THEORY OF COMPUTING, Volume 8 (2012), pp. 321–350
www.theoryofcomputing.org

SPECIAL ISSUE IN HONOR OF RAJEEV MOTWANI

Approximate Nearest Neighbor: Towards Removing the Curse of Dimensionality

Sariel Har-Peled* Piotr Indyk † Rajeev Motwani ‡

Received: August 20, 2010; published: July 16, 2012.

Abstract: We present two algorithms for the approximate nearest neighbor problem in high-dimensional spaces. For data sets of size n living in $\mathbb{R}^d$, the algorithms require space that is only polynomial in n and d, while achieving query times that are sub-linear in n and polynomial in d. We also show applications to other high-dimensional geometric problems, such as the approximate minimum spanning tree.

The article is based on the material from the authors' STOC'98 and FOCS'01 papers. It unifies, generalizes and simplifies the results from those papers.

ACM Classification: F.2.2

AMS Classification: 68W25

Key words and phrases: approximate nearest neighbor, high dimensions, locality-sensitive hashing

1 Introduction

The nearest neighbor (NN) problem is defined as follows: Given a set P of n points in a metric space defined over a set X with distance function D, preprocess P to efficiently answer queries for finding the point in P closest to a query point $q \in X$. A particularly interesting case is that of the d-dimensional Euclidean space where $X = \mathbb{R}^d$ under some ℓ_s norm. This problem is of major importance in several

*Supported by NSF CAREER award CCR-0132901 and AF award CCF-0915984.

†Supported by a Stanford Graduate Fellowship, NSF Award CCR-9357849 and NSF CAREER award CCF-0133849.

‡Supported by a Sloan Fellowship, and IBM Faculty Partnership Award, an ARO MURI Grant DAAH04-96-1-0007 and NSF Young Investigator Award CCR 9357849.

DOI: 10.4086/toc.2012.v008a014

areas, such as data compression, databases, data mining, information retrieval, image and video databases, machine learning and signal processing. The diverse interest in the problem stems from its wide applicability. Specifically, many large data sets consists of objects that can be represented as a vector of features (i. e., a point in $\mathbb{R}^d$); in such cases, finding an object similar to a given one can be achieved by finding a nearest neighbor in the feature space. The number of features (i. e., the dimensionality) ranges anywhere from tens to millions. For example, one can represent a 1000×1000 image as a vector in a 1,000,000-dimensional space, one dimension per pixel.

The problem and its variants is one of the prototypical questions in computational geometry. Originally posed in the 1960s by Minsky and Papert ([47], p. 222), it has been the subject of substantial research efforts since then. Many efficient solutions have been discovered for the case when the points lie in a space of *constant* dimension. For example, if the points lie in the plane, the nearest neighbor problem can be solved with $O(\log n)$ time per query, using only $O(n)$ storage [59, 45].

Unfortunately, as the dimension grows, these algorithms become less and less efficient. More specifically, their space or time requirements grow *exponentially* in the dimension. In particular, the nearest neighbor problem has a solution with $O(d^{O(1)} \log n)$ query time, but using $n^{O(d)}$ space ([46], building on [22]). This is partly because the *Voronoi decomposition* of P, i. e., the decomposition of $\mathbb{R}^d$ into cells such that all points within each cell have the same nearest neighbor in P, has complexity $n^{\Theta(d)}$. Alternatively, if one insists on linear (or near-linear) storage, the best known running time bound even for *random* point sets is of the form $\min(2^{O(d)}, dn)$, which is essentially linear in n even for moderate d. Worse still, the exponential dependence of space and/or time on the dimension (called "curse of dimensionality") has been observed in practice as well [60].

The lack of success in removing the exponential dependence on the dimension led many researchers to conjecture that no efficient solutions exists for these problems when the dimension is large enough (e. g., see [47]). At the same time, it raised the question whether it is possible to remove the exponential dependence on d, if we allow the answers to be *approximate*. Specifically, in the *c-approximate nearest neighbor* problem, instead of reporting the point p closest to q, the algorithm is allowed to report *any* point within distance c times the distance from q to p, for some approximation factor $c > 1$. The appeal of this approach is that, in many cases, an approximate nearest neighbor is almost as good as the exact one. In particular, if the distance measure accurately captures some notion of quality, then small differences in the distance should not matter much. Moreover, an efficient approximation algorithm can be used to solve the exact nearest neighbor problem, simply by enumerating all approximate nearest neighbors and returning the closest point encountered.

Our results. In this paper, we provide several results for the approximate nearest problem under the ℓ_s norms for $s \in [1,2]$. For concreteness, we start with the case of $s = 1$. Let $c = 1+\varepsilon$, where $\varepsilon > 1/n$. We show:

1. A deterministic data structure for $(1+\gamma)(1+\varepsilon)$-NN, for any constant $\gamma > 0$, with space and preprocessing time bounded by $O(n \log^2 n) \times O(1/\varepsilon)^d$, and query time $O(d \log n)$.

1b. A randomized data structure for $(1+\gamma)(1+\varepsilon)$-NN, for any constant $\gamma > 0$, with space and preprocessing time bounded by $n^{O(\log(1/\varepsilon)/\varepsilon^2)}$, and query time polynomial in d, $\log n$ and $1/\varepsilon$. It is obtained by reducing the dimension d to $O(\log n/\varepsilon^2)$ (using the Johnson-Lindenstrauss lemma [38]), and then utilizing the result above.

2. A randomized data structure for $c(1+\gamma)$-NN, for any constant $\gamma > 0$, with space that is sub-quadratic in n and query time sub-linear in n. Specifically, the data structure uses space $O(dn + n^{1+1/c}\log^2 n \log\log n)$, and has query time of $O(dn^{1/c}\log^2 n \log\log n)$. The algorithm is based on *Locality-Sensitive Hashing*, see below for further details.

The results can be further extended to ℓ_s norm for $s \in [1,2]$. This is done by using an embedding from ℓ_s^d into $\ell_1^{d'}$, for $d' = O(d\log(1/\varepsilon)/\varepsilon^2)$, which preserves all distances by a factor of $1+\varepsilon$ [39]. The reduction increases the query complexity by an additive term of $O(dd')$.

The above results naturally complement each other. The first one (1 and 1b) shows that it is possible to construct approximate data structures that suffer from only a mild form of the "curse of dimensionality." The data structure (1) is the first to support $(1+\varepsilon)$-NN queries in time logarithmic both in n and $1/\varepsilon$, while requiring only space that is near linear in n. If ε is strictly separated from zero, then data structure (1b) removes the "curse" completely. Unfortunately, the resulting data structure is mostly of theoretical interest, since the $\log(1/\varepsilon)/\varepsilon^2$ term in the exponent makes the data structure highly memory-intensive even for relatively large ε. On the other hand, the second algorithm (2) provides a more modest improvement in the running time, but the space penalty is much less severe. As a result, the second algorithm is much more practical when the dimension d is high.

An additional benefit of the first algorithm is that in fact it provides a low-complexity *approximate Voronoi decomposition*. Specifically, we show that the algorithm can be used to construct a decomposition of $\mathbb{R}^d$ into $O(n\log^2 n) \times O(1/\varepsilon)^{d+1}$ simple cells, such that for each cell C one can identify a single point in P that is an $(1+\varepsilon)$-approximate nearest neighbor for any $q \in C$. This is an approximate analog of the Voronoi decomposition, which provides such a structure for the case of the exact nearest neighbor. However, the complexity of our decomposition is much lower than that of the exact one.

Finally, we show how to use our efficient approximate nearest neighbor algorithm to solve other proximity problems. In particular, we show how to approximate the Geometric Minimum Spanning Tree (MST) of a given set on n points by performing near-linear number of calls to an approximate nearest neighbor data structure. By plugging in the second data structure (based on Locality-Sensitive Hashing) we can find a c-approximate MST in time $O(dn^{1+1/c}\log^3 n)$.

Our techniques Our approximate nearest neighbor data structures are obtained in two steps. First, we show how to solve a "decision version" of the approximate nearest neighbor problem, that we term the *approximate* ***near*** *neighbor* problem.[1] Here, we are given a fixed radius r, and the goal is to build a data structure for a given point-set P that for any query q does the following: if there is a point in P within distance r from q, then it reports a point $p' \in P$ within distance cr from q. We show how to construct two such data structures. One utilizes the aforementioned approximate Voronoi decomposition, leading to a fast lookup time and space that is mildly exponential in d. The second one is based on the notion of *locality-sensitive hashing (LSH)*. Its key idea is to hash the points in a way that the probability of collision is much higher for objects which are close to each other than for those which are far apart. Then, one can

[1] In [35, 30], this problem was referred to as *"Point Location in Equal Balls" (PLEB)*. In this paper we are using more recent terminology from [34].

determine near neighbors by hashing the query point and retrieving elements stored in buckets containing that point. We show that such families of hash functions exist for Hamming distance and its variants, and extend them to ℓ_s norms.

In the second step, we show how to reduce the approximate *nearest* neighbor problem to the *near* neighbor problem. A simple way of performing this task is to build several data structures for the latter problem, with radii $r = \Delta, \Delta/c, \Delta/c^2, \ldots$, where Δ is the largest possible distance between the query and the point-set. Unfortunately, in general this approach leads to space complexity that is not bounded by any function of n. We overcome this problem by providing a more efficient reduction, which utilizes only some of the radii r. Our reduction multiplies the space complexity of the near neighbor data structure by a factor of $O(\log^2 n)$, and the query time by $O(\log n)$. Composing the reduction with the aforementioned algorithms for the approximate near neighbor problem provides algorithms for the approximate nearest neighbor problem.

Relation to conference papers Thanks to the amount of time that has passed since the publication of the initial conference papers [35, 30] on which this paper is based, we were able to simplify some of the arguments considerably. As a result, several algorithms in this paper are simpler, and sometimes more general, than the algorithms in the original papers. In particular, the reduction from the near to the nearest neighbor presented here is a simplification of the reduction in [30] (which itself was much simpler and more efficient than the reduction in [35]). It works for general metric spaces, and can be performed using near-linear number of approximate near neighbor queries. In contrast, the preprocessing algorithm in [30] was tailored to ℓ_s spaces, while the algorithm in [35] ran in quadratic time.

We note, however, that a side-effect of our reduction (as well as the reduction of [30]) is that in the approximate near neighbor instances the ratio of the diameter of the point-sets to search radius r is no longer small (polylogarithmic). The latter property is useful, e. g., when designing efficient LSH functions for ℓ_1^d space. We can ensure a somewhat weaker property directly by exploiting the "randomized bucketing" approach of Bern [11] (Lemma 3.1). Luckily, the property suffices for our purpose.

Another result that was a subject to a substantial generalization is the algorithm for computing an approximate MST. The algorithm outlined in the manuscript [36] relied on a data structure for maintaining approximately close pairs of points under updates, which was used to simulate Kruskal's MST algorithm. Instead, in this paper we present a *general* reduction from the approximate MST problem to dynamic approximate near neighbor data structure, that works in *arbitrary* metrics. The reduction is still based on Kruskal's algorithm. However, it has a particularly simple form, that was inspired by the algorithm of [12] (see the comments in the next subsection).

1.1 Related work

Prior Work There has been a substantial amount of work on approximate nearest neighbor problem in the computational geometry literature. However, all work prior to this paper yielded algorithms that involved factors exponential in d in either the space or in the query time bound. One of the earliest works on this topic is [9] (improved upon in [22] and [19]), who gave an algorithm with query time $1/\varepsilon^{O(d)} \cdot \log n$ and space $1/\varepsilon^{O(d)} \cdot n$. Another line of research [10] resulted in an algorithm with linear space $O(dn)$ but query time $(d/\varepsilon)^{O(d)} \cdot \log n$. Other authors [11, 19] considered algorithms with approximation factor polynomial in d, and provided algorithms that avoid exponential factors in the space and query time bounds. Finally, two approximation algorithms were given in [42]: one with $O(dn\log d)$ space and $O(n)$ query time and one with $n^{O(d)}$ space and $(d+\log n)^{O(1)}$ query time (for fixed $\varepsilon > 0$). The last algorithm, although still suffering from the curse of dimensionality, has significantly influenced the developments in this paper.

The locality-sensitive hashing approach introduced in this paper has several ancestors in the literature, which investigated multi-index hashing-based algorithms for retrieving similar pairs of vectors with respect to the Hamming distance. Although the analytic framework adopted in those papers made the results generally incomparable to ours, some of the insights are shared. A few of the papers considered a closely related problem of finding all "close" pairs of points in a data set. For simplicity, we translate them into the near neighbor framework, since they can be solved by performing essentially n separate near neighbor queries.

Typically, the hash functions projected the vectors on some subset of the coordinates $\{1,\dots,d\}$. In some papers [54, 29] the authors considered the probabilistic model where the data points are chosen uniformly at random, and the query point is a "random" point "close" to one of the points in the data set. A different approach [40] is to assume that the data set is arbitrary, but almost all points are far from the query point. Finally, the paper [17] proposed an algorithm which did not make any assumption on the input, and provided a method for determining the parameters (denoted here by k and L) to achieve desired level of sensitivity and accuracy.

On a related front, the authors of [13, 14] considered similarity search between sets (say A,B) using the Jaccard coefficient $s(A,B) = |A\cap B|/|A\cup B|$. They proposed a family of hash functions $h(A)$ such that $\Pr[h(A) = h(B)] = s(A,B)$, which can be plugged into our framework. Although their main motivation was to construct short similarity-preserving "sketches" of sets, they also discuss methods for performing similarity search using these functions.

Concurrent developments Parallel to our conference paper [35], the paper [43] presented an algorithm with bounds similar to our result (1b). Specifically, it provides a data structure that, in case of the ℓ_1^d norm, achieves $O(d(\log n+1/\varepsilon)^{O(1)})$ query time using space $(dn)^{O(1/\varepsilon^2)}$. For the ℓ_2 norm, the query time becomes $O(d^2(\log n+1/\varepsilon)^{O(1)})$. The probabilistic guarantee provided by their data structure is somewhat stronger: if the construction procedure is correct (which happens with a controlled probability) then the data structure returns a correct approximate nearest neighbor to all queries q.[2] Although the technical development is somewhat different, the general approach is similar, in that a form of a randomized dimensionality reduction is used to reduce the dimension to $O(\log n/\varepsilon^2)$.

In another parallel development, the paper [12] presented a $(1+\varepsilon)$-approximation algorithm for MST (for $\varepsilon < 1$), with running time $O(dn^{1-a\varepsilon^2})$ for some absolute constant $a > 0$.

Further developments Since the conference versions of this paper have appeared in [35, 36, 30], there have been many further developments on approximate nearest neighbor search. In this section, we briefly discuss some of those results. For a more in-depth treatment of the material see [34].

[2]Intuitively, this guarantee is obtained by setting a probability of failure to be inversely exponential in d, and applying a union bound over the appropriately discretized set of all queries.

The reduction from the approximate nearest to the near neighbor problem, and the resulting approximate Voronoi decompositions were further improved in [7] by using a well-separated pairs decomposition [18] to generate the cells needed to construct the approximate Voronoi diagram (note, that unlike our construction, the number of cells is linear in n but has exponential dependency on the dimension). This construction uses the same framework as suggested in [30]. This result was extended to improve the tradeoff between the dependency on ε on the query time and space used. See [8] for further details and related work. This construction results in an approximate Voronoi diagram of complexity $n/\varepsilon^{O(d)}$.

An alternative construction was suggested in [57]. Building on the reduction from nearest-neighbor search to near neighbor suggested in [30], they reduced the space requirements by a logarithmic factor. By slightly changing the near-neighbor problem, they reduce the space bound to linear, which yields an approximate Voronoi diagram of complexity $n/\varepsilon^{O(d)}$.

The exponent in the space bound achieved by the algorithm (1b) is likely to be close to the optimal. Specifically [6] showed that any $(1+\varepsilon)$-approximate data structure in Hamming space that makes only a constant number of memory accesses needs $n^{\Omega(1/\varepsilon^2)}$ storage (our data structure makes only one memory access). There has been plenty of work on lower bounds for the exact and approximate nearest neighbor problem, see [34] for an overview.

There has been substantial progress in designing LSH functions for various measures and understanding their limitations, see [4] for an overview. In particular, it is known that for the ℓ_1 norm, the $1/c$ bound for the running time exponent given in this paper is tight ([50], building on [48]). Lower bounds in more general models of computations were provided in [53]. In contrast, for the ℓ_2 norm, one can further reduce the exponent to $1/c^2$ ([3], building on [26]). In addition, a different algorithm exploiting LSH functions was proposed in [52]; this data structure achieves near-linear storage, at the price of increasing the exponent by a constant factor.

From a more general perspective, the existence of fast approximate nearest neighbor algorithms for high-dimensional ℓ_1 spaces enabled solving this problem for other metrics, by embedding them into ℓ_1 with low distortion. This approach has been useful for variants of edit distance [49, 25, 24, 20, 51], earth-mover distance [21, 37, 28, 5] and other metrics. Lower bounds for such embeddings have been investigated as well [41].

Finally, the LSH algorithm and its variants have become popular practical algorithms for similarity search in high dimensions. They have been successfully applied to computational problems in a variety of areas, including web clustering [33], computational biology [15, 16], computer vision (see selected articles in [58]), computational drug design [27] and computational linguistics [56].

1.2 Notation

We use (X,D) to denote a metric space defined over X with the distance function D. Typically, we will use $X=\mathbb{R}^d$ for some dimension $d>0$ and $D(p,q)=\|p-q\|_s$ for some $s\geq 1$. In those cases, we will assume that $d\leq n$, where $n=|P|$. For convenience, we assume that n is even.

For any point $p\in X$, and $r>0$, we use $B(p,r)$ to denote the set $\{q\in X : D(p,q)\leq r\}$. For a parameter $r>0$, let $\mathrm{UB}_P(r)=\bigcup_{p\in P}B(p,r)$ denote the union of equal balls of radius r centered at the points of P. Moreover, let $\mathrm{CC}_P(r)$ be a partitioning of P induced by the connected components of $\mathrm{UB}_P(r)$. That is, two points $p,q\in P$ belong to the same set in the partitioning if there is a path contained in $\mathrm{UB}_P(r)$

connecting p to q; that is, there is a sequence $p = p_1, \dots, p_k = q \in P$, such that $D(p_i, p_{i+1}) \leq 2r$, for $i = 1, \dots, k-1$.

Define $D_P(q) = \min_{p \in P} D(q, p)$. For a parameter $c \geq 1$, $p' \in P$ is a *c-approximate nearest neighbor* of q if $D(q, p') \leq cD_P(q)$.

Definition 1.1. The *c-approximate nearest neighbor problem* (or c-NN) with failure probability f is to construct a data structure over a set of points P in metric space (X, D) supporting the following query: given any fixed query point $q \in X$, report a c-approximate nearest neighbor of q in P with probability $1 - f$. We use NearestNbr(P, c, f) to denote a data structure solving this problem. We skip the argument f if it is equal to some absolute constant from $(0, 1)$.

Definition 1.2. The *(c, r)-approximate **near** neighbor problem* (or (c, r)-NN) with failure probability f is to construct a data structure over a set of points P in metric space (X, D) supporting the following query: given any fixed query point $q \in X$, if $D_P(q) \leq r$, then report some $p' \in P \cap B(q, cr)$, with probability $1 - f$. We use NearNbr(P, c, r, f) to denote a data structure solving this problem. We skip the argument f in this or other definitions involving it if it is equal to some absolute constant from $(0, 1)$.

Note that, according to the above definition, if $q \notin \mathrm{UB}_P(r)$ then the algorithm is allowed to report any point in P, or no point at all (i. e., p' is null). For convenience, if the reported point p' is further than cr from q (a condition that the algorithm can check), then we set p' to null.

Let $n = |P|$. If the data structure NearNbr(P, c, r) is defined by the context, we will use $T(n, c, r, f)$ and $Q(n, c, r, f)$ to denote the construction and query time of the data structure, respectively. We will also use $S(n, c, r, f)$ to denote its space complexity. Additionally, if the data structure supports updates (insertions or deletions of points to or from P), we denote the update time by $U(n, c, r, f)$. The time to perform a deletion alone is denoted by $D(n, c, r, f)$. We will skip the argument r if the time and space functions do not depend on it, and the argument f if it is assumed to be equal to some small absolute constant (e. g., $1/3$).

For the algorithms discussed in this paper, we assume that $T(n, c, r, f) = \Omega(nd)$, $S(n, c, r, f) = \Omega(nd)$ and $Q(n, c, r, f) = \Omega(d)$, i. e., the time and space bounds are at least linear in the input size. Moreover, by standard replication arguments, we have $T(n, c, f) = O(\log 1/f) T(n, c)$; a similar relationship holds for the other complexity measures.

Insertions and deletions in randomized data structures In this paper we present several randomized data structures over sets of points P that support approximate nearest neighbor queries and updates. There exist several possible types of guarantees that one can require from such a data structure. In this paper we consider two models: "oblivious" and "adaptive."

Both models require the notion of a well-formed sequence. Formally, consider any sequence S of m operations on the data structures, where the ith operation is of the form (op_i, p_i), where $\mathrm{op}_i \in \{\mathit{Query}, \mathit{Delete}, \mathit{Insert}\}$. Given two sets of points P and Q, the sequence is called *well-formed* with respect to P and Q if it satisfies the following conditions:

(A) Only the points in P can be deleted or inserted.

(B) Only the points in Q can be queried.

(C) A point can be deleted only after it has been inserted first.

If Q (or P, respectively) is the set of all points in the metric space, we skip Q (or P, respectively) in the above definition.

We start by defining the oblivious model where we require that for any well-formed sequence, the data structure should be correct with some probability. This is the default model used in this paper. The formal definition is as follows.

Definition 1.3. A randomized fully dynamic (r,c)-NN data structure $\mathtt{Z}$ with failure probability f is called *oblivious* if for any sequence S that is well-formed:

$$\Pr[\text{the last operation in the sequence } S \text{ is executed correctly by } \mathtt{Z}] \geq 1 - f\,.$$

Note that one can bound the probability of correctness for *all* operations in the sequence via the union bound.

In the adaptive model, the requirements are stronger than in the oblivious case. Specifically, once a data structure is constructed correctly (which happens with probability $1-f$), then it should correctly execute *any* well-formed sequence of update and query operations, as long as the points come from a pre-defined set. This in particular implies that the query and update operations can depend on the results of prior operations.

Definition 1.4. Consider any two sets of points P and Q. A randomized fully dynamic (c,r)-NN data structure $\mathtt{Z}$ with failure probability f is called *adaptive with respect to* P,Q if

$$\Pr[\mathtt{Z} \text{ is correct for any sequence } S \text{ that is well-formed with respect to } P,Q] \geq 1 - f\,.$$

Miscellaneous For any $P \subset X$, let $\mathcal{P}$ and $\mathcal{P}'$ be two partitions of P. We say that $\mathcal{P}$ *refines* $\mathcal{P}'$ (denoted by $\mathcal{P} \sqsubseteq \mathcal{P}'$) if for any $S \in \mathcal{P}$ we have $S' \in \mathcal{P}'$ such that $S \subset S'$.

We use $D_H(p,q)$ to denote the Hamming distance between vectors p and q, i. e., the number of coordinates i such that $p_i \neq q_i$.

2 Reduction from the approximate nearest to near neighbor

2.1 Outline

In this section, we outline the reduction from the approximate *nearest* neighbor problem to a sequence of approximate *near* neighbor problems.

We start from a description of an "ideal" reduction, which assumes a solution to the *exact* near neighbor problem (i. e., for $c = 1$), and also assumes some constraints on the point set. The actual reduction overcomes these requirements at the price of making it somewhat more complicated.

For simplicity, assume n is even. Let $\gamma \in (1/n, 1/2)$ be a parameter to be determined later (the lower bound of $1/n$ ensures that $\log(n/\gamma) = O(\log n)$, which simplifies some expressions). Let r_{med} be equal to the smallest value of r such that $\mathrm{UB}_P(r)$ has a component of size at least $n/2+1$, and let $\mathrm{UB}_{\mathrm{med}} = \mathrm{UB}_P(r_{\mathrm{med}})$. For the purpose of exposition, we assume that the largest component of $\mathrm{UB}_{\mathrm{med}}$ has size *exactly* $n/2+1$.

We first construct a NearNbr data structure $\mathrm{Z}_{\mathrm{med},2} = \mathrm{NearNbr}(P, c, r_{\mathrm{med}}/2)$. Given a query point q, if q is within distance $r_{\mathrm{med}}/2$ to some point $p \in P$ (which can be decided by one near neighbor query in $\mathrm{Z}_{\mathrm{med},2}$), then we continue the search for the nearest neighbor recursively in the connected component of $\mathrm{UB}_{\mathrm{med}}$ that contains p. Note that such connected component is unique. Moreover, observe that the search continues recursively into a subset of P of cardinality at most $n/2+1$.

Alternatively, if $D_P(q) \geq r_{\mathrm{top}}$ for $r_{\mathrm{top}} \gg r_{\mathrm{med}}$ (which can be decided by a single near neighbor query in $\mathrm{NearNbr}(P, c, r_{\mathrm{top}})$), then q is "far away" from the points of P, and we can continue the search on a decimated subset of P. Namely, from each connected component of $\mathrm{UB}_{\mathrm{med}}$, we extract one point of P that lies inside it. This results in a set $P' \subset P$ that contains at most $n/2$ points. We continue the recursive search into P'. Although continuing the search into P' introduces cumulative error into the search results, we show that the overall error introduced is smaller than $1+O(\gamma)$ if we set $r_{\mathrm{top}} = \Theta(n r_{\mathrm{med}} \log(n)/\gamma)$.

The only case that remains unresolved, is when $r_{\mathrm{med}} \leq D_P(q) \leq r_{\mathrm{top}}$. Observe that $r_{\mathrm{top}}/r_{\mathrm{med}} = O((n \log n)/\gamma)$. Namely, we can cover the interval $[r_{\mathrm{med}}, r_{\mathrm{top}}]$ by

$$M = \log_{1+\gamma} \frac{r_{\mathrm{top}}}{r_{\mathrm{med}}} = O((1/\gamma)\log(n/\gamma))$$

NearNbrs $\mathrm{Z}_1, \ldots, \mathrm{Z}_M$, where

$$\mathrm{Z}_i = \mathrm{NearNbr}(P, c, r_{\mathrm{med}}(1+\gamma)^i),$$

for $i = 1, \ldots, M$.

By performing a binary search on $\mathrm{Z}_1, \ldots, \mathrm{Z}_M$ we can find, using $O(\log M)$ near neighbor queries, the index i, such that $q \notin \mathrm{UB}(P, r_{\mathrm{med}}(1+\gamma)^i)$ and $q \in \mathrm{UB}_P(r_{\mathrm{med}}(1+\gamma)^{i+1})$. Namely, we have found an $(1+\gamma)$-approximate nearest neighbor of q in P.

Overall, given a query point, either we found its $(1+\gamma)$-approximate nearest neighbor using $O(\log M) = O(\log(n/\gamma))$ near neighbor queries, or alternatively, we performed two near neighbor queries and continued the search recursively into a set having at most $n/2+1$ points of P. Thus, one can find an $(1+O(\gamma))$-approximate nearest neighbor by performing $O(\log(n/\gamma))$ NearNbr queries.

This concludes the description of the ideal algorithm. Unfortunately, it suffers from several problems: (i) it assumes a reduction to the exact near neighbor problem, (ii) it assumes that a component of size $n/2+1$ always exists, and (iii) it requires computing the value of r_{med}, which is expensive. Instead, we will only compute an approximation r^*_{med} to r_{med}, such that $r_{\mathrm{med}} \leq r^*_{\mathrm{med}} = O(n r_{\mathrm{med}})$. This will require defining r^*_{bot} that will play the role of r_{med} in the above argument, and readjusting the value of r_{top}.

2.2 Subroutines

Approximating r_{med} Recall that r_{med} is equal to the smallest value of r such that $\mathrm{UB}_P(r)$ has a component of size at least $n/2+1$, and $\mathrm{UB}_{\mathrm{med}} = \mathrm{UB}_P(r_{\mathrm{med}})$. We start from describing an algorithm that approximates the value of r_{med}.

Lemma 2.1. *There exists a randomized algorithm that, given a set P of n points and a distance function D, returns an estimate r^*_{med} such that $r_{med} \leq r^*_{med} \leq (n-1)r_{med}$ with probability at least $1/2$. The algorithm runs in time $O(n)$.*

Proof. The algorithm first selects a point p uniformly at random from P. Then r^*_{med} is defined to be the median of the set $D(p, p')$ over $p' \in P$. Note that the median of $n-1$ distances is uniquely defined.

To prove the lemma, define $C \subset P$ to be the set of points inducing the largest connected component in $\text{UB}_P(r_{\text{med}})$. By definition of r_{med}, we have $|C|/|P| > 1/2$. Therefore, the point p belongs to C with given probability. From now on we condition the analysis on this event.

First, we show that $r_{\text{med}} \le r^*_{\text{med}}$. Indeed, the point p, together with the $n/2$ points closest to p, induces a connected component in $\text{UB}_P(r^*_{\text{med}})$ of size $n/2+1$. By definition, r_{med} is the smallest radius that induce such a component in $\text{UB}_P(r^*_{\text{med}})$.

On the other hand, all points in C are within the distance of $(n-1)r_{\text{med}}$ from p. Since $|C| > n/2$, it follows that the median of the distances (i. e., r^*_{med}) is at most $(n-1)r_{\text{med}}$. □

Let $c > 1$ be the approximation factor of the near-neighbor data structure that we reduce to. For $\lambda = O(\log n/\gamma)$ where $\gamma > 0$ is a parameter to be determined, we define $r^*_{\text{bot}} = r^*_{\text{med}}/(nc)$ and $r^*_{\text{top}} = r^*_{\text{med}} n\lambda c$. Note that $r^*_{\text{top}}/r^*_{\text{bot}} = \Theta(n^2 \log n)$. Assuming that r^*_{med} is computed correctly, i. e., that $r_{\text{med}} \le r^*_{\text{med}} \le (n-1)r_{\text{med}}$, then we also have $r^*_{\text{bot}}c < r_{\text{med}}$.

Claim 2.2. *Suppose that r^*_{med} is computed correctly, i. e., that $r_{med} \le r^*_{med} \le (n-1)r_{med}$. Then*

- *each connected component of $UB_P(r^*_{bot}c)$ is induced by at most $n/2$ points, and*
- *there exists at least one connected component $C \in CC_P(r^*_{med})$ that has size at least $n/2+1$, and thus there are at most $n/2$ connected components in $CC_P(r^*_{med})$.*

Approximating $\mathbf{CC}_P(r)$ We now describe an algorithm for approximating the connected components $\text{CC}_P(r)$. Specifically, for given parameters r and c, we will show how to compute a partitioning $\mathcal{P}$ of P such that $\mathcal{P}$ is a refinement of $\text{CC}_P(cr)$ and $\text{CC}_P(r)$ is a refinement of $\mathcal{P}$. The algorithm uses a $\text{NearNbr}(P, c, r, f)$ data structure that is *adaptive* with respect to the query set P with failure probability at most f, where f is some constant.

The algorithm is presented as Algorithm 1. The basic idea of the algorithm is simple: we compute connected components of a graph with edges of the form (q, p'), where $q \in P$ and p' is a point output by an approximate NearNbr query on q. This is done by implementing a standard graph search. However, some care is needed to ensure that the running time of the algorithm is low, even though the graph itself can have $\Omega(n^2)$ edges. This is achieved by deleting from NearNbr the points that have been reached during the search. This ensures that each new edge found by querying NearNbr leads to a vertex that has not been reached yet.

Claim 2.3. *The algorithm* APPROXIMATECC *computes a partitioning $\mathcal{P}$ that refines $CC_P(cr)$ and is refined by $CC_P(r)$, i. e., $CC_P(r) \sqsubseteq \mathcal{P} \sqsubseteq CC_P(cr)$. Its running time is*

$$O\big(T(n,c,f) + nD(n,c,f) + nQ(n,c,f)\big)$$

and the probability of failure is at most f.

Interval near neighbor Our algorithm will make use of a collection of NearNbr data structures which solve the approximate nearest neighbor assuming the distance from the query to the data set belongs to a given range. We will refer to this collection as *Interval* NearNbr, or INearNbr.

```
procedure APPROXIMATECC(P, c, r)
    Construct a NearNbr(P, c, r, f) data structure for P.
    𝒫 = ∅
    E′ = ∅
    while P ≠ ∅ do
        Select any p ∈ P.
        Delete p from P and from the associated NearNbr structure.
        S = {p}
        C = ∅                                   ▷ C is the next connected component to be computed
        repeat
            Select any q from S.                ▷ We will enumerate all edges going out of q
            Add q to C.
            Delete q from S.
            repeat
                Let p′ be the answer of the NearNbr structure on q.
                if p′ is not null then
                    Add {q, p′} to E′.
                    Delete p′ from P and the associated NearNbr structure.
                    Add p′ to S.
                end if
            until p′ is null
        until S = ∅                             ▷ Connected component C has been completed
        Add C to the partition 𝒫.
    end while
end procedure
```

Algorithm 1: Finding an approximation to $\mathrm{CC}_P(r)$.

Lemma 2.4. *For $1/2 > \gamma > 0$ and $c \geq 1$, given range $[a,b]$ and a point-set P, one can construct a collection of $M = O((\log b/a)/\gamma)$* NearNbr *data structures with approximation parameter c and failure probability $f = 1/(3\log M)$, so that given a query point q, one can decide that either:*

(i) $D_P(q) \leq a$,

(ii) $D_P(q) \geq b$, *or*

(iii) find a point $p' \in P$, so that $D(q,p') \leq c(1+\gamma)D_P(q)$.

The answer returned is correct with probability at least $2/3$. If either case (i) or case (ii) occurs then only two NearNbr *queries are carried out, while if case (iii) occurs then $O(\log(\log(b/a)/\gamma))$* NearNbr *are performed.*

Proof. Let $r_i = a(1+\gamma)^{i-1}/c$, for $i = 1,\ldots,M$, where

$$M = \lceil \log_{(1+\gamma)}(cb/a) \rceil = O\left(\frac{\log(cb/a)}{\gamma}\right)$$

(since $\gamma < 1/2$). Let $\mathrm{Z}_i = \mathrm{NearNbr}(P,c,r_i,f)$ for $i = 1,\ldots,M$.

Clearly, if a query to Z_1 does not return null, we conclude that case (i) has occurred. Similarly, if a query to Z_M returns null, we conclude that case (ii) must have happened. Note that in both cases we only perform one NearNbr query.

Otherwise, it must be the case that $a/c \le D_P(q) \le cb$. By performing a binary search on $\mathsf{Z}_1, \ldots, \mathsf{Z}_M$ we can find an index i such that, on query q, Z_i reports null and Z_{i+1} reports some point p'. Clearly, we have

$$r_i \le D_P(q) \le D(q,p') \le cr_{i+1} \le c(1+\gamma)r_i \le c(1+\gamma)D_P(q)\,.$$

Thus, p' is a $c(1+\gamma)$-approximate nearest neighbor of q in P. □

2.3 The reduction

Data structure construction Let $f = 1/(E\log n)$ for some constant $E > 1$. We will use NearNbr data structures with probability of failure f. The preprocessing algorithm is described as Algorithm 2. It can be viewed as building a search tree defined by the recursive calls, with each tree node containing a subset of points and a collection of NearNbr data structures over that subset.

```
procedure CONSTRUCT(P, c, γ)
    if |P| = 1 then
        Store P.
        return
    end if
    repeat
        Compute r*_med, r*_bot and r*_top (Lemma 2.1 and Claim 2.2).
                                  ▷ Note that the last step succeeds with probability 1/2.
        Compute a partitioning 𝒫 = {P_1, ..., P_k} of P s. t. CC_P(r*_bot) ⊑ 𝒫 ⊑ CC_P(c r*_bot) (Claim 2.3).
        Compute another partitioning 𝒫' = {P'_1, ..., P'_k'} of P s. t. CC_P(r*_med) ⊑ 𝒫' ⊑ CC_P(c r*_med).
    until |𝒫'| ≤ n/2 and |P_i| ≤ n/2, for all i = 1, ..., k.
    for i = 1, ..., k' do
        Select a representative p'_i from P'_i.
        Construct INearNbr over P with range [r*_bot/2, r*_top] and parameters γ and c (Lemma 2.4).
        Continue recursively on P_1, ..., P_k and P' = {p'_1, ..., p'_k'}.
    end for
end procedure
```

Algorithm 2: Constructing the approximate nearest neighbor data structure.

We now analyze the complexity of the construction algorithm.

Lemma 2.5. *The number of points stored in all* NearNbr *structures in the constructed tree is $O(Mn\log n)$, where $M = O(\log(n)/\gamma)$ is the number of times each point is replicated in* INearNbr.

Proof. Let $B(n)$ be the maximum number of points stored in the INearNbr structures. We analyze $B(n)$ and multiply the result by $O(M)$.

First, observe that when the procedure terminates, we have $k' \leq n/2$ and $k' \leq k$. The latter inequality holds since $cr^*_{\mathrm{bot}} \leq r^*_{\mathrm{med}}$ and therefore $\mathcal{P} \sqsubseteq \mathcal{P}'$. Without the loss of generality assume that $n \geq 3$. We have the following recurrence:

$$B(n) = \max_{k',k,n_1,\ldots,n_k} \sum_{i=1}^{k} B(n_i) + B(k') + n \tag{2.1}$$

subject to $k' \leq n/2$, $k' \leq k$, $1 \leq n_i \leq n/2$ and $\sum_{i=1}^{k} n_i = n$. We show this solves to $B(n) \leq Cn\log n + 1$ for some $C > 1$.

The proof is by induction. First, we apply the inductive assumption and substitute for $B(n_i)$ and $B(k')$ in equation (2.1). Since $k' \leq k$ we have

$$B(n) \leq \max_{k,n_1,\ldots,n_k} \sum_{i=1}^{k} [Cn_i \log n_i + 1] + Ck\log k + 1 + n \tag{2.2}$$

subject to the same constraints as above.

We relax the assumption that n_i's are integers. Using the convexity of the negated entropy function $H(p) = p\log p + (1-p)\log(1-p)$, applied on pairs of n_i's, we observe that for each k, the maximum is achieved when at most two values n_i (say, n_1 and n_2) are set to a value greater than 1. Since $1 \cdot \log(1) = 0$, the right hand side of equation (2.2) simplifies to

$$\max_{k,n_1,n_2} [Cn_1 \log n_1 + Cn_2 \log n_2 + k + Ck\log k] + n + 1 \leq C \max_{k,n_1,n_2} [n_1 \log n_1 + n_2 \log n_2 + k\log k] + 2n\,,$$

where $n_1 + n_2 + (k-2) = n$, $k \leq n/2$ and $n_i \leq n/2$.

By using convexity again, we upper bound the right hand side by

$$2C(n/2)\log(n/2) + C2\log(2) + 2n = 2C + Cn\log n - Cn + 2n\,,$$

which is at most $Cn\log n$ if $n \geq 3$ and C is large enough. □

Corollary 2.6. *For any $\gamma > 0$, if the space/time complexity functions $S(n,c,f)$ and $T(n,c,f)$ are convex, then the data structure constructed in Algorithm 2 uses $O(S(n,c,f)/\gamma \cdot \log^2 n)$ space, and is constructed in expected time*

$$O\left(\frac{T(n,c,f)}{\gamma \cdot \log^2 n} + n\log n\big[Q(n,c,f) + D(n,c,f)\big]\right).$$

Search The search procedure follows the divide-and-conquer approach used in the data structure tree construction. Let q be the query point. The procedure is described in Algorithm 3.

Observe that the height of the data structure tree is at most $h = \log n + O(1)$. Therefore, we have the following claim.

Claim 2.7. *The search procedure performs at most $O(\log n)$* NearNbr *queries and distance computations.*

Proof. The search procedure encounters case (iii) of an INearNbr query at most once; this takes time $O(\log M)$. In other cases, it performs only $O(1)$ NearNbr queries per tree level. The time complexity follows. □

procedure SEARCH(P, q)
  if $|P| = 1$ **then**
    Report the point in P
  else
    Perform an INearNbr query with argument q (Lemma 2.4).
    if the outcome is case (i) **then**
      Let P_i be the set such that $D_{P_i}(q) \le r^*_{\text{bot}}/2$.
      Perform Search(P_i, q).
    else
      if the outcome is case (ii) **then**
        Perform Search(P', q).
      else
        if the outcome is case (iii) **then**
          Report the point p' found by INearNbr.
        end if
      end if
    end if
  end if
end procedure

Algorithm 3: Searching for approximate nearest neighbor.

Recall that the probability of failure of the NearNbr data structures is equal to $f = 1/(E \log n)$. We have:

Lemma 2.8. *The search procedure reports a* $(1+\gamma)^2 c$*-approximate nearest neighbor of* q *in* P*, with probability of failure at most* $O(1/E)$.

Proof. Assume that the recursive call in the search procedure (in cases (i) or (ii)) reports an a-approximate nearest neighbor. Depending on the outcome of the INearNbr query, we have the following three cases:

Case (i) The search procedure continues on the subset P_i that is guaranteed to contain the nearest neighbor. Thus, the point reported by the recursive call is a a-approximate nearest neighbor of q in P.

Case (ii) The search procedure continues on the set P' of representatives, with the property that for any point $p \in P$, there is a point $p' \in P'$ such that $D(p, p') \le cnr^*_{\text{med}}$. Moreover, we know that

$$D_P(q) \ge r^*_{\text{top}} = r^*_{\text{med}} n \lambda c \ge \lambda D(p, p') .$$

It follows that if the recursive procedure finds an a-approximate nearest neighbor of q in P', then this point is also a $(1 + 1/\lambda)a$-approximate nearest neighbor of q in P.

Case (iii) The search procedure determines a $(1+\gamma)c$ approximate nearest neighbor directly by querying INearNbr.

It follows that the search procedure reports an $(1+1/\lambda)^h(1+\gamma)c$-approximate nearest neighbor, where $h = \log n + O(1)$ is the height of the data structure tree. We set $\lambda = O(\log(n)/\gamma)$ such that $(1+1/\lambda)^h = 1+\gamma$.

The above analysis required that each of the $O(\log n)$ NearNbr data structures encountered in the process was correct. The probability that this does not happen is at most $O(1/E)$. □

2.4 The result

We now state our main result of this section. The result is stated in two forms: without preprocessing, and with preprocessing.

Theorem 2.9. *Let P be a given set of n points in a metric space, and let $c = 1+\varepsilon > 1$, $f \in (0,1)$, and $\gamma \in (1/n, 1)$ be prescribed parameters.*

Assume that we are given a data structure for the (c,r)-approximate near neighbor that uses space $S(n,c,f)$, has query time $Q(n,c,f)$, and has failure probability f. Then there exists a data structure for answering $c(1+O(\gamma))$-NN queries in time $O(\log n)Q(n,c,f)$ with failure probability $O(f \log n)$. The resulting data structure uses $O(S(n,c,f)/\gamma \cdot \log^2 n)$ space.

Theorem 2.10. *Let P be a given set of n points in a metric space, and let $c = 1+\varepsilon > 1$, $f \in (0,1)$, and $\gamma \in (1/n, 1)$ be prescribed parameters.*

Assume that we are given a data structure for the (c,r)-approximate near neighbor over P that can be constructed in $T(n,c,f)$ time, uses $S(n,c,f)$ space, has query time $Q(n,c,f)$, has deletion time $D(n,c,f)$, and has failure probability f. Moreover, assume that the data structure is adaptive with respect to any set of size $n^{O(1)}$.

Then one can build, in expected

$$O\left(\frac{T(n,c,f)}{\gamma \cdot \log^2 n} + \big[Q(n,c,f) + D(n,c,f)\big] n \log n\right)$$

time, a data structure for answering $c(1+O(\gamma))$-NN queries in time $O(\log n)Q(n,c,f)$ with failure probability $O(f \log n)$. The resulting data structure uses $O(S(n,c,f)/\gamma \cdot \log^2 n)$ space. The number of points stored in this data structure is $O((n/\gamma) \cdot \log^2 n)$.

3 Approximate near neighbor

In this section, we describe data structures for the approximate *near* neighbor problem (i. e., (c,r)-NN) in ℓ_s^d. They can be used as subroutines in the reduction from the earlier section. Note that by scaling we can assume $r = 1$. We start by describing two simplifying reductions, followed by two algorithms for the approximate near neighbor problem.

3.1 Reductions

Coordinate range reduction We start by describing a method for reducing the range of coordinate values of points in $P \cup \{q\}$, for the case of the ℓ_1 norm. It is essentially due to Bern [11], who used it for the purpose of designing approximate nearest neighbor algorithms with approximation factor polynomial in d. This subroutine improves the efficiency of the data structures.

Lemma 3.1. *Fix $a > 1$, and suppose there is a data structure for $(c,1)$-NN in $[0,a]^d$ under the ℓ_1 norm that uses space $S(n,d,c,r,f)$, has query time $Q(n,d,c,r,f)$, and has failure probability f. Then there exists a data structure for the same problem over ℓ_1^d, with asymptotically the same time and space bounds, and failure probability $f + 1/a$.*

Proof. First, we impose an orthogonal cubic grid on ℓ_1^d, where each grid cell has side length a. The grid is translated by a random vector chosen uniformly at random from $[0,a]^d$. Consider any pair of points $p,q \in \ell_1^d$ such that $\|p-q\|_1 \le 1$. The probability that both p and q fall into the same grid cell is at least

$$1 - \sum_i \frac{|p_i - q_i|}{a} \ge 1 - \frac{1}{a}.$$

Therefore, we can proceed as follows:

- For each grid cell C such that $C \cap P \neq \emptyset$, build a separate data structure for the set $C \cap P$.
- In order to answer a query $q \in C$, query the data structure for $C \cap P$.

In this way we reduced the $(c,1)-$NN problem for points in ℓ_1^d to the case when the coordinates of points live in a cube of side a. The query time is increased by an additive term $O(d)$, which (as per the assumptions in Section 1.2) is subsumed by $O(Q(n,d,c,r,f))$. Since (as per the assumptions in Section 1.2) $S(n,d,c,r,f)$ is at least linear in dn, it follows that the space bound of the new data structure remains $O(S(n,d,c,r,f))$. The probability of failure is increased by an additive term of $1/a$. □

Hamming cube Next, we show that the problem can be simplified further by assuming that the points live in a low-dimensional Hamming cube.

Lemma 3.2. *Fix $\delta > 0$, $a > 1$, and suppose there is a data structure for (c,r)-NN in a dM-dimensional Hamming cube for $M = O(ad/\delta)$ that uses space $S(n,d,c,r,f)$, has query time $Q(n,d,c,r,f)$, and has failure probability f. Then there exists a data structure for $(c(1+\delta),1)$-NN for ℓ_1^d with the same time and space bounds, and failure probability $f + 1/a$.*

Proof. By Lemma 3.1 we can assume all points live in $[0,a]^d$. If we round all coordinates of points and queries to the nearest multiple of δ/d, then the inter-point distance are affected only by an additive factor of δ. Thus, solving a $(c,1+\delta)$-NN on the resulting point-set and queries yields a solution to $(c(1+\delta),1)$-NN over the original space ℓ_1^d.

In order to solve the $(c,1+\delta)$-NN problem, define a scaling factor $s = d/\delta$. Observe that, if we multiply all coordinates of the data points and queries by s, then all coordinates become integers in the range $\{0,\ldots,M\}$. In this case, we use a mapping unary : $\{0,\ldots,M\}^d \to \{0,1\}^{dM}$, such that

$$\mathrm{unary}((x_1,\ldots,x_d)) = \mathrm{unary}(x_1),\ldots,\mathrm{unary}(x_d),$$

where $\mathrm{unary}(x)$ is a string of x ones followed by a string of $M-x$ zeros. Observe that the ℓ_1 distance between any two points is equal to the Hamming distance of their images [44]. Thus, it now suffices to solve an $(c,(1+\delta)s)$-NN on the point-set and queries mapped by unary. □

In the following we focus on solving the problem in ℓ_s norms for $s \in \{1,2\}$. The generalization to any $s \in [1,2]$ follows from the theorem of [39], that states that for any d, γ, there exists a mapping $f : \ell_s^d \to \ell_1^{d'}$, $d' = O(d \log(1/\gamma)/\gamma^2)$, so that for any $p, q \in \ell_1^d$ we have

$$\|p-q\|_s \le \|f(p)-f(q)\|_1 \le (1+\gamma)\|p-q\|_s .$$

Note that the mapping f is defined for all points in ℓ_s^d, even if they are not known during the preprocessing. The mapping f is chosen at random from a certain distribution over linear mappings from ℓ_s^d to $\ell_1^{d'}$, which takes time $O(dd')$. This allows us to reduce $(c(1+\gamma), r)$-NN in ℓ_s^d to (c, r)-NN in $\ell_1^{d'}$.

3.2 Locality-sensitive hashing

In this section, we describe an algorithm based on the concept of *locality-sensitive hashing (LSH)*. The basic idea is to hash the data and query points in a way that the probability of collision is much higher for points that are close to each other than for those which are far apart. Formally, we require the following.

Definition 3.3. A family $\mathcal{H} = \{h : X \to U\}$ is (r_1, r_2, p_1, p_2)*-sensitive* for (X, D) if for any $q, p \in X$ we have

- if $D(p,q) \le r_1$ then $\Pr_{\mathcal{H}}[h(q) = h(p)] \ge p_1$,
- if $D(p,q) > r_2$ then $\Pr_{\mathcal{H}}[h(q) = h(p)] \le p_2$.

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

We can solve the (c, r)-NN problem using a locality-sensitive family 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 $\mathcal{G} = \{g : X \to U^k\}$ such that $g(p) = (h_{i_1}(p), \ldots, h_{i_k}(p))$, where $h_{i_t} \in \mathcal{H}$, for $I = \{i_1, \ldots, i_k\} \subset \{1, \ldots, |\mathcal{H}|\}$. For an integer L we choose L functions $g_1, \ldots, g_L$ from $\mathcal{G}$ independently and uniformly at random. Let I_j denote the multi-set defining g_j. During preprocessing, we store a pointer to each $p \in P$ in the buckets $g_1(p), \ldots, g_L(p)$. Since the total number of buckets may be large, we retain only the non-empty buckets by resorting to "standard" hashing.

To process a query q, we carry out a brute-force search for the neighbor of q in buckets $g_1(q), \ldots, g_L(q)$. As it is possible that the total number of points stored in those buckets is large, we interrupt the search after finding the first $3L$ points (including duplicates). Let $p_1, \ldots, p_t$ be the points encountered during this process. If there is any point p_j such that $D(p_j, q) \le r_2$ then we return the point p_j, else we return null.

Correctness The parameters k and L are chosen so as to ensure that with some constant probability the following two events hold. For any p^* we define two events:

- $E_1(q, p^*)$ occurs iff either $p^* \notin B(q, r)$ or $g_j(p^*) = g_j(q)$ for some $j = 1, \ldots, L$.
- $E_2(q)$ occurs iff the total number of collisions of q with points from $P - B(q, r_2)$ is less than $3L$, i. e.,

$$\sum_{j=1}^{L} |(P - B(q, r_2)) \cap g_j^{-1}(g_j(q))| < 3L .$$

Our first result is established for the oblivious case.

Theorem 3.4. *Suppose there is a (r, cr, p_1, p_2)-sensitive family $\mathcal{H}$ for (X, D), where $p_1, p_2 \in (0,1)$ and let $\rho = \log(1/p_1)/\log(1/p_2)$. Then there exists a fully dynamic data structure for (c,r)-NN over a set $P \subset X$ of at most n points, such that:*

- *The query procedure requires at most $O(n^\rho/p_1)$ distance computations, evaluations of the hash functions from $\mathcal{G}$ (that is, $O(n^\rho/p_1 \cdot \lceil \log_{1/p_2} n \rceil)$ evaluations of the hash functions from $\mathcal{H}$), or other operations; the same bound hold for the updates.*
- *The data structure uses at most $O(n^{1+\rho}/p_1)$ words of space, in addition to the space needed to store the set P.*

The failure probability of the data structure is at most $1/3 + 1/e < 1$.

Proof. Let q be a query point, and let P be the set of points at the time when the query is executed. Assume that there exists $p^* \in B(q, r_1)$ (otherwise there is nothing to prove).

Observe that if the events $E_1(q, p^*)$ and $E_2(q)$ hold, then the algorithm is correct. Thus it suffices to ensure that E_1 holds with probability P_1 and E_2 holds with probability P_2 such that both P_1 and P_2 are strictly greater than half.

Set $k = \lceil \log_{1/p_2} n \rceil$. Then the probability that $g(p') = g(q)$ for $p' \in P - B(q, r_2)$ is at most $p_2^k = 1/n$. 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. By Markov's inequality, the probability that this number exceeds $3L$ is less than $1/3$. Therefore the probability that event $E_2(q)$ holds is $P_2 \geq 2/3$.

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

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

Thus the probability that such a g_j exists is at least $\zeta = 1 - (1 - p_1 n^{-\rho})^L$. By setting $L = n^\rho/p_1$ we get $\zeta > 1 - 1/e$. The theorem follows. □

Remark 3.5. In the conference paper [35], the theorem and the proof implicitly assumed that the probabilities p_1 and p_2 are bounded away from 0. Although this assumption holds for the LSH functions described in this paper (see Section 3.2.1), this does not have to be the case in general. Therefore, in this paper we make the dependence on p_1 and p_2 explicit. See [50] for a further discussion on this point.

We now consider the adaptive case. Let m be an upper bound on the number of queries performed by the data structure, and let n be an upper bound on the number of points in the data structure at any time. Compared to the oblivious case, the main difference is that now we will need to ensure that all possible events E_1 and E_2 hold simultaneously. To this end we reduce the probability of failure by constructing $I = c\log(m+n)$ instances $\mathsf{Z}_1, \ldots, \mathsf{Z}_I$ of NearNbr using independent random coins, maintaining the same set P. In order to answer a query q, we query all Z_i's and return any valid answer.

Lemma 3.6. *There exists an absolute constant c such that*

$$\Pr[\textit{for all } p \in P \textit{ and } q \in Q\textit{, there exists } t \in \{1, \ldots, I\} \textit{ such that both } E_1(q,p) \textit{ and } E_2(q) \textit{ hold}] \geq 1 - 1/n\,.$$

Proof. We have

$$\begin{aligned}
\Pr\left[\exists_{p,q}\forall_t(E_1(p,q)\wedge E_2(q)) \text{ does not hold for } \mathsf{Z}_i\right] &\leq \sum_{p,q}\Pr\left[\forall_t(E_1(p,q)\wedge E_2(q)) \text{ does not hold for } \mathsf{Z}_i\right]\\
&= \sum_{p,q}\Pr\left[(E_1(p,q)\wedge E_2(q)) \text{ does not hold for } \mathsf{Z}_1\right]^I\\
&\leq nm(1/e+1/3)^I,
\end{aligned}$$

which is at most $1/n$ for an appropriate absolute constant $c>1$. □

Since E_1 and E_2 are monotone properties, if they hold for all the points of P and Q they also hold for any subsets, we have the following.

Corollary 3.7. *Consider any point-set Q of size m, and any point-set P of size n. There exists an absolute constant c such that the data structure obtained by running $c\log(n+m)$ copies of the algorithm from Theorem 3.4 is adaptive with respect to P and Q with probability of failure at most $1/n$.*

3.2.1 Hamming metric

In order to use Theorem 3.4 for the Hamming metric, we need to specify the proper family of hash functions. To this end we use the family of all projections of the bit vector onto one of its coordinates.

Proposition 3.8. *Let $D(p,q)$ be the Hamming metric for $p,q\in\Sigma^d$, where Σ is any finite alphabet. Then for any $r,\varepsilon>0$, the family*

$$\mathcal{H}=\{h_i : h_i((b_1,\dots,b_d))=b_i,\ i=1,\dots,d\}$$

is $(r,rc,1-r/d,1-rc/d)$-sensitive.

Remark 3.9. Note that by padding the input and query points with dummy coordinates equal to 0, we can increase d by a constant factor, and ensure that the probabilities p_1 and p_2 in the above family are bounded away from 0. This allows us to drop the dependence on these parameters in the reminder of this paper.

Corollary 3.10. *For any $c>1$, there exists an algorithm for (c,r)-NN in Hamming metric over Σ^d using $O(n^{1+1/c})$ space (in addition to space needed to store the input point set). The query and update time are bounded by the time needed to perform $O(n^{1/c})$ distance computations (each taking at most $O(d)$ time) and hash function evaluations (each taking $O(d/r\cdot\log n)$ time).*

Proof. We use Proposition 3.8 and Theorem 3.4. First, we need to estimate the value of

$$\rho=\frac{\ln 1/p_1}{\ln 1/p_2},$$

where $p_1=1-r/d$ and $p_2=1-rc/d$. Let $t=(1-p_2)/(1-p_1)$ and $x=1-p_1$. Then

$$\rho \;=\; \frac{\log(1-x)}{\log(1-tx)} \;\leq\; \frac{1}{t} \;=\; \frac{1-p_1}{1-p_2}$$

by the following claim.

Claim 3.11. *For $x \in [0,1)$ and $t \geq 1$ such that $1 - tx > 0$, we have*

$$\frac{\log(1-x)}{\log(1-tx)} \leq \frac{1}{t}.$$

Proof. Observing that $\log(1-tx) < 0$, the claim is equivalent to $t\log(1-x) \geq \log(1-tx)$. This in turn is equivalent to

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

This is trivially true for $x = 0$. Furthermore, taking the derivative, we see $f'(x) = -t + t(1-x)^{t-1}$, which is non-positive for $x \in [0,1)$ and $t \geq 1$. Therefore, f is non-increasing in the region in which we are interested, and so $f(x) \leq 0$ for all values in this region. □

Since $\rho = (1-p_1)/(1-p_2) = 1/c$, the bound on the exponent follows. The running time now follows from Theorem 3.4 since $\lceil \log_{1/p_2} n \rceil = O(d/r \cdot \log n)$. Finally, the space need to store the set P is $O(dn)$. □

3.2.2 Extension to normed spaces

In this section, we show how to extend the LSH algorithm to the ℓ_1 norm. The algorithm follows by composing Lemma 3.2 and Corollary 3.10.

Theorem 3.12. *For any $c > 1$, $\delta > 0$, there exists a randomized data structure (with constant probability of success) for the $(c(1+\delta),1)$-NN over a set of at most n points in ℓ_1^d, using $O(dn + n^{1+1/c})$ space and with $O(d/\delta \cdot n^{1/c} \log n)$ query and update time.*

Proof. The algorithm follows by composing Lemma 3.2 and Corollary 3.10. The latter lemma is invoked with $r = (1+\delta)d$ and the dimension of the Hamming cube bounded by $O(d^2/\delta)$. The theorem follows by observing that the distance computation (as in the statement of Corollary 3.10) can be done in $O(d)$ time. □

Corollary 3.13. *For any $c > 1$, $\delta > 0$, $\gamma \in (1/n, 1)$, there exists a randomized data structure (with constant probability of success) for the $c(1+\gamma)(1+\delta)$-NN over a set of at most n points in ℓ_1^d, using $O(dn + n^{1+1/c}\log^2(n)\log\log(n)/\gamma)$ space and with $O(d/\delta \cdot n^{1/c}\log^2(n)\log\log(n))$ query time.*

Proof. Follows by composing Theorem 3.12 and Theorem 2.9, with $f = O(1)/\log n$. □

Corollary 3.14. *For any $c > 1$, $\delta > 0$, there exists a randomized data structure for the $(c(1+\delta),1)$-NN over a set of at most n points in ℓ_1^d, using $O(dn + n^{1+1/c}\log n)$ space and with $O(d/\delta \cdot n^{1/c}\log^2 n)$ query and update time. For any point-set P of size n and query set Q of size $n^{O(1)}$ the algorithm is adaptive with respect to P and Q, with the failure probability of at most $1/n$.*

3.3 Bucketing

In this section, we show how to solve the $(c,1)$-NN problem in any norm ℓ_s^d, for any $c = 1+\varepsilon$, for $\varepsilon \in (0,1/2)$. The query time is very fast, i. e., $O(d)$. At the same time, the algorithm uses $(C/\varepsilon)^d n$ space for some constant $C > 1$. Furthermore, in case of the ℓ_s norm for $s \in \{1,2\}$ we can use the Johnson-Lindenstrauss lemma [38] to reduce d to $O(\log n/\varepsilon^2)$. This leads to space bound polynomial in n, for fixed $\varepsilon > 0$.

Assume for now that $s = 2$. Impose a uniform grid of side length $\varepsilon/\sqrt{d}$ on $\mathbb{R}^d$. Clearly, the distance between any two points belonging to one grid cell is at most ε. For each ball $B_i = B(p_i,1)$, $p_i \in P$, define $\overline{B}_i$ to be the set of grid cells intersecting B_i. Store all elements from $\bigcup_i \overline{B}_i$ 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 $0 < \varepsilon < 1$, $|\overline{B}_i| = O(1/\varepsilon)^d$. To see this, observe that $|\overline{B}_i|$ is bounded by the volume of a d-dimensional ball of radius $R = 2/\varepsilon \cdot \sqrt{d}$. We use the following fact [55, page 11].

Fact 3.15. *The* volume $V_s^d(R)$ *of a ball of radius R in* ℓ_s^d *is equal to*

$$\frac{(2\Gamma(1+1/s))^d}{\Gamma(1+n/s)} R^d\,,$$

where $\Gamma(\cdot)$ *is Euler's Gamma function. Specifically,*

$$V_2^d(R) = \frac{2\pi^{d/2}}{d\,\Gamma(d/2)} R^d\,.$$

Thus, $\left|\overline{B}\right| = 2^{O(d)} r^d/d^{d/2} \le (C/\varepsilon)^d$. Hence, the total space required is $O(n) \times O(1/\varepsilon)^d$. The query time is the time to compute the hash function.

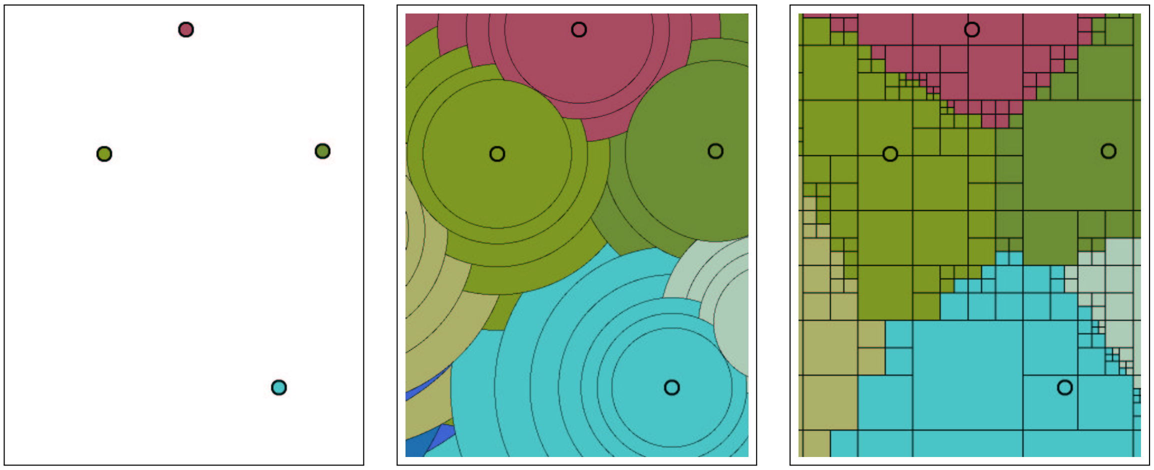
For general ℓ_s norms, we set the grid side length to $\varepsilon/d^{1/s}$. The bound on $|\overline{B}|$ applies unchanged.

Theorem 3.16. *For* $0 < \varepsilon < 1/2$*, there is an algorithm for* (c,r)*-NN in* ℓ_s^d *using* $O(n) \times O(1/\varepsilon)^d$ *space and preprocessing time and* $O(d)$ *query time.*

Dimensionality reduction If the dimension d is high, we can reduce the space bound using the Johnson-Lindenstrauss (JL) lemma [38]. For ℓ_2 we can apply the JL lemma directly and reduce d to $d' = O(\log n/\varepsilon^2)$. This leads to $n^{O(\log(1/\varepsilon)/\varepsilon^2)}$ space and preprocessing time, while the query time[3] becomes $O(dd')$.

For ℓ_1 we proceed as follows. First, we apply Lemma 3.2, and reduce the points to dM-dimensional Hamming space for $M = O(d)$. Since for any two points p,q in the Hamming space we have $D_H(p,q) = \|p-q\|^2$, we can now apply the dimensionality reduction to the points in the Hamming space. This increases the query time by an additive term of $O(dMd')$. This can be further reduced to $O(d)$, by pre-computing and storing dot products of unary(p_l) and the random vectors used to perform the

[3] Note that one can instead use *Fast Johnson-Lindenstrauss Transform*, which provides similar guarantees while reducing the embedding time [1, 2]. See the papers for the exact results.



(A) (B) (C)

Figure 1: (A) The point-set. (B) The generated set of NearNbr instances (different colors correspond to different data points). (C) The approximate Voronoi decomposition.

dimensionality reduction. Since each p_l is in the range $\{0,\dots,M\}$, it follows that for each j we need to store $M = O(d/\delta)$ numbers, for the total storage of $O(d^2/\delta)$.

Thus, we obtain the following theorem.

Theorem 3.17. *For $s \in \{1,2\}$, and $0 < \varepsilon < 1/2$, there is an algorithm for (c,r)-NN in ℓ_s^d using $n^{O(\log(1/\varepsilon)/\varepsilon^2)}$ space and preprocessing time and $O(d \log n/\varepsilon^2)$ query time.*

3.4 Approximate Voronoi diagram

Section 3.3 and Theorem 2.10 reduce the c-NN problem to performing $O(\log n)$ lookups in appropriate grids. It is natural to ask if one can collapse all these grids together, and obtain some natural geometric representation of the input point set. In this section we show that this is indeed the case. Specifically, we show the following theorem.

Theorem 3.18. *There is an absolute constant $C > 0$ such that given a set P of n points in $\mathbb{R}^d$ and a parameter $\varepsilon \in (0,1/2)$, one can compute, in $O((n(C/\varepsilon)^{d+1})\log^3 n)$ time, an approximate Voronoi diagram of P of size $O((n(C/\varepsilon)^{d+1})\log^2 n)$, such that*

- *every cell in this diagram is either a cube or a set difference of two cubes, and*
- *every such cell has a point of P associated with it that is a $(1+\varepsilon)$-approximate nearest neighbor of all the points of the cell.*

Proof. We follow the reduction from the approximate nearest to near neighbor in Section 2 combined with the bucketing approach to approximate near neighbor given in Section 3.3. First, observe that by scaling and minor readjustment of parameters, we can assume that the side lengths of the grids used in the reduction are powers of 2. Also, we can assume that they are centered at the origin. As a result, the grids are nested—a cell in a finer grid is fully contained in some cell of a coarser grid.

Consider instances $\mathtt{Z}_1,\ldots,\mathtt{Z}_k$ of NearNbr generated by the reduction. Let r_i be the radius of the instance $\mathtt{Z}_i$, and let t_i be the cell side length of the grid corresponding to $\mathtt{Z}_i$. For every $\mathtt{Z}_i$ and every point p represented by that data structure, we label every such grid cell intersecting the ball $B(p,r_i)$ with p. If a grid cell is marked by several balls, its label is the smallest such ball. In the end of this process we have a total of $O(n(C/\varepsilon)^{d+1}\log^2 n)$ grid cells (of different resolution) that were labeled.

Consider any query point q, and let U be the grid cell of the smallest side length that contains q. By straightforward but tedious argument following the recursive construction in Theorem 2.10 we have that the label of U is a correct approximate nearest neighbor for q. The decomposition can be now obtained by superimposing all grids onto the space $\mathbb{R}^d$, where any point takes the label of the smallest grid cell containing it. □

This concludes the description of the approximate Voronoi decomposition construction. In the reminder of this section, we describe various ways in which the construction can be used for *finding* an approximate nearest neighbor.

From the perspective of this paper, the simplest approach is to apply the search algorithm from Section 2.3. An alternative (described, e. g., in [30, 32, 31]) is to construct a so-called *compressed d-dimensional quadtree* for the grid cells constructed in Theorem 3.18. The data structure enables locating the smallest grid cell in time $O(d\log n)$, which matches the time achieved by our search algorithm. However, it is possible to improve it further. In particular, consider data sets with "low" *aspect ratio*, i. e., the ratio of their diameter to the closest distance between any two distinct points. For such point sets, one can perform searches in quadtrees in time faster than $O(\log n)$. See [32, 31] for a more detailed description.

4 Minimum Spanning Tree

In this section, we show how to use NearNbr data structure(s) to find a $c(1+\gamma)$-approximate MST for a set of points in ℓ_s^d, for $s\in\{1,2\}$. Our algorithm is based on Kruskal's method [23]: we keep adding edges between points in approximately increasing order, and merging components that are connected by the edges. The algorithm proceeds in stages. In the ith stage, we identify pairs of points that are (approximately) within a distance of r_i from each other, and merge the components connecting the points. This is done using an algorithm APPROXIMATECC from Algorithm 1.

Recall that APPROXIMATECC(P,c,r) returns a partition $\mathcal{P}'$ of P such that $\mathrm{CC}_P(r)\sqsubseteq\mathcal{P}'\sqsubseteq cc_P(cr)$.

Lemma 4.1. *The above algorithm reports a $c(1+2\gamma)$-approximate MST.*

Proof. Let $e_1,e_2,\ldots,e_{n-1}$ be the edges in the set E, ordered by the stage when they were picked (within the same stage the edges are ordered arbitrarily). Consider the exact Kruskal algorithm, which examines the edges between points in non-decreasing order, and selects those whose endpoints belong to different sets; let $e_1^*,e_2^*,\ldots,e_{n-1}^*$ be the edges picked by that algorithm. We will show that the sum of the costs of e_t's is at most $c(1+\gamma)$ times the sum of the costs of e_t^*'s.

We start from edges processed at stage $i=0$. Since the algorithm collects at most $n-1$ edges, and each edge collected at stage 1 has length at most $cr_0=cr\gamma/n$, the total cost of edges collected at this stage is at most $cr\gamma$. This is at most $c\gamma$ times the MST cost.

```
E = ∅
Select q ∈ P
Let r be the maximum value of D(q,p) over p ∈ P
Define r_i = r/(1+γ)^{M−i}, for i = 0,...,M, where M = log_{1+γ}(n/γ)
Create 𝒫 = {p_1},{p_2},...,{p_n}, where P = {p_1,...,p_n}
for i = 0,...,M do
    Invoke APPROXIMATECC(P,c,r_i)
    Let E′ be the set of edges generated by the algorithm
    for e = {u,v} ∈ E′ do
        if u and v belong to different sets P_i,P_j in the partition 𝒫 then
            Merge P_i and P_j in 𝒫
            Add e to E
        end if
    end for
end for
```

Algorithm 4: Approximate minimum spanning tree algorithm.

Consider an edge e_t processed at stage i. We have that the cost of e_t is greater than r_{i-1}, since otherwise this edge would have been picked up in the earlier stage. Let l be the largest index such that the cost of e^*_l is at most r_{i-1}. Since all nodes connected by edges $e^*_1,\ldots,e^*_l$ are connected before stage i, it follows that $t > l$.

Since the cost of e_t is at most cr_i and the cost of e^*_t is greater than r_{i-1}, we have that the cost of e_t is at most $c(1+\gamma)$ times the cost of e^*_t. The lemma follows. □

Theorem 4.2. *A $c(1+2\gamma)$-approximate MST of n points in ℓ_1^d can be computed in $O(dn^{1+1/c}\log^3(n)/\gamma)$ time.*

Proof. The algorithm makes $O(\log n)$ calls to APPROXIMATECC, each using $O(n)$ query and delete operations on a data structure provided by Corollary 3.14. □

The extension to ℓ_s^d norms for $s \in [1,2]$ follows from the embedding of [39].

Acknowledgments

The authors would like to thank Alexandr Andoni, Jelani Nelson, Steve Oudot, Ashish Goel and the referees for their very helpful comments on this paper.

References

[1] NIR AILON AND BERNARD CHAZELLE: The fast Johnson–Lindenstrauss transform and approximate nearest neighbors. *SIAM J. Comput.*, 39(1):302–322, 2009. Preliminary version in STOC'06. [doi:10.1137/060673096] 341

[2] NIR AILON AND BERNARD CHAZELLE: Faster dimension reduction. *Commun. ACM*, 53(2):97–104, 2010. [doi:10.1145/1646353.1646379] 341

[3] ALEXANDR ANDONI AND PIOTR INDYK: Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. In *Proc. 47th FOCS*, pp. 459–468. IEEE Comp. Soc. Press, 2006. [doi:10.1109/FOCS.2006.49] 326

[4] ALEXANDR ANDONI AND PIOTR INDYK: Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. *Commun. ACM*, 51(1):117–122, 2008. [doi:10.1145/1327452.1327494] 326

[5] ALEXANDR ANDONI, PIOTR INDYK, AND ROBERT KRAUTHGAMER: Earth mover distance over high-dimensional spaces. In *Proc. 19th Ann. ACM-SIAM Symp. on Discrete Algorithms (SODA'08)*, pp. 343–352. ACM Press, 2008. [ACM:1347082.1347120] 326

[6] ALEXANDR ANDONI, PIOTR INDYK, AND MIHAI PĂTRAŞCU: On the optimality of the dimensionality reduction method. In *Proc. 47th FOCS*, pp. 449–458. IEEE Comp. Soc. Press, 2006. [doi:10.1109/FOCS.2006.56] 326

[7] SUNIL ARYA AND THEOCHARIS MALAMATOS: Linear-size approximate Voronoi diagrams. In *Proc. 13th Ann. ACM-SIAM Symp. on Discrete Algorithms (SODA'02)*, pp. 147–155. ACM Press, 2002. [ACM:545400] 326

[8] SUNIL ARYA, THEOCHARIS MALAMATOS, AND DAVID M. MOUNT: Space-time tradeoffs for approximate nearest neighbor searching. *J. ACM*, 57(1):1:1–1:54, 2009. [doi:10.1145/1613676.1613677] 326

[9] SUNIL ARYA AND DAVID M. MOUNT: Approximate nearest neighbor queries in fixed dimensions. In *Proc. 4th Ann. ACM-SIAM Symp. on Discrete Algorithms (SODA'93)*, pp. 271–280. ACM Press, 1993. [ACM:313768] 324

[10] SUNIL ARYA, DAVID M. MOUNT, NATHAN S. NETANYAHU, RUTH SILVERMAN, AND ANGELA Y. WU: An optimal algorithm for approximate nearest neighbor searching in fixed dimensions. *J. ACM*, 45(6):891–923, 1998. Preliminary version in SODA'94. [doi:10.1145/293347.293348] 324

[11] MARSHALL W. BERN: Approximate closest-point queries in high dimensions. *Inform. Process. Lett.*, 45(2):95–99, 1993. [doi:10.1016/0020-0190(93)90222-U] 324, 335

[12] ALLAN BORODIN, RAFAIL OSTROVSKY, AND YUVAL RABANI: Subquadratic approximation algorithms for clustering problems in high dimensional spaces. *Machine Learning*, 56(1-3):153–167, 2004. Preliminary version in STOC'99. [doi:10.1023/B:MACH.0000033118.09057.80] 324, 325

[13] ANDREI Z. BRODER: On the resemblance and containment of documents. In *Proc. Conf. Compres. Complex. Sequences (CCCS'97)*, pp. 21–29, 1997. [doi:10.1109/SEQUEN.1997.666900] 325

[14] ANDREI Z. BRODER, STEVEN C. GLASSMAN, MARK S. MANASSE, AND GEOFFREY ZWEIG: Syntactic clustering of the web. *Computer Networks and ISDN Systems*, 29(8-13):1157–1166, 1997. (Proc. 6th Int. World Wide Web Conf. (WWW'97)). [doi:10.1016/S0169-7552(97)00031-7] 325

[15] JEREMY BUHLER: Efficient large-scale sequence comparison by locality-sensitive hashing. *Bioinformatics*, 17(5):419–428, 2001. [doi:10.1093/bioinformatics/17.5.419] 326

[16] JEREMY BUHLER AND MARTIN TOMPA: Finding motifs using random projections. *J. Computational Biology*, 9(2):225–242, 2002. Preliminary version in RECOMB'01. [doi:10.1089/10665270252935430] 326

[17] ANDREA CALIFANO AND ISIDORE RIGOUTSOS: FLASH: A fast look-up algorithm for string homology. In *Proc. 1st Int. Conf. on Intelligent Systems for Molecular Biology (ISMB'93)*, pp. 56–64, 1993. [ACM:645630.662865] 325

[18] PAUL B. CALLAHAN AND S. RAO KOSARAJU: A decomposition of multidimensional point sets with applications to k-nearest-neighbors and n-body potential fields. *J. ACM*, 42(1):67–90, 1995. Preliminary version in STOC'92. [doi:10.1145/200836.200853] 326

[19] TIMOTHY M. CHAN: Approximate nearest neighbor queries revisited. *Discrete Comput. Geom.*, 20(3):359–373, 1998. Preliminary version in SCG'97. [doi:10.1007/PL00009390] 324

[20] MOSES CHARIKAR AND ROBERT KRAUTHGAMER: Embedding the Ulam metric into ℓ_1. *Theory of Computing*, 2(1):207–224, 2006. [doi:10.4086/toc.2006.v002a011] 326

[21] MOSES S. CHARIKAR: Similarity estimation techniques from rounding algorithms. In *Proc. 34th STOC*, pp. 380–388. ACM Press, 2002. [doi:10.1145/509907.509965] 326

[22] KENNETH L. CLARKSON: A randomized algorithm for closest-point queries. *SIAM J. Comput.*, 17(4):830–847, 1988. [doi:10.1137/0217052] 322, 324

[23] THOMAS H. CORMEN, CHARLES E. LEISERSON, RONALD L. RIVEST, AND CLIFFORD STEIN: *Introduction to Algorithms*. MIT Press, 2nd edition, 2001. 343

[24] GRAHAM CORMODE: *Sequence Distance Embeddings*. Ph.D. Thesis. University of Warwick, 2003. 326

[25] GRAHAM CORMODE, S. MUTHUKRISHNAN, AND SÜLEYMAN CENK SAHINALP: Permutation editing and matching via embeddings. In *Proc. 28th Internat. Colloq. on Automata, Languages and Programming (ICALP'01)*, pp. 481–492, 2001. [doi:10.1007/3-540-48224-5_40] 326

[26] MAYUR DATAR, NICOLE IMMORLICA, PIOTR INDYK, AND VAHAB S. MIRROKNI: Locality-sensitive hashing scheme based on p-stable distributions. In *Proc. 20th Ann. Symp. on Computational Geometry (SCG'04)*, pp. 253–262, 2004. [doi:10.1145/997817.997857] 326

[27] DEBOJYOTI DUTTA, RAJARSHI GUHA, PETER C. JURS, AND TING CHEN: Scalable partitioning and exploration of chemical spaces using geometric hashing. *Journal of Chemical Information and Modeling*, 46(1):321–333, 2006. [doi:10.1021/ci050403o] 326

[28] KRISTEN GRAUMAN AND TREVOR DARRELL: The pyramid match kernel: Efficient learning with sets of features. *J. Machine Learning Research*, 8:725–760, 2007. Preliminary version in ICCV'05. [ACM:1314524] 326

[29] DAN GREENE, MICHAL PARNAS, AND FRANCES YAO: Multi-index hashing for information retrieval. In *Proc. 35th FOCS*, pp. 722–731. IEEE Comp. Soc. Press, 1994. [doi:10.1109/SFCS.1994.365720] 325

[30] SARIEL HAR-PELED: A replacement for Voronoi diagrams of near linear size. In *Proc. 42nd FOCS*, pp. 94–103. IEEE Comp. Soc. Press, 2001. [doi:10.1109/SFCS.2001.959884] 323, 324, 325, 326, 343

[31] SARIEL HAR-PELED: *Geometric Approximation Algorithms*. Amer. Math. Soc., 2011. 343

[32] SARIEL HAR-PELED AND SOHAM MAZUMDAR: On coresets for k-means and k-median clustering. In *Proc. 36th STOC*, pp. 291–300. ACM Press, 2004. [doi:10.1145/1007352.1007400] 343

[33] TAHER H. HAVELIWALA, ARISTIDES GIONIS, DAN KLEIN, AND PIOTR INDYK: Evaluating strategies for similarity search on the web. In *Proc. 11th Internat. Conf. on World Wide Web (WWW'02)*, pp. 432–442. ACM Press, 2002. [doi:10.1145/511446.511502] 326

[34] PIOTR INDYK: Nearest neighbors in high-dimensional spaces. In J. E. GOODMAN AND J. O'ROURKE, editors, *Handbook of Discrete and Computational Geometry*, chapter 39, pp. 877–892. CRC Press LLC, 2nd edition, 2004. 323, 325, 326

[35] PIOTR INDYK AND RAJEEV MOTWANI: Approximate nearest neighbors: Towards removing the curse of dimensionality. In *Proc. 30th STOC*, pp. 604–613. ACM Press, 1998. [doi:10.1145/276698.276876] 323, 324, 325, 338

[36] PIOTR INDYK AND RAJEEV MOTWANI: Approximate nearest neighbors: towards removing the curse of dimensionality. *A final version of the STOC'98 paper, available at* `http://people.csail.mit.edu/indyk/nndraft.ps`, 1999. 324, 325

[37] PIOTR INDYK AND NITIN THAPER: Fast color image retrieval via embeddings. In *Work. Statis. Comput. Theo. Vision*, 2003. Held at ICCV'03. [PDF]. 326

[38] WILLIAM B. JOHNSON AND JORAM LINDENSTRAUSS: Extensions of Lipschitz mapping into Hilbert space. *Contemporary Mathematics*, 26:189–206, 1984. 323, 341

[39] WILLIAM B. JOHNSON AND GIDEON SCHECHTMAN: Embedding l_p^m into l_1^n. *Acta Mathematica*, 149:71–85, 1982. [doi:10.1007/BF02392350] 323, 337, 344

[40] RICHARD M. KARP, ORLI WAARTS, AND GEOFFREY ZWEIG: The bit vector intersection problem. In *Proc. 36th FOCS*, pp. 621–630. IEEE Comp. Soc. Press, 1995. [doi:10.1109/SFCS.1995.492663] 325

[41] SUBHASH KHOT AND ASSAF NAOR: Nonembeddability theorems via Fourier analysis. *Mathematische Annalen*, 334(4):821–852, 2006. Preliminary version in FOCS'05. [doi:10.1007/s00208-005-0745-0] 326

[42] JON M. KLEINBERG: Two algorithms for nearest-neighbor search in high dimensions. In *Proc. 29th STOC*, pp. 599–608. ACM Press, 1997. [doi:10.1145/258533.258653] 325

[43] EYAL KUSHILEVITZ, RAFAIL OSTROVSKY, AND YUVAL RABANI: Efficient search for approximate nearest neighbor in high dimensional spaces. *SIAM J. Comput.*, 30(2):457–474, 2000. Preliminary version at STOC'98. [doi:10.1137/S0097539798347177] 325

[44] NATHAN LINIAL, ERAN LONDON, AND YURI RABINOVICH: The geometry of graphs and some of its algorithmic applications. *Combinatorica*, 15(2):215–245, 1995. Preliminary version in FOCS'94. [doi:10.1007/BF01200757] 336

[45] RICHARD J. LIPTON AND ROBERT ENDRE TARJAN: Applications of a planar separator theorem. *SIAM J. Comput.*, 9(3):615–627, 1980. [doi:10.1137/0209046] 322

[46] STEFAN MEISER: Point location in arrangements of hyperplanes. *Inform. and Comput.*, 106(2):286–303, 1993. [doi:10.1006/inco.1993.1057] 322

[47] MARVIN MINSKY AND SEYMOUR PAPERT: *Perceptrons*. MIT Press, Cambridge, MA, 1969. 322

[48] RAJEEV MOTWANI, ASSAF NAOR, AND RINA PANIGRAHY: Lower bounds on locality sensitive hashing. *SIAM J. Discrete Math.*, 21(4):930–935, 2007. Preliminary version at SCG'06. [doi:10.1137/050646858] 326

[49] S. MUTHUKRISHNAN AND SÜLEYMAN CENK SAHINALP: Approximate nearest neighbors and sequence comparison with block operations. In *Proc. 32nd STOC*, pp. 416–424. ACM Press, 2000. [doi:10.1145/335305.335353] 326

[50] RYAN O'DONNELL, YI WU, AND YUAN ZHOU: Optimal lower bounds for locality sensitive hashing (except when q is tiny). In *Proc. 2nd Symp. Innovations in Comput. Sci. (ICS'11)*, pp. 275–283, 2011. [ICS'11], [arXiv]. 326, 338

[51] RAFAIL OSTROVSKY AND YUVAL RABANI: Low distortion embeddings for edit distance. *J. ACM*, 54(5), 2007. Preliminary version in STOC'05. [doi:10.1145/1284320.1284322] 326

[52] RINA PANIGRAHY: Entropy based nearest neighbor search in high dimensions. In *Proc. 17th Ann. ACM-SIAM Symp. on Discrete Algorithms (SODA'06)*, pp. 1186–1195. ACM Press, 2006. [doi:10.1145/1109557.1109688] 326

[53] RINA PANIGRAHY, KUNAL TALWAR, AND UDI WIEDER: Lower bounds on near neighbor search via metric expansion. In *Proc. 51st FOCS*, pp. 805–814. IEEE Comp. Soc. Press, 2010. [doi:10.1109/FOCS.2010.82] 326

[54] RAMAMOHAN PATURI, SANGUTHEVAR RAJASEKARAN, AND JOHN REIF: The light bulb problem. *Inform. and Comput.*, 117(2):187–192, 1995. Preliminary version in COLT'89. [doi:10.1006/inco.1995.1038] 325

[55] GILLES PISIER: *The volume of convex bodies and Banach space geometry*. Cambridge University Press, 1989. 341

[56] DEEPAK RAVICHANDRAN, PATRICK PANTEL, AND EDUARD HOVY: Randomized algorithms and NLP: Using locality sensitive hash functions for high speed noun clustering. In *Proc. Conf. 43rd Ann. Meeting of the Association for Computational Linguistics (ACL'05)*, 2005. [ACM:1219917] 326

[57] YOGISH SABHARWAL, NISHANT SHARMA, AND SANDEEP SEN: Nearest neighbors search using point location in balls with applications to approximate Voronoi decompositions. *J. Comput. System Sci.*, 72(6):955–977, 2006. Preliminary version at FSTTCS'02. [doi:10.1016/j.jcss.2006.01.007] 326

[58] GREGORY SHAKHNAROVICH, TREVOR DARRELL, AND PIOTR INDYK, editors. *Nearest Neighbor Methods in Learning and Vision*. Neural Processing Information Series, MIT Press, 2006. 326

[59] MICHAEL IAN SHAMOS AND DAN HOEY: Closest-point problems. In *Proc. 16th FOCS*, pp. 151–162. IEEE Comp. Soc. Press, 1975. [doi:10.1109/SFCS.1975.8] 322

[60] RICHARD WEBER, HANS-J. SCHEK, AND STEPHEN BLOTT: A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In *Proc. 24th Internat. Conf. on Very Large Data Bases (VLDB'98)*, pp. 194–205, 1998. [ACM:671192] 322

AUTHORS

Sariel Har-Peled
Associate Professor
Department of Computer Science, University of Illinois
sariel@cs.uiuc.edu
http://www.uiuc.edu/~sariel/

Piotr Indyk
Professor
Department of Electrical Engineering and Computer Science, MIT
indyk@mit.edu
http://people.csail.mit.edu/indyk/

Rajeev Motwani
Former Professor
Department of Computer Science, Stanford University
http://cs.stanford.edu/~rajeev/

ABOUT THE AUTHORS

SARIEL HAR-PELED graduated from Tel-Aviv University in 1999, under the supervision of Micha Sharir. He works mainly on geometric approximation algorithms. He often teaches the “Randomized Algorithms" course based on the book by Rajeev Motwani and Prabhakar Raghavan.

PIOTR INDYK received his Ph. D. from Stanford University in 2000, under the supervision of Rajeev Motwani. His research interests include high-dimensional computational geometry, sketching and streaming algorithms, sparse recovery, and compressive sensing.

RAJEEV MOTWANI was born on March 24, 1962 in Jammu, India. He died on June 5, 2009. He received a B. Tech degree in Computer Science from IIT Kanpur in 1983 and a Ph. D. degree in Computer Science from University of California at Berkeley in 1988 under the supervision of Richard Karp. The list of his research interests is long and eclectic, and includes graph theory, approximation algorithms, randomized algorithms, on-line algorithms, complexity theory, web search and information retrieval, databases, data mining, computational drug design, robotics, streaming algorithms, and data privacy. He received the Gödel Prize in 2001 for his research on probabilistically checkable proofs and hardness of approximation.