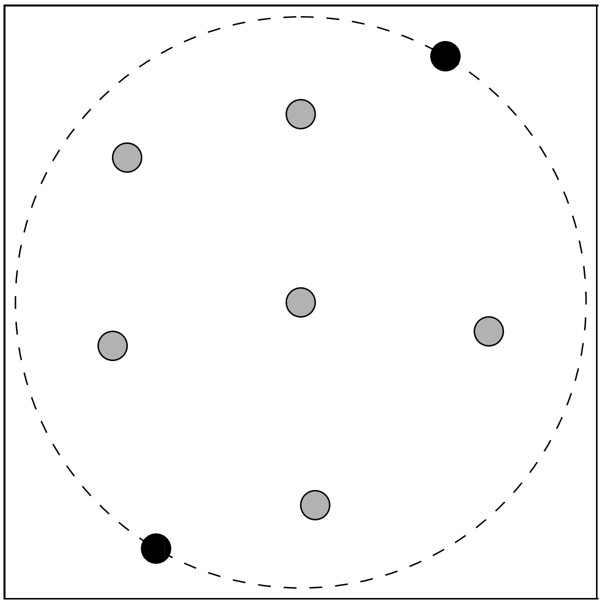

Figure 3.4: An example set of points and its Minimum Enclosing Ball. The solid points form the set R. One can observe that each hemisphere contains one point from R.

a random vector v_i such that $|v_i| = \gamma$ and define $p'_i = p_i + v_i$. Consider now any q from the "δ-precision cube" $C = [0,1]^d \cap (\delta\mathcal{Z})^d$($\mathcal{Z}$ denotes the set of integers) and $r = |q - p|$ for any $p \in C$ s.t. $r \geq 1/2$. Define $P(q,r) = P \cap S^\delta(q,r)$.

Lemma 13 *There exist constants $c_1, c_2 > 1$ such that if $r - \sqrt{r^2 - \gamma^2} \leq \delta/c_1$ and $\gamma/\delta \geq c_2 n\sqrt{d}$, then there is at least a constant probability that $|P(q,r)| = O(d \log n/\gamma)$ for* all *q, r as above.*

Remark 1 *Notice that the first condition is implied by $\gamma^2 \leq \delta r^2/c'_1$ for some $c'_1 \geq 1$. Also, it is possible to satisfy both conditions simultaneously by setting $\delta = \Theta(\frac{1}{dn^2})$ and $\gamma = \Theta(\frac{1}{\sqrt{dn}})$.*

Proof: Take any $p \in P$. Let $S = S(q,r)$ and let $S' = S(p,\gamma)$. Define $A = S^\delta \cap S'$ (see Figure 3.5 for the illustration). We will first show that $\frac{|A|}{|S'|} \leq 1/n$ where $|A|$

denotes the measure of A (notice that the latter quantity is exactly the probability of $p' \in S^{\delta}$).

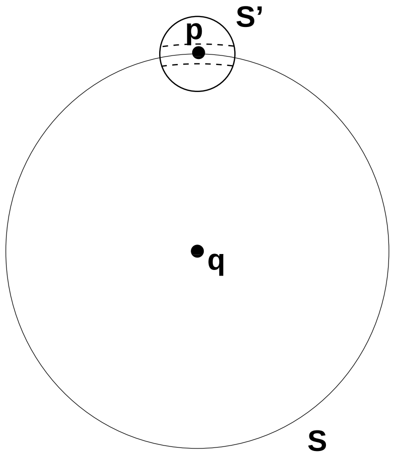

Figure 3.5: Two spheres: S (with large radius r) and S' (with small radius γ). The region A is bounded using dashed lines.

Let H be a $(d-1)$-dimensional hyperplane orthogonal to the vector $\vec{pq}$ and tangent to S. It can be verified that (for a suitable c_1) if $r - \sqrt{r^2 - \gamma^2} \leq \delta / c_1$, then $A \subset H^{2\delta}$ (i.e. A is almost "flat"). On the other hand, it is known (e.g., see [IMRV97]) that $\frac{|H^{2\delta} \cap S'|}{S'} = O(\sqrt{d}\delta/\gamma)$. Therefore, if we choose $\gamma \geq c_2 \delta n \sqrt{d}$ (for a suitable c_2) then the latter quantity becomes smaller than $1/n$.

Now we can upper bound the probability that for *fixed* pair q and r we have $|P(q,r)| > t$ by

$$\binom{n}{t} / n^t \leq (en/t)^t / n^t = (e/t)^t$$

because each point from P falls into $P(q,r)$ independently with probability $1/n$. Since the number of all pairs q, r is at most $1/\delta^{2d}$, for $t = cd \log n$ we get that the probability that there exist q, r such that $P(q,r) > t$ is at most $1/\delta^{2d} \cdot \left(\frac{e}{cd \log n}\right)^{cd \log n}$, which is less than $1/2$ for a suitable c. ■

Remark 2 *Notice that the dependence on n in the above bound can be reduced to $\log n / \log \log n$.*

The following "hemisphere lemma" shows that if $B(O,r)$ is an approximate minimum enclosing ball of P, then for any point $q \neq O$, there has to be at least one point in $S^{\delta}(O,r)$ "close" to the hemisphere of $S(O,r)$ opposite q. Formally, it has to be close to the set $H(q) = \{p \in S(O,r) : (q-O)\cdot(p-O) \leq 0\}$.

Lemma 14 *Let $B = B(O,r)$ be a feasible solution for the MEB problem over P such that $r = r^*(1+\beta)$ where the optimal solution has radius r^*, for $0 \leq \beta \leq 1$. Then for any point $q \in S = S(O,r)$ there exists a point $p \in P$ within distance $a\sqrt{\beta}r$ from $H(q)$, for a constant a.*

Proof: We can assume $r^* = 1$. Also, without loss of generality we can assume that $q - O = (1, 0, \ldots 0)$. Let $H = H(q)$. Let $\Delta > \sqrt{\frac{\beta}{1+\beta}}$. By contradiction, assume that $D(P,H) \geq 2\Delta$. Define $O' = (\Delta, 0, \ldots, 0)$; we will show that for all points $p \in P$ we have $|p - O'| < \frac{1}{1+\beta}$.

Rotate the space such that the vector $O - O'$ and p lie on the same plane. The point configuration is then depicted as on Figure 3.6.

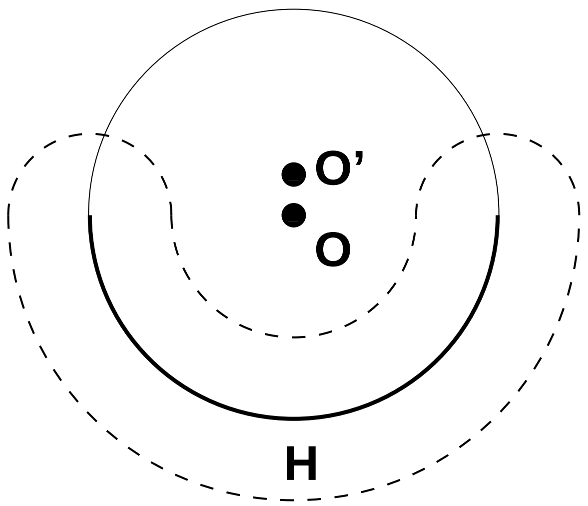

Figure 3.6: The point configuration after the rotation. All points from P must lie outside of the region surrounded by the dashed line.

It is easy to verify by inspection that in order for a point $p = (x, y)$ to be as far from O' as possible, it must be that $y \geq \Delta$. However, since $x^2 + y^2 \leq 1$, we conclude that

$$|p - O'| = (y - \Delta)^2 + x^2 \leq 1 - 2y\Delta + \Delta^2 \leq 1 - \Delta^2 < \frac{1}{1+\beta}$$

which was to be shown. ■

Final analysis. Our algorithm consists of finding an approximate minimum enclosing ball $B(O, r)$ of P, where $r = r^*(1 + \beta)$ and $\beta = \Theta(\delta^2)$, for $\delta = 1/L$. We then take R to be the set $P \cap S^\delta(O, r)$, the set of points in P lying within δ of the sphere $S(O, r)$. From Lemma 13, R has size $O(d \log n)$ with constant probability. The procedure to find R takes $\tilde{O}(d^3 n))$ time. Given any query point q, we simply return the point in R farthest from q as our approximate farthest neighbor.

It follows from Lemmas 14 and 13 that for any query point q, there is a point $u \in R$ within distance $a\delta r$ from $H(q)$. Let $t = |q - O|$. We know that $|q - u| \geq \sqrt{t^2 + r^2} - a\delta r$, while the furthest point from q (say p) is within distance at most $|q - O| + |p - O| \leq t + r$ from q. Thus

$$\frac{|q-u|}{|q-p|} \geq \frac{\sqrt{t^2+r^2} - a\delta r}{t+r}.$$

The latter ratio an be easily shown to be at least $(1/\sqrt{2})(1 - 1/n^{\Theta(1)})$, and so u is a $\sqrt{2}(1 + 1/n^{\Theta(1)})$-furthest neighbor of q.

Remark 3 *We mention that if one is only interested in solving the diameter or discrete center problem (not the furthest neighbor search problem), the algorithms can be considerably simplified. In both cases, we start from finding (approximate) minimum enclosing ball $B = B(O, r)$. For the diameter, we take a point $p \in P$ closest to the boundary of B, find a point $p' \in P$ furthest from p and return (p, p'). For the discrete center, we find a point $p \in P$ closest to O and return it as a center. Since the approximation factor and the analysis remain essentially the same, we omit further description.*

3.4.2 A $(1+\epsilon)$-approximation for FNS

In this section we show how to answer $(1+\epsilon)$-FNS queries by performing $\tilde{O}(1)$ $(1+\epsilon)$-approximate nearest neighbor queries; the same overhead holds for the preprocessing.

The reduction is as follows. At the beginning, similarly as in the previous section, we define an approximate minimum enclosing ball $B = B(O, r)$ of the given set of points P. Since we can get arbitrarily good approximation at essentially no cost, in the following we will assume for simplicity that B is *exact*; this incurs only a multiplicative error of $1 + 1/n^{\Theta(1)}$, as in the previous section.

Let $\alpha > 0$ (specified later). Let $r_i = r/(1+\alpha)^i$, let $k = O(1/\alpha)$ be the first integer such that $r_k \leq \sqrt{2} - 1$ and let $B = B(O, r_i)$ for $i = 0 \ldots k$ be a sequence of concentric balls around O. We round each point in P to the nearest sphere $S_i = S(O, r_i)$ (notice that in this way we make only αr additive error). For each ball B_i we build an $(1+\epsilon)$-approximate nearest neighbor data structure for the set of points on S_i.

Now, for each point $p \in P$ and each ball B_i, we define the "antipode" p^i of p w.r.t. B_i defined as follows. Let p_1 and p_2 be the two points of the intersection of the sphere of B_i with the line passing through p and O. Let h_p denote the hyperplane through O that is perpendicular to the line through p and O. We define p^i to be the one of the points p_1, p_2 which lies on the side of h_p different from the side containing p.

Now, in order to find the furthest neighbor of q, we issue a $(1+\epsilon)$-approximate nearest neighbor query with the point q^i in the data structure for the points on S_i, for all i. Among the points found, we return the one furthest from q.

The complexity of the algorithm is clear from the description. In the following we will show that (modulo the rounding phase) the algorithm indeed computes a $(1+\epsilon)$-FN. To this end, consider the actual furthest neighbor p of q. We can assume that $p \in S_i$ and $q \in S_j$. Let $H(q)$ denote the hemisphere of B (or B_j) "opposite" q, that is $H(q) = \{t \in S | (t - O) \cdot (q - O) \leq 0\}$. By an argument as in the previous section we have that point p has to lie in $H(q)$.

Let s be the point returned as the $(1+\epsilon)$-approximate nearest neighbor of q^i in the data structure for $P \cap S_i$. From the definition it has to satisfy the property

$\frac{|s-q^i|}{|p-q^i|} - 1 \leq \epsilon$. We will show that

$$L = \frac{\frac{|q-p|}{|q-s|} - 1}{\frac{|s-q^i|}{|p-q^i|} - 1} \leq 1$$

which implies $\frac{|q-p|}{|q-s|} \leq 1+\epsilon$. To this end, observe that we can rotate and scale the space such that O is the origin and the ball B_i has radius 1. Moreover, we can assume that all points q, q^i, p, s lie in a 3-dimensional space. In fact, we can move s to the plane spanned by q, q^i, p such that all distances of interest remain unchanged. Therefore, we can assume that all four points lie on the plane. Let $q^i = 1 = (0,1)$, $q = (0,-y)$, $y \geq 0$ and $p, s \in S(0,1)$ lie in the upper left quadrant, so that $p = (\cos\,\theta, \sin\theta)$ and $s = (\cos\theta', \sin\,\theta')$, such that $\theta, \theta' \in [\pi/2, \pi]$ and $\theta < \theta'$. We now want to maximize

$$\frac{\frac{|q-p|}{|q-s|} - 1}{\frac{|s-1|}{|p-1|} - 1}$$

We will do that in two phases. Firstly, we show that the above expression is maximized for $q = -1 = (0,-1)$. Then we bound

$$L' = \frac{\frac{|-1-p|}{|-1-s|} - 1}{\frac{|s-q^i|}{|p-q^i|} - 1}$$

Lemma 15 *For any fixed p and s, the value of $L = L(q)$ is maximized when $q = -1$.*

Proof: Notice that the denominator of L does not depend on q, therefore it is sufficient to maximize $F(y) = \frac{|q-p|}{|q-s|}$, or equivalently, $F(y)^2 - 1$. Then

$$F(y)^2 - 1 = \frac{\cos^2\theta + (\sin^2\theta + y)^2}{\cos^2\theta' + (\sin^2\theta' + y)^2} - 1 = \frac{1 + y^2 + 2y\sin\theta}{1 + y^2 + 2y\sin\theta'} - 1 = \frac{2y(\sin\theta - \sin\theta')}{1 + y^2 + 2y\sin\theta'}$$

The latter expression can be easily seen to achieve maximum for $y = 1$, independently of the value of θ and θ', as long as $\sin\theta - \sin\theta' \geq 0$ and $\sin\theta' \geq 0$. ∎

Lemma 16 *$L' \leq 1$.*

Proof: We can expand L' into

$$\left(\frac{\sqrt{1+\sin\theta}}{\sqrt{1+\sin\theta'}}-1\right) / \left(\frac{\sqrt{1-\sin\theta'}}{\sqrt{1-\sin\theta}}-1\right)$$

Let $\phi = \theta/2 - \pi/4$ and $\phi' = \theta'/2 - \pi/4$; note that $\phi, \phi' \in [\pi/4, \pi/2]$. We use the identities $1 + \sin\theta = 2\cos^2\phi$ and $1 - \sin\theta = 2\sin^2\phi$. In this case, the expression becomes

$$L' = \frac{\frac{\cos\phi}{\cos\phi'} - 1}{\frac{\sin\phi'}{\sin\phi} - 1}.$$

In order to show $L' \leq 1$ it is sufficient to show $\cos\phi\ \sin\phi - \sin\phi'\ \cos\phi' \leq 0$. However, the LHS of this expression is just $(\sin 2\phi - \sin 2\phi')/2$, which is is smaller than 0 since $\phi \leq \phi'$. ∎

Thus, the above procedure returns a $(1+\epsilon+O(\alpha))$-approximate furthest neighbor.

Theorem 13 *The static $(c+\delta)$-approximate furthest neighbor can be solved with the preprocessing time $\tilde{O}(d^{O(1)}n)$ plus the cost of initiating $O(1/\delta)$ data structures for static c-PLEB; the query time is bounded by the cost of $O(1/\delta)$ c-PLEB queries.*

3.5 Bottleneck matching

Let M^* denote the perfect matching of P with the smallest bottleneck cost, and let c^* be the cost of M^*. In this section, we describe an algorithm that computes a perfect matching whose cost is at most $2(1+\epsilon)c^*$. This algorithm appeared first in [GIV99].

For any real $r \geq 0$, let $G(r)$ denote the graph whose vertex set is P and whose edge set is $\{(p_i, p_j) | p_i \neq p_j, D(p_i, p_j) \leq r\}$. Let r^* be the smallest value of r for which each connected component of $G(r)$ has an even number of vertices. Let $G_1, \ldots, G_k$ be the connected components of $G(r^*)$, and for $1 \leq i \leq k$, let T_i be any spanning tree of G_i. Note that the length of any edge in T_i is at most r^*.

Lemma 17 *The cost c^* of the optimal matching is at least r^*.*

Proof: Fix some $r < r^*$. We will show that for any perfect matching M of P, the cost $c(M)$ of M is greater than r.

By definition, there is a connected component G' of $G(r)$ which has an odd number of vertices. Thus for any perfect matching M, there is a pair $(u,v) \in M$ such that u is a vertex of G' and v is not. Clearly $D(u,v) > r$, and therefore $c(M) \geq D(u,v) > r$. ■

On the other hand, we can get a perfect matching of P whose cost is at most $2r^*$ by applying the algorithm of the following lemma to each T_i.

Lemma 18 *Let T be a tree whose vertex set $V = \{p_1, \ldots, p_{2m}\}$ has even cardinality, and let ℓ denote the length of the longest edge in T. We can construct a perfect matching of V whose bottleneck cost is at most 2ℓ. Given T, such a matching can be computed in $O(m)$ time.*

Proof: We make T into a rooted tree by picking an arbitrary vertex as the root. Let u be any internal vertex of T such that all of u's children are leaves of the tree. It is easy to see that such a vertex exists.

1. If u has two or more children, let v and w be any two children of u. Delete v (resp. w) and the edge (v,u) (resp. (w,u)) incident to it. Let T' be the resulting tree. Recursively compute a perfect matching M' of the vertices in T'. Return $M' \cup \{(v,w)\}$ as the perfect matching for the vertices of T. Note that $D(v,w) \leq D(v,u) + D(u,v) \leq 2\ell$.

2. If u has only one child v, delete v and u from T. In the resulting tree T', recursively compute a perfect matching M' of its vertices. Return $M' \cup \{(u,v)\}$ as the perfect matching for the vertices of T. Clearly, $D(u,v) \leq \ell$.

It is easy to see that the bottleneck cost of the matching M returned by the above procedure is at most 2ℓ. We now sketch an $O(m)$ time implementation of the procedure. Let us call an internal vertex of a rooted tree T *interesting* if all its children are leaves. Clearly, it suffices to maintain the set of interesting vertices of T as leaves get deleted from T. At each internal vertex u of the tree, we keep a count of all the children of u that are not leaves. These counts can be computed initially in $O(m)$ time by a post-order traversal of T. The interesting vertices are those whose count is zero. When a leaf gets deleted, the count changes at only a constant number of vertices near the leaf. Hence, updating the counts takes $O(1)$ time. Thus the entire procedure can be implemented in $O(m)$ time. ∎

Reduction to nearest-neighbor. We now compute a spanning forest $\{T_1, \ldots, T_k\}$ of P with the property that each edge of of the forest has length at most $r^*(1+\epsilon)$, and each tree in the forest has an even number of vertices. Such a forest can be easily computed by running Kruskal's MST algorithm until each component is even, with $O(n \log n)$ calls to K-chromatic dynamic closest pair. Only, we replace the calls to the exact data structure by calls to a data structure for $(1+\epsilon)$-approximate K-chromatic dynamic closest pair. Using the algorithm of the above lemma, we can then compute a matching whose bottleneck cost is $2r^*(1+\epsilon)$.

Theorem 14 *The $2(1+\epsilon)$-approximate bottleneck matching can be solved using $O(n \log n)$ calls to $(1+\epsilon)$-CCP.*

3.6 Approximate counting PLEB

In this section we show how to reduce the approximate counting PLEB problem, with parameters r, α and c, to c-PLEB with radius r. The reduction overhead is $O(\log^2 n/\alpha^3)$. The problem will be used as a subroutine for the facility location algorithm in the next section.

The algorithm will use the following fact, which is a simple corollary of Chernoff bound.

Fact 4 *Let $\mathcal{D}$ be a distribution over $\{0,1\}$ such that the probability of 1 is equal to p. Then there is an algorithm which takes $s = O(\frac{\log(1/\delta)}{ab^2})$ independent samples of $\mathcal{D}$ and in $O(s)$ time produces an answer YES or NO such that:*

- $\Pr[YES|p \in [a(1-b/2), a(1+b/2)]] \geq 1-\delta$
- $\Pr[NO|p \notin [a(1-b), a(1+b)]] \geq 1-\delta$

We will focus on solving the decision version of the problem, namely we will give an algorithm such that with high probability

1. If $N_{cr}(q) \leq T(1-\alpha)$, it will answer SMALLER

2. If $N_r(q) \geq T$, it will answer GREATER

Clearly, we can obtain an algorithm for approximate range counting by producing $\log(n)/\alpha$ data structures for various values of T.

The algorithm for the decision problem is as follows. We sample a multiset S of n/T points from P with replacement. Then we build a c-PLEB data structure for S. Observe that if the case (1) from the above occurs, then the probability that the c-PLEB data structure will answer YES is at most $1-(1-T(1-\alpha)/n)^{n/T}$. For ease of exposition, we will assume that n/T is large enough so that we can approximate the latter quantity by $P_1 = 1-e^{1-\alpha}$ (otherwise, we proceed as in Section 2.2.1). Similarly, if the case (2) from the above occurs, then the probability of the YES answer is at least $P_2 = 1 - e^{-1}$. Thus

$$P_1 - P_2 \geq 1 - e^{1-\alpha} - 1 + e^{-1} = e^{-1}(e^{\alpha} - 1) \geq e^{-1}\alpha$$

Therefore (by Fact 4), if we build $O(1/\alpha^2 \log n)$ such data structures, we will be able to solve the decision problem with high probability of correctness.

Theorem 15 *The (r, c, α)-CPLEB query/update can be implemented by performing $O(\log^2 n/\alpha^3)$ (r, c)-PLEB queries/updates.*

3.7 Facility location

In this section we show how to implement the algorithm of Jain and Vazirani [JV99] as a sequence of closest pair operations[1]; this reduction appeared first in [GIV99]. The original algorithm gives factor 3 approximation to this problem and runs in

[1]We give here only a brief description of their algorithm and its proof of correctness; the reader is advised to read this section after or concurrently to reading of their paper.

$O(n^2 \log n)$ time, where n is the number of vertices[2]. Although there are algorithms which achieve lower approximation factor, their result is interesting because of its low time complexity.

We first reduce the bit precision needed to represent the distances to $O(\log n)$, in the following way. Firstly, we compute a lower bound L for the optimal facility location cost C^*, such that $L \leq C^* \leq aLn$, for a constant a. To this end, observe that the cost C' of a minimum spanning forest with k components is a lower bound for C^*; it is also easy to see $C^* \leq nC'$. We compute an a-approximate spanning forest using the techniques of Section 3.3; if we set $a > c = 1 + \epsilon$ then this computation can be done in time negligible compared with the running time of the rest of this procedure. We can then set $L = C'/a$.

After that, we collapse all pairs of vertices whose distance is less than $\delta \frac{L}{n}$ for $\delta \approx \epsilon$, forming disjoint clusters; clearly, this operation changes the value of C^* by at most $\pm \delta C^*$. This clustering can be achieved using $\tilde{O}(n)$ calls to $O(1)$-PLEB, along with a simple Union-Find data structure. We also open all facilities which have cost at most $\delta L/n$. Again, this can increase the cost of the solution by at most δL. Thus, as a result of the above transformation, the cost of the solution can go up by at most a factor of $1 + O(\delta)$. Finally, we set the value of all edges with distance greater than aLn to aLn; note that the resulting distance matrix is still a metric. We do the same for the facilities which cost more aLn. At the of this procedure we can represent each distance using only $O(\log n)$ bits of precision.

Now we proceed with the main algorithm of Jain and Vazirani. The algorithm is based on the primal-dual approach (see [GW97] for the background information on this method). The basic idea is to formulate the problem as a 0-1 integer program, relax it to a linear program, and apply primal-dual ascent to obtaining approximate solution. The 0-1 programming formulation of facility location problem is as follows. We use two kinds of variables: x_{ij} and y_i. The event $y_i = 1$ is intepreted as that the facility i is open; the event $x_{ij} = 1$ is interpreted as that the jth point (city) is using the ith facility. With this in mind, one can formulate the problem as follows.

[2]Infact their running time is $O(n_c n_f \log(n_c n_f))$, where n_c and n_f are the number of cities and facilities, respectively.

$$\begin{aligned}
\text{minimize} \quad & \sum_{i \in P_f, j \in P_c} D(i,j)x_{ij} + \sum_{i \in P_f} f(i)y_i \\
\text{such that} \quad & \\
& \forall j \in P_c : \sum_{i \in P_f} x_{ij} \geq 1 \\
& \forall i \in P_f, \forall j \in P_c : y_i - x_{ij} \geq 0 \\
& \forall i \in P_f, \forall j \in P_c : x_{ij} \in \{0,1\} \\
& \forall i \in P_f : y_i \in \{0,1\}
\end{aligned}$$

After the relaxation, the program becomes:

$$\begin{aligned}
\text{minimize} \quad & \sum_{i \in P_f, j \in P_c} D(i,j)x_{ij} + \sum_{i \in P_f} f(i)y_i \\
\text{such that} \quad & \\
& \forall j \in P_c : \sum_{i \in P_f} x_{ij} \geq 1 \\
& \forall i \in P_f, \forall j \in P_c : y_i - x_{ij} \geq 0 \\
& \forall i \in P_f, \forall j \in P_c : x_{ij} \geq 0 \\
& \forall i \in P_f : y_i \geq 0
\end{aligned}$$

The primal-dual method operates mainly on the dual formulation of the linear program, which is as follows.

$$\begin{aligned}
\text{maximize} \quad & \sum_{j \in P_c} \alpha_j \\
& \forall i \in P_f, \forall j \in P_c : \alpha_j \leq \beta_{i,j} + D(i,j) \\
& \forall i \in P_f : \sum_{j} \beta_{i,j} \leq f(i) \\
& \forall i \in P_f, \forall j \in P_c : \beta_{i,j} \geq 0 \\
& \forall j \in P_c : \alpha_i \geq 0
\end{aligned}$$

The algorithm tries to maximize the dual objective while maintaining dual feasibility. It is done by growing balls around each active city (initially all cities are active) at a uniform rate; α_j represents the radius of the ball around j. There are two types of events. The first type of event occurs when a ball hits a facility (i.e. $\alpha_j = D(i,j)$ for some pair (i,j)); to mainitain dual feasibility, $\beta_{i,j}$ must also be increased now as α_j grows. The second type of event occurs when $\sum_j \beta_{k,j} = f(k)$ for some facility k. Now none of the $\beta_{k,j}$'s can increase, and therefore we have to stop growing all α_j such that $\alpha_j \geq D(k,j)$ - the corresponding cities become inactive and the corresponding edges are declared *tight*. The facility which causes an event of the second type is marked as being temporarily open. If an event of the first type results in a ball hitting a temporarily open facility then the corresponding edge is declared tight and the ball stops growing.

The number of events of the first type can be as large as $\Omega(n^2)$ and hence it might seem impossible to obtain a sublinear (in the size of the metric space) implementation of the algorithm of Jain and Vazirani. We get around this problem by dividing events of the first type into two categories - essential and non-essential. Events which result in a ball hitting an already open facility are declared essential - there are only $O(n)$ of these events. The remaining events are used merely to help in detecting events of the second type and are not used in the subsequent phase. We call these events inessential.

We now have to explain how we detect events of the second type and essential events of the first type. Notice that all distances are larger than $\delta L/n$ (by our preprocessing phase) and hence no events can happen till all the balls grow to a radius of $r_0 = L$. During each ball growing step, the radius of each active ball will be increased to the next discrete value.

Define a PLEB$_k$ to be a c-approximate PLEB data structure used to locate points within balls of radius $r_0 \cdot c^k$. The essential events of the first type are easy to detect - after the k-th ball growing step, add all the temporarily open facilities to a PLEB$_k$ structure. Then, for each active city, add the city to the data structure, see if there is a facility that is close enough, and then delete the city from the data structure. If a close enough facility was found, then this city will be marked inactive and the corresponding edge declared tight.

Detecting events of the second type is somewhat harder. Let X_k denote a PLEB$_k$

data structure defined on all cities which were active during the k-th ball growing step. For facility i, let $N_k(i)$ denote the number of cities in X_k which are within a distance r_k from facility i.

Lemma 19 *For any unopened facility i, $\sum_j \beta_{ij} = \sum_{k=1}^{K}(r_k - r_{k-1})N_k(i)$ after K ball growing steps.*

Proof: Let $\gamma_{i,j,k} = 1$ iff each of the following is true

1. City j is active after the k-th ball growing step
2. Facility i is unopened after the k-th ball growing step
3. $D(i,j) < r_k$

and $\gamma_{i,j,k} = 0$ otherwise. Now, after K ball growing steps, for any unopened facility i,

$$\begin{aligned}
\sum_j \beta_{ij} &= \sum_j \max\{\alpha_j - D(i,j), 0\} \\
&= \sum_j \sum_{k=1}^{K} (r_k - r_{k-1})\gamma_{ijk} \\
&= \sum_{k=1}^{K} (r_k - r_{k-1}) \sum_j \gamma_{ijk} \\
&= \sum_{k=1}^{K} (r_k - r_{k-1}) N_k(i)
\end{aligned}$$

■

Since k takes $\tilde{O}(1)$ values, we now try to estimate $N_k(i)$ in time polynomially less than n. To this end notice that $N_k(i)$ is just the number of cities within distance r_k from i. Therefore, we can apply the algorithm for approximate counting PLEB from Section 3.6. Thus the total running time of phase 1 is bounded by $\tilde{O}(n)$ c-PLEB operations.

The second phase of the algorithm by Jain and Vazirani is relatively straightforward. Let F' denote the set of temporary open facilities, T denote a graph defined by set of tight edges and let H be the square of T. The facilities in F' are ordered in the increasing order of the time they have been declared open; let $p_1 \dots p_t$ denote this sequence. Then the algorithm builds the final set of facilities, by doing the following step for $i = 1 \dots t$: if no neighbor of p_i in H has been yet added to F, p_i is added to F; otherwise, if such a neighbor exists, p_i is *not added* to F.

For our purpose this phase can be viewed as interleaved with the first phase as follows. Whenever a facility i is declared to be open, it checks if there is any city j such that the edge (i, j) is tight and j has been already assigned to a facility (initially, no city has been assigned to any facility). If such a city does not exist, then all cities j such that the edge (i, j) is tight are assigned to i and i is added to F. Otherwise, we do nothing.

It is easy to see that this process is equivalent to the one described earlier. Moreover, we can again implement it efficiently using the c-PLEB data structure. Specifically, we need it to perform the following tasks:

- check if there is any assigned city within distance a distance (say r_k) from a facility i. To this end, we create an approximate PLEB data structure which contains only assigned cities and then check if there is any point within distance r_k from i

- retrieve all unassigned cities within distance r_k from i and assign them to i. This can be done using enumerative approximate PLEB.

It is not difficult to see that the algorithm performs at most $\tilde{O}(n)$ calls to an c-PLEB data structure, and thus its running time is bounded accordingly. It remains to bound the approximation ratio. In principle, this could be done by rewriting the analysis of [JV99]. However, in order to avoid copying a significant part of that paper, we will instead show that our implementation behaves *exactly* like the algorithm of [JV99] on a slightly perturbed input. The distances in the latter will not *exactly* satisfy metric axioms, but we will show that it is not a problem for the analysis of [JV99].

The basic idea of the reduction is to represent the errors in distance computation (caused by the fact that we use approximate PLEB instead of the exact one) as adjustments to the distance matrix, and the the errors in the estimation of $N_k(i)$

as adjustments to $f(i)$. Consider the first error type. While running the algorithm, we will be specifying an *adjusted* distance function[3] $\tilde{D}(\cdot,\cdot)$ in the following way. Let $H = H(Q, S, r)$ be a c-PLEB data structure with radius r, over the set of points S and a set of queries Q (note that in our case S and Q are always subsets of the initial pointset). Recall that if $r(t)$ is the radius of a data structure used by the algorithm at time step t, then $r()$ is non-decreasing. For any $q \in Q$, let $S(q)$ be the set of all points returned by H as an answer to q during the course of the algorithm. Note that in most cases $S(q)$ is either empty (if no point has been returned) or contains only one element (an example point within distance cr from q). The only exception is the structure retrieving all unassigned cities within radius, in which case $S(q)$ can have size greater than 1.

Clearly, we have $D(q,p) \leq cr$ for any $p \in S(q)$ and $S \cap B(q,r) \subset S(q)$. We will interpret the set $S(q)$ as a set of constraints on $\tilde{D}$. Specifically:

- if $S(q) = \emptyset$, for any $p \in S$ we impose the constraint $\tilde{D}(q,p) \geq cr$ for any future assignment.
- if $S(q) = \{p\}$ and the data structure was answering simple c-PLEB query, then we set $\tilde{D}(q,p) = r$. If this assignment contradicts earlier constraint, the process fails.
- if the data structure was answering enumerative c-PLEB query, then we set $\tilde{D}(q,p) = r$ for all $p \in S(q)$ and for any $p \in S - S(q)$ we impose the constraint $\tilde{D}(q,p) \geq cr$ for any future assignment, with the above caveat.

Clearly, if the above process succeeds, we can interpret the behavior of our algorithm on $D(\cdot,\cdot)$ as the behavior of the ideal algorithm on $\tilde{D}(\cdot,\cdot)$. Therefore, it is sufficient to show that the process never fails. To this end, observe first that the constraints imposed by data structures with different values of r are consistent, due to the fact that $r(\cdot)$ is non-decreasing and the PLEB data structures are c-approximate. Therefore, we only need to make sure that the conflict does not occur between constraints imposed by two different data structures with the same radius r. If we were

[3] Note that this function is not necessarily a metric, i.e., does not have to satisfy triangle inequality, although we will make sure it is symmetric.

using the c-PLEB data structure from Section 2.2.2, we could guarantee the consistency by using the same LSH functions for all data structures with the same radius r. However, in order to make the reduction work for any data structure, we will enforce this consistency in the following way. For any query q we create a set $U(q)$, which is a union of $S(q)$'s for all data structures involving q as a query. After all queries are performed, we update the answers by using the set $U(q)$. In particular:

- if $S(q)$ is an answer to an enumerative query, we set $S(q) = U(q) \cap S$
- if $S(q) = \emptyset$ is an answer to a simple query and $U(q) \neq \emptyset$, we add arbitrary element of $U(q)$ to $S(q)$

We also apply the symmetric operation, to make sure that $\tilde{D}(\cdot,\cdot)$ is symmetric. In this way all answers produced are consistent. Note that the cost of enforcing the consistency is asymptotically subsumed by the cost of invoking c-PLEB data structures.

Now we proceed with the second type of error. To this end, observe that the comparison of $B(k) = \sum_j \beta_{k,j}$ with $f(k)$ is only performed until our approximation $\tilde{B}(k)$ of $B(k)$ is greater than $f(k)$. Moreover, $B(k)$ as a function of the time is non-decreasing and $\tilde{B}(k)$ approximates B up to a factor of $1 \pm \delta$ (with high probability). Thus, if at any point we have $\tilde{B}(k) \geq f(k)$, we set $\tilde{f}(k) = B(k)$. As a result, the behavior of our algorithm is identical to the behavior of the algorithm of [JV99] ran on in instance $< \tilde{D}(\cdot,\cdot), \tilde{f}(\cdot) >$.

If $\tilde{D}(\cdot,\cdot)$ was a metric, we would be done. Unfortunately, it does not have to be. However, we are guaranteed that $\tilde{D}(x,y) + \tilde{D}(y,z) \leq c\tilde{D}(x,z)$. By using this "approximate triangle inequality" in the proof of Lemma 6 of [JV99], we obtain approximation guarantee of $3c^2$ instead of 3. Since $\tilde{D}(\cdot,\cdot) \leq D(\cdot,\cdot) \leq c\tilde{D}(\cdot,\cdot)$, we obtain a guarantee of $3c^3$ for the whole algorithm (with respect to the original metric). Note that if $0 < \epsilon < 1$, then $c^3 = 1 + O(\epsilon)$.

Theorem 16 *Our implementation of the algorithm of Jain and Vazirani performs $\tilde{O}(n)$ calls to c-PLEB and produces a $3(1+\delta)c^3$-approximate solution, for any constant $\delta > 0$.*

Chapter 4

Embeddings

In this chapter we focus on *embeddings* between different spaces. The embeddings we use here can be roughly classified into two types: *continuous* embeddings and *threshold* embeddings. A continuous embedding between a metric space (X, D) and (X', D') with *distortion* t is a function $f : X \rightarrow X'$ such that for any two points $p, q \in X$ we have:

- $D'(f(p), f(q)) \leq D(p, q)$ (i.e., f is a contraction)
- $D'(f(p), f(q)) \geq D(p, q)/t$

A threshold embedding is parametrized by a four-tuple (r_1, r_2, r'_1, r'_2) and is a function f such that:

- if $D(p, q) \leq r_1$ then $D'(f(p), f(q)) \leq r'_1$
- if $D(p, q) > r_2$ then $D'(f(p), f(q)) > r'_2$

We will call such f an (r_1, r_2, r'_1, r'_2)-embedding.

We mention that in most cases specifying all four parameters is an overkill, so we will try to reduce their number whenever possible. In particular, if f is a contraction, we can skip both r_1 and r'_1; therefore, we will call f a (r_2, r'_2)-embedding. In this case we will call the ratio r_2/r'_2 the distortion of f.

Both of the above definitions can be generalized to the case when the embedding f is not a deterministic construct, but instead is a random variable. In this case we only require that the above guarantees are satisfied for a *fixed* pair of points p, q with certain probability (say $1-\delta$). We call it a *randomized* embedding, while we refer to the previous type of embeddings as *total*.

By using probabilistic method we can usually convert randomized embedding into total one by setting δ to be small enough. However, in many situations it is better to use randomized embedding; for example, if we want to preserve the distances only between a fixed set of points. The dimensionality reduction of Section 2.2.1 is an example of such situation, where we are only interested in distances from the (unknown) query point to the points in the database. In general, dimensionality reduction is usually used to reduce the complexity of algorithms, by replacing d by $O(\log n)$). The total embeddings, on the other hand, allow us to solve the c-PLEB problem in new metrics. Note that since the reductions for almost all problems in the previous chapter work for *any* metric, we automatically obtain solutions to those problems in the new metrics. Therefore, the total number of new results is equal to the number of algorithmic reductions *times* the number of embeddings.

Another generalization of an embedding which we consider in this chapter is *asymmetric* embedding. The difference here is that we use *two* mappings $L, R : X \to X'$ and we use $D'(L(p), R(q))$ instead of $D(f(p), f(q))$ in the host metric. This definition has been motivated by application to nearest neighbor, in which case we can apply different mappings to database points and to the query point. Note that D does not have to be symmetric, even if D' is such (hence the name).

We will use asymmetric embeddings in two scenarios. The first application of such embeddings is to design *reverse* embeddings. A reverse embedding is parametrized by a four-tuple (r_1, r_2, r'_1, r'_2) and is a pair (L, R) such that:

- if $D(p, q) \leq r_1$ then $D'(L(p), R(q)) > r'_1$
- if $D(p, q) > r_2$ then $D'(L(p), R(q)) \leq r'_2$

(note the modifed directions of inequalities). We will call such f an (r_1, r_2, r'_1, r'_2)-reverse embedding. We will use reverse embeddings to reduce (the decision version) of the r_2/r_1-approximate furthest neighbor to the $(r'_2, r'_1/r'_2)$-PLEB.

Another application of asymmetric embeddings is to embed directed graphs (especially posets) into l_∞ in the following way. Let $G = (X, E)$ be a directed graph and $M = (X', D')$. We say that $L, R : X \to X'$ embed G into M with *robustness* c, if for any $p, q \in X$:

- if $(p, q) \in E$ then $D'(L(p), R(q)) \leq 1$
- if $(p, q) \notin E$ then $D'(L(p), R(q)) \geq c$

Again, we can generalize this embedding to the *randomized* case in the analogous way.

Algorithmic applications of embeddings. Before we describe the embedding results, we will first clarify what properties an embedding has to have in order to be useful for algorithmic purposes (as we see in a moment, this issue is somewhat non-trivial). As hinted earlier, the basic application of an embedding $f : X \to X'$ is to reduce problems defined over X to problems defined over X'. Since in Chapter 3 we showed how to reduce a variety of problems to c-PLEB, it is sufficient that f allows us to solve this problem in X given a data structure solving c'-PLEB in X'. For this purpose, a threshold embedding (say with parameters (r_1, r_2, r_1', r_2')) is good enough, since if we can solve the $(r_1', r_2'/r_1')$-PLEB in X', we can automatically solve the $(r_2/r_1, r_1)$-PLEB in X. Note that in the case of randomized algorithms, if we can solve the $(r_1', r_2'/r_1')$-PLEB in X' with probability C, and the randomized embedding is correct with probability $1 - \delta$, then an operation on the resulting data structure DS for X is correct with probability at least $1 - n\delta - (1 - C)$, where n denotes the number of points in DS when the operation has been performed.

We mention however that continuous embedding is always more desirable than threshold embedding, since (a) by definition it subsumes the latter and (b) it often allows one to obtain somewhat more efficient algorithms.

Another fairly obvious but important property is that the host metric X' should be algorithmically tractable. This means that the dimension of X' should not be much larger than the dimension of X. Finally, f should be computable in time bounded by a slowly growing function (e.g., polynomial) of the size of an *individual point* of X, not the size of the whole metric. The algorithm for computing f does not have to be uniform, in the sense that we can allow the algorithm to build (during the preprocessing) an auxiliary data structure which will later allow it to perform the computation of $f(p)$ fast. Clearly, if we want the whole data structure to be dynamic, the auxiliary data structure has to be dynamic as well.

Most of the embeddings presented in this chapter have complexity which can be easily bounded by a polynomial in dimensions of the original space and the host space. The two exceptions are the results from Section 4.6.2 and Section 4.6.5. In both cases our goal is to embed the Hausdorff space over a metric M into l_∞. In the first case a natural embedding algorithm has complexity depending on certain properties of the argument, which could be exponentially larger than the number of dimensions of the original/host space. However, we will show later how to reduce this dependence to polynomial. On the other hand, in the second case, the complexity of the embedding is bounded by the polynomial of the dimensions of the original/host space *times* the cost of performing approximate nearest neighbor search in the metric M. Thus if M is tractable, the embedding is fast. Note that this result bears some similarity to reductions from Chapter 3, in the sense that we use an algorithmic subroutine for M to perform the embedding.

The types of embeddings and the relevant techniques heavily depend on the space. As mentioned earlier, we will be interested mainly in embeddings into either l_∞ or the Hamming space. We describe the results and the relevant techniques below.

Hamming space. The Hamming space is a useful for both randomized and total embeddings. Probably the most natural metric to embed into Hamming space is l_1 norm, since it is a proper generalization of Hamming space to vectors with coordinates different from $\{0, 1\}$. It is known that if the coordinates of the points from the l_1^d space belong to the set $\{0 \dots M\}$, then this metric can be embedded into dM-dimensional Hamming cube with *no* distortion. To see this, we use the function $U : \{0 \dots M\} \rightarrow \{0, 1\}^d$ which for any x produces the unary representation of x (i.e., it outputs x ones followed by $M - x$ zeros). It is immediate to see that the Hamming distance between $U(x)$ and $U(y)$ is equal to $|x - y|$. Therefore, for any two points $p, q \in \{0 \dots M\}^d$ we have $d_h(U(p), U(q)) = |p - q|_1$. We note that this embedding is usually not particularly good, since the dimensionality of the host space is very large. However, we know that the dimensionality can be considerably reduced while approximately preserving the distances. Moreover, from Fact 2 we know that if the problem we wish to solve in l_1^d is the approximate PLEB over n points (which can be used to solve other problems as in Chapter 3), then can assume $M = \log^{O(1)} n$, so in fact we don't need to go through the very high-dimensional space at all.

Since we know how to embed l_1 into Hamming space, we can now use l_1 as a host metric. In Section 4.3 we prove that l_2^d can be embedded into $l_1^{O(d)}$ with low distortion (this has been originally proved in [FLM77], but here we will give a simpler proof which also allows us to obtain additional results in Chapter 5). In fact, this result has been extended to any l_p norm for $p \in [1, 2]$ in [JS82]. Thus we can extend the algorithmic results of Chapters 2 and 3 to wide class of norms, including the Euclidean norm. Unfortunately, these embeddings use probabilistic method and therefore offer only probabilistic guarantees (although the probability of correctness is very large). To make the embeddings explicit, we will use pseudorandom generators for bounded spaces [Nis90] to derandomize them; the resulting dimensionality of host space is $d^{O(\log d)}$.

Since we can embed l_1 and l_2 norms into Hamming space and reduce the dimensionality there, in principle we do not need dimensionality reduction for neither of these norms. However, it is usually advantageous to perform dimensionality reduction in the original space, e.g., because the dimensionality reduction for Hamming cube is based on threshold embedding, as opposed to the continuous embedding. Therefore we will show how to perform dimensionality reduction in both norms, via randomized embeddings. For l_2 the dimensionality of the host space is $O(\log n/\epsilon^2)$ with high probability, if we allow $(1+\epsilon)$ distortion (this has been originally proved in [JL84]; here we give a much simpler proof which also yields better constants). For l_1 the dimensionality is higher, i.e., roughly $\log^{O(1/\epsilon)} n$. Also, in the latter case, we only guarantee the contraction for a *fixed* pair of points with a constant probability (the lower bound on all distances is still guaranteed with high probability). Finally, we give a reverse $(r, (1+\epsilon)r, (1+\epsilon')R, R)$-embedding of Hamming metric into itself, such that $\epsilon' \geq \epsilon/6$ as long as $\epsilon \leq 2$ (here we present only a computational proof). This will allow us to obtain a dynamic $(1+\epsilon)$-furthest neighbor data structure with

$O(d^{O(1)}n^{1/(1+\epsilon/6)})$ time per operation, by reducing the problem to approximate PLEB. Note that the data structure in Section 3.4 is static.

l_∞ **norm.** The l_∞ space is considerably different from Hamming space. For one, the dimensionality reduction is not possible, if we require only $(1+\epsilon)$ distortion for arbitrary $\epsilon > 0$ [Mat92]; if we want to reduce the dimension to $n^{1/t}$, the distortion needs to be at least $\Omega(t)$ [Mat92], there is also a matching upper bound [Mat92]. On the other, it is a very "hospitable" host: any n-point metric can be embedded into l_∞^n with *no* distortion [Mat92], while some metrics require distortion $\Omega(\log n)$ if we want to embed them into l_1 or l_2 [LLR94].

We exploit the usefulness of l_∞ norm by showing that Hausdorff metrics can be embedded into l_∞^d with low dimension d. Recall that the Hausdorff metrics are a general class of metric defined over *subsets* of any metric $M = (X, D)$. The Hausdorff distance D_M between any two subsets $A, B \subset X$ is defined as

$$D_M(A, B) = \max(\max_{p \in A} D(p, B), \max_{p \in B} D(p, A))$$

Hausdorff distance is of considerable practical importance, since it enables to measure distance between *sets* of points. For example, D_M for $M = l_2^2$ allows us to measure distance between two dimensional *shapes*, represented as sets of points. Therefore it is not surprising that there has been a considerable interest in computing Hausdorff distance between *two* sets of points in both computer vision and computational geometry literature (e.g., see [ABB91, AAR94] and references therein). More specifically, most of these papers aimed to compute *generalized* Hausdorff distance defined as $D_M^{\mathcal{T}}(A, B) = \min_{T \in \mathcal{T}} D_M(T(A), B)$, where $\mathcal{T}$ is a transformation group, e.g., the set of translations.

In Section 4.6 we show several results for embedding Hausdorff metrics into l_∞. We start from showing a no-distortion embedding of a Hausdorff metric $H(M) = (P(X), D_M)$ over *any* n-point metric space into l_∞ with dimension n. Note that the size of $H(M)$ is 2^n, and hence applying the general embedding theorem of [Mat92] would result in dimension exponential in n.

Although a no-distortion result is appealing, the dimension n is still too high to be algorithmically tractable. Therefore, for the case when $M = l_p^d$, we provide a threshold embedding of $H^s(M)$ into l_∞ with dimension roughly $1/\epsilon^{O(d)} \cdot s^2$, where s is the maximum size of any embedded set and we guarantee $(1+\epsilon)$ distortion. The dimension can be further reduced to roughly $1/\epsilon^{O(d)} s \log 1/\delta$ if we are satisfied with randomized embedding. Note that the cases of practical of interest are $d = 2$ and $d = 3$, and therefore the dimension of the host space is fairly low.

Then we investigate Hausdorff metrics over general metric spaces. In particular, we will show that for *any* finite metric $M = (X, D)$ the Hausdorff metric $H^s(M)$ can be threshold embedded into l_∞^d with $d = |X|^\alpha s^{O(1)}$ for any $\alpha > 0$, with constant distortion.

Finally, in Section 4.7 we show several asymmetric embeddings into l_∞. In particular, we will first show that a poset $(P(X), \subset)$ can be embedded into $l_\infty^{|X|}$ with robustness 3. This immediately implies that any c-PLEB algorithm for l_∞^d gives an algorithm for the subset query problem with the same parameters. Since the latter problem is conjectured to be hard, the former problem seems hard as well.

We complement the above result with Theorem 24 which implies that the *exact* PLEB problem (and in fact the nearest neighbor problem as well) for l_∞^d can be reduced to subset query problem for subsets of $\{1 \ldots O(d^2 \log^2 n)\}$. Thus, if we can solve c-PLEB in l_∞ efficiently for $c < 3$, we can solve it for $c = 1$ as well !

We also show a robust embedding result for Hausdorff metric. In particular, we show that a poset $(P_s(X), \subset)$ can be embedded into $H^s(l_\infty^{\log|X|})$ with robustness 3. This also implies a hardness result similar to the l_∞ case. Moreover, we show that a robust embedding of $(P_s(X), \subset)$ into l_∞ with low dimension implies the existence of superimposed codes with small length. By applying a lower bound for superimposed code length (cf. Preliminaries) we conclude that any embedding of $H^s(l_\infty^k)$ into l_∞^d for $2^k > \mathrm{poly}(s)$ requires $d = \Omega(s^2 k / \log s)$. Thus we obtain a superlinear lower bound for the dimension of l_∞ needed to embed Hausdorff space. Note that both the upper and lower bounds use superimposed codes in an explicit way.

4.1 Dimensionality reduction in Hamming metric

We will start by taking a closer look at the reduction from Section 2.2.1. There, we essentially showed how to achieve a randomized $(r, cr, r'-1, r')$-embedding, where $c = 1+\epsilon$, for $\epsilon < 1$. Although it was sufficient for the purpose of the algorithm from that section, in general the $r'-1 : r'$ gap in the host space is not large enough to apply any approximate algorithm. Therefore, we will aim to achieve a gap as close to $(1+\epsilon)$ as possible.

To this end, we will apply essentially the same technique as before, but with a different analysis. Specifically, we will give a randomized embedding from $\{0,1\}^d$ to Σ^d (also equipped with Hamming metric), where Σ is an alphabet with possibly unbounded size. This result will have further applications in Section 5.3.

The mapping is defined $g(p) = g_1(p) \ldots g_{d'}(p)$ such that $g_i(p) = p_{|I_i}$. The multisets I_i are defined exactly as in Section 2.2.1; in particular their size is $k = \alpha r/d$. Also from that section we know that if $D(p,q) = \delta r$, then

$$\Pr[g_i(p) = g_i(q)] = Q(\delta)$$

where

$$e^{-\alpha\delta}(1-\eta) \leq Q(\delta) \leq e^{-\alpha\delta}$$

and

$$\eta = \alpha\delta r/d$$

Therefore

$$Q_1 = 1 - Q(1) \leq 1 - e^{-\alpha}(1-\eta) \leq 1 - e^{-\alpha} + \eta \leq \alpha + \eta$$

and

$$Q_2 = 1 - Q(c) \geq 1 - e^{-\alpha c} \geq \alpha c - (\alpha c)^2/2 = \alpha c(1 - \alpha c/2)$$

Set $\eta = \alpha \cdot \gamma$ and $\alpha = \gamma$. Then $Q_1 \leq \alpha(1+\gamma)$ and $Q_2 \geq \alpha c(1-\gamma)$, if we assume $c \leq 2$. Note that the gap $Q_2 : Q_1$ is almost as big as $c : 1$. Therefore, by using Chernoff bound, if we set $d' = O(\log(1/\beta)/\gamma^3)$, then with probability $1-\beta$ we have

$$D(g(p), g(q)) \leq d'\alpha(1+2\gamma), \text{ if } D(p,q) \leq r$$

and

$$D(g(p), g(q)) > d'\alpha c(1-2\gamma), \text{ if } D(p,q) > cr$$

Theorem 17 *For any $\epsilon \in (0,1]$, there exists a randomized $(r, (1+\epsilon)r, d'\alpha(1+\gamma), d'\alpha(1+\epsilon)(1-\gamma))$-embedding of d-dimensional Hamming cube into a Hamming metric over $\Sigma^{d'}$ for $d' = O(\log(1/\beta)/\gamma^3)$, with the probability of correctness $1-\beta$.*

By replacing each element $a \in \Sigma$ by a binary codeword $C(a)$, where $C(\cdot)$ is a sequence of $\lambda = O(\log(1/\beta)/\gamma^2)$ random bits, we obtain a randomized $(r, (1+\epsilon)r, d'\alpha(1+\gamma)^2\lambda/2, d'\alpha(1+\epsilon)(1-\gamma)^2\lambda/2)$ embedding into $\{0,1\}^{d'\lambda}$. The correctness follows easily from the Chernoff bound.

4.2 Dimensionality reduction in l_2

In this section we show a randomized embedding of l_2^d into $l_2^{d'}$, where d' is smaller than d. In particular, we will present the proof of the following lemma (this proof first appeared in [IM98], see also [DG99]).

Lemma 20 *Let u be a unit*[1] *vector in $\Re^d$. For any even positive integer k, let $U_1, \ldots, U_k$ be random vectors chosen independently from the d-dimensional Gaussian distribution*[2] *$N^d(0,1)$. For $X_i = u \cdot U_i$, define $W = W(u) = (X_1, \ldots, X_k)$ and $L = L(u) = |W|^2$. Then, for any $\beta > 1$,*

1. $E(L) = k$,
2. $\Pr[L \geq \beta k] < O(k) \times \exp(-\frac{k}{2}(\beta - (1+\ln\beta)))$,
3. $\Pr[L \leq k/\beta] < O(k) \times \exp(-\frac{k}{2}(\beta^{-1} - (1-\ln\beta)))$.

Before giving the proof, we show how to interpret the above inequalities in a somewhat more intuitive way. Let $\beta = 1+\epsilon$, $\epsilon > 0$. From Taylor expansion we know

[1] *Note that by norm properties the following holds in fact for* any *vector u, not necessarily of unit length.*

[2] *Each component is chosen independently from the standard normal distribution $N(0,1)$.*

that $\ln \beta \le \epsilon - \epsilon^2/2 + \epsilon^3/3$. By plugging it into the upper bound for $\Pr[L \ge \beta k]$ we get

$$\begin{aligned}\Pr[L \ge \beta k] &\le O(k) \times \exp(-\frac{k}{2}(\beta - 1 - \epsilon + \epsilon^2/2 - \epsilon^3/3)) \\ &\le O(k) \times \exp(-\frac{k}{2}(\epsilon^2/2 - \epsilon^3/3))\end{aligned}$$

On the other hand, we also know that $\ln \beta \ge \epsilon - \epsilon^2/2$. Therefore

$$\begin{aligned}\Pr[L \le k/\beta] &\le O(k) \times \exp(-\frac{k}{2}(\beta^{-1} - 1 + \epsilon - \epsilon^2/2)) \\ &\le O(k) \times \exp(-\frac{k}{2}(\frac{1}{1+\epsilon} - 1 + \epsilon - \epsilon^2/2)) \\ &\le O(k) \times \exp(-\frac{k}{2}(\frac{\epsilon^2}{1+\epsilon} - \epsilon^2/2)) \\ &\le O(k) \times \exp(-\frac{k}{2} \cdot \frac{\epsilon^2 - \epsilon^3}{2(1+\epsilon)})\end{aligned}$$

Thus, the probability that a given unit vector gets distorted by more than $(1+\epsilon)$ in any direction[3] grows roughly as $\exp(-k\epsilon^2/4)$, for small $\epsilon > 0$. Therefore, if we are given a set P of n points in l_2, we can set k to roughly $8\ln(n)/\epsilon^2$ and make sure that with non-zero probability we obtain projection which does not distort distances[4] between *any* two different points from P by more than $(1+\epsilon)$ in each direction. This is (up to the constants) the statement of *Johnson-Lindenstrauss* lemma [JL84, FM88]. We mention that our constant 8 yields a slight improvement over the earlier constant 9 obtained in [FM88].

Now we proceed with the proof of Lemma 20.

Proof: By the spherical symmetry of $N^d(0,1)$ each X_i is distributed as $N(0,1)$ [Fel91, page 77]. Define $Y_i = X_{2i-1}^2 + X_{2i}^2$, for $i = 1, \ldots, k/2$. Then, Y_i follows the exponential distribution with parameter $\lambda = \frac{1}{2}$ (see [Fel91, page 47]). Thus $E(L) = \sum_{i=1}^{k/2} E(Y_i) = (k/2) \times 2 = k$; also one can see that L follows the Gamma distribution with parameters $\alpha = \frac{1}{2}$ and $v = k/2$ (see [Fel91, page 46]). Since this distribution is a dual of the

[3] Note that this implies distortion $(1+\epsilon)^2$ if we require the mapping to be a contraction.

[4] In fact, this statement holds even for the *square* of the distances.

Poisson distribution, we obtain that

$$\Pr[L \geq \beta k] = \Pr[P_{\beta k}^{1/2} \leq v-1],$$

where P_t^α is a random variable following the Poisson distribution with parameter αt. Clearly

$$\Pr[P_t^\alpha \leq v-1] = \sum_{i=0}^{v-1} e^{-\alpha t}\frac{(\alpha t)^i}{i!}$$

and therefore

$$\begin{aligned}
\Pr[L \geq \beta k] &= \sum_{i=0}^{v-1} e^{-\beta v}\frac{(\beta v)^i}{i!} \\
&\leq v e^{-\beta v}\frac{(\beta v)^v}{v!} \\
&\leq v e^{-\beta v}\frac{(\beta v)^v}{\frac{v^v}{e^v}} \\
&= v(e^{-\beta}\beta e)^v = v e^{-v(\beta-(1+\ln\beta)}
\end{aligned}$$

which implies the desired result since $v = k/2$.

Finally, we claim the following bound, for some large constant $\gamma \gg 1$.

$$\begin{aligned}
\Pr[L \leq k/\beta] &= \sum_{i=v}^{\infty} e^{-v/\beta}\frac{(v/\beta)^i}{i!} \\
&\leq e^{-v/\beta}\sum_{i=v}^{\infty}\left(\frac{ev}{i\beta}\right)^i \\
&= e^{-v/\beta}\left[\sum_{i=v}^{\gamma ev/\beta}\left(\frac{ev}{i\beta}\right)^i + \sum_{i=\gamma ev/\beta+1}^{\infty}\left(\frac{ev}{i\beta}\right)^i\right]
\end{aligned}$$

The second sum is very small for $\gamma \gg 1$ and we bound only the first one. As the sequence $(\frac{ev}{i\beta})^i$ is decreasing for $i \geq v/\beta$, we can bound the first sum by

$$\left(\frac{\gamma ev}{\beta}\right) e^{-v/\beta}\left(\frac{e}{\beta}\right)^v = O(v)e^{-v(\beta^{-1}-(1-\ln\beta))}.$$

Since $v = k/2$, we obtain the desired result. ∎

4.3 Embedding of l_2 into l_1

In this section we show that *exactly* the same embedding as used in previous section can be used to embed l_2 into l_1. In order to do this, we need to prove an analog of Lemma 20 for $L = L(u) = |W|_1 = \sum_i |X_i|$ (as opposed to $L(u) = |W|_2$); this proof first appeared in [Ind00b]. In particular, we show:

Lemma 21 *There exists a constant $A > 0$ such that*

1. $E(L) = k\sqrt{2/\pi}$
2. *For $0 < \epsilon < 1$ we have*

$$\Pr[L \geq (1+\epsilon)E[L]] \leq \exp(-k\epsilon^2 A)$$

and

$$\Pr[L \leq (1-\epsilon)E[L]] \leq \exp(-k\epsilon^2 A)$$

Note that, modulo the constant A (which has not been optimized here) the dependence of the tail bound on k and ϵ is the same as in the previous section.

Proof: Firstly, recall that all X_i's are distributed according to $N(0,1)$. Let X be a random variable with $N(0,1)$ distribution. Clearly

$$E[|X|] = 1/\sqrt{2\pi}\int_{-\infty}^{\infty} x\exp(-x^2/2)dx = 1/\sqrt{2\pi}\cdot 2 = \sqrt{2/\pi}$$

By linearity of expectation, we get $E[L] = k\sqrt{2/\pi}$.

The sharp concentration of L around its mean is, unfortunately, somewhat more complicated than in the l_2 case, mainly due to the fact that the distribution of L does not seem to have a closed form. Therefore, we use exponential moment inequalities to upper bound the tail of L's distribution.

Let $\beta = 1+\epsilon$. Then for any $a > 0$

$$\begin{aligned}\Pr[\sum|X_i| \geq \beta E[\sum|X_i|]] &\leq \Pr[\exp(a\sum|X_i|) \geq \exp(a\beta E[\sum|X_i|])] \\ &\leq \frac{E[\exp(a\sum|X_i|)]}{\exp(a\beta E[\sum|X_i|])}\end{aligned}$$

$$
\begin{aligned}
&= \left(\frac{E[\exp(aX)]}{\exp(a\beta\sqrt{2/\pi})}\right)^k \\
&= (N/D)^k
\end{aligned}
$$

By using Taylor expansion we know that $D \geq 1 + a\beta\sqrt{2/\pi}$. It is sufficient to upper bound $N = \sqrt{2/\pi}\int_0^\infty \exp(ax)\exp(-x^2/2)dx$. To this end, we split N into the sum of N_1 and N_2, where for a $z > 0$ defined later we have

$$
\begin{aligned}
N_1 &= \sqrt{2/\pi}\int_z^\infty \exp(ax)\exp(-x^2/2)dx \\
&= \sqrt{2/\pi}\int_0^\infty \exp(ax+az)\exp(-(x+z)^2/2)dx \\
&\leq \sqrt{2/\pi}\exp(az - z^2/2)\int_0^\infty \exp(ax)\exp(-x^2/2)dx \\
&\leq \exp(az - z^2/2)N
\end{aligned}
$$

Since $\exp(az - z^2/2)$ is small for large z, we can focus only on the initial part of the integral N, namely $N_2 = \sqrt{2/\pi}\int_0^z \exp(ax)\exp(-x^2/2)dx$. In particular, let $z = \frac{1}{2a}$. Then for all $x \leq z$ we have $|ax| \leq 1/2$. Therefore, we know that $\exp(ax) \leq 1 + ax + (ax)^2$. Hence we can write

$$
\begin{aligned}
N_2 &\leq \sqrt{2/\pi}\left(\int_0^z \exp(-x^2/2)dx + \int_0^z ax\exp(-x^2/2)dx + \int_0^z (ax)^2\exp(-x^2/2)\right)dx \\
&\leq \sqrt{2/\pi}(\sqrt{\pi/2} + a + a^2 \cdot I) \\
&= 1 + \sqrt{2/\pi}a + \sqrt{2/\pi}I \cdot a^2
\end{aligned}
$$

where $I = \int_0^z x^2\exp(-x^2/2)dx$ is upper bounded by a constant. Thus we can bound

$$
\begin{aligned}
N_2/D &\leq \frac{1 + \sqrt{2/\pi}a + \sqrt{2/\pi}I \cdot a^2}{1 + a\beta\sqrt{2/\pi}} \\
&= \frac{1 + \sqrt{2/\pi}a + \sqrt{2/\pi}I \cdot a^2}{1 + \sqrt{2/\pi}a + \sqrt{2/\pi}\epsilon a} \\
&= 1 - \frac{\sqrt{2/\pi}a(\epsilon - Ia)}{1 + \sqrt{2/\pi}a + \sqrt{2/\pi}\epsilon a}
\end{aligned}
$$

If we set $a = \frac{\epsilon}{2I}$ then

$$N_2/D \leq 1 - \frac{\sqrt{2/\pi}a(\epsilon/2)}{1+\sqrt{2/\pi}a+\sqrt{2/\pi}\epsilon a}$$

Recall that $N = N_1 + N_2 = \exp(az - z^2/2)N + N_2$, and thus $N = N_2/(1-\exp(az - z^2/2))$. Also $z = \frac{1}{2a} = I/\epsilon$ and therefore

$$\begin{aligned} N/D &\leq \left(1 - \frac{\sqrt{2/\pi}\frac{\epsilon}{2I}(\epsilon/2)}{1+\sqrt{2/\pi}\frac{\epsilon}{2I}+\sqrt{2/\pi}\epsilon\frac{\epsilon}{2I}}\right) / \left(1 - \exp(1/2 - \frac{I^2}{2\epsilon^2})\right) \\ &\leq (1 - C_1\epsilon^2)/(1 - \exp(1/2 - C_2/\epsilon^2)) \\ &\leq (1 - C_3\epsilon^2) \end{aligned}$$

Therefore

$$\Pr[\sum |X_i| \geq \beta E[\sum |X_i|]] \leq (1 - C_3\epsilon^2)^k$$

which was to be shown.

The second inequality can be proved in *exactly* the same way as shown above, with the difference that $a < 0$ and $\beta = 1 - \epsilon$. ■

This concludes the proof for the randomized embedding. To obtain a total embedding of l_2 into l_1 we derandomize the randomized embedding by using the following well-known fact (cf. [FLM77], Lemma 2.4):

Fact 5 *For any $\epsilon > 0$ and dimension d there exists a set $P \subset \Re^d$ of cardinality $\exp(O(d\log(1/\epsilon)))$ such that if a linear embedding $f : P \to l_1^k$ has distortion $1+\epsilon/2$, then the extension of f to l_2^d has distortion $(1+\epsilon)$.*

The proof proceeds essentially by focusing on unit vectors only (since the mapping is linear) and observing that it is enough to obtain a low-distortion embedding of a dense covering of a unit sphere.

Now, if we set $k = C\log|P|/\epsilon^2$, then (from Lemma 21) we have a constant probability of obtaining an embedding of P to l_1^k with distortion $(1+\epsilon/2)$. Therefore

Lemma 22 *For any $0 < \epsilon < 1$ there exists an embedding of l_2^d into $l_1^{O(d/\epsilon^2)}$ with distortion $(1+\epsilon)$.*

4.4 Explicit embedding of l_2 into l_1

The embedding from the previous section has been obtained using probabilistic method. This is enough if we want to use it to get randomized algorithm. However, in order to use it for deterministic algorithm, we need to construct the embedding in a deterministic way. The simplest way of doing it is to enumerate all $d \times O(d)$ (properly discretized) matrices and choose the one which results in best embedding. Unfortunately, this would take time exponential in d^2, while we would like to get time as close to poly(d) as possible. Ideally, the embedding should be explicit.[5] In this section we show two results of this type (these results first appeared in [Ind00b]). The first one gives an embedding of l_2^d into $l_1^{O(\log d)}$ with distortion $1 + 1/d^{\Theta(1)}$; the embedding is described by a closed-form formula. The second result reduces the number of random bits used by the embedding from the previous section from about d^2 to $O(d \log d)$. As a result, we can (a) represent the embedding using only $O(d \log d)$ bits, and (b) find the right embedding by exhaustive enumeration in time $\exp(d \log d)$, as opposed to $\exp(d^2)$. We mention that an explicit embedding of l_2^d into $l_1^{d^{O(1)}}$ with constant (but not arbitrarily small) distortion has been known [Ber97, LLR94].

Both results are obtained by derandomizing the embedding from the previous section using *pseudorandom generators (PRGs)*. As in [Nis90] we consider PRGs which fool any finite state machine (FSM) which uses at most $O(S)$ bits of space (or $2^{O(S)}$ states). Assume that a FSM $Q \in \text{space}(S)$ uses at most k chunks of random bits, where each chunk is of length b. The generator $G : \{0,1\}^m \to (\{0,1\}^b)^k$ expands a "small number" m of "truly random" bits into kb bits which "look random" for Q. Formally, it is defined as follows. Let $\mathcal{D}^t$ be a uniform distribution over $\{0,1\}^t$. For any (discrete) random variable X let $\mathcal{D}[X]$ be the distribution of X, interpreted as a vector of probabilities. Let $Q(x)$ denote the state of Q after using the random bits sequence x. Then we say that G is a PRG *with parameter* $\epsilon > 0$ for a class $\mathcal{C}$ of FSMs, if for every $Q \in \mathcal{C}$

$$|\mathcal{D}[Q_{x \in \mathcal{D}^{bk}}(x)] - \mathcal{D}[Q_{x \in \mathcal{D}^m}(G(x))]|_1 \leq \epsilon$$

[5] We skip here a "definition" of which construction we consider explicit and which not, hoping that the readers intuition is compatible to the author's

where $|y|_1$ denotes an l_1 norm of a vector y.

Fact 6 ([Nis90]) *There exists a PRG G for space(S) with parameter $\epsilon = 2^{-O(S)}$ such that:*

- *G expands $O(S \log R)$ bits into $O(R)$ bits*
- *G requires only $O(S)$ bits of storage (in addition to its random input)*
- *any length-$O(S)$ chunk of $G(x)$ can be computed using $O(\log R)$ arithmetic operations on $O(S)$-bit words*

Now we proceed with the embedding.

Lemma 23 *There exists an explicit embedding of l_2^d into $l_1^{d^{O(\log d)}}$ with $(1 + 1/d^{O(1)})$ distortion.*

Proof: We start from illustrating the embedding by providing an intuitive (although not exactly formal) embedding of l_2^d into l_1 with infinite dimension. Let $X_1 \dots X_d$ be a sequence of i.i.d. random variables with Gaussian distribution. Then, from Lemma 21 we have that for any $p = (u_1, \dots, u_d) \in l_2^d$

$$E[|\sum_i u_i X_i|] = \sqrt{2/\pi}|p|_2$$

Thus if we create an "infinite matrix" A with n columns and "infinite" number of rows, one for "each" configuration of $(X_1, \dots, X_d)$, then $|Ap|_1$ is "proportional" to $|p|_2$, which is what we need.

To reduce the dimension of the host space, observe that the above linear mapping essentially can be computed using $O(\log n)$ bits with precision $(1 + 1/d^c)$ for any $c = O(1)$, if the random variables X_i also have $O(\log d)$ bits of precision. In order to make sure that the limited precision does not jeopardize the correctness of embedding, we observe the following. Let X_i' be i.i.d. variables having the "truncated Gaussian" distribution, i.e. such that:

- if $|X_i| \leq t$, then $X_i' = X_i$

- if $|X_i| > t$, then $X'_i = 0$

We use $t = 2c\sqrt{\log d}$, so $\Pr[|X_i| > t] \leq a/d^c$, for some $a > 0$. We will relate $E[\sum_i X_i u_i]$ and $E[\sum_i X'_i u_i]$ as follows. Let $p = \Pr[\exists i : |X_i| > t]$; notice that $p \leq a/d^{c-1}$, i.e. is small. Then we can write

$$\begin{aligned} E &= E[\sum_i X_i u_i] \\ &= (1-p)E[|\sum_i X_i u_i| : \forall i |X_i| \leq t] + pE[|\sum_i X_i u_i| : \exists i |X_i| > t] \\ &= (1-p)E_1 + pE_2 \end{aligned}$$

and

$$\begin{aligned} E' &= E[\sum_i X_i u_i] \\ &= (1-p)E[|\sum_i X'_i u_i| : \forall i X'_i \neq 0] + pE[|\sum_i X'_i u_i| : \exists i X'_i = 0] \\ &= (1-p)E'_1 + pE'_2 \end{aligned}$$

Notice that $E_1 = E'_1$. Moreover, it is easy to see that $E_2 = O(dt)$ and $E'_2 = O(dt)$. Thus E and E' differ only by a factor of $(1 + 1/d^{\Theta(1)})$.

It is easy to see that limiting the precision of X'_i's and u_i's to $O(\log n)$ introduces only a $(1 + 1/d^{\Theta(1)})$ factor in distortion. Thus we can compute the embedding using only $O(\log n)$ bits of storage. Therefore, we can use Fact 6 to reduce the total number of configurations of random variables to $d^{O(\log d)}$.

■

Embedding of l_2^n into $l_2^{O_\epsilon(n)}$ with $O_\epsilon(n \log^2 n)$ non-uniform bits and $(1+\epsilon)$ distortion. We follow the scheme from the previous section. It is easy to see that each row of A can be in fact generated using small random seed of length $O(\log n)$ bits by using PRG. Thus, we need only $O(n \log^2 n)$ bits (for $1/\epsilon = O(1)$) to represent A.

4.5 Dimensionality reduction in l_1

In this section we show how to obtain an embedding which maps an n-element point set $P \subset l_1^d$ into l_1^k, where k is polylogarithmic in n. Specifically, we prove the following

Theorem (first showed in [Ind00b]):

Theorem 18 *For any $\epsilon, \delta > 0$, there is a probability space $\mathcal{D}$ over linear mappings $f : l_1^d \to l_1^k$, where $k = (\ln(1/\delta)/\epsilon)^{O(1/\epsilon)}$, such that for any point $p \in l_1^d$ we have:*

- *the probability that $|f(p)|_1 \leq |p|_1$ is smaller than δ*
- *the probability that $|f(p)|_1 \geq (1+\epsilon)|p|_1$ is smaller than $1-\epsilon$*

Note that the embedding is randomized but asymmetric: the probability of small expansion is only ϵ, while the probability of small contraction is $1-\delta$.

Proof: The basic idea of the proof is (again) to use linear mapping using a matrix whose coefficients are chosen independently at random from certain distribution $\mathcal{D}$. However, we have to choose the distribution quite carefully, to make sure that a product of $(X_1 \dots X_d)$ and a unit vector u (i.e., such that $|u|_1 = 1$) is more or less the same, independently on u. In other words, we would like to have the property that if $u = (a_1 \dots a_d)$ and both $X_1 \dots X_d$ and X independent random random variables from $\mathcal{D}$, then $\sum_i a_i X_i$ has the same distribution as $(\sum_i |a_i|)X$. It turns out that there exists essentially only *one* such distribution, namely *Cauchy distribution.* In general, a distribution $\mathcal{D}$ over $\Re$ is called *p-stable*, if there exists $p \geq 0$ such that for any d real numbers $a_1 \dots a_d$ and i.i.d. variables $X_1 \dots X_d$ variables with distribution $\mathcal{D}$, the random variable $\sum_i a_i X_i$ has the same distribution as the variable $(\sum_i |a_i|^p)^{1/p} X$, where X is a random variable with distribution $\mathcal{D}$. It is known [Zol86] that stable distributions exist for any $p \in (0, 2]$. In particular:

- a *Cauchy distribution* $\mathcal{D}_C$, defined by the density function $c(x) = \frac{1}{\pi}\frac{1}{1+x^2}$, is 1-stable
- a *Gaussian (normal) distribution* $\mathcal{D}_G$, defined by the density function $g(x) = \frac{1}{\sqrt{2\pi}} e^{-x^2/2}$, is 2-stable

We define the random mapping f such that for $j = 1 \dots k$ the jth coordinate of $f((a_1, \dots, a_d))$ is equal to $Y_j = \sum_i X_i^j a_i$, where X_i^j are i.i.d random variables having Cauchy distribution. Again, since f is linear, we can assume $|p|_1 = 1$. Since the

Cauchy distribution is 1-stable, each Y_j has a Cauchy distribution. Thus it is sufficient to prove the following fact: for any sequence $Y_1 \dots Y_k$ of i.i.d. variables with Cauchy distribution, let $Y = \sum_j |Y_i|$. Show that there exists a *threshold* $T = T(m, \delta, \epsilon)$, such that:

- $\Pr[Y < (1-\epsilon)^c T] \le \delta$, for some $c = O(1)$
- $\Pr[Y > (1+\epsilon)T] \le \frac{1+\epsilon/2}{1+\epsilon}$

We will first establish T which is "good" for the second condition. Let $U = a \cdot k$, for some $a \ge 1$. Since (what is easy to verify) $\Pr[|Y_i| > t] \le b/t$ for some $b = O(1)$, by the "union bound" we have $\Pr[\exists_i |Y_i| \ge U] \le kb/U = b/a$. We define

$$T = E[Y : \forall_i |Y_i| \le U]$$

By the linearity of expectation

$$\begin{aligned} T &= \sum_i E[|Y_i| : |Y_i| \le U] \\ &\le \frac{k\frac{2}{\pi}\int_0^U \frac{x}{1+x^2}dx}{1-b/a} \\ &\le \frac{2}{\pi} k \frac{\ln(U^2+1)}{2(1-b/a)} \end{aligned}$$

Set $b/a = \frac{\epsilon/2}{1+\epsilon}$. Then

$$\begin{aligned} & \Pr[Y \ge (1+\epsilon)T] \\ \le\ & \Pr[\exists_i |Y_i| > U] \\ +\ & \Pr[Y \ge (1+\epsilon)T : \forall_i |Y_i| \le U] \\ \le\ & b/a + \frac{1}{1+\epsilon} \\ =\ & \frac{1+\epsilon/2}{1+\epsilon} \end{aligned}$$

where the second last inequality is an application of Markov inequality.

Now we need to show the first condition holds. Define $I_i = [(1+\epsilon)^{i-1}, (1+\epsilon)^i]$ for $i \geq 1$. Let $p_i = \Pr[Y_1 \in I_i]$. We can bound p_i from below as follows:

$$\begin{aligned}
p_i &= \frac{2}{\pi}\int_{(1+\epsilon)^{i-1}}^{(1+\epsilon)^i} \frac{1}{1+x^2}dx \\
&\geq \frac{2}{\pi}[(1+\epsilon)^i - (1+\epsilon)^{i-1}] \cdot \frac{1}{1+(1+\epsilon)^{2i}} \\
&= \frac{2}{\pi}\frac{\epsilon(1+\epsilon)^{i-1}}{1+(1+\epsilon)^{2i}}
\end{aligned}$$

Let n_i be the number of Y_i's falling to the interval I_i. In the following we introduce a parameter $L \geq 1$ such that for all $i \leq l = \log_{1+\epsilon} L$ we have $n_i \geq (1-\epsilon)p_i k$. Note that this implies the following lower bound T' for $|Y|$

$$\begin{aligned}
T' &\geq \sum_{i=1}^{l}(1+\epsilon)^{i-1}n_i \\
&\geq \sum_{i=1}^{l}(1+\epsilon)^{i-1}(1-\epsilon)p_i k \\
&\geq \frac{2}{\pi}k(1-\epsilon)\epsilon\sum_{i=1}^{l}\frac{(1+\epsilon)^{i-1}\cdot(1+\epsilon)^{i-1}}{1+(1+\epsilon)^{2i}} \\
&= \frac{2}{\pi}k\frac{1-\epsilon}{(1+\epsilon)^2}\epsilon\sum_{i=1}^{l}\frac{(1+\epsilon)^{2i}}{1+(1+\epsilon)^{2i}} \\
&\geq \frac{2}{\pi}k\frac{1-\epsilon}{(1+\epsilon)^2}\epsilon(l - (\log_{1+\epsilon} 1/\epsilon)/2)\frac{1}{1+\epsilon} \\
&\geq \frac{2}{\pi}k\frac{1-\epsilon}{(1+\epsilon)^3}\epsilon((\ln L)/\epsilon - (\log_{1+\epsilon} 1/\epsilon)/2) \\
&\geq \frac{2}{\pi}k\frac{1-\epsilon}{(1+\epsilon)^3}(\ln L - \epsilon(\log_{1+\epsilon} 1/\epsilon)/2)
\end{aligned}$$

We will see later that

$$\ln L \geq (\log_{1+\epsilon} 1/\epsilon)/2 \tag{4.1}$$

Therefore, the final lower bound for T' is $\frac{2}{\pi}(1-\epsilon)^5 k \ln L$.

We will make L as close to U as possible (so that T' is close to T). However, we have to make sure that for all $i \leq l$ we have $n_i \geq (1-\epsilon)p_i k$ with probability $1-\delta$. To this end, notice that if $i \leq l$ then $p_i \geq q = \frac{2}{\pi}\frac{\epsilon}{1+L(1+\epsilon)}$. Therefore, by Chernoff bound, we just need to make sure that

$$\exp(-\epsilon^2/3 \cdot qk) \leq \delta/l$$

The latter condition can be rewritten as

$$k \geq 3/\epsilon^3 \cdot 1/q \cdot (\ln(1/\delta) + \ln l)$$

Substituting for q we get

$$k \geq 3/\epsilon^3 \cdot \pi/2 \cdot (1 + L(1+\epsilon))[\ln \ln_{1+\epsilon} L + \ln 1/\delta]$$

Again, we will see later that

$$\ln \ln_{1+\epsilon} L \quad \leq \quad \ln 1/\delta \tag{4.2}$$

Since also $L > 1/\epsilon$, we get that it is sufficient to make sure that

$$k \geq 3\pi/\epsilon^3(1+\epsilon)^2 L \ln 1/\delta$$

which can be satisfied by setting

$$L = \frac{\epsilon^3}{3\pi(1+\epsilon)^2} k/\ln(1/\delta) = Ck/\ln(1/\delta)$$

Finally, in order to show that $T' \geq T(1-\epsilon)^f$ for some $f = O(1)$, it is sufficient to ensure that $\ln U \leq (1+\epsilon)\ln L$. Substituting for U and L we get the constraint

$$\ln(ak) \leq \ln[Ck/\ln(1/\delta)](1+\epsilon)$$

which solves to

$$k \geq [a(\frac{\ln 1/\delta}{C})^{1+\epsilon}]^{1/\epsilon}.$$

Notice that if we take (without loss of generality) $1/\delta = 1/\epsilon^{\Omega(1)}$, that both inequalities we assume on the way are satisfied. ∎

4.6 Embedding of Hausdorff metric into l_∞

In this section we investigate the embeddings into the second available "algorithmically tractable" norm, namely l_∞. The results described here appeared first in [FCI99].

We use $H^s(M)$ to denote a Hausdorff metric over metric M in which all sets are of size at most s. Also, we use the notation $H_p^d = (\Re^d, D_p^d)$ to denote $H(l_p^d)$.

4.6.1 Isometric embedding of $H(M)$ into l_∞

We begin by showing a 1-embedding into l_∞. Our approximate embeddings will be based on this exact embedding. It is well known that any metric $M = (X, D)$ can be 1-embedded into $l_\infty^{|X|}$. Thus any Hausdorff metric $H(M)$ can be 1-embedded into l_∞, with the number of dimensions equal to the number of sets. Here, we show that much fewer dimensions suffice.

Theorem 19 *For any finite metric $M = (X, D)$ the space $H(M)$ can be embedded into $l_\infty^{|X|}$ with no distortion.*

Proof: Assume $X = \{p_1, \ldots, p_n\}$. For any $S \subset X$, the value $f(S)$ is defined as

$$f(S) = (D(p_1, S), \ldots, D(p_n, S)).$$

Notice that this mapping is a contraction; therefore it is sufficient to show $|f(S) - f(S')|_\infty \geq 1$ for any $S, S' \subset X$ such that $D_M(S, S') \geq 1$. To this end note that if $D_M(S, S') = t$, then there exists $p \in S$ such that $D(p, S') = t$ (or $p' \in S'$ such that $D(p', S) = t$, we will assume the first case without loss of generality). Then

$$|f(S) - f(S')|_\infty \geq |D(p, S) - D(p, S')| \geq |0 - t| = t,$$

thus establishing the claim. ∎

4.6.2 Threshold embedding of $H^s(l_\infty^d)$ into l_∞

In this section we modify the techniques used above to obtain embeddings which are *approximate* but which require smaller number of dimensions. More specifically:

Theorem 20 *For any integer $I \geq 2$, $s > 0$ and $0 < c < 1$ the space $H^s([0,I]^d_\infty)$ can be $(1,c)$-embedded into l^E_∞ where*

$$E = O\left(ds^2 \left\lceil \frac{1+c}{1-c} \right\rceil^{2d} \left(\log^2 I + \log^2 \frac{1}{1-c}\right)\right).$$

Proof: We need a mapping $f : H^s([0,I]^d_\infty) \to l^E_\infty$ such that for any $A, B \subset [0,I]^d$ of cardinality s and $p \in A$ such that $D(p,B) \geq 1$ we have $|f(A) - f(B)|_\infty \geq c$. The embedding will be defined by a sequence of E subsets $S_1 \dots S_E$ of $[0,I]^d$ as

$$f(A) = (D(A,S_1), \dots, D(A,S_E)).$$

Firstly, observe that f is a contraction. To prove this, we need to show that $|D(A,S) - D(B,S)| \leq D_M(A,B)$. Assume $D(A,S) - D(B,S) \geq 0$ (the opposite case is symmetric). In this case it is sufficient to show that $D(A,S) \leq D(B,S) + D_M(A,B)$. Let $a \in A$, $s \in S$ and $b \in B$ be such that $D(b,s) = D(B,S)$ and $D(a,b) \leq D_M(A,B)$. By triangle inequality

$$D(a,s) \leq D(a,b) + D(b,s) \leq D_M(A,B) + D(B,S)$$

By definition, $D(A,S) \leq D(a,s)$, which concludes the proof.

For the future purpose we mention that f would be a contraction even if we replaced $D(A,S)$ by $\tilde{D}_T = \min(D(A,S),T)$ for any T. Indeed, consider $\Delta = |\tilde{D}_T(A,S) - \tilde{D}_T(A,B)|$. If both $\tilde{D}_T(A,S)$ and $\tilde{D}_T(A,B)$ are at most T, nothing changes. If both are greater than T, then $\Delta = 0$. Otherwise, assume $\tilde{D}_T(A,S) = T$ (the other case is symmetric). One can observe that $\Delta = |T - \tilde{D}_T(B,S)| \leq D(A,S) - D(B,S)$, which by the previous argument is smaller than $D_M(A,B)$, which was to be shown.

It remains now to provide the sets $S_1 \dots S_E$ such that for any A, B and q as above there exists S_i such that the following conditions are true:

- $D(p, S_i) \leq a$
- $D(B, S_i) \geq b$
- $a - b \geq c$

which is clearly sufficient for our purpose. Again, note that replacing D by $\tilde{D}_T$ is good enough if $T = 1 \geq a, b$.

Set $a = \frac{1-c}{2}$ and $b = \frac{1+c}{2}$. Note that $a + b = 1 \leq D_M(A, B)$. Impose a regular cubic grid on $\Re^d$ with a side $2a$. Among all cells intersecting $B(p, a)$ choose the one which contains p and call it $\overline{p}$. Notice that the center of $\overline{p}$ belongs to $B(p, a)$. Define $\overline{C}$ to be the set of all cubes intersecting $[0, I]^d$ and let $\overline{B}$ be the set of cubes whose centers belong to $\cup_{p \in B} B(p, b)$. Notice that $\overline{p} \notin \overline{B}$. Also, we can bound

- $|\overline{C}| \leq (\frac{I}{2a} + 2)^d$
- $|\overline{B}| \leq u(c) = s\left(\left\lceil\frac{1+c}{1-c}\right\rceil\right)^d$

By using superimposed codes we know there exists $E = u(c)^2 \log^2 |\overline{C}|$ sets $\overline{S_1} \ldots \overline{S_E}$ such that for any p, B as above there exists $\overline{S_i}$ containing $\overline{p}$ but none of then elements from $\overline{B}$. We then construct S_i from $\overline{S_i}$ by replacing each grid cell by its center. These sets satisfy the above requirements. ∎

Remark 4 *One can observe that we do not need to use superimposed codes in the construction. If we pick each cell with probability $1/2u(c)$, then with probability $\Theta(1/u(c))$ we get a set S such that $D(p, S) \leq a$ and $D(B, S) \geq b$. Choosing $\Theta(u(c) \log 1/\beta)$ such sets gives a randomized embedding with probability of correctness $1 - \beta$. In fact, by using the probabilistic method one can improve the dependence on c in the above theorem from $\left(\lceil\frac{1+c}{1-c}\rceil\right)^{2d}$ to $\left(\lceil\frac{1+c}{1-c}\rceil\right)^d$, by setting $\beta = 1/(I/a)^{ds}$. Unfortunately, we do not have any explicit construction which yields such a bound.*

Complexity of the embedding. It is easy to see that the complexity of the straightforward implementation would be $O(s \cdot \sum_{i=1}^{E} |S_i|)$, since for each set S_i we need to find the closest pair between S_i and the embedded set (say A), which is of size s. The latter bound is at most $sI^d/u(C)$. In many practical situations this is not that much (e.g., the input to optical character recognition system is often represent by a raster image of low resolution). However, in general, it could be very large.

Fortunately, we can exploit the fact that the embedding still remains correct if we replace D by $\tilde{D}_T$ (call the resulting mapping $\tilde{f}$). Now, in order to compute $\tilde{D}_T(A, S)$, it is sufficient for each $a \in A$ to perform a search of a point in S within distance 1 from from a. The number of all points on a grid of side $2a$ which are within distance 1 from a is upper bounded by $O(1/(1-c))^d$. For each such point we can check if it belongs to S or not in poly$(d, \log |U|)$ time (see Preliminaries). If any such point belongs to S, we return the distance to the closest one. Otherwise, we return 1. Thus, the whole embedding can be computed in time $s \cdot E \cdot O(1/(1-c))^d \cdot (d \log |U|)^{O(1)}$, which is polynomial in d and E.

4.6.3 Generalization to l_p for $p < \infty$

Our embedding algorithm for $H^s(l_p)$ closely follows that of $H^s(l_\infty)$. However, instead of having a grid of side $2a = 1 - c$, we have a grid of size $2a/d^{1/p}$. We need only bound $|\overline{B}|$ and $|\overline{C}|$, which we do as follows.

- $|\overline{C}| \leq (\frac{Id^{1/p}}{2a} + 2)^d$
- $|\overline{B}| \leq u_p(c) = sO\left(\left\lceil\frac{1+c}{1-c}\right\rceil\right)^d$

The first bound is direct, and the second comes from the volume of a unit ball under l_p norm (see Preliminaries). We once again use superimposed codes and get $E_p = u_p(c)^2 \log^2 |\overline{C}|$, thus achieving the following theorem:

Theorem 21 *For any $I > 1$, $s > 0$ and $0 < c < 1$ the space $H^s([0, I]_p^d)$ can be c-embedded into l_∞^E where*

$$E = O\left(ds^2 O\left(\left\lceil\frac{1+c}{1-c}\right\rceil\right)^{2d}\left(\log^2 I + \log^2 \frac{1}{1-c} + \frac{\log^2 d}{p}\right)\right).$$

4.6.4 Translations and rotations

When the underlying metric is l_2, the above results can be easily extended to the Hausdorff metric under translations and rotations, using the *reference point* approach of [ABB91], which is as follows. Let C^d denote the set of compact subsets of $\Re^d$, and let $\mathcal{T}$ be a set of transformations on $\Re^d$. A mapping $c : C^d \to \Re^d$ is called a *reference point* w.r.t. $\mathcal{T}$ if the following two conditions are satisfied:

1. For all sets $A \in C^d$ and $T \in \mathcal{T}$ we have $c(T(A)) = T(c(A))$

2. There exists some constant $a \geq 0$ such that for all $A, B \in C^d$ we have $|c(A) - c(B)|_2 \leq a \cdot D_{l_2}(A, B)$

Several such mappings has been proposed in [ABB91, AAR94]. Also, it has been shown that:

1. If we translate A by translation T so that $c(T(A)) = c(B)$ then the distance between $T(A)$ and B is within a constant factor from the minimal

2. If we translate $c(A)$ to $c(B)$ via T and then find a rotation R minimizing the distance from $R(T(A))$ to B, then the distance between $(T \cdot R)(A)$ and B is within a constant factor from the minimal

Property (1) gives an immediate solution for embedding the generalized Hausdorff metric with translations: before embedding A just translate $c(A)$ to the origin. Property (2) yields a similar solution for the case of translations and rotations in the following way. Initially we align all $c(A)$'s to the origin (this includes both the database sets and the query set). Moreover, for any database set A_i let $p_i \in A_i$ be the point in A_i with the largest distance from the origin. Rotate each A_i around 0 such that the *vector* p_i is parallel to the X axis. Then build a nearest neighbor data structure DS for the resulting sets A_i. In order to process the query set Q, align its centroid to the origin. Then, for each $p \in Q$, rotate Q around the origin such that the vector p is parallel to the X axis and query the data structure DS with the rotated Q as an argument. Return the answer with the smallest distance from Q. Note that this increases the number of queries by a factor of $|Q|$.

The correctness of these procedure (i.e. the fact that it returns an $O(1)$-approximate nearest neighbor w.r.t. Hausdorff distance under translations and rotations) follows from the results of [ABB91, AAR94] and we omit the proof here. As mentioned in [ABB91], by exploring $O(1/\epsilon^2)$ points close to the origin and trying $O(1/\epsilon)$ different rotations (for $d = 2$) one can guarantee that the approximation guarantee is $(1 + \epsilon) \cdot C$, where C is the approximation guarantee for the approximate NN data structure for the Hausdorff metric. Note that this increases the number of queries by a factor of $O(1/\epsilon^3)$.

4.6.5 Threshold embeddings of Hausdorff metrics over general spaces

Let (X, D) be an arbitrary metric space. Then our method for embedding a Hausdorff of a normed space does not work anymore. The crucial problem is that $|B|$ cannot be bounded, that is, $|B(p,r)|/|B(p,r(1-\epsilon))|$ can be unbounded. Here, we give a randomized embedding procedure which is a modification of the randomized embedding given above. Our main modification will be that we select our reference sets via non-uniform sampling according to local properties of the metric.

In order to solve $1/\alpha$-approximate nearest neighbor, we need to be able to generate $(r, \alpha r)$-embeddings, for many values of r. Instead, we will be able to generate (r', α)-embeddings, for some $r' < r$. Notice, that an $(r', \alpha r')$-embedding is (by definition) also an $(r, \alpha r')$-embedding. Thus the dependence on α in the nearest neighbor search algorithm will become worse by a constant factor.

Theorem 22 *For any $\epsilon > 0$, $r > 0$ and $0 < \alpha < 1$, there is $r' < r$ such that $r' \geq \frac{r}{(1+\frac{2}{\alpha})^{1/\epsilon+1}}$ such that any metric X, $|X| = n$ can be $(r', (1-\alpha)r')$-embedded into l_∞^d with $d = O(s^2 n^\epsilon \log n/\epsilon)$.*

Proof: Let $\beta = \frac{\alpha+2}{\alpha}$ and let $r_i = r/\beta^i$ for $i = 0 \ldots 1/\epsilon + 1$. Our embedding will have a set of dimensions for each r_i. We will show that for each pair A, B, some dimension corresponding to some r_i will correctly approximately represent their Hausdorff distance. Since our embedding is into l_∞, we can simply concatenate all dimensions for all r_i in order to achieve our final embedding.

First note that, $\forall p \in X$ there is an $r(p) = r_i$ such that:

$$|B(p, \alpha r_i)| > 1/n^\epsilon \cdot |B(p, r_i(\alpha+2))| = 1/n^\epsilon \cdot |B(p, \alpha r_{i-1})|$$

since the volume of the ball around B can grow by a factor n^ϵ at most $1/\epsilon$ times. Now consider some $p \in A$ such that $D_M(A, B) = D(p, B)$, and let $r' = r(p)$ and $t = 2r'$.

As before, we need a mapping $f : H^s([0,I]^d_\infty) \to l^E_\infty$ such that for A, B and $p \in A$ such that $D(p,B) \geq 1$ we have $|f(A) - f(B)|_\infty \geq c$. The embedding will be defined by a sequence of E subsets $S_1 \ldots S_E$ of X as

$$f(A) = (D(A,S_1), \ldots, D(A,S_E)).$$

Once again, we will provide the sets $S_1 \ldots S_E$ such that for any A, B as above there exists S_i such that the following conditions are true:

- $D(p,S_i) \leq \alpha r'$
- $D(B,S_i) \geq r'$

Call the reference set S_i a *witness* to A, B if it satisfies these conditions. We will select points to go into the reference sets according to the density of their neighborhood, that is, node v is selected with probability $P(v) = 1/s|B(v,t)|$. Let $P(A)$ be the probability of selecting some element in $A \subset X$. We show the following:

Claim 2 *The probability that S_i is a witness to A, B is $\Omega(1/sn^\epsilon)$*

Proof: The proof proceeds by showing that $P(B(p,\alpha r')) = \Omega(1/sn^\epsilon)$ and $1 - P(\cup_{q\in B} B(q,r')) = \Omega(1)$. Consider $P(B(p,\alpha r'))$ first. We know that for all $x \in B(p,\alpha r')$, we have $B(x,t) \subseteq B(p,t+\alpha r')$, so $|B(x,t)| \leq |B(p,r'(\alpha+2))| \leq |B(p,\alpha r')|n^\epsilon$. Therefore $P(x) = 1/s|B(x,t)| \geq 1/sn^\epsilon|B(p,\alpha r')|$, and thus $P(B(p,\alpha r'))$ is $\Omega(1/sn^\epsilon)$.

Consider now $1 - P(\cup_{q\in B} B(q,r'))$. Let $q \in B(q,r')$. Since $t = 2r'$,for any $x \in B(q,r')$, we know that $B(q,r') \subset B(x,t)$. Therefore $|B(x,t)| \geq |B(q,r')|$, and $P(x) \leq 1/s|B(q,r')|$. We get $P(B(q,r')) \leq 1/s$. The probability that $D(B,S_i) \geq r'$ is then $\Omega(1)$.

Therefore, we get both events with probability $\Omega(1/sn^\epsilon)$, as required. ∎

In order to have a witness for all $n^{O(s)}$ pairs A and B such that $r(p) = r'$ we need to repeat the above procedure $O(s \log n)$ times. Taking all $1/\epsilon$ values of r' gives a total of $O(s^2 n^\epsilon \log n/\epsilon)$ dimensions. ∎

Complexity of the embedding. In principle we could apply this embedding to solve the approximate PLEB in $H^s(X)$. However, a direct approach would require advance knowledge of all queries, as we would need to embed all points (including query points) during the preprocessing. In order to avoid this problem, we show that embedding of a query point q can be done quickly "on-line" provided we can perform approximate nearest neighbor query in the underlying metric X.

We first show how to perform the embedding using an *exact* nearest neighbor oracle for X. Firstly, observe that (as before) if replace D by $\tilde{D}_r$ (and f by $\tilde{f}$) the embedding still remains correct. Next, observe that the proof of Theorem 22 clearly guarantees a t-approximate embedding of the *database set*. Also, recall that the embedding proceeds by computing several values of $\tilde{D}_r(A, S_i)$, which can be decomposed into $|A|$ computations of the exact nearest neighbor of $a \in A$ in S_i. Finally, recall that the $\tilde{f}$ is a contraction. Thus we use Lemma 5 and conclude that by embedding a query set Q using the sets $S_1 \ldots S_E$ (which can be done using sE exact nearest neighbor queries in X) the distance between $f(Q)$ and $f(A)$ for any set A from the database is distorted by at most a factor of $O(t)$.

Assume now that we are only given a $(1+\delta)$-approximate nearest neighbor oracle in X. The difficulty now is that $\tilde{f}$ is no longer contraction. However, one can observe that this can change the values of $|\tilde{D}_r(A, S_i) - \tilde{D}_r(A, S_i)|$ by at most an additive factor of Δ such that $|\Delta| \leq \delta r$. Thus, $\tilde{f}$ is a $(r, r', (1+\delta)r, r' - r\delta)$-embedding.

Altogether, we can reduce one approximate nearest neighbor query in $H^s(X)$ to $O(s \cdot s^2 n^\epsilon \log n/\epsilon)$ approximate nearest neighbor queries in X plus one query in $l_\infty^{s^2 n^\epsilon \log n/\epsilon}$.

4.7 Asymmetric embeddings

4.7.1 Reverse embedding of Hamming metric

In this section we use the randomized dimensionality reduction technique from Section 4.1 to obtain a *reverse* embedding of Hamming metric into itself. This has a natural application to reductions between furthest neighbor and nearest neighbor. In particular, this gives a $(1+\epsilon)$-furthest neighbor data structure with $O(d^{O(1)} n^{1/(1+\epsilon/6+\delta)})$ time per operation, for any $\delta > 0$. This result appeared first in [Ind00a].

The basic idea is similar to the approximate FN to NN reduction as in Section 3.4. Specifically, let $p, q \in \{0,1\}^d$ and let $\overline{q}$ denote a complement of q; note that that $\overline{q}$ is the antipodal point of q with respect to the (Euclidean) ball of radius $\sqrt{d}/2$ centered at the point $(1/2, \ldots, 1/2)$, hence the similarity to the technique of Section 3.4. Observe that $D(p,q) = d - D(p, \overline{q})$. Therefore the mapping $(L(p), R(q)) = (p, \overline{q})$ is an $(r, R, d-r, d-R)$-reverse embedding. Unfortunately, the gap $d-r : d-R$ could be much smaller than the original gap $R : r$, due to the fact that both r and R could be very small. In order to amplify the gap, we will use randomized dimensionality reduction of Section 4.1. Recall, that it yielded an $(r, cr, (1-e^{-\alpha}(1+\eta))(1+\gamma)/2 \cdot D', (1-e^{-c\alpha}(1+\eta))(1+\gamma)/2 \cdot D')$-embedding with probability of correctness $1-\beta$, for $D' = O(\log^2(1/\beta)/\gamma^{O(1)})$. For simplicity of calculations we assume here $\eta = \gamma = 0$ (note that these term will only affect the calculations by an arbitrarily small constant factor). In this case composing the randomized embedding (say F) with the asymmetric embedding (L, R) results in the $(r, cr, (1+e^{-\alpha})/2 \cdot D', (1+e^{-c\alpha})/2 \cdot D')$-reverse embedding. Thus, the gap achieved by the resulting embedding is at least $E(\alpha, c) = \frac{1+e^{-\alpha}}{1+e^{-c\alpha}}$.

Let $c = 1+\epsilon$ and $1+\epsilon'(\epsilon) = \max_{\alpha>0} E(\alpha, c)$. By numerical calculations we obtain the plot for $\epsilon'(\epsilon)/\epsilon$ as given on Figure 4.1.

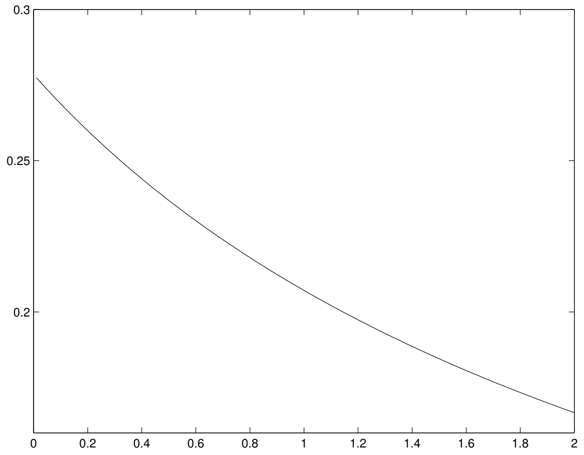

Figure 4.1: The optimized ratio ϵ'/ϵ as a function of ϵ.

For the case when $\epsilon = 2$, the optimization can be done explicitly yielding optimum

α to be such $e^{-\alpha} = 1/2$ and therefore $1 + \epsilon' = E = (1 + 1/2)/(1 + 1/2^3) = 1 + 1/3 = 1 + \epsilon/6$. By combining this observation with the fact that the ratio ϵ'/ϵ achieves its minimum over the interval $(0, 2]$ at the point $\epsilon = 2$ (see the plot), we conclude that $\epsilon'/\epsilon \geq 1/6$ for $\epsilon \in (0, 2]$.

4.7.2 Hardness results

All of the embeddings described earlier were designed to achieve *upper bounds*, i.e., positive algorithmic results. In this section we focus on embeddings which do not seem to imply any algorithmic results; instead, they indicate *hardness* of certain problems.

We start from the following simple theorem. Let $G = (P(X), \subset)$ be a poset defined over a powerset of X, $|X| = d$.

Theorem 23 *G can be embedded into l_∞^d with robustness 3.*

As a result, we conclude that any algorithm which solves c-PLEB in l_∞ for $c < 3$ automatically solves the subset query problem. This transfers the conjectured hardness of the latter problem to the former one.

Proof: Define function L as

$$L(x) = \begin{cases} 0 & \text{if } x = 0 \\ 2 & \text{if } x = 1 \end{cases}$$

and a function R as

$$R(x) = \begin{cases} 1 & \text{if } x = 0 \\ 3 & \text{if } x = 1 \end{cases}$$

Now we extend L to vectors via equality $L((b_1 \ldots b_d)) = (L(b_1), \ldots, L(b_d))$; we also extend it to subsets of $\{1 \ldots d\}$ via equality $L(A) = L(\chi_S)$, where χ_A denotes the characteristic vector of a set A. We extend R in the same way.

The pair (L, R) now forms the desired asymmetric embedding. Its correctness follows from the following claim.

Claim 3 *For $S, Q \subset X$ we have* $|R(Q) - L(S)|_\infty = \begin{cases} 1 & \text{if } Q \subset S \\ 3 & \text{otherwise} \end{cases}$

■

Our second theorem proves an analogous fact for Hausdorff metric. Let $G = (P_s(X), \subset)$ be a poset defined over all s-subsets of X, $|X| = 2^d$.

Theorem 24 *G can be embedded into $H^{s+1}(l_\infty^d)$ with robustness 3.*

Proof: Firstly, let $1 = (1, \ldots, 1)$. For simplicity, we assume that X itself is a set of all d-dimensional vectors with coordinates from $\{0,1\}$. For any $x = (x_1 \ldots x_d) \in X$ we use $c \cdot x$ to denote $(cx_1, \ldots, cx_d)$.

For any $A \subset X$ we define:

- $L(A) = \{3 \cdot (x, 1 - x) : x \in A\} \cup \{1\}$
- $R(A) = \{2 \cdot (x, 1 - x) : x \in A\}$

Note that $1 - x$ is a bit-wise -complement of x.

The correctness follows from the following claim.

Claim 4 *For any $A, B \subset X$ we have*

- *if $A \subset B$ then $D_{l_\infty}(L(A), R(B)) = 1$*
- *otherwise $D_{l_\infty}(L(A), R(B)) = 3$*

Proof: Assume first that $A \subset B$. In this case for any $a \in A$ the distance from $3(a, 1-a)$ to $R(B)$ is 1 (since $2(a, 1-a)$ belongs to $R(B)$). Also, the distance from 1 to any point in $R(B)$ is at most 1. On the other hand, the distance from any point in $R(B)$ to 1 (which belongs to $L(A)$) is 1. Therefore, $D_{l_\infty}(L(A), R(B)) = 1$.

On the other hand, assume that $A \not\subset B$. In this case there exists $a \in A - B$. Take any $b \in B$. The distance from $3(a, 1-a)$ to $2(b, 1-b)$ is 3, since $a \neq b$. This implies $D_{l_\infty}(L(A), R(B)) = 3$. ■

■

In the rest of this section we show two additional applications of the above theorems. Our first application follows from the following sequence of lemmas.

Let $G = (X, E)$ and $G' = (X', E')$ be a directed graph, $|X| \le |X'|$. We say that $f, g : X \to X'$ embed G into G', if for all $p, q \in X$ we have $(p, q) \in E$ iff $(f(p), g(q)) \in E'$; this definition can be extended to randomized embeddings in a natural way. Let $G = (X, \subset)$ be a poset defined over the set X of all intervals $I \subset \{1 \dots u\}$, and let $G' = (X', EMPTY)$ be a directed graph over the set $X' = P_s(\{1 \dots v\})$ such that for any $A, B \in X'$ we have $(A, B) \in EMPTY$ iff $A \cap B = \emptyset$.

Lemma 24 *There is an embedding of G into G' such that $s = O(\log u)$ and $v = O(u)$.*

Proof: For simplicity we assume u is a power of 2. We start from solving the following problem: design mappings $h_1 : \{1 \dots u\} \to P_s(\{1 \dots O(u)\})$ and $h_2 : X \to P_s(\{1 \dots O(u)\})$ such that $i \in I$ iff $h_1(i) \cap h_2(I) \neq \emptyset$ (intuitively, I will play the role of a "forbidden interval"). To this end, consider a binary tree T with the set of leafs corresponding to the elements of $\{1 \dots u\}$ and the set of internal nodes corresponding to the dyadic intervals over $\{1 \dots u\}$ (i.e., we have nodes $\{1, 2\}$, $\{3, 4\}$ etc. on the second level, $\{1 \dots 4\}$, $\{5 \dots 8\}$ etc on the third level and so on). We define $h_1(i)$ to be the set of all nodes on the path from the root to the leaf i. We also define $h_2(I)$ to be the set of nodes in T corresponding to intervals from the dyadic decomposition of I (i.e., the unique minimum cardinality partitioning of I into dyadic intervals). Note that $|T| = O(u)$.

The correctness of h_1 and h_2 follows from the fact that $i \in I$ iff there is a node on the path from the root to i corresponding to a dyadic interval contained in I.

Now, we observe that $\{a \dots b\} \not\subset \{c \dots d\}$ iff $a \in \{1 \dots c-1\}$ or $a \in \{d+1 \dots u\}$ or $b \in \{1 \dots c-1\}$ or $b \in \{d+1 \dots u\}$. Therefore, we can define $f(\{a \dots b\}) = h_1(a) \cup h_1(b)$ and $g(\{c \dots d\}) = h_2(\{1 \dots c-1\}) \cup h_2(\{d+1 \dots u\})$. ∎

Lemma 25 *There is a randomized embedding of G' into $(P_s(\{1 \dots s^2/\delta\}, EMPTY)$ which preserves* nonintersection *between any two elements of G' with probability $1-\delta$, for $0 < \delta < 1$. The* intersection *is always preserved.*

Proof: It is sufficient to use a random function $h : \{1 \ldots v\} \rightarrow \{1 \ldots s^2/\delta\}$ and extend it to sets. ■

Consider now the orthogonal range query problem in d-dimensional space. By using the above lemmas we can reduce this problem to the subset query problem for $|P|$ subsets of $\{1 \ldots d^2 \log^2 |P|/\delta\}$ in the following way. For $i = 1 \ldots d$ use Lemma 24 to obtain functions L_i and R_i. For simplicity we assume that the values of L_i and L_j do not intersect for $i \neq j$; we make the same assumption for R_i's. For any point $p = (x_1 \ldots x_d)$ define $L(p) = \cup_i L_i(\{x_i, x_i\})$. Also, for any box $B = \{a_1 \ldots b_1\} \times \{a_2 \ldots b_2\} \times \ldots \times \{a_d \ldots b_d\}$ define $R(B) = \cup_i R_i(\{a_i \ldots b_i\})$. We have that $p \in B$ iff $L(p) \cap R(B) = \emptyset$. Since $L(p)$ and $R(B)$ are of size $O(d \log u)$, we can apply Lemma 25 to reduce the universe size to $O(d^2 \log u)$ (say via function h). The condition $A \cap B = \emptyset$ is equivalent to $A \subset \overline{B}$, where $\overline{B}$ is the complement of B. Therefore, with a constant probability, there exists $p \in P$ which belongs to B iff $h(R(B)) \subset \overline{h(L(p))}$. Thus, we proved the following theorem.

Theorem 25 *If there is a data structure which solves the subset query problem for n subsets of $\{1 \ldots k\}$ having query time $Q(n, k)$ and using space $S(n, k)$, then there exists a randomized algorithm for solving the orthogonal range query problem for n points in $\{1 \ldots u\}^d$ with query time $Q(n, O(d^2 \log^2 u))$ and using space $S(n, O(d^2 \log^2 u))$.*

Note that one can always reduce u to $O(n)$ by sorting all coordinates first and replacing them by their ranks.

The above theorem, together with Lemma 23 implies an interesting fact: if we efficiently solve the c-PLEB for l_∞^d for $c < 3$, then we can solve this problem *exactly* with similar resources.

Our second application shows the following relation between the robust embedding of $(P_s(\{1 \ldots u\}), \subset)$ into l_∞ and superimposed codes.

Theorem 26 *Assume there is an embedding of $(P_s(\{1 \ldots u\}), \subset)$ into l_∞^d with robustness $1 + \epsilon$, $\epsilon > 0$. Then there exist a superimposed (s, u)-code with length $O(d/\epsilon)$.*

Before we show the proof, we point out the obvious application.

Corollary 6 *If f is an embedding of $H^{s+1}_{[0,2]^k_\infty}$ into l^d_∞ with distortion $c < 3$, for $2^k >$ poly(s), then $d = \Omega(s^2 k/\log s)$.*

Proof: By Lemma 24, there exist embedding of $(P_s(\{1 \dots u\})$ into $H^{s+1}_{[0,2]^k_\infty}$ with $k = \log u$ with robustness 3. By the assumption we can embed $H^{s+1}_{[0,2]^k_\infty}$ into l^d_∞ with distortion $c < 3$. By composing these two embeddings we obtain an embedding of $(P_s(\{1 \dots u\})$ into l^d_∞ with robustness $3/c = 1 + \epsilon$. By Theorem 26 this implies the existence of a (s, u)-code of length $O(d)$. The lower bound follows from the lower bound in Preliminaries. ∎

Proof:(of Theorem 26). Let (L, R) be the embedding with robustness $1 + \epsilon$. We start from the following technical observation.

Claim 5 *Let P be the set of all points $L(A)$ and $R(B)$ where $A, B \in P_s(\{1 \dots u\})$. Then the diameter of P is at most 4.*

Proof: Since the empty set is included in any set, we have $|L(\emptyset) - R(B)|_\infty \leq 1$ for any B. Also, for any set A we have $A \subset A$, and therefore

$$|L(A) - L(\emptyset)|_\infty \leq |L(A) - R(A)|_\infty + |R(A) - L(\emptyset)|_\infty \leq 1 + 1 = 2$$

Therefore all points from P are within distance 2 from $L(\emptyset)$. ∎

We can therefore assume that $P \subset [0, 4]^d$. Impose a uniform grid on $[0, 4]^d$ of side $\epsilon/2$. For any point $p \in [0, 4]^d$ and $i = 1 \dots d$ let $g_i(p)$ be the ith coordinate of the cell containing p; for convenience we assume that the coordinate values for distinct i's are distinct. Then we define:

- $L'(A) = \{g_1(L(A)), \dots, g_d(L(A))\}$
- $R'(A) = \cup_i g_i(B(R(A), 1))$

The following claim shows that L' and R' "almost" form a superimposed code

Claim 6 *For any A and B as above we have $L'(A) \subset R'(B)$ iff $A \subset B$.*

Now we show how to obtain an actual superimposed code from L' and R'. For any $a \in U$ we define $C(a) = L'(\{a\})$. We will show that $C(a)$ forms an (s, u)-code. Consider any a, B such that $a \notin B$. Notice that

- for each $b \in B$, we have $C(b) = L'(\{b\}) \subset R'(B)$. Therefore, $\cup_{b \in B} C(b) \subset R'(B)$.
- $C(a) = L'(\{a\}) \not\subset R'(B)$, as $a \notin B$.

Therefore $C(a) \not\subset \cup_{b \in B} C(b)$. Thus $C(a)$ is an $(s, 2^d)$ code. Moreover, let $U' = \cup_a C(a)$. It is easy to see that $|U'| = O(d/\epsilon)$. The theorem follows. ∎

Chapter 5

Extroverted applications

In this chapter we address a set of problems which have the following two properties in common:

- they do not belong to the class of problems covered by the name "high-dimensional computation geometry"
- they can be solved using the tools developed earlier in this thesis

Thus, we call them "extroverted applications", i.e., applications of techniques developed within an area to problems outside of it.

In this chapter we present several such applications. The first application addresses the problem of computing with small memory. In particular, we will be interested in computing *sketches* of a stream of coordinates such that (a) the sketches are short (b) they can be computed using small memory and (c) given two sketches we can recreate the difference between the original streams up to a small constant factor (cf. Section 5.1 for more details of the computational model). We will show that this problem can be viewed as an embedding of the stream into a low dimensional space, which will allow us to use the techniques of Chapter 3. As a result, we unify previous results of [AMS96, FKSV99] and also solve the main open problem of [FKSV99].

In Section 5.2 we focus on problems in combinatorial pattern matching. An example problem we consider is the approximate nearest neighbor problem when the set of data points consists of all m-length subsequences of a given "text" sequence

$t = t[1]\dots t[n]$, $t[i] \in \{0,1\}$. By using the tools from Chapter 3 we can solve this problem in $\tilde{O}(n^{1/(1+\epsilon)})$-time after $O(mn^{1+1/(1+\epsilon)})$-time preprocessing. However, in case when $m = \Theta(n)$, the resulting preprocessing time is superquadratic. Therefore, we show how to reduce the preprocessing time to roughly $O(n^{1+1/(1+\epsilon)} \log m)$. To get this bound we use locality sensitive hashing to index the substrings of t, but we take advantage of the fact that the data points are substrings, in order to compute the hash functions in $O(n \log m)$ time, instead of $O(nm)$. To this end, we use Fast Fourier Transform, which essentially allows us to compute a dot product of a given sequence with all substrings of a given string in $O(n \log n)$ time.

Since the dimensionality reduction techniques described earlier are also based on dot product, we can easily perform a mapping of all m-length substrings of t into a lower-dimensional space (i.e., with polylogarithmic dimension) in $O(n\mathrm{poly}(\log m))$ time, even when $t[i]$ are numbers, not bits. Having the mapping done, we can approximate a distance between any pair of m-length substrings of t in polylogarithmic time. In Section 5.2, we remove the restriction on m being fixed in advance, by using a more complicated data structure.

Our next application area is low-dimensional computational geometry. Here, we focus on the following problem: given n points and m hyperplanes in $\Re^d$, find a spanning tree which spans all the points and crosses a minimum number of hyperplanes. This problem has applications for optimization of a data structure for hyperplane range queries. It can be trivially solved in $O(n^2 m)$ time, or in $O(n^2 \log n)$ using more sophisticated algorithm. Here, we show that (a) the crossing numbers between all pairs of points form a metric which can be embedded into low dimensional l_2 norm, and (b) the embedding can be done in $O(n^{2d/(d+1)+\delta})$ time, for any $\delta > 0$. After the embedding is done, we can apply high-dimensional MST algorithms from Section 3.3 to obtain a $(1+\epsilon)$-approximate solution in $\tilde{O}(n^{1+1/(1+\epsilon)})$ time.

5.1 Stream computation

The first problem we address is defined as follows [FKSV99] (see also [HRR98] for a background on stream computation). Assume that we have an access to a stream S of data, where each chunk of data is of the form (i, a), where $i \in [n]$ and $a \in \{-M \dots M\}$. We want to approximate (up to the multiplicative factor $(1 \pm \epsilon)$) the quantity $L_1(S)$, where

$$L_p(S) = (\sum_{i \in [n]} | \sum_{(i,a) \in S} a|^p)^{1/p}.$$

The problem has a variety applications to estimating the size of self-join [AMS96, GM99] and applications to estimation of statistics of Net-Flow data (cf. [FKSV99] and Section 7.3). An obvious solution to this problem is to maintain a counter c_i for each i and compute the sum of $|c_i|$'s at the end. Unfortunately, this solution requires $\Theta(n)$ words of storage. In their influential paper, Alon, Matias and Szegedy [AMS96] proposed a very nice and simple scheme for approximating $L_2(S)$[1] in space $O(1/\epsilon^2)$ with (arbitrarily large) constant probability. Feigenbaum, Kannan, Strauss and Viswanathan [FKSV99] proposed an algorithm (using similar amount of memory) for $L_1(S)$ for the case where (roughly) for each i the stream S contains at most two pairs (i, a). An alternative way to view their result is to assume two streams, one (S_r) containing red pairs and another one (S_b) containing blue pairs; for each i there is at most one pair (i, a) of each color. The goal is to compute *sketches* $C(S_r)$ and $C(S_b)$ of small size, such that the approximate value $L_1(S_r, S_b) = \sum_i |\sum_{(i,a) \in S_r} a - \sum_{(i,a) a \in S_b}|$ can be quickly evaluated from $C(S_r)$ and $C(S_b)$ by applying some function F (see [FKSV99] for more details of the model).

In this section we propose a unified framework for approximating $L_p(S)$ for $p \in \{1, 2\}$[2], using $O(\log n/\epsilon^2)$ memory words, each word containing $O(\log(M + n))$ bits. Our algorithm (first published in [Ind00b]) does not have the aforementioned restrictions of [FKSV99]; thus, it solves the main open problem from that paper (see [FKSV99], comments after Corollary 16). Moreover, our algorithm maintains only linear combinations of the input values, and therefore extends also to the sketch model (again, without the restrictions of [FKSV99]). Since the algorithm is simple and free of large constants, it is likely to find practical uses for the compression applications mentioned in [FKSV99].

[1] In their original paper they assumed all pairs are of the form $(i, +1)$, but it was shown in [FKSV99] that their algorithm actually works for the general case.

[2] We also discuss the extension to any $p \in (0, 2]$.

We now start the description of the algorithm. Let S be the data stream sequence containing pairs (i, a), for $i \in [n]$ and $a \in \{-M \ldots M\}$. We present the algorithm for calculating $L_1(S)$; the extension to $p \neq 1$ is discussed at the end.

We present our algorithm in three step. In the first step we present an algorithm which approximates well $L_1(S)$, but suffers from two major drawbacks:

1. It assumes infinite precision of the calculations (i.e. uses arithmetic operations on real numbers)

2. Although it uses only $O(1/\epsilon^2)$ words for storage, it performs random (and multiple) access to as many as $\Theta(n)$ random numbers. Thus a natural implementation of the algorithm would require $\Theta(n)$ storage.

Despite these limitations, the algorithm will serve well as an illustration of our main ideas. In the next two steps, we will remove its limitations.

An ideal algorithm. Let $l = c/\epsilon^2 \log 1/\delta$ for a constant $c > 1$ specified later. The algorithm works as follows.

1. Initialize nl independent random variables $X_i^j, i \in [n], j \in [l]$ with Cauchy distribution; set $S^j = 0$, for $j \in [l]$

2. For each new pair (i, a): perform $S^j = S^j + aX_i^j$ for all $j \in [l]$

3. Return median$(|S^0|, \ldots |S^{l-1}|)$

Let $c_i = \sum_{(i,a) \in S} a$; if there is no $(i, a) \in S$, we define $c_i = 0$. Thus $L_1(S) = C = \sum_i |c_i|$. The following claim justifies the correctness of the algorithm.

Claim 7 *Each S^j has the same distribution as CX where X has Cauchy distribution.*

Proof: Follows from the 1-stability of Cauchy distribution (cf. Section 4.5). ■

Therefore, it is sufficient to estimate C from independent samples of CX, i.e. from $S^0 \ldots S^{l-1}$. To this end, we use the following Lemmas.

Lemma 26 *If X has Cauchy distribution, then median$(|X|) = 1$. Therefore, median$(a|X|) = a$, for any $a > 0$.*

Proof: If X has Cauchy distribution, then the density function of $|X|$ is $f(x) = \frac{2}{\pi}\frac{1}{1+x^2}$. Therefore, the distribution function of X is equal to

$$F(z) = \int_0^z f(x)dx = \frac{2}{\pi}\arctan(z)$$

Since $\tan(\pi/4) = 1$, we have $F(1) = 1/2$. Thus median$(X) = 1$. ∎

Lemma 27 *For any distribution D on $\Re$ with the distribution function F, take $l = c/\epsilon^2 \log 1/\delta$ independent samples $X_0 \ldots X_{l-1}$ of $\mathcal{D}$; also, let $X =$ median$(X_0 \ldots X_{l-1})$. Then for a suitable constant c we have*

$$\Pr[F(X) \in [1/2 - \epsilon, 1/2 + \epsilon]] > 1 - \delta$$

Proof: Folklore. ∎

Lemma 28 *Let F be the distribution function of $|X|$ where X has Cauchy distribution, and let $z > 0$ be such that $F(z) \in [1/2 - \epsilon, 1/2 + \epsilon]$. Then, if ϵ is small enough, we have $z \in [1 - 4\epsilon, 1 + 4\epsilon]$.*

Proof: Follows from the fact that $F^{-1}(x) = \tan(x\pi/2)$ has bounded derivative around the point 1/2. In particular, $(F^{-1})'(1/2) = \pi$. ∎

Therefore, for a suitable constant c, we have the following Theorem.

Theorem 27 *The "ideal" algorithm correctly estimates $L_1(S)$ up to the factor $(1 \pm \epsilon)$ with probability at least $1 - \delta$.*

Bounded precision. Now we show how to remove the assumption that the numbers on which we perform operations have infinite precision. Since the number is in the data stream are integers, we only need to take care of the random variables X_i^j. Specifically, we show that it is sufficient to assume that they take values in the set $V_L = \{p/q : p, q \in \{-L, L\}, q \neq 0\}$, where L is small.

Consider the following way of generating X_i^j. Let Y_i^j be a random number from the set $[0,1)$. We define $X_i^j = F^{-1}(Y_i^j) = \tan(\pi Y_i^j/2)$. Now we define an approximation $\tilde{X}_i^j$ of X_i^j. Let $\tilde{Y}_i^j$ be equal to Y_i^j rounded to the nearest multiple of $1/L$. We define $\tilde{X}_i^j$ to be $F^{-1}(\tilde{Y}_i^j)$, again rounded to the nearest multiple of $1/L$. Consider the case when $\tilde{Y}_i^j < 1 - K/L = 1 - \alpha$ (K to be specified later). Since the derivative of $F^{-1}(x)$ for $x < 1-\alpha$ is $O(1/\alpha^2)$, it follows that in this case $\tilde{X}_i^j = X_i^j + E_i^j$, where $|E_i^j| = O(\frac{1}{\alpha^2 L}) = O(K^2/L) = \beta$.

Now we set K and L such that $K/L < 1/(n/\delta)^{\Theta(1)}$ and $\beta << \epsilon$, in which case we know that $\tilde{X}_i^j = X_i^j \pm \beta$ for all i, j with high probability. Then the value

$$\tilde{S}^j = \sum_i \sum_{(i,a)\in S} a\tilde{X}^j = \sum_i c_i \tilde{X}_i^j = \sum_i c_i (X_i^j \pm \beta) = S^j \pm \beta \sum_i c_i$$

Since $\text{median}(S^j) = \sum_i |c_i|$, by making β to be sufficiently smaller than ϵ, we can ignore the contribution of $\beta \sum_i c_i$ to the estimated quantity.

Randomness reduction. Consider a fixed S^j. From the above it follows that the value of S^j can be represented using $O(\log M)$ bits; also, we need only $O(\log n)$ bits to generate each $\tilde{X}_i^j$. Unfortunately, we still need $O(n)$ memory words to make sure that if we access a specific $\tilde{X}_i^j$ several times, its value is always the same. We avoid this problem in the following way. Assume for a moment that the pairs (i,a) are coming in the increasing order of i. In this case we do not have to store X_j^i, since we can generate them on the fly. Thus, the algorithm uses only $O(\log M)$ storage and $O(n)$ chunks of randomness, and therefore there exists a pseudo-random generator G (cf. Section 4.4) which given a random seed of size $O(\log M \log(n/\delta))$ expands it to a sequence $\overline{X}_0^j \ldots \overline{X}_{n-1}^j$, such that using $\overline{X}_i^j$ instead of $\tilde{X}_i^j$ results in negligible probability of error and therefore the resulting value of $\tilde{S}_i^j$ (call it $\overline{S}_i^j$) can be used to estimate $L_1(S)$. However, observe that for a fixed random seed r, the value S^j *does not depend* on the order in which the pairs (i,a) come (since addition is commutative). Therefore, G is good as well if the input is unsorted, i.e. the pairs come in arbitrary order. Since we use l random seeds for each $j \in [l]$, we obtain the following result.

Theorem 28 *There is an algorithm which estimates $L_1(S)$ up to a factor $(1 \pm \epsilon)$ with probability $1-\delta)$ and uses*

- $O(\log M \log(1/\delta)/\epsilon^2)$ *bits of random access storage*
- $O(\log M \log(n/\delta) \log(1/\delta)/\epsilon^2)$ *random bits (which can be stored in a random access storage)*
- $O(\log(n/\delta))$ *arithmetic operations per pair* (i, a)

Now we mention extensions to $p \neq 1$. For $p = 2$, the algorithm and analysis remains essentially the same, with Cauchy distribution replaced by Gaussian. For general $p \in (0, 2]$ the algorithm and analysis become more involved, mainly due to the fact that no closed formulas are known for densities and/or distribution functions of general p-stable distribution. However, it is known [CMS76] that one can generate p-stable random variables essentially from two independent variables distributed uniformly over $[0, 1]$; therefore, one can implement our algorithm for general p. As far as the analysis is concerned, it seems (we did not perform a rigorous verification of this fact) that the distribution functions of p-stable distributions are Lipschitz around the median (i.e. an analog of Lemma 3 holds); also their median can be computed numerically for any p. Therefore, it seems likely that the algorithm is provably correct also for general p.

5.2 String matching problems

We start from some basic facts about *convolutions.* Let $R = (\oplus, \otimes)$ be any semiring. Let $a[0 \dots n-1]$ and $b[0 \dots m-1]$ be vectors of elements from R. The *convolution* of a and b is defined as a vector $a * b$ such that $(a * b)[i] = \oplus_{j=0}^{m-1}(a[i-j] \otimes b[j])$ for $0 \leq i \leq n-m$ (out-of-range values are assumed to be 0). The "$*$" will be indexed with R unless it follows from the context. Notice that convolution of a and b corresponds naturally to multiplication of univariate polynomials $\oplus_{j=0}^{n-1} a[j]x^j$ and $\oplus_{j=0}^{m-1} b[j]x^j$. We also define $a \cdot b = \oplus_i a[i] \otimes b[i]$.

In this section we will be mainly interested in two cases of R: real numbers $\Re$ and the "modulo P" field Z_P. For both cases it is known that the convolution can be computed in $O(n \log n)$ time [AHU74].

We also define $a \circ b$ to be an array such that $(a \circ b)[i] = \oplus_{j=0}^{m-1}(a[i+j] \otimes b[j])$ for $0 \le i \le n+m-2$. It it easy to see that this operation can be computed within the same bound as convolution and vice versa. Thus in the following we will use either $\circ$ or $*$ depending on which is more convenient.

We now start to describe the first application, i.e., the approximate nearest neighbor search for substrings of a string. From Section 2.2.2 we know that in order to obtain the bound claimed, we need to compute the values of $h(t^m(i)_{|I})$ for all $i = 1 \ldots n-m$ in $O(n \log m)$-time, where:

1. $t^m(i)$ is a m-dimensional vector $(t[i], \ldots, t[i+m-1])$
2. $I \subset \{1 \ldots m\}$ is a set of bits onto which we project the vectors
3. h is a "usual" hash function which is used to map the projected vectors to the range $\{0 \ldots n^c\}$ for some constant c

We can combine projection and hashing in a single formula

$$h(x) = \sum_{i \in I} a_i \cdot x_i \bmod P$$

where P is large prime and a_i are random numbers from $\{0 \ldots P-1\}$. However, observe that $h(x)$ essentially computes a dot product of x and a vector containing a_i at the position i and zeros elsewhere. Therefore, we can compute $h(t^m(i)_{|I})$ for all i in $O(n \log n)$ time. The time can be further reduced to $O(n \log m)$ by using a standard technique of splitting the sequence into overlapping parts of length $O(m)$ and performing the convolution separately for each part. Since we need to compute $O(n^{1/(1+\epsilon)})$ indices, the bound follows.

We now switch to the second application, namely the $\tilde{O}(n)$-space data structure which given a triple (i, j, m) computes an approximate distance between $t^m(i)$ and $t^m(j)$ in polylog(n) time; in this case the elements of the vector t can be real numbers. The distance function can be induced by any l_p norm for $p \in (0, 2]$. Here we focus on the case $p = 2$, but by using the techniques from the previous section we can generalize the result to other p's. The algorithm first appeared in [IKM00].

As mentioned in the introduction to this chapter, if the window length m is fixed, we can easily perform a mapping of all m-length substrings of t into a $O(\log n)$-dimensional space in $O(n\text{poly}(\log m))$ time, by computing a logarithmic number of dot product with random vectors; we call the value of each such product a *sketch*) and denote it by $S(t^m(i))$. Having the mapping done, we can approximate a distance between any pair of m-length substrings of t in logarithmic time. In order to remove the "fixed m assumption", we will construct only a *pool* of sketches; this will be small subset of the set of all sketches we need. Following this preprocessing, we will be able to determine the sketch of any substring in the original string in $O(1)$ time fairly accurately. In what follows, we will explain this procedure in more detail.

First we focus on the preprocessing. We choose $l = 1, 2, 2^2, \ldots, 2^i, \ldots, 2^{\log n} = n$ and for each such l, we compute the sketches of all substrings of t of that length using standard tools. In fact, we compute *two* versions of sketches, each using different random variables; they are called S^1 and S^2. The resulting set of sketches is what we call the *pool*. This is of size $O(n \log^2 n)$ altogether and it takes time $O(n \log^3 n)$ to compute.

Second, we focus on determining the sketch for any substring $t[i, \ldots, i + l - 1]$. Let us fix a particular component, say, $S(t[i, \ldots, i + l - 1])$ for now. Consider the following two possibilities.

- $l = 2^r$, in which case we have the sketches for this substring in our pool, so we merely lookup the desired sketch in $O(1)$ time.

- $2^r < l < 2^{r+1}$. In that case, we compute the sketch S' as follows:

  $$S'(t[i, \ldots, i+l-1])[j] = S^1(t[i, \cdots, i+2^r-1])[j] + S^2(t[i+l-2^r, \ldots, i+l-1])[j])$$

We claim that S' satisfies a property very similar to the one in Lemma 20. To make it specific, we use the notion of *stochastic dominance*. Recall that for two random variables X and Y we say that Y *dominates* X (or $X \preceq Y$), if for any z we have

$$Pr[Y > z] \geq Pr[X > z]$$

The dominance relation is known to be preserved under monotone functions, i.e., if $X_1 \preceq Y_1, \ldots, X_l \preceq Y_l$, then $f(X_1, \ldots, X_l) \preceq f(Y_1, \ldots, Y_l)$ if f is a monotone function.

Lemma 29 *For any two vectors u, w of same length, we have*

$$|S(u) - S(w)| \preceq |S'(u) - S(w)'| \preceq \sqrt{2}|S(u) - S(w)|$$

Therefore, we can "sandwich" $|S'(u) - S'(w)|$ between $|S(u) - S(w)|$ and $\sqrt{2}$ times the latter value. Since Lemma 20 shows a $(1+\epsilon)$-approximation guarantee for S, we automatically obtain a $\sqrt{2}(1+\epsilon)$-approximation for S'.

Proof: For any interval $I \subset \{1 \ldots l\}$, define 1_I to be a vector of length l containing 1's at positions belonging to I and zeros elsewhere. Define $I_1 = \{1 \ldots 2^r\}$ and $I_2 = \{l - 2^r + 1 \ldots l\}$. Note that $1_{I_1} + 1_{I_2}$ contains only 1's and 2's (no zeros), since $I_1 \cup I_2 = \{1, \ldots, l\}$. Let v^b be the vectors used for sketch S^b, $b \in \{1, 2\}$. Let $x = u - w$ and observe that

$$[S'(u) - S'(w)] = S'(x) = v^1 \cdot x + v^2 \cdot x = (v^1 + v^2) \cdot x$$

By additivity property, the jth coordinate of $v_i^1 + v_i^2$ has distribution $N(0, 2)$ if $j \in I_1 \cap I_2$ and $N(0, 1)$ otherwise. Thus $S'(x)$ has distribution $N(0, D)$, where $D = \sum_j (1_{I_1} + 1_{I_2})[j](x[j])^2$. For comparison, note that $S(x)$ has distribution $N(0, \sum_j (x[j])^2)$ and $\sqrt{2}S(x)[i]$ has distribution $N(0, 2\sum_j (x[j])^2)$. Therefore (by monotonicity)

$$|S(x)| \preceq |S'(x)| \preceq \sqrt{2}|S(x)| = \sqrt{2}|S(u) - S(w)|$$

■

In order to reduce the factor $\sqrt{2}$ to $1 + \epsilon$, observe that if we chose the values of l to be $1, (1+\eta), (1+\eta)^2 \ldots$ etc, then we could choose I_1 and I_2 such that they overlapped in only a ηl positions. On the average, this would increase the variance of $S'(x)$ by a factor of $(1+\eta)$. However, we could be unlucky and it could happen that most of the weight is concentrated in the intersection of I_1 and I_2. To avoid this problem, we need to create $O(1/\eta)$ sketches, making sure that the overlapping parts are disjoint. In this way we contribute at most a factor of $(1 + O(\eta))$ of error. Note that this increases the total representation size by a factor of $O(1/\eta^2)$.

In this way we showed the following theorem.

Theorem 29 *For any $0 < \epsilon < 1$, given a string t of n numbers, it is possible to build a data structure of size $O(n \log^2 / \epsilon^4)$ which given any triple (i, j, m) reports*

$|t^m(i) - t^m(j)|$ *with error* $(1+\epsilon)$ *in time* $O(\log n/\epsilon^4)$. *The preprocessing time is* $O(n \log^3 /\epsilon^4)$.

5.3 MST with low crossing number

Given a set of lines in the plane, a natural measure of distances between any two points, is the number of lines one has to cross to reach from one point to the other. This is a discrete distance measure, it is quite general and can be used to approximate the Euclidean distance and other distance measures. However, since this measure is defined by an arrangement of lines, it is not locally defined and is thus computationally inconvenient.

For a set of n points in the plane, finding the minimum spanning tree (MST) of the points, so that the number of intersections between the tree and the set of lines is minimized, quantify how the set of points interact with the set of lines. In fact, when the set of lines is the set of all possible lines, then this MST is the standard Euclidean MST [AF97] (here one minimizes the average number of edges of the MST crossed when picking a random line). Such an MST is related to a spanning tree of low stabbing number (STLSN) [Wel92, Aga91], as follows: While the spanning tree of low stabbing number guarantee that any line intersects at most $O(\sqrt{n})$ edges of the spanning tree, the MST guarantees that the overall number of intersections between the tree and a given set of lines is minimized. Thus, if we have in advance the set of lines of interest, then the MST will perform better than the STLSN. The spanning tree of low-stabbing number was used in several applications, see for example [Aga91, MWW91]. In particular, having such an MST enables one: (i) to answer half-plane range queries in an efficient manner using a near linear space [GHS91], (ii) bound the complexity of the faces of the arrangement of lines that contain the points [HPS99], and (iii) traverse between the points in an efficient way, so that the number of updates needed is minimized. (Imagine traversing among the points and maintaining the set of half-planes that contain the current point. Each time we cross a line we perform an update operation.)

Computing the MST for the general case of arcs can be done in $O(n^2 \log n)$ time by performing wavefront propagation from each of the points. As for approximation algorithms, Har-Peled and Sharir [HPS99] gave recently an approximation algorithm for the case of arcs, computing a Steiner tree in expected running time $w = O((n + W_{opt})\alpha(n)\log(n))$, where $\alpha(n)$ is the inverse Ackerman function and W_{opt} is the weight of the optimal Steiner tree.[3] The algorithm outputs a tree of weight w (and thus gives roughly $O(\log n)$-approximation).

For a set L of n lines, let $\mathcal{D}_L(p,q)$ denote the number of lines of L that the segment pq intersects in its interior. This is a metric as the triangle inequality holds for it (we assume without loss of generality that none of the points lies on the given lines). The main contribution of this section is to show how one can apply the subquadratic approximation algorithms for high-dimensional geometry problems (in particular MST) to $\mathcal{D}_L(p,q)$. This is done by associating with each point in the plane, an n-dimensional binary vector, where i-th coordinate indicate on which side of the i-th line the point lies. In this way, we map our input points into points lying on the n-dimensional hypercube. The crossing metric is no more than the Hamming distance between the mapped points. We can now deploy the techniques of Section 3.3 to those mapped points, yielding an approximation algorithm for the MST problem. Bringing down the running time to be subquadratic requires some additional work. This algorithm appeared first in [HPI00].

Specificly, we show how to compute a mapping of the points into space of dimension $O(\log^4 n)$ (in this part we ignore constants that are polynomials of $1/\epsilon$); this embedding can be computed in $\tilde{O}(n^{4/3})$ time for n points, for $d = 2$. For $d > 2$ dimensions, we show how to embed the crossing metric induced by n hyperplanes over a set of n points in $\Re^d$, in $O(n^{2d/(d+1)+\delta})$ time, where $\delta > 0$ is arbitrary. After the embeding is done, we can apply the results from Chapter 3 to get subquadratic approximation algorithms for various problems, including MST.

For the special case of $d = 2$, we can "unfold" the algorithm and obtain a $(1+\epsilon)$-approximation algorithm for MST with running time $O((n/\epsilon^5)\alpha^3(n)\log^5 n)$; the details are in [HPI00]. We are not aware of any such result for $d > 2$.

[3] It is easy to verify that if we have triangle inequality then the Steiner tree weight is at least half the weight of the MST.

5.3.1 Embedding of the crossing metric over $\Re^2$

In this section we show how to approximately embed $\mathcal{D}_L$ into low-dimensional Hamming cube. Since $\mathcal{D}_L$ can be already thought of as a m-dimensional Hamming cube, it is sufficient to show how to implement efficiently the threshold embedding from Section 4.1. To this end, recall that that mapping has been defined as

$$g(p) = (p_{|I_1}, \ldots, p_{|I_{d'}})$$

where I_1 are subsets of $\{1 \ldots m\}$. Since in our scenario each coordinate corresponds to a line, it means that $p_{|I} = q_{|I}$ iff both p and q are in the same cell of the arrangement defined by the lines corresponding to I. Thus, we compute each dimension by computing a function $f : P \to \Sigma$ which for each p returns a label of arrangement cell to which p belongs.

Lemma 30 *Given a set P of n points, and a set I of m lines, one can compute the function $f(\cdot)$ for all the points of P in $\tilde{O}(m^{2/3}n^{2/3} + m + n)$ time.*

Proof: What one needs to do, is to compute all the faces of the arrangement that contains points of P. This can be done in $O(m^{2/3}n^{2/3}\log^{2/3}(m/\sqrt{n}) + (m+n)\log m)$ expected time [AMS98]. ∎

Therefore, the total time needed to perform the embedding is $\tilde{O}(m^{2/3}n^{2/3} + m + n)d'$, where $d' = O(\log(1/\beta)/\gamma^3)$ as in Section 4.1. We need to set β to $1/n^{\Theta(1)}$ and $\gamma = 1/\log n$. Therefore, $d' = O(\log^4 n)$ is sufficient.

5.3.2 Embedding of the crossing metric over $\Re^d$

In this section, we extend the methods from the previous section to the crossing metric defined by $(d-1)$-dimensional hyperplanes in $\Re^d$, for any fixed $d \geq 2$. To this end, it is sufficient to design an efficient procedure, which given a set of n points $P = p_1, \ldots, p_n$ and a set of m hyperplanes $\mathcal{H} = \{H_1, \ldots, H_m\}$, assigns a symbol $\sigma_i \in \Sigma$ to each p_i in such a way that $\sigma_i \neq \sigma_j$ iff there exists H_k which separates p_i from p_j. Unfortunately, the idea from the previous section does not give subquadratic time algorithm for $d > 2$, since even in $d = 3$ the complexity of n cells in an arrangement formed by n

planes could be $\Omega(n^2)$. Fortunately, for our purpose, we do not need to compute the actual cells containing p_is. Rather, it is just sufficient to find the *labels* for those cells, or more specifically, a function $h : P \to \Sigma$ such that $h(p) = h(q)$ iff p and q belong to the same arrangement cell.

Abusing notations, we denote by $H_k(p)$ the function returning 1 if p lies on one side of H_k and zero otherwise. We use the following hashing function

$$h(x) = \sum_i a_i H_i(x),$$

where $a_1 \dots a_m$ are independent and identically distributed random variables with uniform distribution over $\{0, \dots, n^c\}$, where c is a constant to be specified shortly. Note, that if $p, q \in \Re^d$ lie in two different full-dimensional faces of the arrangement of $\mathcal{H}$, then, as noted above, there must be a hyperplane $H_k \in \mathcal{H}$, so that $H_k(p) \neq H_k(q)$, and say that $H_k(p) = 1$. That is, $h(p) = h'(p) + a_k$ and $h(q) = h'(q)$, where $h'(x) = \sum_{i \neq k} a_i H_i(x)$. Since the a_i were picked independently, it follows that $h(p) = h(q)$ only if $h'(p) - h'(q) = a_k$. But the probability of that to happen is $1/n^c$. We conclude, that the probability of two points belonging to two different faces to be mapped to the same value by $h(\cdot)$ is $1/n^c$. Thus, since we have $O(n^2)$ pairs of points to consider in our algorithm, it follows that the probability of the hashing to fail is n^{2-c} which can be made to be arbitrarily small by picking c to be large enough.

In order to construct $h(x)$, we associate a weight a_i with each half-space induced by a hyperplane H_i. For each point p_j, we compute the total weight of all the half-spaces that contain it, and all the points having the same total weight are associated with the same label. Computing the weight of a point p_j falls into the class of problems known as intersection-searching [Aga97]. In particular, one can construct a data-structure in $O(m^{1+\delta})$ time, so that one can answer intersection-searching queries in $O((n/m^{1/d}) \log^{d+1} n)$ time, where $\delta > 0$ is arbitrarily small constant. As the algorithm needs to perform a linear number of such queries, we set $m = n^{2d/(d+1)}$. Thus, the algorithm computes the required labels in $O(n^{2d/(d+1)+\delta})$ time. We conclude:

Theorem 30 *By performing a $O(n^{2d/(d+1)+\delta})$-time preprocessing, where $\delta > 0$ is arbitrary constant, one can reduce the problem of maintaining dynamic $(1+\epsilon)$-approximate nearest neighbor for any n-point crossing metric over n hyperplanes in*

$\Re^d$, *to the problem of maintaining dynamic* $(1+\epsilon)(1-O(1)/\log n)$*-approximate nearest neighbor in Hamming space over unbounded alphabet with* $O(\log^4 n)$ *dimensions.*

Remark 5 *Note, that the constants in the bounds of Theorem 30 depend exponentially (or worse) on the dimension d.*

Chapter 6

General metrics

In this section we will investigate what happens if we abandon the assumption that the data sets are living in a normed (or norm-embeddable) space, and we focus on the case when $D(\cdot,\cdot)$ is a general distance function. The answer to this question is somewhat complex and seems to depend very heavily on the problem. For example, for the closest pair problem, it is very easy to see that there is *no* algorithm running in time $o(n^2)$ producing a pair with *any* approximation ratio ! To see this, consider a metric M in which all distances are equal to 1, except for one pair of points whose distance is set to arbitrarily small $\epsilon > 0$. One can verify that M is indeed a metric. However, clearly any algorithm which finds the "hidden pair" needs $\Omega(n^2)$ time on the average. This contrasts with our earlier results from Chapter 3, where we observed that the closest pair can be approximately solved in subquadratic time for several norms.

The basic difficulty in designing the algorithms for general metric spaces seems to be the fact that the size of the *input* is $\Omega(n^2)$; e.g., we need $\binom{n}{2}$ bits in order to fully describe a metric with all distances equal to either 1 or 2. Therefore, any algorithm which runs in subquadratic time needs to work based on incomplete information.

Nevertheless, we will show that for several problems one can design algorithms which produce answers in subquadratic time (the results were first published in [Ind99a, Ind99b, TI00]). These problems are: k-median, furthest pair, maximum spanning tree, maximum traveling salesman problem, minimum routing cost spanning tree,

problem	output	objective
maximum traveling salesman (MaxTSP)	a tour T	$\max \sum_{e \in T} d(e)$
maximum spanning tree (MaxST)	a tree T	$\max \sum_{e \in T} d(e)$
minimum routing cost spanning tree (MRCST)	a tree T	$\min \sum_{p,q} d_T(p,q)$
average distance (AvgD)	$\frac{1}{\binom{\lvert X \rvert}{2}} \sum_{p,q} d(p,q)$	

Table 6.1: The definitions of the problems.

average distance and 2-clustering (see the definitions in Table 6 and Preliminaries). No subquadratic-time algorithm has been known before for any of these problems, although the (almost trivial) algorithm for furthest pair is based on a "folk theorem". In case of 2-clustering, it was even not known if a subexponential time $(1+\epsilon)$-approximation algorithm for this problem exists (see Section 6.7 for the history of the problem).

Three of the problems have exact $O(n^2)$-time algorithms (MaxST, AvgD, furthest pair), while the other ones are NP-hard.

We mention that minimum routing cost spanning tree (MRCST) has applications to network design [WLB+98] and molecular biology. The latter application follows from its connection to multiple sequence alignment (MSA), where the goal is to find a common alignment tree of n genetic sequences. Specifically, Gusfield [Gus93] showed that any tree within cost at most c times the sum of all distances between the sequences can be used to find an alignment within factor c away from optimal. The alignment computation step is linear in n and thus our tree computation procedure speeds up Gusfield's algorithm as well.

The results of this chapter are given in Table 6.2. Almost all of our algorithms employ random sampling (furthest pair is the exception). There are two basic properties of a random sample of a metric space points which we use. The first property is that "large" distances cannot "hide" in a metric space, unlike the "small" ones. More specifically, if there is any pair of vertices of distance Δ, then there are at least n other pairs of distance at least $\Delta/2$ (by triangle inequality). Thus we can spot one such a pair by random sampling or inspection of a neighborhood of one vertex; this for example gives an immediate 1/2-approximation algorithm for the Furthest Pair,

problem	running time	approximation factor
furthest pair	$O(n)$	$\frac{1}{2}$
k-median [1]	$\tilde{O}(nk/\delta^2)$	$[(1+\delta)(2+4\alpha), 2\beta]$
1-median (Fermat-Weber problem)	$O(n/\delta^2)$	$1+\delta$
MaxTSP	$O(n/\delta)$	$\frac{1}{2}-\delta$
MaxST	$\tilde{O}(n/\delta)$	$\frac{1}{2}-\delta$
MRCST	$O(n/\delta)$	$2+\delta$
AvgD	$O(n/\delta^{7/2})$	$1+\delta$
2-clustering	$O(n2^{1/\epsilon^{O(1)}}\log^{O(1)} n)$	$1+\epsilon$

Table 6.2: Results for general metric spaces.

if we assume Δ is the diameter. Another property we use is that a randomly sampled edge e from a set S has expected length $D(e)$ equal to the average weight of edges in S. This can be used for finding approximation of S or its cost, provided that we can prove additional properties of the expectation or variance of $D(e)$.

6.1 1-median and extensions

In this section, we will prove that we can find a $(1+\epsilon)$-approximate 1-median by computing all distances from $O(\log n/\epsilon^2)$ random points. This fact alone implies an $O(n\log n/\epsilon^2)$ approximation algorithm for the 1-median; we will show later how to obtain a linear time algorithm. Then we will show additional application of this fact to computing approximate maximum cut.

Given a metric (X, D), for $a \in X$, $Q \subseteq X$ define

$$S_Q(a) = \sum_{x\in Q} D(a,x)$$

and $S(a) = S_X(a)$.

Theorem 31 *Let $\epsilon \leq 1$. Let a, b be a pair of arbitrary points in X with $S(b) > (1+\epsilon)S(a)$. Then*

$$\Pr\left(\sum_{x\in Q} D(a,x) \geq \sum_{x\in Q} D(b,x)\right) < e^{-\epsilon^2|Q|/64} \tag{6.1}$$

Note that this implies that if we take $|Q| = \Omega(\log n/\epsilon^2)$ and find point a with the smallest $S(a)$, then a is (with high probability) a $(1+\epsilon)$-approximate median of X.

Proof: To prove the theorem, we study the random variable

$$Y = \sum_{x \in Q} \frac{D(b,x) - D(a,x) + D(a,b)}{2D(a,b)}.$$

By the triangle inequality, this is the sum of random variables between 0 and 1. Our bad event is that $Y \leq |Q|/2$. On the other hand,

$$E(Y) = \frac{|Q|}{|X|} \sum_{x \in X} \frac{D(b,x) - D(a,x) + D(a,b)}{2D(a,b)}.$$

Since $D(b,x) + D(a,x) \geq D(a,b)$ and $\sum_{x \in X} D(x,b) > (1+\epsilon) \sum_{x \in X} D(x,a)$, we have

$$\sum_{x \in X} (D(b,x) - D(a,x)) \geq \sum_{x \in X} \frac{\epsilon}{2+\epsilon} D(a,b)$$

so $E(Y) \geq |Q|(1 + \frac{\epsilon}{2+\epsilon})/2$. Consequently, $|Q|/2 \leq (1-\delta)E(Y)$ with $\delta = \frac{\epsilon}{2+\epsilon}/(1 + \frac{\epsilon}{2+\epsilon}) \geq \epsilon/4$ for $\epsilon \leq 1$. Applying a Chernoff bound we get,

$$\Pr(Y \leq |Q/2|) \leq \Pr(Y \leq (1-\delta)E(Y)) < e^{-\delta^2 E(Y)/2} < e^{-\epsilon^2 |Q|/64}$$

This completes the proof. ■

6.1.1 Faster algorithm

We will now get the running time down to $O(n/\epsilon^2)$ distance queries. We can interpret the previous section as a one round system for identifying a single good 1-median. Here we will make a multiround system, where in each round we will rule out more and more points while using larger and larger samples. More precisely, we will show (the constants are horrible to facilitate easy analysis):

Theorem 32 *The algorithm below finds a $(1+\epsilon)$-optimal 1-median with probability $1 - 1/e$ in $O(n/\epsilon^2)$.*

- $R := X$

- For $i = 1, ..., \lfloor \log_8 n \rfloor$,
  - Pick a set Q of $256(\ln 8 + i + 1)4^i/\epsilon^2$ random points from R
  - Remove the $7/8$ of the points $a \in R$ maximizing $S_Q(a)$
- Among the at most 7 points in R return the one minimizing $S(a)$

Proof: First, concerning the number of queries, in round i, we query $256(\ln 8 + i + 1)4^i/\epsilon^2$ about distances to $n/8^i$ points, hence $O(n/(\epsilon^2 4^i))$, so the total number of queries is $O(n/\epsilon^2)$.

In round i, we will argue that with high enough probability, a point which is at most a factor $1 + \epsilon_i$, $\epsilon_i = \epsilon/2^{i+1}$, times worse than the best before round i survives. As a result, when we are done with all rounds, a point survives which is at most a factor

$$\prod_{i \geq 1}(1 + \epsilon_i) < 1 + \epsilon$$

from optimality. The inequality follows from $\prod_{i\geq 1}(1+1/2^{i+2}) < \prod_{i\geq 1} e^{1/2^{i+1}} = e^{1/2} < 2$.

We will now show that in round i we fail with probability at most $1/e^{i+1}$. Then the probability of failing in any round is

$$\sum_{i \geq 1} 1/e^{i+1} < 1/e$$

We say a point is bad in round i, if it is off by more than a factor $1 + \epsilon_i$. If we can show that the expected number of bad points is at most a factor $1/8e^{i+1}$ we are done by Markov's inequality. By linearity of expectation, we just show that the probability that a bad guy is better than the best, is $1/8e^{i+1}$. Using Theorem 31, we have that this probability is $e^{-(\epsilon/2^{i+1})^2(256(\ln 8+i+1)4^i/\epsilon^2)/64} = 1/(8e^{i+1})$, as desired. ∎

6.1.2 Maximum cut

We start from listing additional implications of Theorem 31. Firstly, observe that for $|Q| \approx \log n/\epsilon^2$, if we sort all points $p \in X$ in the increasing order of $S_Q(p)$ (obtaining, say, a sequence $p_1 \dots p_n$), then we have $S(p_i) \le (1+\epsilon)S(p_j)$ for any $i < j$. In other words, we can approximately sort points p according to their cost.

Having the sequence $p_1 \dots p_n$ approximately sorted, we can quickly approximate the value of $S(p_i)$ for all p_i up to the factor of $(1 + O(\epsilon))$ (note that computing these values exactly would cost n^2 time). To this end, we first find a sequence of *breakpoints* $i_1 \dots i_l$, such that for each p_i we have $i_j \ge i$ such that $(1+4\epsilon)S(p_i) \ge S(p_{i_j})$. It is easy to see that there exists such a sequence with $l = O\left(\log_{1+\epsilon} \frac{S(p_n)}{S(p_1)}\right)$. Since for every p we have $\Delta \le S(p) \le n\Delta$ where Δ is the diameter of X, it follows that $l = O(\log_{1+\epsilon} n)$. In fact, we can find a sequence of breakpoints (of length $O(\log_{1+\epsilon} n)$) by performing $O(\log n \cdot \log_{1+\epsilon} n)$ evaluation of $S(p)$, each taking n steps. The basic idea is to start with $i_1 = 1$ and perform a binary search for $j \ge i_1$ such that $S(p_{j+1}) \ge (1+2\epsilon)S(p_j)$ and $S(p_j) < (1+2\epsilon)S(p_j)$. If $j > i_1$, we set $i_2 = j$; otherwise we set $i_2 = j+1$. Whenever binary search needs to probe certain point p_i, we compute $S(p_i)$ exactly. After we have i_2, we find i_3 in the same way and so on. The binary search takes $O(\log n)$ probes and thus the whole process takes $O(n \log n \cdot \log_{1+\epsilon} n)$.

After the breakpoints are computed, we approximate $S(p_i)$ by $\tilde{S}(p_i) = S(p_{i_j})$, where i_j is the smallest index not smaller than i. Clearly,

$$\frac{\tilde{S}(p)}{(1+4\epsilon)(1+\epsilon)} \le S(p) \le (1+\epsilon)\tilde{S}(p)$$

After computing $\tilde{S}(p)$'s we can finally get to approximating the maximum cut. We will proceed by implementing the algorithm of [dlVK98], which is as follows. Firstly, we scale the distances so that $\sum_{p,q} D(p,q) = n^2$. Let $w(p) \ge 1$ for $p \in X$ be arbitrary weight function with integer values. We create a weighted *clone* graph G, which contains $w(p)$ *clone* of each $p \in X$. Between any pair p', q', where p' is a clone of p and q' is a clone q, we create an edge with weight $\frac{D(p,q)}{w(p)w(q)}$. It was shown in [dlVK98] that if $w(p) = \Theta(S(p))$, then G is a *dense weighted graph*, i.e., the maximum weight (say w_{max}) exceeds the average weight by at most a constant factor. In this case it has been observed that if we replace each weight w in G by $\lfloor c/\epsilon \cdot w/w_{max} \rfloor$ for a constant c (obtaining graph G') and find a $(1+\epsilon)$-approximate maximum cut (S, S') in G, then (S, S') is also a $(1+O(\epsilon))$-approximate maximum cut in G. However, the weights of G' are integers of constant size. Thus, finding approximate maximum cut in G' can be done via the algorithm of [GGR96]. That algorithm samples $1/\epsilon^{O(1)}$ edges of G' and based their weights it creates $2^{1/\epsilon^{O(1)}}$ *oracles*. Each oracle, given any vertex from G', decides (in time $1/\epsilon^{O(1)}$) if it belongs to S or S'. Moreover, at least one of the oracles induces a cut whose weight is at least $1-\epsilon$ times the maximum cut value. Thus, we can find a similar set of oracles for G within the same time bound.

The final and crucial observation of [dlVK98] is that the values of maximum (metric) cut for X and of maximum cut in G are equal[2]. They also provided an algorithm for converting a cut in G into a cut for X. Since their algorithm runs a bit too slow for our needs, we give here a different method which works in $O(n \log n)$ time. Suppose that we are given a cut (S, S') for G'. For $p \in X$ let f_p be the fraction of clones of p in G which belong to S. The value of the cut in G is equal to $\sum_{p,q}[f_p(1-f_q)+(1-f_p)f_q]D(p,q)$. Now, we create a cut for X by putting each point to p to the "left" with probability f_p and to the "right" with probability $1-f_p$. It is easy to see that the expected value of the resulting cut (say (L, R)) is exactly equal to the the value of the cut in G. Since (L, R) is approximately a maximum cut, it follows that with a constant probability we obtain a cut with a value at least $1-\epsilon$ time the value of the cut in G (see Lemma 32).

It is easy to see that if instead of the exact values of f_p we use instead estimations $\tilde{f}_p = f_p \pm \epsilon$, the value of the cut is still preserved up to a factor of $1 \pm O(\epsilon)$. However, we can find such $\tilde{f}_p$'s (correct with high probability) by randomly sampling $O(\log n/\epsilon^2)$ clones of p and checking if they belong to S or S'. Note that each such check can be done in $1/\epsilon^{O(1)}$ time, thus the total time needed to create (L, R) from (S, S') is $O(n \log n/\epsilon^{O(1)})$.

To recap, an approximate maximum cut in X is created by the following sequence of steps:

- estimating $S(p)$: time $O(n \log^2 n/\epsilon)$
- choosing an oracle for the cut (S, S'): multiplies the time of the remaining procedures by $2^{1/\epsilon^{O(1)}}$

[2]Strictly speaking, they showed that the values are very close, due to the fact that they did not assume $w(p)$ is an integer, which created rounding problems. Since the latter fact is true in our case, we can prove the equality.

- estimating f_p from (S', S): time $O(n \log n/\epsilon^2)$
- choosing a cut for X: time $O(n)$

Altogether, the algorithm runs in $O\left((2^{1/\epsilon^{O(1)}} + \log n) n \log n\right)$ and produces a $(1+\epsilon)$-approximate maximum cut of X with a constant probability.

Theorem 33 *A $(1+\epsilon)$-approximate metric maximum cut can be computed in $O(2^{1/\epsilon^{O(1)}} n \log^{O(1)} n)$ time.*

6.2 k-median

In this section we present an $O(n)$-time approximation algorithm for the k-median problem. Our procedure uses a black-box $[\alpha, \beta]$-approximation $\tilde{O}(n^2)$-time algorithm, which we refer to as BB. Let $s = a\sqrt{kn \log k}$ for $a > 1$ determined later. The algorithm is as follows.

1. Choose a set S of s points sampled without replacement from X
2. Run the BB k-median algorithm on S, let $C' = c'_1 \ldots c'_{\beta k}$ denote the output
3. Assign each point $p \in X$ to a point in C' within smallest distance from p; let $D(p, C')$ denote that distance
4. Select the set M containing the points p with $m = b\frac{kn}{s} \log k$ largest values of $D(p, C')$, for b determined later
5. Run the BB algorithm on M, let C'' denote the output
6. Output C' and C''

In the following we will use $a = \Theta(1/\delta\sqrt{\log 1/\delta})$ and $b = \Theta(1/\delta^2 \log 1/\delta)$. It is easy to verify that the running time of the algorithm is $\tilde{O}(kn/\delta^2)$. It remains to prove its correctness.

Theorem 34 *For any constant $\delta > 0$ the above algorithm computes a $[(1+\delta)3(2+\alpha), 2\beta]$-approximate solution for the k-median problem with probability $\Omega(\delta)$.*

By running the above procedure $O(1/\delta)$ times and taking the solution with the smallest cost, we can reduce the error probability to any constant.

Proof: The 2β factor is clear, thus we focus on proving the first bound. Let $c_1 \ldots c_k$ denote the optimal medians and let D denote the optimal cost. Let C_i denote the set of points in X closer to c_i than to any other c_j (for simplicity assume there are no ties). Also, let $C'_i = S \cap C_i$. Let $t = b\frac{n}{s}\log k$. Let $L = \{i : |C_i| \geq t\}$, $l = \sum_{i \in L} |C_i|$ and $l' = \sum_{i \in L} |C'_i|$. Clearly $l \geq n - kt$.

Claim 8

$$\Pr[\sum_{i \in L} \sum_{p \in C'_i} D(c_i, p) \geq \frac{s}{n}(1+\epsilon) \sum_i \sum_{p \in C_i} D(c_i, p)] \leq \frac{1}{1+\epsilon}$$

Proof: Consider any sampled element p. With a probability of $\frac{l}{n}$ this element belongs to C_i for $i \in L$. Conditioned on this event, the expected value of $D(p, C)$ is $\frac{1}{l}\sum_{i \in L}\sum_{p \in C_i} D(c_i, p)$. Thus the expected value of $\sum_{i \in L}\sum_{p \in C'_i} D(c_i, p)$ is $s\frac{1}{l}\frac{l}{n}\sum_i \sum_{p \in C_i} D(c_i, p)$. The Claim follows from Markov inequality. ∎

Claim 9 *For any $\gamma > 0$ we have*

$$\Pr[\textit{for every } i \in L, \frac{|C_i|}{|C'_i|} \leq \frac{n}{s}(1+\gamma)] \geq 1 - 2ke^{\frac{-st}{3n\gamma^2}}$$

Proof: Follows from Chernoff bound. ∎

Assume that both assertions of the above Claims hold (which is true with probability $\Theta(\epsilon)$ for a proper choice of a and b). We show that the centers C' provide a good solution for all points from $\cup_{i \in L} C_i$. Fix any such C_i and consider any function $q_i : C_i \rightarrow C'_i$ such that any element from C'_i has at most $\frac{|C_i|}{|C'_i|}$ elements assigned to it. For any $q \in C'_i$ let $c'(q)$ denote the median c'_i closest to q. By triangle inequality

$$D(p, C') \leq D(p, c_i) + D(c_i, q_i(p)) + D(q_i(p), c'(q_i(p))).$$

Thus

$$\begin{aligned}
\sum_{i\in L}\sum_{p\in C_i} D(p,C') &\le \sum_{i\in L}[\sum_{p\in C_i} D(p,c_i) \\
&\quad + \frac{|C_i|}{|C_i'|}\sum_{q\in C_i'}(D(c_i,q)+D(q,c'(q)))] \\
&\le D+\frac{n}{s}(1+\gamma)(1+2\alpha)\sum_{i\in L}\sum_{p\in C_i'} D(c_i,p) \\
&\le D+\frac{n}{s}(1+\gamma)(1+2\alpha)(1+\epsilon)\frac{s}{n}D \\
&= (1+\gamma)(1+\epsilon)(2+2\alpha)D \\
&= (1+\delta)(2+2\alpha)D
\end{aligned}$$

As $l = |\cup_{i\in L} C_i| \ge n-kt$, we know that the cost of the points in $X-M$ with respect to medians C' does not exceed $(1+\delta)(2+2\alpha)$ as well. Thus it is sufficient to bound the cost of clustering of M. To this end notice, that there exists a clustering of M with a cost at most $2D$ (just replace each c_i by its closest neighbor in M). Thus the algorithm will find a clustering of M with cost $2\alpha D$ and the total cost does not exceed $(1+\delta)(2+4\alpha)D$. ■

6.3 Maximum spanning tree

In this section we present two approximation algorithms for MaxST, both running in $O(n)$ time. The first one outputs a tree of cost at least $\frac{1}{4}$ times the optimal, while the second one improves the factor to $\frac{1}{2}$.

Let T denote the tree with maximum cost $C(T)$ and let Δ be the diameter of the graph. The first algorithm follows from the following Lemma.

Lemma 31 *A tree T of cost at least $\frac{n}{4}\Delta$ can be found in $O(n)$ time.*

Proof: Let $a, b \in X$ be points such that $D(a,b)=\Delta$. Consider arbitrary point q. As $\Delta = D(a,b) \le D(a,q)+D(q,b)$, we can find a vertex u such that $D(q,u) \ge \frac{\Delta}{2}$. Consider now two stars: S_q centered at q and S_u centered at u. We can lower bound the sum of their costs by

$$\sum_p D(q,p) + D(u,p) \geq \sum_p d(q,u) \geq n\frac{\Delta}{2}$$

Thus one of S_q and S_u has cost at least $\frac{\Delta}{4}n$. ∎

The second algorithm uses similar idea, but the pair (q,u) is found by random sampling rather than using the above algorithm. Specifically, we prove that the longest among the $O(n)$ sampled edges has length close to $\frac{C(T)}{n-1}$. The bound then follows from the above argument.

In order to analyze the sampling algorithm, we need the following Lemma (which can be thought of as a reverse version of Markov inequality).

Lemma 32 *Let X be a random variable with values from the set $[0,B]$ such that $E[X] \geq \frac{B}{a}$ for some $a > 1$. Then for any $\alpha > 0$*

$$\Pr[X \geq (1-\alpha)E[X]] \geq \frac{\alpha}{a}.$$

Proof: Similar to the proof of Markov inequality. ∎

We can now proceed with the proof of correctness of the sampling procedure. Let $e_1 \ldots e_{cn}$ denote cn edges sampled at random; let $X = \max_i D(e_i)$. As for any $\epsilon > 0$ and $c = O(\log 1/\epsilon)$ we have the probability of $1-\epsilon$ of hitting a (random) edge from T, we have $E[X] \geq (1-\epsilon)\frac{C(T)}{n-1}$. Therefore (by Lemma 32) we conclude that with a probability of $\Omega(\alpha)$ we have $X \geq (1-\alpha-\epsilon)\frac{C(T)}{n-1}$. Thus we have proven the following theorem.

Theorem 35 *For any $\delta > 0$ there is a $\frac{1-\delta}{2}$-approximation algorithm for MaxST running in $O(n\frac{\log 1/\delta}{\delta})$ time.*

6.4 Maximum TSP

Let T denote the tour with maximum cost $C(T)$ and let T' be tour chosen uniformly at random from the space of all $(n-1)!$ tours.

Lemma 33 $E[C(T')] \geq \frac{C(T)}{2}$.

Proof: We can write $E[C(T')]$ as $\sum_p \sum_{q \neq p} \frac{D(p,q)}{n-1}$. Fix p and let $T(p)$ denote the successor of p in the tour T. By triangle inequality we know that for any q we have $D(p, T(p)) \geq D(p,q) + D(T(p), q)$ which implies

$$nD(p, T(p)) \geq \sum_q D(p,q) + D(T(p), q).$$

Therefore

$$\begin{aligned} C(T) &= \sum_p D(p, T(p)) \\ &\leq \sum_p \frac{1}{n} \sum_q D(p,q) + D(T(p), q) \\ &= \frac{2}{n} \sum_p \sum_q D(p,q) \\ &\leq 2E[C(T')] \end{aligned}$$

■

Thus by Lemmas 32 and 33 we have the following theorem.

Theorem 36 *For any $\delta > 0$ there is a $\frac{1-\delta}{2}$-approximation algorithm for MaxTSP running in $O(\frac{n}{\delta})$ time.*

6.5 Minimum routing cost spanning tree

In this section we describe the approximation algorithm for the MRCST problem. The algorithm proceeds by choosing $q \in X$ at random and reporting a star S_q rooted at that vertex. Below we show that the probability the cost $C(S_q)$ exceeds $C = 2\sum_{p,q} D(p,q)(1+\delta)$ is at most $1 - \frac{1}{1+\delta}$, thus by repeating this procedure $O(1/\delta)$ times and taking best star we obtain a constant probability of success.

The probability bound is obtained as follows. Consider the expectation $E[C(S_q)]$. We know that

$$E[C(S_q)] \leq \frac{1}{n} \sum_q C(S_q)$$

$$
\begin{aligned}
&= \frac{1}{n}\sum_q \sum_{p,r} D(p,q) + D(q,r) \\
&= 2\sum_{p,r} D(p,r)
\end{aligned}
$$

The probability bound follows by Markov inequality.

6.6 Average distance

In this section we give an approximation algorithm for the average distance problem. The algorithm finds the value of the sum of all the distances (call it A) with multiplicative error $(1+\delta)$ in time $O(n/\delta^{7/2})$. The actual algorithm is simple - we sample a set S (of cardinality $s = an$) of edges by including each edge with probability $\frac{s}{m}$, compute the sum of their length and multiply by $\frac{m}{s}$, where $m = \binom{n}{2}$. The running time of the algorithm is clearly $O(s)$ with high probability. Below we prove that for $s = O(n/\delta^{7/2})$ the result is a good approximation of A. Note that the assumption of (X, D) being a metric is crucial for this result: otherwise one could assign arbitrarily large weight (say w) to one edge, which would not be included in S with high probability and thus the estimation of A would be incorrect. The metric space assumption allows us to avoid this problem, as we know that if $D(p,q) > w$ then for each $a \in X$ either $D(p,x)$ or $D(q,x)$ is greater than $w/2$, thus with a good probability we hit one of these edges.

Let Δ be the diameter of the metric and assume that the minimum interpoint distance is 1. Let $c = 1+\epsilon$, for some $0 < \epsilon < \delta$. Split the interval $[1\ldots\Delta]$ of possible distances into intervals $I_i = [c^i, c^{i+1})$, for $i \geq 0$. Let n_i be the number of distances falling into interval I_i and let s_i be the number of distances from S falling into I_i. Define $\tilde{A} = \sum_i c^i n_i$, $A' = \sum_{e\in S} D(e)$ and $\tilde{A}' = \frac{m}{s}\sum c^i s_i$. Clearly, $A = \tilde{A}(1\pm\epsilon)$ and $A' = \tilde{A}'(1\pm\epsilon)$, thus it is sufficient to show that $\tilde{A}'$ well approximates $\tilde{A}$. To this end notice that $E[\tilde{A}'] = \tilde{A}$, thus it is sufficient to bound the variance of $\tilde{A}'$ and use Chebyshev inequality. The variance $D^2[\tilde{A}']$ can be bounded by

$$\frac{m^2}{s^2}\sum_i c^{2i} n_i \frac{s}{m} = \frac{m}{s}\sum c^{2i} n_i.$$

Thus by Chebyshev inequality we get

$$\begin{aligned} P &= \Pr[|\tilde{A}' - E[\tilde{A}'] \geq \epsilon \cdot E[\tilde{A}]] \\ &\leq \frac{1}{\frac{\epsilon^2 E^2[\tilde{A}']}{D^2[\tilde{A}']}} \\ &= \frac{D^2[\tilde{A}']}{\epsilon^2 E^2[\tilde{A}']} \end{aligned}$$

As $E^2[\tilde{A}'] \geq \sum c^{2i} n_i^2$, it is sufficient to bound from the above the ratio

$$F = \frac{\sum c^{2i} n_i}{\sum c^{2i} n_i^2}.$$

In order to do this we make the following observation (this is the only place where we use the fact that d is a metric). Let $a, b \in X$ be a pair of points such that $D(a,b) = \Delta$. Then for any $p \in X$ we know that $D(a,p) + D(p,b) \geq \Delta$, thus at least one of $D(a,p)$ and $D(p,b)$ is greater than $\Delta/2$. Let k be such that $c^k = \Delta$. By pigeonhole principle there exists $0 \leq j \leq \log_c 2$ such that $n_{k-j} \geq n/\log_c 2$. This enables us to bound F as follows. Let $P = \{i : n_i \geq t\} - \{k-j\}$ for $t = \alpha n$, where α is chosen later. We can write F as $\frac{N_1+N_2}{M_1+M_2}$, where $M_1 = \sum_{i \in P} c^{2i} n_i^2$, $M_2 = \sum_{i \notin P} c^{2i} n_i^2$, $N_1 = \sum_{i \in P} c^{2i} n_i$ and $N_2 = \sum_{i \notin P} c^{2i} n_i$.

Now we observe that $\frac{N_1}{M_1} \leq \frac{1}{t}$ (from the definition) and

$$N_2 \leq t \sum c^{2i} \leq t \frac{c^{2(k+1)}}{c^2 - 1} \leq \frac{\Delta^2 (1+\epsilon)^2}{\epsilon} t$$

while

$$M_2 \geq \left(\frac{\Delta}{2} \frac{n}{\log_c^2 2}\right)^2$$

thus

$$\frac{N_2}{M_2} \leq \frac{1}{n} \frac{4 \log_c^2 2 \alpha (1+\epsilon)^2}{\epsilon}.$$

Therefore

$$F \leq \max\left(\frac{N_1}{M_1}, \frac{N_2}{M_2}\right) = \frac{1}{n} \max\left(\frac{4 \log_c^2 2 (1+\epsilon)^2}{\epsilon} \alpha, \frac{1}{\alpha}\right)$$

and by setting $\alpha = \Theta(\epsilon^{3/2})$ we obtain that $F = O(\frac{1}{\epsilon^{3/2}} \frac{1}{n})$. Thus

$$P = O\left(\frac{1}{\epsilon^2} \frac{m}{an} \frac{1}{\epsilon^{3/2}} \frac{1}{n}\right) = O\left(\frac{1}{a \epsilon^{7/2}}\right)$$

and by setting $\epsilon = \Theta(\delta)$ and $a = O(\frac{1}{\delta^{7/2}})$ we prove the approximation bound. In this way we proved the following Theorem.

Theorem 37 *For any $\delta > 0$, there is a $(1+\delta)$-approximation algorithm for the average distance problem running in $O(\frac{n}{\delta^{7/2}})$ time.*

We also mention that in fact the algorithm works very fast even in case when the metrix (X, D) is *dynamic*, i.e., when we can add and subtract point from X and need an update of the average distance after every operation. To see that, observe that after each operation we just need to maintain the sum of all edges chosen at random, and possibly delete some of them or add some new ones. In each case, the expected number of such edges incident to any vertex (in particular, to the one added/deleted) is $O(1/\delta^{7/2})$, so after constant number of operations we obtain an updated value. Therefore, each update costs only $O(1/\delta^{O(1)})$ time.

6.7 2-clustering

In this section we consider the k-clustering problem for $k = 2$. Recall that the problem is in general defined as follows: given a weighted graph $G = (X, D)$, $|X| = n$, where $D(\cdot, \cdot)$ is a weight function, partition X into k sets $S_1 \dots S_k$ such that the value of

$$\sum_i \sum_{\{u,v\} \subset S_i} D(u, v)$$

is minimized. This problem was first formally posed by Sahni and Gonzalez [SG76]. They observed that the problem is NP-complete (for $k \geq 2$) and by reduction from k-coloring showed that it cannot be approximated up to any constant (for $k \geq 3$). Instead, they proposed a $1/k$-approximation algorithm for the dual version of this problem where the goal is to *maximize* the weight of edges which *do not* belong to any cluster (i.e. k-max cut). Unfortunately, the latter result does not (and cannot) imply any bounded approximation ratio for the k-clustering problem. The hardness of k-clustering (for $k \geq 3$) was later strengthened by Kann et al [KKLP96]. The case $k = 2$ is also known as the *minimum edge deletion bipartition problem.* Garg et al [GVY95] gave a $O(\log n)$-approximation algorithm for this case.

A standard way to restrict the complexity of clustering problems is to assume that the weight function D is a metric. For this case Guttman-Beck and Hassin [GBH] showed a 2-approximation algorithm running in time $n^{O(k)}$ (they also gave slightly improved bounds for the case $k = 2$ assuming $|S_1| = |S_2|$). Recently, de la Vega and Kenyon [dlVK98] gave a polynomial time approximation scheme for the dual of metric 2-clustering (i.e. metric max cut). Unfortunately (as they mention) their result does not imply an approximation scheme for the 2-clustering problem.

In this paper we show an approximation scheme for the 2-clustering problem. We obtain this result by focusing on the case when the de la Vega-Kenyon algorithm does not work, i.e., when the value of max cut is much higher than the value of the 2-clustering. We observe that in this case the clusters must be "well-separated", i.e. the average intracluster distance (for at least one cluster) has to be much smaller than the average intercluster distance. Therefore, we can assign a vertex to one of the two clusters by examining its average distance to "central" points from each cluster (we find these points by random sampling). If done correctly, this procedure yields a $(1 + \epsilon)$-approximate randomized algorithm.

Before we describe the algorithm, we need to introduce new notation. Let $M = (X, D)$ be a metric space. For any two sets $A, B \subset X$ we define

$$D_s(A, B) = \sum_{a \in A, b \in B} D(a, b).$$

We use $D_s(A)$ to denote $D_s(A, A)/2$ (notice that this is equal to the sum of pairwise distances of points in A). Also, we will write $D_s(u, B)$ or $D_s(B, u)$ instead of $D_s(\{u\}, B)$ or $D_s(B, \{u\})$. Moreover, we define

$$\tilde{D}_s(A, B) = \frac{D_s(A, B)}{|A| \cdot |B|}$$

Observe, that $\tilde{D}_s$ is a metric, while D_s is not. One can interpret $\tilde{D}_s(A, B)$ as the expected distance from a random point in A to a random point in B.

We now proceed with the algorithm. For simplicity, we will show how to get a $1 + O(\epsilon)$-approximation (instead of a factor $1 + \epsilon$). Using the above notation, we want to partition X into two sets S_1, S_2 such that $D_s(S_1) + D_s(S_2) = D_s(X) -$

$D_s(S_1, S_2)$ is minimal. Let (A, B) denote an optimal pair (S_1, S_2). Observe that if $D_s(A, B)\epsilon^2 \le D_s(A) + D_s(B)$, then a $(1 + \epsilon^3)$-approximate solution for maximizing $D_s(S_1, S_2)$ also yields a $(1+O(\epsilon))$-approximate solution for minimizing $D_s(A)+D_s(B)$. The latter problem can be done fast using the algorithm of de la Vega-Kenyon, using our implementation described in Section 6.1. Therefore, we can focus on the case when $D_s(A, B)\epsilon^2 > D_s(A) + D_s(B)$, i.e., the cost of 2-clustering is much smaller than the cost of the maximum cut.

Assume that $|A| \ge |B|$. Also, we assume that we know the cardinality of A (and therefore of B), since we can enumerate all n values of A in polynomial time and pick the best value (later we will show how to reduce the number of guesses to $O(\log n)$). We will consider two cases:

1. $\epsilon|A| \le |B|$: in this case we apply the algorithm BC (for Balanced Cut), which is as follows. Firstly, we find (via random sampling) two points z_A and z_B such that $D_s(z_A, A) \le 2D_s(A, A)/|A|$ and $D_s(z_B, B) \le 2D_s(B, B)/|B|$; one can observe that a random pair of points z_A, z_B satisfies these conditions with probability $\Omega(|B|/|X|) = \Omega(\epsilon)$. Then, for each $p \in X$ we add p to S_1 if $|B|D_s(p, z_B) > |A|D_s(p, z_A)$ and to S_2 otherwise.

2. $|B| \le \epsilon|A|$: in this case we apply the algorithm UC (for Unbalanced Cut), which is as follows. We start from finding (again, via random sampling) a points $z \in A$ such that $D_s(z, A) \le 2D_s(A, A)/|A|$; note that a random point z satisfies this condition with constant probability. Then we set S_1 to contain the $|A|$ points of X which are closest to z.

Both algorithms can be clearly implemented in polynomial time (in Section 6.7.1 we show how to implement them in near-linear time). Now we proceed with the proofs of correctness. Let (S_1, S_2) be the solution found by the algorithm. We define $A' = S_1 - A$, $A'' = S_1 \cap A$, $B' = S_2 - B$ and $B'' = S_2 \cap B$; note that $|A'| = |B'|$. In both cases our proof will follow similar scheme, namely we will bound the errors induced by misclassifying A' (equal to $D_s(A', B'') - D_s(A', A'')$) and misclassifying B' (equal to $D_s(B', A'') - D_s(B', B'')$).

Unbalanced cut. From the definition of the algorithm it follows that $\tilde{D}_s(A', z) \geq \tilde{D}_s(B', z)$. We will bound the misclassification error. We start from $\tilde{D}_s(A', B'')$. Observe that

$$\begin{aligned}\tilde{D}_s(A', B'') &\leq \tilde{D}_s(A', z) + \tilde{D}_s(z, B') + \tilde{D}_s(B', B'') \\ &\leq 2\tilde{D}_s(A', z) + \tilde{D}_s(B', B'') \\ &\leq 2\tilde{D}_s(A', A) + 2\tilde{D}_s(z, A) + \tilde{D}_s(B', B'')\end{aligned}$$

By replacing $\tilde{D}_s$ by D_s we obtain

$$\begin{aligned}D_s(A', B'') &\leq 2D_s(A', A)\tfrac{|B''|}{|A|} + 2D_s(z, A)\tfrac{|A'||B''|}{|A|} + D_s(B', B'')\tfrac{|A'|}{|B'|} \\ &\leq 2D_s(A, A) \cdot \epsilon + 4D_s(A, A)\tfrac{|A'||B''|}{|A|^2} + D_s(B', B'') \\ &\leq 2D_s(A)(2\epsilon + 4\epsilon) + D_s(B', B'')\end{aligned}$$

Thus $D_s(A', B'') - D_s(B', B'') \leq 12\epsilon D_s(A)$. Now we proceed with $D_s(B', A'') - D_s(A', A'')$. We have

$$\begin{aligned}\tilde{D}_s(B', A'') &\leq \tilde{D}_s(B', z) + \tilde{D}_s(z, A'') \\ &\leq \tilde{D}_s(A', z) + \tilde{D}_s(z, A'') \\ &\leq \tilde{D}_s(A', A'') + 2\tilde{D}_s(z, A')\end{aligned}$$

By replacing $\tilde{D}_s$ by D_s we get

$$\begin{aligned}D_s(B', A'') &\leq D_s(A', A'')\frac{|B'|}{|A'|} + 2D_s(A'', z)|B'| \\ &\leq D_s(A', A'') + 2D_s(A, z)|B'| \\ &\leq D_s(A', A'') + 4D_s(A, A)\frac{|B'|}{|A|} \\ &\leq D_s(A', A'') + 8\epsilon D_s(A)\end{aligned}$$

Thus $D_s(B', A'') - D_s(A', A'') \leq 8\epsilon D_s(A)$.

Balanced cut. In this case, we will show first that $|A'| \leq O(\epsilon^2)|A|$, which we do as follows. Firstly, observe that from our algorithm it follows that $|B|\tilde{D}_s(A', z_B) \leq |A|\tilde{D}_s(A', z_A)$ and $|A|\tilde{D}_s(B', z_A) \leq |B|\tilde{D}_s(B', z_B)$. Then

$$\tilde{D}_s(A, B) \leq \tilde{D}_s(A, z_A) + \tilde{D}_s(z_A, B') + \tilde{D}_s(B', B)$$

$$
\begin{aligned}
&\leq \tilde{D}_s(A, z_A) + \tilde{D}_s(z_B, B')\frac{|B|}{|A|} + \tilde{D}_s(B', B) \\
&\leq \tilde{D}_s(A, z_A) + \left(\tilde{D}_s(z_B, B) + \tilde{D}_s(B, B')\right)\frac{|B|}{|A|} + \tilde{D}_s(B', B) \\
&\leq \tilde{D}_s(A, z_A) + \tilde{D}_s(z_B, B)\frac{|B|}{|A|} + 2\tilde{D}_s(B', B)
\end{aligned}
$$

Replacing $\tilde{D}_s$ by D_s we get

$$
\begin{aligned}
D_s(A, B) &\leq D_s(A, z_A)|B| + D_s(z_B, B)|B| + 2D_s(B', B)\frac{|A|}{|B'|} \\
&\leq 2D_s(A, A) + 2D_s(B, B) + 2D_s(B, B)\frac{|A|}{|B'|} \\
&\leq 4\left(1 + \frac{|A|}{|B'|}\right)(D_s(A) + D_s(B))
\end{aligned}
$$

Since

$$1/\epsilon^2 \cdot (D_s(A) + D_s(B)) \leq D_s(A, B) \leq 4\left(1 + \frac{|A|}{|B'|}\right)(D_s(A) + D_s(B))$$

we conclude that

$$\frac{|A|}{|A'|} = \frac{|A|}{|B'|} \geq \frac{1/\epsilon^2}{4} - 1.$$

Now we proceed with bounding the misclassification error. We start from $D_s(A', B'')$. Observe that

$$
\begin{aligned}
\tilde{D}_s(A', B'') &\leq \tilde{D}_s(A', z_B) + \tilde{D}_s(z_B, B'') \\
&\leq \tilde{D}_s(A', z_A)\frac{|A|}{|B|} + \tilde{D}_s(z_B, B'') \\
&\leq \left(\tilde{D}_s(A', A'') + \tilde{D}_s(A'', z_A)\right)\frac{|A|}{|B|} + \tilde{D}_s(z_B, B'')
\end{aligned}
$$

Replacing $\tilde{D}_s$ by D_s we obtain

$$
\begin{aligned}
D_s(A', B'') &\leq D_s(A', A'')\frac{|A|}{|A''|} \cdot \frac{|B''|}{|B|} + D_s(A, z_A)\frac{|A|}{|A''|} \cdot \frac{|A'||B''|}{|B|} + D_s(z_B, B'')|A'| \\
&\leq D_s(A', A'') \cdot \frac{1}{1 - O(\epsilon^2)} + 2D_s(A, A) \cdot \frac{1}{1 - O(\epsilon^2)} \cdot \frac{|A'|}{|A|} + 2D_s(B, B)\frac{|A'|}{|B|} \\
&\leq D_s(A', A'')\left(1 + O(\epsilon^2)\right) + 4D_s(A)\left(1 + O(\epsilon^2)\right) \cdot O(\epsilon^2) + 4D_s(B)O(\epsilon) \\
&\leq D_s(A', A'')\left(1 + O(\epsilon^2)\right) + 4\left(1 + O(\epsilon^2)\right) \cdot O(\epsilon) \cdot (D_s(A) + D_s(B))
\end{aligned}
$$

Thus

$$D_s(A', B'') - D_s(A', A'') \leq O(\epsilon)\,(D_s(A) + D_s(B))$$

By using a symmetric argument we can show

$$D_s(B', A'') - D_s(B', B'') \leq O(\epsilon)\,(D_s(B) + D_s(A))$$

Therefore, both UC and BC incur only an additive error of $O(\epsilon(D_s(A) + D_s(B)))$ when their respective assumptions are satisfied. Therefore, either UC or BC or the algorithm of Section 6.1.2 returns an $(1 + O(\epsilon))$-approximate minimum clustering, so we can run all of them in parallel to obtain an approximate solution in polynomial time.

6.7.1 Fast implementation

In order to reduce the running time, we first identify the major bottlenecks in the running time of both algorithms, which are as follows:

- guessing the value of $|B|$ multiplies the remaining running time by n
- computing a cost a given partition (S_1, S_2) (for a fixed guess $|B|$) takes roughly time $|S_1|^2 + |S_2|$, which is $\Theta(n^2)$

We will deal with these two issues differently for each of the algorithms. We start from BC. Firstly, we observe that in fact we do not need to guess the exact value of $|B|$, but it is just sufficient to guess a value $b \leq |B| \leq (1+\epsilon)b$, and run BC with $|A|$ replaced by $n - b$ and $|B|$ by b; note that this requires making only $O(\log_{1+\epsilon} n)$ guesses. The proof that the resulting algorithm still guarantees a $(1 + O(\epsilon))$-approximation is essentially the same as before, so instead of reproving the approximation bound, we just point out that in the right hand sides of the inequalities in the proof of BC's correctness, $|A|$ and $|B|$ occur only as positive terms (i.e., are not involved in any subtraction). Therefore, changing any of them by a factor $(1 \pm \epsilon)$ changes the upper bound by at most $(1+\epsilon)^{O(1)}$.

The second problem (i.e., evaluating the cost of a given clustering) can be quickly resolved by using the approximate algorithm from Section 6.6. Therefore, the final running time of BC is $O(n \log n/\epsilon^{O(1)})$.

Now consider the algorithm UC. Unfortunately, here we need to know the exact value of A, since otherwise the final outcome could be very bad if A and B are very separated (note that UC *forces* the size of both clusters). Therefore, we need to iterate over all $O(n)$ potential values of $|A|$. However, we can still achieve fast running time by careful examination of what our algorithm. To this end, observe that UC can be stated as follows:

1. Choose a random point z
2. Sort all points $p \in X$ in the increasing order of $D(z,p)$, creating a sequence $p_1 \ldots p_n$
3. For $i = n - \epsilon n \ldots n$, include first i points to S_1, the remaining ones to S_2; output the cut with the smallest cost

Steps 1 and 2 can be implemented in $O(n \log n)$ time. Moreover, increasing i by one in Step 3 is implemented by removing p_{i+1} from S_2 and moving it to S_1. Updating the approximate value of $D_s(S_1) + D_s(S_2)$ therefore only requires two update operations on data structure maintaining average interpoint distance. Such a data structure (with expected update time $O(1/\epsilon^{O(1)})$) has been also given in Section 6.6. Thus, the algorithm UC can be implemented to run in time $O(n(\log n + 1/\epsilon^{O(1)}))$.

By combining the times for UC, BC and the maximum cut algorithm of Section 6.1.2, we obtain the following theorem.

Theorem 38 *A $(1+\epsilon)$-approximate 2-clustering problem can be solved in $O(2^{1/\epsilon^{O(1)}} n \log^{O(1)} n)$ time.*

6.8 Lower bounds

In this section we investigate limitations of sublinear time algorithms in metric spaces. Our results are depicted in the Table 6.8; they hold for randomized algorithms. When combined with our algorithms, one can observe an interesting phenomenon: the minimization problems are difficult to approximate, while the maximization problems are approximable to within a small constant factor. Intuitively, this is due to the fact that small distances can be easily "hidden" in the metric space, while the triangle inequality prevents large distances from "hiding".

problem	approx. factor	not achievable in time
closest pair	any	$o(n^2)$
furthest pair	$> \frac{1}{2}$	$o(n^2)$
MaxST/MaxTSP	$> \frac{1}{2}$	$o(n^2)$
MinST/MinTSP	$O(1)$	$o(n^2)$

Table 6.3: Lower bounds for metric space problems.

Proof: The proofs are as follows

closest pair : Set all distances to 1 except for one edge chosen at random, which is set to $\epsilon > 0$

furthest pair : Set all distances to 1 except for one edge chosen at random, which is set to 2

MaxST/MaxTSP : Set all distances to 1 except for edges on a random path of length $n-1$, which are set to 2. The optimal cost of both MaxST and MaxTSP is $(2-\epsilon)n$, but finding δn edges set to 2 for any fixed $\delta > 0$ requires $\Omega(n^2)$ time.

MinST/MinTSP : For any $B = O(1)$ choose a random path of length $n-1$, set all edges on that path to 1 and consider the metric obtained by taking the shortest paths metric induced by the unit edges and limiting the maximum distance to some B. One can observe that the optimum solution has cost $n + B$. On the other hand, finding δn edges with cost $< B$ for any fixed $\delta > 0$ requires $\Omega(n^2/B)$ time.

■

Chapter 7

Applications

Up till now the focus of this thesis has been predominantly theoretical. In this chapter we depart from this approach and describe several situations where the algorithms from this thesis have been applied. Since the number of possible settings where the algorithms can be used is roughly equal to the total number of pairs (problem, metric), which is quite large, our experimental experience with the algorithms certainly will not be exhaustive. Instead, we will describe several "case studies", i.e. scenarios in which some of the algorithms have been applied and discuss the results.

7.1 Locality sensitive hashing

A majority of the applications in this chapter uses various forms of locality sensitive hashing (LSH) from Section 2.2.2. We will group them into two categories: LSH for the Hamming metric and LSH for other measures.

7.1.1 LSH for the Hamming metric

Color histograms. The first experimental evaluation of LSH as a tool for performing approximate nearest neighbor search appeared in [GIM99]. In that paper we have implemented the LSH for the *secondary storage* case, i.e., when the data resides on disk. This allowed us to compare LSH with many other algorithms described in database literature. We focused on comparing LSH with an algorithm based on SR-tree from [KS97], since in that paper it was shown that the latter data structure compares very favorably with the other data structures.

When the data resides on disk, a natural measure of complexity of an algorithm is the number of disk blocks accessed while processing queries. Both LSH and SR-tree perform only random access to the data. Therefore, the I/O time is proportional to the number of buckets accessed. Since the I/O time dominates the total time, counting the number of blocks accessed gives a fair comparison of the two algorithms.

We assumed that each block is 8KB of data. Each coordinate in our data sets is represented using 1 byte. Therefore, we can store up to $8192/d$ d-dimensional points per block.

We performed experiments on two data sets. Here we focus on the first one (cf. [GIM99] for more information). This data set consists of 20,000 histograms of color thumbnail-sized images of various contents taken from the COREL library. The histograms have been extracted after transforming the pixels of the images to the 3-dimensional CIE-Lab color space [WS82]; the property of this space is that the distance between each pair of points corresponds to the perceptual dissimilarity between the colors that the two points represent. Then we partitioned the color space into a grid of smaller cubes, and given an image, we create the color histogram of the image by counting how many pixels fall into each of these cubes. By dividing each axis into u intervals we get a total of u^3 cubes (for most experiments we assumed $u = 4$ obtaining a 64-dimensional space). Each histogram cube (i.e., color) then corresponds to a dimension of space representing the images. At the end quantization is performed in order to fit each coordinate in 1 byte.

The distance between two histogram vectors is usually measured by the Euclidean norm. However, for our data set we observed that the nearest neighbor of an average query point computed under the l_1 norm was also an ϵ-approximate neighbor under the l_2 norm with an average value of ϵ less than 3%. Moreover, for 67% of the queries the nearest neighbors under l_1 and l_2 norms were *exactly* the same. Therefore, instead of embedding l_2 into l_1 and then into Hamming space, we just treated the vectors as members of l_1 space and embedded them into the Hamming cube as in Chapter 4.

Note that the *exact* nearest neighbor was defined with respect to the l_2 norm.

The error of LSH is measured as follows. Following [AMN+94] we define the *effective error* as

$$E = \frac{1}{|Q|} \sum_{\text{query } q \in Q} \cdot \frac{\text{(the distance from } q \text{ to a point found by LSH)}}{\text{(the distance from } q \text{ to the closest point)}}$$

where the sum is taken of all queries for which a nonempty index was found. We also measure the (small) fraction of queries for which no non-empty bucket was found; we call this quantity *miss ratio*. In our experiments, e.g., for $E = 5\%$, the miss ratios were quite high (10%) for small data size, but decreased to around 1% for the whole data set.

Below we show two experimental results. In the first one we show the influence of the number of indices L on E. The results (for the whole data set) are shown on Figure 7.1.

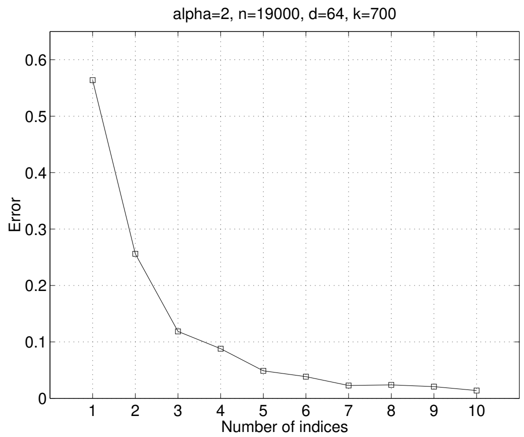

Figure 7.1: Error vs. the number of indices.

As expected, one index is not sufficient to achieve reasonably small error — the effective error can easily exceed 50%. The error however decreases very fast as L increases. This is due to the fact that the probabilities of finding empty bucket are independent for different indices and therefore the probability that all buckets are empty is exponential in L.

In order to compare the performance of LSH with SR-tree, we computed the minimal number of indices needed to achieve a specified value of error E equal to 2%,

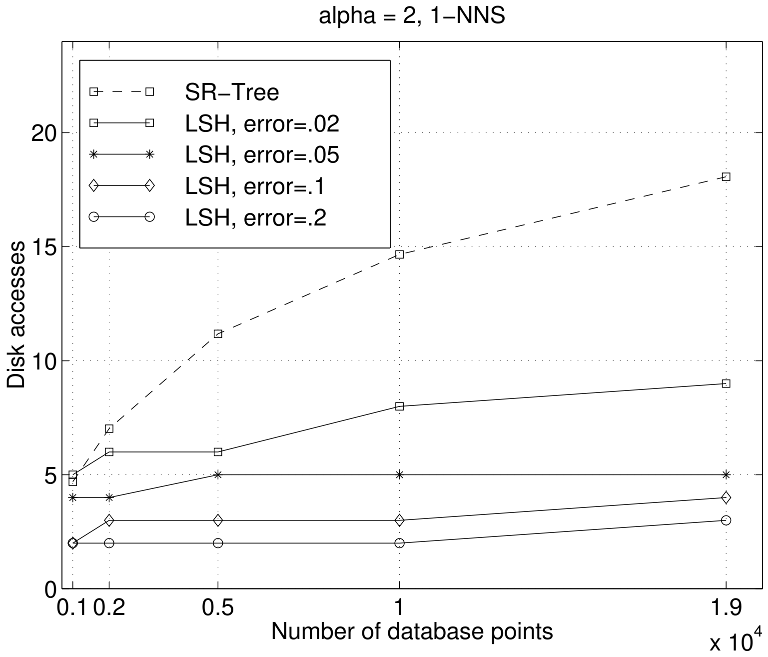

Figure 7.2: Number of indices vs. data size.

5% , 10% or 20%. Then we investigated the performance of the two algorithms while varying the data size. We performed the simulations for $d = 64$ and the data sets of sizes 1000, 2000, 5000, 10000 and 19000. The results are presented on Figure 7.2.

Video sequences. In [IIS99] we applied LSH to finding *pirated video sequences*. The basic idea here is that, give a large collection of video sequences, we would like to find pairs of sequences which are *syntactically* similar. Specifically, we are interested in pairs of sequences such that one sequence has been obtained from the other in an automatic way, e.g., via format conversion, noise addition, changing the aspect ratio of the screen etc. For this kind of applications it turned out that we can represent a video sequence as a set of *shot transitions*, i.e., times at which one scene changed into another one. Then, we computed the distance between two representations (say $T = (t_1 \dots t_k)$ and $T' = (t'_1 \dots t'_l)$) as follows. Assume that t_i's belong to $\{1 \dots d\}$ (this can be easily achieved by a proper discretization). Given T we produce a sequence of d bits $b_1 \dots b_d$ such that $b_i = 1$ iff there exists t_j such that $|t_j - i| \leq a$, for a parameter

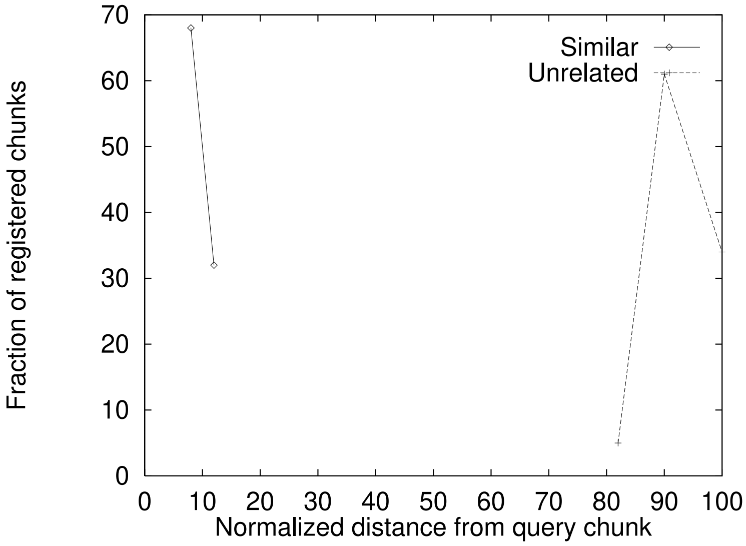

Figure 7.3: Histogram of distances.

a.

This measure has several nice properties. In this section we restrict our attention to the most important one, namely that the distance between two syntactic copies of one sequence should be much smaller than between two unrelated sequences. In Figure 7.3, we show the average distances for pairs of related and unrelated documents.

One can observe that the ratio between the unrelated and related distances is very large, i.e., about 4. Therefore, we can use approximate data structures without compromising the quality. As a result, the running time of our algorithms is about $n^{5/4}$, instead of n^2 if all pairs of video sequences were to be compared.

For more information, (e.g, how to compute the shot transition times, timing experiments, etc.), see [IIS99] or [Shi00].

Genetic sequences. Jeremy Buhler [Buh00] from University of Washington applied the LSH-based close pairs algorithm for finding similar substrings of long genetic sequences. Specifically, given a genetic sequence S (over the alphabet $\{A, G, T, C\}$) of length n, the goal was to find all pairs of substrings of S of length d (called

d-mers) which are within Hamming[1] distance ρd from each other, for some $\rho > 0$. The actual values of the parameters n, d, ρ depend on the genetic sequence; for human chromosome we have $n \approx 34,000,000$, $d = 81$ and $\rho = 0.37$. His experiments showed that the algorithm is more sensitive (i.e., recovers more matches) than known heuristics like BLAST, and is faster than exact algorithms (e.g., from [PW95]). For more information, see [Buh00].

7.1.2 LSH for other measures

The two other experimental papers which we describe here were dealing with a similarity measure somewhat different than Hamming metric. This measure is defined for pairs of *subsets* of some universe U. Given $A, B \subset U$, the similarity between A and B is defined as

$$S(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

This measure can be also extended to multisets. Let $\chi_A(u)$ denote the multiplicity of $u \in U$ in A. Then we define

$$\chi_{A \cap B}(u) = \min(\chi_A(u), \chi_B(u))$$

and

$$\chi_{A \cup B}(u) = \max(\chi_A(u), \chi_B(u))$$

With these definitions $S(A, B)$ extends naturally to multisets.

It is easy to see that if A, B are sets, then $S(A, B)$ is very closely related to the Hamming distance $D(\chi_A, \chi_B)$ between χ_A and χ_B. In particular, we know that

$$|A \cap B| = |A \cup B| - |A \oplus B| = |A \cup B| - D(\chi_A, \chi_B)$$

Thus

$$S(A, B) = 1 - D(\chi_A, \chi_B)/|A \cup B| = 1 - \frac{D(\chi_A, \chi_B)}{a + b - D(\chi_A, \chi_B)}$$

[1]The distance between genetic distance should be properly measured using various variants of edit distance. However, insertions and deletions occur much less frequently than substitutions; therefore, if d is short enough, we can assume only substitution occur [Buh00]

if we set $|A| = a$ and $|B| = b$. Therefore, in principle, we can use any Hamming metric PLEB algorithm to solve problems for $S(A,B)$. However, it is easy to see that solving c-PLEB for Hamming metric does not guarantee c-approximation for $S(.,.)$; in fact the approximation factor could be arbitrarily bad. Therefore, it make sense to use algorithms designed especially for S.

In order to apply LSH to this measure, we will use the following observation [BGMZ97].

Fact 7

$$\Pr_{h:U \xrightarrow{1:1} U} [\min(h(A)) = \min(h(B))] = S(A,B)$$

Therefore, one can use the functions $\min(h(A))$ for a random permutation h as LSH functions. This observation has been made in [IM98] and also announced in [Bro98]; earlier, similar algorithm has been proposed in [KWZ95].

The first experimental evaluation of this method has been given in [CDF⁺00]. The data set considered there contained a log of HTTP requests to web pages of Sun Microsystems registered during the period of about 9 days. We were interested in finding pairs of web pages which have similar user access pattern. Therefore, for each web page p, we extracted the set of IP addresses I_p from which the access to p has been requested. Then, the goal was to find pairs $(I_p, I_{p'})$ such that $S(I_p, I_{p'})$ was greater than a certain threshold $\rho > 0$. To this end, we essentially used the LSH algorithm from Section 2.2.2 with the functions $\min(h(A))$ as the basic LSH functions. The comparison of this algorithm with other techniques showed that the LSH algorithm was running about 2 to 4 times faster than other algorithms, although it reported smaller number of interesting pairs. See [CDF⁺00] for more information.

A more recent application of this method has been published in [HGI00]. In that paper we have used LSH for clustering and similarity search of web documents. Our goal was to cluster together web pages of similar *topics* using *text information*. There are several known tools for solving this problem [BGMZ97, FGMM⁺98, Shi00]. However, their methods can not scale to the case where we are looking for similar, as opposed to almost identical, documents. Computing the document-document similarity matrix essentially requires processing the self-join of the relation $DOCUMENTS(doc, word)$ on the $word$ attribute, and counting the number of words each pair of documents has in common. The clustering algorithms [BGMZ97, FGMM+98], on the other hand, use *shingles* or sentences rather than words to reduce the size of the self-join. Therefore, they can be only applied to *syntactic* clustering, which aims to cluster together (slightly modified) copies of the *same document*, not to the "semantic" clustering, which is our goal. Although clever algorithms exist which do not require the self-join to be represented explicitly [FGMM+98], their running time is still proportional to the size of the self-join. The latter could contain as many as 0.4×10^{13} tuples (for about 35 million pages) if we are looking for similar, rather than identical, documents[2]. Assuming the processing power of, say, 0.5×10^6 pairs per second[3], the running time of those algorithms could easily exceed 90 days.

In [HGI00] we show that using LSH allows us to circumvent the "self-join bottleneck" and make web clustering possible in a matter of a few days on modest hardware (for the same number of web pages), rather than months if the aforementioned techniques were used. Since the details of the algorithms (e.g., the features used to represent the web pages) are rather orthogonal to the topic of this thesis, we refer the reader to the original paper [HGI00].

7.2 String compression

The next application we describe uses the techniques from Section 5.2 for fast distance computation between substrings of a string. More specifically, in [IKM00] we presented an algorithm for computing *relaxed* period of a sequence of numbers, which uses the aforementioned data structure. The relaxed period is a generalization of a standard notion of a period of a string, and it is defined in the following way. Let F be any function which given an ordered sequence of points in $\Re^d$ for some d returns a positive real number. Moreover, for a sequence $t = t_1 \dots t_n$ of real numbers, let $t_{/p}$ be the sequence of at most n/p p-dimensional points $(t_1, \dots t_p), (t_{p+1}, \dots t_{2p})$ etc (we

[2]Our estimation using the sketching approach of [BGMZ97]. Unfortunately, the relation is too large to compute its exact size.

[3]Estimation based on the experiment in [Shi00], page 45

do not include the last point if its dimension is smaller than p). Then we say that p^* is the relaxed period of t if $F(t_{/p^*})$ is minimal over all $p \in \{1 \dots n/2\}$.

The functions F used to define a period could be quite general, but in this section we restrict our attention to functions of the form:

$$F(\{u_1 \dots u_k\}) = G(|u_{i_1} - u_{i'_1}|_q, |u_{i_2} - u_{i'_2}|_q, \dots, |u_{i_l} - u_{i'_l}|_q)$$

where $q \in [1, 2]$ and G is a function computable in time $T(k)$ given a sequence of $l = l(k)$ arguments; we can assume $T(k) \geq l(k)$ and $l(k) \geq k - 1$. This class is fairly general and includes functions like:

- $\sum_i |u_1 - u_i|_q$ - $T(k) = O(k)$
- $\max_i |u_1 - u_i|_q$ - $T(k) = O(k)$
- $\sum_i |u_i - u_{i+1}|_q$ - $T(k) = O(k)$

If we have a data structure which in time $R(n)$ computes a distance between any substrings of t, then we can use it to compute the relaxed period of t in time $O(T(n)R(n)\log n)$. The idea is to enumerate $p = 1 \dots n/2$ and for each p compute the $l(n/p)$ distances and apply the function G. This takes the time

$$\sum_{p=1}^{n} T(n/p)R(n) \leq O(T(n)R(n)\log n)$$

Since in our case $R(n) = O(\log n)$ and $T(n) = O(n)$, we obtain an algorithm running in time $O(n \log^{O(1)} n)$ (we excluded here the dependence on the approximation factor $(1+\epsilon)$).

We have applied the above approach (for $q = 2$) to computing a relaxed period of large sequences, extracted from one of the warehouses maintained at AT&T research. All the data sets are time series containing utilization information of one of the services AT&T provides to customers. The data are collected in the granularity of a second and the time duration ranges from a month (approximately 16MB of data) to a year (approximately 256MB). We assumed that the data resided on disk.

We observed that this approach results in algorithms 3 to 6 times faster than the brute force algorithm, if we search for periods of length up to 8,000. As the potential length of the period increases, the improvement we get becomes larger and larger. This is not surprising, since our algorithm is nearly linear while the brute force algorithm is quadratic. The value of ϵ ranges from 30% (using sketches of length $2 \log n$) to 80% (using sketch length $\log n$). See [IKM00] for more information.

7.3 Stream computation

The last application we describe here is a subject of an on-going research project at AT&T Shannon Labs [Str00]. The idea is to use stream algorithms, including those from Section 5.1, for detecting fluctuations in traffic data. Specifically, the data set consists of a very large number of records, where each record describes a single flow of traffic. The record is of the form (s, d, t, v), where s is the source address, d is the destination address, t is the time period, v is the traffic volume. The goal of the project is build a system which maintains statistical information about the flow data in order to be able to detect unusual user behavior. Since the amount of data is very large, storing all records is not feasible and thus we need to be able to maintain sketches of the data.

A possible approach to this problem is as follows. We can discretize the space of triples (s, t, v) into cubes. For each destination d we maintain a high-dimensional vector which has one coordinate per each cube. Each coordinate counts the number of records (s, d, t, v) generated during certain period of time (say, during one day), such that (s, t, v) falls into the corresponding cube. As a result, for each destination d we obtain a set of points P_d which represent the behavior of the user at destination d during a period of one day. Now, we can say that an unusual behavior occurred during a certain day if the corresponding vector is very far from the other points in P_d.

Since we are only interested in computing distances between the points, we can represent the vectors in compressed form, as in Section 5.1. In this way we can collect large amount information without worrying about the storage.

Chapter 8

Conclusions and open problems

In this thesis we showed how many basic geometric problems can be efficiently solved (approximately) in high dimensional spaces. The algorithms were designed in a modular way. We started from "core" problem in "core" metrics, namely approximate PLEB in Hamming metric and l_∞ norm. Then we extended the list of solvable problems via algorithmic reductions, and the list of tractable metrics via embeddings. The techniques developed on the way turned out to be useful for problems apparently outside of the scope of high-dimensional computational geometry. We also investigated problems defined over general metrics.

It is interesting to note that the techniques and approaches used to develop these results seem to be very different from the techniques used for low-dimensional geometric problems. We summarize the differences in Figure 8.1. In particular:

Aspect	Low dimension	High dimension
Solutions	approximate and exact	approximate
Mathematical tools	combinatorial geometry	functional analysis
Randomization	Las Vegas	Monte Carlo

Figure 8.1: Low vs high dimensions.

- all algorithms in this thesis are approximate, and it seems plausible that the exact versions of these problems are provably hard. This is quite different from the low-dimensional scenario, where most of these problems have efficient *exact* solutions (with near-linear cost for off-line problems and sublinear cost for on-line problems).

- our results do not seem to use any tools of combinatorial geometry, mainly because the resulting bounds are almost always exponential in the dimension. Instead, we use tools from finite dimensional functional analysis; in particular the notion of embedding turns out to be very useful. This is again a situation quite different from the low-dimensional case, where the solutions for most of the problems we consider here (e.g., nearest neighbor, closest pair, minimum spanning tree) involve combinatorial bounds (e.g., complexity of arrangements, packing bounds etc).

- the most natural algorithms for solving most of the problems here are randomized Monte Carlo. On the other hand, we are aware of only a handful Monte Carlo algorithms for low-dimensional problems[1] Moreover, by now, there exist standard tools which enable to derandomize essentially any randomized low-dimensional algorithm, multiplying the running time only by a factor of n^α for arbitrarily small $\alpha > 0$.

The results in this thesis give rise to a variety of questions and open problems, which we list below.

Core algorithms. The tree-like structure of the reductions gives a natural measure of "importance" of a problem: the higher it is in the hierarchy, the more benefit comes from designing an improved algorithm for it. Therefore, improving the results for c-PLEB in Hamming metric and l_∞ norm seems to be the most important open problem in this thesis. However, it is plausible that there are limits for potential improvements for these problems, coming from the conjectured hardness of the subset query problem. In particular:

[1] Of course, any Las Vegas algorithm can be "degraded" to Monte Carlo by stopping it sufficiently early; here we refer to the intuitive notion of a "natural" algorithm.

- as shown in Chapter 4, solving c-PLEB for l_∞ norm with $c < 3$ implies solving subset query within the same bounds. This suggests that getting efficient algorithms for $c < 3$ might be hard.

- it is easy to see that if we solve c-PLEB for d-dimensional Hamming cube with $c = 1 + \epsilon = 1 + 1/O(d)$, then in fact we are solving the *exact* PLEB problem (which also implies a solution for the subset query problem within the same bounds). Thus, a static c-PLEB algorithm with fast query time and $2^{o(1/\epsilon)}$ space would imply a sub-exponential-space algorithm for SQ. Therefore, the $\Omega(1/\epsilon)$ exponent in the space bound seems necessary in order to achieve fast query time. Note that the algorithm from Section 2.2.1 gives the upper bound of $O(1/\epsilon^2 \cdot \log n)$ for the space bound exponent.

- by a similar argument, it could be difficult to simultaneously achieve small space (i.e., polynomial, with the exponent *independent* on ϵ) and $O(n^{1-\epsilon^a})$ query time, for $0 < a < 1$. Otherwise, we could find $b > 1$ such that the following is true. Firstly, we set $d = \log^b n$ and $\epsilon = 1/O(d)$, so that we solve PLEB exactly. By the assumption the algorithm uses only $n^{O(1)} = 2^{O(\log n)} = 2^{o(d)}$ space. Moreover, the query time is $O(\frac{n}{n^{1/\log^{ab} n}}) = O(\frac{n}{n^{1-ab}})$. If we set $b > 1$ so that $\alpha = 1 - ab > 0$, then we obtain a sublinear query time $O(n^{1-\alpha})$. Thus we achieve subexponential (in d) storage and sublinear (in n) query time.

  Observe, that the algorithm from Section 2.2.2 achieves a approach 1 as ϵ tends to 0.

The above observations seem to decrease the likelihood of obtaining certain types of results for approximate PLEB. Nevertheless, there is a large set of possible bounds which do not seem to suffer from such limitation; therefore, we leave them as open problems. We list such potential results below; we assume $0 < \alpha < 1$

- can we solve $O(1)$-PLEB for l_∞, using $n^{O(1)}$ (i.e., polynomial) space and $d^{O(1)} n^\alpha$ query time ?

- can we solve $O(1)$-PLEB for general l_p norms, using $2^{d^\alpha \log^{O(1)} n}$ (i.e., subexponential) space and $d^{O(1)} n^\alpha$ query time ?

Of course, any evidence of impossibility of obtaining such results would be also of interest.

Reductions. Many reductions given in this thesis are *amortized*. It would be interesting to convert them into worst-case bounds, although it seems that a few of such results could be obtained using variations of standard techniques [Ove83]. As far as the individual problems are concerned, reducing c-FN to c-PLEB in the *dynamic* setting for arbitrary $c > 1$ seems to a major open problem.[2] Note, that by the reduction of Eppstein [Epp95] this would also imply dynamic c-approximation for the diameter.

Embeddings. A large number of open problems comes from the need of extending the algorithms given in this thesis to other metrics. It is likely that such algorithms can be obtained via embeddings. In particular, it would be very interesting to obtain efficient embeddings (either into l_∞ or into Hamming space) of the following metrics.

- *matching metrics*: let A, B be subsets of a metric space $M = (X, D)$ of equal cardinality. The *bottleneck matching* distance between A and B is defined as

  $$\min_{\pi : A \xrightarrow{1:1} B} \max_{a \in B} D(a, f(a))$$

  Another version of this metric (which we called a *matching* metric) is defined by replacing "max" in the above definition by "sum". The latter metric is a special case of the *Earth Mover Distance* [RGT97], which essentially generalizes the matching metric to the multisets. These metrics are very useful for the computer vision applications (cf. [RGT97] and references therein).

- *edit distance*: let A, B be strings over an alphabet Σ. The edit distance between A and B is (in the simplest case) equal to the smallest number of insertions and deletions of characters from Σ needed to transform A into B. There are many variations of this basic metric, obtained by adding other string operations, e.g., substitutions of a single character by another one, cutting/pasting whole blocks of characters etc. For one the variants Muthukrishnan and Sahinalp [MS00] showed that the edit distance metric over n points can be embedded

[2] Very recently it was shown by the author that one can reduce $(1+\epsilon)$-FN to $(1+\epsilon+O(\epsilon^2))$-PLEB (unpublished). Note that this result does not resolve the aforementioned open problem.

into $O(\log n)$-dimensional Hamming cube with distortion $O(\log d)$, and therefore efficient algorithms can be obtained for this metric. However, embedding of the simplest version of edit distance into a tractable metric (with distortion $o(d)$) remains an open problem, as well as improving the $O(\log d)$ distortion to, say, $O(1)$.

Again, an evidence of impossibility of such embeddings would be of large interest as well. It should be noted, that there exist techniques for proving non-embeddability of certain metrics into normed spaces (e.g., see [LLR94]); thus, unlike in the case of algorithmic reductions, there is a real possibility of actually proving the lower bounds, instead of just indicating hardness.

Extroverted applications. We believe that the approach of viewing problems in "high-dimensional way" and then solving them using high-dimensional tools, should be useful for many new problems. One potential areas of applications are coding and lattice theory, where the problem of finding the closest codeword/lattice point are of crucial importance. Although the data set for those problems is defined *implicitly* (e.g., by specifying the lattice basis) rather than explicitly as in our case, it seem likely that some of the algorithmic or embedding tools will be useful for those problems as well.

We already provided several algorithms for the case when the set of points is defined as the set of all fixed length substrings of a given string. For example, we showed how to solve the approximate PLEB over such data set. It would be intersting to extend the latter result to the case when the length of substrings is not fixed. An obvious solution is to prepare a data structure for each substring length, which unfortunately multiplies the storage by linear factor. Is there any solution to this problem which uses less space and/or time ? Note that if we are only interested in approximating the distance between a given pair of substrings of arbitrary length, the solution given in Section 5.2 gives a data structure of almost linear size and polylogarithmic query time.

General metrics. We have shown that several problems defined over general metrics can be solved in sublinear time. On the other hand, some other problems provably cannot be solved within this bound. The general pattern of hardness does not seem to be very clear at this point. Thus, it would be very interesting to develop general results characterizing the class of problems which are solvable in sublinear time.

Another important set of open problems concerns developing sublinear time algorithms for specific problems. The list of such problems includes:

- k-clustering, for $k \geq 3$: is there a $(1+\epsilon)$-approximation algorithm for this problem, with running time $O(T(k,\epsilon) \cdot n \log^{O(1)} n)$ for some function T ? Unfortunately, natural extensions of the algorithm from Section 6.7 seem to involve analysis of a very large number of cases, which makes this approach difficult to follow.

- k-median, for $k \geq 2$: is there a subquadratic-time $(1+\epsilon)$-approximation algorithm for this problem ? (at least for small value of k) Our approach for $k=1$ would clearly require significant extension, since a small random sample can easily miss a small cluster, which can result in large error.

Bibliography

[AAR94] H. Alt, O. Aichholzer, and Gunter Rote. Matching shapes with a reference point. *Proceedings of the Tenth ACM Symposium on Computational Geometry*, pages 85–92, 1994.

[ABB91] H. Alt, B. Behrends, and J. Blomer. Approximate matching of polygonal shapes. *Proceedings of the Seventh ACM Symposium on Computational Geometry*, pages 186–193, 1991.

[AF97] B. Aronov and S. Fortune. Average-case ray shooting and minimum weight triangulations. *Proceedings of the 13th Annu. ACM Sympos. Comput. Geometry*, pages 203–211, 1997.

[Aga91] P.K. Agarwal. *Intersection and Decomposition Algorithms for Planar Arrangements*. Cambridge University Press, New York, NY, 1991.

[Aga97] P.K. Agarwal. Range searching. In Jacob E. Goodman and Joseph O'Rourke, editors, *Handbook of Discrete and Computational Geometry*, chapter 31, pages 575–598. CRC Press LLC, Boca Raton, FL, 1997.

[AHU74] A. Aho, J. Hopcroft, and J. Ullman. *The design and analysis of computer algorithms*. Addison-Wesley, 1974.

[AM92] P.K. Agarwal and J. Matoušek. Ray shooting and parametric search. *Proceedings of the Twenty-Fourth Annual ACM Symposium on Theory of Computing*, pages 517–526, 1992.

[AM93] S. Arya and D. Mount. Approximate nearest neighbor searching. *Proceedings of the Fourth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 271–280, 1993.

[AMN+94] S. Arya, D.M. Mount, N.S. Netanyahu, R. Silverman, and A. Wu. An optimal algorithm for approximate nearest neighbor searching. *Proceedings of the Fifth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 573–582, 1994.

[AMRT96] A. Andersson, P. B. Miltersen, S. Riis, and M. Thorup. Static dictionaries on ac^0 rams: Query time $\theta(\sqrt{\log n / \log\log n})$ is necessary and sufficient. *Proceedings of the 37th Annual IEEE Symposium on Foundations of Computer Science*, pages 441–450, 1996.

[AMS96] N. Alon, Y. Matias, and M. Szegedy. The space complexity of approximating the frequency moments. *Proceedings of the Symposium on Theory of Computing*, pages 20–29, 1996.

[AMS98] P.K. Agarwal, J. Matoušek, and O. Schwarzkopf. Computing many faces in arrangements of lines and segments. *SIAM J. Comput.*, 27(2):491–505, 1998.

[AP00] P. Agarwal and C. M. Procopiuc. Exact and approximation algorithms for clustering. *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms*, 2000.

[AS00] N. Alon and J. Spencer. *Probabilistic method.* 2000.

[Ber93] M. Bern. Approximate closest-point queries in high dimensions. *Information Processing Letters*, 45:95–99, 1993.

[Ber97] B. Berger. The fourth moment method. *SIAM Journal on Computing*, 26, 1997.

[BGMZ97] A. Broder, S. Glassman, M. Manasse, and G. Zweig. Syntactic clustering of the web. *Proceedings of the Sixth International World Wide Web Conference*, pages 391–404, 1997.

[BOR99a] A. Borodin, R. Ostrovsky, and Y. Rabani. Lower bounds for high dimensional nearest neighbor search and related problems. *Proceedings of the Symposium on Theory of Computing*, 1999.

[BOR99b] A. Borodin, R. Ostrovsky, and Y. Rabani. Subquadratic approximation algorithms for clustering problems in high dimensional spaces. *Proceedings of the Symposium on Theory of Computing*, 1999.

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

[Buh00] J. Buhler. Efficient large-scale sequence comparison by locality-sensitive hashing. *Submitted for publication, available from* `jbuhler@cs.washington.edu`, 2000.

[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. *Proceedings of the 16th International Conference on Data Engineering (ICDE)*, 2000.

[CG99] M. Charikar and S. Guha. Improved combinatorial algorithms for the facility location and k-median problems. *Proceedings of the Symposium on Foundations of Computer Science*, 1999.

[CH67] T.M. Cover and P.E. Hart. Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, 13:21–27, 1967.

[Cha97] T.M. Chan. Approximate nearest neighbor queries revisited. *Proceedings of the 13th Annual ACM Symposium on Computational Geometry*, pages 352–358, 1997.

[CL97] E. Cohen and D. Lewis. Approximating matrix multiplication for pattern recognition tasks. *Proceedings of the Second Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 682–691, 1997.

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

[Cla94] K. Clarkson. An algorithm for approximate closest-point queries. *Proceedings of the Tenth Annual ACM Symposium on Computational Geometry*, pages 160–164, 1994.

[CLR90] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms.* MIT Press and McGraw-Hill, 1990.

[CMS76] J. M. Chambers, C. L. Mallows, and B. W. Stuck. A method for simulating stable random variables. *J. Amer. Statist. Assoc.*, 71:340–344, 1976.

[CR96] S. Chaudhuri and R. Radhakrishnan. Deterministic restrictions in circuit complexity. *Proceedings of the Symposium on Theory of Computing*, pages 30–36, 1996.

[DG99] S. Dasgupta and A. Gupta. An elementary proof of the johnson-lindenstrauss lemma. *ICSI technical report TR-99-006, Berkeley, CA*, 1999.

[dlVK98] W. F. de la Vega and C. Kenyon. A randomized approximation scheme for metric max-cut. *Proceedings of the Symposium on Foundations of Computer Science*, pages 468–471, 1998.

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

[EK75] S. Even and O. Kariv. An $o(n^{2.5})$ algorithm for maximum matching in general graphs. *Proceedings of the Symposium on Foundations of Computer Science*, pages 100–112, 1975.

[EK89] O. Egecioglu and B. Kalantari. Approximating the diameter of a set of points in the euclidean space. *Information Processing Letters*, 32:205, 1989.

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

[FCI99] M. Farach-Colton and P. Indyk. Approximate nearest neighbor algorithms for hausdorff metrics via embeddings. *Proceedings of the Symposium on Foundations of Computer Science*, 1999.

[Fel91] W. Feller. *An Introduction to Probability Theory and its Applications.* John Wiley & Sons, NY, 1991.

[FG88] T. Feder and D. Greene. Optimal algorithms for approximate clustering. *Proceedings of the Symposium on Theory of Computing*, 1988.

[FGMM+98] M. Fang, H. Garcia-Molina, R. Motwani, N. Shivakumar, and J. Ullman. Computing iceberg queries efficiently. *Proceedings of the 24th International Conference on Very Large Databases (VLDB)*, 1998.

[FH51] E. Fix and J. Hodges. Discriminatory analysis. nonparametric discrimination: Consistency properties. *Technical Report 4, Project Number 21-49-004, USAF School of Aviation Medicine, Randolph Field, TX*, 1951.

[FKS84] M.L. Fredman, J. Komlós, and E. Szemerédi. Storing a sparse table with O(1) worst case access time. *Journal of the ACM*, 31:538–544, 1984.

[FKSV99] J. Feigenbaum, S. Kannan, M. Strauss, and M. Viswanathan. An approximate l1-difference algorithm for massive data streams. *Proceedings of the Symposium on Foundations of Computer Science*, 1999.

[FLM77] T. Figiel, J. Lindenstrauss, and V. D. Milman. The dimension of almost spherical sections of convex bodies. *Acta Mathematica*, 139:53–94, 1977.

[FM88] P. Frankl and H. Maehara. The johnson-lindenstrauss lemma and the sphericity of some graphs. *Journal of Combinatorial Theory B*, 44:355–362, 1988.

[FP99] D. Finocchiaro and M. Pellegrini. On computing the diameter of a point set in high dimensional euclidean space. *Proceedings of the European Symposium on Algorithms*, 1999.

[Gar95] B. Gartner. A subexponential algorithm for abstract optimization problems. *SIAM Journal of Computing*, 24:1018–1035, 1995.

[GBH] N. Guttman-Beck and R. Hassin. Approximation algorithms for min-sum p clustering. *Discrete Applied Mathematics*.

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

[GGR96] O. Goldreich, S. Goldwasser, and D. Ron. Property testing and its connection to learning and approximation. *Proceedings of the Symposium on Foundations of Computer Science*, pages 339–348, 1996.

[GHS91] L.J. Guibas, J. Hershberger, and J. Snoeyink. Compact interval trees: a data structure for convex hulls. *Internat. J. Comput. Geom. Appl.*, 1(1):1–22, 1991.

[GIM99] A. Gionis, P. Indyk, and R. Motwani. Similarity search in high dimensions via hashing. *Proceedings of the 25th International Conference on Very Large Data Bases (VLDB)*, 1999.

[GIV99] A. Goel, P. Indyk, and K. Varadarajan. Reductions among high-dimensional geometric problems. *Manuscript*, 1999.

[GM99] P. Gibbons and Y. Matias. Synopsis data structures for massive data sets. *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms*, pages S909–S910, 1999.

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

[GT] H. Gabows and R. Tarjan. Faster scaling algorithms for general graph matching problems. *Proceedings of the ACM Symposium on Theory of Computing.*

[Gus93] D. Gusfield. Efficient methods for multiple sequence alignments with guaranteed error bounds. *Bulletin of Mathematical Biology*, 55:141–154, 1993.

[GVY95] N. Garg, V.V. Vazirani, and M. Yannakakis. Siam journal on computing. *Approximate max-flow min-(multi)-cut theorems and their applications*, 25:235–251, 1995.

[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*, 2000.

[HPI00] S. Har-Peled and P. Indyk. When crossings count - approximating the minimum spanning tree. *Proceedings of the Annual ACM Symposium on Computational Geometry*, 2000.

[HPS99] S. Har-Peled and M. Sharir. On-line point location in planar arrangements and its applications. *Manuscript*, 1999.

[HRR98] M. Henzinger, P. Raghavan, and S. Rajagopalan. Computing on data streams. *Technical Note 1998-011, Digital Systems Research Center, Palo Alto, CA*, 1998.

[HS86] D. Hochbaum and D. Shmoys. A unified approach to approximate algorithms for bottleneck problems. *Journal of the ACM*, 33:533–550, 1986.

[IIS99] P. Indyk, G. Iyengar, and N. Shivakumar. Finding pirated video sequences on the internet. *Manuscript*, 1999.

[IKM00] P. Indyk, N. Koudas, and S. Muthukrishnan. Identifying representative trends in massive time series datasets using sketches. *Proceedings of the 26th International Conference on Very Large Databases (VLDB)*, 2000.

[IM98] P. Indyk and R. Motwani. Approximate nearest neighbor: towards removing the curse of dimensionality. *Proceedings of the Symposium on Theory of Computing*, 1998.

[IMRV97] P. Indyk, R. Motwani, P. Raghavan, and S. Vempala. Locality-preserving hashing in multidimensional spaces. *Proceedings of the 29th Annual ACM Symposium on Theory of Computing*, 1997.

[Ind98] P. Indyk. On approximate nearest neighbors in non-euclidean spaces. *Proceedings of the Symposium on Foundations of Computer Science*, 1998.

[Ind99a] P. Indyk. Sublinear time algorithms for metric space problems. *Proceedings of the Symposium on Theory of Computing*, 1999.

[Ind99b] P. Indyk. A sublinear-time approximation scheme for clustering in metric spaces. *Proceedings of the Symposium on Foundations of Computer Science*, 1999.

[Ind00a] P. Indyk. Dimensionality reduction techniques for proximity problems. *Proceedings of the Ninth ACM-SIAM Symposium on Discrete Algorithms*, 2000.

[Ind00b] P. Indyk. Stable distributions, pseudorandom generators, embeddings and data stream computation. *Proceedings of the Symposium on Foundations of Computer Science*, 2000.

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

[JL84] W.B. Johnson and J. Lindenstrauss. Extensions of lipshitz mapping into hilbert space. *Contemporary Mathematics*, 26:189–206, 1984.

[JS82] W.B. Johnson and G. Schechtman. Embedding l_p^m into l_1^n. *Acta Mathematica*, 149:71–85, 1982.

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

[KKLP96] V. Kann, S. Khanna, J. Lagergren, and A. Panconesi. On the hardness of max k-cut and its dual. *Procedings of the 5th Israel Symposium on Theory and Computing Systems (ISTCS)*, pages 61–67, 1996.

[Kle97] J. Kleinberg. Two algorithms for nearest-neighbor search in high dimensions. *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing*, 1997.

[Knu73] D. Knuth. *The Art of Computer Programming: Sorting and Searching.* 1973.

[KOR98] E. Kushilevitz, R. Ostrovsky, and Y. Rabani. Efficient search for approximate nearest neighbor in high dimensional spaces. *Proceedings of the Thirtieth ACM Symposium on Theory of Computing*, pages 614–623, 1998.

[KR99] S. Kolliopoulos and S. Rao. A nearly linear-time approximation scheme for the euclidean k-median problem. *Proceedings of the European Symposium on Algorithms*, 1999.

[KS64] W. H. Kautz and R. C. Singleton. Nonrandom binary superimposed codes. *IEEE Transactions on Information Theory*, 10:363–377, 1964.

[KS97] N. Katayama and S. Satoh. The sr-tree: an index structure for high-dimensional nearest neighbor queries. *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD)*, pages 369–380, 1997.

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

[LLR94] N. Linial, E. London, and Y. Rabinovich. The geometry of graphs and some of its algorithmic applications. *Proceedings of 35th Annual IEEE Symposium on Foundations of Computer Science*, pages 577–591, 1994.

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

[Mat92] J. Matoušek. Reporting points in halfspaces. *Computational Geometry: Theory and Applications*, 2:169–186, 1992.

[Mat99] J. Matousek. On approximate geometric k-clustering. *Manuscript*, 1999.

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

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

[MR95] R. Motwani and P. Raghavan. *Randomized Algorithms.* Cambridge University Press, 1995.

[MS77] F. J. MacWilliams and N. J. A. Sloane. *The Theory of Error-Correcting Codes*. Elsevier/North-Holland, Amsterdam, 1977.

[MS00] S. Muthukrishnan and C. Sahinalp. Approximate sequence nearest neighbors. *Proceedings of the Symposium on Theory of Computing*, 2000.

[MWW91] J. Matoušek, Emo Welzl, and L. Wernisch. Discrepancy and ε-approximations for bounded vc-dimension. *Proceedings 32nd Annual Sympos. Found. Comput. Sci.*, pages 424–430, 1991.

[Nis90] N. Nisan. Pseudorandom generators for space-bounded computation. *Proceedings of the ACM Symposium on Theory of Computing*, pages 204–212, 1990.

[NN93] J. Naor and M. Naor. Small-bias probability spaces: efficient constructions and applications. *SIAM Journal on Computing*, 22:838–856, 1993.

[Ove83] M. H. Overmars. *The Design of Dynamic Data Structures*. Springer Verlag, LNCS 156, 1983.

[Pis89] G. Pisier. *The volume of convex bodies and Banach space geometry*. Cambridge University Press, 1989.

[PW95] P. Pevzner and M.S. Waterman. Multiple filtration and approximate pattern matching. *Algorithmica*, 13:135–154, 1995.

[Rab76] M. O. Rabin. Probabilistic algorithms. *Algorithms and Complexity, J. F. Traub, editor. Academic Press*, pages 21–39, 1976.

[RGT97] Y. Rubner, L. Guibas, and C. Tomasi. The earth mover's distance, multi-dimensional scaling, and color-based image retrieval. *Proceedings of the ARPA Image Understanding Workshop*, pages 661–668, 1997.

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

[Sal89] G. Salton. *Automatic Text Processing.* Addison-Wesley, Reading, MA, 1989.

[SG76] S. Sahni and T. Gonzalez. P-complete approximation problems. *JACM*, 23:555–566, 1976.

[SH75] M. I. Shamos and D. Hoey. Closest point problems. *Proceedings of the Sixteen IEEE Symposium of Foundations of Computer Science*, pages 152–162, 1975.

[Sha75] I. Shamos. Geometric complexity. *Proceedings of the Seventh ACM Symposium on Theory of Computing*, pages 224–233, 1975.

[Shi00] N. Shivakumar. *Detecting digital copyright violations on the Internet (Ph.D. thesis).* Department of Computer Science, Stanford University, 2000.

[SM83] G. Salton and M.J. McGill. *Introduction to Modern Information Retrieval.* McGraw-Hill Book Company, New York, NY, 1983.

[Str00] M. Strauss. Detecting traffic fluctuations using sketches of vectors in l_p norms. *Personal communication*, 2000.

[SWY75] G. Salton, A. Wong, and C. S. Yang. A vector space model for automatic indexing. *Communication of the ACM*, 18:613–620, 1975.

[TI00] M. Thorup and P. Indyk. Approximate 1-median. *Manuscript*, 2000.

[Vai84] P. Vaidya. A fast approximation for minimum spanning trees in k-dimensional space. *Proceedings of the Symposium on Foundations of Computer Science*, pages 403–407, 1984.

[Wel92] Emo Welzl. On spanning trees with low crossing numbers. *Data Structures and Efficient Algorithms, Final Report on the DFG Special Joint Initiative*, LNCS 594:233–249, 1992.

[WLB+98] B. Y. Wu, G. Lancia, V. Bafna, K Chao, R. Ravi, and C. Y. Tang. A polynomial time approximation scheme for minimum routing cost spanning trees. *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms*, pages 21–32, 1998.

[WS82] G. Wyszecki and W.S. Styles. *Color science: concepts and methods, quantitative data and formulae.* John Wiley and Sons, New York, 1982.

[Yao82] A. C. C. Yao. On constructing minimum spanning trees in k-dimensional spaces and related problems. *SIAM Journal of Computing*, 22:721–736, 1982.

[Zol86] V.M. Zolotarev. *One-Dimensional Stable Distributions.* Vol. 65 of Translations of Mathematical Monographs, American Mathematical Society, 1986.