Almost Navigable Graphs

Pratyush Avi
New York University
pratyushavi@nyu.edu

Christopher Musco
New York University
cmusco@nyu.edu

ABSTRACT

Graph-based methods like HNSW, DiskANN, NSG, and others have become an increasingly popular choice for implementing approximate nearest neighbor search (ANNS) in Vector Databases (VecDBs). The success of these methods has motivated the study of how to best construct a search graph for a given dataset. To that end, *navigability* has been identified as a desirable graph property which ensures good ANNS performance when combined with greedy search.

However, for a dataset with n vectors, the sparsest navigable graph requires $O(n\sqrt{n})$ edges in the worst-case, and we show empirically that, for typical billion node datasets, 100s of edges are needed per node. This leads to slow search and high memory requirements. Moreover, under standard complexity theoretical assumptions, it was recently established that constructing a sparse navigable graph requires $\Omega(n^{2-\epsilon})$ time, which is prohibitive for large datasets.

We address these concerns by introducing a relaxed notation of navigability called "γ-almost navigability" for any $\gamma \in [0, 1]$, with $\gamma = 1$ corresponding to full navigability. We prove that any dataset (under any distance) admits a γ-almost navigable graph with just $O\left(\frac{n}{1-\gamma}\right)$ edges, linear in the dataset size. We present a randomized algorithm for constructing such a graph in near-linear time.

While we prove that γ-almost navigability sacrifices the worst-case search guarantees enjoyed by navigability, we show empirically that greedy beam search still performs well in such graphs when $\gamma < 1$. Indeed, we obtain improved recall-runtime tradeoffs on a variety of datasets compared to fully navigable graphs. Moreover, our graphs are more space efficient, with degree typically less than half that of a fully navigable graph for comparable performance.

VLDB Workshop Reference Format:
Pratyush Avi and Christopher Musco. Almost Navigable Graphs. VLDB 2026 Workshop: The 2nd Workshop on Vector Databases.

1 INTRODUCTION

Approximate nearest neighbor search (ANNS) is the central algorithmic problem underlying vector databases (VecDBs). Formally, we are given a set of points $P = \{p_1, \ldots, p_n\} \subset \mathbb{R}^m$ and a distance function $d : \mathbb{R}^m \times \mathbb{R}^m \to \mathbb{R}$. The goal is to preprocess the points into a data structure so that, given any query vector $q \in \mathbb{R}^m$, we can efficiently find the k-nearest neighbors of q in P, or at least k points that are near-minimizers of $d(x, q)$ over $x \in P$.

For standard distance functions, the complexity of classical approaches for the ANN problem, like k-d trees, scales exponentially in the data dimension, m. This has motivated the a wide variety of alternative approaches specifically targeted at high-dimensional datasets, including locality-sensitive hashing [2, 3, 20, 29], product quantization/clustering methods [14, 22, 23], and tree-based methods [4, 7, 8]. More recently, *graph-based methods* have emerged as a popular option for high-dimensional search, performing well on ANN benchmarks and competitions [6, 32, 36, 37]. Such methods build on ideas dating back to the 1990s [5, 10, 27, 34], and include the Hierarchical Navigable Small World Graph method (HNSW) [31], Microsoft's DiskANN [28, 38], the Navigating Spreading-out Graph method (NSG) [16], and others [15, 19, 30].

While graph-based methods vary in details, they follow a common theme: a directed graph G is constructed with one node corresponding to every point in P. Queries are serviced by running greedy search on the graph. In particular, we start at some $p \in P$, then move to p's out-neighbor in G that gets us closest to q, repeating this process until no further improvement is possible. Typically, a "back-tracking" variant of greedy search called *beam search* is used in practice to avoid getting stuck in local minima.

1.1 Graph Navigability

Given the simplicity of the greedy search process, a key differentiator between graph-based methods is how the search graph G is constructed.[1] There is a trade-off: higher degree graphs typically yield more accurate results, but also take more space to store. Moreover, since the per-iteration cost of greedy search scales with the degree of the current node, high degree leads to slower search time.

While most existing constructions are heuristic, there has been recent interest in formally defining desirable search graph properties, and then separately designing algorithms to construct *sparse graphs* with those properties [12, 21, 25]. Of particular interest is the property of *navigability*, which dates back to work on Stanley Milgram's "small-world" phenomenon [9, 11, 26, 33, 39]. Informally, navigability means that, if a simple greedy search is initialized at any point $p \in P$, it will successfully find any "in-distribution" query $q \in P$ (if q is in P, it is of course its own nearest neighbor).

Under the standard assumption that distances between points in P are unique[2], navigability has the following equivalent definition:

Definition 1 (Navigable Graph)**.** Let $P = \{p_1, \ldots, p_n\}$ be a set of points and let $d : P \times P \to \mathbb{R}^{\geq 0}$ be any distance function satisfying $d(p, p) = 0$ for all $p \in P$ and $d(p, r) > 0$ for all $p, r \in P$, $p \neq r$.

A graph $G = (P, E)$ is *navigable* if, for all pairs $p, r \in P$, $p \neq r$, there is a directed edge $(p, s) \in E$ such that $d(s, r) < d(p, r)$.

In words, navigability demands that every node p has some out-edge that brings it closer to any other node r in the dataset. That

Proceedings of the VLDB Endowment. ISSN 2150-8097.

[1] Some methods, like HNSW, actually utilize a hierarchy of multiple search graphs, running greedy search at each level of the hierarchy.
[2] Without this assumption, "greedy search" is not fully specified unless we define a tie-breaking rule. It is simpler to assume unique distances, as this can be ensured, e.g., by adding an arbitrarily small random perturbation to the points in P.

out-edge could be to r itself, so we observe that the complete graph is trivially navigable.

Navigability is an attractive property for a number of reasons. First, a desirable property of any ANN method is that, for any query q, the method returns some approximate nearest neighbor $\tilde{x}$ satisfying $d(\tilde{x}, q) < C \cdot \min_{x \in P} d(x, q)$ for some approximation factor $C > 1$. This is the guarantee promised, e.g., by locality sensitive hashing methods. Navigability ensures that this guarantee at least holds for all $q \in P$: in this case, the right hand side is 0, so greedy search must return q itself to obtain a multiplicative approximation.

Moreover, while navigability alone does not ensure good accuracy when $q \notin P$, natural strengthenings of the definition do, including α-reachability [17, 21, 38] and τ-monotonicity [35]. Multiplicative approximation can also be ensured by combining a navigable graph with the Adaptive Beam Search algorithm from [1].

Finally, beyond theoretical motivation, navigability is relevant practically. Many existing methods target the construction of navigable graphs [38], and indeed the property lends its name to popular methods like the Hierarchical Navigable Small World Graph method (HNSW) and the Navigating Spreading-out Graph method (NSG).

Limitations of Navigability. Despite its current importance in the literature on graph-based search, navigability has a number of limitations. First, it was recently established that, for any distance function, all datasets admit a navigable graph with $O(n\sqrt{n})$ edges [12]. While far sparser than the complete graph, this bound is also known to be tight, even for random points in $O(\log n)$ dimensions [13]. Sparse graphs can be obtained under further data assumptions like bounded doubling dimension [18, 21], but we find that, for modern billion node datasets, constructing a navigable graph requires hundreds of edges per node, which is much higher than the degree of search graphs used in practice (see Section 3 for details).

Beyond high degree requirements, which drive up index size and search time, constructing a navigable graphs is expensive. While recent progress shows how to build near-optimally sparse navigable graphs in $\tilde{O}(n^2)$ time, it was also established that no faster runtime is possible under the Strong Exponential Time Hypothesis. Even for points in $O(\log n)$ dimensional Euclidean space, for any constant ϵ, constructing a navigable graph with $o(n^{2-\epsilon})$ edges (only slightly better than the complete graph) requires $\Omega(n^{2-\epsilon})$ time [12, 25].

1.2 Our Contributions

Given the above limitations, it seems natural to consider *relaxed* variations on the navigability property that allow for sparser graphs, faster construction times, or both. Indeed, few practical graph-based methods construct *truly navigable graphs*. Instead, it is common to use fast heuristics that ideally return something "close" to navigable. An important example is Microsoft's DiskANN method. If initialized with a complete graph the "Robust Prune" algorithm introduced in that work would indeed return a navigable graph. However, doing so is too slow, so a heuristic initialization is used instead [38].

Nevertheless, to the best of our knowledge, there have not been efforts to quantify how well existing methods approximate the navigability property, or to design methods targeting a specific notation of approximation. In this work, we address that gap by introducing and studying one possible relaxation of navigability. In particular, we define the notation of γ-almost navigability:

Definition 2 (γ-Almost Navigable Graph)**.** Let $P = \{p_1, \ldots, p_n\}$ be a set of points and let $d : P \times P \to \mathbb{R}^{\geq 0}$ be any distance function satisfying $d(p, p) = 0$ for all $p \in P$ and $d(p, r) > 0$ for all $p, r \in P$, $p \neq r$. For a direct graph $G = (P, E)$, let

$$C_p = \{r \in P \mid \exists (p, s) \in E \text{ such that } d(s, r) < d(p, r)\}.$$

For a parameter $\gamma \in [0, 1]$, G is *γ-almost navigable* if for every point $p \in P$, $|C_p| \geq \gamma \cdot (n - 1)$.

In words, γ-almost navigability demands that every node has an out edge that takes it closer to a γ fraction of the other nodes in the dataset. Setting $\gamma = 1$ recovers navigability. The definition becomes easier to satisfy for smaller γ.

Our first result is that this natural relaxation significantly reduces the worst-case density required to construct a search graph. In contrast to fully navigable graphs, which require $\Omega(n^{3/2})$ edges [13], we show that any data set, under any distance function, admits a γ-almost navigable graph with just $O(n)$ edges. The leading constant scales with $1/(1 - \gamma)$.

Theorem 1. *For any point set P, distance function $d : P \times P \to \mathbb{R}^{\geq 0}$, and $\gamma \in [0, 1)$, there exists a γ-almost navigable graph with average out-degree at most $\frac{4}{1-\gamma}$.*

Theorem 1 is proven in Section 2.1 via a constructive argument. We show how to build a linear-sized almost navigable graph using a "clique peeling'we' method that is reminiscent of recent techniques used to construct fully navigable graphs [12]. We start by partitioning P into arbitrary sets of $O(1/(1 - \gamma))$ points. For each such set, S, we add a clique to the search graph G. After doing so, any $p \in S$ has an edge closer to all points in the dataset *except those for which it is the closest point in S*. On average, each point $p \in S$ is closest to $(1 - \gamma)n$ points, so an expectation argument shows that at least half of the points in S have edges closer to a γ fraction of points in P. I.e., half of the points satisfy the constraint of Definition 2.

We can then repeat the construction, arbitrarily grouping any remaining points that do not satisfy the constraints of Definition 2 and adding a new set of cliques. After $O(\log n)$ rounds, all points will have edges close to a γ fraction of points in P, as required.

Naively, the above procedure runs in $O(n^2)$ time, as we need to compute distances between all pairs of points to determine which points have satisfied the γ-almost navigability requirement, and which points should continue to the next round of the algorithm. However, if we introduce randomization, this cost can be reduced: checking a sample of roughly $O(1/(1 - \gamma))$ points suffices to determine, with high probability, if a given points p has an edge closer to a γ-fraction of points in the dataset. This leads to an algorithm whose runtime scales *linearly* instead of quadratically with n:

Theorem 2. *There is an algorithm that, given any point set P, distance function $d : P \times P \to \mathbb{R}^{\geq 0}$ computable in time T, $\gamma \in [0, 1)$, and $\delta \in (0, 1)$, constructs a γ-almost navigable graph with average out-degree $O\left(\frac{1}{1-\gamma}\right)$ in $O\left(\frac{nT \log(n/\delta)}{1-\gamma}\right)$ time, with prob. at least $1 - \delta$.*

Theorem 2 is proven in Section 2.2. Combined with Theorem 1, it establishes polynomial improvements in graph size and construction time over fully navigable graphs. In particular, $O(n^{3/2})$ size and $\tilde{O}(n^2)$ construction time are reduced to $O(n)$ and $\tilde{O}(n)$, respectively, when we relax the definition to γ-almost navigability.

Value for Greedy Search. Given these efficiency gains, it remains to determine is γ-almost navigable graphs preserve the performance of fully navigable graphs for approximate nearest neighbor search. We explore this question both theoretical and empirically.

On the theoretical size, we show a negative result in Section 2.3. Recall that a desirable property of navigability is that greedy search run on a navigable graph is at least guaranteed to correctly answer any "in-distribution" query, $q \in P$. This is a necessary property for good ANN performance, even if it is not sufficient in the worst-case.

It is natural to ask if γ-almost navigability leads to a relaxed version of this guarantee. For example, we might hope that greedy search correctly answers a large fraction of in-distribution queries. Unfortunately, we show that this is not the case, even for γ very close to 1, and even for points in low-dimensional Euclidean space: we construct a family of 2D point sets and corresponding γ-almost navigable graphs so that greedy search fails on all but $1/(1-\gamma)$ in-distribution queries, no matter how large n is.

Our work on the empirical side, however, is more positive. We show that, on standard ANN benchmarking datasets, γ-almost navigable graphs achieve similar search quality at much lower search and graph storage costs in comparison to fully navigable graphs. For example, for a fixed target accuracy of 90% recall, we find that γ-almost navigable graphs require about 50% less storage space and incur just 35-47% of the search costs of navigable graphs. Section 3 contains more details about the practical performance of almost navigable graphs and our experimental setup.

2 ALMOST NAVIGABLE GRAPHS

In this section, we restate and prove our main theoretical results on the existence and construction of sparse *almost* navigable graphs (Definition 2). We also provide a construction showing that, unfortunately, almost navigable graphs do not enjoy the same worst-case greedy search guarantees as fully navigable graphs. Nevertheless, as shown in the next section, these graphs still appear empirically useful for greedy graph-based ANN search.

2.1 Bounded Degree Almost Navigable Graphs

Our first result is that, for fixed γ, it is always possible to construct a γ-almost navigable graph whose edge count grows just linearly in the dataset size. This contrasts with the necessary $O(n^{3/2})$ edges required to construct a fully navigable graph in the worst-case [13]:

Theorem 1. *For any point set P, distance function $d : P \times P \to \mathbb{R}^{\geq 0}$, and $\gamma \in [0, 1)$, there exists a γ-almost navigable graph with average out-degree at most $\frac{4}{1-\gamma}$.*

Our proof is constructive: we show how to build a γ-almost navigable graph with average out-degree $O(1/(1-\gamma))$ using a technique similar to the navigable graph construction from [12]. We make the construction efficient (near linear time) in Section 2.2.

Specifically, our construction leverages what [12] calls the "power of cliques" idea. We begin by partitioning our dataset P into *arbitrary* sets of size $\lceil 2/(1-\gamma) \rceil$. For each such set S, we add a clique to G, i.e. we add a directed edge from u to v for all $u, v \in S$. While doing so only adds $O(n/(1-\gamma))$ edges to the graph, we claim that after adding these cliques, the requirements of γ-almost navigability are satisfied for at least half of the points in P. Concretely:

Claim 3. *Let P be set of n points and suppose $G = (P, E)$ contains a clique connecting all nodes in some subset $S \subseteq P$. For all $v \in S$, define:*

$$U_{S,v} := \{p \in P \mid v = \arg\min_{x \in S} d(p, x)\}$$

Then, for at least half the points in S, have $|U_{S,v}| \leq 2n/|S|$.

Observe that, for all $v \in S$, for all $r \notin U_{S,v}$, there is some edge $(v, s) \in E$ such that $d(s, r) < d(v, r)$: simply take s to be the point in S that is closest to r. This means that the number of nodes in $P \setminus v$ that v does not have an edge closer to is at most $|U_{S,v}| - 1$.[3] If we set $|S| = O(1/(1-\gamma))$, Claim 3 ensures that $|U_{S,v}| \leq (1-\gamma)n$. We can conclude that at least half of the points in S have edges closer to a γ fraction of the remaining points in P, as required by Definition 2.

Proof of Claim 3. Observe that the "ownership" sets $U_{S,v}$ partition the dataset P. We thus have:

$$\frac{1}{|S|} \sum_{v \in S} |U_{S,v}| = \frac{|P|}{|S|} = \frac{n}{|S|}.$$

Using the simple fact that for positive numbers, the median is at most twice the mean, the median sized set has size at most $2n/|S|$. Therefore, for at least half of $v \in S$ have $|U_{S,v}| \leq 2n/|S|$. □

Claim 3 provides a simple way of ensuring that the majority of points in P satisfy the requirements of γ-almost navigability. We obtain our main result by applying this idea iteratively to construct a full γ-almost navigable graph. After the first round of adding cliques to G, we set aside all points that satisfy the γ-almost navigability requirements. We then group any remaining points that do not satisfy the constraint into a new set of cliques. After doing so, we will again have that half of the remaining points are satisfied. Continuing in this way, we will have a γ-almost navigable graph after $O(\log n)$ rounds. We formally analyze this procedure below:

Proof of Theorem 1. Specifically, consider the following procedure for constructing a graph $G = (P, E)$:

(1) Arbitrarily partition P into subsets $S_1, \ldots, S_k$ of size $\lceil 2/(1-\gamma) \rceil$ each and a set $\bar{S}$ with $< \lceil 2/(1-\gamma) \rceil$ left over points.
(2) Add a clique to each S_i.
(3) Let $\bar{P}$ contain all points in $\bar{S}$ and, for every S_i, all $v \in S_i$ such that $|U_{S_i,v}| > (1-\gamma)n$.
(4) If $|\bar{P}| \geq \lceil 2/(1-\gamma) \rceil$, remove all out-edges from every $v \in \bar{P}$ and repeat from Step 1 with points in $\bar{P}$.
(5) Otherwise, add an edge from the $< \lceil 2/(1-\gamma) \rceil$ points in $\bar{P}$ to every point in P.

We first establish correctness of the above procedure. First, observe that, by Claim 3, the procedure terminates, and indeed terminates after at most $O(\log n)$ rounds. In particular, by Claim 3, for each S_i, at least $|S_i|/2$ points $v \in S_i$ satisfy:

$$|U_{S_i,v}| \leq \frac{2n}{|S_i|} \leq (1-\gamma)n.$$

Even counting the left over points in $\bar{S}$, it follows that the size of $\bar{P}$ shrinks by at least a factor of $1/4$ at every iteration of the procedure.

Having established convergence, consider any point $v \in P$. If v remains in $\bar{P}$ until the last step of the algorithm, then v clearly has

[3] v itself is in $U_{S,v}$, so there are only $|U_{S,v}| - 1$ points from $P \setminus v$ in $U_{S,v}$.

edges satisfying the γ-almost navigability requirements of Definition 2, as it is connected to all of P.

If not, then at some point in the algorithm, v was placed in a subset S_i and it was determined that $|U_{S_i,v}| \leq (1-\gamma)n$. When this happens, an edge is added from v to every point in S_i and is never removed. v thus has an edge closer to any $r \in P$ that is not in $U_{S_i,v}$. We conclude that v *does not* have an out edge closer to at most $|U_{S_i,v} \setminus \{v\}| \leq (1-\gamma)n-1 \leq (1-\gamma)(n-1)$ points in P. The γ-almost navigability requirement of Definition 2 thus holds for v.

Finally, we consider the total number of edges in G. We have at most $\left\lfloor \frac{2}{1-\gamma} \right\rfloor$ points in $\bar{P}$ at the end of the procedure, each with $n-1$ edges. Any other points v that is not in $\bar{P}$ at the end of the procedure has exactly $\lceil 2/(1-\gamma) \rceil$ edges. So in total we have:

$$|E| < \left\lfloor \frac{2}{1-\gamma} \right\rfloor \cdot (n-1) + \left\lceil \frac{2}{1-\gamma} \right\rceil \cdot n \leq \frac{4}{1-\gamma} \cdot n. \quad \square$$

It is interesting to ask if Theorem 1 can be improved. Notably, in the extreme case when $\gamma > 1-1/\sqrt{n}$, we have $O(n/(1-\gamma)) > O(\sqrt{n})$, so we can actually obtain a better bound on the degree required for γ-almost navigability by appealing to the existing upper bounds of $O(\sqrt{n})$ for the average degree of a *fully* navigable graph. This suggests that it might be possible to obtain an improved dependence on γ. We note that a bound of the form $O(n/\sqrt{1-\gamma})$ is not possible, however, as it can be checked that the lower bound of [13] for navigable graph construction extends to the setting when only a $1 - O(1/\sqrt{n})$ fraction of constraints need to be satisfied. I.e., a $(1 - O(1/\sqrt{n}))$-almost navigable graph requires $\tilde{\Omega}(\sqrt{n})$ degree in the worst case, even for points in Euclidean space.

2.2 Almost Navigable Graphs in Linear Time

We next establish that, in contrast to fully navigable graphs, sparse almost-navigable graphs can be constructed in sub-quadratic time (actually, in linear time). We restate the formal result below:

Theorem 2. *There is an algorithm that, given any point set P, distance function $d : P \times P \to \mathbb{R}^{\geq 0}$ computable in time T, $\gamma \in [0,1)$, and $\delta \in (0,1)$, constructs a γ-almost navigable graph with average out-degree $O\left(\frac{1}{1-\gamma}\right)$ in $O\left(\frac{nT\log(n/\delta)}{1-\gamma}\right)$ time, with prob. at least $1-\delta$.*

This result is proven via an efficient implementation of the procedure described in the proof of Theorem 1. The most expensive operation in each round of that procedure is computing the "ownership" set of each point $v \in S_i$, $U_{S_i,v}$. Exactly computing this set requires pairwise distance computations between every v with every other point in P, resulting in $O(n^2)$ distance computations.

However, we do not actually need $U_{S_i,v}$ itself: we only need to know the *size* of the set to determine if v proceeds to the next round of the procedure. The size $|U_{S_i,v}|$ can be approximated more efficiently based on a random sample of data points from P. In particular, we argue that $\tilde{O}(1/(1-\gamma))$ samples from P suffice by a standard Chernoff bound. The distance computations from each $v \in P$ to these samples dominates the runtime. Our approach is formalized in Algorithm 1 and analyzed in detail below:

Algorithm 1 Almost navigable graph construction

Input: Set of n points P, blackbox access to distance function $d : P \times P \to \mathbb{R}^{\geq 0}$, $\gamma \in [0,1)$, failure probability $\delta \in (0,1)$.
Output: γ-navigable graph $G = (P, E)$ with probability $1-\delta$.

Initialize $E \leftarrow \emptyset$, $\Pi^{(0)} \leftarrow P$, $i \leftarrow 0$, $w \leftarrow \frac{16\log(n/\delta)}{1-\gamma}$.
while $|\Pi^{(i)}| \geq \lceil 4/(1-\gamma) \rceil$ **do**
  Arbitrarily partition $\Pi^{(i)}$ into subsets $S_1, \ldots, S_k$ of size $\left\lfloor \frac{4}{1-\gamma} \right\rfloor$ and a set $\bar{S}$ with $< \left\lfloor \frac{4}{1-\gamma} \right\rfloor$ left over points.
  Initialize $\Pi^{(i+1)} \leftarrow \bar{S}$
  Draw a multiset, W, consisting of w points from P selected uniformly at random with replacement.
  for $j \in 1, \ldots, k$ **do**
    for $v \in S_j$ **do**
      $\hat{U}_{S_j,v} \leftarrow \{p \in W \mid v = \arg\min_{x \in S_j} d(p,x)\}$
      if $|\hat{U}_{S_j,v}| \leq (1-\gamma)w/2$ **then**
        Add (v,u) to E for every $u \in S_j$.
      else
        Add v to $\Pi^{(i+1)}$.
      end if
    end for
  end for
  $i \leftarrow i+1$
end while
Connect any remaining points in $\Pi^{(i)}$ to all points in P.
return $G = (P, E)$

Proof of Theorem 2. We first analyze the correctness of Algorithm 1 assuming that the algorithm terminates after $c \log n$ iterations of the main while loop for a fixed constant c. We then bound the number of iterations and the running time of the algorithm.

Correctness. To establish correctness, we need to prove that every point v that is *not* added to $\Pi^{(i+1)}$ during round i of the algorithm satisfies the constraints of γ-almost navigability. As in the proof Theorem 1, since any such v is connected to all other $u \in S_j$ at the end of the round, it suffices to show that $|U_{S_j,v}| \leq (1-\gamma)n$, where

$$U_{S_j,v} := \{p \in P \mid v = \arg\min_{x \in S} d(p,x)\}$$

Since v in not added to $\Pi^{(i+1)}$ if $|\hat{U}_{S_j,v}| \leq (1-\gamma)w/2$, our task reduces to showing that, if $|U_{S_j,v}| > (1-\gamma)n$, then $|\hat{U}_{S_j,v}| > (1-\gamma)w/2$ with high probability. This follows from standard concentration bounds. It particular, observe that:

$$|\hat{U}_{S_j,v}| = \sum_{p \in W} \mathbb{1}[p \in U_{S_j,v}] = \sum_{i=1}^{w} x_i,$$

where each x_i is a Bernoulli random variable with mean $|U_{S_j,v}|/n$. By a Chernoff bound, we thus have that:

$$\Pr\left[|\hat{U}_{S_j,v}| \leq \frac{w}{n}|U_{S_j,v}|/2\right] \leq e^{-w|U_{S_j,v}|/8n}.$$

Setting $w = \frac{16\log(n/\delta)}{1-\gamma}$, we conclude that, if $|U_{S_j,v}| > (1-\gamma)n$, $|\hat{U}_{S_j,v}| > (1-\gamma)w/2$ with probability at least $1 - \delta/n^2$. Taking a union bound over all v, over all $c \log n$ rounds of the algorithm, we conclude that, with probability at least $1-\delta$, every point v with

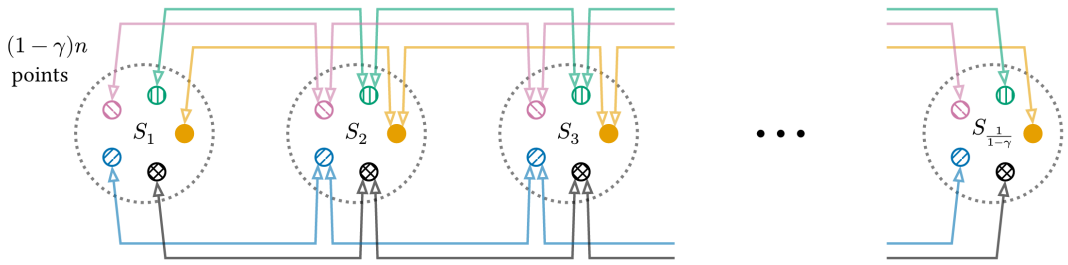

Figure 1: This figure illustrates the hard instance used to prove Lemma 4. Each cluster, $S_1, \ldots, S_k$, consists of $(1-\gamma)n$ points and the edges comprise a γ-almost navigable graph for the dataset. However, if we initialize greedy search at any point, s, it can only reach points with the same color as s. We will thus fail on all but $k = \frac{1}{1-\gamma}$ in-distribution queries.

$|U_{S_j,v}| > (1-\gamma)n$ at any round i was correctly added to $\Pi^{(i+1)}$. As discussed above, the graph G is thus γ-almost navigable.

Runtime. It remains to bound the number of iterations of the main while loop, and consequently the runtime of the algorithm. To do so, observe that, for any S_j, at any iteration, $\sum_{v \in S_j} |\hat{U}_{S_j,v}| = n$. As in the proof of Claim 3, since S_j has size $\lceil 4/(1-\gamma) \rceil$, it follows that at least half of $v \in S_j$ have $|\hat{U}_{S_j,v}| \leq (1-\gamma)w/2$. Accordingly, at iteration of the while loop, at most half of the points in S_j are added to $\Pi^{(i+1)}$. Conservatively accounting for $\bar{S}$, we conclude that:

$$|\Pi^{(i+1)}| \leq \frac{3}{4}|\Pi^i| \qquad \text{for all } i.$$

Accordingly, the while loop terminates after at most $3\log_2 n$ iterations. Moreover, the cost of Algorithm 1 is dominated by the $|\Pi^i| \cdot w$ distance computations performed at each round i of the while loop. Since the size of $|\Pi^{(i+1)}|$ is decreasing geometrically, we just conclude an overall runtime of:

$$O(nwT) = O\left(\frac{n \log(n/\delta) T}{1-\gamma}\right),$$

where T is the cost of a single distance computation. □

2.3 Performance Under Greedy Search

With Theorems 1 and 2 in place, we have established that almost navigable graphs require much lower degree and less construction time than their fully navigable counterparts. It remains to determine if these relaxed graphs remain valuable for ANN search.

In the next section, we address this question empirically, establishing overall positive results. However, we first provide some initial evidence that theoretically understanding the performance of almost navigable graphs maybe be even more difficult than understanding fully navigable graphs. In particular, the limited theoretical guarantees available for fully navigable graphs do not extend to almost navigable graphs, even approximately.

To be more concrete, recall that full navigability at least ensures correctness for *in-distribution queries*. If we receive a query q that happens to be in the dataset P, then greedy search on a navigable graph G will return q itself. While this property alone does not imply good performance on out-of-distribution queries $q \notin P$, it is a natural starting point for understanding search accuracy.

We might hope to show that γ-almost navigable graphs enjoy a similar guarantee. Perhaps we do not succeed *for all* in-distribution queries, but for some large fraction that depends on γ. Unfortunately, we prove that this is not the case:

Lemma 4. *For any n and $\gamma \in [0, 1)$, there is a set of n points, P, in 2-dimensional Euclidean space with a γ-almost navigable graph G such that greedy search on G, originating at any point $s \in P$, will not correctly return q for $n - \frac{1}{1-\gamma}$ in-distribution queries $q \in P$.*

Proof. We prove the result using a hard instance that is illustrated in Figure 1. To construct P, we place balls $S_1, \ldots, S_k$ of radius ϵ on a straight line at intervals of length greater than 2ϵ and then place $(1-\gamma) \cdot n$ points in each ball S_i. Here, $k = \frac{1}{1-\gamma}$. Next, pick $(1-\gamma) \cdot n$ colors $\mathcal{X} = \{x_1, \ldots, x_{(1-\gamma)n}\}$ and arbitrarily assign colors so that no two points in the same ball share the same color. From here on, we let p_i^x denote the point in cluster S_i with color $x \in \mathcal{X}$.

For each p_i^x, add an edge to p_{i-1}^x and p_{i+1}^x. For $i = 1$ or $i = k$, we just add edges to p_2^x and p_{k-1}^x, respectively. In other words, G connects all points of like color in adjacent clusters.

By triangle inequality, is not hard to see that G is γ-almost navigable. The two edges out of each p_i^x allow it to get closer to all points in other clusters. Because there are only $(1-\gamma)n$ points in S_i, p_i^x thus satisfies the γ-almost navigability requirement.

On the other hand, notice that G is disconnected and there are $(1-\gamma)n$ separate connected components. For any query $q \in P$, greedy search will only return q if its color matches that of the starting node s. There are only $n - k = n - \frac{1}{1-\gamma}$ such points. □

3 EXPERIMENTS

We conclude by presenting experimental evaluations of almost navigable graphs on real world datasets, comparing them to their fully navigable counterparts. We demonstrate practical improvements in graph sparsity and in search efficiency when using beam search, the standard back-tracking variant of plain greedy search used in almost all practical graph-based ANN systems. All our experiments use standard Euclidean distance as a distance metric.

3.1 Graph Density Experiments

We begin by comparing the density of navigable and almost navigable graphs on on 5 standard datasets, MNIST, Fashion MNIST, COCO-i2i, Glove25, and SPACEV1B. Details of these dataset are included in Section B, Table 1.

Experimental Setup. Ideally, we would like to understand the minimum number of edges required to construct a γ-almost navigable graph for various choices of $\gamma < 1$ and compare to the $\gamma = 1$ (fully navigable) case. While minimizing sparsity is computational intractable, as established in [12, 25] the problem of constructing a navigable graph amounts to solving a set cover instance for each node in the dataset. Likewise, γ-almost navigable graph construction involves n *partial* set cover problems. We can thus obtain graphs with sparsity within $\log n$ of optimal by using the standard greedy set cover algorithm to select out edges for each node [24].

Even greedy set cover is computationally expensive, however, requiring $O(n^2)$ time for each node. Instead, we implement the popular "robust prune" algorithm from Microsoft's DiskANN library [38]. Pseudocode is provided in Section A. For a given node v, the algorithm first adds an edge to v's nearest neighbor in P, eliminate all nodes in the dataset that it now has an edge closer to. It then adds and edge to its nearest neighbor among the remaining points, continuing until it has eliminated a γ fraction of the dataset. Robust prune requires just $O(n \log n)$ time per node and, while it does not have any guarantees, the order of edge additions seems competitive with that of greedy set cover: in initial experiments on small datasets, we only noticed small (1-2 edge) improvements in average degree when running the more expensive greedy method.

For the SPACEV1B dataset, which has over 1 billion nodes, we could not afford to build full graphs, even using robust prune, so we instead construct neighborhoods that satisfy the γ-almost navigability requirement for 10,000 randomly selected nodes. We report average degree and other statistics over that subset, noting that the average degree is an unbiased estimate of the average degree that would be obtained by the same algorithm run on the full dataset.

Results. Table 1 compares degree statistics between almost navigable graphs ($\gamma < 1$) with those of fully navigable graphs ($\gamma = 1$). Across all the datasets, we observe a marked improvement in graph sparsity, even for very high values of γ. For $\gamma = 0.995$, which means 99.5% of constraints are satisfied, we saw roughly a 50% reduction in the mean and median out-degree on the smaller datasets with less than 1 million points, like MNIST, Fashion-MNIST, and COCO-i2i. For larger datasets, there was an even greater reduction, with Glove25 and SPACEV1B needing fewer than 20% edges of the entire navigable neighborhood to achieve 99.5% coverage. Remarkably, we only needed ~ 23 edges on average to achieved 99.5% navigability for SPACEV1B, as opposed to ~ 550 for full navigability.

Dataset	# of Points	Dim.	γ	Out-Degree			
				Mean	Median	Min	Max
MNIST	60K	784	1 (navigable)	19.6	19	2	67
			0.995	10.3	10	2	32
			0.950	5.8	5	2	20
Fashion MNIST	60K	784	1 (navigable)	13.5	12	1	108
			0.995	6.6	6	1	70
			0.950	4.3	4	1	39
COCO-i2i	113K	512	1 (navigable)	28.6	27	2	131
			0.995	13.5	12	2	75
			0.950	6.7	6	2	52
Glove25	1.2M	25	1 (navigable)	50.4	50	1	141
			0.995	8.3	7	1	71
			0.950	4.2	4	1	44
SPACEV1B	1.4B	100	1 (navigable)	555	500	85	2984
			0.995	23.1	20	4	142
			0.950	8.4	8	2	77

Table 1: This table compares degree statistics for γ-almost navigable graphs to those for fully navigable graphs ($\gamma = 1$). As we can see, almost navigability allows for much lower degree, which leads to more compact, more efficient graph indices. For $\gamma = .995$ we saw at least a 47% reduction in required degree, and a reduction of more then 95% for the billion node dataset we tested on, SPACEV1B.

3.2 Retrieval Experiments

In addition to graph sparsity, we evaluate nearest neighbor query performance on the smaller of the five datasets studied above (those for which we were able to construct full search graphs. In particular, we consider the standard recall@k metric: we use the graphs to obtain a set of k-nearest neighbors for a given query q, then compute the fraction of these answers that are true top-k neighbors of q.

Experimental Setup. To obtain k-nearest neighbors, we run *beam search* on the constructed almost navigable and fully navigable graphs Beam search has become the defacto back-tracking variant of greedy search used in most graph-based methods [31, 38]. Beam search maintains a list of the closest points found so far for a given query q. At each step, it picks the closest point from that list, computing the distance between q and all of that point's out-neighbors. It then updates the list, removing the "explored" point. When run with beam width parameter b, search terminates when the best node in the list is no closer to q than b points explored previously. Increasing b increases how long the search executes for, so provides a natural way to trade-off between recall and runtime. For a more detailed discussion of the method and full pseudocode, see [1].

Since the runtime of graph-based search is dominated by the cost of computing distances between the query q and candidate nearest neighbors, we report the total number of distance computations as our main performance metric. This metric is less noisy and system dependent than, e.g., wall-clock runtime. Concretely, for each graph, for each dataset, we run search using beam widths $\{1, 2, 4, 16, 32, 64, 100, 128, 256\}$. For each beam width we compute the average recall@k, where $k \in \{1, 10, 100\}$, and the average number of distance computations over all dataset test queries.

Results. Figure 2 visualizes the resulting cost/performance tradeoffs. Each point on each plot corresponds to a particular choice of the beam width b (indicated by the points color) and a value of γ between .8 and 1 that was used to construct the search graph. For each choice of b, we can trace a line of points with increasing recall and distance computations, which corresponds to increasing γ, i.e., increasing the graph coverage. Observe that in each of the plots, there are almost navigable graph search configurations that lie above the recall-distance-computation curve of fully navigable

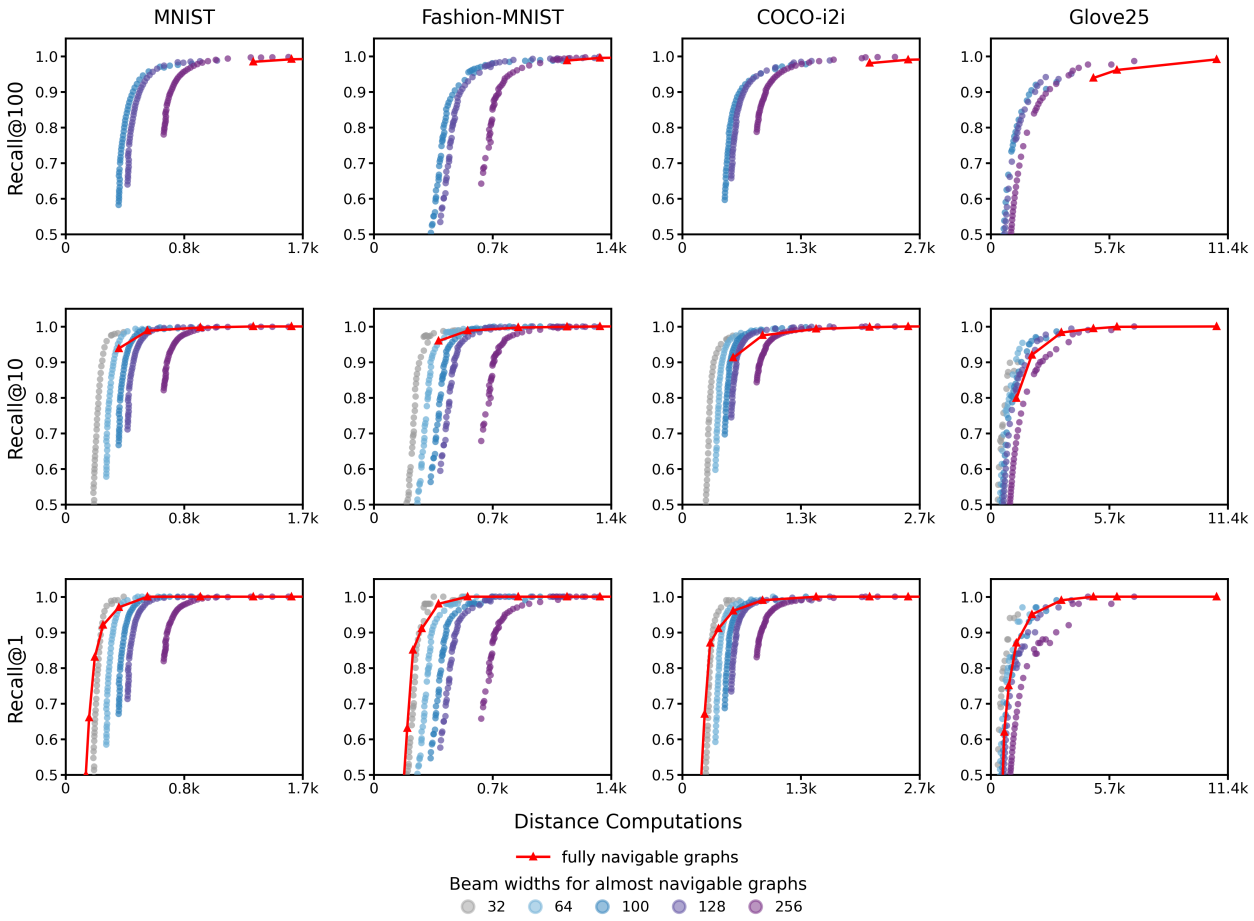

Figure 2: These plots show average recall@k vs. average distance computations for various choices of navigability parameter, γ, and beam width parameter, b. The red solid line represents the recall curve for a fully navigable graphs for various choices of beam width. We only plot points corresponding to beam widths $\geq k$, since having a lower beam width places a limit on the maximum achievable recall. In these plots, points up and to the left correspond to better performance: higher recall and fewer distance computations. As can be seen, for all data sets, there are almost navigable graphs which, for an appropriate choice of beam width, match or beat the performance of search on the fully navigable graph.

graphs. The improvement is most striking for recall@100, where almost navigable graphs achieve higher or similar recall for much fewer distance computations.

For a more detailed analysis, Table 2 describes the efficiency trade-off to achieve certain target recalls. More specifically, for a chosen target recall, we find beam width and γ settings for almost navigable graphs that achieve that recall and compare them to their fully navigable counterparts. The table displays the fraction of distance computations performed during search on almost navigable graphs relative to the distance computations performed on navigable graphs to achieve the target recall at recall@k. Additionally, Table 2 includes the ratio of the average out-degrees of almost navigable graphs to the average out-degree of navigable graphs. In both of these statistics, a score lower than 1 indicates greater search efficiency for almost navigable graphs.

Across all datasets, almost navigable graphs are more efficient at achieving their recall targets in terms of the number of distance computations required to perform search on them and the amount of space required to store them (graph sparsity). On average, γ-almost navigable graphs used 47% fewer distance computations and 55% less space than fully navigable graphs to achieve a given target recall@100. We see similar efficiency gains for recall@1 and recall@10, with an average reduction of 35% in distance computations and $\sim$ 50% reduction in graph size. **??** contains a more detailed report of the distance computations and average graph degree statistics we compiled for Table 2.

On choosing γ. The dataset's size and structure appear to be important in determining the right choice of γ. In these preliminary evaluations, $\gamma = 0.995$ is sufficient for COCO-i2i and MNIST to

Dataset	Target Recall@k	Ratio of Distance Computations			Ratio of Average Degree		
		$k = 1$	$k = 10$	$k = 100$	$k = 1$	$k = 10$	$k = 100$
MNIST	0.90	0.69	0.61	0.39	0.53	0.55	0.33
	0.95	0.66	0.67	0.44	0.55	0.41	0.41
	0.97	0.62	0.62	0.51	0.58	0.53	0.53
	0.99	0.68	0.70	0.69	0.58	0.53	0.41
Fashion-MNIST	0.90	0.69	0.62	0.44	0.55	0.50	0.38
	0.95	0.66	0.68	0.50	0.51	0.64	0.50
	0.97	0.68	0.65	0.56	0.64	0.50	0.57
	0.99	0.63	0.79	0.70	0.72	0.77	0.64
COCO-i2i	0.90	0.74	0.59	0.33	0.32	0.50	0.27
	0.95	0.63	0.56	0.42	0.50	0.39	0.32
	0.97	0.56	0.65	0.52	0.58	0.32	0.53
	0.99	0.55	0.61	0.67	0.50	0.39	0.39
Glove25	0.90	0.53	0.53	0.41	0.50	0.41	0.37
	0.95	0.66	0.56	0.70	0.60	0.41	0.60
	0.97	0.57	0.70	0.59	0.41	0.41	0.37
	0.99	0.93	0.81	0.65	0.60	0.60	0.60
Average		**0.65**	**0.65**	**0.53**	**0.54**	**0.49**	**0.45**

Table 2: This table compares the ratio of computational cost, measured as average distance computations (DC) and average degree, of achieving a target recall@k for γ-almost navigable graphs (when $\gamma < 1$) relative to fully navigable graphs ($\gamma = 1$). I.e., for some target recall@k, the table shows $\frac{\text{avg. DC for } \gamma < 1}{\text{avg. DC for } \gamma = 1}$ and $\frac{\text{avg. degree for } \gamma < 1}{\text{avg. degree for } \gamma = 1}$. Values < 1 mean the almost navigable graph had lower cost; values ≥ 1 mean otherwise. For every target recall, γ-almost navigable graphs requires fewer distance computations ($\sim 35 - 47\%$ reduction) at smaller graph sizes ($\sim 46 - 55\%$ reduction) on average. The data points for this table were chosen by interpolating the curves in Figure 2.

achieve recall@$k > 0.97$ for all k, but not for Glove25 and Fashion-MNIST; those required $\gamma = 0.9995$ or higher to achieve the same performance. Interestingly, as reported in Table 2, these graphs are still sparse and efficient relative to their fully navigable counterparts. See Appendix B for more detailed tables containing graph degree statistics.

4 DISCUSSION

In this work, we introduce a natural relaxation of the navigability property, which has become central in work on graph-based approximate nearest neighbor search. Our "almost navigable graphs" have a linear number of edges, and can be constructed in near linear time, both polynomial improvements over full navigability. Moreover, initial experimental results appear promising. Looking towards next steps, we hope to provide a more thorough empirical evaluation for search on almost navigable graphs. On the theoretical side, it would be valuable to explore other relaxations of navigability and related concepts like α-shortcut reachability. We are particular interested in relaxations which, unlike γ-almost navigability, at least partially preserve some of the theoretical properties of navigable graphs.

5 ACKNOWLEDGMENTS

Pratyush Avi was partially supported by a GAANN fellowship from the US Department of Education.

REFERENCES

[1] Yousef Al-Jazzazi, Haya Diwan, Jinrui Gou, Cameron Musco, Christopher Musco, and Torsten Suel. 2025. Distance Adaptive Beam Search for Provably Accurate Graph-Based Nearest Neighbor Search. In *Advances in Neural Information Processing Systems 38 (NeurIPS)*.

[2] Alexandr Andoni and Piotr Indyk. 2008. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. *Commun. ACM* 51, 1 (2008), 117–122.

[3] Alexandr Andoni, Piotr Indyk, Thijs Laarhoven, Ilya Razenshteyn, and Ludwig Schmidt. 2015. Practical and optimal LSH for angular distance. In *Advances in Neural Information Processing Systems 28 (NeurIPS)*.

[4] Alexandr Andoni, Ilya Razenshteyn, and Negev Shekel Nosatzki. 2017. LSH Forest: Practical Algorithms Made Theoretical. In *Proceedings of the 28th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. 67–78.

[5] Sunil Arya and David M. Mount. 1993. Approximate nearest neighbor queries in fixed dimensions. In *Proceedings of the 4th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*.

[6] Martin Aumüller, Erik Bernhardsson, and Alexander Faithfull. 2020. ANN-Benchmarks: A benchmarking tool for approximate nearest neighbor algorithms. *Information Systems* 87 (2020).

[7] Mayank Bawa, Tyson Condie, and Prasanna Ganesan. 2005. LSH forest: self-tuning indexes for similarity search. In *Proceedings of the 14th International Conference on World Wide Web*.

[8] Alina Beygelzimer, Sham Kakade, and John Langford. 2006. Cover trees for nearest neighbor. In *Proceedings of the 23rd International Conference on Machine Learning (ICML)*.

[9] Marián Boguñá, Dmitri Krioukov, and K. C. Claffy. 2009. Navigability of Complex Networks. *Nature Physics* 5, 1 (2009), 74–80.

[10] Kenneth L. Clarkson. 1994. An algorithm for approximate closest-point queries. In *Proceedings of the 10th Annual Symposium on Computational Geometry (SOCG)*.

[11] Aaron Clauset and Cristopher Moore. 2003. How Do Networks Become Navigable? *arXiv:0309415* (2003).

[12] Alex Conway, Laxman Dhulipala, Martin Farach-Colton, Rob Johnson, Ben Landrum, Christopher Musco, Yarin Shechter, Torsten Suel, and Richard Wen. 2026. Efficiently Constructing Sparse Navigable Graphs. In *Proceedings of the 37th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*.

[13] Haya Diwan, Jinrui Gou, Cameron Musco, Christopher Musco, and Torsten Suel. 2024. Navigable Graphs for High-Dimensional Nearest Neighbor Search: Constructions and Limits. In *Advances in Neural Information Processing Systems 37 (NeurIPS)*.

[14] Matthijs Douze, Alexandr Guzhva, Chengqi Deng, Jeff Johnson, Gergely Szilvasy, Pierre-Emmanuel Mazaré, Maria Lomeli, Lucas Hosseini, and Hervé Jégou. 2026. The Faiss Library. *IEEE Transactions on Big Data* 12, 2 (2026), 346–361.

[15] Cong Fu and Deng Cai. 2016. EFANNA: An extremely fast approximate nearest neighbor search algorithm based on kNN graph. *arXiv:1609.07228* (2016).

[16] Cong Fu, Chao Xiang, Changxu Wang, and Deng Cai. 2019. Fast Approximate Nearest Neighbor Search with the Navigating Spreading-out Graph. *Proceedings of the VLDB Endowment* 12, 5 (2019), 461–474. Data acccessed at: https://github.com/ZJULearning/nsg.

[17] Siddharth Gollapudi, Ravishankar Krishnaswamy, Kirankumar Shiragur, and Harsh Wardhan. 2025. Sort Before You Prune: Improved Worst-Case Guarantees of the DiskANN Family of Graphs. In *Proceedings of the 42nd International Conference on Machine Learning (ICML)*.

[18] Sariel Har-Peled, Benjamin Raichel, and Eliot W. Robson. 2026. The Road to the Closest Point is Paved by Good Neighbors. In *Proceedings of the 9th Symposium on Simplicity in Algorithms (SOSA)*.

[19] Ben Harwood and Tom Drummond. 2016. FANNG: Fast Approximate Nearest Neighbour Graphs. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[20] Piotr Indyk and Rajeev Motwani. 1998. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of the 30th Annual ACM Symposium on Theory of Computing (STOC)*. 604–613.

[21] Piotr Indyk and Haike Xu. 2023. Worst-case Performance of Popular Approximate Nearest Neighbor Search Implementations: Guarantees and Limitations. In *Advances in Neural Information Processing Systems 36 (NeurIPS)*.

[22] Herve Jégou, Matthijs Douze, and Cordelia Schmid. 2011. Product Quantization for Nearest Neighbor Search. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 33, 1 (2011), 117–128.

[23] Jeff Johnson, Matthijs Douze, and Hervé Jégou. 2021. Billion-Scale Similarity Search with GPUs. *IEEE Transactions on Big Data* 7, 03 (2021), 535–547.

[24] Michael J Kearns. 1990. *The Computational Complexity of Machine Learning*. MIT Press, London, England.

[25] Sanjeev Khanna, Ashwin Padaki, and Erik Waingarten. 2026. Sparse Navigable Graphs for Nearest Neighbor Search: Algorithms and Hardness. In *Proceedings of the 37th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*.

[26] Jon Kleinberg. 2000. The small-world phenomenon: An algorithmic perspective. In *Proceedings of the 32nd Annual ACM Symposium on Theory of Computing (STOC)*. 163–170.

[27] Robert Krauthgamer and James R Lee. 2004. Navigating nets: simple algorithms for proximity search.. In *Proceedings of the 35th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*.

[28] Ravishankar Krishnaswamy, Magdalen Dobson Manohar, and Harsha Vardhan Simhadri. 2024. The DiskANN library: Graph-Based Indices for Fast, Fresh and Filtered Vector Search. *IEEE Data Eng. Bull.* 48, 3 (2024), 20–42.

[29] Qin Lv, William Josephson, Zhe Wang, Moses Charikar, and Kai Li. 2007. Multi-probe LSH: efficient indexing for high-dimensional similarity search. In *Proceedings of the 33rd International Conference on Very Large Data Bases (VLDB)*.

[30] Yury Malkov, Alexander Ponomarenko, Andrey Logvinov, and Vladimir Krylov. 2014. Approximate nearest neighbor algorithm based on navigable small world graphs. *Information Systems* 45 (2014), 61–68.

[31] Yu A. Malkov and D. A. Yashunin. 2020. Efficient and Robust Approximate Nearest Neighbor Search Using Hierarchical Navigable Small World Graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 42, 4 (2020), 824–836.

[32] Magdalen Dobson Manohar, Zheqi Shen, Guy Blelloch, Laxman Dhulipala, Yan Gu, Harsha Vardhan Simhadri, and Yihan Sun. 2024. ParlayANN: Scalable and Deterministic Parallel Graph-Based Approximate Nearest Neighbor Search Algorithms. In *Proceedings of the 29th ACM SIGPLAN Annual Symposium on Principles and Practice of Parallel Programming (PPoPP)*.

[33] Stanley Milgram. 1967. The small world problem. *Psychology today* 2, 1 (1967), 60–67.

[34] G. Navarro. 1999. Searching in metric spaces by spatial approximation. In *6th International Symposium on String Processing and Information Retrieval. 5th International Workshop on Groupware*.

[35] Yun Peng, Byron Choi, Tsz Nam Chan, Jianye Yang, and Jianliang Xu. 2023. Efficient Approximate Nearest Neighbor Search in Multi-dimensional Databases. *Proceeding of the ACM on Management of Data* 1, 1 (2023).

[36] Harsha Vardhan Simhadri, Martin Aumüller, Matthijs Douze, Dmitry Baranchuk, Amir Ingber, Edo Liberty, George Williams, Ben Landrum, Magdalen Dobson Manohar, Mazin Karjikar, Laxman Dhulipala, Meng Chen, Yue Chen, Rui Ma, Kai Zhang, Yuzheng Cai, Jiayang Shi, Weiguo Zheng, Yizhuo Chen, Jie Yin, and Ben Huang. 2026. Results of the Big ANN: NeurIPS'23 competition. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems Datasets and Benchmarks Track*.

[37] Harsha Vardhan Simhadri, George Williams, Martin Aumüller, Matthijs Douze, Artem Babenko, Dmitry Baranchuk, Qi Chen, Lucas Hosseini, Ravishankar Krishnaswamny, Gopal Srinivasa, Suhas Jayaram Subramanya, and Jingdong Wang. 2022. Results of the NeurIPS'21 Challenge on Billion-Scale Approximate Nearest Neighbor Search. In *Proceedings of the NeurIPS 2021 Competitions and Demonstrations Track*.

[38] Suhas Jayaram Subramanya, Devvrit, Rohan Kadekodi, Ravishankar Krishaswamy, and Harsha Vardhan Simhadri. 2019. DiskANN: Fast Accurate Billion-Point Nearest Neighbor Search on a Single Node. In *Advances in Neural Information Processing Systems 32 (NeurIPS)*.

[39] Jeffrey Travers and Stanley Milgram. 1977. An Experimental Study of the Small World Problem. In *Social Networks*. Academic Press, 179–197.

A ALGORITHMS

In this section, we present pseudocode for the various algorithms used for the experimental analysis presented in this paper. The robust prune and set cover algorithms described here largely follow their standard descriptions, with the only change being the stopping criteria. Constructing a fully navigable graph would require the algorithm to keep picking edges until there is no point left uncovered. In Algorithm 2 of the robust prune algorithm and Algorithm 3 of the greedy set cover algorithm, we instead stop when the number of uncovered points falls below $(1-\gamma)n$. Notably, this is the same as regular navigability when $\gamma = 1$.

Algorithm 2 Robust Prune w/ Early Stopping

Input: Set of n points P, blackbox access to distance function $d : P \times P \to R^{\geq 0}$, $\gamma \in [0, 1]$
Output: γ-navigable graph $G = (P, E)$

$E \leftarrow \{\}$
for each $p \in P$ **do**
  $U \leftarrow P \setminus \{p\}$
  while $|U| > (1-\gamma)n$ **do**
    $v \leftarrow \arg\min_{x \in U} d(p, x)$
    Add edge (p, v) to E
    $C \leftarrow \{y \in U \mid d(v, y) < d(p, y)\}$
    $U \leftarrow U \setminus \{C\}$, remove covered points from U
  end while
end for
return $G = (P, E)$

Algorithm 3 Partial Greedy Set Cover

Input: Set of n points P, blackbox access to distance function $d : P \times P \to R^{\geq 0}$, $\gamma \in [0, 1]$
Output: γ-navigable graph $G = (P, E)$

$E \leftarrow \{\}$
for each $p \in P$ **do**
  $U \leftarrow P \setminus \{p\}$
  For each $x \in U$, define $\mathcal{S}_{p \to x} := \{y \in P \mid d(x, y) < d(p, y)\}$
  while $|U| > (1-\gamma)n$ **do**
    $v \leftarrow \arg\min_{x \in U} |\mathcal{S}_{p \to x} \cap U|$
    Add edge (p, v) to E
    $U \leftarrow U \setminus \{\mathcal{S}_{p \to x}\}$, remove covered points from U
  end while
end for
return $G = (P, E)$

B ADDITIONAL TABLES AND PLOTS

The following tables include more details about the characteristics of almost navigable graphs on real world datasets. Table 3 expands on Table 1 and includes out-degree and in-degree statistics for a larger range of coverage values. Table 4 and Table 5, expand on Table 2 and include details about the exact distance computations, average graph degree, and value of γ that achieve the desired target recalls.

Dataset	Points	Dim.	γ	Out-degree				In-degree		
				Mean	Median	Min	Max	Median	Min	Max
MNIST	60,000	784	1.0000	19.64	19.0	2	67	17.0	1	134
			0.9950	10.32	10.0	2	32	10.0	1	63
			0.9500	5.79	5.0	2	20	6.0	1	31
Fashion-MNIST	60,000	784	1.0000	13.55	12.0	1	108	12.0	1	117
			0.9999	12.13	11.0	1	107	11.0	1	70
			0.9995	9.72	8.0	1	106	9.0	1	55
			0.9950	6.59	6.0	1	70	6.0	1	54
			0.9500	4.33	4.0	1	39	4.0	1	51
COCO-i2i	113,287	512	1.0000	28.59	27.0	2	131	24.0	1	268
			0.9950	13.54	12.0	2	75	13.0	1	87
			0.9500	6.69	6.0	2	52	6.0	1	49
Glove25	1,183,514	25	1.0000	50.41	50.0	1	141	43.0	1	422
			0.9999	25.13	24.0	1	98	24.0	1	99
			0.9995	16.02	15.0	1	87	15.0	1	78
			0.9950	8.29	7.0	1	71	7.0	1	61
			0.9500	4.23	4.0	1	44	4.0	1	49
SPACEV1B	1,402,020,720	100	1.0000	554.80	500.0	85	2984	1.0	1	4
			0.9950	23.06	20.0	4	142	1.0	1	2
			0.9500	8.36	8.0	2	77	1.0	1	2

Table 3: This table expands on Table 1. For every dataset, this table describes the out-degree and in-degree statistics for a range of γs. Note, when $\gamma = 1$, the graph is fully navigable. Observe that reducing γ leads to large decreases in the graph size. Even when we consider $\gamma = 0.9999$ for Glove25, we observe a nearly 50% decrease in average out-degree. Remarkably, we achieve a nearly 90% decrease in graph size for our 1 billion sized point set. We notice a similar trend for the maximum in-degree of the graph, where we achieve a 50-75% decrease. In-degree statistics can be valuable in establishing the overall structure of the constructed graph.

Dataset	Target Recall	Distance Computations											
		$k = 1$				$k = 10$				$k = 100$			
		F	**A**	**R**	γ	**F**	**A**	**R**	γ	**F**	**A**	**R**	γ
MNIST	0.90	254	175	0.69	0.99500	356	217	0.61	0.99600	1248	481	0.39	0.96500
	0.95	335	220	0.66	0.99600	429	285	0.67	0.98500	1303	568	0.44	0.98500
	0.97	380	236	0.62	0.99700	511	316	0.62	0.99500	1325	672	0.51	0.99500
	0.99	517	350	0.68	0.99700	669	471	0.70	0.99500	1557	1073	0.69	0.98500
Fashion-MNIST	0.90	266	184	0.69	0.99750	341	212	0.62	0.99550	1039	460	0.44	0.98000
	0.95	330	216	0.66	0.99600	366	249	0.68	0.99900	1079	538	0.50	0.99550
	0.97	357	244	0.68	0.99900	434	284	0.65	0.99550	1095	613	0.56	0.99800
	0.99	454	287	0.63	0.99950	602	477	0.79	0.99970	1163	815	0.70	0.99900
COCO-i2i	0.90	383	282	0.74	0.98000	562	333	0.59	0.99600	1966	649	0.33	0.96500
	0.95	540	338	0.63	0.99600	772	435	0.56	0.99000	2054	872	0.42	0.98000
	0.97	683	382	0.56	0.99800	876	567	0.65	0.98000	2089	1093	0.52	0.99700
	0.99	902	495	0.55	0.99600	1406	857	0.61	0.99000	2534	1697	0.67	0.99000
Glove25	0.90	1516	797	0.53	0.99990	1856	985	0.53	0.99980	4728	1917	0.41	0.99970
	0.95	1977	1305	0.66	0.99995	2653	1492	0.56	0.99980	5487	3854	0.70	0.99995
	0.97	2686	1526	0.57	0.99980	3103	2165	0.70	0.99980	7440	4357	0.59	0.99970
	0.99	3396	3163	0.93	0.99995	4370	3519	0.81	0.99995	10656	6894	0.65	0.99995

Table 4: This table shows the average distance computations to reach each target recall, broken out by Recall@k ($k \in \{1, 10, 100\}$). Each block reports Fully-navigable (F, coverage = 1), Almost-navigable (A, Pareto frontier over coverage < 1), their ratio R = A / F, and γ (coverage of the chosen almost-navigable point. For every target recall, γ-almost navigable graphs requires fewer distance computations ($\sim 35 - 47\%$ reduction) on average. The data points for this table were chosen by interpolating the curves in Figure 2.

Dataset	Target Recall	Average Degree											
		$k = 1$				$k = 10$				$k = 100$			
		F	**A**	**R**	γ	**F**	**A**	**R**	γ	**F**	**A**	**R**	γ
MNIST	0.90	19.64	10.32	0.53	0.99500	19.64	10.81	0.55	0.99600	19.64	6.42	0.33	0.96500
	0.95	19.64	10.81	0.55	0.99600	19.64	8.04	0.41	0.98500	19.64	8.04	0.41	0.98500
	0.97	19.64	11.45	0.58	0.99700	19.64	10.32	0.53	0.99500	19.64	10.32	0.53	0.99500
	0.99	19.64	11.45	0.58	0.99700	19.64	10.32	0.53	0.99500	19.64	8.04	0.41	0.98500
Fashion-MNIST	0.90	13.55	7.41	0.55	0.99750	13.55	6.71	0.50	0.99550	13.55	5.16	0.38	0.98000
	0.95	13.55	6.85	0.51	0.99600	13.55	8.65	0.64	0.99900	13.55	6.71	0.50	0.99550
	0.97	13.55	8.65	0.64	0.99900	13.55	6.71	0.50	0.99550	13.55	7.70	0.57	0.99800
	0.99	13.55	9.72	0.72	0.99950	13.55	10.48	0.77	0.99970	13.55	8.65	0.64	0.99900
COCO-i2i	0.90	28.59	9.15	0.32	0.98000	28.59	14.29	0.50	0.99600	28.59	7.60	0.27	0.96500
	0.95	28.59	14.29	0.50	0.99600	28.59	11.27	0.39	0.99000	28.59	9.15	0.32	0.98000
	0.97	28.59	16.66	0.58	0.99800	28.59	9.15	0.32	0.98000	28.59	15.27	0.53	0.99700
	0.99	28.59	14.29	0.50	0.99600	28.59	11.27	0.39	0.99000	28.59	11.27	0.39	0.99000
Glove25	0.90	50.41	25.13	0.50	0.99990	50.41	20.70	0.41	0.99980	50.41	18.47	0.37	0.99970
	0.95	50.41	30.41	0.60	0.99995	50.41	20.70	0.41	0.99980	50.41	30.41	0.60	0.99995
	0.97	50.41	20.70	0.41	0.99980	50.41	20.70	0.41	0.99980	50.41	18.47	0.37	0.99970
	0.99	50.41	30.41	0.60	0.99995	50.41	30.41	0.60	0.99995	50.41	30.41	0.60	0.99995

Table 5: This table shows the average degree to reach each target recall, broken out by Recall@k ($k \in \{1, 10, 100\}$). Each block reports Fully-navigable (F, coverage = 1), Almost-navigable (A, Pareto frontier over coverage < 1), their ratio R = A / F, and γ (coverage of the chosen almost-navigable point. For every target recall, γ-almost navigable graphs requires fewer edges ($\sim 46 - 55\%$ reduction) on average. The data points for this table were chosen by interpolating the curves in Figure 2.