A Simple Framework for the Generalized Nearest Neighbor Problem

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

Abstract

The problem of finding a nearest neighbor from a set of points in $\mathbb{R}^d$ to a complex query object has attracted considerable attention due to various applications in computational geometry, bio-informatics, information retrieval, etc. We propose a generic method that solves the problem for various classes of query objects and distance functions in a unified way. Moreover, for linear space requirements the method simplifies the known approach based on ray-shooting in the lower envelope of an arrangement.

1 Introduction

During the last decades the nearest neighbor problem and its variants have attracted considerable attention in computational geometry, information retrieval, pattern recognition, bio-informatics, and many more areas of computer science. In its *classical version*, the problem is stated as follows: Given an n-point set $P \subset \mathbb{R}^d$ together with a metric distance function $\mathcal{D} : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}^+$, preprocess P such that for a query point $\rho \in \mathbb{R}^d$ we can efficiently find a point $\pi \in P$ with $\mathcal{D}(\pi, \rho) \leq \mathcal{D}(\pi', \rho)$ for all $\pi' \in P$. The point π is usually called a *closest point* or a *nearest neighbor* of the query point. There are situations in which we want the query to be an object more complex than a point, as for example a line or simplex in $\mathbb{R}^d$. Consequently, we generalize the notion of a query and define a *query object* Q to be a subset of $\mathbb{R}^d$ that has a constant description complexity. Furthermore, we denote the collection of all allowed query objects by Θ. As an example, consider Θ as the set of all lines in $\mathbb{R}^3$ where a query object $Q \in \Theta$ is the point set of a line. A natural generalization of the problem statement is as follows. Preprocess P such that for a query object $Q \in \Theta$ we can efficiently find a point π in P with $\mathcal{D}(\pi, Q) \leq \mathcal{D}(\pi', Q)$ for all $\pi' \in P$, where $\mathcal{D}(\pi, Q) = \min\{\mathcal{D}(\pi, \rho) \mid \rho \in Q\}$ is the distance between the point π and the query object Q. We refer to this version as the *generalized nearest neighbor problem (GNN problem)*. In this paper we consider the dimension d as a fixed constant.

1.1 Our contribution

We present a novel, simple framework that solves the generalized nearest neighbor problem uniformly for many natural query objects by a combination of ε-nets and range searching algorithms. Although the results rely on known concepts, we are not aware of any prior work that brings the following contributions:

- Our solution to the GNN problem uses range searching data structures, but does not rely on Megiddo's parametric search [18] that usually causes a logarithmic overhead. Parametric search is a powerful but complex technique to reduce optimization problems to decision problems which we substitute by an application of ε-nets (see Section 2).
- Applying the framework to three sets of query objects improves previously published results: If the query objects are lines in 2-dimensional Euclidean space [19] we improve the query time by a factor of $\mathcal{O}\left(n^{0.195}\right)$. In the case of query planes in 3-dimensional Euclidean space [20] and in the case of query circles [21] we improve two recent results by reducing the space requirements by a logarithmic factor (see Section 3).
- In the case of query lines in 3-space with Manhattan and Euclidean distance, a problem that has not been considered before, we show that, for a parameter $f > 0$, our framework obtains a query time of $\mathcal{O}\left(n^{2/3+f}\right)$ while using linear space (see Section 3).
- The current state-of-the-art idea to approach the GNN problem, which is referred to as generalized Voronoi diagrams, exploits duality properties of the distance function. In the case of linear space requirements our framework simplifies this approach by avoiding any consideration of duality (see Section 4). Furthermore, as opposed to the concept provided by Voronoi diagrams, we give easy to derive bounds on query time and space requirements (see Section 2).

Furthermore, our method is intuitive, natural and could also be applied for the classical nearest neighbor problem. From a practical point of view, the method can use range searching data structures as a black-box.

1.2 Intuitive idea of our approach

When designing a data structure to solve the GNN problem we have a trade-off between space and query time that manifests in two extreme cases: Either we want logarithmic query time and accept larger space requirements, or we aim for linear space but accept larger, still sub-linear, query time. It is common and reasonable to focus on these extreme cases, since space/query time trade-offs in between can usually be derived by a combination of the respective data structures [15]. In this paper we focus on linear space data structures.

Our framework relies on the existence of data structures for the range searching problem that can be formulated as follows [8]: Preprocess a set $P \subset \mathbb{R}^d$ of

n points, such that for a given query range $R \subset \mathbb{R}^d$ one can efficiently report all points in $P \cap R$. Now, consider a concrete nearest neighbor problem for a query collection Θ and a distance function $\mathcal{D}$. Let us define the range $\mathcal{B}_r(Q)$ as the set of points within distance r from Q. Observe that this range has the property that for a large enough value of r it contains a nearest neighbor of Q. For example, when the query objects are lines in Euclidean 3-space, for a query Q the range $\mathcal{B}_r(Q)$ is a cylinder with axis Q and radius r (see Section 3). Since known range searching data structures can efficiently report all points in $P \cap \mathcal{B}_r(Q)$, the only problem to identify a nearest neighbor is to find a value r^* such that the nearest neighbor lies in $P \cap \mathcal{B}_{r^*}(Q)$, but not too many other points. This is achieved by identifying a candidate nearest neighbor $\alpha \in P$ such that there are only few points in P closer to the query object Q. The candidate point is found by searching a nearest neighbor in a preprocessed ε-net N of P; the definition and properties of ε-nets are introduced in Section 2. Krauthgamer et al. [12] also use ε-nets to bound search space for proximity search, but their definition describes a completely different notion, which does not provide general worst case guarantees.

Intuitively, the method is correct since either α is a closest point to Q or one of the points in $P \cap \mathcal{B}_{r^*}(Q)$. Both settings are checked by the algorithm. The obtained query time is sub-linear and it depends on the time needed to search the set N and the time needed to report and check all points in $P \cap \mathcal{B}_{r^*}(Q)$. To search N one can either perform a linear scan or recursively apply of the above mentioned procedure. The time for the range searching depends on how much space we accept for the data structure. For many natural query objects there are suitable range searching algorithms with space/query time trade-offs in between the above mentioned two extreme cases. The size of the resulting set $P \cap \mathcal{B}_{r^*}(Q)$ is guaranteed to be at most εn which follows from the properties of the ε-net N (see Section 2). The space requirements of the framework are dominated by the space requirements of the range searching data structure. For details on the algorithm and the analysis we refer the reader to Section 2.

1.3 Related work

Cole and Yap considered the case of 2-dimensional query lines in Euclidean space [7]. They presented an algorithm which preprocesses n points in $\mathcal{O}\left(n^2\right)$ time and space such that a point closest to a query line can be found in $\mathcal{O}\left(\log n\right)$ time. The problem was later reconsidered for space/query time trade-offs: Mitra et al. [19] presented an algorithm with $\mathcal{O}\left(n \log n\right)$ preprocessing time, using $\mathcal{O}\left(n\right)$ space and $\mathcal{O}\left(n^{0.695}\right)$ query time, and Mukhopadhyay [22] provided an algorithm based on the Partition Theorem [15] with preprocessing time in $\mathcal{O}\left(n^{1+f}\right)$, space requirements in $\mathcal{O}\left(n \log n\right)$ and a query time in $\mathcal{O}\left(n^{1/2+f}\right)$ for any $f > 0$.

Another type of query objects, namely planes in Euclidean 3-space, was studied by Mitra et al. [20] who provided an algorithm with $\mathcal{O}\left(n^{1+f}\right)$ preprocessing time, $\mathcal{O}\left(n \log n\right)$ space and $\mathcal{O}\left(n^{2/3+f}\right)$ query time for any $f > 0$. The underlying data structure is also based on Matoušek's partition theorem [15].

For the case in which queries are disc boundaries in 2-dimensional Euclidean space, the authors of [21] provided two algorithms that give space/ query time trade-offs: the first has $\mathcal{O}\left(n^3\right)$ preprocessing time and space, and $\mathcal{O}\left(\log^2 n\right)$ query time, and the second has $\mathcal{O}\left(n^{1+f}\right)$ preprocessing time, $\mathcal{O}\left(n \log n\right)$ space and $\mathcal{O}\left(n^{2/3+f}\right)$ query time.

There has also been a general idea how to approach the GNN problem. This approach is based on a generalization of Voronoi diagrams (see e.g. [1]) and it can be understood as a kind of guide to solve GNN related problems. Roughly speaking, the idea is based on ray-shooting in the lower envelope of an arrangement of surfaces that are induced by the distance function $\mathcal{D}$ and the point set P. An application of this idea to the above problems works analogously to the analysis in Section 3.1 and yield similar improvements as our framework. However, the idea of applying Voronoi diagrams for the GNN problem is a high level concept that does not directly imply any space or query time guarantees. In contrast, our framework provides these guarantees (Theorem 2 and 3) while being less complex in the application (Section 2 and 4).

2 A Unified Framework for the GNN Problem

We formally introduce the main concepts in the theory of range searching and its connection to our approach. A *range space* $\mathfrak{R}$ is a tuple (X, Γ), where X is a set and Γ is a collection of subsets of X. The elements of X are called points and the elements of Γ are called *ranges*. An example for a well-studied range space is $(\mathbb{R}^d, \Gamma_H)$, where Γ_H is the set of all half-spaces. The *range searching problem* for a range space (X, Γ) and a finite point set $P \subseteq X$ can be stated as follows: Preprocess P such that one can efficiently answer the following query. For a range $R \in \Gamma$, report all points in $R \cap P$. This formulation is the so-called reporting version of the problem [3, 6]. There is also the counting version, where one is interested in computing $|R \cap P|$. The *restriction of* Γ *to a set* $Y \subseteq X$, denoted by $\Gamma|_Y$, is the set $\{Y \cap R \mid R \in \Gamma\}$. An important measure for the complexity of a range space is its *VC-dimension*: A range space $\mathfrak{R}$ has VC-dimension z if there exists a subset $Y \subseteq X$ of maximal cardinality z such that $\Gamma|_Y$ equals the power-set of Y [10]. Though range spaces are formulated on a set theoretic level, in this paper we only need the special case for which $X = \mathbb{R}^d$.

As a first step to use range searching, we need to define an appropriate range space $(\mathbb{R}^d, \Gamma)$ for a collection Θ of query objects and a distance function $\mathcal{D}$. The *Minkowski sum* of two sets A and B is $A + B = \{\alpha + \beta \mid \alpha \in A, \beta \in B\}$. By $\mathcal{B}_r(X)$ we denote the *r-neighborhood* of a set $X \subset \mathbb{R}^d$, which equals the open set $X + \left\{\rho \in \mathbb{R}^d \mid \mathcal{D}(\rho, 0) < r\right\} = \left\{\rho \in \mathbb{R}^d \mid \mathcal{D}(\rho, X) < r\right\}$. The r-neighborhood $\mathcal{B}_r(Q)$ of all $Q \in \Theta$ is a natural set for a range in Γ, and consequently, we define the desired range space to be $\mathfrak{R}_\Theta = (\mathbb{R}^d, \{\mathcal{B}_r(Q) \mid Q \in \Theta, r \geq 0\})$. For example, consider the query collection of all lines in Euclidean 3-space. The 1-neighborhood of a line ℓ is the Minkowski sum of ℓ with the Euclidean unit ball centered at the origin. This forms a cylinder of radius 1 with axis ℓ. Our framework depends on the existence of an efficient range searching algorithm for $\mathfrak{R}_\Theta$.

When performing range queries, the resulting sets should not be too large, as otherwise the performance degenerates. We obtain the desired small sets by using ε-nets: Let P be an n-point set of the range space (X, Γ) and let $\varepsilon \leq 1/2$. A subset N of P is an *ε-net of P for (X, Γ)* if the following holds. For all $R \in \Gamma$, if $|R \cap P| \geq \varepsilon\,|P|$ then $R \cap N \neq \emptyset$. This concept was first introduced to computational geometry by Haussler and Welzl [10], but had also significant impact on other fields of computer science. To simplify notation an ε-net of P is denoted by $\mathcal{N}_\varepsilon(P)$. A notable fact is that for range spaces of finite VC-dimension one can always find ε-nets of P with a size that depends only on ε and not the size of P as the following lemma states.

Lemma 1 (see e.g. [16])**.** *Let (X, Γ) be a range space with finite VC-dimension $z \geq 1$. For a constant c_z, a parameter $\varepsilon \leq 1/2$ and an n-point subset P of X, there exists an ε-net of P for (X, Γ) of size at most $(c_z/\varepsilon)\log(1/\varepsilon)$.*

2.1 The framework

We present our framework for the generalized nearest neighbor problem, which we call *GNN-framework* for short. Let $P \subseteq \mathbb{R}^d$ be an n-point set, Θ the collection of query objects, and $\mathcal{D}$ the underlying metric distance function. The query collection together with the distance function form the range space $\mathfrak{R}_\Theta = \left(\mathbb{R}^d, \{\mathcal{B}_r(Q) \mid Q \in \Theta, r \geq 0\}\right)$, which is a fundamental element of the described framework. The framework solves the GNN problem if the range space $\mathfrak{R}_\Theta$ satisfies two properties:

1. There is an algorithm $\mathcal{A}_\mathcal{N}$ that constructs small ε-nets for $\mathfrak{R}_\Theta$
2. There is a reasonable efficient range searching algorithm $\mathcal{A}_\mathfrak{R}$ for $\mathfrak{R}_\Theta$.

These are the only limitations of the framework. Due to Lemma 1 small ε-nets exist for range spaces of finite VC-dimensionality and can be found either by random sampling or deterministically [5]. The existence of an efficient range searching algorithm is not implied by finite VC-dimensionality, but for many natural range spaces efficient algorithms have been found. For example, range spaces that are defined by a constant number of bounded polynomials have been studied by Agarwal et al. [3]. This indicates the potential for solving various concrete instances of the GNN problem as shown in Section 3. The GNN-framework works as follows:

Preprocessing. We preprocess $P_0 = P$ into a data structure $\mathfrak{D}$ which is a $(k+1)$-tuple $((P_0, \mathfrak{S}_0), (P_1, \mathfrak{S}_2), \ldots, (P_k, \emptyset))$ of the following elements. The sets P_i are hierarchically constructed ε-nets for the range space $\mathfrak{R}_\Theta$ that are build by $\mathcal{A}_\mathcal{N}$ in the following way: For a parameter $a \in (0, 1/2)$ we define $\varepsilon_i = n^a / |P_{i-1}|$ and $P_i = \mathcal{N}_{\varepsilon_i}(P_{i-1})$ for $1 \leq i \leq k$. The choice of a depends on the application and influences the query time; We only require that a is chosen such that $|P_i| < |\mathcal{N}_{\varepsilon_i}(P_{i-1})|$. The $\mathfrak{S}_i$ are range searching data structures for $\mathfrak{R}_\Theta$ build by $\mathcal{A}_\mathfrak{R}$ on the sets P_i for $0 \le i < k$. The parameter k also depends on the application and has only impact on the query time.

Query processing. (see Algorithm 1) For a query object $Q \in \Theta$ we describe how a closest point in P_0 is found. Note that the algorithm works in a recursive fashion and each recursive instance has access to $\mathfrak{D}$ while processing the query. The initial call of the algorithm works on $(P_0, \mathfrak{S}_0)$, the i-th recursive call works on $(P_i, \mathfrak{S}_i)$ and the recursion stops at the k-th recursive call. We describe the initial call of the algorithm since it is representative for the others: First, the algorithm finds a point α closest to Q in the ε-net P_1 by recursively calling itself on P_1. Then, the distance r from α to Q is used to define a range $\mathcal{B}_r(Q) = \mathcal{B}_{\mathcal{D}(\alpha,Q)}(Q)$. The range searching algorithm $\mathcal{A}_\mathfrak{R}$ utilizing the data structure $\mathfrak{S}_0$ is asked to retrieve the points $R = P_0 \cap \mathcal{B}_r(Q)$. The resulting set R is searched point-by-point for the nearest neighbor of Q. If the resulting set is empty, α is outputted. As stated above, the last recursive call at recursion depth k is handled specially: The set P_k is searched point-by-point for a nearest neighbor of Q.

Algorithm 1. *Data Structure:* $\mathfrak{D} = ((P_0, \mathfrak{S}_0), (P_1, \mathfrak{S}_1), \ldots, (P_k, \emptyset))$

Initial call: `Query`(Q, 0) *Input:* query object Q *Output:* nearest neighbor of Q;
Comment: `RangeSearch`$(\mathcal{B}_{\mathcal{D}(\alpha,Q)}(Q), i)$ returns $\mathcal{B}_{\mathcal{D}(\alpha,Q)}(Q) \cap P_i$;

```
1  Function: Query($Q$, $i$)
2     If $i < k$ then
3        $\alpha$ = Query($Q, i+1$)
4        $R$ = RangeSearch($\mathcal{B}_{\mathcal{D}(\alpha,Q)}(Q), i$)
5     else
6        $R = P_k$
7     end
8     For each $\pi \in R$ do
9        If $\mathcal{D}(\alpha,Q) \ge \mathcal{D}(\pi,Q)$ then
10          $\alpha = \pi$
11       end
12    end
13
14    return $\alpha$
```

Theorem 1. *Let P be an n-point set, Θ a query collection and $\mathcal{D}$ a distance function. For a query object $Q \in \Theta$, Algorithm 1 correctly outputs a point $\pi \in P$ with $\mathcal{D}(\pi, Q) \le \mathcal{D}(\pi', Q)$ for all $\pi' \in P$.*

Proof. We prove the theorem by induction on the recursion depth. All line numbers refer to Algorithm 1. As induction basis we focus on the highest level of recursion, where $i = k$. In this case R is set to P_k (line 6) and the for-loop (line 8-12) finds a point α in $P_k = R$ that is closest to Q. This point is returned, so the last recursive call of the algorithm returns a point in P_k that is closest to Q.

As induction hypothesis, we assume that at recursion depth $i+1$ the algorithm returns a point in P_{i+1} that is closest to Q.

For the induction step let $i < k$. The point returned by the recursive call is α (line 3), and by the induction hypothesis it is a point in P_{i+1} closest to Q. The range searching algorithm returns all points in $P_i \cap \mathcal{B}_{\mathcal{D}(\alpha,Q)}(Q)$, consequently all the points in the $\mathcal{D}(\alpha, Q)$-neighborhood of P_i are put in R. If R is non-empty it must contain the desired point and the for-loop (line 8-12) finds it. If R is empty, the point α, which is a point in P_i and P_{i+1}, is closest to Q. Thus the algorithm works correctly. □

To bound query time and space requirements, we introduce further notation for the time and space requirements of the subroutines. For the range searching algorithm $\mathcal{A}_{\mathfrak{R}}$ let $\mathcal{T}^*_{\mathfrak{R}}(n)$ be the time needed to preprocess an n-point set P into a range searching data structure of size $\mathcal{S}_{\mathfrak{R}}(n)$, and let $\mathcal{T}_{\mathfrak{R}}(n, m)$ be the time needed to report all m points in the intersection of P with any query range. Furthermore, let $\mathcal{T}^*_{\mathcal{N}}(n)$ be the time needed by the ε-net algorithm $\mathcal{A}_{\mathcal{N}}$. The time needed to compute the distance between the query object and a point is at most $\mathcal{T}_{\mathcal{D}}$. We proceed with a bound on the query time.

Theorem 2. *Let P be an n-point set, Θ a query collection and $\mathcal{D}$ a distance function. Furthermore, let $a \in (0, 1/2]$ and $k \in \mathbb{N}$ be parameters of the data structure $\mathfrak{D}$ described above. Then, Algorithm 1 using Algorithm $\mathcal{A}_{\mathfrak{R}}$ finds a nearest neighbor of $Q \in \Theta$ in time $\mathcal{T}(n) \leq k\mathcal{T}_{\mathfrak{R}}(n, n^a) + kn^a\mathcal{T}_{\mathcal{D}} + |P_k|\,\mathcal{T}_{\mathcal{D}}$.*

Proof. We prove the theorem by solving a recurrence equation that we derive from the query algorithm. All line numbers refer to Algorithm 1. At the highest depth of recursion the algorithm works on $(P_k, \emptyset)$ and searches P_k for a closest point to Q (line 6, line 8-12). Thus, we get $\mathcal{T}(|P_k|) = \mathcal{T}_{\mathcal{D}}\,|P_k|$. The running time $\mathcal{T}(|P_i|)$ of all recursive calls at depth $i < k$ working on $(P_i, \mathfrak{S}_i)$ is bounded by the following components. First, by $\mathcal{T}(|P_{i+1}|)$ which is the running time of the recursive call on P_{i+1} (line 3). Then, by $\mathcal{T}_{\mathfrak{R}}(|P_i|, |R|)$ which is the running time of the range searching algorithm $\mathcal{A}_{\mathfrak{R}}$ on P_i, and finally, by the time needed to search the resulting set R for the closest point to Q which is bounded by $\mathcal{T}_{\mathcal{D}}\,|R|$ (line 8-12). From this we get

$$\mathcal{T}(|P_i|) \leq \mathcal{T}(|P_{i+1}|) + \mathcal{T}_{\mathfrak{R}}(|P_i|, |R|) + \mathcal{T}_{\mathcal{D}}\,|R|,$$

for $i < k$. The size of the resulting set $R = P_i \cap \mathcal{B}_{\mathcal{D}(\alpha,Q)}(Q)$ can be bounded by the following arguments. Since α is point in P_{i+1} closest to Q, by the definition of $\mathcal{B}_r(.)$ we deduce that $P_{i+1} \cap \mathcal{B}_{\mathcal{D}(\alpha,Q)}(Q)$ is empty. The set P_{i+1} is an $(n^a/|P_i|)$-net of P_i. Thus, by the definition of ε-nets the size of $R = P_i \cap \mathcal{B}_{\mathcal{D}(\alpha,Q)}(Q)$ is at most $(n^a/|P_i|)\,|P_i| = n^a$. This yields

$$\mathcal{T}(|P_i|) \leq \mathcal{T}(|P_{i+1}|) + \mathcal{T}_{\mathfrak{R}}(|P_i|, n^a) + n^a\mathcal{T}_{\mathcal{D}}.$$

We solve this recurrence equation from $i = 0$ up to k and obtain

$$\mathcal{T}(|P_0|) \leq \sum_{i=0}^{k-1} \left(\mathcal{T}_{\mathfrak{R}}(|P_i|, n^a) + n^a\mathcal{T}_{\mathcal{D}}\right) + |P_k|\,\mathcal{T}_{\mathcal{D}}$$

We have chosen a such that the size of P_i is monotone decreasing in i, thus by $|P_i| \leq |P_0| = n$ we get $\mathcal{T}(n) \leq k\mathcal{T}_{\mathfrak{R}}(n, n^a) + kn^a\mathcal{T}_{\mathcal{D}} + |P_k|\,\mathcal{T}_{\mathcal{D}}$, which proves the theorem. □

Theorem 3. *Let P be an n-point set, Θ a query collection and $\mathcal{D}$ a distance function. Furthermore, let $a \in (0, 1/2]$ and $k \in \mathbb{N}$ be parameters of the data structure $\mathfrak{D}$ described above. The time needed to preprocess P into $\mathfrak{D}$ using algorithms $\mathcal{A}_{\mathcal{N}}$ and $\mathcal{A}_{\mathfrak{R}}$ is $\mathcal{T}^*(n) \leq k\,(\mathcal{T}^*_{\mathfrak{R}}(n) + \mathcal{T}^*_{\mathcal{N}}(n))$ and the space requirement is $\mathcal{S}(n) \leq k(n + \mathcal{S}_{\mathfrak{R}}(n))$.*

Proof. The parameter a used for the construction of the ε-nets $P_1, P_2, \ldots, P_k$ is chosen such that the size of P_i is monotone decreasing in i. Both, the time $\mathcal{T}^*_{\mathcal{N}}(n)$ to create an ε-net on n points as well as the time $\mathcal{T}^*_{\mathfrak{R}}(n)$ increase in n. Thus, the preprocessing time is

$$\mathcal{T}^*(n) \leq \sum_{i=0}^{k} (\mathcal{T}^*_{\mathfrak{R}}(|P_i|) + \mathcal{T}^*_{\mathcal{N}}(|P_i|)) \leq k\,(\mathcal{T}^*_{\mathfrak{R}}(|P_0|) + \mathcal{T}^*_{\mathcal{N}}(|P_0|))\,,$$

which proves the first part of the theorem.

The space requirements $\mathcal{S}_{\mathfrak{R}}(n)$ of the range searching data structure is monotone increasing in n. Hence the space requirement of $\mathfrak{D}$ is

$$\mathcal{S}(n) \leq \sum_{i=0}^{k} (|P_i| + \mathcal{S}_{\mathfrak{R}}(|P_i|)) \leq k(|P_0| + \mathcal{S}_{\mathfrak{R}}(|P_0|)),$$

which proves the second part of the theorem. □

We observe that setting the recursion depth k larger than 1 is important when searching the points of the ε-net takes more time than the range searching.

3 Solutions for Concrete Query Collections

In this section we consider several types of query objects and distance functions, some which have been considered before (Section 3.3) and some which we analyze for the first time (Section 3.1 and 3.2). During the explanation we also follow the ideas of Voronoi diagrams to provide a better understanding of the similarities and differences to our framework (Section 3.1). Moreover, in the case of the ℓ_1-distance, we show that the details of a full analysis needed for the a generalization of Voronoi diagrams can lead to large difficulties, however, our framework stays easily applicable (Section 3.2).

3.1 Query lines in 3-dimensional space under ℓ_2-norm

The problem of searching an n-point set P in $\mathbb{R}^3$ for a nearest neighbor to a given query line has not been considered in literature. The analysis can be generalized to $\mathbb{R}^d$, however, to stay in a geometrically well studied space we focus on $d = 3$. To compare both concepts, we first solve the problem by following the ideas of Voronoi diagrams and afterwards by applying our framework, which obtains the results in a more direct way.

The idea of generalized Voronoi diagrams is based on ray-shooting in the lower envelope of the arrangement induced by the distance function. This arrangement lives in the space of query objects which can be considered as space dual to the point space. Therefore, the first step is to identify the distance function between a point and a line for a suitable representation of a line in $\mathbb{R}^3$. Let the Euclidean norm be denoted by $\|.\|$, the corresponding metric by $\mathcal{D}$ and standard scalar product by $\langle \rho, \rho' \rangle$. Any line Q can be represented by 4 parameters $\chi_Q = (l_1, l_2, l_3, l_4)$; one possible interpretation for these parameters is that $\alpha = (l_1, l_2, 0)$ and $\beta = (l_3, l_4, 1)$ determine the intersection of the line with the plane through the origin and the plane shifted upwards (w.r.t. the last coordinate) by one. The collection of all query lines is $\Theta_\ell = \big\{\{\alpha + t(\beta - \alpha) \mid t \in \mathbb{R}\} \mid (l_1, l_2, l_3, l_4) \in \mathbb{R}^4\big\}$.

The distance between a line $\alpha + t(\beta - \alpha)$ and a point $\pi \in \mathbb{R}^3$ equals the distance between $(\alpha - \pi) + t(\beta - \alpha)$ and the origin, for $t \in \mathbb{R}$. Let $\delta \in \mathbb{R}^3$ be the vector of shortest distance pointing to the line. It satisfies two properties: First, $\langle \delta, (\beta - \alpha) \rangle = 0$ and secondly, $\delta = (\beta - \pi) + s(\beta - \alpha)$ for some $s \in \mathbb{R}$. Inserting the second into the first property yields $s = -\langle \alpha - \pi, \beta - \alpha \rangle / \|\beta - \alpha\|^2$. After some calculation based on properties of the scalar product, the squared length of δ is $\|\delta\|^2 = \|\alpha - \pi\|^2 - \langle \alpha - \pi, \beta - \alpha \rangle^2 / \|\beta - \alpha\|^2$. We translate this rational function to the following polynomial function in π, χ_Q and denote $r = \|\delta\|^2$:

$$\mathcal{F}(\pi, \chi_Q, r) = \|\alpha - \pi\|^2 \|\beta - \alpha\|^2 - \langle \alpha - \pi, \beta - \alpha \rangle^2 - r \|\beta - \alpha\|^2 = 0. \qquad (1)$$

We write $\mathcal{F}_\pi(\chi, r)$ for the polynomial $\mathcal{F}(\pi, \chi, r)$ in which the parameter space is χ, r but π is fixed. We call the space in which the polynomial $\mathcal{F}_\pi$ lives *dual space*. Analogously, we call the space where $\mathcal{F}_{\chi,r}(\pi)$ lives *primal space*. The solutions of $\mathcal{F}_\pi(\chi, r) = 0$ for all $\pi \in P$ form an arrangement of algebraic varieties, which have the property that the first algebraic variety hit by a ray starting from $(\chi_Q, 0)$ going into the direction $(\chi_Q, 1)$ represents a nearest neighbor of the line Q.

As described in the introduction it is not known how to perform ray-shooting in an arrangement with linear space. This is indicated by the fact that the complexity of the lower envelope defined by a $(d-1)$-variate function is generally $\Omega(n^{d-1})$ [23]. Thus, we translate the situation to primal space: Every surface $\mathcal{F}_\pi = 0$ is dual to the point π and a ray $\{(\chi, r) \mid r \geq 0\}$ translates to a family of ranges $\{\mathcal{F}_{\chi,r}(\pi) \mid r \geq 0\}$. If the ray hits the first surface $\mathcal{F}_\pi$ at the point (χ, r^*), then the boundary of the range $\mathcal{F}_{\chi,r^*}$ intersects the point π. So, in primal space the ray-shooting translates to (algebraic) range searching that is solved by using a partitioning tree as data structure [3, 23]. Additionally the data structure can be equipped with ϵ-nets for every node to constrain the resulting range search operations [17].

Our method, which has the same asymptotic complexity as the method above, does not consider duality at all: We only need to define the range space $\mathfrak{R}_{\Theta_\ell} = \big(\mathbb{R}^3, \{\mathcal{B}_r(Q) \mid Q \in \Theta_\ell, r \geq 0\}\big)$ of all cylinders around all possible query lines using the algebraic variety $\mathcal{F}_{\chi,r} \leq 0$. On P we construct the range searching data structure of [3] for the range space $\mathfrak{R}_{\Theta_\ell}$ which yields a query time of

$\mathcal{O}\left(n^{2/3+f}\right)$ and space requirements in $\mathcal{O}\left(n\right)$. Secondly, on the point set P we generate a $(n^{-1/2})$-net N of size $\mathcal{O}\left(n^{0.5}\log n\right)$ (Lemma 1). We directly derive the following Observation:

Observation 1. *An n-point set P from 3-dimensional Euclidean space can be preprocessed in a linear space data structure, such that, for a constant parameter $f > 0$, a closest point to given query line can be found in time $\mathcal{O}\left(n^{2/3+f}\right)$.*

3.2 Query lines in 3-dimensional space under ℓ_1-norm

The GNN problem in 3-dimensional space with Manhattan distance has not been considered in literature. We show that, when following the ideas of generalized Voronoi diagrams, the details get complicated since the distance function is not described by a polynomial any more. In contrast our framework is easily applicable.

The distance between a line Q represented by χ_Q as above and a point $\pi = (p_1, p_2, p_3)$ is

$$\begin{aligned}\mathcal{D}\left(\pi, Q\right) \quad = \min\Big\{ \quad & |p_1 - l_3 + p_3(l_1 - l_3)| + |p_2 - l_4 + p_3(l_2 - l_3)|, \\ & \left|p_1 - l_3 + \frac{(p_2 + l_4)(l_1 - l_3)}{l_2 - l_4}\right| + \left|p_3 + \frac{p_2 + l_4}{l_2 - l_4}\right|, \\ & \left|p_2 - l_4 + \frac{(p_1 + l_3)(l_2 - l_4)}{(l_1 - l_3)}\right| + \left|p_3 + \frac{p_1 + l_3}{l_1 - l_3}\right| \Big\},\end{aligned}$$

because the distance can be computed as a minimum of ℓ_1-distances between the point π and the points $\sigma_1, \sigma_2, \sigma_3$, where σ_i is the intersection of the line Q with the i-th hyperplane that is orthogonal to the coordinate axis and intersects π. The fact that the distance function is a minimum is not harmful since we can take all three functions into the lower envelope with only constant overhead. The real problem comes from the functions themselves, because they are absolute values of rational functions. Known techniques for the decomposition of the lower envelope induced by such functions cannot be directly applied. Further investigation is needed to see that the intersection of the functions is linear so that a projection yields a suitable Voronoi diagram.

On the other hand, the presented framework is easier to apply: It requires only to identify the range space that is naturally given by constant number of simplices. To obtained a range $\mathcal{B}_r(Q)$, it is necessary to project the points of the r-ball of the l_1-metric to the hyperplane that is orthogonal to the query line. The convex hull of the projected points implicitly describes the range. With a standard simplex range searching data structure [4] we obtain the following result.

Observation 2. *An n-point set P from 3-dimensional space with Manhattan distance can be preprocessed into a linear space data structure, such that, for a constant parameter $f > 0$, a closest point to given query line can be found in time $\mathcal{O}\left(n^{2/3+f}\right)$.*

3.3 Previously considered query collections

In the same way we can easily derive data structures and query time bounds for several kind of query objects. Here, we focus on the Euclidean distance.

The case of 2-dimensional query lines has been considered by Mitra et al. and Mukhopadhyay [19, 22] who used ham-sandwich cuts or Matoušek's simplicial partion theorem to solve the problem. For linear space with $\mathcal{O}(n \log n)$ preprocessing time, the query time is $\mathcal{O}\left(n^{0.695}\right)$ and for $\mathcal{O}(n \log n)$ space with $\mathcal{O}\left(n^{1+f}\right)$ preprocessing time, the query time is $\mathcal{O}\left(n^{1/2+f}\right)$, for $f > 0$. Furthermore, the case of query 3-dimensional hyperplanes has been considered by Mitra et al. [20] who obtained a query time of $\mathcal{O}\left(n^{2/3+f}\right)$ using $\mathcal{O}\left(n^{1+f}\right)$ preprocessing time and $\mathcal{O}(n \log n)$ space.

An application of our framework for general query hyperplanes in Euclidean space of dimension d achieves a faster query time while using less space. The required range space contains all possible cuts of two parallel half-spaces. An appropriate standard simplex range searching data structures as e.g.[15] can be used. An epsilon-net can be obtained by random sampling in $\mathcal{O}(n)$ time which, however, leads to a randomized algorithm. At the cost of larger preprocessing time, one could also use deterministic ε-net algorithms [14].

Observation 3. *An n-point set P from d-dimensional Euclidean space can be preprocessed in $\mathcal{O}(n \log n)$ time into a linear space data structure, such that, for a parameter $f > 0$, a closest point to a given query hyperplane can be found in time $\mathcal{O}\left(n^{1-1/d+f}\right)$ w.h.p.*

In [21] the authors consider circles as query objects and obtain a query time of $\mathcal{O}\left(n^{2/3+f}\right)$ and $\mathcal{O}(n \log n)$ space. Using a standard lifting transform, the simplex range searching algorithm from above and sampling for the ε-net, we improve the space requirements by a logarithmic factor.

Observation 4. *An n-point set P from 2-dimensional Euclidean space can be preprocessed in $\mathcal{O}(n \log n)$ time into a linear space data structure, such that, for a parameter $f > 0$, a closest point to given query circle can be found in time $\mathcal{O}\left(n^{2/3+f}\right)$ w.h.p.*

These bounds could also be achieved by following the ideas of generalized Voronoi diagrams, which lead to analogous steps as in Section 3.1, but there is no publication considering such an analysis.

4 Comparison with Generalized Voronoi Diagrams

In this section we want to compare the presented framework with the ideas behind generalized Voronoi diagrams. There are two aspects in favor of our framework that are worth mentioning. First, the approach of using Voronoi diagrams has been formulated on a high-level. The authors are not aware of any work that allows to directly derive a data structure with bounds on query time or space for a given query collection. In contrast, these properties are contained in the presented Theorems. Secondly, following the ideas of generalized Voronoi diagrams might be more complex than using our framework, especially if a linear space data structure is required. The reasons for that lie in the details. Applying the ideas of Voronoi diagram to a concrete query collection might end up in a complex analysis in dual space as e.g. in Section 3 (query lines in 3-space with ℓ_1-norm). Such a complicated analysis is not necessary for the application of the here presented framework.

The idea behind generalized Voronoi diagrams benefits from geometric duality defined by the distance function. The concept of geometric duality is usually a powerful tool [13] that provides a different view on the problem, even if the mathematics might be very similar. We discuss that in the case of linear space requirements, solving a concrete variant of the GNN problem might be disadvantageous due to a detour through dual space.

Let $\chi_Q \in \mathbb{R}^q$ be a vector representing a query object Q and let $\mathcal{D}(\pi, \chi_Q)$ be the distance between a point $\pi \in P$ and Q. For a fixed parameter $\pi \in P$ the distance $\mathcal{D}(\pi, \chi_Q)$ is a function in χ_Q. Consider the graphs $(\chi_Q, \mathcal{D}(\pi, \chi_Q))$ of this function for all $\pi \in P$. These graphs produce an arrangement of surfaces in a $(q+1)$-dimensional space called *dual space*. The d-dimensional space in which the points P live is called *primal space*. We introduce these notions because the distance function plays the role of a geometric duality transformation [8]: A point in dual space, say (χ_{Q_1}, t), is below (w.r.t. to the last coordinate) a surface $(\chi_Q, \mathcal{D}(\pi_1, \chi_Q))$, if and only if π_1 is not contained in the set of all points within distance t from Q_1. The standard approach makes use of this property in the following way. For a query object Q_1, a nearest neighbor is found by shooting a ray that starts in $(\chi_{Q_1}, 0)$ and goes into the direction of $(\chi_{Q_1}, 1)$. The first surface that is hit, say $(\chi_Q, \mathcal{D}(\pi_1, \chi_Q))$, represents the desired point π_1 [9, 1]. So, the nearest neighbor problem reduces to ray-shooting in an arrangement of surfaces, more precisely, the ray starts below all surfaces in the cell that is bounded by the point-wise minima of all graphs $(\chi_Q, \mathcal{D}(\pi, \chi_Q))$ for $\pi \in P$. This cell, or its boundary, is commonly known as the *lower envelope* of the arrangement [8, 1]. Note that the projection of the lower envelope onto $\mathbb{R}^q$ corresponds to a Voronoi diagram of the points P for the query collection Θ [1] and the distance function $\mathcal{D}$.

Ray-shooting in the lower envelope of an arrangement that is induced by arbitrary surfaces is generally a hard task and cannot be done efficiently. However, if the arrangement is induced by algebraic varieties, it is standard practice to reduce ray-shooting to the segment-emptiness problem combined with parametric search [2, 17, 11]. Generally, this works by turning an optimization problem into a decision problem, with a parallel decision algorithm, and querying this algorithm several times in a kind of binary search to find the optimal value of the optimization problem. In our case we are interested in a data structure that decides whether a line segment $(\chi_{Q_1}, 0), (\chi_{Q_1}, t)$ intersects with any of the given algebraic varieties. Then, this data structure is queried several times to find the value t^* such that (χ_{Q_1}, t^*) is the first intersection of the ray with the algebraic variety corresponding to a point π_1 of smallest distance $\mathcal{D}(\pi_1, \chi_{Q_1})$. The parametric search technique usually generates a logarithmic overhead. For details we refer the reader to [18, 2]. The decision problem for the segment emptiness problem is directly related to point location: If a point (χ_{Q_1}, t) is contained in the lower envelope of the arrangement, the segment $(\chi_{Q_1}, 0), (\chi_{Q_1}, t)$ does not intersect any of the given algebraic varieties.

Point location cannot be done in linear time when working with arrangements, since their complexity is usually super-linear in n. Therefore, the algebraic varieties are translated back to primal space using the distance function as the duality transformation, such that an algebraic variety $(\chi_Q, \mathcal{D}(\pi_1, \chi_Q))$ becomes the point π_1 again. In this way, in primal space, the point location problem reduces to (algebraic) range searching: A point (χ_{Q_1}, t) in dual space is contained in the lower envelope if in primal space none of the points in P are contained in the range that is defined by $\mathcal{D}(\chi_{Q_1}, \pi) - t < 0$, which is a function in $\pi \in \mathbb{R}^d$ with fixed parameters χ_{Q_1} and t [17]. In the end the problem is also reduced to a range searching data structure, but with a kind of mental overhead. If one aims for linear space, it is a simpler alternative to stay directly in primal space and use the presented framework.

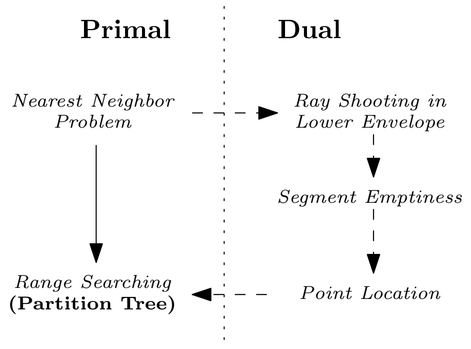

Figure 1: The diagram illustrates the transformations from the nearest neighbor problem to the final problem and its corresponding data structure. Primary and Dual denote spaces between which problem specific duality transformations are defined. The dashed path describes the chain of transformations considered in the standard approach while the solid arrow shows our approach.

Acknowledgments The authors would like to thank Peter Widmayer for stimulating discussions and continuous support of this project. We are also grateful to Jiri Matoušek for several discussions on the duality, arrangements and tricky details.

References

[1] P. Agarwal and M. Sharir. Arrangements and their Applications. *Handbook of Computational Geometry*, pages 49–119, 1998.

[2] P. K. Agarwal and J. Matoušek. Ray Shooting and Parametric Search. In *STOC'92: Proceedings of the 24th Annual ACM Symposium on Theory of Computing*, pages 517–526. ACM, 1992.

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

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

[5] B. Chazelle. *The Discrepancy Method.* Cambridge University Press, 2000.

[6] B. Chazelle and E. Welzl. Quasi-optimal range searching in spaces of finite VC-dimension. *Discrete and Computational Geometry*, 4(1):467–489, 1989.

[7] R. Cole and C.-K. Yap. Geometric Retrieval Problems. In *FOCS'83: Proceedings of the 24th Annual IEEE Symposium on Foundations of Computer Science*, pages 112–121, 1983.

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

[9] H. Edelsbrunner and R. Seidel. Voronoi Diagrams and Arrangements. *Discrete and Computational Geometry*, 1(1):25–44, 1986.

[10] 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.

[11] 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.

[12] R. Krauthgamer and J. Lee. Navigating Nets: Simple Algorithms for Proximity Search. In *SODA'04: Proceedings of the 15th annual ACM-SIAM Symposium on Discrete Algorithms*, pages 798–807. ACM, 2004.

[13] D. T. Lee and Y. T. Ching. The Power of Geometric Duality Revisited. *Information Processing Letters*, 21:117–122, 1985.

[14] J. Matoušek. Construction of epsilon-Nets. *Discrete & Computational Geometry*, 5:427–448, 1990.

[15] J. Matoušek. Efficient Partition Trees. *Discrete and Computational Geometry*, 8(1):315–334, 1992.

[16] J. Matoušek. *Geometric Discrepancy.* Springer, 1999.

[17] J. Matoušek and O. Schwarzkopf. On Ray shooting in Convex Polytopes. *Discrete and Computational Geometry*, 10(1):215–232, 1993.

[18] N. Megiddo. Applying Parallel Computation Algorithms in the Design of Serial Algorithms. *Journal of the ACM*, 30(4):852–865, 1983.

[19] P. Mitra and B. B. Chaudhuri. Efficiently computing the closest point to a query line. *Pattern Recognition Letters*, 19(11):1027–1035, 1998.

[20] P. Mitra and A. Mukhopadhyay. Computing a Closest Point to a Query Hyperplane in Three and Higher Dimensions. In *ICCSA'03: Proceedings of the 2003 International Conferecnce on Computational Science and Its Applications*, pages 787–796, 2003.

[21] P. Mitra, A. Mukhopadhyay, and S. V. Rao. Computing the Closest Point to a Circle. In *CCCG'03: Proceedings of the 15th Canadian Conference on Computational Geometry*, pages 132–135, 2003.

[22] A. Mukhopadhyay. Using simplicial partitions to determine a closest point to a query line. *Pattern Recognition Letters*, 24(12):1915–1920, 2003.

[23] 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.