Partially Specified Nearest Neighbor Search

Tomas Hruz and Marcel Schöngens
Institute of Theoretical Computer Science, ETH Zurich, Switzerland
{tomas.hruz, schoengens}@inf.ethz.ch

Abstract

We study the Partial Nearest Neighbor Problem that consists in preprocessing n points $\mathcal{D}$ from d-dimensional metric space such that the following query can be answered efficiently: Given a query vector $Q \in \mathbb{R}^d$ and an axes-aligned query subspace represented by $S \in \{0,1\}^d$, report a point $P \in \mathcal{D}$ with $\mathrm{d}_S(Q,P) \leq \mathrm{d}_S(Q,P')$ for all $P' \in \mathcal{D}$, where $\mathrm{d}_S(Q,P)$ is the distance between Q and P in the subspace S. This problem is related to similarity search between feature vectors w.r.t. *a subset* of features. Thus, the problem is of great practical importance in bioinformatics, image recognition, etc., however, due to exponentially many subspaces, each changing distances significantly, the problem has a considerable complexity. We present the first exact algorithms for ℓ_2- and ℓ_∞-metrics with linear space and sub-linear worst-case query time. We also give a simple approximation algorithm, and show experimentally that our approach performs well on real world data.

1 Introduction

One of the fundamental problems in the study of index structures is the *Nearest Neighbor (NN) Problem*: For a database $\mathcal{D}$ that contains n points from a d-dimensional metric space, build a data structure such that given a query point $Q \in \mathbb{R}^d$, one can efficiently find a point $P \in \mathcal{D}$ closest to Q. Such a point P is called *closest point* or *nearest neighbor* of Q and closest is meant with respect to the underlying metric. To stay in a well-studied space, in this paper we consider the ℓ_2 and ℓ_∞-metric, though the presented approach also works for other ℓ_p-norms. By $\mathrm{d}(P,Q)$ we denote the distance between two points $P, Q \in \mathbb{R}^d$, and, if not stated otherwise, it refers to the ℓ_2-distance. We understand the points in $\mathcal{D}$ as feature representations, or feature vectors, of real world objects. Hence, a nearest neighbor represents an object similar to the object the query represents.

The Nearest Neighbor Problem has been studied extensively in the last decades. However, in some important practical applications the problem formulation above is too restrictive in the sense that one is often interested in finding a point in $\mathcal{D}$ that is similar to a query with respect *to subset of features*. This leads to the definition of the *Partial Nearest Neighbor (PNN) Problem*: Build a data structure such that for a query point $Q \in \mathbb{R}^d$ together with a

query subspace represented by a vector $S \in \{0,1\}^d$, one can efficiently find a point $P = (p_1, p_2, \ldots, p_d) \in \mathcal{D}$ that is closest to $Q = (q_1, q_2, \ldots, q_d)$ with respect to the subspace $S = (s_1, s_2, \ldots, s_d)$. More formally, in Euclidean space we search for a point P with $\mathrm{d}_S(P,Q) \leq \mathrm{d}_S(P',Q)$ for all $P' \in \mathcal{D}$ where $\mathrm{d}_S(P,Q) = \left(\sum_{i=1}^{d} s_i(p_i - q_i)^2\right)^{1/2}$. The subspace represented by S is always orthogonal to the coordinate axes. The dimension of the subspace is denoted by w and we may say a query has w *relevant dimensions*. There are exponentially many subspaces and in general for each subspace the point distances change significantly, which is the main reason for the problem's complexity. Our approach considers small to medium feature space, which is supported by our experimental evaluation that shows good results for up to 64 dimensions (Section 4).

From the application perspective it is often sufficient to find a point in $\mathcal{D}$ that is relatively close to the query point, but not necessarily the closest. This relaxation is captured by the *Approximate Partial Nearest Neighbor (APNN) Problem* in which the objective is to find a point $P \in \mathcal{D}$ with $\mathrm{d}_S(P,Q) \leq (1+\gamma)\mathrm{d}_S(P',Q)$ for all $P' \in \mathcal{D}$ and a predefined $\gamma > 0$. In the case in which we restrict ourselves to the full space, that means $S = (1, 1, \ldots)$, the problem specializes to the well-known *Approximate Nearest Neighbor Problem* [13].

The PNN problem occurs in many application areas such as bioinformatics, image recognition, business data mining, and many more. A specific example, which was the motivation to study the problem, is knowledge retrieval in gene expression databases, in which biologists are interested in finding genes with a similar response profile to a given gene. In the database, a feature vector represents a gene and a feature represents the gene's activity in a certain anatomical part of the organism like a brain cell, muscle cell, blood cell, etc. The anatomical parts are organized in classes according to an ontology, in which the number of classes is the dimension d of the full feature space (see for example [24, 12]). The set of classes according to which the search is made changes dynamically, hence, this problem is essentially the (A)PNN Problem.

For the classical NN Problem worst-case efficient algorithms have been developed in the last decades, but they are still missing for the PNN Problem. A trivial approach is to preprocess a NN data structure for each of the 2^d subspaces, such that the query time is the time needed to find the correct NN data structure plus its query time. This approach has exponential space requirements which especially unpractical for main memory data structures. We focus on data structures that have only linear space requirements. Known data structures for nearest neighbor and range searching were studied assuming constant d, thus the influence of d on the query time is usually not known but believed to be exponential. Obtaining good bounds is sophisticated even in the case $d = 2$ [19]. Hence, in the query time analysis we express the dependence of d as some unknown function $\mathrm{g}(d)$.

Our Results. Space is the most critical resource when dealing with the PNN problem (even for moderate $d \geq 10$). Already for small data sets there is no

chance to deal with the brute force' exponential overhead in space, which is not a worst-case but a tight estimate. Consequently, we consider linear space data structures, which come with a worse but still sub-linear query time.

The presented method combines epsilon-nets and range searching data structures to provide an efficient and simple approach to the PNN problem, leading to the first linear space algorithm that for the ℓ_2- and ℓ_∞-metric has a guaranteed worst-case query time sub-linear in n. This result can easily be extended to ℓ_p-metrics for fixed p.

For the Euclidean metric and a d-dimensional database, we obtain a query time of $\mathcal{O}\left(\mathrm{g}(d)n^{1-1/(2d-4)+\delta}\right)$ w.h.p., for $\delta > 0$, and $d > 3$ using $\mathcal{O}(dn)$ space (Lemma 4). On the other hand, the known lower bounds on space decomposition [22, 14] give less hope to obtain fundamentally better exact algorithms.

For the ℓ_∞-metric we obtain an exact algorithm that has $\mathcal{O}\left(dn^{1-1/d}\right)$ query time w.h.p. while using $\mathcal{O}(dn)$ space (Lemma 5). The approach is very flexible and can be adapted to any metric that can be described by a constant number of bounded degree polynomials (using Lemma 3), which also includes the ℓ_1-metric.

We present a $\sqrt{w}$-approximation algorithm that has the same complexity as the ℓ_∞-metric case (Lemma 7). This approach is of high practical relevance: An experimental evaluation shows that it is more than 60 times faster than a linear scan while the approximation ratio is only 1.2 on average, and the maximal rank over all queries was always less than 0.025% of the database (Section 4).

2 Related Work

In theory and practice little is known about the PNN Problem, which might be due to the problem complexity that manifests in an exponential number of possible subspaces. There are some heuristics for the problem that perform well on real-world data, but usually they have no sublinear worst-case bound on the query time, or do not guarantee a good solution quality.

The problem was first considered in a paper by Eastman and Zemankova in 1982 [9] in which KD Trees are used as the data structure of choice. The authors prove that for uniformly distributed database points under the maximum norm the expected query time is $\mathcal{O}\left(dn^{1-1/d}\right)$. With the KD tree, our approach achieves the same result under the maximum norm, but for any point distribution.

A theoretical paper by Andoni et. al. [2] treats a special case of the PNN problem, namely, the case if all but one feature are specified. The authors study the problem in high-dimensional space in which the dimension d is not considered constant. A $(1+\gamma)$-approximation can be obtained in time $\mathcal{O}\left(d^3 n^{0.5+\delta}\right)$ using $\mathcal{O}\left(d^2 n^{c(1/\gamma^2+1/\delta^2)}\right)$ space for any $\delta > 0, \gamma > 0$, and a constant c. The result makes use of the LSH scheme [13], which is a very successful approach for the classical NN Problem in high-dimensional space. However, the approach has not been extended to smaller relevant dimensions than $w = d - 1$. In contrast

our approach works for all w in an axes-parallel setting.

In [5] the authors develop and experimentally examine a heuristic method to solve the PNN Problem in the ℓ_p-metric using R-trees (resp. R*-trees). The algorithm uses an ordered active page list (APL) which contains the R-tree nodes (enclosing rectangles) sorted in ascending order according to the distance between query and the nearest edge of the enclosing rectangle. This is essentially a min-heap, which is initialized with the tree's root node. The algorithm iteratively pops the first node in the APL and pushes the node's children into the APL if their distance to the query is smaller than the current candidate nearest neighbor. The algorithm provides accurate results and it is fast in many cases as the experiments in [5] show, however, it cannot provide a worst-case time complexity better than a linear scan.

A heuristic for partial range searching, which is related to the PNN Problem, was proposed in [15] in which the authors study partial vector approximation files (VA-files) [21]. One dimensional VA-files with variable interval distribution are constructed with the consequence that the variable intervals allow to order the VA-files according to the so-called sensitivity. The sensitivity is defined as number of VA interval end-points lying in the search range. It allows to correct results in many situations. This heuristic has no guarantee on the accuracy of the results, nor a sublinear worst-case guarantee on the query time.

3 Our Approach

We show that a combination of epsilon-nets and range searching can be used to obtain the first exact algorithm for the PNN Problem. Furthermore, the algorithm can be modified such that one obtains better query time for the APNN Problem. The algorithm is based on the concepts of epsilon-nets and range spaces: A *range space* $\mathfrak{S}$ is a 2-tuple $(\mathcal{X}, \mathfrak{R})$, where $\mathcal{X}$ is a usually infinite set and $\mathfrak{R}$ is a collection of subsets of $\mathcal{X}$. The elements of $\mathcal{X}$ are called points and the elements of $\mathfrak{R}$ are called *ranges*. For example, a range could be an axes-parallel, in some dimensions unbounded, hyper-rectangle $\mathcal{H}_{L,R} = \{(x_1, x_2, \ldots, x_d) \mid l_i \leq x_i \leq r_i\}$, represented by two vectors $L = (l_1, l_2, \ldots, l_d)$ and $R = (r_1, r_2, \ldots, r_d)$, where $l_i, r_i \in \mathbb{R} \cup \{-\infty, \infty\}$. Let us denote the set of all hyper-rectangles by $\mathfrak{H}$ so that we can specify the range space $(\mathbb{R}^d, \mathfrak{H})$. To simplify notation, we always denote a vector from $\mathbb{R}^d$ by a capital letter and its components by the lowercase.

An important measure for the complexity of a range space is its VC-dimension: A range space $(\mathcal{X}, \mathfrak{R})$ has *VC-dimension* h if there exists a subset $X \subset \mathcal{X}$ of maximal cardinality h such that $\{\mathcal{R} \cap X \mid \mathcal{R} \in \mathfrak{R}\}$ equals the power-set of X [11]. For example, consider the range space of all half-spaces in $\mathbb{R}^2$: There exists a 3-point set $\mathcal{P}$ in $\mathbb{R}^2$ such that any subset of $\mathcal{P}$ can be generated by cutting $\mathcal{P}$ with a half-space. This is not possible for any 4-point set in $\mathbb{R}^2$. Hence, the VC-dimension of the half-space range space in $\mathbb{R}^2$ is 3.

Range spaces with small VC-dimension have the property that for any set $\mathcal{D} \subset \mathcal{X}$ there exist a small subset that is sensitive to large ranges. Such sets

are called epsilon-nets: For a finite n-point set $\mathcal{D} \subset \mathcal{X}$ of a range space $(\mathcal{X}, \mathfrak{R})$, a subset $\mathcal{N}$ of $\mathcal{D}$ is called *epsilon-net* for a parameter $\varepsilon \leq 1/2$ if for any range $\mathcal{R} \in \mathfrak{R}$ with $|\mathcal{R} \cap \mathcal{D}| \geq \varepsilon n$ there is at least one point in $\mathcal{R} \cap \mathcal{N}$. Epsilon-nets were introduced to computational geometry in an influential paper by Haussler and Welzl [11], who show that epsilon-nets of small size exist for range space of small VC-dimension:

Lemma 1 (Epsilon-net Lemma [20])**.** *Let $(\mathcal{X}, \mathfrak{R})$ be a range space with finite VC-dimension $h \geq 2$. For an absolute constant c, a parameter $\varepsilon \leq 1/2$ and an n-point subset $\mathcal{D}$ of $\mathcal{X}$, there exists an epsilon-net $\mathcal{N}$ of $\mathcal{D}$ for $(\mathcal{X}, \mathfrak{R})$ of size at most $(ch/\varepsilon)\log(1/\varepsilon)$.*

For many natural ranges, such as half-spaces, balls, and ranges defined by small-degree polynomials, the VC-dimension is usually small. Our approach uses properties of epsilon-nets that get emphasized by the following reformulation of their definition: A subset $\mathcal{N} \subset \mathcal{D}$ is an epsilon-net for a range space $(\mathcal{X}, \mathfrak{R})$, if for any range $\mathcal{R} \in \mathfrak{R}$ with $\mathcal{R} \cap \mathcal{N} = \emptyset$, the cardinality of $\mathcal{R} \cap \mathcal{D}$ is bounded by εn.

The *Range Searching Problem* for a range space $(\mathcal{X}, \mathfrak{R})$ and an n-point database $\mathcal{D} \subset \mathcal{X}$ can be formulated as follows: Preprocess $\mathcal{D}$ such that for a query range $\mathcal{R} \in \mathfrak{R}$ one can either efficiently report or count all points in $\mathcal{D} \cap \mathcal{R}$. We refer to these variants as the *Range Reporting Problem* and *Range Counting Problem* respectively. The Range Searching Problem for the range space $(\mathbb{R}^d, \mathfrak{H})$, in which the ranges are axes-parallel hyper-rectangles, is important in theory and practice [8, 10], thus one usually refers to it as the *Orthogonal Range Searching Problem*. Range queries that are unbounded in some dimensions are usually called *partial range queries* [10].

NN Search. We now show how to combine epsilon-nets and range reporting to answer PNN queries. We first explain the algorithm for the special case of the NN Problem in Euclidean space, and then adapt the algorithm for the PNN Problem with other metrics.

On an intuitive basis, for a NN query $Q \in \mathbb{R}^d$ we find a nearest neighbor if we can report and check all points in the d-dimensional Euclidean ball $\mathcal{B}^d_{Q,r}$ that is centered at Q and has an appropriate radius r. The problem in this procedure is not the range reporting, since efficient algorithms exist, but the determination of a suitable radius r. On the one hand, if r is too large the performance of the procedure may degenerate to a linear scan. On the other hand, if r is too small, there is no point in the intersection and hence no nearest neighbor is found. The idea to solve this problem is to generate an epsilon-net $\mathcal{N}$ on $\mathcal{D}$ and use the distance of a nearest neighbor in $\mathcal{N}$ to bound r.

We start with the definition of an appropriate range space. A d-dimensional Euclidean ball is defined as $\mathcal{B}^d_{Q,r} = \left\{X \in \mathbb{R}^d \mid \mathrm{d}(X, Q) < r\right\}$ and the set of all Euclidean balls, denoted by $\mathfrak{B}$, equals $\left\{\mathcal{B}^d_{Q,r} \mid Q \in \mathbb{R}^d, r \geq 0\right\}$. For the Euclidean distance the range space $(\mathbb{R}^d, \mathfrak{B})$ is *suitable* with respect to NN search. Note that it is only a technical detail to define the Euclidean balls to be open,

which is clarified later. The following scheme is the basis for all subsequent algorithms.

Preprocessing: First, we build a range searching data structure for $(\mathbb{R}^d, \mathfrak{B})$ on $\mathcal{D}$. Secondly, for the range space $(\mathbb{R}^d, \mathfrak{B})$ and a parameter $\varepsilon > 0$ we generate an epsilon-net $\mathcal{N}$ on $\mathcal{D}$. This can be done by random sampling [11, 20] or deterministically as described in [6]. The latter algorithm has a running time exponential in the VC-dimension, hence we stick to random sampling which comes at the cost of obtaining Las-Vegas algorithms. The overall preprocessing time is dominated by the preprocessing of the range searching data structure.

Query algorithm: When processing a query $Q \in \mathbb{R}^d$, we compare each point of $\mathcal{N}$ with Q and obtain a nearest neighbor N of the epsilon-net. We set the radius $r = \mathrm{d}(Q, N)$ and ask the range searching data structure to report all points in $\mathcal{B}^d_{Q,r} \cap \mathcal{D}$. These points are finally compared with Q to obtain a nearest neighbor in $\mathcal{D}$. If no points are in this range, then N is not only a nearest neighbor of the epsilon-net, but also of $\mathcal{D}$.

We require an efficient range searching data structure for the range space $(\mathbb{R}^d, \mathfrak{B})$. Using a standard lifting transform [8], in which all points are lifted to the $(d+1)$-dimensional paraboloid, the ball range searching problem transforms to the half-space range searching problem [18]. The most efficient range searching data structure for this problem is presented by Chan [7] and for linear space it achieves a query time of $\mathcal{O}\left(\mathrm{g}(d) n^{1-1/\lfloor d/2 \rfloor}\right)$ w.h.p.. As an intermediate result we obtain a Lemma for the NN Problem.

Lemma 2. *Given an n-point set $\mathcal{D} \subset \mathbb{R}^d$ a nearest neighbor of a query $Q \in \mathbb{R}^d$ can be found in time $\mathcal{O}\left(\mathrm{g}(d) n^{1-1/\lceil d/2 \rceil}\right)$ w.h.p., using $\mathcal{O}(dn)$ space.*

Proof. We show the correctness, give bounds on the query time and the space requirements.

Correctness. In the first step of the query algorithm, the epsilon-net is searched for a closest point N to Q. If N is not a nearest neighbor, but a different point P' is, it holds that $\mathrm{d}(P', Q) < \mathrm{d}(N, Q) = r$, which, by definition of $\mathcal{B}^d_{Q,r}$, is in $\mathcal{B}^d_{Q,r} \cap D$. Since the algorithm reports and checks all points in this set, it will recognize P' as the nearest neighbor. On the other hand, if N is the nearest neighbor, no point is contained in $\mathcal{B}_{Q,r} \cap \mathcal{D}$, and thus the algorithm outputs N.

Query time. Let us set $\varepsilon = 1/\sqrt{n}$. We first search the epsilon-net with brute force, which by Lemma 1 takes time $\mathcal{O}(hc/\varepsilon \log 1/\varepsilon)$. The VC-dimension h is linear in d for the range space $(\mathbb{R}^d, \mathfrak{B})$ [20] and the variable c is an absolute constant. This results in a time complexity of $\mathcal{O}(d\sqrt{n} \log n)$.

Secondly, we query $\mathcal{B}^d_{Q,r}$ using the range reporting data structure for $(\mathbb{R}^d, \mathfrak{B})$. As we stated above, using a transform to $(d+1)$-dimensional space and Chan's half-space range searching data structure, we need $\mathcal{O}\left(\mathrm{g}(d) n^{1-1/\lceil d/2 \rceil} + \mathrm{g}(d) m\right)$ time w.h.p. to query m points in $\mathcal{B}^d_{Q,r} \cap \mathcal{D}$. Finally, we search the m points with brute force in time $\mathcal{O}(d\varepsilon n)$, which is due to the following fact: By the definition of the ranges as *open* balls and by setting the radius to $r = \mathrm{d}(Q, N)$, we know that $\mathcal{B}^d_{Q,r} \cap \mathcal{D}$ contains no point of $\mathcal{N}$. By the definition of epsilon-nets

it directly follows that the size of $\mathcal{B}^d_{Q,r} \cap \mathcal{D}$ is at most $\mathcal{O}(\varepsilon n) = \mathcal{O}(\sqrt{n})$. So, the epsilon-net guarantees that the radius r is not too large. This results in an overall query time is $\mathcal{O}\left(d\sqrt{n}\log\sqrt{n} + \mathrm{g}(d)n^{1-1/\lceil d/2\rceil} + \mathrm{g}(d)\sqrt{n}\right)$ w.h.p..

Space requirements. The space requirements depend only on the space requirements of the range searching data structure since the epsilon-net has size smaller than n. Thus the overall space requirement is linear in n. □

For linear space and $d > 3$, our result already attains the best known query time for NN search [7, 18]. The dependence of d should roughly be of the same order since the same techniques are used.

PNN Search. The principle feature of our approach is that it is extensible to the PNN problem. The distance between two points is now measured with respect to the subspace S as defined by d_S. Instead of querying ranges that have the form of balls, we query ranges that have the form $\mathcal{C}_{Q,S,r} = \left\{X \in \mathbb{R}^d \mid \mathrm{d}_S(Q,X) < r\right\}$. Their projection onto S equals $\mathcal{B}^w_{Q,S}$, so we may refer to $\mathcal{C}_{Q,S,r}$ as a *partially defined ball*. For example when $d = 3$ and the dimension of S is $w = 2$, the range corresponds to a cylinder of radius r. Let us define the range space $(\mathbb{R}^d, \mathfrak{C})$ for which $\mathfrak{C} = \left\{\mathcal{C}_{Q,S,r} \mid Q \in \mathbb{R}^d, S \in \{0,1\}^d, r \geq 0\right\}$.

In order to build a range searching data structure for $(\mathbb{R}^d, \mathfrak{C})$ we require an efficient description of the ranges in $\mathfrak{C}$. Though, there are exponentially many subspaces, the ranges in $\mathfrak{C}$ can be described by a single polynomial of the form $\mathrm{f}(x_1,\ldots,x_d,q_1,\ldots,q_d,s_1,\ldots,s_d,r) = \sum_{i=1}^d s_i(q_i - x_i)^2 - r^2 \leq 0$. The corresponding range space has VC-dimension of $\mathcal{O}\left(d^2\right)$ (see [20], Proposition 10.3.2). A range searching data structure that queries the desired ranges is described by Agarwal and Matoušek [1]. We restate a simplified version of this result to show the flexibility of our approach. Combining with the latest results on Tarski-Cell decompositions [14] leads to the following lemma:

Lemma 3 ([1, 14])**.** *Let* $\mathrm{f}(x_1,\ldots,x_d,a_1,\ldots,a_p)$ *be a* $(d+p)$*-variate polynomial where* d, p*, and the degree of* f *is bounded. Let* $(\mathbb{R}^d, \mathfrak{R})$ *be a range space with* $\mathfrak{R} = \{\mathcal{R}_{A_1,\ldots,A_u} \mid A_1,\ldots,A_u \in \mathbb{R}^u\}$*, for a fixed constant* u*, and* $\mathcal{R}_{A_1,\ldots,A_u} = \left\{X \in \mathbb{R}^d \mid \mathrm{f}(X,A_1) \leq 0,\ldots,\mathrm{f}(X,A_u) \leq 0\right\}$*. Then, the range searching problem for* $(\mathbb{R}^d, \mathfrak{R})$ *can be solved with* $\mathcal{O}(n)$ *space, and* $\mathcal{O}\left(n^{1-1/b+\delta}\right)$ *query time for* $\delta > 0$*. The parameter* $b = d$ *for* $d \leq 3$ *and* $b = 2d-4$ *for* $d > 3$*.*

This Lemma also allows the construction of a PNN data structure for ℓ_p-metrics with fixed p. We only need to adapt the polynomial f. Note that for increasing p the degree of the polynomial and thus the VC-dimension increases, and consequently, the query time of the range searching algorithm increases. On the other hand, in the case the ℓ_∞-metric the induced partially defined balls are hyper-rectangles, thus, a KD Tree [8] is a suitable structure. The advantage of our algorithm is that it can handle different range searching data structures. Hence, we formulate our main result on the PNN problem in a generic way, more precisely, for a generic metric that has suitable range space $\mathfrak{S}$ of VC-dimension h. For a range searching data structure build on $\mathcal{D}$ which uses $\mathrm{s}_{\mathfrak{S}}(n)$ space, let

$\mathrm{t}^*_{\mathfrak{S}}(n)$ be the preprocessing time and let $\mathrm{t}_{\mathfrak{S}}(n,m)$ be the time it takes to report $m = |\mathcal{D} \cap \mathcal{R}|$ points in the intersection of a query range $\mathcal{R}$.

Theorem 1. *Let d_S be a distance function and $\mathfrak{S}$ a suitable range space of VC-dimension h. Then, an n-point set $\mathcal{D} \subset \mathbb{R}^d$ can be preprocessed in time $\mathrm{t}^*_{\mathfrak{S}}(n)$ into a data structure of $\mathrm{s}_{\mathfrak{S}}$ space, such that a PNN query Q, S can be answered in time $\mathcal{O}\left(h\sqrt{n}\log n + \mathrm{t}_{\mathfrak{S}}(n,\sqrt{n})\right)$ w.h.p..*

Proof. This proof uses essentially the same arguments as the proof of Lemma 2, so we keep it small and only highlight the differences.

Preprocessing time. We generate the ε-net in $\mathcal{O}(n)$ time by random sampling, thus the given guarantees hold w.h.p.. The overall preprocessing time is dominated by the preprocessing time of the range searching data structure $\mathrm{t}^*(n)$.

Query time. For the epsilon-net we choose the parameter $\epsilon = 1/\sqrt{n}$. As in Lemma 2, the time to search the epsilon-net with brute force is $\mathcal{O}(h\sqrt{n}\log n)$.

For the range space $\mathfrak{S}$ we constructed the range searching data structure that queries m points in time $\mathrm{t}_{\mathfrak{S}}(n,m)$ and which has space requirements of $\mathrm{s}_{\mathfrak{S}}(n)$. The epsilon-net guarantees w.h.p. that any range queried in the proposed algorithm is at most of size $m = \mathcal{O}(\varepsilon n) = \mathcal{O}(\sqrt{n})$ (see proof of Lemma 2). So, querying the range and searching the resulting set with brute force takes time $\mathcal{O}\left(\mathrm{t}_{\mathfrak{S}}(n,\sqrt{n}) + d\sqrt{n}\right)$ w.h.p.. Together with the time needed to search the ε-net we obtain the stated result.

Space requirements. Since the epsilon-net has size less than dn, the range searching data structure dominates the space requirements, which are $\mathrm{s}_{\mathfrak{S}}(n)$. □

Theorem 1 and the data structure from Lemma 3 directly imply an algorithm for the Euclidean PNN problem. The VC-dimension of the range space $(\mathbb{R}^d, \mathfrak{C})$ is at most $\mathcal{O}\left(d^2\right)$ (see [20], Proposition 10.3.2).

Lemma 4. *An n-point set $\mathcal{D} \subset \mathbb{R}^d$ from Euclidean space can be preprocessed in time $\mathcal{O}(\mathrm{g}(d)n\log n)$ into a $\mathcal{O}(dn)$ space data structure, such that an PNN query Q, S can be answered in time $\mathcal{O}\left(\mathrm{g}(d)n^{1-1/b+\delta}\right)$ w.h.p., for $\delta > 0$ and for $b = d$ if $d \leq 4$ and $b = (2d-4)$ otherwise.*

Opposed to the brute force approach, with $\mathcal{O}\left(2^d n\right)$ space, $\mathcal{O}(\mathrm{g}(d)n\log n)$ preprocessing time and $\mathcal{O}\left(\mathrm{g}(d)n^{1-1/\lceil d/2\rceil}\right)$ query time, we obtain a worse query time, however, we save space exponentially in d.

As we stated above, the ℓ_∞-metric uses the range space $(\mathbb{R}^d, \mathfrak{H})$, that has VC-dimension $2d$. There are reasonable fast and well-studied data structures for $(\mathbb{R}^d, \mathfrak{H})$ such as KD Trees [8], Range Trees [8]. If we use a KD Tree for range searching, which reports m points in a partially defined range in time $\mathrm{t}_{(\mathbb{R}^d,\mathfrak{H})}(n,m) = \mathcal{O}\left(dn^{1-1/d} + dm\right)$ [16], and apply Theorem 1 we obtain the following Lemma.

Lemma 5. *Given an n-point set $\mathcal{D} \subset \mathbb{R}^d$ together with the ℓ_∞-metric. There is a data structure of $\mathcal{O}(dn)$ space, which can be constructed in $\mathcal{O}(dn\log n)$ time, that answers a PNN query Q, S in time $\mathcal{O}\left(dn^{1-1/d}\right)$ w.h.p..*

Approximate PNN Search. The performance of the range searching algorithm for Euclidean distance (Lemma 3) might not be satisfying for practical applications. Even with more efficient Tarski-cell decompositions, which are the current limiting factor in the range searching data structure, the worst-case time cannot be better than $\mathcal{O}\left(\mathrm{g}(d)n^{1-1/d}\right)$ [22]. Consequently, it might be worth to consider approximation algorithms that work well on real world data. We present a simple and practically efficient $\sqrt{w}$-approximation algorithm that queries partially specified axes-parallel hyper-rectangles instead of $\mathcal{C}_{Q,S,r}$. Better approximation ratios require the study of data structures for approximate partial ball range reporting.

For simplicity, we describe a specific unbounded hyper-rectangle $\mathcal{H}_{Q,S,r}$ with center Q and radius r that is equal to $\mathcal{H}_{L,R}$ with $l_i = q_i - r$ ($r_i = q_i + r$ resp.) if $s_i = 1$ and $l_i = -\infty$ ($r_i = \infty$ resp.) otherwise. As above, an epsilon-net for the range space $(\mathbb{R}^d, \mathfrak{C})$ is denoted by $\mathcal{N}$ and the point of the epsilon-net closest to Q with respect to S is denoted by $N \in \mathcal{N}$. Let $r = \mathrm{d}_S(Q, N)$ be the radius of the desired range $\mathcal{C}_{Q,S,r}$.

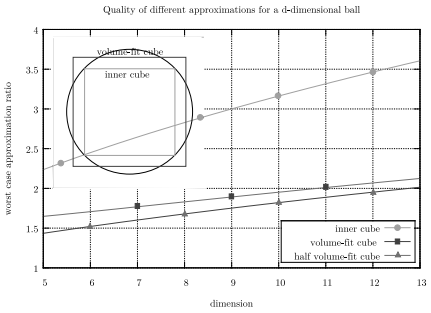

Figure 1: Left: Worst-case approximation ratios for some ranges that approximate the Euclidean ball.

The smallest enclosing rectangle $\mathcal{H}_{Q,S,r}$, called *outer cube range*, is not suitable as a query range since its volume grows exponentially faster than the volume of $\mathcal{C}_{Q,S,r}$. Thus, we use the hyper-rectangle $\mathcal{H}_{Q,S,r/\sqrt{w}}$ of maximal size that completely fits inside the desired range $\mathcal{C}_{Q,S,r}$. Let us call this range *inner cube*. The resulting point size is bounded by the epsilon-net, however, we can only get an $\sqrt{w}$-approximation as the subsequent lemma states.

Lemma 6. *Given a point N of an n-point set $\mathcal{D}$, a query point Q with w-dimensional query subspace S, and a radius $r = \mathrm{d}_S(Q, N)$. If the hyper-rectangle $\mathcal{H}_{Q,S,r/\sqrt{w}}$ is queried, then the returned point is a $\sqrt{w}$-approximate PNN of $\mathcal{D}$.*

Proof. If $\mathcal{H}_{Q,S,r/\sqrt{w}} \cap \mathcal{D}$ is not empty, all reported points are closer to Q than N, so one of them is an exact PNN. Assume $\mathcal{H}_{Q,S,r/\sqrt{w}} \cap \mathcal{D}$ is empty. Let P be a PNN in $\mathcal{C}_{Q,S,r} \setminus \mathcal{H}_{Q,S,r/\sqrt{w}} \cap \mathcal{D}$ with minimal distance to Q. This distance is at least $\mathrm{d}_S(P, Q) > r/\sqrt{w}$ as otherwise $P \in \mathcal{H}_{Q,S,r/\sqrt{w}}$. The point N is now a $\sqrt{w}$-approximate PNN, because

$$\frac{\mathrm{d}_S(N, Q)}{\mathrm{d}_S(P, Q)} < \frac{r}{r/\sqrt{w}} = \sqrt{w}.$$

□

As a consequence we can obtain a $\sqrt{w}$-approximation data structures for the Euclidean PNN Problem.

Lemma 7. *Given an n-point set $\mathcal{D} \subset \mathbb{R}^d$ from Euclidean space and suppose we use a KD Tree as range searching data structure. For a PNN query Q, S we obtain a $\sqrt{w}$-approximation in time $\mathcal{O}\left(dn^{1-1/d}\right)$ w.h.p., consuming $\mathcal{O}(dn)$ space and using $\mathcal{O}(dn \log n)$ preprocessing time.*

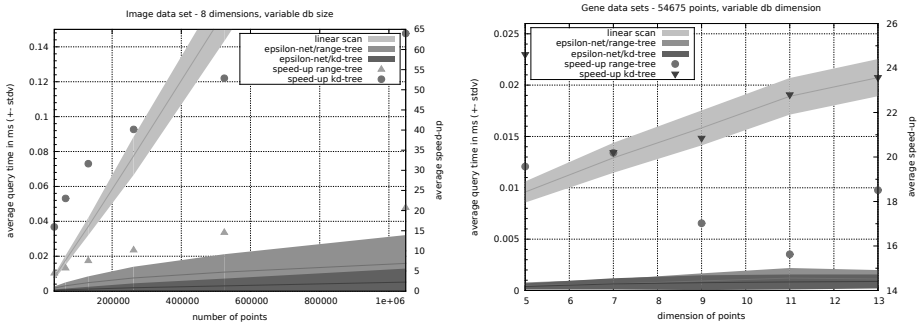

Figure 2: The average over 10000 queries is shown as straight lines within filled area that represents the standard deviation. The speed-up is shown as bullets and triangles with axes on the right. Left: The query time depending on the database size. Right: The query time depending on the database dimension.

It is also possible to use different, application adapted range searching data structures, as for example the Priority R-tree [4] (or PR-tree) if the goal is an IO-optimal and practically efficient algorithm for memory hierarchies.

4 Experiments

We implemented the proposed APNN algorithm for the Euclidean distance using KD Tree and Range Tree data structure to show that the theoretical foundations are valid and the concept performs well on a wide range of real world data. The performance results also indicate good behavior for the exact PNN data structure using the ℓ_∞-metric, since the same range searching data structures can be used. The implementation was done using C++ in a standard way [8] without any optimizations to exploit architecture related specialties. The PNN query points were generated by randomly perturbing points from the database and, unless stated otherwise, the query subspace $S \subset \{0,1\}^d$ was generated by setting each component to 1 with probability 1/2. For each experiment we performed 10000 queries to get reliable results.

For the experimental evaluation we used data sets from two completely different domains to emphasize the data independence of our approach. The first data set comes from a gene database and contains feature vectors that represent the genes by their activity in multiple experimental settings and were all extracted from the Genevestigator database [12]. A feature represents an anatomical class (category) such as a certain type of neoplasmic cells or cells

#	n	d	$\lvert\mathcal{N}\rvert$	$\lvert\mathcal{C}\cap\mathcal{D}\rvert$ (avg)	(stdv)	(max)
4	2^{14}	8	2^{10}	14.5 (0.09%)	14.7	143 (0.87%)
4	2^{15}	8	2^{10}	30.4 (0.09%)	31.6	284 (0.87%)
4	2^{16}	8	2^{10}	57.8 (0.09%)	59.3	576 (0.88%)
4	2^{17}	8	2^{10}	118.0 (0.09%)	116.2	1181 (0.90%)
4	2^{18}	8	2^{10}	246.8 (0.09%)	250.7	2388 (0.91%)
4	2^{19}	8	2^{10}	515.4 (0.10%)	509.3	5487 (1.05%)
4	2^{20}	8	2^{10}	1074.2 (0.10%)	1055.3	8735 (0.83%)
1	54675	5	2^{10}	51.4 (0.09%)	52.6	517 (0.95%)
2	54675	7	2^{10}	47.9 (0.09%)	48.5	535 (0.98%)
3	54675	9	2^{10}	51.7 (0.09%)	53.5	521 (0.95%)
3	54675	11	2^{10}	56.7 (0.10%)	54.7	555 (1.02%)
3	54675	13	2^{10}	46.9 (0.09%)	48.2	570 (1.04%)

Table 1: This table summarizes the dependence of $|\mathcal{C}_{Q,S,r}\cap\mathcal{D}|$ for various experimental settings (columns from left: data set number, number of points, number of dimensions, epsilon-net size).

from different parts of the human brain. An element p_i of the feature vector representing gene $P\in\mathcal{D}$ corresponds to the gene's activity in a cell of class i. The activity is a floating point number in $[0,100]$. We consider three data sets show in Table 2. We want to emphasize the practical relevance of the query type we use: it is the same type as widely used in the data mining application of Genevestigator.

The second data set comes from an image collection crawled by Standford Universities WebBase project [3, 23]. We extracted an 8-dimensional (data set #4) and a 64-dimensional (data set #5) color histogram of $n=2^{20}$ images such that each image is represented by its color intensities. This type of data was recently used in a study dedicated to the NN problem [17] and it is also well suitable for PNN queries: a PNN query asks for an image that is similar to the query image with respect to a subset of colors. The coordinates of a point are floating point numbers from the interval $[0,100]$.

We use randomly sampled ε-nets to study their failure probability. Our experiments indicate that small epsilon-nets bound the number of points in a query range extremely well. The average and standard deviation (see Table 1) of the number of points in the query range is very small. Not only the average, but also the maximal number of points behaves well and is roughly what Lemma 1 predicts. We observe that the dependence on the number of points n and dimension d is nearly linear. One could roughly state that if we used 1000 points for the epsilon-net over 10000 queries, then a range cuts roughly a $1/1000$-fraction of points on the average (Table 1).

There are several heuristics that naturally stem from our approach, which have guarantees on the approximation ratio but not on the query time. Nevertheless, our experimental evaluation indicates that the worst-case query time does not occur in practice. A heuristic to approximate $\mathcal{C}_{Q,S,r}$, denoted by *volume-fit cube*, corresponds to a hyper-rectangle that is centered at Q and its projection onto S has the same volume as a w-dimensional Euclidean ball $\mathcal{B}_w(r)$. Thus, the edge length of the hyper-rectangle equals $\mathrm{vol}(\mathcal{B}_w(r))^{1/w}$. We obtain the hyper-rectangle $\mathcal{H}_{L,R}$ with $l_i = q_i - r\pi^{1/2}/(2\Gamma(w/2+1)^{1/w})$ and

#	n	d	range	ar (avg)	(stdv)	(max)	rank
4	2^{20}	8	inner cube	1.08	0.15	2.30	153.67
5	2^{20}	64	inner cube	1.01	0.29	5.66	87.50
4	2^{20}	8	volume-fit	1.02	0.05	1.45	81.72
5	2^{20}	64	volume-fit	1.02	0.09	2.74	54.80

Table 2: Approximation ratio for the KD Tree implementation using the image data sets (columns from left: data set number, number of points, number of dimensions, approximation ratio average, standard deviation, maximum, and average rank).

$r_i = q_i + r\pi^{1/2}/(2\Gamma(w/2+1)^{1/w})$ for the relevant dimensions. This hyper-rectangle exceeds the boundaries of $\mathcal{B}_w(r)$, and thus the number of reported points is no longer bounded by the epsilon-net, so the query time might degenerate to a linear scan. The approximation ratio is $2\Gamma(w/2+1)/\pi^{1/2}$ (see Figure 1) and is much smaller than $\sqrt{w}$ as the dimension grows. This heuristic performs markedly as Table 2 and Figure 3 indicate.

Evaluation. We show results for the algorithms' performance in terms of query time, space requirements, approximation quality and rank of the answer. The query time is compared with a linear scan and we define the speed-up to be the ratio of the average query time of a linear scan and the average query time of our implementation.

The dependence of the query time on the size of the data-base is presented in Figure 2. We observe a sub-linear dependence of the query time, and the algorithm performs better if a KD tree is used for range searching. A factor of over 60 times faster than a linear scan gives strong evidence for the algorithms practical usability. The results were obtained for an approximation with the inner cube range that has a theoretical bound on the approximation ratio of 2.8, however, the measured average approximation ratio was way less, roughly around 1.08 (see Table 2).

The dependence of the database dimension on the query time is shown in Figure 2. The ranges were approximated with the inner cube range and we observe that the dimensionality of the data set has only a negligible impact on the query time, at least for dimensions up to 13.

In Figure 3 and Table 2 we present the measured approximation quality in terms of approximation ratio and rank of the answer. The *rank* is the number of points that are closer to the query than the point found. Figure 3 shows that the obtained approximation ratios are reasonable small. The maximal rank is less than than 0.025 percent of the database.

We compared our approach with the standard query algorithm for nearest neighbor search on the KD tree. The standard query algorithm traverses the tree from the root to a leaf into the direction of the query point. On this root-to-leaf path a point closest to the query is chosen as a candidate nearest neighbor, and subsequently, the algorithm unwinds the recursion and checks nodes "in the vicinity" for a better candidate [8]. To the best of our knowlegde the standard algorithm was solely considered for the PNN problem in a paper by Eastman

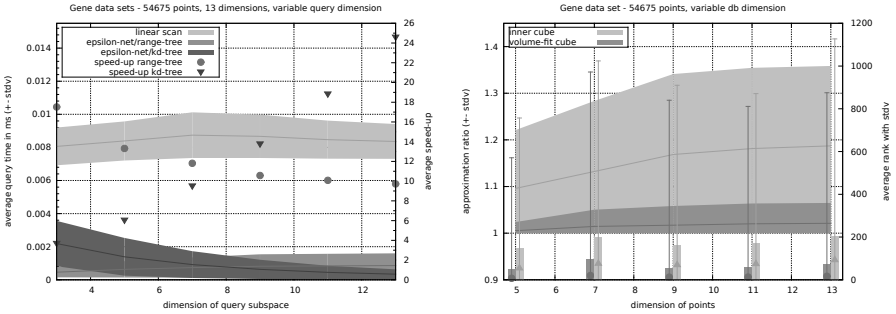

Figure 3: The description of this figure is the same as of Figure 2. Left: Query time depending on the dimension of the query subspace. Right: The quality of the approximation (average over 10k queries: straight lines; standard deviation: filled area) and rank of the answer point (average over 10k queries: box; standard deviation: whisker).

et al. [9], who analyzed its performance for nicely distributed data points under the ℓ_∞-metric. Our approach yields easy-to-derive guarantees on the query time and space requirements for any set of data points. The experiments show that our algorithm is at least as fast as the standard algorithm, hence, the reduction to range searching does not cause any overhead.

We also studied the behavior of our algorithm for data with medium dimensionality between 10^1 and 10^2. The 64-dimensional data set #5 using the KD Tree as range searching data structure has a query time that is still a factor of 5 to 10 faster than the linear scan. The theoretical observation would suggest that the approximation ratio is fairly large for this dimension, however, as Table 2 shows, the approximation ratio and rank are only slightly worse than for the low dimensional case.

References

[1] P. Agarwal and J. Matoušek. On Range Searching with Semialgebraic Sets. *Discrete and Computational Geometry*, 11(1):393–418, 1994.

[2] A. Andoni, P. Indyk, R. Krauthgamer, and H. L. Nguyen. Approximate Line Nearest Neighbor in High Dimensions. In *SODA'09: Proceedings of the 20th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 293–301. ACM, 2009.

[3] A. Arasu, J. Cho, H. Garcia-Molina, A. Paepcke, and S. Raghavan. Searching the Web. Technical Report 2000-37, Stanford InfoLab, 2000.

[4] L. Arge, M. D. Berg, H. Haverkort, and K. Yi. The priority R-tree: A practically efficient and worst-case optimal R-tree. *ACM Trans. Algorithms*, 4:9:1–9:30, March 2008.

[5] T. Bernecker, T. Emrich, F. Graf, H. Kriegel, P. Kröger, M. Renz, E. Schubert, and A. Zimek. Subspace Similarity Search: Efficient k-NN Queries in Arbitrary Subspaces. In *SSDBM'10: Scientific and Statistical Database Management*, pages 555–564. Springer, 2010.

[6] H. Brönnimann, B. Chazelle, and J. Matoušek. Product Range Spaces, Sensitive Sampling, and Derandomization. *SIAM Journal on Computing*, 28(5):1575, 1999.

[7] T. M. Chan. Optimal Partition Trees. In *SCG'10: Proceedings of the 2010 Annual Symposium on Computational Geometry*, pages 1–10. ACM, 2010.

[8] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry - Algorithms and Applications*. Springer, 2nd edition, 2000.

[9] C. M. Eastman and M. Zemankova. Partially Specified Nearest Neighbor Searches Using k-d Trees. *Information Processing Letter*, 15(2):53–56, 1982.

[10] J. E. Goodman and J. O'Rourke, editors. *Handbook of Discrete and Computational Geometry*. CRC Press, 2 edition, 2004.

[11] D. Haussler and E. Welzl. Epsilon-nets and Simplex Range Queries. In *SCG'86: Proceedings of the 2nd Annual Symposium on Computational Geometry*, page 71. ACM, 1986.

[12] T. Hruz, M. Wyss, et al. RefGenes: identification of reliable and condition specific reference genes for RT-qPCR data normalization. *BMC genomics*, 12(1):156, 2011.

[13] P. Indyk and R. Motwani. Approximate Nearest Neighbor: Towards Removing the Curse of Dimensionality. In *STOC'98: Proceedings of the 30th Annual ACM Symposioum on Theory of Computing*, pages 604–613. ACM, 1998.

[14] V. Koltun. Almost Tight Upper Bounds for Vertical Decompositions in Four Dimensions. In *FOCS'01: Proceedings of the 42nd Annual IEEE Symposium on Foundations of Computer Science*, pages 56–65. IEEE, 2001.

[15] H. Kriegel, P. Kroger, M. Schubert, and Z. Zhu. Efficient Query Processing in Arbitrary Subspaces Using Vector Approximations. In *SSDBM'06: Proceedings of the 18th International Conference on Scientific and Statistical Database Management*, pages 184–190, 2006.

[16] D. T. Lee and C. Wong. Worst-case Analysis for Region and Partial Region Searches in Multidimensional Binary Search Trees and Balanced Quad Trees. *Acta Informatica*, 9(1):23–29, 1977.

[17] Q. Lv, W. Josephson, Z. Wang, M. Charikar, and K. Li. Multi-probe LSH: Efficient Indexing for High-Dimensional Similarity Search. In *VLDB'07: Proceedings of the 33rd International Conference on Very Rarge Data Bases*, pages 950–961, 2007.

[18] J. Matoušek. Reporting Points in Halfspaces. *Computational Geometry*, 2:169–186, 1992.

[19] J. Matoušek. On Constants for Cuttings in the Plane. *Discrete and Computational Geometry*, 20(4):427–448, 1998.

[20] J. Matoušek. *Lecture Notes on Discrete Geometry*. Sp, 2002.

[21] H. Samet. *Foundations of Multidimensional and Metric Data Structures*. Morgan Kaufmann Publishers Inc., 2005.

[22] M. Sharir and H. Shaul. Ray Shooting Amid Balls, Farthest Point from a Line, and Range Emptiness Searching. In *SODA'05: Proceedings of the 16th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 525–534, 2005.

[23] Stanford WebBase Project. `http://diglib.stanford.edu:8091/~testbed/doc2/WebBase/`.

[24] P. Zimmermann, O. Laule, J. Schmitz, T. Hruz, S. Bleuler, and W. Gruissem. Genevestigator transcriptome meta-analysis and biomarker search using rice and barley gene expression databases. *Molecular plant*, 1(5):851, 2008.