Proceedings of the Thirty-Fifth International Conference on Automated Planning and Scheduling (ICAPS 2025)

Instance-based Approximation Guarantees for Graph-based Nearest Neighbor Search

Yannick Bosch, Sabine Storandt

University of Konstanz
{yannick.bosch,sabine.storandt}@uni-konstanz.de

Abstract

Nearest Neighbor Search (NNS) in high-dimensional point sets is an important building block in many application areas, including pattern recognition, machine learning, planning, data mining, and computational geometry. Graph-based approaches that offer approximate NNS (ANNS) are ubiquitously used for these applications, and a variety of suitable graph structures have been proposed for this purpose. However, these approaches do not come with a priori approximation guarantees, often not even in low dimensions. Thus, there may be query points for which the distance to the returned ANN is significantly larger than the distance to the true NN. A common way to assess the quality of graph-based search and to compare different variants is the evaluation of query point samples. However, since the space of potential query points is infinite, it is likely that the samples will give biased results and that critical points will be missed. To systematically evaluate the ANNS quality of a given graph structure, we propose an algorithm that identifies the query point with the worst ratio r between ANN distance and true NN distance. This ratio provides a tight instance-based approximation guarantee. Our algorithm relies on a new geometric data structure called search-path diagram. In our experiments on established base graphs, we demonstrate that sampling based evaluation heavily underestimates r, while our method provides a robust quality assessment.

Solutions for t — https://tinyurl.com/2ufpuk7a

Introduction

Given a set S of n d-dimensional points and a query point $q \in \mathbb{R}^d$, the question which point $p \in S$ is closest to q is one of the most fundamental query types in computational geometry, also called the *nearest neighbor* (NN) search problem. Naively, queries can be answered in $\mathcal{O}(nd)$. However, for many applications, sublinear query times are vital. Prominent examples are ML and pattern recognition applications, where input sets often come with both, large n and d, and a large number of NN queries need to be issued to achieve the desired outcome (Lei et al. 2019; Kramer 2013). There are also diverse applications in motion-planning (Yershova and LaValle 2007; Kleinbort, Salzman, and Halperin 2020) and machine or drone scheduling (Wang, Reinelt, and Tan 2012; Lei and Chen 2022). Geometrical data structures can be leveraged for exact NN search (Yianilos 1993; Dasgupta and Freund 2008), but they suffer from the so called *curse of dimensionality*, that is, their performance deteriorates severely with growing d. The best known NN search time is $n^{1-\Theta(1/d)}$ which is essentially linear for large d (Lee and Wong 1977). However, for many application scenarios, it is sufficient to obtain an approximate NN (ANN). A point $p' \in S$ is said to be a r-approximate NN of a query point q for some constant $r > 1$ if $d(q, p') \leq r \cdot d(q, p)$ where p denotes the NN of q. A plethora of methods for ANN search have been proposed (Cao et al. 2017; Li et al. 2019; Abbasifard, Ghahremani, and Naderi 2014). In particular, graph-based approaches gained a lot of traction in the last years (Fu and Cai 2016; Wang et al. 2021; Liu et al. 2022). The core idea is quite simple: *Construct a graph $G(S, E)$ on the point set S. Given a query point q, start at some node s in G. If s is closer to q than all of its neighbors, return s as ANN. Otherwise, proceed to the neighbor of s that is closest to q and repeat.*

There are different variants depending on the choice of the base graph, the initial start node, and the precise search algorithm (Fu et al. 2017; Munoz et al. 2019; Zhao et al. 2023; Gollapudi et al. 2023; Chen et al. 2023). To decide which approach offers the best ANN quality is non-trivial, though. Graph-based approaches rarely come with any (practically useful) theoretical approximation guarantee. Instead, the ANN quality is typically assessed empirically by choosing a sample of query points and comparing for each point the distance of the returned ANN to the true NN distance (computed with the naive algorithm). But since the space of query points is infinite, it is questionable how expressive the respective results really are. Critical query points which induce large r-values can easily be missed in a random sample. Thus, using such an evaluation method, the existence of query points with bad ANN results might go unnoticed. When such query points occur in practice, the performance of ML, data mining or outlier detection algorithms that rely on the query result can be severely distorted.

The main goal of the paper is to propose a new and formally sound method to faithfully assess the ANN quality of graph-based approaches on a given instance. For this purpose, we devise an algorithm that identifies the query point with worst ratio r between ANN and NN distance. The value

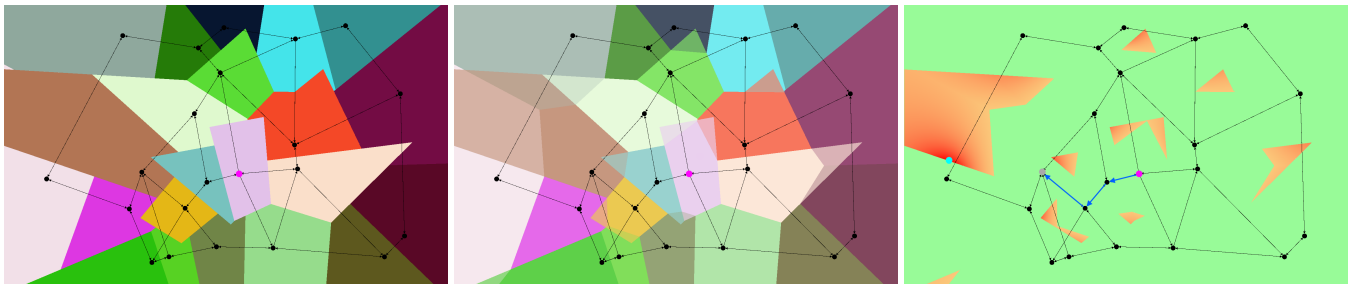


Figure 1: Depicted on the left is the SPD computed for the pink start node. Overlayed in the middle is the VD of the point set. Shown on the right are the ratios r for any given query point in the frame. The green area represents queries for which the actual NN is returned. Higher values of r correspond to redder regions. The light blue point shows the worst query point in this graph instance. Its ANN-path and vertex are depicted in blue and gray respectively.

of r is thus a tight instance-based approximation guarantee. Considering the r-values of different approaches allows for a systematic and fair comparison between them. Furthermore, finding query points for which a given graph-based approach performs poorly helps to detect structural issues and to guide refinement strategies (as e.g. edge insertions).

Related Work

There is vast literature on ANN search and on graph-based methods. We thus focus on approaches that come with theoretical guarantees and on practical evaluation protocols.

Arya and Mount (Arya and Mount 1993) introduced a randomized neighborhood graph construction algorithm for ANN queries. For $d \in \mathcal{O}(1)$, the preprocessing takes $\mathcal{O}(n)$ time and requires $\mathcal{O}(n \log n)$ space. ANN queries are answered in $\mathcal{O}(\log^3 n)$ time and the returned point is guaranteed to be within a factor of $(1+\varepsilon)$ of the true NN, where the choice of $\varepsilon > 0$ impacts the aforementioned running times. The approach was shown to outperform the classical k-d-tree data structure, for which theoretical guarantees can only be given under strong assumptions about the input point distribution (Friedman, Bentley, and Finkel 1977). A conditional lower bound proven by Rubinstein (Rubinstein 2018) implies that similar results cannot be achieved for ANN search data structures on high-dimensional inputs where $d \gg \log n$ (unless SETH fails), as any algorithm that uses polynomial preprocessing time requires near-linear time to answer ANN queries. Thus, the *curse of dimensionality* applies to any ANN search algorithm.

Given this lack of algorithms with meaningful theoretical guarantees, different graph-based methods are usually compared in an experimental way. While preprocessing time and space consumption can be measured easily, the ANN quality is more difficult to quantify. There are established quality measures for a given query point set, as recall, precision and accuracy (Fu et al. 2017; Singh et al. 2021; Wang et al. 2021). A measure that combines recall and accuracy is the r-approximative recall where the fraction of query results is considered for which the returned point is within a factor of r of the true NN distance (Aumüller, Bernhardsson, and Faithfull 2020). Clearly, the respective values cannot be computed for all possible query points, as there is an infinite number of candidates. Thus, the main problem is to select a representative query point set that allows for fair comparison between different approaches. Often, those samples are simply chosen uniformly at random or supplied with the benchmark data (Aumüller, Bernhardsson, and Faithfull 2020; Ren, Zhang, and Li 2020). However, it is unclear how expressive those statistical results are and whether some other sample might favor a completely different approach.

Contribution

In this paper, we propose an algorithm that helps to overcome the prevailing issue with the experimental methodology to evaluate the quality of graph-based ANN search. As the query space is the whole $\mathbb{R}^d$, choosing any finite (random) sample of query points is not sufficient to faithfully estimate how far the query result can be from the true NN. Our new algorithm is the first to compute for a given base graph and start node the smallest value of r such that for any query point p the returned ANN is within a factor of r of the true NN distance. The main idea behind the algorithm is to consider all possible ANN searches simultaneously. This is realized by our newly introduced data structure, called search-path diagram (SPD), which incrementally partitions the space into cells of query points for which the greedy search procedure reaches the same graph node after a fixed number of steps. We then intersect this space partition with the Voronoi diagram (VD) induced by the input points, which encodes the true NN for each possible query point. To obtain the query point with worst ANN quality, we prove that it needs to be located on the boundary of the intersection of the SPD and the VD and provide an efficient algorithm to identify the respective point by analytical means. The three main steps of our approach are visualized in Figure 1.

Our algorithm works for d-dimensional inputs and query points. To demonstrate its correctness and usefulness, we provide an exact implementation for $d = 2$ and a discretized implementation for higher dimensions, and test them on six different base graphs. Our experiments confirm that the evaluation of random query point samples tends to severely underestimate r even for huge sample sizes, sometimes by orders of magnitude. We demonstrate that the comparison of r-values for different base graphs allows for a more fair and robust assessment of their respective quality and also provides further interesting structural insights.

Preliminaries

In this section, we describe graph-based ANN search in a formal matter and introduce notation used throughout the paper. From now on, we always assume to be given a directed graph $G(V, E)$, where $V \subset \mathbb{R}^d$ is the input point set and E denotes the edge set. The most fundamental definition we need is that of a nearest neighbor (NN).

Definition 1 ($\text{NN}_q(G)$)**.** *A nearest neighbor of a point $q \in \mathbb{R}^d$ is defined as :* $\text{NN}_q(G) = \arg\min_{v \in V} d(q, v)$.

For a given point set, the space can be partitioned into cells that enclose the points with the same NN.

Definition 2 (Voronoi diagram $\text{VD}(V)$)**.** *Given a point $v \in V$ a Voronoi cell is defined as follows:*

$$\text{VC}_v = \left\{ q \in \mathbb{R}^d \;\middle|\; v \in \arg\min_{u \in V} d(q, u) \right\}. \tag{1}$$

$\text{VD}(V)$ *is defined as the tuple* $(\text{VC}_v)_{v \in V}$.

In graph-based search, given G as well as a start node $s \in V$ and a query point $q \in \mathbb{R}^d$, a greedy path traversal is performed from s to find an approximate NN (ANN) of q. Paths leading to an approximate nearest neighbor might be ambiguous due to equidistant neighbors, thus we define them accordingly.

Definition 3 (ANN-paths $P_q^s(G)$)**.** *Given a directed graph $G = (V, E)$, a node $s \in V$ and a query point $q \in \mathbb{R}^d$, the* ANN*-paths of q are defined as follows:*

$$\begin{aligned} P_q^s = \Bigg\{ p \;\Bigg|\; & p = (v_1, \ldots, v_n) \in S(G, s) \\ & \wedge \forall (u, v) \in p : v \in \arg\min_{w \in \{u\} \cup N(u)} d(q, w) \\ & \wedge v_n \in \arg\min_{w \in \{v_n\} \cup N(v_n)} d(q, w) \Bigg\}, \end{aligned} \tag{2}$$

where $N(\cdot)$ defines the outgoing neighbors of a node and $S(G, s)$ the simple paths in G starting at s.

Based on this query algorithm, we can formally define the ANN of a query point for a given graph and start node.

Definition 4 ($\text{ANN}_q^s(G)$)**.** *The ANN of a query point q given a start node $s \in V$ are defined as follows:*

$$\text{ANN}_q^s(G) = \{ v_n \mid p = (v_1, \ldots, v_n) \in P_q^s \} \cup X, \tag{3}$$

where X defines whether or not the start node s is also a valid approximate nearest neighbor:

$$X = \begin{cases} \{s\} & s \in \arg\min_{u \in \{s\} \cup N(s)} d(q, u), \\ \emptyset & otherwise. \end{cases} \tag{4}$$

For given G and s, we are interested in the smallest $r \geq 1$ such that $\forall q \in \mathbb{R}^d : d(q, \text{ANN}_q^s(G)) \leq r \cdot d(q, \text{NN}_q(G))$.

Search-Path Diagram

To compute the instance-based approximation ratio r for given G and s, we first introduce a new data structure which we call search-path diagram (SPD). The main idea is that the SPD encodes for each node in G the set of points for which the ANNS procedure returns said node. Exemplary depictions of an SPD are shown in Figures 1 and 7.

Definition 5 (Search-path diagram $\text{SD}_s(G)$)**.** *Given a start node $s \in V$ and a node $v \in V$, a search-path cell is defined as follows:* $\text{SC}_v^s = \{ q \in \mathbb{R}^d \mid v \in \text{ANN}_q^s \}$. *The search-path diagram* $\text{SD}_s(G)$ *is defined as the tuple* $(\text{SC}_v^s)_{v \in V}$.

However, this definition does not provide us with a way of actually constructing the search path diagram. For this we need to mimic the ANN search procedure for every point $q \in \mathbb{R}^d$. Considering the first step of this procedure, a neighbor of the start node is chosen that brings us closest to q. This already subdivides the entirety of $\mathbb{R}^d$ and can be modeled using restricted Voronoi diagrams.

Definition 6 (Restricted Voronoi Diagram)**.** *Given a point $v \in V$, the restricted Voronoi diagram is defined as* $VD_v(G) := \text{VD}(\{v\} \cup N(v))$. $\text{VD}_v[u]$ *denotes the cell for the point $u \in \{v\} \cup N(v)$.*

In the next step, the space defined by the cells of said diagram is further subdivided using yet again a locally restricted Voronoi diagram of the neighbor that was chosen in the previous step. Leveraging this fact, we can construct the SPD as described in Theorem 1. In it we go over every simple path p from s to v and union the aggregated intersections of the edges of p. Finally the intersection with $\text{VD}_v[v]$ is computed.

Theorem 1. *Let G be a directed graph and $s, v \in V$ then the following holds:*

$$\text{SC}_v^s = \begin{cases} \text{VD}_s[s] & v = s, \\ \left(\bigcup_{p \in S(G,s,v)} \bigcap_{(u,w) \in p} \text{VD}_u[w] \right) \cap \text{VD}_v[v] & else. \end{cases} \tag{5}$$

$S(G, s, v)$ *is the set of simple paths in G from s to v.*

Proof. We start with the case $v = s$. Let $q \in \text{SC}_s^s$, thus it is $s \in \text{ANN}_q^s$. Clearly this can only be the case if $s \in S$ (as defined in 4), as s cannot be an end node to a simple path starting at s. Thus it follows that $q \in \text{VD}_s[s]$. Let now $q \in \text{VD}_s[s]$. Then $s \in S$ follows, and hence $q \in \text{SC}_s^s$.

Let now $v \neq s$ and $q \in \text{SC}_v^s$. Since $v \neq s$, there exists a path $p = (v_1, \ldots, v_n)$ with $v_n = s$, fulfilling the conditions of 2. Since the following is to show:

$$(\exists p \in S(G, s, v) : \forall (u, w) \in p : q \in \text{VD}_u[w]) \wedge q \in \text{VD}_v[v],$$

and aforementioned conditions perfectly coincide, it clearly is $q \in A$, where A denotes the second case of 5. Let now $q \in A$. As this shows that $v \in \text{ANN}_q^s$, we are done. □

Clearly, building the SPD as in Theorem 1 is not very efficient, however it can be drastically improved by not considering every path p from s to v. The idea is to discard a path as soon as the aggregated intersection, denoted $\text{PM}[p]$

Algorithm 1: Search-path diagram

```
1  def searchPathDiagram(G, s):
2      PM ← {} // Map from paths to sets of polytopes with
         holes.
3      PM[⟨s⟩] ← ℝ^d
4      SD_s ← {} // Map from nodes to cells.
5      for v ∈ V do
6          SD_s[v] ← searchPathCell(G, PM, s, v)
7      return SD_s
```

of the currently consider path p, intersected with $\text{VD}_v[v]$ is empty, meaning p does not contribute to SC_v^s. Further improvements can be achieved by using a map from paths to sets of polytopes with holes as previously aggregated intersections can be reused. The respective pseudo-code is given in Algorithm 1 which calls Algorithm 2 for every $v \in V$. The computation of the cells can be parallelized if PM is a concurrent hash map or adequate locking is used.

Algorithm 2: Search-path cell

```
1  def searchPathCell(G, PM, s, v):
2      SC_v^s ← ∅
3      visited ← {}
4      CP ← ⟨s⟩ // Current path
5      simplePathsDFS(CP, v, SC_v^s, PM, visited)
6      return SC_v^s ∩ VD_v[v]
7  def simplePathsDFS(CP, v, SC_v^s, PM, visited):
8      currentPolytopeSet ← PM[CP]
9      if currentPolytopeSet ∩ VD_v[v] = ∅ then
10         CP.popBack()
11         return
12     u ← CP.back()
13     visited[u] ← true
14     if u = v then
15         SC_v^s ← SC_v^s ∪ currentPolytopeSet
16         CP.popBack()
17         visited[v] ← false
18         return
19     for w ∈ N(u) do
20         if ¬visited[w] then
21             CP.pushBack[w]
22             if CP ∉ PM then
23                 PM[CP] ←
                     currentPolytopeSet ∩ VD_u[w]
24             simplePathsDFS(CP, v, SC_v^s, PM,
                 visited)
25     CP.popBack()
26     visited[u] ← false
```

The complexity of Algorithm 2 is determined by precomputing the restricted Voronoi diagrams VD_u for each $u \in V$, with a running time in $\mathcal{O}(n \cdot N_{max} \log(N_{max}))$, for N_{max} denoting the maximum out-degree in the graph, as well as traversing all simple paths emerging from s for the cell computation. The geometric operations also have an impact, however, for this analysis we focus on the dependence on the number of nodes. In the worst case, a graph can have an exponential number of simple paths between two nodes. However, as the algorithm only considers viable monotone paths to the destination v, as enforced by the termination condition, the true number of paths that are explored is typically much smaller. For example, if the base graph is a tree, the number is linear. Further analysis of this aspect is provided in the experimental section.

Instance-Based Approximation Guarantee

In order to find for a given graph G and a start node s the minimal value r satisfying $d(q,w) \leq r \cdot d(q,v),\ w \in \text{ANN}_q^s,\ v \in \text{NN}_q$, for any query $q \in \mathbb{R}^d$, we need to compute the *worst case query*. For this we maximize the following ratio: $\frac{d(q,w)}{d(q,v)} \leq r, w \in \text{ANN}_q^s,\ v \in \text{NN}_q$. However, as square roots used in the distance computations are inconvenient, we will optimize the squared version of the inequality. This is possible as the fraction is always positive and squaring is a monotonic operation, thus the extreme points stay the same.

By computing the intersection between the Voronoi diagram and the search-path diagram we can encode information of the NN_q versus the ANN_q^s of all points $q \in \mathbb{R}^d$. Said intersection can be modeled by a matrix IM_s.

Definition 7 (Intersection matrix $\text{IM}_s(G)$)**.** *Given a start node $s \in V$ and nodes $v, w \in V$, the values of the intersection matrix $\text{IM}_s(G)$ are defined as follows: $I_{vw} = \text{VC}_v \cap \text{SC}_w^s$, where VC_v are the cells of $\text{VD}(V)$ and SC_w^s are the cells of $\text{SD}_s(G)$.*

This encourages us to use the following objective as for every cell in the matrix the nearest and approximate nearest neighbor is fixed:

$$g(q,v,w) = \frac{d(q,w)^2}{d(q,v)^2} = \frac{\sum_{i=0}^{d}(q_i - w_i)^2}{\sum_{i=0}^{d}(q_i - v_i)^2},\ v,w \in V$$

Algorithm 3 now computes the worst case query as follows. It starts by computing all cells I_{vw} of IM_s and storing them in a list I. If the node w of the search-path cell SC_w^s is not contained in itself, the algorithm returns as there is no approximation factor (as for a query point $q = w$, ANN_q^s won't be w). If I_{vw} is non-empty and $v \neq w$, an entry is added to I with the corresponding nodes and the intersection cell I_{vw}. For $v = w$, I_{vw} only contains points for which the optimal nearest neighbor is found, thus it is not added.

The next step iterates over the entries in the list I. We search the worst case query along the boundary of I_{vw} (as it is an intersection of sets of polytopes with holes it is again a set of polytopes with holes). For each edge we define a $d-1$ dimensional object C, embedded in d dimensional space, with parameter vector $\boldsymbol{t}$ ($d-1$ dimensional) and set the gradient of g in line 15 to the zero vector (v being the nearest neighbor and w being the approximate nearest neighbor). Solving it with respect to $\boldsymbol{t}$ results in a solution set S (link to 2D solutions using WolframAlpha below abstract). For each possible solution $\boldsymbol{t}_k \in S$ we check if it lies on I_{vw}. If it does the rooted objective is evaluated for $C(\boldsymbol{t}_k)$ and added to a

set of possible worst queries. Furthermore the corner points of I_{vw} are added to N in case no solution contained in I_{vw} is found. The last step simply finds the query point with the maximum ratio in N.

Algorithm 3: Worst query point detection

```
I ← ∅ // tuples of two nodes and a set of polytopes with holes.
for VC_v ∈ VD do
    for SC^s_w ∈ SD_s do
        if w ∉ SC̊^s_w then
            return // There is no approximation factor.
        I_vw ← VC_v ∩ SC^s_w
        if I_vw ≠ ∅ ∧ v ≠ w then
            I.append((v, w, I_vw))
N ← ∅ // tuples of a point in ℝ^d and a real number.
for (v, w, I_vw) ∈ I do
    for F ∈ Facets(I_vw) do
        u = F.points[0]
        E_u = F.incident(u)
        C(t) = u + Σ_{e_i ∈ E_u} e_i t_i
        S = solve(∇g(C(t), v, w) = 0)
        for t_i ∈ S do
            if C(t_i) ∈ I_vw then
                N.append(t_i, sqrt(g(C(t_i), v, w)))
        for p ∈ F.points do
            N.append(p, sqrt(g(p, v, w)))
return max(N, x → x.second)
```

In order to prove the correctness of Algorithm 3, we first need two helping lemmas:

Lemma 2. *Given a graph G and a start node $s \in V$, it holds that $A \subseteq B$ for A, B as:*

$$A = \{q \in \mathbb{R}^d \mid |\text{ANN}_q^s| > 1\}$$
$$B = \{q \in \mathbb{R}^d \mid \exists u, v : q \in bisector(u, v)\}.$$

Proof. Let $q \in A$, if it is $|P_q^s| > 1$, let $p_1, p_2 \in P_q^s$. Let $(u, v) \in p_1$ and $(u, w) \in p_2$ be the first differing edges of p_1 and p_2. As both of these paths are contained in P_q^s they fulfill the following: $v, w \in \underset{z \in \{u\} \cup N(u)}{\arg\min} d(z, q)$, meaning they are equidistant from q. Hence it is $q \in bisector(v, w)$. If now $|P_q^s| = 1$, let $p \in P_q^s$ and $(s, v) \in p$ be the first edge of p. Again it holds that: $s, v \in \underset{z \in \{s\} \cup N(s)}{\arg\min} d(z, q)$. Hence it follows $q \in bisector(s, v)$. □

Lemma 3. *Given a graph G, a start node $s \in V$ and $u, v \in V$, $u \neq v$, then the following holds: $\mathring{\text{SC}}_u^s \cap \mathring{\text{SC}}_v^s = \emptyset$, where $\mathring{\cdot}$ denotes the interior of the set.*

Proof. Clearly, $\text{SC}_u^s \cap \text{SC}_v^s \subseteq A \subseteq B$ for A, B as in Lemma 2. As B is $d-1$ dimensional, $\text{SC}_u^s \cap \text{SC}_v^s$ is as well. Since S_u^s and S_v^s are sets of polytopes with holes as of Lemma 1, a $d-1$ dimensional intersection can only occur on boundaries. This implies $\mathring{\text{SC}}_u^s \cap \mathring{\text{SC}}_v^s = \emptyset$. □

Now we are ready to show our main theorem.

Theorem 4. *Given a graph G, a start node $s \in V$ and for all $u \in V$ it is $u \in \mathring{\text{SC}}_u^s$, then the following holds. For all $v, w \in V$ with $v \neq w$, the query $q \in I_{vw}$ of a cell of the intersection matrix IM_s maximizing $\hat{g}(q) = g(q, v, w)$ lies along the boundary of I_{vw}.*

Proof. Let now $v, w \in V$ with $v \neq w$. Firstly the node $w \notin \text{VC}_v$ and hence $w \notin I_{vw}$. Further $v \notin I_{vw}$, as if it were it would also be contained in SC_w^s, contradicting Lemma 3. We show that for $x \in I_{vw}$, $\nabla_q \hat{g}(x) \neq \mathbf{0}$, proving that I_{vw} does not contain an extreme point of $\hat{g}$ and thus the maximum must lie on the boundary. In a contradictory step we assume that there exists an $x \in I_{vw}$ such that $\nabla_q \hat{g}(x) = \mathbf{0}$. For $i \in \{1, \dots, d\}$ the i-th element of the gradient is as follows:

$$(\nabla_q \hat{g}(x))_i = \frac{2(x_i - w_i)d(x, v)^2 - 2(x_i - v_i)d(x, w)^2}{d(x, v)^4}.$$

As $x \neq v$ the denominator is never zero.

We consider two cases. The first one is that x is equidistant from v and w. Thus it is $z = 2d(x, w)^2 = 2d(x, v)^2$. Further it is $z \neq 0$ as $x \neq w$ and $x \neq v$. Let j be the index for which $v_j \neq w_j$. Now the numerator for the j-th partial can be arranged as $z((x_j - w_j) - (x_j - v_j))$. Clearly, $(x_j - w_j)$ cannot be equal to $(x_j - v_j)$, hence the expression cannot be zero, contradicting the assumption. The second case is that x is not equidistant from v and w. Let $j \in \{1, \dots, d\}$. For this case the numerator can be arranged as $2((x_j - w_j)e - (x_j - v_j)c)$, where $e = d(x, v)^2$ and $c = d(x, w)^2$. Again as $x \neq v$ and $x \neq w$ it is $e \neq 0$ and $c \neq 0$. The numerator can now only ever be zero if one of the following cases hold:

$$\begin{aligned}&((x_j - w_j) = 0) \wedge (x_j - v_j) = 0)\\ \vee&((x_j - w_j) = c \wedge (x_j - v_j) = e)\\ \vee&((x_j - w_j) = -c \wedge (x_j - v_j) = -e).\end{aligned}$$

Let now $k_w = |\{i \in \{1, \dots, d\} \mid x_i - w_i \neq 0\}|$ and $k_v = |\{i \in \{1, \dots, d\} \mid x_i - v_i \neq 0\}|$. Clearly $k_v = k_w$, as the above cases always appear together. It now is:

$$c = \sum_{i=1}^{d} (x_i - w_i)^2 = \sum_{i=1, x_i - w_i \neq 0}^{d} (x_i - w_i)^2$$

Since for the elements of the constrained sum it is $w_i = x_i \pm c$ and thus $(x_i - x_i + c)^2 = c^2 = (x_i - x_i - c)^2$, it is:

$$\sum_{i=1, x_i - w_i \neq 0}^{d} (x_i - w_i)^2 = k_w c^2 \Leftrightarrow c = k_w c^2 \Leftrightarrow c = \frac{1}{k_w}$$

As $x \neq w$ it clearly is $k_w \geq 1$. The same transformation can now be applied to e, leading to $c = \frac{1}{k_w} = \frac{1}{k_v} = e$, which is a contradiction as $c \neq e$. □

This completes the correctness proof of Algorithm 3. To not have to compute infinite VD cells, the following theorem captures how r behaves for points outside the bounding volume of a graph.

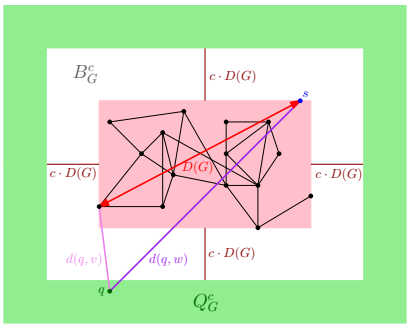


Figure 2: 2D representation of the area Q_G^c.

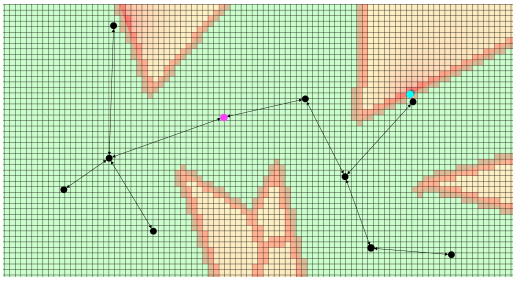


Figure 3: Execution of the discrete algorithm on an MST with 10 nodes.

Definition 8 (Padded bounding volume B_G^c)**.** *Given a graph G and its diameter $D(G)$, we define the padded bounding volume as follows:*

$$B_G^c = \prod_{i=1}^{d} \left[\left(\min_{v \in V} v_i \right) - cD(G), \left(\max_{v \in V} v_i \right) + cD(G) \right].$$

Theorem 5. *For all $c \in \mathbb{R}_{>0}$ and $q \in Q_G^c = \mathbb{R}^d \setminus B_G^c$, it holds that $\frac{d(q,w)}{d(q,v)} < 1 + \frac{1}{c}$, for $w \in \text{ANN}_q^s$ and $v \in \text{NN}_q$.*

Proof. From the triangle inequality it holds that $d(q,w) \leq d(q,v) + d(v,w)$. As $d(v,w) \leq D(G)$ it further is $d(q,w) \leq d(q,v) + D(G)$. From the definition of Q_G^c and as seen in Figure 2, $d(q,v) > cD(G)$. Hence the fraction can be transformed as follows:

$$\frac{d(q,w)}{d(q,v)} \leq \frac{d(q,v) + D(G)}{d(q,v)} = 1 + \frac{D(G)}{d(q,v)} < 1 + \frac{D(G)}{cD(G)}$$

which equals $1 + \frac{1}{c}$, concluding the theorem. □

Hence, for points that are arbitrarily far away from G the approximation ratio tends to 1. This allows to consider only a finite subspace of $\mathbb{R}^d$ for the computation of r.

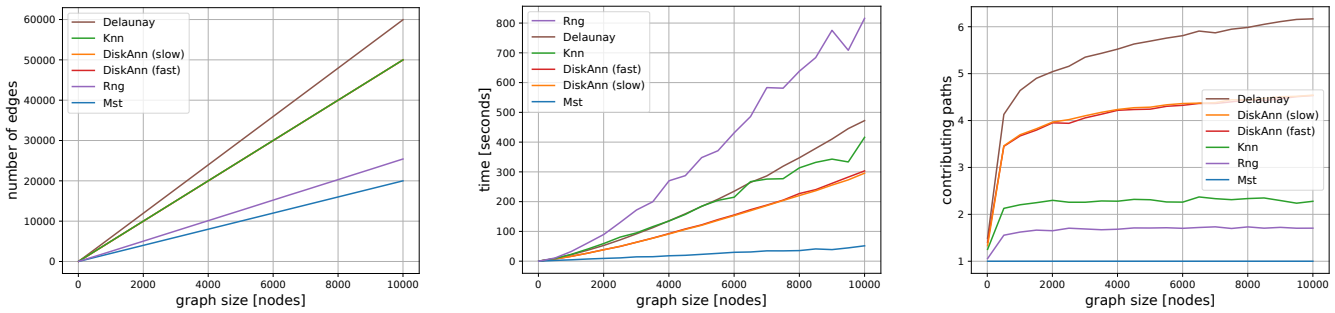
Discretization Algorithm

Our algorithm works for arbitrary d. However, a robust implementation for $d > 2$ is non-trivial, as it demands to compute multidimensional Voronoi diagrams and to intersect polytope sets in higher dimensions without running into numerical issues. To avoid this, we propose discretized version of Algorithm 3. As we know from Theorem 4, the worst query point will reside in a facet of the intersection matrix. The goal is now to find a discrete description of the subspace that contains these facets. The first step is to compute discrete versions of the VD and SPD. This is done by splitting the padded bounding volume of a graph into hypercubes and finding the NN^{q_c} and $\text{ANN}_s^{q_c}$ respectively, where q_c is the center of the current box. The intersection can then be computed by going over all boxes in the volume. For each box we check if the NN and the ANN are different for one of their isometrically aligned neighboring boxes (or the current box) and if they do not have only the same neighbors in those boxes. The first condition guarantees that only the red and yellow regions of the cover diagram are identified while the second condition further refines it to only consider the facets of the intersection matrix. After said computation we can sort all ratios for all boxes and only keep the m ones with the worst ratios, which subsequently can be used for sampling only from them, providing a reduced space that already contains bad query points. An exemplary execution of this algorithm can be found in Figure 3 for $d = 2$. It sampled 1000 points from the red boxes, and found a point almost identical to the actual worst query point of the instance.

Experiments

We evaluated our algorithms computing the search-path diagram and the worst-query point on six different graph types, namely minimum spanning trees (MST) (Munoz et al. 2019), k-nearest neighbor graphs (KNN) (Fu and Cai 2016), relative neighborhood graphs (RNG) (Jaromczyk and Toussaint 1992), DiskAnn (Subramanya et al. 2019) graphs, (Navarro 2002), as well as Delaunay graphs (acting as a sanity check and baseline), see Figure 4. For DiskAnn, the original version as well as a faster version starting out from a random regular-graph is implemented. Evaluated graph sizes range from 10 to 10000 nodes. The nodes were selected u.a.r. in $[0,1]^2$. If not stated otherwise, parameters for the KNN graphs are $k = 5$ and parameters for the DiskANN graphs are $\alpha = 1.15$, an out-degree bound of 5 and for the fast variant a queue length in its best-first-search procedure of 70. The start node will be the graphs centroid if not specified differently. As parameter for bounding box padding, we use $c = 0.2$. To combat numerical instability we used CGAL (The CGAL Project 2024) in our implementation. In order to speed up the computation of the intersection matrix in Algorithm 3 we further used the box intersections package of CGAL. All experiments were run on a single core of an AMD Ryzen Threadripper 3970X.

Approximable instances. Our first finding is that not every instance $I = (G, s)$ of a graph and a start node produces a finite approximation ratio r for every query point $q \in \mathbb{R}^d$. The reason for this is that if a node $v \in V$ is not contained in its search-path cell SC_v^s, the query $q = v$ does not have an approximation ratio and r tends to infinity for queries approaching v. An exemplary depiction of this situation is shown in Figure 5. Hence we deem an instance *approximable* if and only if all of its nodes lie in the interior of their search-path cells.



(a) (b) (c)

Figure 4: Depicted in (a) is the average number of edges. KNN and DiskANN graphs coincide in this plot as they have the same out-degree bound. Figure (b) shows the average search-path diagram construction time. Portrayed in (c) is the average number of contributing paths for non-empty cells.

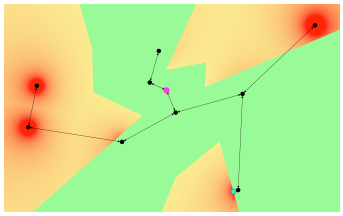
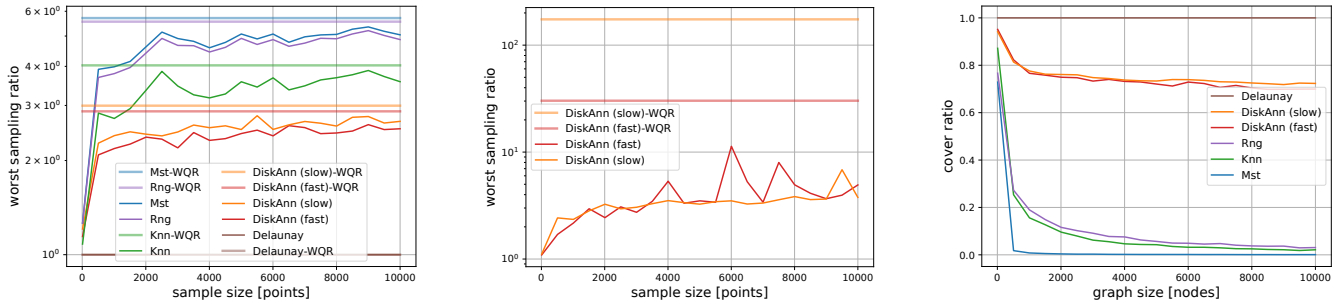


Figure 5: Example of a cover diagram of an MST of 10 nodes for the green start node. As there are nodes of the graph that do not fall in their search-path cell, this instance is *not approximable*. The light blue point is the result of Algorithm 3 despite the existence of aforementioned nodes (indicated by the red border).

SPD construction time. To measure the trade-off between sparsity of a graph G and its ANNS quality, Figure 4a gives us an overview of the amount of directed edges in the analyzed graphs. The construction time of the SPD is dependent on the choice of the base graph as can be observed in Figure 4b. The MST is performing well in this regard as it is sparse and acyclic and hence the SPD does not need to explore many simple paths. In general, the construction time is within a few minutes for all considered base graphs with up to 10000 nodes, and can be greatly sped up if the cells are computed in parallel. In theory the number of simple paths that need to be explored per cell may be exponential in n. However, Algorithm 2 is very far from reaching this bound. Figure 4c shows that the total number of contributing paths per search-path cell is very small and grows only lightly with the graph size. This confirms that most paths do not monotonically converge toward the destination vertex defining the cell and thus the exploration of paths can be stopped early, ensuring a practical running time.

Quality Assessment for $d = 2$. As discussed above, one of the most commonly adopted techniques for evaluating the ANNS performance of a graph is sampling a set of points from the query space and reporting the returned ratios. However, we show that this approach is not very successful in coming close to the worst query ratios, even on *approximable* instances. In our first experiment, we constructed graphs G on only 10 nodes, computed the true value of r with our new algorithm and then computed the extended bounding box B_G^c which is guaranteed to contain the worst query point as shown in Theorem 5. Subsequently, we sample up to 10000 query points from B_G^c per instance. Figure 6a shows the results for all graph types. We observe that even for these small graphs and substantial sample sizes, the sampling based estimated r values do not match the true one. For larger input graphs, this effect increases tremendously as shown in Figure 6b for the two considered variants of DiskAnn graphs. Even if the sample size is increased such that the execution time of Algorithm 3 and sampling match ($3 \cdot 10^6$ points), the ratio observed is still up to a factor of 5 from the true worst query ratio. For the other graph types, the percentage of approximable instances shrank dramatically with growing graph size, which sampling could not detect. The DiskAnn graphs turned out to be robust in that regard. But we clearly see in Figure 6b that even sampling 10000 points does not give us an estimated r value that is remotely close to the worst query ratio. The gap is more than an order of magnitude. We likewise discovered that the order of the worst query ratios can arbitrarily toggle between the DiskAnn variants given different graph sizes. For example, for 9500 samples, the slow variant appears to be superior and for 6000 and 10000 samples the fast variant. We thus conclude that computing the precise value of r with our algorithm can help to assess the quality of different methods in a more robust and systematic way.

But actually we can get more information from our SPD data structure than just r. We can also efficiently determine for which cells in the intersection of the SPD and the VD the true NN is returned. We refer to the respective part of the intersection as cover diagram (represented by the diagonal of the intersection matrix, see Definition 7) and define the cover ratio as the fraction of B_G^c which is part of the cover diagram. This measure can be interpreted as a kind of global recall. The cover diagram can provide valuable insight into the quality of a given instance I by showing us structurally degenerate regions for ANNS as depicted in Figure 5. Evaluating the cover ratio in Figure 6c, we observe that DiskAnn instances appear to be the most structurally sound, leveling out at a cover ratio of about 0.7. RNG, KNN and MST instances perform rather poorly with a cover ratio of $0 - 0.03$,



(a) (b) (c)

Figure 6: Depicted in (a) is the average worst sampling ratio for approximable instances of size 10. The opaque lines are the upper bound, namely the true average worst query ratio. (b) shows the average worst sampling ratio for approximable instances of size 10000. Portrayed in (c) is the average cover ratio for the different graph types.

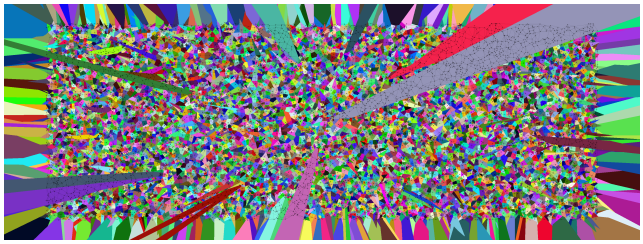


Figure 7: Exemplary SPD of a DiskAnn graph with 10000 nodes, where the center node is the start node. Our SPD data structure is capable of identifying degenerate areas as represented by the large cells.

meaning these instances return the correct NNs for only up to three percent of the space in B_G^c. This is due to their low connectivity (e.g. KNN might not be strongly connected) and the likelihood of getting stuck in local optima.

Another use case of our algorithm is illustrated in Figure 7. Here, one can clearly see large cells in the SPD that contain a lot of graph structure. This shows that many nodes in the graph are never reached on monotone paths from the start node and are thus never returned as NN of any query. Inserting additional edges to better connect these graph regions could help to significantly improve the result quality.

Quality Assessment for Higher Dimensions. To demonstrate the usefulness of our algorithm in higher dimensions, we implemented the discretized version and tested it for $d = 2, \ldots, 10$ on the DiskAnn graphs with 10^6 hypercubes. Figure 8 shows the results for $d = 3$. We clearly see that restricting the sampling to the hypercube facets identified by our algorithm leads to a significantly larger r already for small sample sizes but even more so for larger ones. The same behavior was observed for larger dimensions. However, as we keep the number of hypercubes the same, the discretization is coarser in higher dimensions. Still, our algorithm produces r-values that are a factor of 7 larger for $d = 5$ and a factor of 2-3 for $d = 10$. Using a finer discretization the gaps are expected to increase further. Moreover, the categorization of hypercubes in those that are covered well and those that are not can also help to guide graph structure improvements.

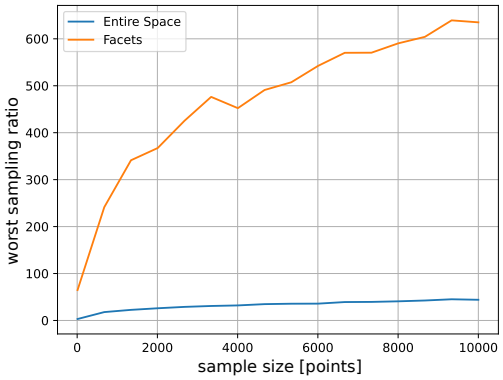


Figure 8: Average ratios produced by classical sampling versus sampling restricted to bad regions (facets) identified with our algorithm for DiskANN graphs in $\mathbb{R}^3$ with 10^5 nodes.

Conclusions and Future Work

This work is supposed to be a first step towards a more systematic evaluation of graph-based ANN search methods. While sampling based evaluation has the advantage of being easy to implement, we demonstrated that solely relying on sampled queries to assess the approximation quality might give a distorted picture. Our approach efficiently identifies regions with poor ANN quality and thus can also be helpful for improving existing graph construction methods.

One direction for future work is to make our algorithm applicable to other types of search algorithms. We focused here on greedy path traversal. But more intricate methods, as e.g. beam search (Prokhorenkova and Shekhovtsov 2020), could be considered as well. Indeed, other search methods could be easily incorporated into our proposed framework as long as they rely on path traversals which can be captured by our SPD data structure.

Finally, more general query types as k-(A)NN queries could be taken into account. Either by considering the worst case ratio between the computed k^{th} NN and the true one, or by using measures as precision and recall. For the latter, the search-path diagram would need to be integrated with a suitable range search data structure.

References

Abbasifard, M. R.; Ghahremani, B.; and Naderi, H. 2014. A survey on nearest neighbor search methods. *International Journal of Computer Applications*, 95(25).

Arya, S.; and Mount, D. M. 1993. Approximate nearest neighbor queries in fixed dimensions. In *SODA*, volume 93, 271–280. Citeseer.

Aumüller, M.; Bernhardsson, E.; and Faithfull, A. 2020. ANN-Benchmarks: A benchmarking tool for approximate nearest neighbor algorithms. *Information Systems*, 87: 101374.

Cao, Y.; Qi, H.; Zhou, W.; Kato, J.; Li, K.; Liu, X.; and Gui, J. 2017. Binary hashing for approximate nearest neighbor search on big data: A survey. *IEEE Access*, 6: 2039–2054.

Chen, P.; Chang, W.-C.; Jiang, J.-Y.; Yu, H.-F.; Dhillon, I.; and Hsieh, C.-J. 2023. Finger: Fast inference for graph-based approximate nearest neighbor search. In *Proceedings of the ACM Web Conference 2023*, 3225–3235.

Dasgupta, S.; and Freund, Y. 2008. Random projection trees and low dimensional manifolds. In *Proceedings of the ACM symposium on Theory of computing*, 537–546.

Friedman, J. H.; Bentley, J. L.; and Finkel, R. A. 1977. An algorithm for finding best matches in logarithmic expected time. *ACM Transactions on Mathematical Software (TOMS)*, 3(3): 209–226.

Fu, C.; and Cai, D. 2016. Efanna: An extremely fast approximate nearest neighbor search algorithm based on knn graph. *arXiv preprint arXiv:1609.07228*.

Fu, C.; Xiang, C.; Wang, C.; and Cai, D. 2017. Fast approximate nearest neighbor search with the navigating spreading-out graph. *arXiv preprint arXiv:1707.00143*.

Gollapudi, S.; Karia, N.; Sivashankar, V.; Krishnaswamy, R.; Begwani, N.; Raz, S.; Lin, Y.; Zhang, Y.; Mahapatro, N.; Srinivasan, P.; et al. 2023. Filtered-diskann: Graph algorithms for approximate nearest neighbor search with filters. In *Proceedings of the ACM Web Conference*, 3406–3416.

Jaromczyk, J. W.; and Toussaint, G. T. 1992. Relative neighborhood graphs and their relatives. *Proceedings of the IEEE*, 80(9): 1502–1517.

Kleinbort, M.; Salzman, O.; and Halperin, D. 2020. Collision detection or nearest-neighbor search? On the computational bottleneck in sampling-based motion planning. In *Proceedings of the Twelfth Workshop on the Algorithmic Foundations of Robotics*, 624–639. Springer.

Kramer, O. 2013. K-nearest neighbors. *Dimensionality reduction with unsupervised nearest neighbors*, 13–23.

Lee, D.-T.; and Wong, C.-K. 1977. Worst-case analysis for region and partial region searches in multidimensional binary search trees and balanced quad trees. *Acta Informatica*, 9(1): 23–29.

Lei, D.; and Chen, X. 2022. An improved variable neighborhood search for parallel drone scheduling traveling salesman problem. *Applied Soft Computing*, 127: 109416.

Lei, Y.; Huang, Q.; Kankanhalli, M.; and Tung, A. 2019. Sublinear time nearest neighbor search over generalized weighted space. In *International Conference on Machine Learning*, 3773–3781. PMLR.

Li, W.; Zhang, Y.; Sun, Y.; Wang, W.; Li, M.; Zhang, W.; and Lin, X. 2019. Approximate nearest neighbor search on high dimensional data—experiments, analyses, and improvement. *IEEE Transactions on Knowledge and Data Engineering*, 32(8): 1475–1488.

Liu, J.; Zhu, Z.; Hu, J.; Sun, H.; Liu, L.; Liu, L.; Dai, G.; Yang, H.; and Wang, Y. 2022. Optimizing Graph-based Approximate Nearest Neighbor Search: Stronger and Smarter. In *2022 23rd IEEE International Conference on Mobile Data Management (MDM)*, 179–184. IEEE.

Munoz, J. V.; Gonçalves, M. A.; Dias, Z.; and Torres, R. d. S. 2019. Hierarchical clustering-based graphs for large scale approximate nearest neighbor search. *Pattern Recognition*, 96: 106970.

Navarro, G. 2002. Searching in metric spaces by spatial approximation. *The VLDB Journal*, 11: 28–46.

Prokhorenkova, L.; and Shekhovtsov, A. 2020. Graph-based nearest neighbor search: From practice to theory. In *International Conference on Machine Learning*, 7803–7813.

Ren, J.; Zhang, M.; and Li, D. 2020. Hm-ann: Efficient billion-point nearest neighbor search on heterogeneous memory. *Advances in Neural Information Processing Systems*, 33: 10672–10684.

Rubinstein, A. 2018. Hardness of approximate nearest neighbor search. In *Proceedings of the 50th annual ACM SIGACT symposium on theory of computing*, 1260–1268.

Singh, A.; Subramanya, S. J.; Krishnaswamy, R.; and Simhadri, H. V. 2021. Freshdiskann: A fast and accurate graph-based ann index for streaming similarity search. *arXiv preprint arXiv:2105.09613*.

Subramanya, S. J.; Devvrit; Kadekodi, R.; Krishaswamy, R.; and Simhadri, H. V. 2019. DiskANN : Fast Accurate Billion-point Nearest Neighbor Search on a Single Node.

The CGAL Project. 2024. *CGAL User and Reference Manual*. CGAL Editorial Board, 5.6.1 edition.

Wang, M.; Xu, X.; Yue, Q.; and Wang, Y. 2021. A comprehensive survey and experimental comparison of graph-based approximate nearest neighbor search. *arXiv preprint arXiv:2101.12631*.

Wang, P.; Reinelt, G.; and Tan, Y. 2012. Self-adaptive large neighborhood search algorithm for parallel machine scheduling problems. *Journal of Systems Engineering and Electronics*, 23(2): 208–215.

Yershova, A.; and LaValle, S. M. 2007. Improving motion-planning algorithms by efficient nearest-neighbor searching. *IEEE Transactions on Robotics*, 23(1): 151–157.

Yianilos, P. N. 1993. Data structures and algorithms for nearest neighbor search in general metric spaces. In *Soda*, volume 93, 311–21.

Zhao, X.; Tian, Y.; Huang, K.; Zheng, B.; and Zhou, X. 2023. Towards efficient index construction and approximate nearest neighbor search in high-dimensional spaces. *Proceedings of the VLDB Endowment*, 16(8): 1979–1991.