

# Sparse Navigable Graphs for Nearest Neighbor Search: Algorithms and Hardness

Sanjeev Khanna\*

Ashwin Padaki\*

Erik Waingarten\*

July 21, 2025

## Abstract

We initiate the study of approximation algorithms and computational barriers for constructing sparse  $\alpha$ -navigable graphs [IX23, DGM<sup>+</sup>24], a core primitive underlying recent advances in graph-based nearest neighbor search. Given an  $n$ -point dataset  $P$  with an associated metric  $d$  and a parameter  $\alpha \geq 1$ , the goal is to efficiently build the sparsest graph  $G = (P, E)$  that is  $\alpha$ -navigable: for every distinct  $s, t \in P$ , there exists an edge  $(s, u) \in E$  with  $d(u, t) < d(s, t)/\alpha$ . We consider two natural sparsity objectives: minimizing the maximum out-degree and minimizing the total size (or equivalently, the average degree).

Our starting point is a strong negative result: the slow-preprocessing version of DiskANN (analyzed in [IX23] for low-doubling metrics) can yield solutions whose sparsity is  $\tilde{\Omega}(n)$  times larger than optimal, even on Euclidean instances. We then show a tight approximation-preserving equivalence between the Sparsest Navigable Graph problem and the classic Set Cover problem, obtaining an  $O(n^3)$ -time  $(\ln n + 1)$ -approximation algorithm, as well as establishing NP-hardness of achieving an  $o(\ln n)$ -approximation. Building on this equivalence, we develop faster  $O(\ln n)$ -approximation algorithms. The first runs in  $\tilde{O}(n \cdot \text{OPT})$  time and is therefore much faster when the optimal solution is sparse. The second, based on fast matrix multiplication, is a bicriteria algorithm that computes an  $O(\ln n)$ -approximation to the sparsest  $2\alpha$ -navigable graph, running in  $\tilde{O}(n^\omega)$  time. Finally, we complement our upper bounds with a query complexity lower bound, showing that any  $o(n)$ -approximation requires examining  $\Omega(n^2)$  distances. This result shows that in the regime where  $\text{OPT} = \tilde{O}(n)$ , our  $\tilde{O}(n \cdot \text{OPT})$ -time algorithm is essentially best possible.

Collectively, these results significantly advance our understanding of the computational complexity of computing sparse navigable graphs.

---

\*University of Pennsylvania {sanjeev@cis.upenn.edu, apadaki@seas.upenn.edu, ewaingar@seas.upenn.edu}

# 1 Introduction

This work is on graph-based methods for similarity search, an algorithmic approach which has emerged over the past decade and demonstrated impressive empirical performance (see [SWA<sup>+</sup>21, SAI<sup>+</sup>24]). These techniques aim to solve the *c*-approximate nearest neighbor search problem: for an underlying metric space  $(X, d)$ , the task is to preprocess a dataset  $P \subset X$  of  $n$  points into a data structure which supports approximate nearest neighbor queries; a new (unseen) query point  $q \in X$  is presented, and the data structure must output a dataset point whose distance to  $q$  is at most  $c$  times that of the closest dataset point. In a graph-based method, the data structure is a directed graph whose vertices correspond to dataset points, and queries are processed by executing a “greedy-like” search over the graph.<sup>1</sup> Until relatively recently, these methods had not been studied from the theoretical computer science perspective, and this work aims to study these methods from the vantage point of *approximation algorithms*. We begin by discussing two recent and related works of [IX23, DGM<sup>+</sup>24], which motivate the algorithmic questions at the heart of this paper.

**Provable Guarantees for Graph-Based Methods via the Doubling Dimension [IX23].** A feature of graph-based methods is that they are often agnostic to the underlying metric. However, from the worst-case perspective, one may construct metric spaces in which nearest neighbor search requires a prohibitive  $\Omega(n)$  query time for any approximation (e.g., a uniform metric [KL04]). As a result, the community has developed notions of intrinsic dimensionality, which seeks to capture the low-dimensional metric structure in a dataset (see the survey [Cla06, DK20]). The notion of intrinsic dimensionality most relevant to nearest neighbor search is the *doubling dimension* [GKL03, KL04]: a set of points has doubling dimension  $\lambda$  if every metric ball can be covered with  $2^\lambda$  balls of half the radius. For datasets with low doubling dimension, efficient nearest neighbor search is possible. For example, cover trees [BKL06] give an  $O(n)$ -space data structure whose query time is  $2^{O(\lambda)} \log n$  for exact nearest neighbor search, and  $2^{O(\lambda)} \log \Delta + (1/\varepsilon)^{O(\lambda)}$  for  $(1 + \varepsilon)$ -approximate nearest neighbor search.<sup>2</sup>

Indyk and Xu [IX23] were the first to parametrize the performance of graph-based methods in terms of the doubling dimension. They study several popular algorithms that are used in practice—DiskANN [JSDS<sup>+</sup>19], NSG [FXWC19], and HNSW [MY18]—and construct datasets with constant doubling dimension (in particular, in  $\mathbb{R}^2$ ) where all of these algorithms require reading  $\sim 10\%$  of the dataset in order to produce an acceptable near neighbor. While seemingly bad news, they also show that DiskANN with *slow-preprocessing time* (removing a heuristic from [JSDS<sup>+</sup>19]) performs provably well in metrics with low doubling dimension, with query time  $(1/\varepsilon)^{O(\lambda)} \log^2 \Delta$  for  $(1 + \varepsilon)$ -approximate nearest neighbor search. This was the first indication that a theoretical computer science perspective could shed light on the worst-case performance of graph-based methods.<sup>3</sup> Since then, multiple works have sought to prove guarantees and limitations on various aspects of these methods [DGM<sup>+</sup>24, XSI24, AIK<sup>+</sup>25, AJDG<sup>+</sup>25, GKSW25]. The underlying structural property in [IX23] as well as the subsequent work [DGM<sup>+</sup>24] is that of *navigability*, which we discuss next.

**Navigability Suffices for Nearest Neighbor Search.** We formalize graph-based methods following [IX23]. A graph-based method for a dataset  $P$ , a finite subset of a metric space  $(X, d)$ , is specified by a directed graph  $G$  whose vertices are the elements of  $P$ . A query  $q \in X$  is processed by starting at an arbitrary vertex of  $G$  and following a greedy search: at a current vertex  $s$ , we compute distances  $d(q, u)$  for all out-neighbors  $u$  of  $s$  in the graph  $G$ ; if the minimum  $d(q, u)$  is strictly smaller than  $d(q, s)$ , we update the current vertex to  $u$  and repeat; otherwise, we output  $s$ . The total space of the data structure is the number of edges in  $G$ .

<sup>1</sup>The specific implementation varies across heuristics; as in recent work of [IX23, DGM<sup>+</sup>24], we analyze the greedy search process (explained formally below).

<sup>2</sup>The parameter  $\Delta$  is the aspect ratio of the input dataset  $P$ . It is the maximum pairwise distance between points divided by the minimum nonzero pairwise distance between points.

<sup>3</sup>Prior theoretical works, such as [Laa18, PS20], study graph-based methods from an average-case complexity perspective, when the dataset is generated uniformly at random on a high-dimensional sphere.

The query time is at most the maximum out-degree of a vertex in  $G$ , multiplied by the length of the longest “greedy path” that may be encountered. The algorithmic challenge, then, is devising a low-degree graph that guarantees fast and accurate convergence of these greedy paths for potentially infinitely-many query points. The key to the analysis in [IX23] is the following definition, which gives a sufficient condition on the graph  $G$  for fast and accurate search.<sup>4</sup>

**Definition 1.1** ( $\alpha$ -Navigability (Definition 3.1 in [IX23])). For  $\alpha \geq 1$ , a directed graph  $G = (P, E)$  is said to be  $\alpha$ -navigable if for every  $s, t \in P$  with  $d(s, t) > 0$ , there is some  $u \in P$  with  $(s, u) \in E$  and  $d(u, t) < d(s, t)/\alpha$ .

An important algorithmic contribution is Theorem 3.4 in [IX23], which shows that in an  $\alpha$ -navigable graph  $G$ , a greedy search always returns an  $(\frac{\alpha+1}{\alpha-1} + \varepsilon)$ -approximate nearest neighbor in at most  $O(\log_\alpha(\frac{\Delta}{(\alpha-1)\varepsilon}))$  steps. The final dependence of the query time on the doubling dimension arises from their analysis of the slow-preprocessing algorithm of DiskANN, which constructs an  $\alpha$ -navigable graph with out-degree at most  $O(\alpha)^\lambda \log \Delta$ .

The subsequent work of [DGM<sup>+</sup>24] studies navigability without the assumption of low doubling dimension. Focusing on the case  $\alpha = 1$  (the weakest form of  $\alpha$ -navigability), they show that every  $n$ -point metric admits a 1-navigable graph with average degree  $O(\sqrt{n \log n})$ . Furthermore, they construct subsets of  $O(\log n)$ -dimensional Euclidean space for which every 1-navigable graph must have average degree at least  $\Omega(n^{1/2-\delta})$ , for any constant  $\delta > 0$ .

**Motivating Questions and Contributions.** Roughly speaking, [IX23] shows that low doubling dimension is a sufficient condition for the existence of sparse  $\alpha$ -navigable graphs, while [DGM<sup>+</sup>24] proves an existential guarantee that every metric admits a nontrivial 1-navigable graph. Motivated by these results, we propose a broader algorithmic question: given an arbitrary metric and  $\alpha \geq 1$ , what is the sparsest  $\alpha$ -navigable graph—and how efficiently can this graph be computed?

More precisely, we ask:

- Is there an efficient (approximation) algorithm that, for any metric, constructs the sparsest  $\alpha$ -navigable graph (minimizing either the maximum out-degree or the total number of edges), thereby yielding instance-by-instance guarantees?
- Since preprocessing time is a central concern in graph-based nearest neighbor search methods, can we design fast algorithms for building  $\alpha$ -navigable graphs which do not sacrifice on provable guarantees?

Our work is the first to address these questions from the viewpoint of approximation algorithms. This perspective offers two key advantages: (i) it yields graphs that are (approximately) optimal in sparsity for use in nearest neighbor search, and (ii) when the output graph is dense, it serves as a certificate that no substantially sparser  $\alpha$ -navigable graphs exist for that instance.

## 1.1 Overview of Results

We provide a comprehensive set of results for computing the sparsest  $\alpha$ -navigable graph, from both the approximation algorithms and computational hardness perspectives.

**Suboptimality of Existing Heuristics (Theorem 1).** We first show that the slow-preprocessing variant of DiskANN, previously analyzed for low-doubling-dimension metric spaces, can be suboptimal in (nearly) the

---

<sup>4</sup>In their work, [IX23] term this notion “ $\alpha$ -shortcut reachability.” We adapt the terminology of [DGM<sup>+</sup>24] and use the term  $\alpha$ -navigability.

strongest possible sense. Specifically, we construct explicit subsets of Euclidean space for which this algorithm produces 1-navigable graphs with  $\Omega(n^2)$  edges and maximum out-degree  $\Omega(n)$ , despite the existence of 1-navigable graphs with only  $O(n \log n)$  edges and maximum out-degree  $O(\log n)$ . Thus, the DiskANN algorithm incurs a worst-case approximation ratio of  $\Omega(n/\log n)$  for both sparsity and degree, even in Euclidean space. Our result bridges a gap in the theoretical analysis of empirical heuristics for graph-based nearest neighbor search.

**Equivalence to Set Cover: Approximation Algorithms and Hardness (Theorem 2).** We show that the problem of constructing the sparsest  $\alpha$ -navigable graph is tightly connected to the classic Set Cover problem, by showing an approximation-preserving reduction in both directions. Specifically, for any  $\alpha \geq 1$ , we prove that:

- Any polynomial-time  $\rho(n)$ -approximation for Set Cover immediately yields a  $\rho(n)$ -approximation for the sparsest  $\alpha$ -navigable graph (with respect to both the number of edges and maximum out-degree).
- Conversely, a  $\rho(n)$ -approximation algorithm for the sparsest 1-navigable graph yields a  $\rho(\text{poly}(n))$ -approximation for Set Cover.

As a consequence, we obtain a polynomial-time  $(\ln n + 1)$ -approximation algorithm for the sparsest  $\alpha$ -navigable graph, and show that obtaining a better-than  $(c \ln n)$ -approximation (for some absolute  $c > 0$ ) is NP-hard, even for  $\alpha = 1$ .

The equivalence above is established by formulating  $\alpha$ -navigability constraints as covering conditions: for each source vertex, the set of outgoing edges must collectively cover all “navigation targets” in accordance with the underlying metric. This enables a direct application of the greedy Set Cover algorithm and its hardness bounds to the navigability setting.

**Fast Algorithms for Sparse Instances and Bicriteria Approximation for Dense Instances (Theorems 3 and 4).** The preceding connection to the Set Cover problem results in an  $O(n^3)$ -time algorithm which is not well-suited for large instances. We next show that it is possible to get faster algorithms, both when the solution size is small, and when it is large, albeit with a slight relaxation in requirements:

- For instances where the optimal  $\alpha$ -navigable graph has OPT edges, we give a randomized algorithm that outputs an  $O(\ln n)$ -approximation in  $\tilde{O}(n \cdot \text{OPT})$  time. The key is a *membership-query* implementation of the greedy Set Cover algorithm, allowing for an output-sensitive runtime that is sublinear when the minimum cover is small. The membership query model relies on queries of the form “*is element  $x$  contained in set  $S$* ” which, in our setting, corresponds to checking a single  $\alpha$ -navigability constraint and can be supported in  $O(1)$  time.<sup>5</sup>
- For general instances, with possibly dense optimal  $\alpha$ -navigable graphs, we give an algorithm running in  $\tilde{O}(n^\omega \log \Delta/\varepsilon)$  time ( $\omega$  is the matrix multiplication exponent) that produces an  $\alpha$ -navigable graph with at most  $O(\ln n)$  times the out-degree and number of edges of the sparsest  $2\alpha(1 + \varepsilon)$ -navigable graph, for any  $\varepsilon \in (0, 1)$ . A key insight underlying our algorithm is that we can use fast Boolean matrix multiplication to batch-verify navigability constraints.

---

<sup>5</sup>To our knowledge, known sublinear-time algorithms for Set Cover [IMR<sup>+</sup>18, KY14] operate in the adjacency-list model, which assumes sets are presented explicitly as lists of elements. In contrast, our setting is akin to an adjacency-matrix access model, and converting between the two models itself takes linear time.

**Lower Bound on Query Complexity (Theorem 5).** We prove a strong lower bound in the black-box metric access model: any algorithm making  $o(n^2)$  queries to the metric cannot guarantee even an  $o(n)$ -approximation to the sparsest  $\alpha$ -navigable graph problem (under both objectives) even when there is an optimal solution with maximum out-degree 3. In other words, any non-trivial approximation necessitates examining essentially all pairwise distances in the metric. Thus in the setting of sparse navigable graphs, our bound obtained in Theorem 3 is essentially best possible.

**Very Recent Independent Work.** Very recently, in an independent work, Conway et al. [CDFC<sup>+</sup>25] also studied sparse navigability and its connection to the Set Cover problem. In particular, for the problem of computing an  $O(\log n)$ -approximation to the sparsest  $\alpha$ -navigable graph, they give an  $\tilde{O}(n^2)$ -time algorithm when  $\alpha = 1$ , and an  $\tilde{O}(\min\{n^{2.5}, n \cdot \text{OPT}\})$ -time algorithm when  $\alpha > 1$ .

## 1.2 Organization of the Paper

Section 2 defines the Sparsest Navigable Graph problem and introduces key preliminaries. Section 3 gives a worst-case approximation lower bound for the slow-preprocessing version of DiskANN. Section 4 establishes a connection to Set Cover, yielding an approximation algorithm and NP-hardness result. In Section 5, we present faster approximation algorithms for both sparse and dense instances. Finally, Section 6 proves a strong query complexity lower bound.

## 2 Preliminaries

### 2.1 The Sparsest Navigable Graph Problem

For any directed graph  $G = (P, E)$ , let  $\deg_G(s)$  denote the *out-degree* of vertex  $s \in P$ .

In the Sparsest Navigable Graph problem, we are given as input a finite metric space  $(P, d)$  and a parameter  $\alpha \geq 1$ . The goal is to output an  $\alpha$ -navigable graph  $G = (P, E)$  that approximates the sparsest  $\alpha$ -navigable graph on  $P$ , with respect to either the maximum out-degree or the total number of edges. We formalize our notion of approximation below.

**Definition 2.1** (Approximate Sparsest Navigable Graph). Let  $(P, d)$  be a finite metric space and  $\alpha \geq 1$ . Let  $G = (P, E)$  be an  $\alpha$ -navigable graph.

- $G$  is a  $c$ -approximation to the sparsest  $\alpha$ -navigable graph under the max-out-degree objective if  $\max_{s \in P} \deg_G(s)$  is at most  $c$  times the maximum out-degree of any  $\alpha$ -navigable graph on  $P$ .
- $G$  is a  $c$ -approximation to the sparsest  $\alpha$ -navigable graph under the average-degree objective if  $|E|$  is at most  $c$  times the number of edges in any  $\alpha$ -navigable graph on  $P$ .

Recall that in the worst-case analysis of graph-based nearest neighbor search introduced by [IX23], the running time to return an  $(\frac{\alpha+1}{\alpha-1} + \varepsilon)$ -approximate nearest neighbor query on an  $\alpha$ -navigable graph  $G$  is bounded by  $O(\log_\alpha(\frac{\Delta}{(\alpha-1)\varepsilon}))$  times the maximum out-degree of  $G$ . The space complexity of the corresponding data structure is proportional the average degree of  $G$ . Therefore, a  $c$ -approximation to the sparsest  $\alpha$ -navigable graph with respect to the max-out-degree (resp. average-degree) objective incurs a  $c$ -factor overhead in query time (resp. space complexity) relative to the best possible such graph under this analysis.

## 2.2 The Set Cover Problem

**Definition 2.2** (Set-Cover). An instance of Set-Cover is given by a pair  $(\mathcal{U}, \mathcal{F})$ , where  $\mathcal{U}$  is a set and  $\mathcal{F}$  is a collection of subsets of  $\mathcal{U}$ . A cover is a subcollection  $\mathcal{T} \subset \mathcal{F}$  whose union covers all elements in  $\mathcal{U}$ , and the task is to output a cover of minimal size.

**Lemma 2.1** (Theorem 4 of [Joh73]). *Let  $(\mathcal{U}, \mathcal{F})$  be an instance of Set-Cover with  $|\mathcal{U}| = n$  and  $|\mathcal{F}| = m$ . There is an algorithm which returns a  $(\ln n + 1)$ -approximation to the optimal set cover in time*

$$O\left(\sum_{S \in \mathcal{F}} |S|\right) = O(mn).$$

**Lemma 2.2** (Corollary 4 of [DS14]). *Let  $\varepsilon > 0$  be a constant, and let  $(\mathcal{U}, \mathcal{F})$  be an instance of Set-Cover with  $|\mathcal{U}| = n$  and  $|\mathcal{F}| = m = n^{1+O(1/\varepsilon)} = n^{O(1)}$ . It is NP-hard to output a  $((1 - \varepsilon) \cdot \ln n)$ -approximation of the size of the minimum set cover.*

## 3 An $\tilde{\Omega}(n)$ -Approximation Lower Bound for Slow-DiskANN

In this section, we examine the slow-preprocessing variant of the DiskANN algorithm [JSDS<sup>+</sup>19] studied by Indyk and Xu [IX23]. This algorithm takes as input a finite metric space  $(P, d)$  and constructs an  $\alpha$ -navigable graph. We show that in the worst case, this particular algorithm outputs an  $\tilde{\Omega}(n)$ -approximation of the sparsest  $\alpha$ -navigable graph, under both the max-out-degree and average-degree objectives.

### 3.1 The Slow-DiskANN algorithm

Slow-DiskANN builds the out-neighborhood of each  $s \in P$  using a greedy covering rule. It repeatedly adds an edge from  $s$  to the nearest vertex in  $P \setminus \{s\}$  whose  $\alpha$ -navigability constraint is not yet covered. After an edge is added, it prunes all newly covered vertices, and this process repeats until all vertices are covered.

---

#### Algorithm 1: Slow-DiskANN [IX23]

---

```

Input: Finite metric space  $(P, d)$ , parameter  $\alpha \geq 1$ 
Output:  $\alpha$ -navigable graph  $G = (P, E)$ 
1 foreach  $s \in P$  do
2   Let  $U \leftarrow P \setminus \{s\}$  and sort  $U$  by increasing  $d(s, \cdot)$ ;
3   foreach  $u \in U$  in this order do
4     Update  $E \leftarrow E \cup \{(s, u)\}$ ;
5     Remove from  $U$  any  $t$  with  $d(u, t) < d(s, t)/\alpha$ ;

```

---

**Lemma 3.1** (Lemma 3.2 of [IX23]). *Algorithm 1 produces an  $\alpha$ -navigable graph  $G = (P, E)$  and runs in  $O(n \cdot |E|) = O(n^3)$  time.*

*Proof.* If  $s \neq t$ , then either  $(s, t) \in E$  or there was some vertex  $u \in P \setminus \{s\}$  responsible for pruning the edge  $(s, t)$ . In the latter case, this implies  $(s, u) \in E$  and  $d(u, t) \leq d(s, t)/\alpha$ , thus proving that  $G$  is  $\alpha$ -navigable. The runtime of  $G$  is bottlenecked by the pruning procedure, wherein the algorithm performs a linear scan of  $U$  every time an edge is added. Hence, the runtime is  $O(n \cdot |E|) = O(n^3)$ .  $\square$

[IX23] also gives a guarantee on the maximum degree of the graph produced by Slow-DiskANN, in terms of the *doubling dimension* [GKL03] of the metric. Specifically, if an  $n$ -point metric has doubling dimension  $\lambda$

and aspect ratio  $\Delta$ , they show that the  $\alpha$ -navigable graph returned by Slow-DiskANN has maximum degree at most  $O((4\alpha)^\lambda \cdot \log \Delta)$ .

### 3.2 Approximation lower bound for Slow-DiskANN

Despite the above guarantee for Slow-DiskANN, we now show that in the worst case, the graph it constructs can be an  $\tilde{\Omega}(n)$ -approximation to the sparsest  $\alpha$ -navigable graph, for both the max-out-degree and average-degree objectives. Indeed, while [IX23] proves that metrics with low doubling dimension admit sparse  $\alpha$ -navigable graphs, we construct high-dimensional Euclidean metrics (which will have large doubling dimension) and admit sparse navigable graphs; yet, Slow-DiskANN on these metrics outputs dense graphs.

**Theorem 1.** Slow-DiskANN (Algorithm 1) incurs a worst-case approximation factor of  $\Omega(n/\log n)$  to the sparsest  $\alpha$ -navigable graph, under both the max-out-degree and average-degree objectives.

The proof of Theorem 1 relies on a binary tree structure on the unit sphere in  $\mathbb{R}^n$ , defined below. This pointset naturally admits a navigable graph of maximum degree  $O(\log n)$ , but Slow-DiskANN produces a graph with  $\Omega(n^2)$  edges.

**Definition 3.1** (Euclidean Binary Tree Structure). Let  $n$  be a power of 2. We define a set of points  $P \subset \mathbb{R}^n$  of size  $2n - 1$ , consisting of unit vectors under the  $\ell_2$ -metric, as follows (see also, the accompanying Figure 1):

- For  $h \in \{0, 1, \dots, \log_2 n\}$ , partition  $[n]$  into  $n/2^h$  contiguous intervals of size  $2^h$ . Namely, for  $j \in [n/2^h]$ , define

$$I_{h,j} := ((j-1) \cdot 2^h, j \cdot 2^h] \cap \mathbb{Z}$$

The number of such intervals is  $2n - 1$ .

- For each interval  $I_{h,j}$ , define the corresponding vector  $x_{h,j} \in \mathbb{R}^n$  by

$$x_{h,j} := \frac{1}{\sqrt{|I_{h,j}|}} \sum_{i \in I_{h,j}} e_i = \frac{1}{2^{h/2}} \sum_{i \in I_{h,j}} e_i,$$

where  $e_i$  denotes the  $i$ -th standard basis vector. Note that by the above normalization,  $\|x_{h,j}\|_2 = 1$ .

Since  $P$  consists only of unit vectors, pairwise distances in  $P$  can be written in terms of inner products. The next claim characterizes the inner products between vectors in  $P$ .

**Claim 3.2.** For any  $x_{h,j}, x_{h',j'} \in P$ , we have

$$\langle x_{h,j}, x_{h',j'} \rangle = \begin{cases} 0 & \text{if } I_{h,j} \cap I_{h',j'} = \emptyset \\ 2^{-|h-h'|/2} & \text{otherwise} \end{cases}.$$

*Proof.* Observe that if two intervals  $I_{h,j}, I_{h',j'}$  in the above construction are not disjoint, then one is fully contained in the other. If  $I_{h,j} \cap I_{h',j'} = \emptyset$ , then it is clear that  $\langle x_{h,j}, x_{h',j'} \rangle = 0$ . Otherwise, if  $I_{h,j} \subseteq I_{h',j'}$ , then  $h \leq h'$  and

$$\langle x_{h,j}, x_{h',j'} \rangle = \frac{2^h}{2^{(h+h')/2}} = 2^{(h-h')/2}.$$

The case that  $I_{h,j} \supseteq I_{h',j'}$  follows similarly.  $\square$

**Lemma 3.3.** There exists a 1-navigable graph  $H = (P, E)$  with maximum out-degree  $O(\log n)$ .

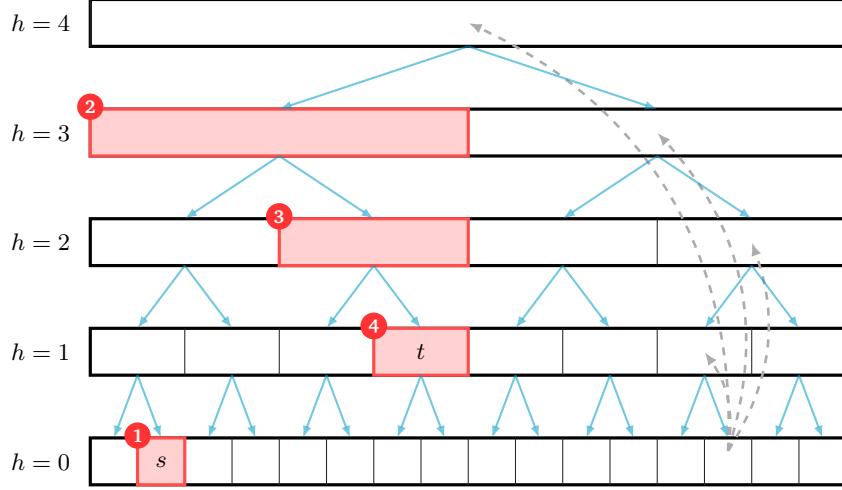


Figure 1: Depiction of a sparse 1-navigable graph on the pointset of Definition 3.1 with maximum degree  $O(\log n)$ , for  $n = 16$ . The horizontal bands represent “levels” of the binary tree  $h = 0$  to  $h = 4$ . Blue edges denote parent–child edges in the tree; grey dashed edges point from a node to its ancestors; the shaded red cells and their circled numbers depict the traversal of a navigable path from  $s = x_{0,2}$  to  $t = x_{1,4}$ .

*Proof.* We construct a navigable graph  $H = (P, E)$  as follows:

- Draw the (directed) binary tree rooted at  $x_{\log_2 n, 1}$ : for all  $h \in \{1, 2, \dots, \log_2 n\}$  and  $j \in [n/2^h]$ , we include the edges  $(x_{h,j}, x_{h-1,2j-1})$  and  $(x_{h,j}, x_{h-1,2j})$ .
- Draw an edge from every node to each of its ancestors in the above tree: for all  $h, h' \in \{0\} \cup [\log_2 n]$  and indices  $j \in [n/2^h]$  and  $j' \in [n/2^{h'}]$  where  $I_{h,j} \subset I_{h',j'}$ , draw the edge  $(x_{h,j}, x_{h',j'})$ .

The graph  $H$  is shown in Figure 1. Since each  $I_{h,j}$  is contained in  $(\log_2 n) - h$  intervals  $I_{h',j'}$ , it follows that  $H$  has maximum out-degree  $\log_2 n + 1 = O(\log n)$ .

We now check navigability of  $H$ . For  $s = x_{h,j}$  and  $t = x_{h',j'}$ , let  $x_{h^*,j^*}$  denote the least common ancestor in the binary tree, i.e.  $I_{h^*,j^*}$  is the smallest interval containing both  $I_{h,j}$  and  $I_{h',j'}$ .

- If  $x_{h',j'} = x_{h^*,j^*}$  then  $I_{h,j} \subset I_{h',j'}$ , which means  $(x_{h,j}, x_{h',j'}) \in E$ .
- If  $x_{h,j} = x_{h^*,j^*}$ , then for a child  $u \in \{x_{h^*-1,2j^*-1}, x_{h^*-1,2j^*}\}$  of  $x_{h^*,j^*}$ , we have  $(x_{h,j}, u) \in E$  and  $\|u - x_{h',j'}\|_2 < \|x_{h,j} - x_{h',j'}\|_2$  by Claim 3.2.
- Otherwise, it must be that  $I_{h,j} \cap I_{h',j'} = \emptyset$ . For  $u = x_{h^*,j^*}$ , then, we have  $(s, u) \in E$  and  $\|u - t\|_2 < \sqrt{2} = \|s - t\|_2$ .

Thus, it follows that  $H$  is navigable.  $\square$

**Lemma 3.4.** *Running Algorithm 1 on  $P$  with  $\alpha = 1$  produces a graph  $G$  containing  $\Omega(n^2)$  edges.*

*Proof.* We will show that for all  $i \in [n]$ , we have  $\deg_G(x_{0,i})$ , encoding the size of the out-degree of  $x_{0,i}$  in  $G$  being  $\Omega(n)$ . Consider the execution of Algorithm 1 with  $s = x_{0,1}$ , corresponding to the interval  $I_{0,1} = \{1\}$ .

Since all vectors in  $P$  have unit norm, sorting by increasing distance to  $x_{0,1}$  is equivalent to storing by decreasing inner product  $\langle x_{0,1}, \cdot \rangle$ . By Claim 3.2, the first  $\log_2 n$  entries of the sorted list  $U$  will be ordered

$$x_{1,1}, x_{2,1}, \dots, x_{\log_2 n, 1}.$$

The remaining entries in  $U$  are equidistant to  $x_{0,1}$ . Algorithm 1 is deterministic, and orders vertices solely based on distance; hence, consider the next  $n - 1$  entries being

$$x_{0,2}, \dots, x_{0,n}.$$

The algorithm will begin by adding the edge  $(x_{0,1}, x_{1,1})$ , which will immediately prune from  $U$  all points  $x_{h,1}$  for  $h > 1$ , since

$$\langle x_{0,1}, x_{h,1} \rangle = 2^{-h/2} < 2^{-(h-1)/2} = \langle x_{1,1}, x_{h,1} \rangle.$$

Moreover, as  $I_{1,1} = \{1, 2\}$ , the algorithm will prune  $x_{0,2}$  from  $U$ , but will not prune any  $x_{0,j}$  for  $j > 2$ . After the first round of pruning, then, the next  $n - 2$  entries in  $U$  will be

$$\{x_{0,3}, \dots, x_{0,n}\}.$$

Since all distances in  $\{x_{0,1}, \dots, x_{0,n}\}$  are equal, the algorithm will add all edges of the form  $(x_{0,1}, x_{0,j})$  for  $j > 2$ . Hence,  $\deg_G(x_{0,1}) = n - 2$ , and by symmetry of  $P$ , it follows that  $\deg_G(x_{0,i}) = n - 2$  for all  $i \in [n]$ . Hence,  $G$  has  $\Omega(n^2)$  edges.  $\square$

We are now ready to prove Theorem 1.

*Proof of Theorem 1.* Consider the set  $P \subset \mathbb{R}^n$  from Definition 3.1 under the Euclidean metric. By Lemma 3.3, there exists a 1-navigable graph on  $P$  with maximum out-degree  $O(\log n)$  and  $O(n \log n)$  edges. However, by Lemma 3.4, the graph output by Algorithm 1 has  $\Omega(n^2)$  edges and maximum out-degree  $\Omega(n)$ . It follows that Algorithm 1 returns an  $\Omega(n/\log n)$ -approximation to the sparsest navigable graph on  $P$ , for both the max-out-degree and average-degree objectives.  $\square$

**Remark 3.1.** The  $\Omega(n^2)$  lower bound in Lemma 3.4 relies on the assumption that the sorting of  $U$  breaks ties in an adversarial manner. This behavior can be enforced deterministically by slightly perturbing the construction: for instance, by shrinking the bottom-level vectors  $x_{0,j}$  by a factor of  $(1 - \varepsilon)$  for sufficiently small  $\varepsilon > 0$ . It can be checked that this modification preserves Lemma 3.3, while ensuring that Algorithm 1 produces a graph with  $\Omega(n^2)$  edges regardless of its tiebreaking rule.

## 4 Sparse Navigability is Equivalent to Set Cover

In this section, we show that the problem of constructing the sparsest  $\alpha$ -navigable graph on a given metric is equivalent, in terms of its polynomial-time *approximability threshold*, to the classic Set-Cover problem (Definition 2.2). This connection yields a polynomial-time  $(\ln n + 1)$ -approximation algorithm for constructing the sparsest  $\alpha$ -navigable graph and a matching hardness of approximation result (up to a constant factor).

**Theorem 2.** Let  $(P, d)$  be an  $n$ -point metric, and let  $\alpha \geq 1$ .

- There is a polynomial-time algorithm that produces a  $(\ln n + 1)$ -approximation to the sparsest  $\alpha$ -navigable graph on  $P$  (under both the max-out-degree and average-degree objectives).
- There is a constant  $c \in (0, 1)$  such that it is NP-hard to produce a  $(c \ln n)$ -approximation to the sparsest  $\alpha$ -navigable graph on  $P$  (under either the max-out-degree or average-degree objectives), even when  $\alpha = 1$ .

The proof of Theorem 2 relies on a two-way approximation-preserving reduction between Sparsest Navigable Graph and Set-Cover. Lemma 4.2 shows that any polynomial-time  $\rho(n)$ -approximation algorithm for Set-Cover on a universe of size  $n$  can be used to construct a  $\rho(n)$ -approximation to the sparsest  $\alpha$ -navigable graph on an  $n$ -point metric. Conversely, Lemma 4.5 maps a Set-Cover instance with  $n$  elements and  $m$  sets to a metric of size  $N = (m + n)^{\Theta(1)}$ , and shows that a  $\rho(N)$ -approximation to the sparsest 1-navigable graph yields a  $\Theta(\rho(N))$ -approximation to the original Set-Cover instance. Since the polynomial-time approximation threshold for Set-Cover is  $\Omega(\ln N) = \Omega(\ln(m + n))$ , it follows that Sparsest Navigable Graph inherits the same approximation threshold up to a constant factor.

## 4.1 Reduction from Sparsest Navigable Graph to Set Cover

We begin by expressing  $\alpha$ -navigability as a set covering condition.

**Definition 4.1** (Set Cover Formulation of  $\alpha$ -Navigability). Let  $(P, d)$  be a metric space and  $\alpha \geq 1$ . For each  $s, u \in P$  with  $u \neq s$ , let  $Z_\alpha(s, u)$  denote the set of  $\alpha$ -navigability constraints covered by the edge  $(s, u)$ . Formally, we define

$$Z_\alpha(s, u) := \{t \in P \mid d(u, t) < d(s, t)/\alpha\} \subseteq P \setminus \{s\}.$$

For each  $s \in P$ , we define an instance  $(P \setminus \{s\}, \mathcal{F}_s)$  of Set-Cover, where  $\mathcal{F}_s := \{Z_\alpha(s, u) \mid u \in P \setminus \{s\}\}$ . We remark that the collection  $\mathcal{F}_s$  can be built in time  $O(n^2)$ .

**Claim 4.1.** *The following statements are equivalent:*

- $G = (P, E)$  is  $\alpha$ -navigable.
- For every  $s \in P$ ,  $\{Z_\alpha(s, u) \mid u \in N^{\text{out}}(s)\}$  forms a valid set cover for the instance  $(P \setminus \{s\}, \mathcal{F}_s)$ .

*Proof.* If  $G$  is  $\alpha$ -navigable, then for all  $s \in P$  and  $t \in P \setminus \{s\}$ , there is some  $u \in N^{\text{out}}(s)$  for which  $t \in Z_\alpha(s, u)$ . If  $G$  is not  $\alpha$ -navigable, then there is some  $s \in P$  and  $t \in P \setminus \{s\}$  such that  $t \notin Z_\alpha(s, u)$  for all  $u \in N^{\text{out}}(s)$ .  $\square$

The above equivalence allows us to reduce the task of producing an  $\alpha$ -navigable graph on  $P$  to solving  $|P|$  instances of Set-Cover. As a result, any approximation algorithm for Set-Cover immediately yields an approximation algorithm for constructing sparse  $\alpha$ -navigable graphs, with the same approximation factor.

**Lemma 4.2.** *Suppose there exists a polynomial-time  $\rho(N)$ -approximation algorithm for Set-Cover on instances  $(\mathcal{U}, \mathcal{F})$  with  $|\mathcal{U}| \leq N$ . Then, for any  $\alpha \geq 1$ , there exists a polynomial-time algorithm that computes a  $\rho(n)$ -approximation to the sparsest  $\alpha$ -navigable graph on any  $n$ -point metric space, under both the max-out-degree and average-degree objectives.*

*Proof.* Let  $\mathcal{A}$  be a  $\rho(N)$ -approximation algorithm for solving Set-Cover on universes of size at most  $N$ . Given an  $n$ -point metric space  $P \subset (X, d)$  and a parameter  $\alpha \geq 1$ , we describe the following algorithm for building an  $\alpha$ -navigable graph  $G = (P, E)$ . We begin with an initially empty graph  $G$ , and for each  $s \in P$ , we run the following process:

1. Compute the set system  $\mathcal{F}_s := \{Z_\alpha(s, u) \mid u \in P \setminus \{s\}\}$ .
2. Run  $\mathcal{A}$  on the Set-Cover instance  $(P \setminus \{s\}, \mathcal{F}_s)$  to obtain a solution  $\{Z_\alpha(s, u_1), \dots, Z_\alpha(s, u_{\hat{k}_s})\}$ .
3. For  $1 \leq i \leq \hat{k}_s$ , add the edge  $(s, u_i)$  to  $G$ .

Since each  $\mathcal{F}_s$  can be built in time  $O(n^2)$  and  $\mathcal{A}$  runs in polynomial time, the above algorithm runs in polynomial time. By Claim 4.1, the graph obtained by the above algorithm  $G$  is  $\alpha$ -navigable. To finish, we now argue that  $G$  is a  $\rho(n)$ -approximation to the sparsest  $\alpha$ -navigable graph on  $P$ , under both the max-out-degree and average-degree objectives.

Take any  $s \in P$  and let  $H$  be an  $\alpha$ -navigable graph on  $P$ . By Claim 4.1, the out-neighborhood of  $s$  in  $H$  yields a solution to the Set-Cover instance  $(P \setminus \{s\}, \mathcal{F}_s)$  of size  $\deg_H(s)$  (here,  $\deg_H(s)$  and  $\deg_G(s)$  denote the number of out-going edges of  $s$  in  $H$  and  $G$ , respectively). Since  $|P \setminus \{s\}| < n$ , the approximation guarantee of  $\mathcal{A}$  implies that it returns a solution of size  $\hat{k}_s \leq \rho(n) \cdot \deg_H(s)$ , which yields

$$\deg_G(s) \leq \rho(n) \cdot \deg_H(s).$$

Since this bound holds for all  $s \in P$ , we have  $\max_s \deg_G(s) \leq \rho(n) \cdot \max_s \deg_H(s)$ , and  $|E(G)| \leq \rho(n) \cdot |E(H)|$ . Therefore,  $G$  is a  $\rho(n)$ -approximation to the sparsest  $\alpha$ -navigable graph on  $P$  under both the max-out-degree and average-degree objectives.  $\square$

## 4.2 Reduction from Set Cover to Sparsest Navigable Graph

Suppose we are given an instance  $(\mathcal{U}, \mathcal{F})$  of Set-Cover, where  $\mathcal{U} = \{x_1, \dots, x_n\}$  and  $\mathcal{F} = \{S_1, \dots, S_m\}$ . We show how to construct, in polynomial time, a metric space  $(P_{(\mathcal{U}, \mathcal{F})}, d)$  such that approximations to the sparsest  $\alpha$ -navigable graph on  $P_{(\mathcal{U}, \mathcal{F})}$  (under either the max-out-degree or average-degree objective) correspond to approximations to the optimal set cover on  $(\mathcal{U}, \mathcal{F})$ .

**Definition 4.2** (Metric Construction for  $(\mathcal{U}, \mathcal{F})$ ). We define a weighted graph and take  $(P_{(\mathcal{U}, \mathcal{F})}, d)$  to be the induced shortest-path metric. The construction will depend on two parameters: a small constant  $\gamma > 0$ , and an integer  $L \geq m + n$ . We build the graph as follows (depicted in Figure 2):

- Create  $L$  distinct root vertices  $r_1, \dots, r_L$  and  $L$  identical gadgets encoding the Set-Cover instance  $(\mathcal{U}, \mathcal{F})$ .
- For each  $q \in [L]$ , the  $q$ -th gadget  $P^{(q)}$  will consist of  $m + n$  vertices corresponding to each  $S_i \in \mathcal{F}$  and  $x_j \in \mathcal{U}$ , labeled  $S_i^{(q)}$  and  $x_j^{(q)}$  respectively.
  - Every root vertex  $r_\ell$  will have an edge  $(r_\ell, S_i^{(q)})$  of weight 1. Additionally, for each  $i \neq i'$ , we draw an edge  $(S_i^{(q)}, S_{i'}^{(q)})$  of weight  $1 - \gamma$ .
  - Each inclusion  $x_j \in S_i$  will correspond to an edge  $(S_i^{(q)}, x_j^{(q)})$  of weight 1. Also, every root vertex  $r_\ell$  will have a direct “shortcut” edge  $(r_\ell, x_j^{(q)})$  of weight  $2 - \gamma$ .

The resulting metric space  $(P_{(\mathcal{U}, \mathcal{F})}, d)$  has size  $|P_{(\mathcal{U}, \mathcal{F})}| = L + L \cdot |P^{(q)}| = L \cdot (m + n + 1)$ .

To see how the Set-Cover instance is encoded in the above construction, consider a single gadget  $P^{(q)}$  and the navigability constraint from a fixed root vertex  $r_\ell$  to a vertex  $x_j^{(q)}$ . Observe that  $d(S_i^{(q)}, x_j^{(q)}) < d(r_\ell, x_j^{(q)}) = 2 - \gamma$  if and only if  $x_j \in S_i$ . Using this property, one can show that satisfying navigability from the root vertex  $r_\ell$  is roughly equivalent to constructing a valid set cover of  $\mathcal{U}$ . By duplicating the root vertices and gadgets by a factor of  $L$ , we magnify the contribution of the optimal cover size of  $(\mathcal{U}, \mathcal{F})$  towards both the max-out-degree and average-degree of the sparsest navigable graph on  $P_{(\mathcal{U}, \mathcal{F})}$ . The formal argument appears below.

**Claim 4.3.** *If  $(\mathcal{U}, \mathcal{F})$  has a solution of size OPT, then the metric  $(P_{(\mathcal{U}, \mathcal{F})}, d)$  admits a 1-navigable graph of maximum out-degree  $\max\{L \cdot \text{OPT}, m + n\}$  and with  $L^2 \cdot \text{OPT} + L(m + n)^2$  edges.*

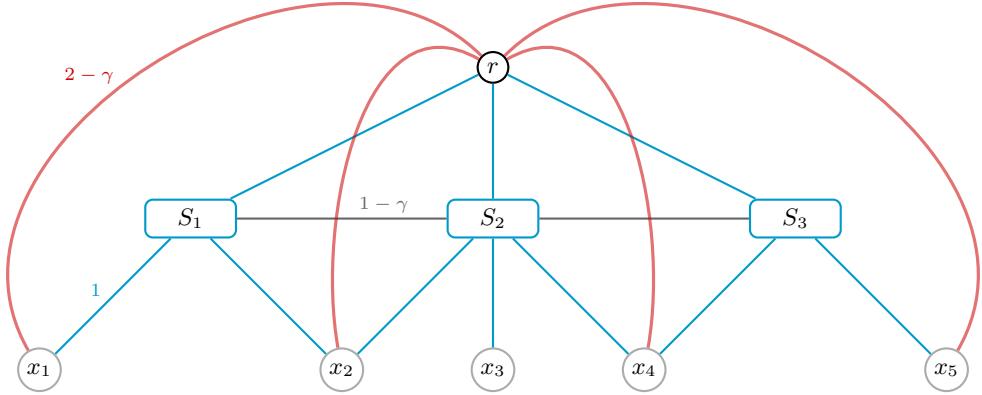


Figure 2: Weighted subgraph induced by a single gadget and one root vertex  $r$  in the metric space of Definition 4.2 for a Set-Cover instance with  $S_1 = \{x_1, x_2\}$ ,  $S_2 = \{x_2, x_3, x_4\}$ , and  $S_3 = \{x_4, x_5\}$ . Blue edges (weight 1) connect  $r$  to each set vertex and each set vertex to the universe elements it contains; red edges (weight  $2 - \gamma$ ) connect  $r$  to every element vertex; and grey edges (weight  $1 - \gamma$ ) connect set vertices. For visual clarity, the edges  $(S_1, S_3)$  and  $(r, x_3)$  are omitted.

*Proof.* Let  $\{S_{i_1}, \dots, S_{i_{\text{OPT}}}\}$  be a solution to  $(\mathcal{U}, \mathcal{F})$ . We define the a 1-navigable graph  $G = (P_{(\mathcal{U}, \mathcal{F})}, E)$ . We will describe the subgraph induced by the root vertices  $r_1, \dots, r_L$  and a single gadget  $P^{(q)}$ , and the entire graph will be the union of these identical subgraphs over all gadgets.

- For  $\ell \in [L]$  and each  $b \in [\text{OPT}]$ , add the edge  $(r_\ell, S_{i_b}^{(q)})$ .
- Include the complete graph on  $P^{(q)}$ , and add an edge from every vertex in  $P^{(q)}$  to  $r_1$  (this will ensure we satisfy the navigability constraints among vertices in different gadgets).

The out-degree of each root vertex  $r_\ell$  is precisely  $L \cdot \text{OPT}$ , and the out-degree of each  $S_i^{(q)}$  and  $x_j^{(q)}$  is  $m + n$ . Therefore,  $G$  has max out-degree  $\max\{L \cdot \text{OPT}, m + n\}$  and  $|E| = L^2 \cdot \text{OPT} + L(m + n)^2$ , as desired. It remains to argue that  $G$  is 1-navigable. Let  $s, t \in P_{(\mathcal{U}, \mathcal{F})}$ , and divide into a small number of cases.

- If  $s, t$  are contained in the same gadget  $P^{(q)}$ , then the direct edge  $(s, t) \in E$  satisfies the constraint.
- If  $s \in P^{(q)}$  and  $t \notin P^{(q)}$  is not  $r_1$ , then  $(s, r_1) \in E$  and  $d(r_1, t) \leq 2 - \gamma < 2 \leq d(s, t)$ .
- If  $s = r_\ell$  and  $t = r_{\ell'}$ , then  $(s, S_{i_1}^{(1)}) \in E$  and  $d(S_{i_1}^{(1)}, r_{\ell'}) = 1 < 2 = d(r_\ell, r_{\ell'})$ .
- If  $s = r_\ell$  and  $t = S_i^{(q)}$ , then since  $\text{OPT} \geq 1$ , we have  $(r_\ell, S_{i'}^{(q)}) \in E$  for some  $i' \in [m]$ , and

$$d(S_{i'}^{(q)}, S_i^{(q)}) = 1 - \gamma < 1 = d(r_\ell, S_i^{(q)}).$$

- Finally, if  $s = r_\ell$  and  $t = x_j^{(q)}$ , then there is some  $b \in [\text{OPT}]$  where  $x_j \in S_{i_b}$ . Thus,  $(r_\ell, S_{i_b}^{(q)}) \in E$  and

$$d(S_{i_b}^{(q)}, x_j^{(q)}) = 1 < 2 - \gamma = d(r_\ell, x_j^{(q)}).$$

Hence,  $G$  is 1-navigable.  $\square$

**Claim 4.4.** Let  $\text{OPT}$  be the size of the minimum set cover for  $(\mathcal{U}, \mathcal{F})$ . Then, in any 1-navigable graph  $G$  on  $(P_{(\mathcal{U}, \mathcal{F})}, d)$ , all root vertices  $r_1, \dots, r_L$  have out-degree at least  $L \cdot \text{OPT}$ .

*Proof.* Let  $G = (P_{(\mathcal{U}, \mathcal{F})}, E)$  be 1-navigable. We show that for every root vertex  $r_\ell$  and every gadget  $P^{(q)}$ , the out-neighborhood of  $r_\ell$ , denoted  $N^{\text{out}}(r_\ell)$  satisfies  $|N^{\text{out}}(r_\ell) \cap P^{(q)}| \geq \text{OPT}$ , which implies the claim. Fix  $r_\ell$  and  $P^{(q)}$ , and consider any element vertex  $x_j^{(q)}$ . Since  $G$  is 1-navigable, the navigability constraint from  $r_\ell$  to  $x_j^{(q)}$  means there is an edge  $(r_\ell, p) \in E$  such that  $d(p, x_j^{(q)}) < d(r_\ell, x_j^{(q)}) = 2 - \gamma$ . The metric is defined so that only two conditions may arise, and we build an assignment  $\sigma: N^{\text{out}}(r_\ell) \cap P^{(q)} \rightarrow \mathcal{F}$  which gives a set cover for  $(\mathcal{U}, \mathcal{F})$  accordingly.

- Either  $p = x_j^{(q)}$ , in which case we let  $\sigma(p) = S_i \in \mathcal{F}$ .
- Or,  $p = S_i^{(q)}$  for some set  $S_i \in \mathcal{F}$  containing  $x_j$ . In this case, we let  $\sigma(p) \in \mathcal{F}$  be an arbitrary set containing  $x_j$  (such a set must exist or  $(\mathcal{U}, \mathcal{F})$  does not admit any cover).

The case that  $p$  is another node can be ruled out. If  $p' = x_{j'}^{(q)}$  for  $j' \neq j$  or  $p' = x_{j'}^{(q')}$  for  $q' \neq q$ , then  $d(p', x_j^{(q)}) \geq 2$ . Note, the resulting assignment from the image of  $\sigma \subset \mathcal{F}$  forms a cover of  $\mathcal{U}$  of size at most  $|N^{\text{out}}(r_\ell) \cap P^{(q)}|$ , which implies  $|N^{\text{out}}(r_\ell) \cap P^{(q)}| \geq \text{OPT}$ . Since  $P^{(q)}$  and  $P^{(q')}$  are disjoint for  $q \neq q'$  and there are  $L$  such indices, the out-degree of each root  $r_\ell$  is of size at least  $L \cdot \text{OPT}$ .  $\square$

**Lemma 4.5.** Suppose there exists a polynomial-time algorithm that computes a  $\rho(N)$ -approximation to the sparsest 1-navigable graph on metric spaces of size at most  $N$ , under either the max-out-degree or average-degree objectives. Then, for any instance  $(\mathcal{U}, \mathcal{F})$  of Set-Cover with  $|\mathcal{U}| = n$  and  $|\mathcal{F}| = m$ , there exists a polynomial-time algorithm that computes a  $2\rho((n + m + 1)^3)$ -approximation to the size of the minimum set cover.

*Proof.* Let  $\mathcal{A}$  be a polynomial-time  $\rho(N)$ -algorithm for computing the sparsest 1-navigable graph (under the max-out-degree or average-degree objective) on metrics of size at most  $N$ . We describe an algorithm for approximating the size  $\text{OPT}$  of the optimal solution of an instance  $(\mathcal{U}, \mathcal{F})$  of Set-Cover.

1. Build the metric  $(P_{(\mathcal{U}, \mathcal{F})}, d)$  from Definition 4.2 with  $L = (m + n + 1)^2$ .
2. Run  $\mathcal{A}$  on  $(P_{(\mathcal{U}, \mathcal{F})}, d)$  to obtain a 1-navigable graph  $G = (P_{(\mathcal{U}, \mathcal{F})}, E)$ .
  - (a) If  $\mathcal{A}$  approximates the max-out-degree objective, choose an arbitrary root vertex  $r_\ell$  and return  $\deg_G(r_\ell)$ , the out-degree of  $r_\ell$  in  $G$ , divided by  $L$ .
  - (b) If  $\mathcal{A}$  approximates the average-degree objective, return  $|E|/L$ .

Since  $(P_{(\mathcal{U}, \mathcal{F})}, d)$  can be constructed in polynomial time and  $\mathcal{A}$  runs in polynomial time, this algorithm runs in polynomial time. We claim that the above algorithm returns a  $2\rho((m + n + 1)^3)$ -approximation of  $\text{OPT}$ . First, suppose that  $\mathcal{A}$  approximates with respect to the max-out-degree objective, and observe that by Claim 4.3 and  $L \geq m + n$ , there is a 1-navigable graph on  $P_{(\mathcal{U}, \mathcal{F})}$  of max out-degree at most  $L \cdot \text{OPT}$ . Therefore,

$$\deg_G(r_\ell) \leq \rho(|P_{(\mathcal{U}, \mathcal{F})}|) \cdot L \cdot \text{OPT} = \rho((m + n + 1)^3) \cdot L \cdot \text{OPT}$$

Also, by Claim 4.4, 1-navigability of  $G$  implies that  $\deg_G(r_\ell) \geq L \cdot \text{OPT}$ , and thus

$$\frac{\deg_G(r_\ell)}{L} \in [\text{OPT}, \rho((m + n + 1)^3) \cdot \text{OPT}] .$$

Next, suppose that  $\mathcal{A}$  approximates with respect to the average-degree objective, and observe that by Claim 4.3 and the setting of  $L = (m + n + 1)^2$ , there is a navigable graph on  $P_{(\mathcal{U}, \mathcal{F})}$  with at most  $2 \cdot (m + n + 1)^4 \cdot \text{OPT} = 2L^2 \cdot \text{OPT}$  edges. Therefore,

$$|E| \leq \rho((m + n + 1)^3) \cdot 2L^2 \cdot \text{OPT} .$$

By Claim 4.4, we must have  $|E| \geq L^2 \cdot \text{OPT}$ , and thus

$$\frac{|E|}{L^2} \in [\text{OPT}, 2\rho((m+n+1)^3) \cdot \text{OPT}],$$

as desired.  $\square$

**Remark 4.1.** Although Lemma 4.5 only establishes an approximation to the size of the optimal set cover, the reduction can be extended to recover an approximately optimal set cover itself using a standard averaging argument.

We are now ready to prove the main theorem of this section.

*Proof of Theorem 2.* We begin with the first statement. By Lemma 2.1, there is a polynomial-time  $(\ln N + 1)$ -approximation algorithm for Set-Cover on instances  $(\mathcal{U}, \mathcal{F})$  with  $|\mathcal{U}| = N$ . Combining with Lemma 4.2, this implies that for any  $n$ -point metric  $(P_{(\mathcal{U}, \mathcal{F})}, d)$  and  $\alpha \geq 1$ , there is a polynomial-time algorithm from computing a  $(\ln n + 1)$ -approximation to the sparsest  $\alpha$ -navigable graph on  $P_{(\mathcal{U}, \mathcal{F})}$ , under both the max-out-degree and average-degree objectives.

We now show that it is NP-hard to compute a  $(c \ln N)$ -approximation to the sparsest  $\alpha$ -navigable graph (under either the max out-degree or average-degree objectives) on metrics of size  $N$ , for  $\alpha = 1$  and a constant  $c \in (0, 1)$  to be specified later. Suppose that there exists a polynomial-time algorithm achieving such an approximation. Then by Lemma 4.5, this would imply a polynomial-time algorithm for approximating the optimal set cover on instances  $(\mathcal{U}, \mathcal{F})$  within factor

$$2c \cdot \ln(n+m+1)^3 \leq 6c \cdot \ln(n+m+1),$$

where  $|\mathcal{U}| = n$  and  $|\mathcal{F}| = m$ . However, by Lemma 2.2 (with  $\varepsilon = 1/2$ ), it is NP-hard to approximate the minimum set cover to within factor  $\frac{1}{2} \ln n$ , even when  $|\mathcal{U}| = n$  and  $|\mathcal{F}| \leq n^D$  for some constant  $D > 1$ . Since  $n+m+1 \leq 2n^D$ , we have:

$$\ln(n+m+1) \leq \ln(2n^D) = D \ln n + \ln 2 \leq 2D \ln n,$$

for sufficiently large  $n$ . Thus, the approximation factor  $6c \ln(n+m+1)$  is at most  $12cD \ln n$ . Thus, for the setting of  $c = \frac{1}{24D}$ , it is NP-hard to compute a  $(c \ln N)$ -approximation to the sparsest navigable graph, as desired.  $\square$

**Remark 4.2.** The  $(\ln n + 1)$ -approximation algorithm from the first part of Theorem 2 runs in time  $O(n^3)$ . This is because the reduction in Lemma 4.2 takes  $O(n^2)$  time to construct each of the  $n$  instances of Set-Cover, and the greedy algorithm from Lemma 2.1 solves each of these instances in time  $O(n^2)$ . The  $O(n^3)$  runtime matches that of Slow-DiskANN (Lemma 3.1), but with a vastly better worst-case approximation guarantee: while Slow-DiskANN can incur an  $\Omega(n/\log n)$ -approximation in the worst case (Theorem 1), the set-cover-based algorithm guarantees a  $(\ln n + 1)$ -approximation for all inputs.

## 5 Fast Approximation Algorithms for Sparse Navigability

Theorem 2 gives a  $(\ln n + 1)$ -approximation algorithm for the sparsest  $\alpha$ -navigable graph, running in  $O(n^3)$  time, and it shows that improving this approximation to  $o(\ln n)$  is NP-hard. A natural question is whether the  $O(n^3)$  runtime can be improved for a similar approximation guarantee. In the first main result of this section, we show that if the sparsest  $\alpha$ -navigable graph on a given metric has  $\text{OPT}_{\text{size}}$  edges, then we can compute an  $O(\ln n)$ -approximation in  $\tilde{O}(n \cdot \text{OPT}_{\text{size}})$  time. This yields up to a near-linear speedup for input metrics which admit sparse  $\alpha$ -navigable graphs.

**Theorem 3.** Let  $(P, d)$  be an  $n$ -point metric space and  $\alpha \geq 1$ .

- There exists a randomized algorithm which outputs an  $O(\ln n)$ -approximation to the sparsest  $\alpha$ -navigable graph on  $P$  (under both the max-out-degree and average-degree objectives), with high probability.
- Letting  $\text{OPT}_{\text{size}}$  denote the minimum number of edges in any  $\alpha$ -navigable graph on  $P$ , the algorithm runs in time  $\tilde{O}(n \cdot \text{OPT}_{\text{size}})$ .

The key ingredient in the proof of Theorem 3 is Lemma 5.1 (stated below), which provides a  $O(\ln n)$ -approximation algorithm for Set-Cover in the membership-query access model. The benefit (as opposed to directly utilizing Lemma 2.1) is that the algorithm of Lemma 5.1 finds a cover faster if the optimal cover is small.

In the second main result of this section, we show that potentially faster algorithms are possible, with running time  $\tilde{O}(n^\omega \log \Delta/\varepsilon)$  for any  $\varepsilon \in (0, 1)$ , even when  $\text{OPT}_{\text{size}}$  is large. The speedup is achieved by allowing a bicriteria approximation (i.e., incorporating a relaxation of the navigability parameter  $\alpha$ ). Specifically, we give an  $\tilde{O}(n^\omega \log \Delta/\varepsilon)$ -time algorithm that produces an  $\alpha$ -navigable graph whose maximum out-degree and number of edges are within an  $O(\ln n)$ -factor of those of the sparsest  $2\alpha(1 + \varepsilon)$ -navigable graph.

**Definition 5.1** (Bicriteria Approximation to the Sparsest Navigable Graph). Let  $(P, d)$  be a finite metric space and  $\beta \geq \alpha \geq 1$ . We say that  $G = (P, E)$  is a  $(\alpha, \beta)$ -bicriteria  $c$ -approximation to the sparsest  $\alpha$ -navigable graph under the  $\{\text{max-out-degree, average-degree}\}$  objective if  $G$  is  $\alpha$ -navigable and:

- (max-out-degree)  $\max_{s \in P} \deg_G(s)$  is at most  $c$  times the max out-degree of any  $\beta$ -navigable graph on  $P$ .
- (average-degree)  $|E|$  is at most  $c$  times the number of edges in any  $\beta$ -navigable graph on  $P$ .

**Remark 5.1.** For  $\beta \geq \alpha \geq 1$ ,  $\beta$ -navigability imposes tighter constraints than  $\alpha$ -navigability. Namely, any edge  $(s, u)$  satisfying the  $\beta$ -navigability constraint from  $s$  to  $t$ , i.e.,  $d(u, t) < d(s, u)/\beta$ , also satisfies the corresponding  $\alpha$ -navigability constraint. Hence, the sparsest  $\beta$ -navigable graph will contain at least as many edges as the sparsest  $\alpha$ -navigable graph, meaning that Definition 5.1 is indeed a relaxation of the Sparsest Navigable Graph problem.

**Theorem 4.** Let  $(P, d)$  be an  $n$ -point metric space with aspect ratio  $\Delta$ , and let  $\alpha \geq 1$ .

- For any  $\varepsilon \in (0, 1)$ , there exists a randomized algorithm which outputs an  $(\alpha, 2\alpha(1 + \varepsilon))$ -bicriteria  $O(\ln n)$ -approximation to the sparsest navigable graph on  $P$  (under both the max-out-degree and average-degree objectives), with high probability.
- This algorithm runs in time  $\tilde{O}(n^\omega \log \Delta/\varepsilon)$ .

## 5.1 Faster Sparse Navigability via Membership Set Cover

Definition 4.1 and Lemma 4.2 reduce the construction of a sparse  $\alpha$ -navigable graph on an  $n$ -point metric to solving  $n$  instances of Set-Cover, where each potential edge  $(s, u)$  corresponds to a set  $Z_\alpha(s, u)$  of navigability constraints it would satisfy. In the standard formulation of Set-Cover (Definition 2.2), each set  $S \in \mathcal{F}$  is represented explicitly as a list of elements from the universe  $\mathcal{U}$ . Under this representation, the runtime of the reduction is  $O(n^3)$ , since there are  $\binom{n}{2}$  such sets to construct, each taking up to  $O(n)$  time to build. However, each set  $Z_\alpha(s, u)$  is implicitly defined by the distances in  $d$ , so we design algorithms for Set-Cover which implicitly work with  $Z_\alpha(s, u)$  via membership queries (defined below).

**Definition 5.2** (Set-Cover in the Membership-Query Model). An algorithm for Set-Cover in the *membership-query* model accesses the input  $(\mathcal{U}, \mathcal{F})$  only through (constant-time) queries of the form

$$\text{contains}(S, x) := \begin{cases} 1 & \text{if } x \in S \\ 0 & \text{if } x \notin S \end{cases}$$

for  $x \in \mathcal{U}$  and  $S \in \mathcal{F}$ .

For  $n = |\mathcal{U}|$  and  $m = |\mathcal{F}|$ , the following lemma gives an algorithm computing an  $O(\ln n)$ -approximation to the minimum set cover in the membership-query model that runs in time  $\tilde{O}(mk + nk)$ , where  $k$  is the size of the minimum cover. Note that for  $k \ll n$ , this runtime is much faster than  $O(mn)$ , the number of membership queries needed to explicitly construct all sets in  $\mathcal{F}$ .

**Lemma 5.1.** *There is a randomized algorithm in the membership-query model (Algorithm 2) that, given an instance  $(\mathcal{U}, \mathcal{F})$  of Set-Cover with  $|\mathcal{U}| = n$ ,  $|\mathcal{F}| = m$ , and minimum cover size  $k$ , runs in time  $\tilde{O}(mk + nk)$  and computes an  $O(\ln n)$ -approximate set cover with probability at least  $1 - O(1/n^2)$ .*

The proof of the lemma is deferred to Subsection 5.1.1. We now prove Theorem 3, assuming Lemma 5.1.

*Proof of Theorem 3.* Let  $(P, d)$  be an  $n$ -point metric and  $\alpha \geq 1$ , and let  $\mathcal{A}$  be the membership-query algorithm for Set-Cover from Lemma 5.1. For each  $s \in P$ , define the set system  $\mathcal{F}_s := \{Z_\alpha(s, u) \mid u \in P \setminus \{s\}\}$ , as in Definition 4.1. By Claim 4.1, a graph is  $\alpha$ -navigable if and only if for each  $s \in P$ , the out-neighbors  $N^{\text{out}}(s)$  form a set cover for  $(P \setminus \{s\}, \mathcal{F}_s)$ .

The algorithm for building an  $\alpha$ -navigable graph proceeds as follows: run  $\mathcal{A}$  on each instance  $(P \setminus \{s\}, \mathcal{F}_s)$  using only membership queries, and construct the graph corresponding to the returned solutions to these instances. For each  $u, t \in P \setminus \{s\}$ , the function  $\text{contains}(Z_\alpha(s, u), t)$  tests whether  $d(u, t) < d(s, t)/\alpha$ , which takes constant time. Each instance succeeds with probability  $1 - O(1/n^2)$ , so all  $n$  instances succeed with probability  $1 - o(1)$ , which we will assume for the remainder of the proof. We now prove the two approximability guarantees.

For each  $s \in P$ , let  $k_s$  denote the minimum out-degree of  $s$  in any  $\alpha$ -navigable graph on  $P$ . It is clear that  $\sum_s k_s \leq \text{OPT}_{\text{size}}$ . By Claim 4.1,  $k_s$  is also the size of the minimum set cover for the instance  $(P \setminus \{s\}, \mathcal{F}_s)$ . By Lemma 5.1, the algorithm  $\mathcal{A}$  runs in  $\tilde{O}(nk_s)$  time for each  $s$ , returning an  $O(\ln n)$ -approximate set cover of size  $O(k_s \ln n)$ . Thus, in time

$$\sum_{s \in P} \tilde{O}(nk_s) = \tilde{O}(n \cdot \text{OPT}_{\text{size}}),$$

we produce an  $\alpha$ -navigable graph with maximum degree

$$\max_{s \in P} O(k_s \ln n) = O(\ln n) \cdot \max_{s \in P} k_s$$

and size

$$\sum_{s \in P} O(k_s \ln n) = O(\ln n) \cdot \text{OPT}_{\text{size}}.$$

Therefore, the output of the algorithm is an  $O(\ln n)$ -approximation to the sparsest  $\alpha$ -navigable graph on  $P$ , under both the max-out-degree and average-degree objectives.  $\square$

### 5.1.1 Proof of Lemma 5.1

**Algorithm Overview.** Let  $(\mathcal{U}, \mathcal{F})$  be an instance of Set-Cover with  $n = |\mathcal{U}|$  and  $m = |\mathcal{F}|$ , and let  $k$  denote the size of the minimum set cover. The classic greedy algorithm (Lemma 2.1) proceeds in rounds, and selects at

each round a set in  $\mathcal{F}$  that covers at least a  $1/k$ -fraction of the uncovered elements (such a set is guaranteed to exist because there is always cover of size at most  $k$ ). After  $r$  rounds, there are at most  $(1 - 1/k)^r n$  uncovered elements, guaranteeing a cover once  $r > k \ln n$ . Our aim is to simulate the above greedy approach via random sampling as efficiently as possible.

We design a subroutine FindHeavySet (Algorithm 3) that simulates a round of the greedy set cover algorithm. The subroutine takes as input a parameter  $\hat{k}$ , which serves as a guess for the size of the optimal cover  $k$ , and uses it to determine the number of randomly drawn sets and elements needed. Specifically, it samples random collections of sets from  $\mathcal{F}$  and estimates their sizes by performing membership queries against a random sample of elements from  $\mathcal{U}^{\text{alive}}$ , the set of currently uncovered elements. The sampling parameters are chosen so that FindHeavySet makes only  $\tilde{O}(m)$  membership queries and, if  $\hat{k} \geq k$ , then with high probability it returns a set covering an  $\Omega(1/k)$ -fraction of the remaining uncovered elements, which is sufficient for the desired  $O(\ln n)$ -approximation.

The main algorithm, FastSetCover (Algorithm 2), maintains the set  $\mathcal{U}^{\text{alive}}$  of uncovered elements, which is initialized to  $\mathcal{U}$ . In each iteration, it attempts to find a set covering a sufficiently large fraction of  $\mathcal{U}^{\text{alive}}$  by calling FindHeavySet with a current guess  $\hat{k}$  for the optimal cover size  $k$ . If a set  $\mathbf{S} \in \mathcal{F}$  is found, it updates  $\mathcal{U}^{\text{alive}}$  to  $\mathcal{U}^{\text{alive}} \setminus \mathbf{S}$ . If no set is found,  $\hat{k}$  is doubled and the process repeats. Once  $\mathcal{U}^{\text{alive}}$  is empty, the algorithm returns the selected sets.

---

**Algorithm 2:** FastSetCover( $\mathcal{U}, \mathcal{F}$ )

---

**Input:** a Set-Cover instance  $(\mathcal{U}, \mathcal{F})$  in the membership-query model  
**Output:** a set cover  $\mathcal{C}$  of  $(\mathcal{U}, \mathcal{F})$

```

1 Initialize  $\mathcal{C} = \{\}$  and  $\mathcal{U}^{\text{alive}} = \mathcal{U}$ ;  

2 Initialize  $\hat{k} = 1$ ;  

3 while  $\mathcal{U} \neq \emptyset$  do  

4    $\mathbf{S} \leftarrow \text{FindHeavySet}(\mathcal{U}^{\text{alive}}, \mathcal{F}, \hat{k}, |\mathcal{U}|)$ ;  

5   if  $\mathbf{S} = \text{FAIL}$  then  

6     Update  $\hat{k} \leftarrow 2 \cdot \hat{k}$ ;  

7     continue  

8   Update  $\mathcal{C} \leftarrow \mathcal{C} \cup \{\mathbf{S}\}$ ;  

9   Update  $\mathcal{U}^{\text{alive}} \leftarrow \mathcal{U}^{\text{alive}} \setminus \mathbf{S}$ ;  

10 return  $\mathcal{C}$ ;
```

---



---

**Algorithm 3:** FindHeavySet( $\mathcal{U}^{\text{alive}}, \mathcal{F}, \hat{k}, n$ )

---

**Input:** set of uncovered elements  $\mathcal{U}^{\text{alive}} \subseteq \mathcal{U}$ , collection of subsets  $\mathcal{F} \subset 2^{\mathcal{U}}$ , parameter  $\hat{k}$   
**Output:** a set  $\mathbf{S} \in \mathcal{F}$  with large intersection  $|\mathbf{S} \cap \mathcal{U}^{\text{alive}}|$

```

1  $m = |\mathcal{F}|$ ;  

2 for  $i = 1$  to  $\lceil \log_2 \hat{k} \rceil$  do  

3   Sample  $R = \lceil m \log n \cdot 2^{-(i-2)} \rceil$  sets  $\mathbf{S}_1, \dots, \mathbf{S}_R \stackrel{\text{i.i.d.}}{\sim} \text{Unif}(\mathcal{F})$ ;  

4   Sample  $T = 48 \log(mn) \cdot \min \left\{ 2^{i+3} \log n, 2\hat{k} \right\}$  elements  $\mathbf{x}_1, \dots, \mathbf{x}_T \stackrel{\text{i.i.d.}}{\sim} \text{Unif}(\mathcal{U}^{\text{alive}})$ ;  

5   for  $r = 1$  to  $R$  do  

6     if  $\#\{t \mid \text{contains}(\mathbf{S}_r, \mathbf{x}_t) = 1\} \geq 24 \log(mn)$  then  

7       return  $\mathbf{S}_r$ ;  

8 return FAIL;
```

---

We start by analyzing the performance of FindHeavySet. We begin with an elementary fact about sequences with bounded sum, which will help us reason about the distribution of set sizes in an optimal cover.

**Claim 5.2.** Let  $\alpha_1 \geq \alpha_2 \geq \dots \geq \alpha_L \geq 0$  be non-negative numbers with  $\sum_{j=1}^L \alpha_j = A$ . Then, there exists an index  $\ell \in [L]$  such that  $\ell \cdot \alpha_\ell \geq A/H_L$ , where  $H_L$  denotes the  $L$ -th harmonic number.

*Proof.* Deferred to Appendix A. □

Next, we quantify the number of uniform samples needed to estimate the size of a given set.

**Claim 5.3.** Let  $S \subset [N]$  be fixed, and let  $\mathbf{X} := \{x_1, \dots, x_T\}$  where  $x_i \stackrel{\text{i.i.d.}}{\sim} \text{Unif}([N])$ . Then for all  $\alpha \in [0, 1]$ , the following bounds hold with probability at least  $1 - \exp(-\alpha T/12)$ :

- (i) If  $|S| \geq \alpha N$ , then  $|S \cap \mathbf{X}| \geq \frac{\alpha T}{2}$ .
- (ii) If  $|S| \leq \frac{\alpha N}{4}$ , then  $|S \cap \mathbf{X}| < \frac{\alpha T}{2}$ .

*Proof.* Deferred to Appendix A. □

Using Claims 5.2 and 5.3, we prove a desired guarantee on the performance of FindHeavySet.

**Lemma 5.4.** Let  $(\mathcal{U}, \mathcal{F})$  be an instance of Set-Cover with  $|\mathcal{U}| = n$ ,  $|\mathcal{F}| = m$ , and with minimum cover size  $k$ . For  $\mathcal{U}^{\text{alive}} \subseteq \mathcal{U}$ , The algorithm  $\text{FindHeavySet}(\mathcal{U}^{\text{alive}}, \mathcal{F}, \hat{k}, n)$  runs in time  $\tilde{O}(m)^6$  and, with probability at least  $1 - O(1/n^3)$ , has the following properties.

- If  $\hat{k} \geq k$ , then  $\text{FindHeavySet}$  does not return FAIL
- If  $\text{FindHeavySet}$  does not return FAIL, then it returns a set  $\mathbf{S} \in \mathcal{F}$  satisfying

$$|\mathbf{S} \cap \mathcal{U}^{\text{alive}}| \geq |\mathcal{U}^{\text{alive}}|/(8\hat{k}).$$

*Proof.* We first bound the runtime of  $\text{FindHeavySet}(\mathcal{U}^{\text{alive}}, \mathcal{F}, \hat{k}, n)$ . The outer loop of  $\text{FindHeavySet}$  runs over at most  $\log_2 k \leq O(\log n)$  iterations. In each iteration, we make  $R + T = O(RT)$  random samples, each of which take  $O(\log n)$  time by Claim 5.5; and we make  $RT$  queries to  $\text{contains}(\cdot, \cdot)$ . Since  $RT = O(m \log^2 n \log(mn))$ , the total runtime is  $O(RT \log^2 n) = O(m \log^3 n \log(mn)) = \tilde{O}(m)$ .

We now show the desired properties. For each set  $S \in \mathcal{F}$ , denote  $\tilde{S} := S \cap \mathcal{U}^{\text{alive}}$ , and write  $N = |\mathcal{U}^{\text{alive}}|$ . By assumption, there is a set cover for  $(\mathcal{U}^{\text{alive}}, \mathcal{F})$  of size  $k$ , meaning there are  $k$  sets  $S_1, \dots, S_k \in \mathcal{F}$  with  $|\tilde{S}_1| \geq \dots \geq |\tilde{S}_k|$  such that

$$|\tilde{S}_1| + \dots + |\tilde{S}_k| \geq N.$$

Let  $L$  be the largest index in  $[k]$  for which  $|\tilde{S}_L| \geq N/(2k)$ . Then, we have

$$|\tilde{S}_1| + \dots + |\tilde{S}_L| \geq N/2.$$

By Claim 5.2, there is some  $\ell \in [L]$  for which  $\ell \cdot |\tilde{S}_\ell| \geq N/(2H_L) > N/(4 \log n)$ , where we have used that  $2H_L \leq 2(\log L + 1) < 4 \log n$ . This implies the following key property:

For some  $\ell \leq k$ , there are  $\ell$  sets  $S \in \mathcal{F}$  with  $|\tilde{S}| \geq N \cdot \max \left\{ \frac{1}{4\ell \log n}, \frac{1}{2k} \right\}$ . (\*)

We now prove the first bullet point. When  $k \leq \hat{k}$ , there is an iteration  $i \leq \lceil \log_2 \hat{k} \rceil$  of the outer loop of  $\text{FindHeavySet}$  for which  $\ell \in [2^i, 2^{i+1}]$ . We argue that with high probability, if the algorithm reaches iteration

---

<sup>6</sup>Omitting log-factors in both  $m$  and  $n$

$i$ , it will terminate during this iteration and avoid returning FAIL. By (\*), there are at least  $2^i$  sets  $S \in \mathcal{F}$  with  $|\tilde{S}| \geq \max\left\{\frac{N}{2^{i+3}\log n}, \frac{N}{2k}\right\}$ . By the choice of  $R \geq \frac{3m\log n}{2^i}$ , the probability that at least one of these sets  $S$  are sampled in  $\{\mathbf{S}_1, \dots, \mathbf{S}_R\}$  is

$$1 - (1 - 2^i/m)^R \leq 1 - \exp(-2^i R/m) \geq 1 - 1/n^3.$$

We condition on this event that  $\mathbf{S}_r = S$  for some  $r \in [R]$ . Suppose we have not terminated when the inner loop is at iteration  $r$ , and let  $\mathbf{X} = \{\mathbf{x}_1, \dots, \mathbf{x}_T\} \subset \mathcal{U}^{\text{alive}}$  denote the random sample during this iteration. We observe that  $|\tilde{S}| \geq \alpha N$  for

$$\alpha = \max\left\{\frac{1}{2^{i+3}\log n}, \frac{1}{2k}\right\} = \frac{48\log(mn)}{T}.$$

By the first statement of Claim 5.3 we conclude that, with probability  $1 - \exp(-\frac{\alpha T}{12}) = 1 - 1/(mn)^4$ , we have  $|S \cap \mathbf{X}| = |\tilde{S} \cap \mathbf{X}| \geq 18\log n$ , and we will return  $S$ . In total, then, FindHeavySet avoids returning FAIL with probability at least  $1 - 1/(mn)^4 - 1/n^3 \geq 1 - 2/n^3$ .

We will now show the second bullet point. Suppose that FindHeavySet does not return FAIL, and instead the outer loop of FindHeavySet terminates at some iteration  $i \leq \log_2 \hat{k}$ . Recall that we only return a set  $S \in \mathcal{F}$  if  $|S \cap \mathbf{X}| = |\tilde{S} \cap \mathbf{X}| \geq 18\log n = \frac{\alpha T}{2}$ , which holds for the setting of  $\alpha = \frac{36\log(mn)}{T} \geq \frac{1}{2\hat{k}}$ . If  $|\tilde{S}| < \alpha N/4$ , then by the second statement of Claim 5.3 implies that  $|\tilde{S} \cap \mathbf{X}| < \frac{\alpha T}{2}$  with probability  $1 - \exp(-\alpha T/12) = 1 - 1/(mn)^4$ . Union bounding over the  $R \log_2 \hat{k} \ll mn$  sets sampled throughout the execution of the algorithm, we conclude that, with probability  $1 - 1/(mn)^3$ , the set  $\mathbf{S}$  returned by FindHeavySet satisfies

$$|\tilde{\mathbf{S}}| \geq \frac{\alpha N}{4} \geq \frac{N}{8\hat{k}}$$

Overall, we conclude that the properties in both bullet points hold with probability  $1 - 2/n^3 - 1/(mn)^3 = 1 - O(1/n^3)$ , as desired.  $\square$

Finally, we show how in the execution of FastSetCover we may efficiently maintain the set of uncovered elements  $\mathcal{U}^{\text{alive}}$ .

**Claim 5.5.** *Let  $\mathcal{U}$  be a set of size  $n$ . There is a data structure that can be built in time  $O(n)$  which maintains a subset  $\mathcal{U}^{\text{alive}} \subseteq \mathcal{U}$  under deletions, and supports:*

- *checking whether  $\mathcal{U}^{\text{alive}} = \emptyset$  in  $O(1)$  time,*
- *deletion of any element from  $\mathcal{U}^{\text{alive}}$  in  $O(\log n)$  time, and*
- *uniform random sampling from  $\mathcal{U}^{\text{alive}}$  in  $O(\log n)$  time*

*Proof.* Deferred to Appendix A.  $\square$

We are now ready to prove the main guarantee for FastSetCover stated in Lemma 5.1.

*Proof of Lemma 5.1.* We condition on the event that every call to FindHeavySet satisfies the two properties of Lemma 5.4. The first property implies that in the main loop of FastSetCover, at most  $1 + \log_2 k$  calls to FindHeavySet will return FAIL, and  $\hat{k} \leq 2k$  throughout the execution of the algorithm. The second property implies that whenever FindHeavySet does not return FAIL, it will return a set  $S$  with

$$|S \cap \mathcal{U}^{\text{alive}}| \geq \frac{|\mathcal{U}^{\text{alive}}|}{8\hat{k}} \geq \frac{|\mathcal{U}^{\text{alive}}|}{16k}.$$

Therefore, at every iteration of the main loop, we have

$$|\mathcal{U}^{\text{alive}}| \leq n \cdot \left(1 - \frac{1}{16k}\right)^{|\mathcal{C}|} \leq n \cdot \exp(-|\mathcal{C}|/16k)$$

and thus upon termination,  $|\mathcal{C}| \leq 1 + 16k \ln n = O(k \ln n)$ , proving the desired  $O(\ln n)$ -approximation guarantee.

To bound the runtime, we observe that FastSetCover makes at most  $1 + \log_2 k + O(k \log n) = O(k \log n)$  calls to  $\text{FindHeavySet}(\mathcal{U}^{\text{alive}}, \mathcal{F}, \hat{k}, n)$ , each with parameter  $\hat{k} \leq 2k$ . By the runtime guarantee of Lemma 5.4, the time needed to execute these calls is  $\tilde{O}(mk)$ . The only other nontrivial computation in FastSetCover is the maintenance of the data structure for  $\mathcal{U}^{\text{alive}}$ . By Claim 5.5, this data structure can be built in time  $O(n \log n)$ . Moreover, each of the  $O(k \log n)$  updates  $\mathcal{U}^{\text{alive}} \leftarrow \mathcal{U}^{\text{alive}} \setminus S$  takes time  $O(n \log n)$ , as  $S$  can be computed in  $n$  calls to  $\text{contains}(S, \cdot)$  and each deletion can be performed in  $O(\log n)$  time. Hence, the total time needed to perform all updates to  $\mathcal{U}^{\text{alive}}$  is at most  $O(nk \log^2 n) = \tilde{O}(nk)$ . Therefore, the overall runtime of FastSetCover is at most  $\tilde{O}(mk + nk)$ .

Finally, we verify the desired guarantee on success probability. Each call to  $\text{FindHeavySet}$  succeeds with probability  $1 - 1/n^3$ , and conditioned on their success, the total number of calls is at most  $1 + \log_2 k + n = O(n)$ . By a union bound, we conclude that FastSetCover satisfies the desired runtime and approximation guarantees with probability  $1 - O(1/n^2)$ . Moreover, by early termination of FastSetCover, we can ensure that the above bound on the runtime holds deterministically.  $\square$

**Remark 5.2.** While Lemma 5.1 yields an  $O(\ln n)$ -approximation to the minimum set cover, the algorithm can easily be modified to achieve, for any  $\varepsilon \in (0, 1)$ , a  $(1 + \varepsilon) \ln n$ -approximation in time  $\tilde{O}\left(\frac{mk}{\varepsilon^{O(1)}} + nk\right)$ .

## 5.2 An $\tilde{O}(n^\omega \log \Delta/\varepsilon)$ -time Bicriteria Approximation

The algorithm from Theorem 3 is efficient when  $\text{OPT}_{\text{size}}$  is small, but its runtime can grow up to  $O(n^3)$  in the worst case when  $\text{OPT}_{\text{size}}$  is large. In this subsection, we present a faster algorithm for constructing sparse  $\alpha$ -navigable graphs that always runs in subcubic time. Specifically, leveraging fast matrix multiplication, we design an  $\tilde{O}(n^\omega \log \Delta/\varepsilon)$ -time algorithm that returns an  $(\alpha, 2\alpha(1 + \varepsilon))$ -bicriteria  $O(\ln n)$ -approximation, for any  $\varepsilon > 0$ .

**Algorithm Overview.** Throughout this subsection, we assume  $d$  is a metric over  $[n]$  with aspect ratio  $\Delta$ , and without loss of generality, that all distances lie in  $[1, \Delta]$ . For  $\varepsilon > 0$ , we say that a metric  $d$  is  $\varepsilon$ -discretized if all pairwise distances are integer powers of  $(1 + \varepsilon)$ . Our algorithm is motivated by the following observation. Suppose we are given a graph  $G = ([n], E)$  and want to verify whether  $G$  is  $\alpha$ -navigable with respect to  $d$ . A naive approach would check all  $\Theta(n^2)$   $\alpha$ -navigability constraints separately, which takes  $\Theta(n^3)$  time. However, if the metric  $d$  is  $\varepsilon$ -discretized, it turns out that this check can be performed in  $O(n^\omega \cdot (\log \Delta)/\varepsilon)$  time using fast matrix multiplication.

The key subroutine in our algorithm, VerifyNav (Algorithm 5), does exactly this. For each of the  $O(\log \Delta/\varepsilon)$  distance scales  $(1 + \varepsilon)^i$ , it creates an  $n \times n$  Boolean matrix  $B^{(i)}$  indicating pairwise distances that are strictly less than  $(1 + \varepsilon)^i/\alpha$ . It then computes  $C^{(i)} = A \cdot B^{(i)}$ , where  $A$  is the adjacency matrix of the current graph  $G$ . The matrix  $C^{(i)}$  satisfies the following property: for any pair  $(s, t)$  such that  $d(s, t) = (1 + \varepsilon)^i$ , the entry  $C_{s,t}^{(i)}$  is nonzero if and only if the  $\alpha$ -navigability constraint from  $s$  to  $t$  is covered in  $G$ .

Assume for now that the metric  $d$  is  $\varepsilon$ -discretized; this introduces an additional  $(1 + \varepsilon)$  factor in the bicriteria guarantee. The main algorithm, BuildNav (Algorithm 4), incrementally constructs an  $\alpha$ -navigable graph  $G$  on  $d$  by repeatedly invoking VerifyNav to identify and eliminate uncovered  $\alpha$ -navigability constraints. For

each source node  $s \in [n]$ , the algorithm maintains: a working set  $\mathcal{U}_s$  of uncovered targets, a sampling budget  $\hat{k}_s$ , and a counter  $c_s$ .

In each round, the algorithm *simultaneously* updates the out-neighborhoods of all sources in parallel. For each source node  $s$ , it samples  $\hat{k}_s$  targets uniformly at random from  $\mathcal{U}_s$ , adds edges from  $s$  to each sampled target, and then calls VerifyNav to compute all updated uncovered sets  $\mathcal{U}_s^{\text{new}}$ . If  $\mathcal{U}_s^{\text{new}}$  remains nonempty, the counter  $c_s$  is incremented. Once  $c_s$  reaches  $\Omega(\log n)$ , the sampling budget  $\hat{k}_s$  is doubled and  $c_s$  is reset to zero. The key lemma of this section is Lemma 5.9, which shows that when  $\hat{k}_s$  exceeds the minimum degree of  $s$  in any  $2\alpha$ -navigable graph on  $d$ , the uncovered set  $\mathcal{U}_s$  shrinks by a constant factor with constant probability. Therefore, it takes only  $O(\log n)$  rounds for  $\mathcal{U}_s$  to become empty. Since each round takes  $O(n^\omega \cdot \log \Delta/\varepsilon)$  time and involves adding at most  $\hat{k}_s$  edges per node, we obtain the  $O(\ln n)$ -approximation guarantee in the desired runtime.

---

**Algorithm 4:** BuildNav( $d, n, \alpha, \varepsilon$ )

---

**Input:** a metric  $d$  on  $[n]$ , parameters  $\alpha \geq 1, \varepsilon > 0$   
**Output:** an  $\alpha$ -navigable graph  $G = ([n], E)$

- 1 Round all distances down to  $\tilde{d}(s, t) := (1 + \varepsilon)^{\lfloor \log_{(1+\varepsilon)} d(s, t) \rfloor}$ ;
- 2 Initialize  $E = \emptyset$  and set budgets  $\hat{k}_s = 1$  and counters  $c_s = 0$  for all  $s \in [n]$ ;
- 3 Initialize  $\mathcal{U}_s = \{t \mid t \neq s\}$  for all  $s$ ;
- 4 **repeat**
- 5     **foreach**  $s \in [n]$  with  $\mathcal{U}_s \neq \emptyset$  **do**
- 6         Sample  $\mathbf{u}_1, \dots, \mathbf{u}_{\hat{k}_s} \stackrel{\text{i.i.d.}}{\sim} \text{Unif}(\mathcal{U}_s)$ ;
- 7         Update  $E \leftarrow E \cup \{(s, \mathbf{u}_i) \mid 1 \leq i \leq \hat{k}_s\}$ ;
- 8          $\{\mathcal{U}_s^{\text{new}}\} \leftarrow \text{VerifyNav}(\tilde{d}, G, n, \alpha(1 + \varepsilon))$ ;
- 9         **foreach**  $s \in [n]$  with  $\mathcal{U}_s^{\text{new}} \neq \emptyset$  **do**
- 10             **if**  $c_s \geq 120 \cdot \log_{16/15} n$  **then**
- 11                 Update  $\hat{k}_s \leftarrow 2 \cdot \hat{k}_s$ ;
- 12                 Reset  $c_s = 0$ ;
- 13             **else**
- 14                 Update  $c_s \leftarrow c_s + 1$ ;
- 15              $\{\mathcal{U}_s\} \leftarrow \{\mathcal{U}_s^{\text{new}}\}$ ;
- 16 **until**  $\bigcup_s \mathcal{U}_s = \emptyset$ ;
- 17 **return**  $G$ ;

---



---

**Algorithm 5:** VerifyNav( $\tilde{d}, G, n, \alpha$ )

---

**Input:**  $\varepsilon$ -discretized metric  $\tilde{d}$  on  $[n]$  with aspect ratio  $\Delta$ , parameter  $\alpha \geq 1$ , and graph  $G = ([n], E)$   
**Output:** set of all  $\alpha$ -navigability constraints  $(s, t)$  unsatisfied by  $G$

- 1 Let  $A$  be the adjacency matrix of  $G$ ;
- 2 Initialize  $\mathcal{U}_s = \emptyset$  for all  $s \in [n]$ ;
- 3 **foreach**  $i \in [\log_{1+\varepsilon} \Delta]$  **do**
- 4     Build matrix  $B^{(i)}$  with  $B_{s,t}^{(i)} = \mathbb{1}\{\tilde{d}(s, t) < (1 + \varepsilon)^i / \alpha\}$ ;
- 5     Compute  $C^{(i)} \leftarrow A \cdot B^{(i)}$ ;
- 6 **foreach**  $s \in [n], t \in [n] \setminus \{s\}$  **do**
- 7     Let  $i = \log_{(1+\varepsilon)} \tilde{d}(s, t)$ ;
- 8     **if**  $C_{s,t}^{(i)} = 0$  **then**
- 9         Update  $\mathcal{U}_s \leftarrow \mathcal{U}_s \cup \{(s, t)\}$ ;
- 10 **return**  $\{\mathcal{U}_s \mid s \in [n]\}$ ;

---

We first prove correctness of the verification algorithm VerifyNav.

**Claim 5.6.** *If  $\tilde{d}$  is an  $\varepsilon$ -discretized metric on  $[n]$  with aspect ratio  $\Delta$ , then VerifyNav( $\tilde{d}, G, n, \alpha$ ) returns the set of  $\alpha$ -navigability constraints unsatisfied by  $G$  in time  $O(n^\omega \cdot \log \Delta/\varepsilon)$ .*

*Proof.* We first check the runtime of VerifyNav. The algorithm has two phases. First, it computes  $\log_{1+\varepsilon} \Delta = O(\log \Delta/\varepsilon)$ -many multiplications of matrices in  $\{0, 1\}^{n \times n}$ , each of which takes time  $O(n^\omega)$ . Next, it constructs the lists  $\mathcal{U}_s$  by iterating over all pairs  $(s, t)$ , which takes time  $O(n^2)$ . Therefore, the total runtime of VerifyNav is bounded by  $O(n^\omega \cdot (\log \Delta)/\varepsilon)$ .

To check correctness, we fix a pair  $(s, t)$  at distance  $d(s, t) = (1 + \varepsilon)^i$  and verify that  $C_{s,t}^{(i)} = 1$  if and only if  $G = ([n], E)$  satisfies the  $\alpha$ -navigability from  $s$  to  $t$ . If indeed  $G$  satisfies the constraint, there is some edge  $(s, u) \in E$  for which  $d(u, t) < d(s, t)/\alpha$ , meaning  $d(u, t) < (1 + \varepsilon)^i/\alpha$ . Therefore,

$$C_{s,t}^{(i)} = \sum_{v=1}^n A_{s,v} \cdot B_{v,t}^{(i)} \geq A_{s,u} \cdot B_{u,t}^{(i)} = 1$$

Next, suppose that  $G$  does not satisfy the constraint. Then, for any edge  $(s, v) \in E$  we must have  $d(v, t) \geq (1 + \varepsilon)^i/\alpha$ , and thus

$$C_{s,t}^{(i)} = \sum_{v=1}^n A_{s,v} \cdot B_{v,t}^{(i)} = 0,$$

as desired.  $\square$

Next, we show  $\varepsilon$ -discretizing a metric only affects navigability up to a factor  $(1 + \varepsilon)$  in the navigability parameter  $\alpha$ .

**Claim 5.7.** *Let  $\tilde{d}$  be obtained by rounding down every distance in  $d$  to the closest power of  $1 + \varepsilon$  as in Algorithm 4. If  $G$  is  $\alpha$ -navigable on  $\tilde{d}$ , then it is  $\alpha/(1 + \varepsilon)$ -navigable on  $d$ . Moreover, if  $H$  is  $\alpha$ -navigable on  $d$ , then it is  $\alpha/(1 + \varepsilon)$ -navigable on  $\tilde{d}$ .*

*Proof.* Suppose  $G$  is  $\alpha$ -navigable on  $\tilde{d}$ . Then, for any  $s, t, u \in [n]$  where  $\tilde{d}(u, t) < \tilde{d}(s, t)/\alpha$ , we have

$$d(u, t) < (1 + \varepsilon) \cdot \tilde{d}(u, t) < \frac{1 + \varepsilon}{\alpha} \cdot \tilde{d}(s, t) \leq \frac{1 + \varepsilon}{\alpha} \cdot d(s, t),$$

which proves that  $G$  is  $\alpha/(1 + \varepsilon)$ -navigable on  $d$ .

Next, suppose that  $H$  is  $\alpha$ -navigable on  $d$ . Then, for any  $s, t, u \in [n]$  where  $d(u, t) < d(s, t)/\alpha$ , we have

$$\tilde{d}(u, t) \leq d(u, t) < \frac{d(s, t)}{\alpha} < \frac{1 + \varepsilon}{\alpha} \cdot \tilde{d}(s, t),$$

which proves that  $H$  is  $\alpha/(1 + \varepsilon)$ -navigable on  $\tilde{d}$ .  $\square$

Recall from Definition 4.1 that  $Z_\alpha(s, u) := \{t \mid d(u, t) < d(s, t)/\alpha\}$ . The following claim relates  $Z_\alpha(s, \cdot)$  with  $Z_{2\alpha}(s, \cdot)$ , and is the key ingredient in our bicriteria approximation analysis.

**Claim 5.8.** *For any metric space  $([n], d)$  and  $\alpha \geq 1$ , suppose that  $s, x, y, z \in [n]$  are such that  $z \in Z_{2\alpha}(s, x)$  and  $d(x, y) \leq d(x, z)$ . Then, it follows that  $z \in Z_\alpha(s, y)$ .*

*Proof.* Since  $d(x, y) \leq d(x, z)$  we have

$$d(y, z) \leq d(x, y) + d(x, z) \leq 2 \cdot d(x, z).$$

Then, since  $z \in Z_{2\alpha}(s, x)$ , we know

$$\mathbf{d}(s, z) > 2\alpha \cdot \mathbf{d}(x, z) \geq \alpha \cdot \mathbf{d}(y, z)$$

which implies  $z \in Z_\alpha(s, y)$ .  $\square$

**Lemma 5.9.** *For any iteration of the main loop in BuildNav, let  $s \in [n]$  be a source whose current uncovered set is  $\mathcal{U}_s$  with  $N = |\mathcal{U}_s|$ . Let  $k_s$  be the minimum degree of  $s$  in any  $2\alpha(1 + \varepsilon)$ -navigable graph on  $\tilde{\mathbf{d}}$ . If  $\hat{k}_s \geq k_s$ , then after updating  $\mathcal{U}_s^{\text{new}} = \text{VerifyNav}(\tilde{\mathbf{d}}, \mathbf{G}, n, \alpha(1 + \varepsilon))$ , we have:*

$$\Pr \left[ |\mathcal{U}_s^{\text{new}}| \leq \frac{15}{16} \cdot |\mathcal{U}_s| \right] \geq 1/15.$$

*Proof.* BuildNav samples  $\mathbf{u}_1, \dots, \mathbf{u}_{\hat{k}_s} \stackrel{\text{i.i.d.}}{\sim} \mathcal{U}_s$  and adds all edges  $(s, \mathbf{u}_i)$  to  $\mathbf{G}$ , and updates

$$\mathcal{U}_s^{\text{new}} = \text{VerifyNav}(\tilde{\mathbf{d}}, \mathbf{G}, n, \alpha(1 + \varepsilon)).$$

To analyze the size of  $\mathcal{U}_s^{\text{new}}$ , observe that since there is a  $2\alpha(1 + \varepsilon)$ -navigable graph on  $\tilde{\mathbf{d}}$  where  $s$  has degree  $k_s$ , we can find  $x_1, \dots, x_{k_s} \in [n]$  with  $\mathcal{U}_s = \bigcup_{i=1}^{k_s} Z_{2\alpha(1+\varepsilon)}(s, x_i)$ .

Partition  $\mathcal{U}_s = Y_1 \sqcup \dots \sqcup Y_{k_s}$  so each  $Y_i \subseteq Z_{2\alpha}(s, x_i)$  is disjoint. For every  $i \in [k_s]$ , further split  $Y_i = C_i \sqcup F_i$  where:

$$C_i = \text{the } \left\lceil \frac{|Y_i|}{2} \right\rceil \text{ closest points to } x_i, \quad F_i = Y_i \setminus C_i.$$

By Claim 5.8, whenever  $u \in C_i$ , the entire set  $F_i \cup \{u\}$  is contained in  $Z_{\alpha(1+\varepsilon)}(s, u)$ .

Now, define

$$\mathbf{X}_i = \begin{cases} 1 & \text{if } C_i \cap \{\mathbf{u}_1, \dots, \mathbf{u}_{\hat{k}_s}\} \neq \emptyset \\ 0 & \text{otherwise} \end{cases}$$

and observe that

$$|\mathcal{U}_s| - |\mathcal{U}_s^{\text{new}}| \geq \sum_{i=1}^{k_s} (|F_i| + 1) \cdot \mathbf{X}_i \geq \sum_{i=1}^{k_s} |C_i| \cdot \mathbf{X}_i := \mathbf{A}.$$

Let  $N := |\mathcal{U}_s|$ . We will show that  $\Pr[\mathbf{A} \geq N/16] \geq 1/15$ . First, the expectation of each  $\mathbf{X}_i$  is given by:

$$\mathbf{E}[\mathbf{X}_i] = \left( 1 - (1 - |C_i|/N)^{\hat{k}_s} \right) \geq \left( 1 - \exp(-\hat{k}_s |C_i|/N) \right).$$

By linearity of expectation,

$$\mathbf{E}[\mathbf{A}] = \sum_{i=1}^{k_s} |C_i| \cdot \mathbf{E}[\mathbf{X}_i] \geq \sum_{i=1}^{k_s} |C_i| \cdot \left( 1 - \exp(-\hat{k}_s |C_i|/N) \right) \geq \frac{N}{2} - \sum_{i=1}^{k_s} |C_i| \cdot \exp(-k_s |C_i|/N)$$

where we have used that  $|C_i| \geq |Y_i|/2$  and  $\hat{k}_s \geq k_s$ . To analyze the right-most sum, observe that

$$|C_i| \cdot \exp(-k_s |C_i|/N) \leq \frac{N}{k_s} \cdot \max_{t \geq 0} (te^{-t}) = \frac{N}{ek_s}.$$

This immediately gives

$$\mathbf{E}[\mathbf{A}] \geq \frac{N}{2} - \frac{N}{e} > \frac{N}{8}.$$

By Markov's inequality, we conclude that  $\Pr[\mathbf{A} \geq N/16] \geq 1/15$ , as desired.  $\square$

**Lemma 5.10.** Consider any source  $s \in [n]$  in the execution of BuildNav. Let  $k_s$  be the minimum degree of  $s$  in any  $2\alpha(1 + \varepsilon)$ -navigable graph on  $\tilde{d}$ . Then, with probability at least  $1 - 1/n^2$ , the set  $\mathcal{U}_s$  becomes empty while  $\hat{k}_s \leq 2k_s$ .

*Proof.* Fix a source  $s \in [n]$ , and suppose that  $\mathcal{U}_s$  is still nonempty at the first moment that  $\hat{k}_s \geq k_s$ . At this point,  $\hat{k}_s \leq 2k_s$ , and we will show that with high probability,  $\mathcal{U}_s$  will become empty without updating  $\hat{k}_s$  further.

For  $R = 120 \cdot \log_{16/15} n$ , the execution of BuildNav involves at most  $R$  rounds of adding  $\hat{k}_s$  random edges from  $s$ . For each round  $i \in [R]$ , define an indicator variable  $\mathbf{Y}_i$  that is equal to 1 if  $|\mathcal{U}_s^{\text{new}}| \leq (15/16) \cdot |\mathcal{U}_s|$  at the end of the  $i$ -th round, and is equal to 0 otherwise. By Lemma 5.9, we have  $\Pr[\mathbf{Y}_i = 1] \geq 1/15$ , independently for each round.

We say that round  $i$  is *good* if  $\mathbf{Y}_i = 1$ . Let  $\mathbf{Y} := \sum_{i=1}^R \mathbf{Y}_i$  be the number of good rounds. Then  $\mathbf{E}[\mathbf{Y}] \geq R/15 = 8 \log_{16/15} n$ . By a standard Chernoff bound, we have

$$\Pr\left[\mathbf{Y} \leq \frac{\mathbf{E}[\mathbf{Y}]}{2}\right] \leq \exp\left(-\frac{\mathbf{E}[\mathbf{Y}]}{8}\right) \leq \exp(-2 \log n) \leq \frac{1}{n^2}.$$

Thus, with probability at least  $1 - 1/n^2$ , there are strictly more than  $\log_{16/15} n$  good rounds. Since  $|\mathcal{U}_s| \leq n$  at the beginning, and each good round reduces the size of  $\mathcal{U}_s$  by a factor of  $15/16$ , then after more than  $\log_{16/15} n$  such rounds, we will have  $|\mathcal{U}_s| = 0$ . Therefore, with probability at least  $1 - 1/n^2$ , we will have  $\mathcal{U}_s = \emptyset$  when  $\hat{k}_s \leq 2k_s$ .  $\square$

We are now ready to prove Theorem 4.

*Proof of Theorem 4.* Let  $(P = [n], d)$  be an  $n$ -point metric with aspect ratio  $\Delta$ , and fix  $\alpha \geq 1$  and  $\varepsilon > 0$ . Let  $\tilde{d}$  be the  $\varepsilon$ -discretized version of  $d$  obtained by rounding each distance down to the nearest power of  $1 + \varepsilon$ , as done in BuildNav. We will show that with probability  $1 - o(1)$ , BuildNav outputs an  $(\alpha(1 + \varepsilon), 2\alpha(1 + \varepsilon))$ -bicriteria  $O(\ln n)$ -approximation to the sparsest navigable graph on  $\tilde{d}$ , in time  $\tilde{O}(n^\omega \cdot \log \Delta/\varepsilon)$ . By Claim 5.7, this output is also an  $(\alpha, 2\alpha(1 + \varepsilon)^2)$ -bicriteria  $O(\ln n)$ -approximation to the sparsest navigable graph on  $d$ , and a rescaling of  $\varepsilon$  gives the desired guarantee.

Correctness follows from the construction: by Claim 5.6, the graph  $G$  returned upon termination of BuildNav is guaranteed to be  $\alpha(1 + \varepsilon)$ -navigable on  $\tilde{d}$ . We will now analyze the sparsity of the graph  $G$ .

For each  $s \in [n]$ , let  $k_s$  denote the minimum degree of  $s$  in any  $2\alpha(1 + \varepsilon)$ -navigable graph on  $\tilde{d}$ . By Lemma 5.10 and a union bound over all  $s$ , we have that with probability at least  $1 - 1/n$ , the budget  $\hat{k}_s$  remains at most  $2k_s$  throughout the execution. In the remainder of the proof, we will condition on this event.

As the budget  $\hat{k}_s$  is held fixed for  $O(\log n)$  rounds, and each round adds at most  $\hat{k}_s$  edges from  $s$ , the number of edges added from  $s$  at the budget level  $\hat{k}$  is  $O(\hat{k}_s \cdot \log n)$ . Since  $\hat{k}_s$  is scaled geometrically up to at most  $2k_s$ , we conclude that  $\deg_G s = O(k_s \log n)$ . Summing over all sources  $s \in [n]$ , the total number of edges in the final graph is at most

$$O(\log n) \cdot \sum_{s=1}^n k_s = O(\log n) \cdot \text{OPT}_{2\alpha(1+\varepsilon)},$$

where  $\text{OPT}_{2\alpha(1+\varepsilon)}$  denotes the edge count of the sparsest  $2\alpha(1 + \varepsilon)$ -navigable graph on  $\tilde{d}$ .

Similarly, the maximum degree is at most  $O(\log n) \cdot \max_s k_s$ , which gives the desired bicriteria guarantee under both the max-out-degree and average-degree objectives.

It remains to bound the runtime, which is given by the number of calls to VerifyNav. Since each source  $s \in [n]$  takes  $O(\log k_s \cdot \log n) = O(\log^2 n)$  rounds of adding edges before  $\mathcal{U}_s$  becomes empty, the total number

of calls to VerifyNav is at most  $O(\log^2 n)$ . By the runtime guarantee of Claim 5.6, we conclude that BuildNav runs in time

$$O(n^\omega \cdot \log^2 n \cdot \log \Delta/\varepsilon) = \tilde{O}(n^\omega \cdot \log \Delta/\varepsilon).$$

The above bound on the runtime occurs with probability  $1 - 1/n$ , but it can be enforced deterministically by early termination of BuildNav.  $\square$

## 6 An $\Omega(n^2)$ Query Lower Bound

Theorem 3 presents an algorithm for producing an  $O(\ln n)$ -approximation to the sparsest  $\alpha$ -navigable graph, running in time  $O(n \cdot \text{OPT}_{\text{size}})$ . Since every  $\alpha$ -navigable graph must be connected and thus contain  $\Omega(n)$  edges, this bound on the runtime can be no better than  $O(n^2)$ . In this section, we show that such a quadratic overhead is unavoidable: any algorithm achieving even an  $o(n)$ -approximation to the sparsest  $\alpha$ -navigable graph must make  $\Omega(n^2)$  distance queries.

**Theorem 5.** *Let  $\mathcal{A}$  be any randomized algorithm with query access to a metric  $\mathbf{d}$  on  $n$  points that (with constant probability) outputs an  $o(n)$ -approximation to the sparsest navigable graph on  $\mathbf{d}$ , under either the max-out-degree or average-degree objective. Then,  $\mathcal{A}$  must make  $\Omega(n^2)$  queries to  $\mathbf{d}(\cdot, \cdot)$ .*

To prove Theorem 5, we construct a distribution over a family of path-like metrics, with a random hidden “shortcut” that must appear as an edge in order to satisfy navigability. By Yao’s minimax principle, it will suffice to show the stated lower bound for deterministic algorithms  $\mathcal{A}$ .

**Definition 6.1** (Perturbed Path Metric). We specify a distribution  $\mathcal{D}$  over metrics  $\mathbf{d}$  on  $[n]$  as follows.

- For all distinct  $i, j \in [n]$ , set

$$\mathbf{d}(i, j) = \mathbf{d}_{\text{path}}(i, j) := 1 + \frac{|i - j|}{n - 1}$$

- Sample  $\{i^*, j^*\} \sim \binom{[n]}{2}$  uniformly at random, and overwrite  $\mathbf{d}(i^*, j^*) = \mathbf{d}(j^*, i^*) = 1$

Since all nonzero distances are in  $[1, 2]$ , the distance function  $\mathbf{d}$  automatically satisfies the triangle inequality and is a metric.

**Lemma 6.1.** *There is a navigable graph on  $\mathbf{d}$  with maximum degree  $\leq 3$ .*

*Proof.* Let  $G = P_n \cup \{i^*, j^*\}$ , where  $P_n$  denotes the (undirected) path graph on  $[n]$ . Observe that  $G$  is navigable on  $\mathbf{d}_{\text{path}}$ , since for any distinct  $s, t$ , we have that  $|u - t| < |s - t|$  for some  $u = s \pm 1$ .

Now consider  $\mathbf{d}$ , the perturbed metric where a single pair  $\{i^*, j^*\}$  has its distance reduced to 1. The only navigability constraints  $(s, t)$  that may differ between  $\mathbf{d}$  and  $\mathbf{d}_{\text{path}}$  are those where  $t \in \{i^*, j^*\}$ . We perform a brief casework on these constraints.

- If  $t = j^*$  and  $s = i^*$ , then there is a direct edge  $(i^*, j^*)$  in  $G$ .
- Suppose  $t = j^*$  and  $s \neq i^*$ . Then, by navigability of  $G$  on  $\mathbf{d}_{\text{path}}$ , there is an edge  $(s, u)$  in  $G$  such that

$$\mathbf{d}(u, t) \leq \mathbf{d}_{\text{path}}(u, t) < \mathbf{d}_{\text{path}}(s, t) = \mathbf{d}(s, t).$$

- The case that  $t = i^*$  can be handled identically to the above cases.

Hence,  $G$  is navigable on  $\mathbf{d}$ , and it has maximum degree  $\leq 3$ .  $\square$

**Lemma 6.2.** *Let  $\mathcal{A}$  be any deterministic algorithm that makes  $o(n^2)$  queries to  $\mathbf{d}$  and outputs a graph  $G$  with  $o(n^2)$  edges. With probability  $1 - o(1)$  over  $\mathbf{d} \sim \mathcal{D}$ , the graph  $G$  is not navigable on  $\mathbf{d}$ .*

*Proof.* Let  $S \subset \binom{[n]}{2}$  denote the set of distance queries made by  $\mathcal{A}$ . Define the event

$$\mathcal{E} := \{\{i^*, j^*\} \notin S\}.$$

Since  $\{i^*, j^*\}$  is drawn uniformly at random from  $\binom{[n]}{2}$  and  $|S| = o(n^2)$ , we have  $\Pr[\mathcal{E}] = 1 - o(1)$ .

Let  $G = ([n], E)$  be the output graph of  $\mathcal{A}$ , which is entirely determined by the responses to the queries in  $S$ . Conditioning on  $\mathcal{E}$ , the pair  $\{i^*, j^*\}$  is uniform over  $\binom{[n]}{2} \setminus S$ . Since  $|E| = o(n^2)$ , the algorithm  $\mathcal{A}$  includes the pair  $(i^*, j^*)$  in  $E$  only with probability  $o(1)$ . Therefore,

$$\Pr_{\mathbf{d}}[(i^*, j^*) \in E] \leq \Pr_{\mathbf{d}}[\mathcal{E}] + \Pr_{\mathbf{d}}[(i^*, j^*) \in E \mid \mathcal{E}] = o(1).$$

Since  $\mathbf{d}(i^*, j^*) = 1$  is the minimum distance in the metric, any navigable graph on  $\mathbf{d}$  must contain the edge  $(i^*, j^*)$ . Therefore,

$$\Pr_{\mathbf{d}}[G \text{ is navigable on } \mathbf{d}] = o(1),$$

as claimed.  $\square$

*Proof of Theorem 5.* By Yao's minimax principle, it suffices to prove the lower bound against deterministic algorithms under the random input metric  $\mathbf{d} \sim \mathcal{D}$  from Definition 6.1. Let  $\mathcal{A}$  be any deterministic algorithm that makes  $o(n^2)$  queries to the metric  $\mathbf{d}$  and outputs a graph  $G = ([n], E)$  that, with constant probability, is an  $o(n)$ -approximation to the sparsest navigable graph on  $\mathbf{d}$ , under either the max-out-degree or average-degree objective.

By Lemma 6.1, there exists a navigable graph on  $\mathbf{d}$  with maximum degree at most 3, and hence  $O(n)$  total edges. Therefore, any  $o(n)$ -approximation to the sparsest navigable graph on  $\mathbf{d}$  (under either objective) must contain  $o(n^2)$  edges.

However, Lemma 6.2 shows that no deterministic algorithm making  $o(n^2)$  queries can, with constant probability, output a navigable graph on  $\mathbf{d}$  with  $o(n^2)$  edges. Therefore, any such algorithm  $\mathcal{A}$  must make  $\Omega(n^2)$  queries.  $\square$

## References

- [AIK<sup>+</sup>25] Piyush Anand, Piotr Indyk, Ravishankar Krishnaswamy, Sepideh Mahabadi, Vikas C. Raykar, Kirankumar Shiragur, and Hsueh-Yih Xu. Graph-based algorithms for diverse similarity search. In *Forty-second International Conference on Machine Learning*, 2025.
- [AJDG<sup>+</sup>25] Yousef Al-Jazzazi, Haya Diwan, Jinrui Gou, Cameron Musco, Christopher Musco, and Torsten Suel. Distance adaptive beam search for provably accurate graph-based nearest neighbor search. *arXiv preprint arXiv:2505.15636*, 2025.
- [BKL06] Alina Beygelzimer, Sham Kakade, and John Langford. Cover trees for nearest neighbor. In *Proceedings of the 23rd International Conference on Machine Learning*, ICML '06, page 97–104, New York, NY, USA, 2006. Association for Computing Machinery.

- [CDFC<sup>+</sup>25] Alex Conway, Laxman Dhulipala, Martin Farach-Colton, Rob Johnson, Ben Landrum, Christopher Musco, Yarin Shechter, Torsten Suel, and Richard Wen. Efficiently constructing sparse navigable graphs, 2025.
- [Cla06] Kenneth L. Clarkson. Nearest-neighbor searching and metric space dimensions. In Gregory Shakhnarovich, Trevor Darrell, and Piotr Indyk, editors, *Nearest-Neighbor Methods in Learning and Vision: Theory and Practice*, pages 15–59. MIT Press, 2006.
- [DGM<sup>+</sup>24] Haya Diwan, Jinrui Gou, Cameron N Musco, Christopher Musco, and Torsten Suel. Navigable graphs for high-dimensional nearest neighbor search: Constructions and limits. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024.
- [DK20] Sanjoy Dasgupta and Samory Kpotufe. *Beyond the Worst-Case Analysis of Algorithms*, chapter Nearest Neighbor Classification and Search. Cambridge University Press, 2020.
- [DS14] Irit Dinur and David Steurer. Analytical approach to parallel repetition. In *Proceedings of the Forty-Sixth Annual ACM Symposium on Theory of Computing*, 2014.
- [FXWC19] Cong Fu, Chao Xiang, Changxu Wang, and Deng Cai. Fast approximate nearest neighbor search with the navigating spreading-out graph. In *Proceedings of the VLDB Endowment*, volume 12, pages 461–474, 2019.
- [GKL03] A. Gupta, R. Krauthgamer, and J.R. Lee. Bounded geometries, fractals, and low-distortion embeddings. In *44th Annual IEEE Symposium on Foundations of Computer Science, 2003. Proceedings.*, pages 534–543, 2003.
- [GKSW25] Siddharth Gollapudi, Ravishankar Krishnaswamy, Kirankumar Shiragur, and Harsh Wardhan. Sort before you prune: Improved worst-case guarantees of the diskANN family of graphs. In *Forty-second International Conference on Machine Learning*, 2025.
- [IMR<sup>+</sup>18] Piotr Indyk, Sepideh Mahabadi, Ronitt Rubinfeld, Ali Vakilian, and Anak Yodpinyanee. *Set Cover in Sub-linear Time*. 2018.
- [IX23] Piotr Indyk and Haike Xu. Worst-case performance of popular approximate nearest neighbor search implementations: Guarantees and limitations. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023.
- [Joh73] David S. Johnson. Approximation algorithms for combinatorial problems. In *Proceedings of the Fifth Annual ACM Symposium on Theory of Computing*, 1973.
- [JSDS<sup>+</sup>19] Suhas Jayaram Subramanya, Fnu Devvrit, Harsha Vardhan Simhadri, Ravishankar Krishnawamy, and Rohan Kadekodi. Diskann: Fast accurate billion-point nearest neighbor search on a single node. In *Advances in Neural Information Processing Systems*, volume 32, 2019.
- [KL04] Robert Krauthgamer and James R. Lee. Navigating nets: simple algorithms for proximity search. In *Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA ’04, page 798–807, USA, 2004. Society for Industrial and Applied Mathematics.
- [KY14] Christos Koufogiannakis and Neal E. Young. A nearly linear-time ptas for explicit fractional packing and covering linear programs. *Algorithmica*, 70(4):648–674, 2014.
- [Laa18] Thijs Laarhoven. Graph-based time-space trade-offs for approximate near neighbors. In *34th International Symposium on Computational Geometry (SoCG 2018)*. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2018.
- [MY18] Yu A Malkov and Dmitry A Yashunin. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 42(4):824–836, 2018.

- [PS20] Liudmila Prokhorenkova and Aleksandr Shekhovtsov. Graph-based nearest neighbor search: From practice to theory. In *International Conference on Machine Learning*, pages 7803–7813. PMLR, 2020.
- [SAI<sup>+</sup>24] Harsha Vardhan Simhadri, Martin Aumüller, Amir Ingber, Matthijs Douze, George Williams, Magdalen Dobson Manohar, Dmitry Baranchuk, Edo Liberty, Frank Liu, Ben Landrum, Mazin Karjikar, Laxman Dhulipala, Meng Chen, Yue Chen, Rui Ma, Kai Zhang, Yuzheng Cai, Jiayang Shi, Yizhuo Chen, Weiguo Zheng, Zihao Wan, Jie Yin, and Ben Huang. Results of the big ann: Neurips’23 competition, 2024.
- [SWA<sup>+</sup>21] Harsha Vardhan Simhadri, George Williams, Martin Aumüller, Matthijs Douze, Artem Babenko, Dmitry Baranchuk, Qi Chen, Lucas Hosseini, Ravishankar Krishnaswamy, Gopal Srinivasa, Shuaa Jayaram Subramanya, and Jingdong Wang. Results of the NeurIPS’21 challenge on billion-scale approximate nearest neighbor search. In *Proceedings of Advances in Neural Information Processing Systems 34 (NeurIPS ’2021)*, pages 177–189, 2021.
- [XSI24] Haike Xu, Sandeep Silwal, and Piotr Indyk. A bi-metric framework for fast similarity search. *arXiv preprint arXiv:2406.02891*, 2024.

## A Missing Proofs from Section 5

**Claim 5.2.** Let  $\alpha_1 \geq \alpha_2 \geq \dots \geq \alpha_L \geq 0$  be non-negative numbers with  $\sum_{j=1}^L \alpha_j = A$ . Then, there exists an index  $\ell \in [L]$  such that  $\ell \cdot \alpha_\ell \geq A/H_L$ , where  $H_L$  denotes the  $L$ -th harmonic number.

*Proof.* Let  $\ell \in [L]$  maximize the quantity  $\ell \cdot \alpha_\ell$ . For each  $j \in [L]$ , we have

$$\alpha_j \leq \frac{\ell \cdot \alpha_\ell}{j}.$$

Summing this inequality over all  $j \in [L]$  yields

$$A = \sum_{j=1}^L \alpha_j \leq \sum_{j=1}^L \frac{\ell \cdot \alpha_\ell}{j} = (\ell \cdot \alpha_\ell) \cdot H_L,$$

as desired.  $\square$

**Claim 5.3.** Let  $S \subset [N]$  be fixed, and let  $\mathbf{X} := \{x_1, \dots, x_T\}$  where  $x_i \stackrel{\text{i.i.d.}}{\sim} \text{Unif}([N])$ . Then for all  $\alpha \in [0, 1]$ , the following bounds hold with probability at least  $1 - \exp(-\alpha T/12)$ :

- (i) If  $|S| \geq \alpha N$ , then  $|S \cap \mathbf{X}| \geq \frac{\alpha T}{2}$ .
- (ii) If  $|S| \leq \frac{\alpha N}{4}$ , then  $|S \cap \mathbf{X}| < \frac{\alpha T}{2}$ .

*Proof.* Suppose that  $|S| \geq \alpha N$ . Then,  $\mathbf{E}[|S \cap \mathbf{X}|] \geq \alpha T$ , and by a lower-tail multiplicative Chernoff bound,

$$\Pr[|S \cap \mathbf{X}| < \alpha T/2] \leq \exp\left(-\frac{\alpha T}{8}\right) \leq \exp\left(-\frac{\alpha T}{12}\right).$$

Suppose that  $|S| \leq \alpha N/4$ . Since  $\Pr[|S \cap \mathbf{X}| \geq \alpha T/2]$  increases monotonically with  $|S|$ , it suffices to consider the setting of  $|S| = \alpha N/4$ , for which  $\mathbf{E}[|S \cap \mathbf{X}|] = \alpha T/4$ . By an upper-tail multiplicative Chernoff bound,

$$\Pr[|S \cap \mathbf{X}| \geq \alpha T/2] \leq \exp\left(-\frac{\alpha T}{12}\right),$$

completing the proof.  $\square$

**Claim 5.5.** Let  $\mathcal{U}$  be a set of size  $n$ . There is a data structure that can be built in time  $O(n)$  which maintains a subset  $\mathcal{U}^{\text{alive}} \subseteq \mathcal{U}$  under deletions, and supports:

- checking whether  $\mathcal{U}^{\text{alive}} = \emptyset$  in  $O(1)$  time,
- deletion of any element from  $\mathcal{U}^{\text{alive}}$  in  $O(\log n)$  time, and
- uniform random sampling from  $\mathcal{U}^{\text{alive}}$  in  $O(\log n)$  time

*Proof.* We implement  $\mathcal{U}^{\text{alive}}$  using a complete binary search tree on  $2^{\lceil \log_2 n \rceil}$  leaves, where each element of  $\mathcal{U}$  corresponding to one leaf. To initialize  $\mathcal{U}^{\text{alive}} = \mathcal{U}$ , we store at each internal node  $v$  a counter

$$c_v = |\mathcal{U} \cap (\text{leaves in the subtree of } v)|,$$

which takes time  $O(n)$ .

- **Checking emptiness.**  $\mathcal{U}^{\text{alive}} = \emptyset$  if and only if the root  $r$  has  $c_r = 0$ , which can be tested in  $O(1)$  time.
- **Deletion.** To delete an element  $x$  from  $\mathcal{U}^{\text{alive}}$ , we first confirm that  $x \in \mathcal{U}^{\text{alive}}$  by searching for its leaf in the tree and checking that  $c_x = 1$ . If so, then for each vertex  $v$  in the unique path from the root to that leaf, we decrement  $c_v$  by 1. Since the tree has height  $O(\log n)$ , this takes  $O(\log n)$  time.
- **Random sampling.** To sample a uniformly random element of  $\mathcal{U}^{\text{alive}}$ , we traverse a random root-to-leaf path as follows. Suppose we are at node  $v$  with children  $v_{\text{left}}, v_{\text{right}}$ . We choose to go to

$$\begin{cases} v_{\text{left}} & \text{with probability } c_{v_{\text{left}}}/c_v \\ v_{\text{right}} & \text{with probability } c_{v_{\text{right}}}/c_v \end{cases}$$

Since  $c_v = c_{v_{\text{left}}} + c_{v_{\text{right}}}$ , this selects a leaf in proportion to the number of elements in  $\mathcal{U}^{\text{alive}}$  beneath each child. Upon reaching a leaf, we return its corresponding element, which is a uniformly random sample from  $\mathcal{U}^{\text{alive}}$ . Each step takes  $O(1)$  time, and there are  $O(\log n)$  steps, so sampling runs in  $O(\log n)$  time.

This completes the proof. □