Proceedings of Machine Learning Research vol 291:1–54, 2025 38th Annual Conference on Learning Theory

Metric Clustering and Graph Optimization Problems using Weak Comparison Oracles

Rahul Raychaudhury RAHUL.RAYCHAUDHURY@DUKE.EDU
Duke University
Wen-Zhi Li WENZHI@CS.CORNELL.EDU
Cornell University
Syamantak Das SYAMANTAK@IIITD.AC.IN
IIIT-Delhi
Sainyam Galhotra SG@CS.CORNELL.EDU
Cornell University
Stavros Sintos STAVROS@UIC.EDU
University of Illinois Chicago

Editors: Nika Haghtalab and Ankur Moitra

Abstract

Traditional clustering methods assume that precise pairwise distances for the input data are readily available. However, this assumption is often impractical in real-world scenarios where data points cannot be measured accurately. For instance, machine learning-based techniques for estimating distances may fail when the dataset consists of images, videos, or natural language. This paper studies clustering and graph problems in settings where direct access to pairwise distances between all pairs is expensive. We adopt oracle-based methods as defined by Galhotra et al. (2024), focusing on two types of oracles: the quadruplet oracle, a weak and inexpensive comparator that answers binary queries of the form "Is A closer to B or C closer to D?" and the distance oracle, a stronger but costlier oracle that returns exact pairwise distances. The quadruplet oracle can be implemented via crowdsourcing, trained classifiers, or other predictive models. As these sources are often unreliable, the oracle's responses may be noisy; we consider both probabilistic and adversarial noise models.

Consider a finite metric space $\Sigma = (\mathcal{V}, d)$ of size $|\mathcal{V}| = n$ that supports the quadruplet and the distance oracle. When the input dataset has low intrinsic (doubling) dimension, for each of the k-center, k-median, and k-means clustering problem on $\mathcal{V}$, we design constant approximation algorithms that perform $\widetilde{O}(n + k^2)$ calls to the quadruplet oracle and $\widetilde{O}(1)$ calls to the distance oracle in both noise models. For general metric spaces, our algorithms achieve constant approximation while making $\widetilde{O}(nk)$ calls to the quadruplet oracle and $\widetilde{O}(1)$ calls to the distance oracle. In all cases, we improve the quadruplet oracle query complexity by a factor of k and the distance oracle call complexity by a factor of k^2 compared to Galhotra et al. (2024).

Furthermore, in low dimensional settings, if the spread of the input data is polynomially bounded, we construct a data structure performing $\widetilde{O}(n)$ queries to the quadruplet oracle and $\widetilde{O}(1)$ queries to the distance oracle, such that given any query pair of vertices $(u, v) \in \mathcal{V} \times \mathcal{V}$, it approximates the distance $d(u, v)$ without using any oracle queries. Once the data structure is constructed, we can emulate standard algorithms for various graph problems on Σ without additional oracle queries.

In summary, our results show that access to a noisy pairwise ranker for distances is to sufficient to efficiently solve a large class of problems while almost entirely bypassing exact distance computations.

Keywords: clustering, k-center, k-median, k-means, comparison oracles, learned rankers, learning-augmented algorithms

1. Introduction

Unsupervised learning techniques are widely used in applications such as data summarization, organization, and document categorization. Classical approaches to unsupervised learning, such as k-center, k-median, and k-means, have been extensively studied for decades and serve as core methods for structuring and understanding datasets across various domains. However, these methods generally assume the availability of precise pairwise distances, a requirement that is frequently challenging in real-world scenarios. For complex data types like images, videos, or natural language, defining high-quality distances that align with semantic meaning often requires sophisticated embedding models, which are generally computationally intense and financially expensive.

Even when a distance metric is well-defined, computing distances between objects can still have high computational complexity. Metrics such as Earth Mover's Distance Rubner et al. (2000); Peyré and Cuturi (2019) and Hausdorff Distance Huttenlocher et al. (1993); Dubuisson and Jain (1994) are valuable in practice yet evaluating distances under these metrics often require substantial computational resources.

Motivated in part by the difficulty of computing exact distances, there has been extensive work on methods that avoid direct distance computation. A common strategy involves the use of *oracles*, which serve as abstractions of ML or human feedback. A well-known example is an oracle that compares relative distances (Is A closer to B, or is C closer to D?), which is commonly known as the *quadruplet oracle*. There is a rich body of work employing such oracles for problems including k-center clustering Addanki et al. (2021), top-down hierarchical clustering Emamjomeh-Zadeh and Kempe (2018); Chatziafratis et al. (2018); Ghoshdastidar et al. (2019), correlation clustering Ukkonen (2017), learning fairness metrics Ilvento (2020), and classification Tamuz et al. (2011); Hopkins et al. (2020). In practice, implementing such oracles is often easier than computing precise distance measurements. This is partly because these oracles are inherently "local" — they do not require reasoning over the entire dataset. This also makes it easier for human annotators to provide labels and for models to learn—mirroring the simplicity of classification over regression.

In this work, we study clustering and other metric graph optimization problems using the quadruplet oracle that, given two pairs of records, provides which pair is closer, effectively functioning as a pairwise ranker for distances. We think of the quadruplet oracle as *weak*; not only does it provide limited information, it is also susceptible to noise. The noise captures the fact that both ML models and human annotators are prone to errors. However, for certain problems such as k-medians/means, it is known that even a perfect quadruplet oracle is insufficient to obtain a constant-factor approximation Galhotra et al. (2024).

Therefore, we assume access not only to the weak quadruplet oracle but also to a *strong distance oracle* capable of providing exact distance measurements. This model was introduced by Galhotra et al. (2024), and we refer to it hereafter as the *rank-and-measure* (RM) model. The RM model applies to any setting where comparing distances is significantly cheaper than computing exact ones. A concrete instantiation of the RM model could involve a pairwise ranker trained on crowd-sourced labels serving as the quadruplet oracle, while the distance oracle corresponds to a commercial embedding service that yields accurate but costly distance estimates. Learned rankers may indeed serve effectively as the quadruplet oracle, as they have demonstrated strong performance across a range of domains, including document retrieval Burges et al. (2005) and image similarity Wang et al. (2014). In fact, recent work suggests that even large language models can act as effective quadruplet oracles without any task-specific training Qin et al. (2024).

Recently, there has been considerable interest in clustering and related tasks within similar *weak-strong* models Bateni et al. (2024); Galhotra et al. (2024); Silwal et al. (2023); Xu et al. (2024). The notions of "weak" and "strong" are contextual, but at a high level, they reflect differences in information content and reliability, with the former being easier to obtain, and the latter being more expensive.

Our goal in this paper is to study the query complexity of clustering and other metric graph optimization problems in the RM model. In addition to the general setting, where no assumptions are made about the similarity measure between objects in the dataset, we also consider the case where the dataset exhibits low intrinsic dimensionality. In practice, many datasets such as word embeddings Yin and Shen (2018), social networks Nakis et al. (2025), and biological sequences Ng and Zhang (2002); Tenenbaum et al. (2000) often have low intrinsic dimensionality with respect to meaningful similarity metrics, even though they are stored in high-dimensional spaces. While this suggests that the underlying structure may be captured more efficiently, many of the earlier challenges associated with computing distances continue to hold.

In this paper, we present algorithms for clustering with near-optimal query complexity in both the general and low intrinsic dimensionality settings. Furthermore, when the intrinsic dimensionality of the dataset is low, we provide a general tool that allows standard algorithms to be used directly for a broad class of metric graph problems (such as metric Minimum Spanning Tree). Our results show that, with access to a noisy pairwise ranker for distances, a large class of problems can be solved efficiently while almost entirely bypassing exact distance computations.

1.1. Problem Statement

Let $\Sigma = (\mathcal{V}, d)$ be a finite metric space of size $|\mathcal{V}| = n$, where $d : \mathcal{V} \times \mathcal{V} \to \mathbb{R}_{\geq 0}$ denotes the distance function. The set $\mathcal{V}$ represents the objects of interest—such as the elements of a dataset we aim to cluster—while d captures a meaningful notion of dissimilarity between them. Any such metric space can be naturally viewed as a weighted complete graph $G_\Sigma = (\mathcal{V}, \mathcal{E}, d)$, where $\mathcal{E}$ contains all edges between pairs of objects and each edge is weighted according to their distance.

Rank-and-Measure (RM) Model. In this paper, we study computational tasks where access to Σ is governed by the RM model. In this setting, direct access to the distance function d is unavailable. Instead, all distance-related information must be obtained through two oracles. The first is a weak but inexpensive *quadruplet oracle*, which, given a pair of distinct edges $(\mathtt{e}_1, \mathtt{e}_2) \in \mathcal{E} \times \mathcal{E}$, reveals only whether $d(\mathtt{e}_1) \leq d(\mathtt{e}_2)$. The second is a strong but expensive *distance oracle*, which returns the exact value of $d(\mathtt{e})$ for any edge $\mathtt{e} \in \mathcal{E}$. While the quadruplet oracle is cheap, its responses are subject to noise. We consider two distinct noise models.

(a) **Quadruplet Oracle with Probabilistic Noise.** For a constant $\varphi \in \left[0, \frac{1}{2}\right)$, a quadruplet oracle with probabilistic noise φ is a function $\tilde{\mathcal{Q}} : \mathcal{E} \times \mathcal{E} \to \{\textsc{Yes}, \textsc{No}\}$ that, given two edges $\mathtt{e}_1, \mathtt{e}_2 \in \mathcal{E}$, where $\mathtt{e}_1 = \{v_1, v_2\}$, and $\mathtt{e}_2 = \{v_3, v_4\}$, outputs

$$\tilde{\mathcal{Q}}(\mathtt{e}_1, \mathtt{e}_2) = \begin{cases} \textsc{Yes}, \text{ with probability at least } 1 - \varphi, & \text{if } d(v_1, v_2) \leq d(v_3, v_4), \\ \textsc{No}, \text{ with probability at least } 1 - \varphi, & \text{if } d(v_1, v_2) > d(v_3, v_4). \end{cases}$$

In other words, the oracle fails to identify the closer pair with probability at most φ. Furthermore, the randomness is independent across distinct queries and is fixed once per edge pair;

thus, repeated calls to $\tilde{\mathcal{Q}}(\mathtt{e}_1, \mathtt{e}_2)$ always return the same result, and flipping the order of the edges always flips the answer. This property is referred to as *persistence*.

(b) **Quadruplet Oracle with Adversarial Noise.** Let $\mu \in \mathbb{R}_{\geq 0}$ be a constant. A quadruplet oracle with adversarial noise μ is a function $\mathcal{Q} : \mathcal{E} \times \mathcal{E} \to \{\textsc{Yes}, \textsc{No}\}$ that, given two edges $\mathtt{e}_1, \mathtt{e}_2 \in \mathcal{E}$, where $\mathtt{e}_1 = \{v_1, v_2\}$, and $\mathtt{e}_2 = \{v_3, v_4\}$, outputs,

$$\mathcal{Q}(\mathtt{e}_1, \mathtt{e}_2) = \begin{cases} \textsc{Yes}, & \text{if } d(v_1, v_2) < \frac{1}{1+\mu} d(v_3, v_4), \\ \textsc{No}, & \text{if } d(v_1, v_2) > (1+\mu) d(v_3, v_4), \\ \text{Adversarially chosen}, & \text{if } \frac{1}{1+\mu} \leq \frac{d(v_1,v_2)}{d(v_3,v_4)} \leq 1+\mu. \end{cases}$$

The oracle accurately identifies the closer pair when the relative distance gap is large, but may respond adversarially otherwise. When $\mu = 0$, responses are exact. We assume that the adversary is non-adaptive, i.e., all responses are fixed a priori and do not depend on the algorithm's queries. As in the probabilistic model, the oracle satisfies persistence: each query yields a fixed response, and flipping the order of the inputs always flips the answer.

Finally, we note that a specific instantiation of the RM model always assumes a single noise model for the quadruplet oracle—either probabilistic or adversarial, but never both simultaneously.

In this paper, we study the following problems under the RM model.

I. Clustering. We begin with the classical task of k-clustering. The goal is to select k representative centers, from a dataset so as to minimize a suitable clustering cost. We formalize this as follows. For two subsets of vertices $\mathcal{U}, \mathcal{W} \subseteq \mathcal{V}$, let $F_{\mathcal{U}}(\mathcal{W})$ denote a clustering cost function measuring the cost of using $\mathcal{U}$ as centers for the points in $\mathcal{W}$. Let $\mathsf{OPT}^F(\mathcal{W}) = \min_{\mathcal{U} \subseteq \mathcal{V}, |\mathcal{U}|=k} F_{\mathcal{U}}(\mathcal{W})$ denote the cost of the optimal k-clustering of $\mathcal{W}$. We focus on three standard objectives: the k-center cost, $\textsc{Cost}^{\infty}_{\mathcal{U}}(\mathcal{W}) = \max_{w \in \mathcal{W}} d(w, \mathcal{U})$; the k-median cost, $\textsc{Cost}^{1}_{\mathcal{U}}(\mathcal{W}) = \sum_{w \in \mathcal{W}} d(w, \mathcal{U})$; and the k-means cost, $\textsc{Cost}^{2}_{\mathcal{U}}(\mathcal{W}) = \sum_{w \in \mathcal{W}} d^2(w, \mathcal{U})$.

Problem OracleCluster. Given a finite metric space $\Sigma = (\mathcal{V}, d)$ accessible under the RM model, a positive integer k, and a clustering cost function $F \in \{\textsc{Cost}^{\infty}_{\mathcal{V}}, \textsc{Cost}^{1}_{\mathcal{V}}, \textsc{Cost}^{2}_{\mathcal{V}}\}$, the goal is to find a set $\mathcal{A} \subseteq \mathcal{V}$ of size $|\mathcal{A}| = k$ such that $F_{\mathcal{A}}(\mathcal{V}) \leq O(1) \cdot \mathsf{OPT}^F(\mathcal{V})$.

In the standard setting, constant-factor approximations to k-clustering require $\Omega(nk)$ distance queries for general metrics Bateni et al. (2023) and $\Omega(n)$ queries for low-dimensional metrics. Naturally, in the RM model one might hope to reduce this cost by leveraging the cheaper quadruplet oracle. Ideally, one would like to solve the OracleCluster problem using only quadruplet queries. However, it is known that approximating k-median and k-means is not possible using only the quadruplet oracle Galhotra et al. (2024). This limitation raises a natural question: while the quadruplet oracle cannot fully replace distance queries, can it *almost* eliminate the need for them? Specifically, we ask:

Can OracleCluster be solved using $\widetilde{O}(nk)$ queries to the quadruplet oracle and $\widetilde{O}(1)$ queries to the distance oracle? [1] If Σ has low intrinsic dimension, can the number of quadruplet queries be further reduced to $\widetilde{O}(n)$?

1. The notation $\tilde{O}(\cdot)$ is used to hide $\log^{O(1)}(n)$ factors.

	Clustering bounded doubling		Clustering bounded doubling and bounded spread		Clustering general	
	Quadr	Dist	Quadr	Dist	Quadr	Dist
Addanki et al. (2021) (k-center)	$\tilde{O}(nk^2)$	0	$\tilde{O}(nk^2)$	0	$\tilde{O}(nk^2)$	0
Galhotra et al. (2024)	$\tilde{O}(nk)$	$\tilde{O}(k^2)$	$\tilde{O}(nk)$	$\tilde{O}(k^2)$	$\tilde{O}(nk)$	$\tilde{O}(k^2)$
NEW	$\tilde{O}(n+k^2)$	$\tilde{O}(1)$	$\tilde{O}(n)$	$\tilde{O}(1)$	$\tilde{O}(nk)$	$\tilde{O}(1)$

Table 1: Comparison of the number of queries to the quadruplet oracle (Quadr) and distance oracle (Dist) of our new algorithms with the state-of-the-art for k-center/median/means OracleCluster problem. All algorithms return constant approximation solutions. The results in Addanki et al. (2021) only hold for the k-center OracleCluster problem under the adversarial noise model. For the probabilistic noise model, their algorithm for k-center only hold if optimal k-center clusters are of size at least $\Omega(\sqrt{n})$.

II. Approximate Distance Data Structure. Beyond clustering, we also aim to understand the cost of learning a model of Σ that can approximately recover all pairwise distances. We formalize this as follows:

Problem OracleDS. Given a finite metric space $\Sigma = (\mathcal{V}, d)$ accessible under the RM model, the goal is to construct a data structure $\mathbf{D}$ such that, for any edge $\mathtt{e} \in \mathcal{E}$, it returns a value $\mathtt{d}(\mathtt{e}) \in \mathbb{R}_{\geq 0}$ satisfying $d(\mathtt{e}) \leq \mathtt{d}(\mathtt{e}) \leq O(1) \cdot d(\mathtt{e})$. Furthermore, $\mathbf{D}$ must be non-adaptive post-construction—i.e., it should not perform any additional oracle queries—and should be efficient in both storage and query time.

An approximate distance data structure is a powerful primitive: for many tasks (including clustering), approximate distances suffice to obtain high-quality solutions. However, this also makes the problem more challenging. We are especially interested in settings where OracleDS can be solved using $o(n^2)$ queries to the quadruplet oracle and $o(n)$ distance queries. Unfortunately, such guarantees seem challenging for general metrics. Hence, we ask:

Can we solve OracleDS using $\widetilde{O}(n)$ queries to the quadruplet oracle and $\widetilde{O}(1)$ queries to the distance queries in structured settings?

1.2. Our Contributions

In this paper, we answer both questions posed in Section 1.1 in the affirmative. Before stating our results, we introduce two standard notions used to capture structure in metric spaces. To model low-dimensional structure, we use the well-established notion of *bounded doubling dimension* Krauthgamer and Lee (2005). The doubling dimension of Σ is the smallest number $\dim(\Sigma)$ such that every ball of radius $\rho > 0$ can be covered by at most $2^{\dim(\Sigma)}$ balls of radius $\rho/2$. We say that Σ has bounded doubling dimension if $\dim(\Sigma) \leq C$ for some fixed constant C. We also consider metrics with *bounded spread*, where the spread of Σ is defined as $\$ = \max\{d(u,v) : u, v \in \mathcal{V}\} / \min\{d(u,v) : u \neq v \in \mathcal{V}\}$. A metric space is said to have bounded spread if $\$ \leq \mathsf{poly}(|\mathcal{V}|)$, where $\mathsf{poly}(\cdot)$ denotes a fixed polynomial.

Throughout the remainder of this paper, *we focus our presentation exclusively on the probabilistic error model*, as it is the more challenging case. However, our approach for the probabilistic model

also encompasses the adversarial model. In fact, all subsequent theorems continue to hold—often with the same or even better bounds.

Our first main contribution is the following result for the OracleCluster problem.

Theorem 1 *There exist randomized algorithms such that, given a finite metric space $\Sigma = (\mathcal{V}, d)$ of size $|\mathcal{V}| = n$, accessible under the* RM *model, for each of the k-center, k-median, and k-means instances of the OracleCluster problem on Σ, the corresponding algorithm, with high probability, computes an $O(1)$-approximation with the following query complexities:*

- *For general metrics, the algorithm uses $\widetilde{O}(1)$ distance queries and $\widetilde{O}(nk)$ quadruplet queries.*
- *If Σ has bounded doubling dimension, the algorithm uses $\widetilde{O}(1)$ distance queries and $\widetilde{O}(n + k^2)$ quadruplet queries.*

We note that our algorithms for OracleCluster are near-optimal, up to polylogarithmic factors for general metric spaces. It is known that any non-trivial approximation for k-center, k-median, or k-means clustering in general metric spaces requires $\Omega(nk)$ distance queries Bateni et al. (2023). Furthermore, Galhotra et al. Galhotra et al. (2024) show that in the RM model, even with a perfect quadruplet oracle, any algorithm for k-median or k-means must make at least $\Omega(1)$ distance queries. Together, these results imply that the combined number of distance and quadruplet queries must be at least $\Omega(nk)$, with at least $\Omega(1)$ distance queries. Additionally, even when the doubling dimension is bounded, any algorithm must examine at least $\Omega(n)$ edges. Hence, our algorithms achieve near-optimal query complexity for both general and low-dimensional metric spaces.

Our second contribution is an algorithm for the OracleDS problem when the underlying metric has bounded doubling dimension and bounded spread.

Theorem 2 *There exists a randomized algorithm which, given any metric space $\Sigma = (\mathcal{V}, d)$ of size $|\mathcal{V}| = n$, with bounded doubling dimension and polynomially bounded spread, and accessible under the* RM *model, constructs a data structure $\mathbf{D}$ that, with high probability, satisfies the following guarantees:*

- *For any edge $\mathtt{e} \in \mathcal{E}$, $\mathbf{D}$ returns a value $\mathtt{d}(\mathtt{e})$ satisfying $d(\mathtt{e}) \leq \mathtt{d}(\mathtt{e}) \leq O(1) \cdot d(\mathtt{e})$, in $O(\log n)$ time and without further oracle access.*
- *The construction of the data structure uses $\widetilde{O}(n)$ queries to the quadruplet oracle and $\widetilde{O}(1)$ queries to the distance oracle.*
- *The data structure requires $\widetilde{O}(n)$ space.*

Once $\mathbf{D}$ is constructed, it serves as a powerful primitive: any *robust* algorithm Alg designed for the full-information setting can be executed on Σ as a black box, using distances provided by $\mathbf{D}$. By *robust*, we mean that the algorithm tolerates distances being scaled by a constant factor. For instance, when Σ has bounded doubling dimension and polynomially bounded spread, $\mathbf{D}$ directly yields $O(1)$-approximation algorithms for the k-center, k-median, and k-means OracleCluster problems, without requiring additional oracle queries (see Table 1). Even when the spread is unbounded, our techniques extend to settings where the optimal solution is within a polynomial

factor of the spread (e.g., the Minimum Spanning Tree problem). For such problems, we derive $O(1)$-approximation algorithms using $\widetilde{O}(n)$ queries to the quadruplet oracle and $\widetilde{O}(1)$ queries to the distance oracle. In fact, Theorem 2 also plays a crucial role in our algorithm for k-median in Theorem 1 (when the spread is unbounded).

Finally, to evaluate the practical effectiveness of our approach, we implement the approximate distance data structure $\mathbf{D}$ and use it to run algorithms for k-center, k-median, and Minimum Spanning Tree. Experiments on both synthetic and real datasets show that our algorithms achieve constant-factor approximation while using only about 1% of the distance queries required by simple baselines.

2. Related Work

Addanki et al. (2021) introduced the study of clustering using the quadruplet oracle, considering both adversarial and probabilistic error models. In the adversarial setting, they designed a randomized $O(1)$-approximation algorithm requiring $O(nk^2 + nk\log^2 n)$ queries. In the probabilistic setting, they could not design a general algorithm without imposing structural assumptions on the optimal clustering. They also experimentally validated their model by training a classifier to emulate the quadruplet oracle from crowdsourced responses and using it for clustering.

Recently, Galhotra et al. (2024) improved the query complexity for k-center under the adversarial error model, achieving $O(nk + k^3)$ queries using only the quadruplet oracle. They also demonstrated that even with a perfect quadruplet oracle, it was not possible to obtain a $O(1)$-approximation for k-means and k-median without distance queries. This motivated them to consider a weak-strong framework, where the weak oracle provides inexpensive quadruplet comparisons, and the strong oracle returns exact distances but is expensive. Under this framework, they designed $O(1)$-approximation algorithms for k-center, k-median, and k-means clustering for general metric spaces, achieving query complexities of $O(nk\mathsf{polylog}\, n)$ for the quadruplet oracle and $O(k^2\mathsf{polylog}\, n)$ for the distance oracle under both adversarial and probabilistic error models. They further showed that when the doubling dimension is bounded, the approximation ratios could be improved without asymptotically increasing the query complexity. In this work, we improve the distance oracle queries by a factor of k^2 in all cases, and the quadruplet oracle by a factor of k when the doubling dimension is bounded.

Besides the aforementioned, the most closely related work to our context is by Bateni et al. (2024), who studied k-clustering, as well as the MST problem, in general metric spaces under a different weak-strong framework. While their strong oracle is identical to our distance oracle, their weak oracle differs: given a pair of vertices $u, v \in \mathcal{V}$, it returns $d(u, v)$ with probability at least $1 - \varepsilon$, and an arbitrary value otherwise. They designed $O(1)$-approximation algorithms for the k-center, k-median, and k-means problems, achieving $O(nk\,\mathsf{polylog}\, n)$ queries to the weak oracle and $O(k^2\mathsf{polylog}\, n)$ queries to the strong oracle. Additionally, they proved lower bounds showing that further improvements in query complexity were not possible within their framework. For the MST problem, they provided a $O(\sqrt{\log n})$-approximation using only queries to the weak oracle. We note that the challenges in our setting are different from those in Bateni et al. (2024).

Xu et al. (2024) also recently considered a weak-strong model, under metric spaces with bounded doubling dimension, where for any u, v, the weak oracle returns a number between $d(u, v)/C$ and $Cd(u, v)$ for some $C > 1$ and the strong oracle returns exact distances. However, their goal in this setting was to construct a data structure for approximate nearest neighbor queries. Silwal et al.

(2023) et al. have also studied the correlation clustering problem using a weak-strong framework, however, their techniques and oracles do not extended to the k-center/median/means clustering problems.

We note that there has been extensive work on using a probabilistic comparison oracle to approximately sort a set of elements. In this setting, it is well established that the *maximum dislocation* cannot be improved beyond $O(\log n)$, i.e., no algorithm can order a set of elements using such an oracle where every element is guaranteed to be placed within $o(\log n)$ positions of its true rank. After a long line of work Braverman and Mossel (2008); Braverman et al. (2016); Geissmann et al. (2017, 2020), it has only recently been resolved how to achieve $O(\log n)$ dislocation with $O(n \log n)$ queries with high probability.

Most of the other work on oracle-based clustering frameworks rely on optimal cluster queries Mazumdar and Saha (2017a); Huleihel et al. (2019); Mazumdar and Saha (2017b); Choudhury et al. (2019); Green Larsen et al. (2020); Galhotra et al. (2021) to identify ground truth clusters. Recent approaches for k-means Ashtiani et al. (2016); Chien et al. (2018); Kim and Ghosh (2017a,b) and k-median Ailon et al. (2018) leverage optimal cluster queries alongside distance information to achieve improved approximation guarantees. Beyond clustering, distance-based comparison oracles have been used to address a variety of problems, including learning fairness metrics Ilvento (2020), top-down hierarchical clustering Emamjomeh-Zadeh and Kempe (2018); Chatziafratis et al. (2018); Ghoshdastidar et al. (2019), correlation clustering Ukkonen (2017), and classification Tamuz et al. (2011); Hopkins et al. (2020). They have also been applied to tasks such as identifying the maximum Guo et al. (2012); Venetis et al. (2012), selecting the top-k elements Klein et al. (2011); Polychronopoulos et al. (2013); Ciceri et al. (2015); Davidson et al. (2014); Kou et al. (2017); Dushkin and Milo (2018), information retrieval Kazemi et al. (2018), and skyline computation Verdugo.

3. Technical Overview

Let $\Sigma = (\mathcal{V}, d)$ be a finite metric space with $|\mathcal{V}| = n$. For any subsets $\mathcal{U}, \mathcal{W} \subseteq \mathcal{V}$, let $\mathcal{E}(\mathcal{U}, \mathcal{W}) = \{\{u, w\} \mid u \in \mathcal{U},\ w \in \mathcal{W},\ u \neq w\}$ denote the edges between $\mathcal{U}$ and $\mathcal{W}$.

Our algorithms for both OracleCluster and OracleDS use $O(\text{polylog}\, n)$ distance queries; hence, they primarily rely on quadruplet queries. Even though a perfect quadruplet oracle only provides relative ordering information, the presence of probabilistic noise significantly complicates matters. For example, let $\mathcal{A} \subseteq \mathcal{V}$ be a subset of points and consider the task of assigning each point in $\mathcal{V} \setminus \mathcal{A}$ to an approximate nearest neighbor in $\mathcal{A}$. This task naturally arises in both of our problems. An intuitive approach would be to first order the edges in $\mathcal{Y} = \mathcal{E}(\mathcal{A}, \mathcal{V} \setminus \mathcal{A})$ using the quadruplet oracle and then use the resulting ordered sequence to assign nearest neighbors. To order $\mathcal{Y}$, one may employ existing algorithms from Geissmann et al. (2020); Klein et al. (2011) or Braverman and Mossel (2008). Hereafter, we denote this procedure by PROB-SORT (see Appendix A.2).

To understand what PROB-SORT does we need to introduce some definitions. For any edge set $\mathcal{X} \subseteq \mathcal{E}$, let $\pi_{\mathcal{X}}$ denote an *ordered sequence* over $\mathcal{X}$. For an ordering $\pi_{\mathcal{X}}$ and an edge $e \in \mathcal{X}$, let $\text{RANK}_{\pi_{\mathcal{X}}}(e)$ denote the index of e in $\pi_{\mathcal{X}}$, and let $\text{RANK}_{\mathcal{X}}(e)$ be the true rank of e when $\mathcal{X}$ is sorted in increasing order of distance. The *maximum dislocation* of $\pi_{\mathcal{X}}$ is then defined as $\max_{e \in \mathcal{X}} |\text{RANK}_{\pi_{\mathcal{X}}}(e) - \text{RANK}_{\mathcal{X}}(e)|$. Define $\mathcal{D}_{\Sigma} = c \log |\mathcal{V}|$, where $c > 1$ is a constant independent of Σ. We refer to $\pi_{\mathcal{X}}$ as a *low-dislocation ordering* of $\mathcal{X}$ if the maximum dislocation of $\pi_{\mathcal{X}}$ is at most $\mathcal{D}_{\Sigma}$. Intuitively, this means that each edge in $\mathcal{X}$ appears in $\pi_{\mathcal{X}}$ not too far from where it would be in the perfectly sorted order.

Returning to our original task, given $\mathcal{Y}$, with high probability PROB-SORT($\mathcal{Y}$) returns a low-dislocation ordering $\pi_{\mathcal{Y}}$ using $O(|\mathcal{Y}| \log n)$ quadruplet queries. Although the maximum dislocation is bounded, $\pi_{\mathcal{Y}}$ may still contain inversions. Therefore, we cannot rely solely on $\pi_{\mathcal{Y}}$ to assign nearest neighbors. One might attempt to resolve this by employing additional distance queries. Indeed, given any low-dislocation ordering of edges (such as $\pi_{\mathcal{Y}}$), we can employ a modified binary search routine, to extract the subset of edges whose weights are at most some threshold $\alpha \in \mathbb{R}$ using only $O(\mathsf{polylog}\, n)$ distance queries. Nevertheless, it remains unclear how to compute approximate nearest neighbors with few distance queries. Without additional assumptions on the spread of Σ, simple methods such as bucketing the edges in $\mathcal{Y}$ into groups of similar weight require too many distance queries. Furthermore, even assuming bounded spread, since our algorithms for bounded doubling dimension aim to use use $O(n\, \mathsf{polylog}\, n)$ quadruplet queries, directly applying PROB-SORT on the entire set $\mathcal{Y}$ may not be feasible if $|\mathcal{Y}| = \omega(n)$.

A more general complication arising from our probabilistic noise model is persistence: if our algorithm reuses the outcome of a probabilistic quadruplet query from an earlier step (i.e., they share randomness), the success probabilities across steps become correlated in a way that complicates the analysis. This issue also arises in sorting with persistent errors Braverman and Mossel (2008); Braverman et al. (2016); Geissmann et al. (2017, 2020); Klein et al. (2011), and makes this setting significantly more challenging than one where the oracle independently returns fresh answers for repeated queries (for example, see Davidson et al. (2014)).

In the following, we give an overview of our main techniques, focusing on the bounded doubling dimension case, which is more technically challenging due to the stronger query complexity bounds. The case for general metrics is simpler requiring only a subset of these techniques.

3.1. k-center OracleCluster

Henceforth, we assume that Σ has bounded doubling dimension. For subsets $\mathcal{X}, \mathcal{Y} \subseteq \mathcal{V}$, let $\textsc{Cost}^{\infty}_{\mathcal{Y}}(\mathcal{X}) := \max_{v \in \mathcal{X}} d(v, \mathcal{Y})$. For a subset $\mathcal{W} \subseteq \mathcal{V}$, the cost of the optimal k-center solution for the vertices in $\mathcal{W}$ is denoted by $\mathsf{OPT}^{\infty}(\mathcal{W}) = \min_{\mathcal{U} \subseteq \mathcal{W}, |\mathcal{U}| \leq k} \textsc{Cost}^{\infty}_{\mathcal{U}}(\mathcal{W})$. If $\mathcal{W} = \mathcal{V}$, we simply write OPT^{∞}. The goal of the k-center problem is to compute a set $\mathcal{A} \subseteq \mathcal{V}$ of k points such that $\textsc{Cost}^{\infty}_{\mathcal{A}}(\mathcal{V}) = \mathsf{OPT}^{\infty}$.

Let us consider a well known property of k-center problem that follows directly from Vapnik and Chervonenkis (1971) (see also Har-Peled (2001)). For any point $v \in \mathcal{V}$ and $\rho \geq 0$, let $\mathfrak{B}(v, \rho) = \{u \in \mathcal{V} \mid d(v, u) \leq \rho\}$ denote the *ball* centered at v with radius ρ.

Proposition 1 (informal) *Let $\mathcal{S} \subseteq \mathcal{V}$ be a random sample of $\Theta(k \log n)$ vertices. Suppose $\mathcal{C} \subseteq \mathcal{S}$ is a k-center solution for $\mathcal{S}$ with cost $\rho = \textsc{Cost}^{\infty}_{\mathcal{C}}(\mathcal{S})$ such that $\rho \leq O(1) \cdot \mathsf{OPT}^{\infty}(\mathcal{S})$. Then with high probability, $\rho \leq O(1) \cdot \mathsf{OPT}^{\infty}$ and $\left|\bigcup_{v \in \mathcal{C}} \mathfrak{B}(v, \rho)\right| \geq \Omega(|\mathcal{V}|)$.*

In other words, a near-optimal k-center solution for a random subset of size $\Theta(k \log n)$ is very likely a near-optimal solution for a large fraction of the vertices in $\mathcal{V}$. Our high-level plan is to design a recursive sampling algorithm motivated by this insight. The principle challenge lies in efficiently identifying the set of *good* vertices $\mathcal{U} = \bigcup_{v \in \mathcal{C}} \mathfrak{B}(v, \rho)$. To meet the desired query complexities bounds, we cannot use more than $O(n\, \mathsf{polylog}\, n)$ quadruplet queries and $O(\mathsf{polylog}\, n)$ distance queries. A natural approach is to construct a nearest-neighbor data structure on $\mathcal{C}$ and then use it to efficiently determine $\mathcal{U}$. However, as discussed, this poses several challenges.

Our first key idea is to transform the task of identifying the vertices in $\mathcal{U}$ from the probabilistic noise model to the adversarial noise model. Under probabilistic noise, if the quadruplet oracle

incorrectly reports that $d(\mathbb{e}_1) \leq d(\mathbb{e}_2)$, for distinct edges $\mathbb{e}_1, \mathbb{e}_2 \in \mathcal{E}$, there is no bound on the error (the ratio $d(\mathbb{e}_1)/d(\mathbb{e}_2)$ can be arbitrarily large). On the other hand, in the adversarial noise setting, when the quadruplet oracle reports that $d(\mathbb{e}_1) \leq d(\mathbb{e}_2)$, it is guaranteed that $d(\mathbb{e}_1) \leq (1+\mu) \cdot d(\mathbb{e}_2)$, where $\mu \geq 0$ is the adversarial noise. To perform this transformation, we require something stronger than Proposition 1. We use the following stronger property:

Proposition 2 (informal) *Let $\mathcal{S}^{(1)} \subseteq \mathcal{V}$ and $\mathcal{S}^{(2)} \subseteq \mathcal{V} \setminus \mathcal{S}^{(1)}$ be random samples of size $\Theta(k \log n)$. Suppose $\mathcal{C} \subseteq \mathcal{S}^{(1)}$ is a k-center solution for $\mathcal{S}^{(1)}$ with cost $\rho = \textsc{Cost}^{\infty}_{\mathcal{C}}(\mathcal{S}^{(1)})$ such that $\rho \leq O(1) \cdot \mathsf{OPT}^{\infty}(\mathcal{S}^{(1)})$. Define $\mathcal{C}^{(H)} \subseteq \mathcal{C}$ to be the set of* center-heavy *points: those $v \in \mathcal{C}$ for which $|\mathfrak{B}(v, \rho) \cap \mathcal{S}^{(2)}| = \Omega(\log n)$. Let $\mathcal{H} \subseteq \mathcal{C}^{(H)}$ be any* maximal *subset such that for every pair of distinct $h_1, h_2 \in \mathcal{H}$, $d(h_1, h_2) > 2\rho$. Then, with high probability $\rho \leq O(1) \cdot \mathsf{OPT}^{\infty}$ and $\left|\bigcup_{h \in \mathcal{H}} \mathfrak{B}(h, 4\rho) \cap \mathcal{V}\right| \geq \Omega(|\mathcal{V}|)$.*

Informally, the proposition says that given a near-optimal k-center solution $\mathcal{C}$ for $\mathcal{S}^{(1)}$, a *maximal well-separated subset* of *center-heavy* vertices also serves as a near-optimal k-center solution for a large fraction of points in $\mathcal{V}$. We now present a sketch of our k-center algorithm.

Overview of ORACLE-CENTER. The algorithm works in two phases. In Phase 1, begin with the entire vertex set $\mathcal{V}$ and repeat the following steps: First, randomly select two subsets, $\mathcal{S}^{(1)} \subseteq \mathcal{V}$ and $\mathcal{S}^{(2)} \subseteq \mathcal{V} \setminus \mathcal{S}^{(1)}$. Then, compute a k-center solution $\mathcal{C}$ for $\mathcal{S}^{(1)}$, and use $\mathcal{S}^{(2)}$ to identify the subset of center-heavy vertices $\mathcal{C}^{(H)} \subseteq \mathcal{C}$. Next, extract a maximal well separated subset $\mathcal{H} \subseteq \mathcal{C}^{(H)}$. For each $v \in \mathcal{V}$, identify an approximate nearest neighbor in $\mathcal{H}$ and determine $\mathcal{U} = \bigcup_{h \in \mathcal{H}} \mathfrak{B}(h, O(1) \cdot \rho) \cap \mathcal{V}$. Remove the combined sample $\mathcal{S} = \mathcal{S}^{(1)} \cup \mathcal{S}^{(2)}$ and the set $\mathcal{U}$ from $\mathcal{V}$, and iterate on the remaining vertices.

In Phase 2, aggregate all centers computed during across the rounds along with the sampled vertices to form a coreset $\hat{\mathcal{S}}$ of size $O(k \,\mathsf{polylog}\, n)$, and compute the final k-center solution on $\hat{\mathcal{S}}$. [2]

To meet our desired query complexities, the two main challenges are: (I) computing an $O(1)$-approximate k-center solution for each $\mathcal{S}^{(1)}$ and ultimately for the coreset $\hat{\mathcal{S}}$, and (II) accurately identifying the set of *good* vertices $\mathcal{U}$. The remaining steps, such as identifying $\mathcal{C}^{(H)}$ from $\mathcal{C}$, can be implemented using PROB-SORT and a modified binary search routine (see FILTER in Appendix A.2).

• **Identifying the good vertices.** We first consider (II), the task of identifying the good vertices. This is the part where the structural properties of $\mathcal{H}$ become relevant. Suppose we have computed the set $\mathcal{H}$. For every $h \in \mathcal{H}$, fix an arbitrary subset $\textsc{core}(h) \subseteq \mathfrak{B}(h, \rho) \cap \mathcal{S}^{(2)}$ of size $\Theta(\log n)$ and refer to it as the *core* of h. Let $\textsc{core} = \bigcup_{h \in \mathcal{H}} \textsc{core}(h)$. Let $\mathcal{V}' = \mathcal{V} \setminus \mathcal{S}$. In Appendix B.1, we prove the following:

Proposition 3 (informal) *There exists a procedure* HEAVY-ANN *such that, given $\mathcal{H}, \textsc{core}, \mathcal{V}'$, with high probability returns a set of edges $\mathcal{N} \subseteq \mathcal{E}(\mathcal{H}, \mathcal{V}')$ of size $O(|\mathcal{V}'| \log n)$ such that for every $v \in \mathcal{V}'$, there exists an edge $\mathbb{e} \in \mathcal{N} \cap \mathcal{E}(v, \mathcal{H})$ satisfying $d(\mathbb{e}) \leq O(1) \cdot d(v, \mathcal{H})$. Furthermore,the procedure makes $O((n + |\mathcal{H}|^2)\,\mathsf{polylog}\,(n))$ queries to the probabilistic quadruplet oracle and no queries to the distance oracle.*

The set $\mathcal{N}$ enables us to recover $\mathcal{U}$ in a straightforward way using only a small number of additional queries . Consequently, we shift our focus to the design of the HEAVY-ANN procedure,

2. A (weighted) set $\mathcal{M} \subseteq \mathcal{V}$ is a coreset for the k-center (resp. k-median, k-means) OracleCluster, if any γ-approximate solution computed on $\mathcal{M}$ is an $O(\gamma)$-approximate solution to $\mathcal{V}$ for any $\gamma \geq 1$.

which relies on two key ideas. In Appendix B.1.1, we show there exists a *rich* subset of queries $\mathcal{J} \subseteq \mathcal{E}(\mathcal{V}', \mathcal{H}) \times \mathcal{E}(\mathcal{V}', \mathcal{H})$, such that with high probability, for any query $(\mathtt{e}_1, \mathtt{e}_2) \in \mathcal{J}$, we can accurately test if $d(\mathtt{e}_1) \geq d(\mathtt{e}_2)$ holds, as long as $d(\mathtt{e}_1)$ and $d(\mathtt{e}_2)$ are not within a factor of two. We denote this tester by $\hat{\mathcal{Q}}$. Observe that $\hat{\mathcal{Q}}$ behaves equivalently to a quadruplet oracle with adversarial noise $\mu = 1$ (see (b)) on queries belonging to $\mathcal{J}$. The design of $\hat{\mathcal{Q}}$ requires three key observations: vertices in $\mathcal{H}_i$ are well-separated (2ρ-separated); vertices in each $\textsc{core}(h)$ are relatively close to h (ρ-close); and each $\textsc{core}(h)$ contains $\Omega(\log n)$ elements (heavy).

In Appendix B.1.2, we use $\hat{\mathcal{Q}}$ to construct a nearest-neighbor data structure on $\mathcal{H}$ and use it to obtain the set $\mathcal{N}$. The data structure is inspired by *ring-separator trees* Indyk and Motwani (1998); Krauthgamer and Lee (2005); Har-Peled and Mendel (2005). At a high level, these trees exploit the property that in doubling metrics, a set of points can be partitioned into dense inner and outer rings, separated by a wide annulus in between. The challenge lies in constructing the tree, particularly in identifying the separating annulus, and subsequently traversing it while using only $\hat{\mathcal{Q}}$. We show that the richness of $\mathcal{J}$ along with the doubling property allows us to do these without additional distance queries.

• **Computing a local k-center solution.** For task (I), the task of computing local k-center solutions on sets of size $O(k \operatorname{polylog} n)$, unlike prior work, we cannot afford to explicitly compute all pairwise distances. We need a different approach. Let $\mathcal{X} \subseteq \mathcal{E}$. We say that $\Pi = \{\pi_{\mathcal{X}_1}, \ldots, \pi_{\mathcal{X}_m}\}$ is a *low-dislocation decomposition* of $\mathcal{X}$ if the sets $\{\mathcal{X}_1, \ldots, \mathcal{X}_m\}$ form a partition of $\mathcal{X}$, and each $\pi_{\mathcal{X}_i} \in \Pi$ is a low-dislocation ordering of $\mathcal{X}_i$. Informally, a low-dislocation decomposition of a set of edges is a partition, where each component is almost sorted. In Appendix B.2, we design a procedure BASIC-CENTER that, given a set of vertices $\mathcal{S}$ and a low-dislocation decomposition of $\mathcal{E}(\mathcal{S}, \mathcal{S})$, returns a k-center solution for $\mathcal{S}$ without using any quadruplet queries and only $O(m^2 \operatorname{polylog} n)$ distance queries, where m is the number of components in the decomposition. BASIC-CENTER relies on a standard reduction of the k-center problem to finding a maximal independent set in an unweighted "threshold" graph Hochbaum and Shmoys (1986). This incurs a factor of 2 in the approximation. To find the correct threshold BASIC-CENTER implements a modified binary search on each sequence in the decomposition. Since each sequence is a low-dislocation ordering as long as the number of sequences in the decomposition is small the overall number of distance queries remains small.

In a particular round of Phase 1 of the main algorithm, we first obtain a low-dislocation ordering of $\mathcal{E}(\mathcal{S}^{(1)}, \mathcal{S}^{(1)})$ using PROB-SORT, which requires only $O(k^2 \operatorname{polylog} n)$ quadruplet oracle queries. We then directly invoke BASIC-CENTER. In this case, the decomposition only contains a single sequence. The more interesting case arises when computing the final solution on the coreset $\hat{\mathcal{S}}$. Due to technical reasons arising from persistence, we cannot directly apply PROB-SORT to $\mathcal{E}(\hat{\mathcal{S}}, \hat{\mathcal{S}})$. Instead, the algorithm constructs a low-dislocation decomposition of $\mathcal{E}(\hat{\mathcal{S}}, \hat{\mathcal{S}})$ comprising of $O(\operatorname{polylog} n)$ sequences and then invokes BASIC-CENTER.

3.2. Approximate Distance Data Structure (OracleDS)

We design a data structure $\mathbf{D}$ for the OracleDS problem when Σ has both bounded doubling dimension and bounded spread. Specifically, given an edge e, $\mathbf{D}$ returns a value $\mathtt{d}(e)$ satisfying $d(e) \leq \mathtt{d}(e) \leq O(1) \cdot d(e)$ in $O(\log n)$ time, without additional quadruplet or distance oracle queries. Furthermore, constructing $\mathbf{D}$ uses $O(n \operatorname{polylog} n)$ quadruplet queries and $O(\operatorname{polylog} n)$ distance queries, and requires $O(n \operatorname{polylog} n)$ storage.

For any subset $\mathcal{U} \subseteq \mathcal{V}$, we define its *diameter* as $\mathsf{diam}(\mathcal{U}) = \max_{u,v \in \mathcal{U}} d(u,v)$ and its *separation* as $\mathsf{sep}(\mathcal{U}) = \min_{\substack{u,v \in \mathcal{U} \\ u \neq v}} d(u,v)$. The *spread* (or aspect ratio) of Σ is given by $\mathbb{S} = \mathsf{diam}(\mathcal{V})/\mathsf{sep}(\mathcal{V})$. If $\mathbb{S} \leq \mathsf{poly}(n)$ where $\mathsf{poly}(\cdot)$ is some fixed polynomial, we say that Σ has *bounded spread*.

A naive approach to building $\mathbf{D}$ would be to obtain a low-dislocation ordering of $\mathcal{E}$ using PROB-SORT and then bucketing the edges into $O(\log n)$ buckets. This would require only $O(\mathsf{polylog}\, n)$ distance queries (as the spread is bounded) but $O(n^2)$ quadruplet queries. To achieve the desired number of quadruplet queries, we need a more sophisticated approach.

For subsets $\mathcal{U}, \mathcal{W} \subseteq \mathcal{V}$, we say that $\mathcal{U}$ is an α-*cover* of $\mathcal{W}$ if for all $w \in \mathcal{W}$, there exists $u \in \mathcal{U}$ such that $d(w,u) \leq \alpha$. The set $\mathcal{U}$ is said to be β-*separated* if for any two distinct points $u, u' \in \mathcal{U}$, we have $d(u,u') > \beta$. A set $\mathcal{U}$ is called an (α, β)-*net* of $\mathcal{W}$ if it is both an α-cover of $\mathcal{W}$ and β-separated. Our goal is to compute a collection of nets to approximate Σ at every scale. We construct a data structure with the following structure.

Structure. Let $\mu = \lceil \log \mathsf{diam}(\Sigma) \rceil$ and $\lambda = \lfloor \log \mathsf{sep}(\Sigma) \rfloor$. The data structure comprises of a collection of vertex-subsets $\Gamma = \{\mathcal{L}_\mu, \mathcal{L}_{\mu-1}, \ldots, \mathcal{L}_{\lambda+1}, \mathcal{L}_\lambda\}$ with each $\mathcal{L}_i \subseteq \mathcal{V}$, such that the following properties hold:

1. For every integer $i \in [\lambda, \mu]$, the set $\mathcal{L}_i$ is a $(2^i, 2^{i-1})$-net of $\mathcal{V}$.
2. For each $v \in \mathcal{L}_i$, there exists a set $\mathfrak{B}_v \subseteq \mathfrak{B}(v, 2^i) \cap \mathcal{V}$ containing v, and these sets form a partition of $\mathcal{V}$.

While several known methods Beygelzimer et al. (2006); Krauthgamer and Lee (2004); Talwar (2004) exist to construct similar structures when all distances are accessible—including efficient variants requiring only $O(n \cdot \mathsf{polylog}\, n)$ distances Har-Peled and Mendel (2005)—it is not clear how to adapt these approaches to our setting while achieving the desired query complexities. We need a new approach. Our algorithm is based on the following observation:

Key Observation. Suppose that $\mathcal{L}_{i+1}$ has already been computed. We know that for each $v \in \mathcal{L}_{i+1}$, there is a set $\mathfrak{B}_v \subseteq \mathfrak{B}(v, 2^i) \cap \mathcal{V}$ containing v, and these sets form a partition of $\mathcal{V}$. Moreover, each $v \in \mathcal{L}_{i+1}$ acts as a 2^{i+1}-cover for its corresponding set $\mathfrak{B}_v$. Let $\zeta = 2^{\dim(\Sigma)}$. By the doubling property, each $\mathfrak{B}_v$ can be covered by ζ balls of radius 2^i. For every $v \in \mathcal{L}_{i+1}$, suppose we independently sample a subset $\mathcal{S}_v \subseteq \mathfrak{B}_v$ consisting of $\kappa = \Theta(\zeta \cdot \log n)$ points. We can use Proposition 1 to assert that, with high probability, for each v there exists a subset of good vertices $\mathfrak{B}'_v \subseteq \mathfrak{B}_v$ satisfying $|\mathfrak{B}'_v| \geq \Omega(|\mathfrak{B}_v|)$ such that every vertex in $\mathfrak{B}'_v$ lies within distance 2^i of some point in $\mathcal{S}_v$. The key insight is all the $\mathfrak{B}'_v$ can be computed concurrently which significantly reduces the number of oracle queries. Specifically, one can first compute a low-dislocation ordering of $\bigcup_{v \in \mathcal{L}_{i+1}} \mathfrak{B}'_v$ and then identify all the $\mathfrak{B}'_v$ concurrently using a modified binary search requiring only $O(\mathsf{polylog}\, n)$ distance queries.

Overview of CONSTRUCT-NET. The algorithm, computes the nets in a top-down manner, from scale μ to scale λ. Suppose that $\mathcal{L}_{i+1}$ has already been computed. To compute $\mathcal{L}_i$, the algorithm operates in two phases.

In the Phase 1, the algorithm essentially finds a k-center coreset (with $k = \Theta(\zeta)$) for each $\mathfrak{B}_v$, where $v \in \mathcal{L}_{i+1}$. It does this by using a recursive sampling algorithm based on Proposition 1. Each $\mathfrak{B}_v$ is processed independently with respect to sampling; however, in every round of the recursive

procedure, the good vertices are identified via shared queries based on the observation above. Consequently, the k-center procedures must be executed in lock-step. At the end of this phase, the covering property is enforced.

In the Phase 2, the algorithm enforces the separation property. The challenge is to ensure that samples from different sets B_v are not too close. To achieve this, the algorithm reduces the problem to computing a maximal independent set over a carefully constructed graph. To prevent the graph from growing too large, it meticulously maintains each center's conflict set—that is, the set of other centers that might violate the separation condition. In particular, it uses the conflict sets from level $\mathcal{L}_{i+1}$ to infer potential conflicts and construct the graph. This step is crucial for keeping the total number of quadruplet queries at $O(n \cdot \mathsf{polylog}\, n)$. In both phases, the algorithm also tracks additional information to ensure that previously issued quadruplet queries are not reused.

Once the construction is complete, the query procedure is straightforward and involves going to each of the $O(\log n)$ scales and inspecting the conflict-sets. Overall, it requires $O(\log n)$ steps.

Applications. The data structure yields a meta-algorithm for many metric graph-optimization problems. Given the metric Σ, one first builds the data structure as in Theorem 2. Then, any standard (exact or approximation) algorithm is run, replacing every distance query with a query to the data structure. Importantly, both the quadruplet and distance queries are used only during the construction of the data structure. The only requirement is that the algorithm is robust to distances being stretched by a constant factor. However, standard algorithms for many metric problems—such as k-center, k-median, and k-means clustering Gonzalez (1985); Arya et al. (2001); Gupta and Tangwongsan (2008); Kanungo et al. (2002), minimum spanning tree Kruskal (1956), facility location Arya et al. (2001)—are either robust or can be made robust with little effort. Thus, for these problems on metric spaces with bounded doubling dimension and polynomially bounded spread, there exist $O(1)$-approximation algorithms that use $O(n \cdot \mathsf{polylog}\, n)$ quadruplet queries and $O(\mathsf{polylog}\, n)$ distance queries.

Interestingly, Theorem 2 can be used even if the spread is not polynomially bounded. Let $\mathsf{diam} = \mathsf{diam}(\mathcal{V})$. We say a problem Prob is *diameter-opt relevant* if the value of the optimal solution on Σ, denoted by $\mathsf{Opt}_\Sigma^{\mathsf{Prob}}$, satisfies $\frac{1}{\mathrm{poly}(n)} \cdot \mathsf{diam} \le \mathsf{Opt}_\Sigma^{\mathsf{Prob}} \le \mathrm{poly}(n) \cdot \mathsf{diam}$ for some fixed polynomial function $\mathrm{poly}(n) > 1$. While classical k-clustering problems are not diameter-opt relevant, other well-known metric problems satisfy this condition. For example, the MST problem is diameter-opt relevant and even when the spread is unbounded, our technique yields $O(1)$-approximation that uses $O(n \cdot \mathsf{polylog}\, n)$ quadruplet queries and $O(\mathsf{polylog}\, n)$ distance queries.

3.3. k-median/means OracleCluster

We focus on the k-median OracleCluster problem—though our approach applies almost verbatim to the k-means problem. The goal of the k-median problem is to compute a set $\mathcal{A} \subseteq \mathcal{V}$ of k points such that $\textsc{Cost}^1_{\mathcal{A}}(\mathcal{V}) = \min_{\mathcal{U} \subseteq \mathcal{V}, |\mathcal{U}|=k} \textsc{Cost}^1_{\mathcal{U}}(\mathcal{V})$. We denote the costs of the optimal k-median solution with respect to a set $\mathcal{W} \subseteq \mathcal{V}$ as $\mathsf{OPT}^1(\mathcal{W})$. If $\mathcal{W} = \mathcal{V}$, we simply write OPT^1.

Overview of ORACLE-MEDIAN. Our algorithm critically uses both our k-center OracleCluster algorithm ORACLE-CENTER and the approximate distance oracle data structure CONSTRUCT-NET. Since we integrate different independent algorithms, due to the persistence of the quadruplet oracle we must take great care to ensure that the success probabilities of each component remain independent. This is a non-trivial task that requires a careful design. At a high-level, or algorithm can be divided into three parts.

I. Partitioning $\mathcal{V}$. Our first step is to decompose $\mathcal{V}$ into components that satisfy certain nice properties (which we discuss below). To obtain such a partition, we first run our algorithm for the k-center OracleCluster problem on $\mathcal{V}$. The algorithm returns a set $\mathcal{C}$ of k centers and a value ρ^* such that $\rho^* \leq O(1) \cdot \mathsf{OPT}^{\infty}(\mathcal{V})$. An important observation here is that the clustering costs of the optimal k-center and k-median solutions are within an n factor of each other, i.e., $\mathsf{OPT}^{\infty}(\mathcal{V}) \leq \mathsf{OPT}^{1}(\mathcal{V}) \leq n \cdot \mathsf{OPT}^{\infty}(\mathcal{V})$. We use $\mathcal{C}$ and ρ^* to partition of $\mathcal{V}$ in to components $\mathcal{U}_1, \ldots, \mathcal{U}_p, \mathcal{W}$, that satisfy the following properties:

1. The diameter of each component $\mathcal{U}_i$ is bounded by $n^5 \cdot \mathsf{OPT}^{1}$.
2. In any optimal k-median solution $\mathcal{A}^* \subseteq \mathcal{V}$, if a vertex $v \in \mathcal{U}_i$ is mapped to some $a \in \mathcal{A}^*$, then a cannot belong to any $\mathcal{U}_j$ for $j \neq i$.

II. Constructing a Data Structure on each Component. Next, the algorithm constructs an approximate distance oracle data structure $\mathbf{D}_i$ on each $\mathcal{U}_i$ using CONSTRUCT-NET. In particular, we only compute the data structure from scale $\mu = \log\lceil n^5 \mathsf{OPT}^{1} \rceil$ down to scale $\lambda = \log\lceil \frac{1}{n^5} \cdot \mathsf{OPT}^{1} \rceil$. Naively, constructing each $\mathbf{D}_i$ could consume too many queries since there can be up to k components, hence, we need to be careful.

Roughly speaking, constructing a particular level of each $\mathbf{D}_i$ is analogous to independently solving a k-center problem (with $k = O(1)$) on a collection of sets, each with the same diameter. The algorithm in Section C shows how the oracle queries can be efficiently shared between these sets. The crucial observation is that, since the different $\mathbf{D}_i$ are built at the same scale, queries can not only be shared within each $\mathbf{D}_i$ but also across different ones. This optimization is essential for achieving the desired query bounds.

III. Computing the Final Solution. In each partition $\mathcal{U}_i$, the data structure $\mathbf{D}_i$ rounds distances below $\frac{1}{n^5}\mathsf{OPT}^{1}$ to $\frac{O(1)}{n^5}\mathsf{OPT}^{1}$ (contributing at most $\frac{O(1)}{n^4}\mathsf{OPT}^{1}$ to the final cost) and stretches larger distances by a constant factor. As a result, we can run a standard k-median algorithm Alg_M on $\mathcal{U}_i$, using $\mathbf{D}_i$ to answer distance queries, and still achieve a solution with cost $O(1) \cdot \mathsf{OPT}^{1}(\mathcal{U}_i)$. Let $\mathcal{A}_i$ denote the set of k vertices returned by the Alg_M on $\mathcal{U}_i$ (using distances from $\mathbf{D}_i$). Assign each vertex $v \in \mathcal{A}_i$ a weight equal to the number of vertices in $\mathcal{U}_i$ that are mapped to v.

By standard properties of k-median, the union $\mathcal{M} = \mathcal{W} \cup \bigcup_{i \in [p]} \mathcal{A}_i$ forms a coreset on $\mathcal{V}$. Finally, the algorithm runs Alg_M on $\mathcal{M}$. It again uses the data structures $\mathbf{D}_i$ to approximate pairwise distances within each $\mathcal{U}_i$. Furthermore, it treats distances between $\mathcal{U}_i$ and $\mathcal{U}_j$ for $i \neq j$ as ∞. Handling edges between each $\mathcal{E}(\mathcal{W}, \mathcal{W})$ and $\mathcal{E}(\mathcal{U}_i, \mathcal{W})$ requires some careful processing but we skip those details here. The crux of the analysis is in showing that the different components remain independent. This is where the choice of $\mathcal{W}$ becomes particularly important.

4. Conclusion

We proposed near-optimal algorithms for clustering and other optimization problems in a metric space having access to a quadruplet (weak) and a distance (strong) oracle. Our results improved all recent known results in this setting. There are multiple open problems derived by our work. First, it is worth exploring whether our data structure can be used to design a meta-algorithm for all graph problems in metric spaces with a constant doubling dimension. Second, we plan to extend our results to non-metric graphs, other realistic oracles, and more complicated error models.

Acknowledgments

This work was partially supported by NSF grants IIS-2348919, IIS-2402823, and a grant by Infosys.

References

Jayadev Acharya, Moein Falahatgar, Ashkan Jafarpour, Alon Orlitsky, and Ananda Theertha Suresh. Maximum selection and sorting with adversarial comparators. *The Journal of Machine Learning Research*, 19(1):2427–2457, 2018.

Raghavendra Addanki, Sainyam Galhotra, and Barna Saha. How to design robust algorithms using noisy comparison oracle. *Proceedings of the VLDB Endowment*, 14(10):1703–1716, 2021.

Nir Ailon, Anup Bhattacharya, Ragesh Jaiswal, and Amit Kumar. Approximate clustering with same-cluster queries. In *9th Innovations in Theoretical Computer Science Conference (ITCS 2018)*, volume 94, page 40. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2018.

Vijay Arya, Naveen Garg, Rohit Khandekar, Adam Meyerson, Kamesh Munagala, and Vinayaka Pandit. Local search heuristic for k-median and facility location problems. In *Proceedings of the thirty-third annual ACM symposium on Theory of computing*, pages 21–29, 2001.

Hassan Ashtiani, Shrinu Kushagra, and Shai Ben-David. Clustering with same-cluster queries. In *Advances in neural information processing systems*, pages 3216–3224, 2016.

Arthur Asuncion, David Newman, et al. Uci machine learning repository, 2007.

MohammadHossein Bateni, Hossein Esfandiari, Hendrik Fichtenberger, Monika Henzinger, Rajesh Jayaram, Vahab Mirrokni, and Andreas Wiese. Optimal fully dynamic k-center clustering for adaptive and oblivious adversaries. In *Proceedings of the 2023 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2677–2727. SIAM, 2023.

MohammadHossein Bateni, Prathamesh Dharangutte, Rajesh Jayaram, and Chen Wang. Metric clustering and mst with strong and weak distance oracles. In *The Thirty Seventh Annual Conference on Learning Theory*, pages 498–550. PMLR, 2024.

Alina Beygelzimer, Sham Kakade, and John Langford. Cover trees for nearest neighbor. In *Proceedings of the 23rd international conference on Machine learning*, pages 97–104, 2006.

Mark Braverman and Elchanan Mossel. Noisy sorting without resampling. In *Proceedings of the nineteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 268–276. Society for Industrial and Applied Mathematics, 2008.

Mark Braverman, Jieming Mao, and S Matthew Weinberg. Parallel algorithms for select and partition with noisy comparisons. In *Proceedings of the forty-eighth annual ACM symposium on Theory of Computing*, pages 851–862, 2016.

Chris Burges, Tal Shaked, Erin Renshaw, Ari Lazier, Matt Deeds, Nicole Hamilton, and Greg Hullender. Learning to rank using gradient descent. In *Proceedings of the 22nd International Conference on Machine Learning (ICML)*, pages 89–96, 2005. doi: 10.1145/1102351.1102363.

Vaggos Chatziafratis, Rad Niazadeh, and Moses Charikar. Hierarchical clustering with structural constraints. In *International conference on machine learning*, pages 774–783. PMLR, 2018.

I Chien, Chao Pan, and Olgica Milenkovic. Query k-means clustering and the double dixie cup problem. In *Advances in Neural Information Processing Systems*, pages 6649–6658, 2018.

Tuhinangshu Choudhury, Dhruti Shah, and Nikhil Karamchandani. Top-m clustering with a noisy oracle. In *2019 National Conference on Communications (NCC)*, pages 1–6. IEEE, 2019.

Eleonora Ciceri, Piero Fraternali, Davide Martinenghi, and Marco Tagliasacchi. Crowdsourcing for top-k query processing over uncertain data. *IEEE Transactions on Knowledge and Data Engineering*, 28(1):41–53, 2015.

Susan Davidson, Sanjeev Khanna, Tova Milo, and Sudeepa Roy. Top-k and clustering with noisy comparisons. *ACM Transactions on Database Systems (TODS)*, 39(4):1–39, 2014.

Marie-Paule Dubuisson and Anil K Jain. A modified hausdorff distance for object matching. In *Proceedings of the 12th International Conference on Pattern Recognition*, pages 566–568, 1994.

Eyal Dushkin and Tova Milo. Top-k sorting under partial order information. In *Proceedings of the 2018 International Conference on Management of Data*, pages 1007–1019, 2018.

Ehsan Emamjomeh-Zadeh and David Kempe. Adaptive hierarchical clustering using ordinal queries. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 415–429. SIAM, 2018.

Sainyam Galhotra, Sandhya Saisubramanian, and Shlomo Zilberstein. Learning to generate fair clusters from demonstrations. In *Proceedings of the 2021 AAAI/ACM Conference on AI, Ethics, and Society*, pages 491–501, 2021.

Sainyam Galhotra, Rahul Raychaudhury, and Stavros Sintos. k-clustering with comparison and distance oracles. *Proceedings of the ACM on Management of Data*, 2(5):1–26, 2024.

Barbara Geissmann, Stefano Leucci, Chih-Hung Liu, and Paolo Penna. Sorting with recurrent comparison errors. In *28th International Symposium on Algorithms and Computation (ISAAC 2017)*, 2017.

Barbara Geissmann, Stefano Leucci, Chih-Hung Liu, and Paolo Penna. Optimal sorting with persistent comparison errors. In *27th Annual European Symposium on Algorithms*, pages 1–14, 2019.

Barbara Geissmann, Stefano Leucci, Chih-Hung Liu, and Paolo Penna. Optimal dislocation with persistent errors in subquadratic time. *Theory of Computing Systems*, 64(3):508–521, 2020.

Debarghya Ghoshdastidar, Michaël Perrot, and Ulrike von Luxburg. Foundations of comparison-based hierarchical clustering. In *Advances in Neural Information Processing Systems*, pages 7454–7464, 2019.

Teofilo F Gonzalez. Clustering to minimize the maximum intercluster distance. *Theoretical Computer Science*, 38:293–306, 1985.

Kasper Green Larsen, Michael Mitzenmacher, and Charalampos Tsourakakis. Clustering with a faulty oracle. In *Proceedings of The Web Conference 2020*, WWW '20, page 2831–2834, 2020.

Stephen Guo, Aditya Parameswaran, and Hector Garcia-Molina. So who won? dynamic max discovery with the crowd. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, pages 385–396, 2012.

Anupam Gupta and Kanat Tangwongsan. Simpler analyses of local search algorithms for facility location. *arXiv preprint arXiv:0809.2554*, 2008.

Sariel Har-Peled. Clustering motion. In *42nd Annual Symposium on Foundations of Computer Science, FOCS 2001, 14-17 October 2001, Las Vegas, Nevada, USA*, pages 84–93. IEEE Computer Society, 2001. doi: 10.1109/SFCS.2001.959883. URL https://doi.org/10.1109/SFCS.2001.959883.

Sariel Har-Peled and Soham Mazumdar. On coresets for k-means and k-median clustering. In *Proceedings of the thirty-sixth annual ACM symposium on Theory of computing*, pages 291–300, 2004.

Sariel Har-Peled and Manor Mendel. Fast construction of nets in low dimensional metrics, and their applications. In *Proceedings of the twenty-first annual symposium on Computational geometry*, pages 150–158, 2005.

Dorit S. Hochbaum and David B. Shmoys. A unified approach to approximation algorithms for bottleneck problems. *J. ACM*, 33(3):533–550, 1986. doi: 10.1145/5925.5933. URL https://doi.org/10.1145/5925.5933.

Max Hopkins, Daniel Kane, Shachar Lovett, and Gaurav Mahajan. Noise-tolerant, reliable active classification with comparison queries. In *Conference on Learning Theory*, pages 1957–2006. PMLR, 2020.

Wasim Huleihel, Arya Mazumdar, Muriel Médard, and Soumyabrata Pal. Same-cluster querying for overlapping clusters. In *Advances in Neural Information Processing Systems*, pages 10485–10495, 2019.

Daniel P Huttenlocher, Gary A Klanderman, and William J Rucklidge. Comparing images using the hausdorff distance. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(9): 850–863, 1993.

Christina Ilvento. Metric learning for individual fairness. In *1st Symposium on Foundations of Responsible Computing (FORC 2020)*, 2020.

Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pages 604–613, 1998.

Tapas Kanungo, David M Mount, Nathan S Netanyahu, Christine D Piatko, Ruth Silverman, and Angela Y Wu. A local search approximation algorithm for k-means clustering. In *Proceedings of the eighteenth annual symposium on Computational geometry*, pages 10–18, 2002.

Ehsan Kazemi, Lin Chen, Sanjoy Dasgupta, and Amin Karbasi. Comparison based learning from weak oracles. In *International Conference on Artificial Intelligence and Statistics*, pages 1849–1858. PMLR, 2018.

Taewan Kim and Joydeep Ghosh. Relaxed oracles for semi-supervised clustering. *arXiv preprint arXiv:1711.07433*, 2017a.

Taewan Kim and Joydeep Ghosh. Semi-supervised active clustering with weak oracles. *arXiv preprint arXiv:1709.03202*, 2017b.

Rolf Klein, Rainer Penninger, Christian Sohler, and David P Woodruff. Tolerant algorithms. In *European Symposium on Algorithms*, pages 736–747. Springer, 2011.

Ngai Meng Kou, Yan Li, Hao Wang, Leong Hou U, and Zhiguo Gong. Crowdsourced top-k queries by confidence-aware pairwise judgments. In *Proceedings of the 2017 ACM International Conference on Management of Data*, pages 1415–1430, 2017.

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. ISBN 089871558X.

Robert Krauthgamer and James R Lee. The black-box complexity of nearest-neighbor search. *Theoretical Computer Science*, 348(2-3):262–276, 2005.

Joseph B Kruskal. On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical society*, 7(1):48–50, 1956.

Arya Mazumdar and Barna Saha. Clustering with noisy queries. In *Advances in Neural Information Processing Systems*, pages 5788–5799, 2017a.

Arya Mazumdar and Barna Saha. Query complexity of clustering with side information. In *Advances in Neural Information Processing Systems*, pages 4682–4693, 2017b.

C.J.H. McDiarmid and R.B. Hayward. Large deviations for quicksort. *J. Algorithms*, 21(3): 476–507, November 1996. ISSN 0196-6774. doi: 10.1006/jagm.1996.0055. URL https://doi.org/10.1006/jagm.1996.0055.

Ramgopal R. Mettu and C. Greg Plaxton. Optimal time bounds for approximate clustering. In Adnan Darwiche and Nir Friedman, editors, *UAI '02, Proceedings of the 18th Conference in Uncertainty in Artificial Intelligence*, pages 344–351. Morgan Kaufmann, 2002.

Nikolaos Nakis, Niels Raunkjær Holm, Andreas Lyhne Fiehn, and Morten Mørup. How low can you go? searching for the intrinsic dimensionality of complex networks using metric node embeddings. In *International Conference on Learning Representations (ICLR)*, 2025. URL https://openreview.net/forum?id=V71ITh2w40.

T.S.E. Ng and Hui Zhang. Predicting internet network distance with coordinates-based approaches. In *Proceedings.Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies*, volume 1, pages 170–179 vol.1, 2002. doi: 10.1109/INFCOM.2002.1019258.

Gabriel Peyré and Marco Cuturi. Computational optimal transport: With applications to data science. *Foundations and Trends in Machine Learning*, 11(5-6):355–607, 2019.

Vassilis Polychronopoulos, Luca De Alfaro, James Davis, Hector Garcia-Molina, and Neoklis Polyzotis. Human-powered top-k lists. In *WebDB*, pages 25–30, 2013.

Zhen Qin, Rolf Jagerman, Kai Hui, Honglei Zhuang, Junru Wu, Le Yan, Jiaming Shen, Tianqi Liu, Jialu Liu, Donald Metzler, Xuanhui Wang, and Michael Bendersky. Large language models are effective text rankers with pairwise ranking prompting. In *Findings of the Association for Computational Linguistics: NAACL 2024*, pages 1504–1518. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.findings-naacl.97. URL https://aclanthology.org/2024.findings-naacl.97/.

Yossi Rubner, Carlo Tomasi, and Leonidas J Guibas. The earth mover's distance as a metric for image retrieval. *International Journal of Computer Vision*, 40(2):99–121, 2000.

Sandeep Silwal, Sara Ahmadian, Andrew Nystrom, Andrew McCallum, Deepak Ramachandran, and Mehran Kazemi. Kwikbucks: Correlation clustering with cheap-weak and expensive-strong signals. In *Proceedings of The Fourth Workshop on Simple and Efficient Natural Language Processing (SustaiNLP)*, pages 1–31, 2023.

Kunal Talwar. Bypassing the embedding: approximation schemes and compact representations for growth restricted metrics. In *36th ACM Symp. on Theory of Computing (STOC)*, pages 281–290, 2004.

Omer Tamuz, Ce Liu, Serge Belongie, Ohad Shamir, and Adam Tauman Kalai. Adaptively learning the crowd kernel. In *Proceedings of the 28th International Conference on International Conference on Machine Learning*, pages 673–680, 2011.

Joshua B. Tenenbaum, Vin de Silva, and John C. Langford. A global geometric framework for nonlinear dimensionality reduction. *Science*, 290(5500):2319–2323, 2000. doi: 10.1126/science.290.5500.2319. URL https://www.science.org/doi/abs/10.1126/science.290.5500.2319.

Antti Ukkonen. Crowdsourced correlation clustering with relative distance comparisons. In *2017 IEEE International Conference on Data Mining (ICDM)*, pages 1117–1122. IEEE, 2017.

V. N. Vapnik and A. Ya. Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probability & Its Applications*, 16(2):264–280, 1971. doi: 10.1137/1116025. URL https://doi.org/10.1137/1116025.

Petros Venetis, Hector Garcia-Molina, Kerui Huang, and Neoklis Polyzotis. Max algorithms in crowdsourcing environments. In *Proceedings of the 21st international conference on World Wide Web*, pages 989–998, 2012.

Victor Verdugo. Skyline computation with noisy comparisons. In *Combinatorial Algorithms: 31st International Workshop, IWOCA 2020, Bordeaux, France, June 8–10, 2020, Proceedings*, page 289. Springer.

Jiang Wang, Yang Song, Thomas Leung, Chuck Rosenberg, Jingbin Wang, James Philbin, Bo Chen, and Ying Wu. Learning fine-grained image similarity with deep ranking. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1386–1393, 2014.

Haike Xu, Sandeep Silwal, and Piotr Indyk. A bi-metric framework for fast similarity search. *arXiv preprint arXiv:2406.02891*, 2024.

Zi Yin and Yuanyuan Shen. On the dimensionality of word embedding. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2018.

Contents of Appendix

A Preliminaries **22**
A.1 Notation and Basic Definitions . . . 22
A.2 Ordering sets of edges . . . 23
A.2.1 By Dislocation . . . 23
A.2.2 By Scale . . . 25

B k-center OracleCluster problem **26**
B.1 Approximate Nearest Neighbors on Heavy Vertices . . . 26
B.1.1 Emulating the adversarial quadruplet oracle using probabilistic queries . . . 27
B.1.2 The procedure HEAVY-ANN . . . 28
B.2 The procedure BASIC-CENTER . . . 32
B.3 The procedure ORACLE-CENTER . . . 33
B.3.1 Analysis of ORACLE-CENTER . . . 35
B.4 Extension to general metric spaces . . . 40

C Approximate Distance Data Structure **41**
C.1 Structure . . . 41
C.2 The procedure CONSTRUCT-NET . . . 42
C.3 Querying edge weights . . . 45

D k-median and k-means OracleCluster problems **46**
D.1 The procedure ORACLE-MEDIAN . . . 47
D.1.1 Analysis of ORACLE-MEDIAN . . . 49
D.2 Extension to general metric spaces . . . 51

E Experiments **51**

Appendix A. Preliminaries

A.1. Notation and Basic Definitions

Metric Spaces. Let $\Sigma = (\mathcal{V}, d)$ be a metric space, where $d : \mathcal{V} \times \mathcal{V} \to \mathbb{R}_{\geq 0}$ is the *distance function*. The *size* of Σ refers to the cardinality of $\mathcal{V}$. In this paper, we work exclusively with finite metric spaces and assume that $|\mathcal{V}| = n$ throughout.

For a point $v \in \mathcal{V}$ and radius $\rho \in \mathbb{R}_{\geq 0}$, the *ball* centered at v with radius ρ is defined as $\mathfrak{B}(v, \rho) = \{u \in \mathcal{V} \mid d(v, u) \leq \rho\}$. The *doubling dimension* of Σ, denoted $\dim(\Sigma)$, is the smallest constant such that every ball of radius $\rho \in \mathbb{R}_{\geq 0}$ can be covered by at most $2^{\dim(\Sigma)}$ balls of radius $\rho/2$. A metric space Σ is said to have *bounded doubling dimension* if $\dim(\Sigma) \leq C$ for some fixed constant C.

For any subset $\mathcal{U} \subseteq \mathcal{V}$, the *diameter* of $\mathcal{U}$ is defined as $\mathsf{diam}(\mathcal{U}) = \max_{u,v \in \mathcal{U}} d(u, v)$. Similarly, the *separation* of $\mathcal{U}$ is given by $\mathsf{sep}(\mathcal{U}) = \min_{\substack{u,v \in \mathcal{U}, \\ u \neq v}} d(u, v)$. The *spread*, also referred to as the *aspect ratio* of Σ, is defined as $\$ = \mathsf{diam}(\mathcal{V})/\mathsf{sep}(\mathcal{V})$. A finite metric space Σ is said to have *bounded spread* if $\$ \leq \mathsf{poly}(n)$, where $\mathsf{poly}(\cdot)$ denotes a fixed polynomial.

Notation. Any finite metric space can be represented as a weighted complete graph. We denote the graph representation of Σ by $G_\Sigma = (\mathcal{V}, \mathcal{E}, d)$, where $\mathcal{V}$ is the set of *vertices*, $\mathcal{E} = \big\{ \{u, v\} \mid u, v \in \mathcal{V},\ u \neq v \big\}$ is the set of *edges*, and each edge is weighted by the distance function d. Throughout this paper, we extensively use graph terminology.

For subsets $\mathcal{U}, \mathcal{W} \subseteq \mathcal{V}$, we denote the set of edges between $\mathcal{U}$ and $\mathcal{W}$ by $\mathcal{E}(\mathcal{U}, \mathcal{W}) = \{\{u, w\} \mid u \in \mathcal{U},\ w \in \mathcal{W},\ u \neq w\}$. In particular, we have $\mathcal{E} = \mathcal{E}(\mathcal{V}, \mathcal{V})$. For an edge $\mathtt{e} = \{u, v\} \in \mathcal{E}$, we use $d(\mathtt{e})$ to denote $d(u, v)$.

For any $\rho \in \mathbb{R}_{\geq 0}$ and a set of edges $\mathcal{X} \subseteq \mathcal{E}$, we define $\mathcal{X}^{(\rho)} = \{\mathtt{e} \in \mathcal{X} \mid d(\mathtt{e}) \leq \rho\}$ as the subset with weight at most ρ. Given a subset $\mathcal{U} \subseteq \mathcal{V}$ and a radius $\rho \in \mathbb{R}_{\geq 0}$, we define $\mathfrak{B}(\mathcal{U}, \rho) = \bigcup_{u \in \mathcal{U}} \mathfrak{B}(u, \rho)$. For a vertex $v \in \mathcal{V}$, the distance to a subset $\mathcal{U}$ is given by $d(v, \mathcal{U}) = \min_{u \in \mathcal{U}} d(v, u)$. More generally, for subsets $\mathcal{U}, \mathcal{W} \subseteq \mathcal{V}$, we define $d(\mathcal{U}, \mathcal{W}) = \min_{u \in \mathcal{U}, w \in \mathcal{W}} d(u, w)$.

Net. For subsets $\mathcal{U}, \mathcal{W} \subseteq \mathcal{V}$, we say that $\mathcal{U}$ is an α-*cover* of $\mathcal{W}$ if for all $w \in \mathcal{W}$, there exists $u \in \mathcal{U}$ such that $d(w, u) \leq \alpha$. The set $\mathcal{U}$ is said to be β-*separated* if for any two distinct points $u, u' \in \mathcal{U}$, we have $d(u, u') > \beta$. A set $\mathcal{U}$ is called an (α, β)-*net* of $\mathcal{W}$ if it is both an α-cover of $\mathcal{W}$ and β-separated.

k-Clustering. For two subsets of vertices $\mathcal{U}, \mathcal{W} \subseteq \mathcal{V}$, let $F_{\mathcal{U}}(\mathcal{W})$ denote a clustering cost function that measures the cost of selecting centers $\mathcal{U}$ with respect to the vertices in $\mathcal{W}$. Let

$$\mathsf{OPT}^F(\mathcal{W}) = \min_{\mathcal{U} \subseteq \mathcal{V},\, |\mathcal{U}| = k} F_{\mathcal{U}}(\mathcal{W})$$

be the cost of the optimum clustering with respect to $\mathcal{W}$.

We consider the three most common clustering cost functions: the k-center, k-median, and k-means costs, defined respectively as

$$\textsc{Cost}^{\infty}_{\mathcal{U}}(\mathcal{W}) = \max_{w \in \mathcal{W}} d(w, \mathcal{U}), \quad \textsc{Cost}^{1}_{\mathcal{U}}(\mathcal{W}) = \sum_{w \in \mathcal{W}} d(w, \mathcal{U}), \quad \text{and} \quad \textsc{Cost}^{2}_{\mathcal{U}}(\mathcal{W}) = \sum_{w \in \mathcal{W}} d^2(w, \mathcal{U}).$$

We denote the costs of the optimal k-center, k-median, and k-means clustering with respect to $\mathcal{W}$ as $\mathsf{OPT}^{\infty}(\mathcal{W})$, $\mathsf{OPT}^{1}(\mathcal{W})$, and $\mathsf{OPT}^{2}(\mathcal{W})$, respectively. If $\mathcal{W} = \mathcal{V}$, we simply write OPT^{∞}, OPT^{1}, OPT^{2}.

A.2. Ordering sets of edges

For any edge set $\mathcal{X} \subseteq \mathcal{E}$, we always use $\pi_{\mathcal{X}}$ to denote an *ordered sequence* over $\mathcal{X}$, when appropriate, $\pi_{\mathcal{X}}$ may also be regarded as a set. Our algorithms will often need to (approximately) sort a set of edges (by distance) using the quadruplet oracle. We will use two notions of approximate sorting.

A.2.1. BY DISLOCATION

For an edge $e \in \mathcal{X}$, let $\text{RANK}_{\mathcal{X}}(e)$ denote the position of e among the edges in $\mathcal{X}$ in perfectly sorted order (by distance). For an ordered sequence of edges $\pi_{\mathcal{X}}$, and an edge $e \in \mathcal{X}$, $\text{RANK}_{\pi_{\mathcal{X}}}(e)$ denotes the index of e in the sequence $\pi_{\mathcal{X}}$. The *dislocation* of an edge $e \in \mathcal{X}$ with respect to the ordering $\pi_{\mathcal{X}}$ is defined as $|\text{RANK}_{\pi_{\mathcal{X}}}(e) - \text{RANK}_{\mathcal{X}}(e)|$. A sequence of edges $\pi_{\mathcal{X}}$ has a *maximum dislocation* $\mathcal{D}$ if, for every $e \in \mathcal{X}$, the condition $|\text{RANK}_{\pi_{\mathcal{X}}}(e) - \text{RANK}_{\mathcal{X}}(e)| \leq \mathcal{D}$ holds, where $\mathcal{D}$ is an integer. Observe that if $\pi_{\mathcal{X}}$ has maximum dislocation $\mathcal{D}$, for any subsequence $\pi \sqsubseteq \pi_{\mathcal{X}}$, the maximum dislocation of π is bounded by $\mathcal{D}$.

Dislocation bound. For any finite metric $\Sigma = (\mathcal{V}, d)$, define $\mathcal{D}_\Sigma := c \log |\mathcal{V}|$, where $c > 1$ is a fixed constant (independent of Σ)

Low-dislocation ordering. Let $\mathcal{X} \subseteq \mathcal{E}$ be any set of edges, and let $\pi_{\mathcal{X}}$ be an ordering of $\mathcal{X}$. We say $\pi_{\mathcal{X}}$ is a *low-dislocation ordering* if its maximum dislocation is at most $\mathcal{D}_\Sigma$.

Low-dislocation decomposition. Let $\mathcal{X} \subseteq \mathcal{E}$ and let $\mathcal{X}_1, \ldots, \mathcal{X}_m$ be a partition of $\mathcal{X}$. Suppose for each i we have a low-dislocation ordering $\pi_{\mathcal{X}_i}$. The collection

$$\Pi = \{\pi_{\mathcal{X}_1}, \ldots, \pi_{\mathcal{X}_m}\}$$

is called a *low-dislocation decomposition* of $\mathcal{X}$.

We will use the following lemma to efficiently compute a low-dislocation ordering.

Lemma 3 *Let $\Sigma = (\mathcal{V}, d)$ be a metric with size $|\mathcal{V}| = n$, and let $\mathcal{E} = \mathcal{E}(\mathcal{V}, \mathcal{V})$ be the set of all edges. Let $\tilde{\mathcal{Q}} : \mathcal{E} \times \mathcal{E} \to \{\text{YES}, \text{NO}\}$ be a probabilistic quadruplet oracle with noise $\varphi \in \left[0, \frac{1}{16}\right]$. There exists an algorithm* PROB-SORT *such that, for any subset $\mathcal{X} \subseteq \mathcal{E}$ of m edges, with probability $1 - \frac{1}{n^{O(1)}}$, $\text{PROB-SORT}(\mathcal{X})$ returns a low dislocation ordering $\pi_{\mathcal{X}}$, while using $O(\max(m, n) \log n)$ queries to $\tilde{\mathcal{Q}}$.*

Proof Theorem 9 of Geissmann et al. (2019) gives an algorithm RIFFLE-SORT that given a set of objects of size n and a noisy comparison oracle that for each pair of objects decides which is larger, independently and persistently, with error probability at most $\varphi \in [0, 1/16]$, can order them such that the maximum dislocation of any element is at most $c \log n$ for some constant c, with probability $1 - \frac{1}{n^{O(1)}}$. This requires $O(n \log n)$ comparison oracle queries. The comparison oracle here is equivalent to the quadruplet oracle when each object is an edge.

Thus, given a set $\mathcal{X} \subseteq \mathcal{E}$ of size m , RIFFLE-SORT gives an ordering with max dislocation $c \log m$ with probability $1 - 1/m^{O(1)}$. Since $\mathcal{X}$ is a set of edges, $|\mathcal{X}| = m < n^2$. Therefore, $c \log m \leq 2c \log n$. Recall that by definition $\mathcal{D}_\Sigma = O(\log n)$. We now fix the hidden constant so that $\mathcal{D}_\Sigma \geq 2c \log n$, where c is the constant in the RIFFLE-SORT guarantee. With this choice, the ordering output by RIFFLE-SORT has dislocation at most $\mathcal{D}_\Sigma$ with probability at least $1 - 1/m^{O(1)}$.

Since we want a high probability guarantee in terms of n, we do the following: If $m < n$, we augment the set with $n - m$ artificial elements so that the total number of objects is at least n. These

artificial elements can be assumed to have ranks larger than other edges, and we can assume that all queries involving these artificial elements are answered accurately and consistently. Now applying Theorem 9, we get that given m edges, we can obtain a low-dislocation ordering with probability $1 - 1/n^{O(1)}$ requiring at most $O(\max(m, n) \log n)$ quadruplet oracle queries. ■

Given a low-dislocation ordering, we will also need to perform the following operations: (i) filter out edges with weight below a certain threshold, and (ii) partition the edges into buckets based on weight.

Lemma 4 *Let $\Sigma = (\mathcal{V}, d)$ be a metric space with size $|\mathcal{V}| = n$. Let $\mathcal{X} \subseteq \mathcal{E}(\mathcal{V}, \mathcal{V})$ be a subset of m edges and let $\pi_{\mathcal{X}}$ be a a low-dislocation ordering of $\mathcal{X}$.*

1. *There exists a procedure* FILTER *such that, given $\pi_{\mathcal{X}}$ and $\rho \geq 0$,* FILTER$(\pi_{\mathcal{X}}, \rho)$ *returns the subset $\mathcal{X}^{(\rho)} \subseteq \mathcal{X}$ using $O(\log^2 n)$ distance queries.*

2. *There exists a procedure* BUCKET *such that given $a, b \in \mathbb{Z}_{\geq 0}$ with $a \leq b$ and $\sigma \geq 1$,* BUCKET$(\pi_{\mathcal{X}}, a, b, \sigma)$ *returns the following set of pairs*

$$\left\{ \left(\mathfrak{e}, \sigma^i\right) \,\middle|\, \mathfrak{e} \in \mathcal{X} \text{ and } i = \min\{j \in \{a, \ldots, b\} \mid d(\mathfrak{e}) \leq \sigma^j\} \right\}.$$

   using $O((b - a) \cdot \log^2 n)$ distance queries.

Proof Since $\pi_{\mathcal{X}}$ is a low–dislocation ordering, its maximum dislocation is $\mathcal{D}_\Sigma = O(\log n)$, so for every edge $\mathfrak{e} \in \mathcal{X}$ we have $|\text{RANK}_{\mathcal{X}}(\mathfrak{e}) - \text{RANK}_{\pi_{\mathcal{X}}}(\mathfrak{e})| \leq \mathcal{D}_\Sigma$.

The routine FILTER$(\pi_{\mathcal{X}}, \rho)$ performs a modified binary search. Fix an index i under consideration and let $\mathfrak{e}$ be the unique edge with $\text{RANK}_{\mathcal{X}}(\mathfrak{e}) = i$. Define the *search window*

$$W_i = [\, i - \mathcal{D}_\Sigma,\, i + \mathcal{D}_\Sigma \,] \cap [\, 1, n \,].$$

Because of the dislocation bound, $\mathfrak{e}$ lies somewhere in W_i. Moreover, for any $\mathfrak{e}' \in \mathcal{X}$ with $\text{RANK}_{\pi_{\mathcal{X}}}(\mathfrak{e}') < \min W_i$ (resp. $> \max W_i$) we have $d(\mathfrak{e}') < d(\mathfrak{e})$ (resp. $> d(\mathfrak{e})$). There are exactly $(i-1)-(\min W_i - 1) = i - \min W_i$ edges in the subsequence $\pi_{\mathcal{X}}[\min W_i, \ldots, \max W_i]$ whose distance is $< d(\mathfrak{e})$ therefore the $(i - \min W_i + 1)$-th smallest edge by weight in $\pi_{\mathcal{X}}[\min W_i, \ldots, \max W_i]$ is precisely $\mathfrak{e}$.

FILTER then compares ρ with $d(\mathfrak{e})$ and proceeds as in standard binary search. Note that since, weights of edges in the window are known we can identify the ones with weight less than ρ and recurse. Each iteration inspects $O(\mathcal{D}_\Sigma) = O(\log n)$ edges and eliminates a constant fraction of the sequence, so the search completes in $O(\log n)$ steps. The total number of distance-oracle calls is $O(\mathcal{D}_\Sigma \log n) = O(\log^2 n)$, and no quadruplet queries are required.

It is straightforward to see that BUCKET$(\pi_{\mathcal{X}}, a, b, \sigma)$ can be computed using $(b - a + 1)$ invocations of FILTER with $\rho = \sigma^a, \sigma^{a+1}, \ldots, \sigma^b$.

■

A.2.2. BY SCALE

Although we primarily work with the probabilistic quadruplet oracle, some of our algorithms require a procedure to order edges using the adversarial quadruplet oracle. In such cases, we use an alternate notion of approximate sorting. For $\alpha \geq 1$, $\pi_{\mathcal{X}}$ is *α-sorted* if for any pair $i, j \in [m]$ such that $i < j$, it holds that $d(\pi_{\mathcal{X}}[i]) \leq \alpha \cdot d(\pi_{\mathcal{X}}[j])$. We use the following lemma to obtain $O(1)$-sorted sequences.

Lemma 5 *Let $\Sigma = (\mathcal{V}, d)$ be a metric space with size $|\mathcal{V}| = n$, and let $\mathcal{E} = \mathcal{E}(\mathcal{V}, \mathcal{V})$ be the set of all edges. Let $\mathcal{Q} : \mathcal{E} \times \mathcal{E} \to \{0, 1\}$ be an adversarial quadruplet oracle with noise $\mu \geq 0$. There exists an algorithm* ADV-SORT *such that, for any subset $\mathcal{X} \subseteq \mathcal{E}$ of m edges, with probability $1 - \frac{1}{n^{O(1)}}$, ADV-SORT$(\mathcal{X})$ outputs a $(1+\mu)^2$-sorted sequence $\pi_{\mathcal{X}}$ using $O(m \cdot \mathsf{polylog}\, n)$ queries to $\mathcal{Q}$.*

Proof Let QUICK-SORT$(\cdot)$ denote an implementation of the classic randomized Quick Sort algorithm that recursively partitions the input around a randomly chosen pivot and then independently sorts each resulting subsequence. Let $T \colon \mathbb{Z}_{\geq 0} \to \mathbb{Z}_{\geq 0}$ be a function defined by $T(0) = 0, T(1) = 0$, and for $m \geq 2$, $T(m) = 2\,(m+1)\,\mathbb{H}_m - 4m$, where $\mathbb{H}_m = \sum_{i=1}^{m} \frac{1}{i}$ denotes the m-th harmonic number. Note that $\mathbb{H}_m = O(\log m)$ for $m \geq 1$.

The algorithm ADV-SORT$(\cdot)$ is essentially an implementation of QUICK-SORT$(\mathcal{X})$ with a restart strategy; that is, given $\mathcal{X}$, run QUICK-SORT$(\mathcal{X})$ and monitor its number of steps. If it exceeds $T(m)$, abort and restart. This is repeated $O(\log n)$ times.

The proof proceeds similarly to the analysis in Acharya et al. (2018), which deals with adversarial comparison oracles subject to additive error. Proving correctness is straightforward. Let $\pi_{\mathcal{X}} =$ QUICK-SORT$(\mathcal{X})$. Consider two arbitrary edges $\mathtt{e}_1, \mathtt{e}_2 \in \mathcal{X}$. If $\textsc{rank}_{\pi_{\mathcal{X}}}(\mathtt{e}_1) \leq \textsc{rank}_{\pi_{\mathcal{X}}}(\mathtt{e}_2)$, then there exists a pivot $\mathtt{e}_3$ such that $\mathcal{Q}(\mathtt{e}_1, \mathtt{e}_3) =$ YES and $\mathcal{Q}(\mathtt{e}_2, \mathtt{e}_3) =$ NO. It follows that $d(\mathtt{e}_1) \leq (1+\mu)^2\, d(\mathtt{e}_2)$, so $\pi_{\mathcal{X}}$ is $(1+\mu)^2$-sorted.

We now bound the number of oracle queries. For a fixed adversarial quadruplet oracle $\mathcal{Q}$, for any $\mathcal{Y} \subseteq \mathcal{E}$, let $\mathbb{E}(\mathcal{Y})$ denote the expected number of queries made by QUICK-SORT$(\mathcal{Y})$. Define

$$\Psi(m) = \max_{\substack{\mathcal{Y} \subseteq \mathcal{E} \\ |\mathcal{Y}| \leq m}} \mathbb{E}(\mathcal{Y}).$$

Thus, $\Psi(m)$ upper bounds the expected query complexity of QUICK-SORT on any subset of m edges (and in particular on $\mathcal{X}$). We prove by strong induction on m that $\Psi(m) \leq T(m)$.

We proceed by bounding $\mathbb{E}(\mathcal{Y})$ for an arbitrary subset $\mathcal{Y} \subseteq \mathcal{E}$ of size m. For each $\mathtt{e} \in \mathcal{Y}$, define

$$\mathbb{L}(\mathtt{e}) = \{\mathtt{e}' \in \mathcal{Y} \mid \mathcal{Q}(\mathtt{e}', \mathtt{e}) = \textsc{Yes}\} \quad \text{and} \quad \mathbb{M}(\mathtt{e}) = \{\mathtt{e}' \in \mathcal{Y} \mid \mathcal{Q}(\mathtt{e}, \mathtt{e}') = \textsc{No}\}.$$

Consider the following set of inequalities,

$$\begin{aligned}
\mathbb{E}(\mathcal{Y}) &\leq (m-1) + \tfrac{1}{m} \sum_{\mathtt{e} \in \mathcal{Y}} \big(\mathbb{E}(\mathbb{L}(\mathtt{e})) + \mathbb{E}(\mathbb{M}(\mathtt{e}))\big) \\
&\leq (m-1) + \tfrac{1}{m} \sum_{\mathtt{e} \in \mathcal{Y}} \big(\Psi(|\mathbb{L}(\mathtt{e})|) + \Psi(|\mathbb{M}(\mathtt{e})|)\big) \\
&\leq (m-1) + \tfrac{1}{m} \sum_{\mathtt{e} \in \mathcal{Y}} \big(T(|\mathbb{L}(\mathtt{e})|) + T(|\mathbb{M}(\mathtt{e})|)\big).
\end{aligned}$$

The first inequality comes from the standard analysis of Quick Sort. The second substitutes the definition of Ψ, and the third applies the induction hypothesis.

Observe that for every $\mathfrak{e} \in \mathcal{Y}$ we have $0 \leq |\mathbb{L}(\mathfrak{e})|, |\mathbb{M}(\mathfrak{e})| \leq m-1$ with $|\mathbb{L}(\mathfrak{e})|+|\mathbb{M}(\mathfrak{e})| = m-1$, and that $\sum_{\mathfrak{e}\in\mathcal{Y}} |\mathbb{L}(\mathfrak{e})| = \sum_{\mathfrak{e}\in\mathcal{Y}} |\mathbb{M}(\mathfrak{e})| = \binom{m}{2}$. Therefore, we can bound $\mathbb{E}(\mathcal{Y})$ by the optimal value of the following convex optimization problem:

$$\mathbb{E}(\mathcal{Y}) \;\leq\; \max_{\substack{0\leq x_{\mathfrak{e}}\leq m-1 \\ \sum_{\mathfrak{e}\in\mathcal{Y}} x_{\mathfrak{e}} = \binom{m}{2}}} \Big((m-1) + \frac{1}{m}\sum_{\mathfrak{e}\in\mathcal{Y}} \big(T(x_{\mathfrak{e}}) + T(m-1-x_{\mathfrak{e}})\big)\Big).$$

By convexity of $T(\cdot)$, the maximum must occur at a corner, corresponding to a permutation of $\{0, \ldots, m-1\}$. This implies

$$\mathbb{E}(\mathcal{Y}) \leq (m-1) + \frac{1}{m}\sum_{i=0}^{m-1} \Big(T(i) + T(m-i)\Big).$$

Since $\mathcal{Y} \subseteq \mathcal{E}$ was arbitrary, we have $\Psi(m) \leq (m-1) + \frac{1}{m}\sum_{i=0}^{m-1}\Big(T(i)+T(m-i)\Big)$. This is well known as the classic *Quick Sort recurrence* (see McDiarmid and Hayward (1996)) and is solved by $T(m)$ as defined above. This completes the induction.

By Markov's inequality, the probability that QUICK-SORT$(\mathcal{X})$ takes more than, say, $100 \cdot T(m)$ steps is at most $\frac{1}{100}$. Thus, with high probability, ADV-SORT$(\cdot)$ terminates after $O(\log n)$ restarts, resulting in a total cost of $O(m \operatorname{polylog} n)$ quadruplet oracle queries.

■

Appendix B. k-center OracleCluster problem

In this section, we design an $O(1)$-approximation algorithm for the k-center problem. For metric spaces with bounded doubling dimension, our algorithm uses $O((n+k^2)\operatorname{polylog} n)$ quadruplet queries and $O(\operatorname{polylog} n)$ distance queries. For general metrics, the algorithm requires $O(nk \operatorname{polylog} n)$ quadruplet queries and $O(\operatorname{polylog} n)$ distance queries. We refer the reader to Section 3.1 for a detailed overview of our approach. Our presentation focuses on the case where the metric has bounded doubling dimension. We begin by introducing two key subroutines, HEAVY-ANN$(\cdot)$ and BASIC-CENTER$(\cdot)$, in Sections B.1 and B.2, respectively. These procedures are essential building blocks for our main algorithm. In Section B.3, we present our full algorithm for metrics with bounded doubling dimension. Finally, in Section B.4, we describe how to extend our approach to general metrics.

B.1. Approximate Nearest Neighbors on Heavy Vertices

Let $\Sigma = (\mathcal{V}, d)$ be a finite metric space of size $|\mathcal{V}| = n$. Let $\mathcal{H}, \text{CORE} \subseteq \mathcal{V}$ and $\rho \in \mathbb{R}_{>0}$ be a positive real number. Let $\Delta_{\text{HA}} = \omega_1 \cdot \log n$, where $\omega_1 > 1$ is a constant.

We say $\mathcal{H}$ is a $(\rho, 2\rho)$-*heavy net* of CORE if the following conditions hold:

(i) *Net*: $\mathcal{H}$ is a $(\rho, 2\rho)$-net of CORE.

(ii) *Heavy*: For every $h \in \mathcal{H}$, there exists a subset $\text{CORE}(h) \subseteq \mathfrak{B}(h, \rho) \cap \text{CORE}$ of size Δ_{HA}. We also assume $\text{CORE}(h)$ is explicitly identified for every $h \in \mathcal{H}$.

In this section, we prove the following lemma.

Lemma 6 *Let $\Sigma = (\mathcal{V}, d)$ be a finite metric space of size $|\mathcal{V}| = n$ with bounded doubling dimension. Let $\mathcal{H}, \text{CORE}, \mathcal{V}' \subseteq \mathcal{V}$ be pairwise disjoint, and let $\rho > 0$ be a parameter such that $\mathcal{H}$ forms a $(\rho, 2\rho)$-heavy net of* CORE. *There exists a procedure* HEAVY-ANN *such that, with probability at least $1 - \frac{1}{n^{\Omega(1)}}$, $\text{HEAVY-ANN}(\mathcal{H}, \text{CORE}, \mathcal{V}')$ outputs a set $\mathcal{N} \subseteq \mathcal{E}(\mathcal{H}, \mathcal{V}')$ of size $O(|\mathcal{V}'| \log n)$ satisfying the following property:*

For every $v \in \mathcal{V}'$, there exists an edge $e \in \mathcal{N} \cap \mathcal{E}(v, \mathcal{H})$ such that $d(e) \leq C_{HA} \cdot d(v, \mathcal{H})$, where $C_{HA} \geq 1$ is a constant. Moreover, the procedure makes $O((n + |\mathcal{H}|^2)\,\mathsf{polylog}\,(n))$ queries to the probabilistic quadruplet oracle and no queries to the distance oracle.

We prove the lemma in two steps. First, in Section B.1.1, we design a tester that can emulate an adversarial quadruplet oracle on a *rich* subset of queries $\mathcal{J} \subseteq \mathcal{E}(\mathcal{V}, \mathcal{H}) \times \mathcal{E}(\mathcal{V}, \mathcal{H})$ using a small number of calls to the probabilistic quadruplet oracle. Next, in Section B.1.2, we use the emulated adversarial oracle to design a data structure on $\mathcal{H}$ for computing the desired edge set $\mathcal{N} \subseteq \mathcal{E}(\mathcal{H}, \mathcal{V}')$.

B.1.1. EMULATING THE ADVERSARIAL QUADRUPLET ORACLE USING PROBABILISTIC QUERIES

Our goal is to design a *tester* (a function)

$$\hat{\mathcal{Q}} : \mathcal{E}(\mathcal{V}' \cup \mathcal{H}, \mathcal{H}) \times \mathcal{E}(\mathcal{H}, \mathcal{H}) \to \{\text{YES}, \text{NO}\},$$

such that for any $v \in \mathcal{V}' \cup \mathcal{H}$ and any $h_1, h_2, h_3 \in \mathcal{H}$,

$$\hat{\mathcal{Q}}(\{v, h_1\}, \{h_2, h_3\}) = \begin{cases} \text{YES} & \text{if } d(v, h_1) < \dfrac{d(h_2, h_3)}{2}, \\ \text{NO} & \text{if } d(v, h_1) > 2 \cdot d(h_2, h_3). \end{cases} \tag{1}$$

Observe that tester $\hat{\mathcal{Q}}$ behaves consistently with an *adversarial quadruplet oracle* (see (b)), with an error tolerance of $\mu \leq 1$, on the subset of queries $\mathcal{J} = \mathcal{E}(\mathcal{V}' \cup \mathcal{H}, \mathcal{H}) \times \mathcal{E}(\mathcal{H}, \mathcal{H})$. The following procedure emulates a query to $\hat{\mathcal{Q}}$ using a small number of queries to $\tilde{\mathcal{Q}}$.

Emulating the Tester. For any query $\big(\{v, h_1\}, \{h_2, h_3\}\big) \in \mathcal{J}$, compute

$$\text{COUNT} = \sum_{c \in \text{CORE}(h_1)} \mathbb{1}\left\{\tilde{\mathcal{Q}}(\{v, c\}, \{h_2, h_3\}) = \text{YES}\right\},$$

where $\mathbb{1}\{\cdot\}$ is the indicator function. Output YES if $\text{COUNT} \geq \frac{\Delta_{\text{HA}}}{2}$; otherwise, output NO.

We prove the following lemma.

Lemma 7 *For every query $\big(\{v, h_1\}, \{h_2, h_3\}\big) \in \mathcal{E}(\mathcal{V}' \cup \mathcal{H}, \mathcal{H}) \times \mathcal{E}(\mathcal{H}, \mathcal{H})$, with probability $1 - \frac{1}{n^{\Omega(1)}}$, the emulation procedure's output satisfies conditions (1). Furthermore, evaluating a query requires $O(\mathsf{polylog}\, n)$ calls to the probabilistic quadruplet oracle $\tilde{\mathcal{Q}}$.*

Proof We consider two cases. Suppose COUNT $\geq \frac{\Delta_{\text{HA}}}{2}$. In this case, the tester outputs YES, which could only be incorrect if $d(v, h_1) > 2 \cdot d(h_2, h_3)$. However, if $d(v, h_1) > 2 \cdot d(h_2, h_3)$, then for every $c \in \text{CORE}(h_1)$, we have

$$\begin{aligned} d(v, c) &\geq d(v, h_1) - d(h_1, c) \\ &\geq d(v, h_1) - \rho \\ &> 2 \cdot d(h_2, h_3) - \rho \\ &> 2 \cdot d(h_2, h_3) - \frac{d(h_2, h_3)}{2} \quad (\text{since } 2\rho < d(h_2, h_3)) \\ &> \tfrac{3}{2} \cdot d(h_2, h_3) \\ &> d(h_2, h_3). \end{aligned}$$

Thus, in this case, every YES returned by $\tilde{\mathcal{Q}}$ must be incorrect, as $d(v, c) > d(h_2, h_3)$ for each $c \in \text{CORE}(h_1)$. However, since COUNT $\geq \frac{\Delta_{\text{HA}}}{2}$ led to a YES response, at least half of the responses by $\tilde{\mathcal{Q}}$ are YES.

Next, suppose COUNT $< \frac{\Delta_{\text{HA}}}{2}$. In this case, the tester outputs NO, and the result could only be incorrect if $d(v, h_1) < \frac{d(h_2,h_3)}{2}$. If $d(v, h_1) < \frac{d(h_2,h_3)}{2}$, then for every $c \in \text{CORE}(h_1)$, we have

$$\begin{aligned} d(v, c) &\leq d(v, h_1) + d(h_1, c) \\ &\leq d(v, h_1) + \rho \\ &< \tfrac{d(h_2,h_3)}{2} + \tfrac{d(h_2,h_3)}{2} \quad (\text{since } \rho < \tfrac{d(h_2,h_3)}{2}) \\ &< d(h_2, h_3). \end{aligned}$$

Thus, in this case, $d(v, c) < d(h_2, h_3)$, meaning every NO returned by $\tilde{\mathcal{Q}}$ must be incorrect. However, since COUNT $< \frac{\Delta_{\text{HA}}}{2}$, the majority of the responses are NO.

For any given query, Chernoff bounds imply that the probability of the majority of $\tilde{\mathcal{Q}}$ responses being incorrect is exponentially small in $\Delta_{\text{HA}} = \omega_1 \cdot \log n$. Specifically, this failure event occurs with probability at most $e^{-\Omega(\Delta_{\text{HA}})} = \frac{1}{n^{100}}$, for an appropriate choice of ω_1. ■

Since the total number of queries to $\hat{\mathcal{Q}}$ is bounded by $|\mathcal{J}| \leq n^4$, and by Lemma 7 the emulation procedure satisfies specification (1) for every query with high probability.

B.1.2. THE PROCEDURE HEAVY-ANN

We now use $\hat{\mathcal{Q}}$ from the previous section to design the procedure HEAVY-ANN$(\mathcal{H}, \mathcal{V}', \text{CORE}, \rho)$ and prove Lemma 6. Throughout this section, we assume every query to $\hat{\mathcal{Q}}$ satisfies the specification (1).

First, we show how to design a *ring-separator data structure* Krauthgamer and Lee (2004); Har-Peled and Mendel (2005) on $\mathcal{H}$ using only $\hat{\mathcal{Q}}$. Next, we show that for every vertex $v \in \mathcal{V}'$, one can query the data structure (again, using only $\hat{\mathcal{Q}}$) to obtain a set $\mathcal{N}_v \subseteq \mathcal{E}(v, \mathcal{H})$ consisting of $O(\log n)$ edges such that there exists at least one edge $e \in \mathcal{N}_v$ with $d(e) \leq C_{\text{HA}} \cdot d(v, \mathcal{H})$.

For simplicity, in the following we only focus on the asymptotic query complexity, however, individual constants and other performance parameters can also be improved.

Rank-Balls. Recall that the tester $\hat{\mathcal{Q}}$ behaves consistently with an adversarial quadruplet oracle with noise $\mu = 1$ on the subset of queries $\mathcal{E}(\mathcal{V}' \cup \mathcal{H}, \mathcal{H}) \times \mathcal{E}(\mathcal{H}, \mathcal{H})$. Hence, we can use it to apply

the routine ADV-SORT$(\cdot)$ from Lemma 5 on the set $\mathcal{E}' = \mathcal{E}(\mathcal{H}, \mathcal{H})$. Let $\pi_{\mathcal{E}'} = \text{ADV-SORT}(\mathcal{E}')$ denote the resulting 4-sorted sequence, satisfying $\frac{\pi_{\mathcal{E}'}[p]}{4} \leq \pi_{\mathcal{E}'}[q] \leq 4\pi_{\mathcal{E}'}[r]$ for all indices $p < q < r$. For a subset $\mathcal{H}^\star \subseteq \mathcal{H}$, a vertex $h \in \mathcal{H}^\star$ and an edge $\mathbb{e} \in \mathcal{E}'$, define

$$\mathfrak{B}'(h, \mathbb{e}, \mathcal{H}^\star) = \left\{ h' \in \mathcal{H}^\star \;\middle|\; \text{RANK}_{\pi_{\mathcal{E}'}}(\{h, h'\}) \leq \text{RANK}_{\pi_{\mathcal{E}'}}(\mathbb{e}) \right\}.$$

Given $\pi_{\mathcal{E}'}$, the set $\mathfrak{B}'(h, \mathbb{e}, \mathcal{H}^*)$ is computable directly by inspection, requiring no additional oracle queries.

Constructing the Tree. We first construct a tree-based data structure with fanout 2. Given a node of the tree along with a subset of vertices from $\mathcal{H}$ that belong in the current node (the entire set $\mathcal{H}$ is stored to the root of our tree-based data structure), we describe what information we store in the node and how we construct its two children. For simplicity, we describe the process for a node t of the tree. The construction follows equivalently for the entire tree by recursion.

Let t be the current node we process and let $\mathcal{H}_t \subseteq \mathcal{H}$ be the vertices stored in node t. Consider the submetric $\Sigma' = (\mathcal{H}_t, d)$. It is a standard fact that $\dim(\Sigma') \leq 2\dim(\Sigma)$. Let $\gamma = 2^{\dim(\Sigma')}$. Let $m = \frac{\bar{N}}{2\gamma^{20}}$, where $\bar{N} = |\mathcal{H}_t|$ is the number of points stored in the node.

(i) We begin by computing

$$(h, h') = \arg \min_{(h,h') \in \mathcal{H}_t \times \mathcal{H}_t} \left\{ \text{RANK}_{\pi_{\mathcal{E}'}}(\{h, h'\}) \;\middle|\; |\mathfrak{B}'(h, \{h, h'\}, \mathcal{H}_t)| = m \right\}. \tag{2}$$

In other words, (h, h') is the pair in $\mathcal{H}_t \times \mathcal{H}_t$ that minimizes $\text{RANK}_{\pi_{\mathcal{E}'}}(\{h, h'\})$ among all pairs (h, h') such that $|\mathfrak{B}'(h, \{h, h'\}, \mathcal{H}_t)| = m$. The pair (h, h') can be computed by inspection without performing any oracle query.

(ii) Set $h = v_1$ and order the points in $\mathcal{H}_t$ as $\{v_1, v_2, \ldots, v_{\bar{N}}\}$ by increasing rank in $\pi_{\mathcal{E}'}$, with $v_m = h'$. Define $j = \gamma^{20}m + 1$. By definition, $\mathfrak{B}'(v_1, \{v_1, v_j\}, \mathcal{H}_t)$ contains exactly $\gamma^{20}m + 1$ vertices.

(iii) Next, identify the edge $\mathbb{e}_2 \in \mathcal{E}(\mathcal{H}_t, \mathcal{H}_t)$ with the minimum rank in $\pi_{\mathcal{E}'}$ such that there exists a set $S \subseteq \mathcal{H}_t$ of size $|S| \leq \gamma^{10}$ satisfying,

$$\mathfrak{B}'(v_1, \{v_1, v_j\}, \mathcal{H}_t) \subseteq \bigcup_{\hat{h} \in S} \mathfrak{B}'(\hat{h}, \mathbb{e}_2, \mathcal{H}_t).$$

The edge $\mathbb{e}_2$ can also be computed without performing any oracle query. We store $v_1, v_m, v_j, \mathbb{e}_2$ in the node t for use in the search process.

(iv) Finally, partition the vertices in $\mathcal{H}_t$ into two sets, L_t and R_t, where the "left" set is defined as $L_t = \{v_1, \ldots, v_j\}$ and the "right" set as $R_t = \{v_m, v_{m+1}, \ldots, v_{\bar{N}}\}$. Construct t_L, t_R as the two children of t and recursively apply the construction to t_L with the set of heavy vertices L_t and t_R with the set of heavy vertices R_t to form a tree structure. If t is a leaf node we set $t_L = t_R = \emptyset$.

Computing the Map. To compute the desired map $\mathcal{N} \subseteq \mathcal{E}(\mathcal{V}', \mathcal{H})$, repeat the following for each $v \in \mathcal{V}'$: Initially, set $\mathcal{N}_v$ as an empty set and t as the root of the constructed tree. Then call the procedure Query$(v, \mathcal{N}_v, t)$:

(i) If $t = \emptyset$, return $\mathcal{N}_s$.

(ii) Add the edges (s, v_1), (s, v_m), and (s, v_j) to $\mathcal{N}_s$.

(iii) Query $\hat{\mathcal{Q}}(\{v_1, s\}, \mathbb{e}_2)$ in node t:

- If YES, Query$(s, \mathcal{N}_s, t_L)$.
- If NO, Query$(s, \mathcal{N}_s, t_R)$.

After considering every vertex $v \in \mathcal{V}'$, return $\mathcal{N} = \bigcup_{v \in \mathcal{V}'} \mathcal{N}_v$.

Analysis. Consider an edge $\mathbb{e} \in \mathcal{E}'$. Since $\pi_{\mathcal{E}'}$ is 4-sorted, for any $\mathbb{e}', \mathbb{e}'' \in \mathcal{E}'$ with $d(\mathbb{e}'') > 4 \cdot d(\mathbb{e})$ and $d(\mathbb{e}) > 4 \cdot d(\mathbb{e}')$, it holds that $\text{RANK}_{\pi_{\mathcal{E}'}}(\mathbb{e}'') > \text{RANK}_{\pi_{\mathcal{E}'}}(\mathbb{e}) > \text{RANK}_{\pi_{\mathcal{E}'}}(\mathbb{e}')$. Consequently, for any subset $\mathcal{H}^* \subseteq \mathcal{H}$, $h \in \mathcal{H}^*$, and $\mathbb{e} \in \mathcal{E}'$, $\mathfrak{B}(h, \frac{d(\mathbb{e})}{4}) \cap \mathcal{H}^\star \subseteq \mathfrak{B}'(h, \mathbb{e}, \mathcal{H}^*) \subseteq \mathfrak{B}(h, 4d(\mathbb{e})) \cap \mathcal{H}^\star$.

Lemma 8 *Let t be a fixed node of the constructed tree. Recall that in node t we store v_1, v_m, v_j, and $\mathbb{e}_2$. Define the following distances: $r_1 = d(v_1, v_m)$, $r_2 = d(\mathbb{e}_2)$, $r_3 = d(v_1, v_j)$. It holds that*

$$64 \cdot r_1 < r_2 \leq \frac{r_3}{64}.$$

Proof We first show the inequality $r_2 \leq \frac{r_3}{64}$. By the doubling property, any ball of radius r in $(\mathcal{H}_t, d)$ can be covered by γ^{10} balls of radius $\frac{r}{1024}$. Since $\mathfrak{B}'(v_1, \{v_1, v_j\}, \mathcal{H}_t) \subseteq \mathfrak{B}(v_1, 4r_3) \cap \mathcal{H}_t$, there exists an edge $\bar{\mathbb{e}} \in \mathcal{E}(\mathcal{H}_t, \mathcal{H}_t)$ such $\mathfrak{B}'(v_1, \{v_1, v_j\}, \mathcal{H}_t)$ can be covered by at most γ^{10} balls of radius $d(\bar{\mathbb{e}}) \leq \frac{4r_3}{1024} = \frac{r_3}{256}$. Recall that $\mathbb{e}_2 \in \mathcal{E}(\mathcal{H}_t, \mathcal{H}_t)$ is the minimum-ranked edge in $\pi_{\mathcal{E}'}$ for which there exists a set $S \subseteq \mathcal{H}_t$ with $|S| \leq \gamma^{10}$ such that $\mathfrak{B}'(v_1, \{v_1, v_j\}, \mathcal{H}_t) \subseteq \bigcup_{\hat{h} \in S} \mathfrak{B}'(\hat{h}, \mathbb{e}_2, \mathcal{H}_t)$.

Since $\pi_{\mathcal{E}'}$ is 4-sorted, it follows that

$$r_2 = d(\mathbb{e}_2) \leq 4d(\bar{\mathbb{e}}) \leq \frac{r_3}{64}.$$

Next, we show that $r_2 > 64 \cdot r_1$. We first argue that there exists an $h' \in S$ such that, every subset $S' \subseteq \mathcal{H}_t$ of size $|S'| \leq \gamma^{10}$ satisfies $\mathfrak{B}'(h', \mathbb{e}_2, \mathcal{H}_t) \not\subseteq \bigcup_{\hat{h} \in S'} \mathfrak{B}'(\hat{h}, \{v_1, v_m\}, \mathcal{H}_t)$. Assume that the argument does not hold. Then, for every $\hat{h} \in S$ there exists a subset $S_{\hat{h}} \subseteq \mathcal{H}_t$ of size $|S_{\hat{h}}| \leq \gamma^{10}$ such that $\mathfrak{B}'(\hat{h}, \mathbb{e}_2, \mathcal{H}_t) \subseteq \bigcup_{\ell \in S_{\hat{h}}} \mathfrak{B}'(\ell, \{v_1, v_m\}, \mathcal{H}_t)$. We set $S^* = \bigcup_{\hat{h} \in S} S_{\hat{h}}$ with $|S^*| \leq \gamma^{20}$. Then, it holds that $\mathfrak{B}'(v_1, \{v_1, v_j\}, \mathcal{H}_t) \subseteq \bigcup_{\hat{h} \in S} \mathfrak{B}'(\hat{h}, \mathbb{e}_2, \mathcal{H}_t) \subseteq \bigcup_{\ell \in S^*} \mathfrak{B}'(\ell, \{v_1, v_m\}, \mathcal{H}_t)$. Since $\mathfrak{B}'(v_1, \{v_1, v_j\}, \mathcal{H}_t) \subseteq \bigcup_{\ell \in S^*} \mathfrak{B}'(\ell, \{v_1, v_m\}, \mathcal{H}_t)$, $|\mathfrak{B}'(h, \{v_1, v_j\}, \mathcal{H}_t)| = \gamma^{20} m + 1$, and $|S^*| \leq \gamma^{20}$, there exists $\bar{h} \in S^*$ such that $|\mathfrak{B}'(\bar{h}, \{v_1, v_m\}, \mathcal{H}_t)| > m$. Hence, there exists $\bar{h}' \in \mathcal{H}_t$ such that $\text{RANK}_{\pi_{\mathcal{E}'}}(\{\bar{h}, \bar{h}'\}) < \text{RANK}_{\pi_{\mathcal{E}'}}(\{v_1, v_m\})$ with $|\mathfrak{B}'(\bar{h}, \{\bar{h}, \bar{h}'\}, \mathcal{H}_t)| = m$ contradicting the optimality of (2).

Consider such an $h' \in S$. By the definition of h' and the fact that $\mathfrak{B}(\hat{h}, \frac{r_1}{4}) \cap \mathcal{H}_t \subseteq \mathfrak{B}'(\hat{h}, \{v_1, v_m\}, \mathcal{H}_t)$ for every $\hat{h} \in \mathcal{H}_t$, we have that the points in $\mathfrak{B}'(h', \mathbb{e}_2, \mathcal{H}_t)$ cannot be covered by γ^{10} balls of radius $\frac{r_1}{4}$. Since $\mathfrak{B}'(h', \mathbb{e}_2, \mathcal{H}_t) \subseteq \mathfrak{B}(h', 4 \cdot r_2) \cap \mathcal{H}_t$ we have that there exists a ball (i.e., $\mathfrak{B}(h', 4 \cdot r_2) \cap \mathcal{H}_t$) of radius $4 \cdot r_2$ that cannot be covered by γ^{10} balls of radius $\frac{r_1}{4}$. By the definition of the doubling dimension, this implies that $\frac{4 \cdot r_2}{1024} > \frac{r_1}{4}$, and therefore $r_2 > 64 \cdot r_1$.

■

Lemma 9 *For every $v \in \mathcal{V}'$, there exists an edge $\mathbb{e} \in \mathcal{N} \cap \mathcal{E}(v, \mathcal{H})$ such that $d(\mathbb{e}) \leq O(1) \cdot d(v, \mathcal{H})$.*

Proof Let $v \in \mathcal{V}'$ be a vertex. Let t be the traversed node (initially the root) during the construction of $\mathcal{N}_s$. Define $r_1 = d(v_1, v_m)$, $r_2 = d(\mathbb{e}_2)$, and $r_3 = d(v_1, v_j)$, where $v_1, v_m, v_j, \mathbb{e}_2$ are stored in node t. By Lemma 7, if the search proceeds to t_L, then $d(v_1, s) \leq 2r_2$; if it proceeds to t_R, then $d(v_1, s) \geq \frac{r_2}{2}$.

For the vertex v on node t, assume that the search is directed to the left node t_L but its actual nearest neighbor (in $\mathcal{H}$), denoted by $\text{NN}_{\mathcal{H}}(v)$, is not in t_L, then $\text{RANK}_{\pi_{\mathcal{E}'}}(\{v_1, \text{NN}_{\mathcal{H}}(v)\}) > \text{RANK}_{\pi_{\mathcal{E}'}}(\{v_1, v_j\})$, implying $d(v_1, \text{NN}_{\mathcal{H}}(v)) > \frac{r_3}{4}$. By the triangle inequality, this further implies

$$d(v, \text{NN}_{\mathcal{H}}(v)) \geq \frac{r_3}{4} - 2r_2 \geq 14r_2,$$

contradicting $d(v_1, v) \leq 2r_2$.

Conversely, if the search is directed to the right node t_R and its actual nearest neighbor is not in t_R, then $d(v_1, v) \geq \frac{r_2}{2}$ and $\text{RANK}_{\pi_{\mathcal{E}'}}(\text{NN}_{\mathcal{H}}(v)) < \text{RANK}_{\pi_{\mathcal{E}'}}(v_m)$. Therefore, $d(v_1, \text{NN}_{\mathcal{H}}(v)) \leq 4r_1$. By the triangle inequality, this further implies

$$d(v, \text{NN}_{\mathcal{H}}(v)) \geq \frac{r_2}{2} - 4r_1.$$

Since $r_2 > 64r_1$, it follows that

$$d(v, \text{NN}_{\mathcal{H}}(v)) \geq \frac{7r_2}{16}.$$

However, the node t_R contains v_m, and we have

$$d(v, v_m) \leq d(v, \text{NN}_{\mathcal{H}}(v)) + d(\text{NN}_{\mathcal{H}}(v), v_1) + d(v_1, v_m) \leq d(v, \text{NN}_{\mathcal{H}}(v)) + \frac{r_2}{8}.$$

Thus, if our algorithm proceeds with the left node t_L then it always makes the correct decision. If the nearest neighbor lies in t_L but our algorithm traverses the right node t_R, then we have that v_m (stored in node t) is an $O(1)$-approximate nearest neighbor of v. Putting everything together, we conclude that for each $v \in \mathcal{V}'$, the set $\mathcal{N}$ contains an $O(1)$-approximate nearest neighbor of v after the search completes. This proves Lemma 6. ■

Lemma 10 HEAVY-ANN *performs $O((n + |\mathcal{H}|^2)\text{polylog}\, n)$ queries to the tester $\hat{\mathcal{Q}}$.*

Proof First, the algorithm runs ADV-SORT$(\mathcal{E}(\mathcal{H}, \mathcal{H}))$ using the tester $\hat{\mathcal{Q}}$. From Lemma 5 and Lemma B.1.1, this subroutine performs $O((n + |\mathcal{H}|^2)\text{polylog}\, n)$ queries to the quadruplet oracle. The rest of the construction of the ring separator does not call any other oracle. For constructing the map $\mathcal{N}$, in each level of the tree we run one query to $\hat{\mathcal{Q}}$ using $O(\text{polylog}\, n)$ queries to the quadruplet oracle.

Since $m = \frac{|\mathcal{H}_t|}{2\gamma^{20}}$, we have,

$$\max(|L_t|, |R_t|) \leq \left(1 - \frac{1}{2\gamma^{20}}\right) |\mathcal{H}_t|.$$

By applying this inequality in every node of the tree recursively, we get that the tree depth is bounded by $O(\log n)$. Overall, the HEAVY-ANN performs $O(n\, \text{polylog}\, n)$ queries to the quadruplet oracle. ■

Putting Lemmas 7, 9 and 10 together implies Lemma 6.

B.2. The procedure BASIC-CENTER

In this section, we prove the following lemma.

Lemma 11 *Let Σ be a finite metric space $\Sigma = (\mathcal{V}, d)$ with size $|\mathcal{V}| = n$. There exists an algorithm* BASIC-CENTER *such that, given $\mathcal{U} \subseteq \mathcal{V}$ and a low-dislocation decomposition $\Pi = \{\pi_{\mathcal{E}'_1}, \ldots, \pi_{\mathcal{E}'_m}\}$, where $\mathcal{E}' = \mathcal{E}(\mathcal{U}, \mathcal{U})$,* BASIC-CENTER$(\mathcal{U}, \Pi, k)$ *returns a set $\mathcal{I}$ of size k such that $\textsc{Cost}^{\infty}_{\mathcal{I}}(\mathcal{U}) \leq 2 \cdot \mathsf{OPT}^{\infty}(\mathcal{U})$. The algorithm performs $O(m^2 \cdot \mathsf{polylog}\, n)$ calls to the distance oracle and no queries to the quadruplet oracle.*

Using Lemma 11, it is straightforward to obtain an $O(1)$-approximate k-center solution that uses $O(n^2 \,\mathsf{polylog}\, n)$ quadruplet queries and $O(\mathsf{polylog}\, n)$ distance queries. To achieve this, set $\Pi = \{\pi_{\mathcal{E}}\}$, where $\pi_{\mathcal{E}} =$ PROB-SORT$(\mathcal{E})$, and apply Lemma 11.

Next, we describe the procedure BASIC-CENTER$(\cdot)$, but before doing so, we discuss how to perform certain elementary operations.

- **Computing a threshold edge set.** Given Π, the threshold edge-set $\mathcal{E}'^{(\alpha)}$ can be computed as follows. For each $\pi_{\mathcal{E}'_i} \in \Pi$, call FILTER$(\pi_{\mathcal{E}'_i}, \alpha)$ and take the union of the resulting edges, i.e., $\bigcup_{i=1}^{m}$ FILTER$(\pi_{\mathcal{E}_i}, \alpha)$. Since every sequence in Π has a dislocation of at most $\mathcal{D}_{\Sigma}$, Lemma 4 implies that for each i, $\mathcal{E}'^{(\alpha)}_i =$ FILTER$(\pi_{\mathcal{E}'_i}, \alpha)$. Because $\bigcup_i \mathcal{E}'_i = \mathcal{E}'$, our claim follows.

  This computation requires $O(|\Pi| \,\mathsf{polylog}\, n)$ distance queries and no additional quadruplet queries.

- **Testing k-Coverability.** For any $\alpha \in \mathbb{R}_{\geq 0}$, define the graph $G^{(\alpha)} = (\mathcal{U}, \mathcal{E}'^{(\alpha)})$. The *square* of $G^{(\alpha)}$, denoted $\hat{G}^{(\alpha)}$, is the graph formed by adding an edge between any pair of vertices $u, v \in \mathcal{U}$ if and only if there exists a path of length at most 2 between u and v in $G^{(\alpha)}$. We say that $G^{(\alpha)}$ is *k-coverable* if the following *greedy covering algorithm* returns a set of size at most k.

  Greedy-Cover: Initialize $\mathcal{I}$ as an empty set. Then, while $\hat{G}^{(\alpha)}$ is not empty, select a vertex x from $\hat{G}^{(\alpha)}$, add x to $\mathcal{I}$, and delete x along with all its neighboring vertices from $\hat{G}^{(\alpha)}$. Notice that this greedy algorithm computes a maximal independent set of $\hat{G}^{(\alpha)}$. Furthermore, this computation does not require any oracle query.

We are now ready to describe the algorithm BASIC-CENTER$(\mathcal{U}, \Pi, k)$.

Algorithm BASIC-CENTER. The algorithm works in two steps.

(i) For each $\pi_{\mathcal{E}'_i} \in \Pi$, find $\mathfrak{e}'_i = \arg\min_{\mathfrak{e} \in \mathcal{E}'_i} \left(d(\mathfrak{e}) \mid G^{\left(d(\mathfrak{e})\right)} \text{ is } k\text{-coverable} \right)$.

  For a particular i, $\mathfrak{e}'_i$, can be computed as follows: Perform a binary search over the indices of $\pi_{\mathcal{E}'_i}$. For each midpoint index j, use the distance oracle to compute exact distances within the interval $[j - \mathcal{D}_{\Sigma}, j + \mathcal{D}_{\Sigma}]$ (see proof of Lemma 4 for more details) to accurately locate the edge $\mathfrak{e}$ corresponding to position j. Then, compute the threshold edge set $\mathcal{E}'^{\left(d(\mathfrak{e})\right)}$ and check whether $G^{\left(d(\mathfrak{e})\right)}$ is k-coverable using the previously defined procedures.

(ii) Let $\mathfrak{e}' = \arg\min\{\mathfrak{e}'_1, \ldots, \mathfrak{e}'_m\}$ denote the minimum weight edge found across all sequences in the previous step. Return $d(\mathfrak{e}')$ as the cost, along with the set of vertices $\mathcal{I} \subseteq \mathcal{V}$, which forms the maximal independent set of $\hat{G}^{\left(d(\mathfrak{e}')\right)}$, as the centers.

Analysis. Let $\mathfrak{e}^* = \arg\min_{\mathfrak{e}\in\mathcal{E}} \left(d(\mathfrak{e}) \mid G^{(d(\mathfrak{e}))} \text{ is } k\text{-coverable}\right)$. It is well-known Hochbaum and Shmoys (1986) that $G^{(d(\mathfrak{e}^*))}$ is coverable and that a maximal independent set $\mathcal{I}^*$ of $\hat{G}^{(d(\mathfrak{e}^*))}$ provides a 2-approximation, i.e., $\textsc{Cost}^{\infty}_{\mathcal{I}^*}(\mathcal{U}) \leq 2 \cdot \mathsf{OPT}^{\infty}(\mathcal{U})$. Therefore, if we can show that $\mathfrak{e}' = \mathfrak{e}^*$, the result follows.

In fact, it suffices to show that, for each sequence $\mathcal{E}'_i$, we find the minimum edge $\mathfrak{e}^*_i$ such that $G^{(d(\mathfrak{e}^*_i))}$ is coverable. Since each sequence $\pi_{\mathcal{E}_i}$ has dislocation at most $\mathcal{D}_\Sigma$, a modified binary search with exact distance calculations over $[j - \mathcal{D}_\Sigma, j + \mathcal{D}_\Sigma]$ at each index behaves identically to standard binary search, ensuring $\mathfrak{e}^*_i$ is found for each i.

Overall, the binary searches across all sequences require $O(|\Pi| \cdot \mathsf{polylog}\, n)$ steps. Therefore, the total number of distance queries—including those needed to compute the threshold edge sets is $O(|\Pi|^2 \cdot \mathsf{polylog}\, n)$. This completes the proof of Lemma 11.

B.3. The procedure ORACLE-CENTER

Let $\Sigma = (\mathcal{V}, d)$ be a finite metric space with bounded doubling dimension and size $|\mathcal{V}| = n$. In this section, we design an algorithm that computes a subset $\mathcal{A} \subseteq \mathcal{V}$ of size $|\mathcal{A}| = k$ such that $\textsc{Cost}^{\infty}_{\mathcal{A}}(\mathcal{V}) \leq O(1) \cdot \mathsf{OPT}^{\infty}$, while using $O((n + k^2)\mathsf{polylog}\, n)$ queries to the quadruplet oracle and $O(\mathsf{polylog}\, n)$ queries to the distance oracle.

The algorithm critically relies on the procedures HEAVY-ANN (Section B.1), BASIC-CENTER (Section B.2), FILTER, and PROB-SORT (Section A). In particular, it does not directly make any oracle queries (quadruplet or distance) but accesses them exclusively through these procedures.

To improve readability, while describing the algorithm we assume that all procedures invoked by the algorithm function as specified. In the analysis, we prove that this holds with high probability.

Algorithm ORACLE-CENTER. The algorithm operates in two phases. In *Phase 1*, it computes a subset $\hat{\mathcal{S}} \subseteq \mathcal{V}$ that serves as a coreset for k-center on Σ. In *Phase 2*, it computes a low-dislocation ordered decomposition of $\mathcal{E}(\hat{\mathcal{S}}, \hat{\mathcal{S}})$ and then uses BASIC-CENTER to obtain the final solution.

- **Phase 1.** This phase proceeds in rounds. We start with an initial set of vertices $\mathcal{V}_1 = \mathcal{V}$. In each round i, we process the current vertex set $\mathcal{V}_i$, and continue until $|\mathcal{V}_i| \leq 200 \cdot k \log n$, at which point we transition to the next phase. If the number of rounds exceeds τ, where $\tau = O(\mathsf{polylog}\, n)$, the algorithm aborts and returns no solution.

  Next, we describe the procedure for a fixed round i. For each step, we first state the objective of the step in *bold*, and then describe how to execute it using only the predefined subroutines.

  (i) **Sample two disjoint subsets $\mathcal{S}_i^{(1)} \subseteq \mathcal{V}_i$ and $\mathcal{S}_i^{(2)} \subseteq \mathcal{V}_i \setminus \mathcal{S}_i^{(1)}$, with $|\mathcal{S}_i^{(1)}| = |\mathcal{S}_i^{(2)}| = \omega_2 \cdot k \log n$, where $\omega_2 > 1$ is a suitable constant. Let $\mathcal{S}_i$ denote their union, i.e., $\mathcal{S}_i = \mathcal{S}_i^{(1)} \cup \mathcal{S}_i^{(2)}$.**

(ii) **Compute an approximate k-center solution $\mathcal{C}_i$ for $\mathcal{S}_i^{(1)}$ such that $\rho_i = \textsc{Cost}^{\infty}_{\mathcal{C}_i}(\mathcal{S}_i^{(1)}) \leq 2 \cdot \mathsf{OPT}^{\infty}(\mathcal{S}_i^{(1)})$.**

Let $\mathcal{X}_i = \mathcal{E}(\mathcal{S}_i^{(1)}, \mathcal{S}_i^{(1)})$ represent all edges between vertices in $\mathcal{S}_i^{(1)}$. Compute a low-dislocation ordering $\pi_{\mathcal{X}_i}$ using PROB-SORT$(\mathcal{X}_i)$. Compute $(\mathcal{C}_i, \rho_i) =$ BASIC-CENTER$(\mathcal{S}_i^{(1)}, \{\pi_{\mathcal{X}_i}\})$.

(iii) **Define a vertex $v \in \mathcal{C}_i$ as *center-heavy* if $|\mathfrak{B}(v, 3 \cdot \rho_i) \cap \mathcal{S}_i^{(2)}| \geq \Delta_{\textbf{HA}}$, where $\Delta_{\textbf{HA}} = \Theta(\log n)$. Identify $\mathcal{C}_i^{(H)} \subseteq \mathcal{C}_i$, the set of center-heavy vertices. Compute $\mathcal{H}_i \subseteq \mathcal{C}_i^{(H)}$, a $(6\rho_i, 6\rho_i)$-net of $\mathcal{C}_i^{(H)}$.**

Identifying center-heavy vertices $\mathcal{C}_i^{(H)} \subseteq \mathcal{C}_i$: Let $\mathcal{Y}_i = \mathcal{E}(\mathcal{C}_i \cup \mathcal{S}_i^{(2)}, \mathcal{S}_i^{(2)})$ represent the edges between $\mathcal{C}_i \cup \mathcal{S}_i^{(2)}$ and $\mathcal{S}_i^{(2)}$. Compute a low-dislocation ordering $\pi_{\mathcal{Y}_i}$ using PROB-SORT$(\mathcal{Y}_i)$. Next, compute the subset $\mathcal{Y}_i^{(3\rho_i)}$ by calling FILTER$(\pi_{\mathcal{Y}_i}, 3\rho_i)$. Inspect $\mathcal{Y}_i^{(3\rho_i)}$ to identify the subset of center-heavy vertices $\mathcal{C}_i^{(H)}$.
Computing $\mathcal{H}_i \subseteq \mathcal{C}_i^{(H)}$: Compute the subset of edges $\mathcal{X}_i^{(6\rho_i)}$ by calling FILTER$(\pi_{\mathcal{X}_i}, 6\rho_i)$. Construct the graph $G = (V', E')$, where the vertex set $V' = \mathcal{C}_i^{(H)}$ and edge set is $E' = \mathcal{X}_i^{(6\rho_i)} \cap \mathcal{E}(\mathcal{C}_i^{(H)}, \mathcal{C}_i^{(H)})$. Finally, set $\mathcal{H}_i$ to be a maximal independent set of the G.

(iv) **Compute $\mathcal{U}_i = \{v \in \mathcal{V}_i \setminus \mathcal{S}_i \mid d(v, \mathcal{H}_i) \leq O(1) \cdot \rho_i\}$.**

(a) **Compute an edge set $\mathcal{N}_i \subseteq \mathcal{E}(\mathcal{V}_i \setminus \mathcal{S}_i, \mathcal{H}_i)$ such that for any $v \in \mathcal{V}_i \setminus \mathcal{S}_i$, there exists an edge $\mathfrak{e} \in \mathcal{N}_i \cap \mathcal{E}(v, \mathcal{H}_i)$ satisfying $d(\mathfrak{e}) \leq C_{\textbf{HA}} \cdot d(v, \mathcal{H}_i)$, where $C_{\textbf{HA}} = O(1)$.**

To perform this step, we use the HEAVY-ANN$(\cdot)$ procedure. We need to ensure that the input to HEAVY-ANN$(\cdot)$ satisfies the conditions in Section B.1.
First, for each $h \in \mathcal{H}_i$, define CORE$_i(h) \subseteq \mathfrak{B}(v, 3\rho_i) \cap \mathcal{S}_i^{(2)}$ to be an arbitrary subset of size Δ_{HA}, where $\Delta_{\text{HA}} = \Theta(\log n)$ is defined in Section B.1. Compute each CORE$_i(h)$ by inspecting $\mathcal{Y}_i^{(3\rho_i)}$. Define CORE$_i = \bigcup_{h \in \mathcal{H}_i}$ CORE$_i(h)$. Compute $\mathcal{N}_i =$ HEAVY-ANN$(\mathcal{H}_i, \mathcal{V}_i \setminus \mathcal{S}_i,$ CORE$_i)$. Observe that $\mathcal{H}_i$ is a $(3\rho_i, 6\rho_i)$-net of CORE. Therefore, it directly follows from Lemma 6, that for every $v \in \mathcal{V}_i \setminus \mathcal{S}_i$, there exists an edge $\mathfrak{e} \in \mathcal{N} \cap \mathcal{E}(v, \mathcal{H}_i)$ such that $d(\mathfrak{e}) \leq C_{\text{HA}} \cdot d(v, \mathcal{H})$.

(b) **Use $\mathcal{N}_i$ to identify $\mathcal{U}_i$.**

For each edge $\{v, h\} \in \mathcal{N}_i$, where $v \in \mathcal{V}_i \setminus \mathcal{S}_i$ and $h \in \mathcal{H}_i$, replace it with an edge $\{v, c\}$, where $c \in$ CORE$_i(h)$ is picked arbitrarily. Let $\hat{\mathcal{N}}_i$ be the modified edge set.[3] Next, compute a low-dislocation ordering $\pi_{\hat{\mathcal{N}}_i} =$ PROB-SORT$(\hat{\mathcal{N}}_i)$. Finally, compute $\hat{\mathcal{N}}_i^{(\alpha)} =$ FILTER$(\pi_{\hat{\mathcal{N}}_i}, \phi)$, where $\alpha = 18 \cdot C_{\text{HA}} \cdot \rho_i$. Set $\mathcal{U}_i = \{v \in \mathfrak{e} \mid \mathfrak{e} \in \hat{\mathcal{N}}_i^{(\phi)}\}$.

(v) **Set $\mathcal{V}_{i+1} = \mathcal{V}_i \setminus (\mathcal{U}_i \cup \mathcal{S}_i)$ and proceed to the next round.**

3. This is done for technical reasons arising due to the persistence of the quadruplet oracle.

- **Phase 2.** Let i^* denote the index of the final round of Phase 1. Let $\hat{\mathcal{S}}_{i^*} = \mathcal{V}_{i^*}$. For each round i of Phase 1, define $\hat{\mathcal{S}}_i = \mathcal{C}_i \cup (\mathcal{S}_i^{(2)} \setminus \text{CORE}_i)$. Let $\hat{\mathcal{S}} = \bigcup_{i=1}^{i^*} \hat{\mathcal{S}}_i$. In the analysis, we show that $\hat{\mathcal{S}}$ is a coreset of $\mathcal{V}$.

  Construct the edge set $\mathcal{X}_{i^*} = \mathcal{E}(\mathcal{V}_{i^*}, \mathcal{V}_{i^*})$ and compute the sorted sequence $\pi_{\mathcal{X}_{i^*}} = \text{PROB-SORT}(\mathcal{X}_{i^*})$.

  Let

$$\mathcal{Z}_i = \mathcal{E}\left(\hat{\mathcal{S}}_i, \hat{\mathcal{S}} \setminus \bigcup_{j=1}^{i} \hat{\mathcal{S}}_j\right)$$

  capture edges between $\hat{\mathcal{S}}_i$ from round i and the remaining vertices in $\hat{\mathcal{S}}$ not included in previous rounds.

  For each round $i \in [1, i^* - 1]$, compute the sorted edge sequence $\pi_{\mathcal{Z}_i} = \text{PROB-SORT}(\mathcal{Z}_i)$.

  Consider the collection of edge sequences $\Pi = \{\pi_{\mathcal{X}_1}, \pi_{\mathcal{Y}_1}, \pi_{\mathcal{Z}_1}, \ldots, \pi_{\mathcal{X}_{i^*}}\}$ generated across all rounds. Let $\Pi' = \{\pi'_{\mathcal{X}_1}, \pi'_{\mathcal{Y}_1}, \pi'_{\mathcal{Z}_1}, \ldots, \pi'_{\mathcal{X}_{i^*}}\}$ be obtained by removing edges that do not belong to $\mathcal{E}(\hat{\mathcal{S}}, \hat{\mathcal{S}})$ from each edge sequence in Π. Observe that $|\Pi| = O(\text{polylog}\, n)$ and $|\Pi'| = O(\text{polylog}\, n)$. In the analysis we show that Π' is a low-dislocation ordered decomposition (see Section B.2) of $\mathcal{E}(\hat{\mathcal{S}}, \hat{\mathcal{S}})$. To obtain the final solution we compute $(\mathcal{A}, \rho^*) = \text{BASIC-CENTER}(\hat{\mathcal{S}}, \Pi')$.

B.3.1. ANALYSIS OF ORACLE-CENTER

We divide the analysis into three parts. In Part 1, we prove that, with high probability, every procedure invoked by the algorithm operates correctly. While in standard algorithm design this is straightforward as different components typically use independent randomness, in our setting we must ensure that subroutines do not share randomness by relying on the same quadruplet query response; otherwise, we cannot invoke the correctness of each subroutine directly. In Part 2, we establish that in each round i of Phase 1, the sample $\mathcal{S}_i$ satisfies certain properties that are essential both for invoking HEAVY-ANN$(\cdot)$ and for ensuring that a constant fraction of the vertices is eliminated in each round. Finally, in Part 3, by combining the correctness guarantees from Part 1 with the properties from Part 2, we prove that, with high probability, our algorithm returns an $O(1)$-approximate solution using $O((n + k^2)\text{polylog}\, n)$ quadruplet queries and $O(\text{polylog}\, n)$ distance queries.

Part 1: Proving Correctness of Operations. We consider each invocation of a subroutine as a single operation. Let the function $T : \mathbb{Z}_{\geq 0} \to \mathbb{Z}_{\geq 1}$ provide an upper bound on the number of operations executed by the algorithm.

By design, we have $T(n) = O(\text{polylog}\, n)$. This follows from the fact that Phase 1 consists of at most $\tau = O(\text{polylog}\, n)$ rounds, where each round involves only a constant number of calls to HEAVY-ANN$(\cdot)$, BASIC-CENTER$(\cdot)$, PROB-SORT$(\cdot)$, and FILTER$(\cdot)$. Similarly, Phase 2 involves only a constant number of calls, ensuring that the total number of operations remains within $O(\text{polylog}\, n)$.

Suppose we order the operations temporally. For any $t \in [T(n)]$, the algorithm executes operation a_t at time step t. We say the event CORRECT(a_t) is TRUE if the algorithm performs a_t correctly. For instance, CORRECT(PROB-SORT$(\mathcal{X}_i)$) = TRUE implies that $\pi_{\mathcal{X}_i}$ has dislocation at most $\mathcal{D}_\Sigma = O(\log n)$. We denote the complement of CORRECT(a_t) by INCORRECT(a_t). We define

$\textsc{Good}(t)$ as the event where all operations up to and including time step t are performed correctly, and $\textsc{Bad}(t)$ as the complement of $\textsc{Good}(t)$. In this section, we prove the following lemma.

Lemma 12 *For any* $t \in [T(n)]$, $\Pr[\textsc{Correct}(a_t) \mid \textsc{Good}(t-1)] \geq 1 - \frac{1}{n^{O(1)}}$.

Proof If a_t corresponds to FILTER or BASIC-CENTER, by Lemma 4 and Lemma 11, the claim holds trivially. Thus, we focus on cases where a_t is PROB-SORT or HEAVY-ANN.

We show that whenever the algorithm invokes PROB-SORT or HEAVY-ANN, each query made to $\tilde{\mathcal{Q}}$ is unique, having not been previously invoked at any step. If this condition is true, we can invoke Lemma 3 and Lemma 6, to establish our claim.

Let Γ denote the set of all queries made in *Phase 1*. For each round i in Phase 1, let $\Gamma(i) \subset \mathcal{E} \times \mathcal{E}$ represent all quadruplet queries in round i. Define $\Gamma(\pi_{\mathcal{X}_i})$, $\Gamma(\pi_{\mathcal{Y}_i})$, $\Gamma(\mathcal{N}_i)$, and $\Gamma(\hat{\mathcal{N}}_i)$ as subsets of $\Gamma(i)$ used to compute $\pi_{\mathcal{X}_i}$, $\pi_{\mathcal{Y}_i}$, $\mathcal{N}_i$, and $\pi_{\hat{\mathcal{N}}_i}$, respectively. Note that $\Gamma(i) = \Gamma(\pi_{\mathcal{X}_i}) \cup \Gamma(\pi_{\mathcal{Y}_i}) \cup \Gamma(\mathcal{N}_i) \cup \Gamma(\hat{\mathcal{N}}_i)$ and $\Gamma = \bigcup_i \Gamma(i)$. The following properties hold:

- Each query $(\mathfrak{e}_1, \mathfrak{e}_2)$ in $\Gamma(\pi_{\mathcal{X}_i})$ involves both edges in $\mathcal{E}(\mathcal{S}_i^{(1)}, \mathcal{S}_i^{(1)})$.
- Each query $(\mathfrak{e}_1, \mathfrak{e}_2)$ in $\Gamma(\pi_{\mathcal{Y}_i})$ involves both edges in $\mathcal{E}(\mathcal{S}_i^{(2)}, \mathcal{S}_i^{(2)}) \cup \mathcal{E}(\mathcal{S}_i^{(1)}, \mathcal{S}_i^{(2)})$.
- Each query $(\mathfrak{e}_1, \mathfrak{e}_2)$ in $\Gamma(\mathcal{N}_i)$ has one edge in $\mathcal{E}(\{\mathcal{H}_i \cup (\mathcal{V}_i \setminus \mathcal{S}_i)\}, \textsc{core}_i)$ and the other in $\mathcal{E}(\mathcal{H}_i, \mathcal{H}_i)$. (This follows from the analysis in Section B.1.)
- Each query $(\mathfrak{e}_1, \mathfrak{e}_2)$ in $\Gamma(\hat{\mathcal{N}}_i)$ involves both edges in $\mathcal{E}(\mathcal{V}_i \setminus \mathcal{S}_i, \textsc{core}_i)$.

These observations confirm that $\Gamma(\pi_{\mathcal{X}_i})$, $\Gamma(\pi_{\mathcal{Y}_i})$, $\Gamma(\mathcal{N}_i)$, and $\Gamma(\hat{\mathcal{N}}_i)$ are pairwise disjoint. Additionally, we can verify that $\Gamma(i) \cap \Gamma(j) = \emptyset$ for all $i \neq j$.

Similarly, let Γ' represent the set of quadruplet queries made in *Phase 2*, and let $\Gamma'(i)$ denote the queries used to compute $\pi_{\mathcal{Z}_i}$. Note that $\Gamma' = \bigcup \Gamma'(i)$, and for all i, j with $i \neq j$, $\Gamma'(i) \cap \Gamma'(j) = \emptyset$ holds since $\mathcal{Z}_i \cap \mathcal{Z}_j = \emptyset$. Additionally, we observe that $\Gamma \cap \Gamma' = \emptyset$ because each query in Phase 2 that involves a vertex from $\mathcal{V}_i$ (that might be in $\mathcal{S}^{(j)}$ for $j > i$) also involves a vertex from $\textsc{core}_i$, however $\bigcup_{\ell \in [\tau]} \textsc{core}_i \cap \hat{\mathcal{S}} = \emptyset$. This completes the proof. ■

Using Lemma 12, a straightforward inductive argument implies the following.

Lemma 13 *Let* $\textsc{Bad}$ *denote* $\textsc{Bad}(T(n))$. *Then,* $\Pr[\textsc{Bad}] \leq \frac{1}{n^{O(1)}}$.

Part 2: Proving Necessary Properties. Recall that in every round i of Phase 1, the algorithm samples $\mathcal{S}_i = \mathcal{S}_i^{(1)} \cup \mathcal{S}_i^{(2)} \subseteq \mathcal{V}_i$, where $\mathcal{S}_i^{(1)} \subseteq \mathcal{V}_i$ and $\mathcal{S}_i^{(2)} \subseteq \mathcal{V}_i \setminus \mathcal{S}_i^{(1)}$.

Let $P_i^{(1)} \subseteq \mathcal{S}_i^{(1)}$ be an optimal k-center solution for $\mathcal{S}_i^{(1)}$ and let $r^* = \textsc{Cost}^{\infty}_{P_i^{(1)}}(\mathcal{S}_i^{(1)}) = \mathsf{OPT}^{\infty}(\mathcal{S}_i^{(1)})$ denote its cost. Call a vertex $s \in \mathcal{S}_i^{(1)}$ *sample-heavy* if $|\mathfrak{B}(s, r^*) \cap \mathcal{V}_i| \geq \frac{|\mathcal{V}_i|}{100k}$ and let $\mathcal{S}_i^{(H)} \subseteq \mathcal{S}_i$ denote the subset of sample-heavy vertices.

Suitability of $\mathcal{S}_i$. We say the event $\textsc{Suitable}(\mathcal{S}_i)$ is true if all of the following conditions hold:

1. $r^* \leq 2 \cdot \mathsf{OPT}^{\infty}$
2. $|\mathfrak{B}(P_i^{(1)}, r^*) \cap \mathcal{V}_i| \geq \frac{99|\mathcal{V}_i|}{100}$.

3. For every sample-heavy $s \in \mathcal{S}_i^{(H)}$,

$$\left|\mathfrak{B}(s, r^*) \cap \mathcal{S}_i^{(2)}\right| \geq \Delta_{\text{HA}} \geq \Omega(\log n).$$

Informally speaking, SUITABLE($\mathcal{S}_i$) implies that an optimal k-center solution $P_i^{(1)}$ for $\mathcal{S}_i$ is a good solution for a large fraction of $\mathcal{V}_i \setminus \mathcal{S}_i$, and furthermore, every heavy vertex in $P_i^{(1)}$ has many representatives from its ball in $\mathcal{S}_i^{(2)}$. Observe that the event SUITABLE($\mathcal{S}_i$) is a *structural property* that is independent of the algorithm's state when $\mathcal{S}_i$ is computed. We prove the following.

Lemma 14 *In any round i, with probability $1 - \frac{1}{n^{O(1)}}$, SUITABLE($\mathcal{S}_i$) is true.*

Proof We prove each condition separately.

Condition 1: Let $\mathcal{A}^*$ denote an optimal k-center solution for $\mathcal{V}$ and let $\text{COST}^{\infty}_{\mathcal{A}^*}(\mathcal{V}) = \mathsf{OPT}^{\infty}$. Therefore, $\text{COST}^{\infty}_{\mathcal{A}^*}(P_i^{(1)}) \leq \mathsf{OPT}^{\infty}$. Since $P_i^{(1)} \subseteq \mathcal{S}_i^{(1)} \subseteq \mathcal{V}_i$ represents an optimal k-center solution on $\mathcal{S}_i^{(1)}$, the triangle inequality implies that $\text{COST}^{\infty}_{P_i^{(1)}}(\mathcal{S}_i^{(1)}) \leq 2 \cdot \mathsf{OPT}^{\infty}$.

Condition 2: Observe that any candidate k-center solution $\mathcal{C} = \{c_1, \ldots, c_k\} \subseteq \mathcal{V}_i$ for $\mathcal{V}_i$ can be represented as k pairs of the form $\text{rep}(\mathcal{C}) = \{(c_1, r_1), \ldots, (c_k, r_k)\}$, where each $r_j \in \mathcal{V}_i$ denotes the vertex defining the radius of the cluster centered by c_j. Under this representation, there are at most $O(n^{2k})$ distinct k-center solutions for $\mathcal{V}_i$. Henceforth, we work with this representation. Observe that this representation also allows for *partial clusterings* that may not cover all points. For a candidate k-center solution $\text{rep}(\mathcal{C}) = \{(c_1, r_1), \ldots, (c_k, r_k)\}$, define the *radius* of $\text{rep}(\mathcal{C})$ as $\text{RAD}(\text{rep}(\mathcal{C})) = \max_{c_i \in \mathcal{C}} d(c_i, r_i)$. We say that a clustering $\text{rep}(\mathcal{C})$ *fully covers* a set $\mathcal{U} \subseteq \mathcal{V}$, if $\text{COST}^{\infty}_{\mathcal{C}}(\mathcal{U}) \leq \text{RAD}(\text{rep}(\mathcal{C}))$.

For a (partial) clustering $\text{rep}(\mathcal{C})$, let $\text{Bad}_{\text{rep}(\mathcal{C})} = \{v \in \mathcal{V}_i \mid d(v, \mathcal{C}) > \text{RAD}(\text{rep}(\mathcal{C}))\}$. Formally, a clustering $\text{rep}(\mathcal{C})$ is ε-bad if $|\text{Bad}_{\text{rep}(\mathcal{C})}| \geq \varepsilon|\mathcal{V}_i|$. Let $\mathcal{F}$ denote the family of all distinct ε-bad k-center clustering of $\mathcal{V}_i$. Suppose $\text{rep}(\mathcal{C}) \in \mathcal{F}$. Since $\mathcal{S}_i^{(1)}$ is randomly sampled from $\mathcal{V}_i$, it holds that,

$$\Pr[\text{rep}(\mathcal{C}) \text{ fully covers } \mathcal{S}_i^{(1)}] \leq (1 - \frac{|\text{Bad}_{\text{rep}(\mathcal{C})}|}{|\mathcal{V}_i|})^{|\mathcal{S}_i^{(1)}|} \leq e^{-\varepsilon|\mathcal{S}_i^{(1)}|}.$$

This implies,

$$\Pr\left[\exists\, \text{rep}(\mathcal{C}) \in \mathcal{F},\ \text{rep}(\mathcal{C}) \text{ fully covers } \mathcal{S}_i^{(1)}\right] \leq |\mathcal{F}| e^{-\varepsilon|\mathcal{S}_i^{(1)}|} \leq n^{O(2k)} e^{-\varepsilon|\mathcal{S}_i^{(1)}|}. \tag{3}$$

Consider a representation $\text{rep}_{P_i^{(1)}}$ of $P_i^{(1)}$ such that each vertex in the representation belongs to $\mathcal{S}_i^{(1)}$. By design $\text{rep}(P_i^{(1)})$ fully covers $\mathcal{S}_i^{(1)}$. Recall that $\mathcal{S}_i^{(1)} = \omega_2 \cdot k \log n$. Set $\varepsilon = \frac{1}{100}$, by appropriate choice of ω_2, (3) implies that in any round i, with probability $1 - \frac{1}{n^{O(1)}}$, $|\text{Bad}_{\text{rep}(P_i^{(1)})}| \leq \frac{|\mathcal{V}_i|}{100}$ i.e., the number of vertices in $\mathcal{V}_i$ farther than $r^* = \text{RAD}(\text{rep}(P_i^{(1)}))$ from any vertex in $P_i^{(1)}$ is at most $\frac{|\mathcal{V}_i|}{100}$ with high probability. This implies

$$|\mathfrak{B}(P_i^{(1)}, r^*) \cap \mathcal{V}_i| \geq \frac{99|\mathcal{V}_i|}{100}.$$

This proves condition 2.

Condition 3: Recall that a vertex $s \in \mathcal{S}_i^{(H)} \subseteq \mathcal{S}_i^{(1)}$ satisfies

$$|\mathfrak{B}(s, r^*) \cap \mathcal{V}_i| \geq \frac{|\mathcal{V}_i|}{100k}.$$

Since $\mathcal{S}_i^{(2)} \subseteq \mathcal{V} \setminus \mathcal{S}_i^{(1)}$ of size $\omega_2 \cdot k \log n$ uniformly at random. Suppose ω_2 is appropriately large. Using Chernoff bounds, with high probability, for every sample-heavy $h \in \mathcal{S}_i^{(H)}$,

$$\left|\mathfrak{B}(h, r^*) \cap \mathcal{S}_i^{(2)}\right| \geq \Delta_{\text{HA}} = \Theta(\log n).$$

■

The next lemma follows in a straightforward manner from the definition of SUITABLE($\mathcal{S}_i$).

Lemma 15 *Given* SUITABLE($\mathcal{S}_i$) *is true, there exists a subset* $\bar{S}_i \subseteq P_i^{(1)} \cap \mathcal{S}_i^{(H)}$*, such that* $|\mathfrak{B}(\bar{S}_i, r^*) \cap \mathcal{V}_i| \geq \frac{98}{100}|\mathcal{V}_i|$*. Hence,* $|\mathfrak{B}(P_i^{(1)} \cap \mathcal{S}_i^{(H)}, r^*) \cap \mathcal{V}_i| \geq \frac{98}{100}|\mathcal{V}_i|$.

Proof Under condition 2 of SUITABLE, we have $|\mathfrak{B}(P_i^{(1)}, r^*) \cap \mathcal{V}_i| \geq \alpha_0 \cdot |\mathcal{V}_i|$, where $\alpha_0 = \frac{99}{100}$. Notice that $|P_i^{(1)}| = k$ so there exists at least one center $p_1 \in P_i^{(1)}$ such that $|\mathfrak{B}(p_1, r^*) \cap \mathcal{V}_i| \geq \frac{\alpha_0}{k} \cdot |\mathcal{V}_i| > \frac{|\mathcal{V}_i|}{100k}$, so $p_1 \in \mathcal{S}_i^{(H)}$. We add $p_1 \in \bar{S}_i$. Let $P_{i,1}^{(1)} \leftarrow P_i^{(1)} \setminus \{p_1\}$, and let $|\mathfrak{B}(P_{i,1}^{(1)}, r^*) \cap \mathcal{V}_i| \geq \alpha_1 \cdot |\mathcal{V}_i|$, where $\alpha_1 \in [0, 1]$. Since $|P_{i,1}^{(1)}| = k - 1$, we have that there exists a center $p_2 \in P_{i,1}^{(1)}$ such that $|\mathfrak{B}(p_2, r^*) \cap \mathcal{V}_i| \geq \frac{\alpha_1}{k-1} \cdot |\mathcal{V}_i|$. If $\alpha_1 \geq \frac{1}{100}$, it also holds that $|\mathfrak{B}(p_2, r^*) \cap \mathcal{V}_i| \geq \frac{\alpha_1}{k-1} \cdot |\mathcal{V}_i| \geq \frac{|\mathcal{V}_i|}{100k}$, so $p_2 \in \mathcal{S}_i^{(H)}$, we add p_2 in $\bar{S}_i$, and we set $P_{i,2}^{(1)} \leftarrow P_{i,1}^{(1)} \setminus \{p_2\}$. We continue with the same way, until $|\mathfrak{B}(P_{i,j}^{(1)}, r^*) \cap \mathcal{V}_i| \leq \alpha_j \cdot |\mathcal{V}_i|$, where $\alpha_j < \frac{1}{100}$. At this point, we have $|\mathfrak{B}(\bar{S}_i, r^*) \cap \mathcal{V}_i| \geq |\mathfrak{B}(P_i^{(1)}, r^*) \cap \mathcal{V}_i| - |\mathfrak{B}(P_{i,j}^{(1)}, r^*) \cap \mathcal{V}_i| \geq \frac{99}{100}|\mathcal{V}_i| - \frac{1}{100}|\mathcal{V}_i| = \frac{98}{100}|\mathcal{V}_i|$.

■

Part 3: Putting everything together. Let SUITABLE denote that for all i, SUITABLE($\mathcal{S}_i$) is true. Let UNSUITABLE denote the complement of SUITABLE. Since the Phase 1 can only run for $\tau = O(\text{polylog}\, n)$ rounds, the combined number of operations in Phase 1 and Phase 2 is bounded by $O(\text{polylog}\, n)$. Therefore, using Lemma 14, it follows that $\Pr[\text{UNSUITABLE}] \leq \frac{1}{n^{O(1)}}$.

Let FAVORABLE denote the event GOOD $\cap$ SUITABLE. By combining the previous fact with Lemma 13, we can conclude the following.

Lemma 16 $\Pr[\text{FAVORABLE}] \geq 1 - \frac{1}{n^{O(1)}}$

We perform the following analysis under the assumption that FAVORABLE is true.

Lemma 17 *Given that* FAVORABLE *is true, in every round* i *of Phase 1, for every* $s \in \mathcal{S}_i^{(H)}$ *there exists* $c_s \in \mathcal{C}_i^{(H)}$ *such that* $d(s, c_s) \leq \rho_i$*. Furthermore, for every* $s \in \mathcal{S}_i^{(H)}$*, there exists* $h \in \mathcal{H}_i$ *such that* $\mathfrak{B}(h, 9 \cdot \rho_i) \cap \mathcal{V}_i \supseteq \mathfrak{B}(s, r^*) \cap \mathcal{V}_i$.

Proof Consider some round i of Phase 1 of the algorithm. Recall the definition of SUITABLE$(\mathcal{S}_i)$. Let s be a vertex in $\mathcal{S}_i^{(H)}$. Let $c_s \in \mathcal{C}_i$ be any vertex such that $d(s, c_s) \leq \rho_i$. Such a vertex must exist since $\mathcal{C}_i$ is a k-center solution of $\mathcal{S}_i^{(1)}$ computed using BASIC-CENTER (See Lemma 11) and FAVORABLE holds. We must show that the algorithm identifies c_s as center-heavy.

By Condition 3 of SUITABLE$(\mathcal{S}_i)$, $|\mathfrak{B}(s, r^*) \cap \mathcal{S}_i^{(2)}| \geq \Delta_{\text{HA}}$. Furthermore, Lemma 11, also implies that $\rho_i \leq 2r^*$. Therefore, $|\mathfrak{B}(c_s, 3\rho_i) \cap \mathcal{S}_i^{(2)}| \geq |\mathfrak{B}(s, r^*) \cap \mathcal{S}_i^{(2)}| \geq \Delta_{\text{HA}}$. This implies that c_s is identified as center-heavy by the algorithm. Equivalently, notice that $\mathfrak{B}(c_s, 3\rho_i) \cap \mathcal{V}_i \supseteq \mathfrak{B}(s, r^*) \cap \mathcal{V}_i$. The subset $\mathcal{H}_i$ is a $(6\rho_i, 6\rho_i)$-net of $\mathcal{C}_i^{(H)}$ so there exists $h \in \mathcal{H}_i$ such that $d(c_s, h) \leq 6\rho_i$. Hence, $\mathfrak{B}(h, 9 \cdot \rho_i) \cap \mathcal{V}_i \supseteq \mathfrak{B}(s, r^*) \cap \mathcal{V}_i$. ■

Lemmas 15, 17 together imply the following lemma.

Lemma 18 *Given that* FAVORABLE *is true, in every round i of Phase 1,*

$$|\mathfrak{B}(\mathcal{H}_i, 9 \cdot \rho_i) \cap \mathcal{V}_i| \geq \frac{98}{100}|\mathcal{V}_i|$$

Proof Consider the set $\bar{\mathcal{S}}_i \subseteq P_i^{(1)} \cap \mathcal{S}_i^{(H)}$ from Lemma 15. Use Lemma 17 to map every $s \in \bar{\mathcal{S}}_i$ to a vertex in $\mathcal{H}_i$. Let $\mathcal{H}_i' \subseteq \mathcal{H}_i$ denote the image of this mapping. By Lemma 15, $\bigcup_{h \in \mathcal{H}_i'} |\mathfrak{B}(h, 9 \cdot \rho_i) \cap \mathcal{V}_i| \geq \frac{98}{100} \cdot \mathcal{V}_i$. This completes the proof. ■

We are now ready to conclude.

Lemma 19 *Given that* FAVORABLE *is true, the set $\mathcal{A}$ returned by the algorithm satisfies* $\text{COST}_{\mathcal{A}}^{\infty}(\mathcal{V}) \leq O(1) \cdot \mathsf{OPT}^{\infty}$. *Moreover, the algorithm makes $O((n + k^2) \cdot \mathsf{polylog}\, n)$ queries to the quadruplet oracle and $O(\mathsf{polylog}\, n)$ queries to the distance oracle.*

Proof We first prove correctness and then bound the query complexity.

Correctness: Consider Step (iv) of some round i of Phase 1 of the algorithm. Let ξ denote the number of vertices in the set $\{v \in \mathcal{V}_i \setminus \mathcal{S}_i \mid \mathcal{E}(v, \mathcal{H}_i) \cap \mathcal{N}_i^{(9 \cdot \rho_i)} \neq \emptyset\}$. By Lemma 6, for every vertex $v \in \big(\mathfrak{B}(\mathcal{H}_i, 9\,\rho_i) \cap \mathcal{V}_i\big) \setminus \mathcal{S}_i$, there exists an edge $\mathfrak{e} \in \mathcal{N}_i$ with $d(\mathfrak{e}) \leq 9 \cdot C_{\text{HA}} \cdot \rho_i$. Combining this with Lemma 18, we obtain

$$\xi + |\mathcal{S}_i| \geq \frac{98}{100} \cdot |\mathcal{V}_i|.$$

Observe that for every $h \in \mathcal{H}_i$ and $v \in \text{CORE}_i(h)$, $d\big(v, h\big) \leq \rho_i$. Furthermore, in Step (iv), $\mathcal{N}_i'$ is filtered with $\alpha = 18 \cdot C_{\text{HA}} \cdot \rho_i$. It follows that $|\mathcal{U}_i \cup \mathcal{S}_i| \geq \frac{98}{100}\,|\mathcal{V}_i|$. Therefore, $O(\log n)$ rounds are sufficient for Phase 1 to complete.

Next, consider the set $\hat{\mathcal{S}}_i = \mathcal{C}_i \cup \big(\mathcal{S}_i^{(2)} \setminus \text{CORE}_i\big)$ in Phase 2. By Condition 1 of SUITABLE$(\mathcal{S}_i)$ we have $\mathsf{OPT}^{\infty}(\mathcal{S}_i^{(1)}) \leq 2 \cdot \mathsf{OPT}^{\infty}$. We also know from Lemma 11 that $\rho_i \leq 2 \cdot \mathsf{OPT}^{\infty}(\mathcal{S}_i^{(1)})$. Since $\mathcal{H}_i \subseteq \mathcal{C}_i$, it follows that $\text{COST}_{\mathcal{C}_i}^{\infty}(\mathcal{U}_i \cup \mathcal{S}_i^{(1)} \cup \text{CORE}_i) \leq O(1) \cdot \mathsf{OPT}^{\infty}(\mathcal{S}_i^{(1)}) \leq O(1) \cdot \mathsf{OPT}^{\infty}$. Therefore, in each round i of Phase 1,

$$\mathcal{U}_i \cup \mathcal{S}_i = \mathcal{U}_i \cup \mathcal{S}_i^{(1)} \cup \Big(\mathcal{S}_i^{(2)} \setminus \text{CORE}_i\Big) \cup \text{CORE}_i \subseteq \mathfrak{B}\big(\hat{\mathcal{S}}_i,\, O(1) \cdot \mathsf{OPT}^{\infty}\big) \cap \mathcal{V}_i.$$

Furthermore, since $\hat{\mathcal{S}} = \bigcup_i \hat{\mathcal{S}}_i$, we can conclude that, $\max_{v \in \mathcal{V}} d\big(v, \hat{\mathcal{S}}\big) \leq O(1) \cdot \mathsf{OPT}^{\infty}$. It is well known (see Har-Peled (2001)) that this implies that $\hat{\mathcal{S}}$ is a coreset for the k-center problem on $\mathcal{V}$. Next, consider the the sequence Π' in Phase 2. Observe that $\mathcal{E}(\hat{\mathcal{S}}, \hat{\mathcal{S}}) = \bigcup_{i=1}^{i^*-1} (\pi'_{\mathcal{X}_i}, \pi'_{\mathcal{Y}_i}, \pi'_{\mathcal{Z}_i}) \cup \pi'_{\mathcal{X}_{i^*}}$. Since FAVORABLE is true each sequence is a low-dislocation ordering and therefore Π' forms a low-dislocation decomposition of $\mathcal{E}(\hat{\mathcal{S}}, \hat{\mathcal{S}})$. Hence, by Lemma 11, $\textsc{Cost}^{\infty}_{\mathcal{A}}(\mathcal{V}) \leq O(1) \cdot \mathsf{OPT}^{\infty}$.

Query Complexity: In each round of Phase 1, the algorithm calls PROB-SORT a constant number of times, resulting in $O((n + k^2) \cdot \mathsf{polylog}\, n)$ quadruplet queries. It invokes FILTER a constant number of times, incurring $O(\mathsf{polylog}\, n)$ distance queries. In addition, it calls BASIC-CENTER and HEAVY-ANN once per round, with HEAVY-ANN contributing an additional $O((n + k^2) \cdot \mathsf{polylog}\, n)$ quadruplet queries and BASIC-CENTER contributing $O(\mathsf{polylog}\, n)$ distance queries. Thus, each round of Phase 1 incurs a total of $O((n + k^2) \cdot \mathsf{polylog}\, n)$ quadruplet queries and $O(\mathsf{polylog}\, n)$ distance queries. Since there are $O(\log n)$ rounds, the overall query complexity for Phase 1 is bounded accordingly.

Phase 2 executes PROB-SORT once, making $O(k^2 \cdot \mathsf{polylog}\, n)$ quadruplet queries. Moreover, the algorithm invokes BASIC-CENTER$(\hat{\mathcal{S}}, \Pi', k)$, which uses $O(|\Pi'|^2 \cdot \mathsf{polylog}\, n) = O(\mathsf{polylog}\, n)$ distance queries (by Lemma 11); note that $|\Pi'| = O(\mathsf{polylog}\, n)$ because the number of rounds in Phase 1 is bounded by $O(\log n)$.

Summing everything, the algorithm makes $O((n + k^2) \cdot \mathsf{polylog}\, n)$ quadruplet queries and $O(\mathsf{polylog}\, n)$ distance queries.

■

Lemma 19 directly implies the following.

Theorem 20 *There exists an algorithm* ORACLE-CENTER *such that, given a metric space $\Sigma = (\mathcal{V}, d)$ of size $|\mathcal{V}| = n$ with bounded doubling dimension, that supports a probabilistic quadruplet oracle with error $\varphi \leq \frac{1}{16}$ and a distance oracle, the algorithm* ORACLE-CENTER(Σ) *returns, with high probability, a set $\mathcal{A} \subseteq \mathcal{V}$ of size k such that $\textsc{Cost}^{\infty}_{\mathcal{A}}(\mathcal{V}) \leq O(1) \cdot \mathsf{OPT}^{\infty}$, using $O((n + k^2) \cdot \mathsf{polylog}\, n)$ queries to the quadruplet oracle and $O(\mathsf{polylog}\, n)$ queries to the distance oracle.*

B.4. Extension to general metric spaces

A simpler variant of ORACLE-CENTER applies to general metric spaces. It follows the same overall structure; we provide a brief overview. In Phase 1, instead of drawing two samples $\mathcal{S}^{(1)}$ and $\mathcal{S}^{(2)}$, it draws only a single sample.

Specifically, in Phase 1, sample a set $\mathcal{S}_i \subseteq \mathcal{V}_i$ of size $\Theta(k \log n)$, and compute $(\mathcal{C}_i, \rho_i) =$ BASIC-CENTER$(\mathcal{S}_i, \{\pi_{\mathcal{X}_i}\})$, where $\mathcal{X}_i = \mathcal{E}(\mathcal{S}_i, \mathcal{S}_i)$. Next, instead of using HEAVY-ANN, set $\mathcal{N}_i = \mathcal{E}(\mathcal{C}_i, \mathcal{V}_i \backslash \mathcal{S}_i)$ and compute the ordered sequence $\pi_{\mathcal{N}_i} =$ PROB-SORT$(\mathcal{N}_i)$. Then compute FILTER$(\pi_{\mathcal{N}_i}, 18 \cdot \rho_i)$ to obtain $\mathcal{N}_i^{(18 \cdot \rho_i)}$, and define $\mathcal{U} = \{v \in \mathcal{V} \mid \exists e \in \mathcal{N}_i^{(18 \cdot \rho_i)}\}$. Finally, recurse on $\mathcal{V}_i \setminus (\mathcal{U} \cup \mathcal{S}_i)$.

In Phase 2, compute a low-dislocation decomposition of $\bigcup_i \mathcal{S}_i$ as before, and then run BASIC-CENTER. A simplified version of the preceding analysis yields the following result.

Theorem 21 *There exists an algorithm such that, given a metric space $\Sigma = (\mathcal{V}, d)$ of size $|\mathcal{V}| = n$, that supports a probabilistic quadruplet oracle with error $\varphi \leq \frac{1}{16}$ and a distance oracle, the algorithm returns, with high probability, a set $\mathcal{A} \subseteq \mathcal{V}$ of size k such that $\textsc{Cost}^{\infty}_{\mathcal{A}}(\mathcal{V}) \leq O(1) \cdot \mathsf{OPT}^{\infty}$, using $O(nk\, \mathsf{polylog}\, n)$ queries to the quadruplet oracle and $O(\mathsf{polylog}\, n)$ queries to the distance oracle.*

Appendix C. Approximate Distance Data Structure

Let $\Sigma = (\mathcal{V}, d)$ be a finite metric space of size $n = |\mathcal{V}|$ with bounded doubling dimension and bounded spread. In this section we prove the following theorem.

Theorem 22 *There exists an algorithm* CONSTRUCT-NET*, such that given a metric space $\Sigma = (\mathcal{V}, d)$ of size $|\mathcal{V}| = n$, with bounded doubling dimension and polynomially bounded spread, that supports a probabilistic quadruplet oracle $\tilde{\mathcal{Q}}$ with error $\varphi \leq \frac{1}{16}$ and a distance oracle, the algorithm* CONSTRUCT-NET(Σ) *returns a data structure $\mathbf{D}$ such that the following hold with high probability:*

1. *Constructing $\mathbf{D}$ requires $O(n \operatorname{polylog} n)$ calls to the quadruplet oracle and $O(\operatorname{polylog} n)$ calls to the distance oracle.*
2. *Querying $\mathbf{D}$ with any $(u, v) \in \mathcal{V} \times \mathcal{V}$ returns a number $\mathfrak{d}(u, v) \in \mathbb{R}_{\geq 0}$ in $O(\log n)$ steps, without requiring any additional oracle queries, such that $d(u, v) \leq \mathfrak{d}(u, v) \leq O(1) \cdot d(u, v)$.*
3. *$\mathbf{D}$ requires $O(n \operatorname{polylog} n)$ space.*

We refer the reader to Section 3.2 for a detailed overview of our approach. We proceed as follows. In Section C.1, we describe the structural properties of the data structure. Next, in Section C.2 we present the construction algorithm. In Section C.3, we explain the query procedure.

C.1. Structure

Let $\mu = \lceil \log \mathsf{diam}(\Sigma) \rceil$, and $\lambda = \lfloor \log \mathsf{sep}(\Sigma) \rfloor$. The data structure comprises a collection of nets Γ that approximate Σ at every scale. More specifically, $\Gamma = \{\mathcal{L}_\mu, \mathcal{L}_{\mu-1}, \ldots, \mathcal{L}_{\lambda+1}, \mathcal{L}_\lambda\}$, where each $\mathcal{L}_i \subseteq \mathcal{V}$, such that the following properties hold: $\mathcal{L}_{i+1} \subseteq \mathcal{L}_i$ for all $i \in [\lambda, \mu - 1]$. For every integer $i \in [\lambda, \mu]$, the set $\mathcal{L}_i$ is a $(2^i, 2^{i-1})$-net of $\mathcal{V}$. Moreover, for each $v \in \mathcal{L}_i$, we store a set $\mathfrak{B}_v \subseteq \mathfrak{B}(v, 2^i) \cap \mathcal{V}$ containing v, and these sets form a partition of $\mathcal{V}$, i.e., $\bigcup_{v \in \mathcal{L}_i} \mathfrak{B}_v = \mathcal{V}$ and $\mathfrak{B}_u \cap \mathfrak{B}_v = \emptyset$ for all distinct $u, v \in \mathcal{L}_i$.

The data structure also keeps track of some additional information.

(i) *Parent:* For each $i \in [\lambda, \mu]$ and each $u \in \mathcal{V}$, let $\text{PAR}(u, i)$ denote the unique vertex $v \in \mathcal{L}_i$ such that $u \in \mathfrak{B}_v$ (we allow $\text{PAR}(u, i) = u$).

(ii) *Conflict-Set:* For each $i \in [\lambda, \mu]$ and each $v \in \mathcal{L}_i$, define $\text{CONFLICT}(v, i) = \mathfrak{B}(v, 4 \cdot 2^i) \cap \mathcal{L}_i$.

Although the above are generally defined, for every $i \in [\lambda, \mu]$, the data structure only tracks $\text{PAR}(v, i)$, $\text{PAR}(v, i+1)$, $\text{CONFLICT}(v, i)$, and $\text{CONFLICT}(\text{PAR}(v, i+1), i+1)$.

Next we prove some essentially properties. Let $\zeta = 2^{\dim(\Sigma)}$,

Lemma 23 *The following properties hold for any integer $i \in [\lambda, \mu]$:*

(i) $|\mathcal{L}_i| \leq O(\zeta^{\mu - i})$.

(ii) For every $v \in \mathcal{L}_i$, $|\text{CONFLICT}(v, i)| \leq \zeta^4 = O(1)$.

(iii) For any $u, v \in \mathcal{L}_i$, if $u \in \text{CONFLICT}(v, i)$, then

$$\text{PAR}(u, i+1) \in \text{CONFLICT}(\text{PAR}(v, i+1), i+1).$$

Proof We prove each part separately. We critically use the *doubling property*, i.e., in a metric space with doubling constant ζ, any ball of radius R can contain at most $\zeta^{\lceil \log_2(\frac{2R}{r}) \rceil}$ points that are r-separated.

(i) Observe that $\mathcal{L}_i$ is a set of 2^{i-1}-separated points in $\mathcal{V}$. Since the diameter of $\mathcal{V}$ is at most 2^μ, all points in $\mathcal{L}_i$ lie within a ball of radius 2^μ. Applying the doubling property with $R = 2^\mu$ and $r = 2^{i-1}$, we get

$$|\mathcal{L}_i| \leq \zeta^{\left\lceil \log_2\left(\frac{2\cdot 2^\mu}{2^{i-1}}\right)\right\rceil} = \zeta^{\lceil \log_2(2^{\mu-i+2})\rceil} = \zeta^{\mu-i+2}.$$

Therefore, $|\mathcal{L}_i| \leq \zeta^{\mu-i+2} = O(\zeta^{\mu-i})$.

(ii) By definition, for any $v \in \mathcal{L}_i$, $\textsc{conflict}(v,i) = \mathfrak{B}(v, 4\cdot 2^i) \cap \mathcal{L}_i$. Applying the doubling property with $R = 4\cdot 2^i$ and $r = 2^{i-1}$, we get that

$$|\textsc{conflict}(v,i)| \leq \zeta^{\lceil \log_2(\frac{2R}{r})\rceil} = \zeta^{\left\lceil \log_2\left(\frac{2\cdot(4\cdot 2^i)}{2^{i-1}}\right)\right\rceil} = \zeta^{\lceil \log_2(16)\rceil} = \zeta^4.$$

(iii) For any $x \in \mathcal{V}$, by design $d(x, \textsc{par}(x, i+1)) \leq 2^{i+1}$. Suppose $u \in \textsc{conflict}(v,i)$. Using the triangle inequality, we get that

$$\begin{aligned} d(\textsc{par}(u,i+1), \textsc{par}(v,i+1)) &\leq d(\textsc{par}(u,i+1), u) + d(u,v) + d(v, \textsc{par}(v,i+1)) \\ &\leq 2^{i+1} + 4\cdot 2^i + 2^{i+1} = 4\cdot 2^{i+1}. \end{aligned}$$

It follows that

$$\textsc{par}(u,i+1) \in \mathfrak{B}\left(\textsc{par}(v,i+1), 4\cdot 2^{i+1}\right) \cap \mathcal{L}_{i+1} = \textsc{conflict}(\textsc{par}(v,i+1), i+1).$$

■

C.2. The procedure CONSTRUCT-NET

We now describe the procedure used to compute the net, which we refer to as CONSTRUCT-NET. Suppose we are given μ and λ. We build Γ level by level in a top-down manner, from level $i = \mu$ down to level $i = \lambda$. During the construction, every time we handle an edge, we mark it as *processed*. We make sure that if an edge $\mathtt{e} \in \mathcal{E}$ has been marked as *processed* during any stage of the construction, in all subsequent stages we can determine **without any oracle queries** whether $d(\mathtt{e}) \leq 2^j$ for every $j \in [\lambda, \mu]$.

We first describe the computation of μ and λ, and then describe the procedure for constructing an arbitrary level.

Computing μ and λ. We assume that a bound on the the spread $\mathfrak{S}$ is known. Pick any vertex $v^* \in \mathcal{V}$ and consider the set of edges $\mathcal{E}_{v^*} = \mathcal{E}(v^*, \mathcal{V})$ incident on v^*. Compute $\pi_{\mathcal{E}^*_v} = \textsc{prob-sort}(\mathcal{E}^*_v)$. Then, perform distance queries on the last $\mathcal{D}_\Sigma = O(\log n)$ entries of $\pi_{\mathcal{E}^*_v}$ and take the maximum value, denoted by α. Set $\mu = \log(2\alpha)$ and $\lambda = \log\left(\frac{\alpha}{2\mathfrak{S}}\right)$. Next, bucket the edges in $\mathcal{E}_{v^*}$ by computing $\textsc{bucket}(\pi_{\mathcal{E}_{v^*}}, \lambda, \mu, 2)$ and mark them as processed.

Construction. Suppose the nets have been constructed from level μ down to level $i+1$. We now describe the process for constructing level i. As we did in the previous section, for every step we first state the objective of that step in **bold** and then describe how to achieve it using predefined procedures in Section A.2.

1. **Compute a 2^{i-1}-cover of each set $\mathfrak{B}_v$, where $v \in \mathcal{L}_{i+1}$.**

   This step proceeds in rounds. For each vertex $v \in \mathcal{L}_{i+1}$, initialize $\mathfrak{B}_v^{(1)} = \mathfrak{B}_v$. Repeat the following steps for $\tau = O(\mathsf{polylog}\, n)$ rounds. In round j:

   (a) **Sample a set of κ_1 points $\mathcal{S}_v^{(j)}$ from each $\mathfrak{B}_v^{(j)}$, where $v \in \mathcal{L}_{i+1}$.**
   For each $v \in \mathcal{L}_{i+1}$, sample a subset $\mathcal{S}_v^{(j)} \subseteq \mathfrak{B}_v^{(j)}$ of size $|\mathcal{S}_v^{(j)}| = \kappa_1 = \Theta(\zeta\, \mathsf{polylog}\, n)$. If it is the first round (i.e., $j = 1$), add v to $\mathcal{S}_v^{(1)}$, so that $\mathcal{S}_v^{(1)} = \mathcal{S}_v^{(1)} \cup \{v\}$.

   (b) **Compute $\mathcal{X}_j$, the union of edges between $\mathcal{S}_v^{(j)}$ and $\mathfrak{B}_v^{(j)} \setminus \mathcal{S}_v^{(j)}$, for each $v \in \mathcal{L}_{i+1}$.**
   Define $\mathcal{X}_j = \bigcup_{v \in \mathcal{L}_{i+1}} \mathcal{E}(\mathcal{S}_v^{(j)}, \mathfrak{B}_v^{(j)} \setminus \mathcal{S}_v^{(j)})$ and $\mathcal{S}^{(j)} = \bigcup_{v \in \mathcal{L}_{i+1}} \mathcal{S}_v^{(j)}$.

   (c) **Bucket the unseen edges in $\mathcal{X}_j$ based on weight.**
   Partition $\mathcal{X}_j$ into $\bar{\mathcal{X}}_j$, the set of unprocessed edges, and $\mathcal{X}_j \setminus \bar{\mathcal{X}}_j$, the set of edges previously processed in the construction. Compute $\pi_{\bar{\mathcal{X}}_j} = \textsc{prob-sort}(\bar{\mathcal{X}}_j)$. Bucket the edges in $\bar{\mathcal{X}}_j$ by computing $\textsc{bucket}(\pi_{\bar{\mathcal{X}}_j}, \lambda, \mu, 2)$ and mark them as processed.

   (d) **Identify the covered vertices, i.e., vertices in $\mathfrak{B}(\mathcal{S}_v^{(j)}, 2^{i-1}) \cap \mathfrak{B}_v^{(j)}$ for each $v \in \mathcal{L}_{i+1}$.**
   Since $(\mathcal{X}_j \setminus \bar{\mathcal{X}}_j)^{(2^{i-2})}$ have already been processed in previous rounds, they can be computed without additional queries. Similarly, we can obtain $\bar{\mathcal{X}}_j^{(2^{i-1})}$ from the previous step. Taking their union, we obtain $\mathcal{X}_j^{(2^{i-1})} = \bar{\mathcal{X}}_j^{(2^{i-1})} \cup (\mathcal{X}_j \setminus \bar{\mathcal{X}}_j)^{(2^{i-1})}$.
   We now use $\mathcal{X}_j^{(2^{i-1})}$ to map each vertex that is within distance 2^{i-1} from at least one sample in $\mathcal{S}^{(j)}$, to one such sample (arbitrarily in case there are multiple such samples). More formally, for each $s \in \mathcal{S}^{(j)}$, we define $\mathcal{M}_s = \emptyset$. We then make a pass over $\mathcal{X}_j^{(2^{i-1})}$. For every edge $(s, b) \in \mathcal{X}_j^{(2^{i-1})}$, where $s \in \mathcal{S}^{(j)}$ we add b to $\mathcal{M}_s$, i.e., $\mathcal{M}_s = \mathcal{M}_s \cup \{b\}$, and remove all edges from $\mathcal{X}_j^{(2^{i-1})}$ that contain the vertex b.

   (e) **Remove the covered vertices from each $\mathfrak{B}_v^{(j)}$, where $v \in \mathcal{L}_{i+1}$.**
   For each $v \in \mathcal{L}_{i+1}$, update $\mathfrak{B}_v^{(j+1)} = \mathfrak{B}_v^{(j)} \setminus \bigcup_{s \in \mathcal{S}_v^{(j)}} \mathcal{M}_s$.

   At the end of the rounds, we set $\mathcal{S}_v = \bigcup_{j=1}^{\tau} \mathcal{S}_v^{(j)}$ for each $v \in \mathcal{L}_{i+1}$ and define $\mathcal{S} = \bigcup_{v \in \mathcal{L}_{i+1}} \mathcal{S}_v$. In the analysis we show that $\mathcal{S}_v$ is a 2^{i-1}-cover of $\mathfrak{B}_v$ for each $v \in \mathcal{L}_{i+1}$.

2. **Compute $\mathcal{L}_i$, a $(2^{i-1}, 2^{i-1})$-net of $\mathcal{S}$.**

   (a) **Identify an edge-set $\mathcal{Z} \subseteq \mathcal{E}(\mathcal{S}, \mathcal{S})$ such that if $d(s_1, s_2) \le 2^{i-1}$, $\{s_1, s_2\} \in \mathcal{Z}$.**
   Let $\mathcal{Z} = \{\{s, s'\} \in \mathcal{E}(\mathcal{S}, \mathcal{S}) \mid \textsc{par}(s', i+1) \in \textsc{conflict}(\textsc{par}(s, i+1), i+1)\}$. Since $|\textsc{conflict}(\textsc{par}(s, i+1), i+1)| \le O(1)$, we have $|\mathcal{Z}| \le O(|\mathcal{S}|)$.

(b) **Bucket the unseen edges in $\mathcal{Z}$ based on weight.** Partition $\mathcal{Z}$ into $\bar{\mathcal{Z}}$, edges not previously processed, and $\mathcal{Z} \setminus \bar{\mathcal{Z}}$, edges that have been processed. Compute $\pi_{\bar{\mathcal{Z}}} = \text{PROB-SORT}(\bar{\mathcal{Z}})$. Bucket the edges in $\bar{\mathcal{Z}}$ by computing $\text{BUCKET}(\pi_{\bar{\mathcal{Z}}}, \lambda, \mu, 2)$ and mark them as processed.

(c) **Compute $\mathcal{L}_i$.** Note that all edges in $\mathcal{Z}$ are already processed. Identify the edges in $\mathcal{Z}^{(2^{i-1})}$. Consider the graph $G = (\mathcal{S}, \mathcal{Z}^{(2^{i-1})})$. Find a maximal independent set $\mathcal{I}$ of G that includes vertices in $\mathcal{L}_i$. By design, vertices in $\mathcal{L}_{i+1}$ do not have an edge between them in G. Hence, this is always possible. Set $\mathcal{L}_i = \mathcal{I}$.

3. **Bookkeeping.** For every $u \in \mathcal{S} \setminus \mathcal{L}_i$, arbitrarily map it to some vertex in $v \in \mathcal{L}_i$ such that $d(u, v) \leq 2^{i-1}$. Note that this must be possible since $\mathcal{L}_i$ is a maximal independent set. Let $\mathcal{C}_v \subseteq \mathcal{S}$ mapped to $v \in \mathcal{L}_i$. Set $\mathfrak{B}_v = \bigcup_{u \in \mathcal{C}_v} \mathcal{M}_u$ as the vertices covered by $v \in \mathcal{L}_i$. For every $v \in \mathcal{V}$, store $\text{PAR}(v, i)$. Use $\mathcal{Z}^{(2^{i+2})}$ to compute $\text{CONFLICT}(v, i)$ for every $v \in \mathcal{L}_i$ and store it.

Analysis. First, we show that the estimates μ and λ are accurate.

Lemma 24 *The following conditions hold:*

(i) $\mathit{diam}(\mathcal{V}) \leq 2^{\mu} \leq 2\,\mathit{diam}(\mathcal{V})$.

(ii) $\dfrac{\mathit{sep}(\mathcal{V})}{2} \leq 2^{\lambda} \leq \mathit{sep}(\mathcal{V})$.

(iii) $\mu - \lambda = O(\log n)$.

Proof Let $\mathsf{diam} = \mathsf{diam}(\mathcal{V})$ and $\mathsf{sep} = \mathsf{sep}(\mathcal{V})$. Consider the estimation procedure. Let $v \in \mathcal{V}$ be the initial vertex. By Lemma 3, the maximum dislocation of the edge $\{v^\star, v\}$, where $v^\star$ is the farthest neighbor of v, is at most $\mathcal{D}_\Sigma = O(\log n)$ from the end of $\pi_{\mathcal{N}_v}$. Hence, the estimate satisfies $\alpha = d(v, v^\star)$. Moreover, $\alpha \leq \mathsf{diam} \leq 2\alpha$, where the upper bound follows from the triangle inequality. This implies $\mathsf{diam} \leq 2^{\mu} \leq 2 \cdot \mathsf{diam}$, and $2^{\lambda} = \frac{\alpha}{2\mathbb{S}} = \frac{\alpha \cdot \mathsf{sep}}{2 \cdot \mathsf{diam}} \leq \mathsf{sep}$, and since $2\alpha \geq \mathsf{diam}$, it also follows that $2^{\lambda} \geq \frac{\mathsf{sep}}{2}$. Therefore, $\mu - \lambda = O(\log \mathbb{S}) = O(\log n)$. ■

Now observe that the construction algorithm never invokes the quadruplet oracle on any pair of edges more than once. Each call to PROB-SORT is applied to a set of edges that has not been processed previously, so the correctness of each invocation is independent. Consequently, by Lemma 3, we can assert that any invocation of PROB-SORT in the algorithm returns an ordered sequence with maximum dislocation at most $\mathcal{D}_\Sigma = O(\log n)$. Furthermore, there are $O(\log n)$ levels (by Lemma 24), each with $\tau = O(\mathsf{polylog}\, n)$ calls to PROB-SORT in Step 1 and a constant number of calls to PROB-SORT in Step 2. By applying the union bound, we conclude that every invocation is correct with high probability. We denote this event by FAVORABLE.

Lemma 25 FAVORABLE *holds with probability* $1 - \frac{1}{n^{O(1)}}$.

We now prove the that the data structure is computed as per the specifications.

Lemma 26 *Given* FAVORABLE *is true, with probability at least* $1-\frac{1}{n^{O(1)}}$*, the family* $\Gamma = \{\mathcal{L}_\lambda, \ldots, \mathcal{L}_\mu\}$ *satisfies the structural conditions in* (C.1)*. The construction requires* $O(n \operatorname{polylog} n)$ *calls to the quadruplet oracle and* $O(\operatorname{polylog} n)$ *distance queries. Moreover, the total space including additional information is* $O(n \log n)$*.*

Proof First, note that since FAVORABLE is true, by Lemma 4, this implies all calls to BUCKET$(\cdot)$ work as intended. We now show that every $\mathcal{L}_i$ is a $(2^i, 2^{i-1})$-net of $\mathcal{V}$. Fix a level $i \in [\lambda, \mu]$.

Correctness: At the end of Step 1, for every $v \in \mathcal{L}_{i+1}$ the set $\mathcal{S}_v$ is a 2^{i-1}-cover of $\mathfrak{B}_v$. Indeed, since $\mathfrak{B}_v \subseteq \mathfrak{B}(v, 2^{i+1}) \cap \mathcal{V}$, the doubling property ensures there exists a set $Q \subseteq \mathfrak{B}_v$ with $|Q| = 4\zeta$ and $\textsc{Cost}_Q^\infty(\mathfrak{B}_v) \le 2^{i-1}$. Given that $\kappa_1 = \Theta(\zeta \operatorname{polylog} n)$, the proof of Condition 2 of Lemma 14 implies that

$$|\mathfrak{B}(\mathcal{S}_v^{(1)}, 2^{i-1}) \cap \mathfrak{B}_v^{(1)}| \ge \frac{99}{100} \cdot |\mathfrak{B}_v^{(1)}|.$$

Repeating for $\tau = O(\operatorname{polylog} n)$ rounds yields $\mathcal{S}_v = \bigcup_{j=1}^{\tau} \mathcal{S}_v^{(j)}$, a 2^{i-1}-cover of $\mathfrak{B}_v$.

At the end of Step 2, let $\mathcal{S} = \bigcup_{v \in \mathcal{L}_{i+1}} \mathcal{S}_v$. We claim that $\mathcal{L}_i$ forms a $(2^{i-1}, 2^{i-1})$-net of $\mathcal{S}$. By construction, the set $\mathcal{Z}$ in Step 2 includes every edge $(s, s') \in \mathcal{E}(\mathcal{S}, \mathcal{S})$ for which $\textsc{par}(s', i+1)$ lies in $\textsc{CONFLICT}(\textsc{par}(s, i+1))$. Lemma 23 then ensures that $\mathcal{Z}$ contains $\textsc{CONFLICT}(v, i)$ and that $|\mathcal{Z}| = O(n \operatorname{polylog} n)$. Moreover, $\mathcal{L}_i \subseteq \mathcal{S}$ is chosen as a maximal independent set in the graph $G = (\mathcal{S}, \mathcal{Z}^{(2^{i-1})})$, which guarantees that for any distinct $u, v \in \mathcal{L}_i$ we have $d(u, v) > 2^{i-1}$, while every vertex in $\mathcal{S} \setminus \mathcal{L}_i$ is within 2^{i-1} of some vertex in $\mathcal{L}_i$. Thus, $\mathcal{L}_i$ is a $(2^{i-1}, 2^{i-1})$-net of $\mathcal{S}$.

At the end of Step 3, $\mathcal{L}_i$ is a 2^i-cover of $\mathcal{V}$. For any $u \in \mathcal{V}$ that belongs to $\mathcal{M}_s$ for some $s \in \mathcal{S}$ in Step 1 and is mapped to $v \in \mathcal{L}_i$ in Step 3, we have $d(u, s) \le 2^{i-1}$ and $d(s, v) \le 2^{i-1}$, so by the triangle inequality $d(u, v) \le 2^i$. Therefore, every $u \in \mathcal{V}$ is within 2^i of some $v \in \mathcal{L}_i$, and $\mathcal{L}_i$ is a $(2^i, 2^{i-1})$-net of $\mathcal{V}$, as required.

Repeating the above argument for every level $i \in [\lambda, \mu]$ completes the correctness proof. Furthermore, since the data structure is constructed as specified and the conflict sets have constant size, the space complexity is obviously $O(n \log n)$.

Query Complexity: First, by Lemma 24 there are only $\ell = (\mu - \lambda + 1) = O(\log n)$ levels. For each level, in Step 1, each set $\bar{\mathfrak{X}}_j$ has size $O(n \operatorname{polylog} n)$ since every vertex in $\mathfrak{B}_v^{(j)} \setminus \mathcal{S}_v^{(j)}$ can only have edges to its $O(\operatorname{polylog} n)$ centers in $\mathcal{S}_v^{(j)}$. As $j \le \tau = O(\operatorname{polylog} n)$, the total number of quadruplet queries in Step 1 is $O(n \operatorname{polylog} n)$ and the number of distance queries is $O(\operatorname{polylog} n)$, incurred by τ invocations of PROB-SORT$(\cdot)$ and $\tau \cdot \ell$ invocations of BUCKET$(\cdot)$ respectively.

Similarly, since $|\mathcal{Z}| \le O(n \operatorname{polylog} n)$, the single call to PROB-SORT$(\cdot)$ in Step 2 requires $O(n \operatorname{polylog} n)$ quadruplet queries, and the subsequent calls to BUCKET$(\cdot)$ incur $O(\operatorname{polylog} n)$ distance queries due to ℓ invocations.

Overall, over ℓ levels the total number of quadruplet queries is $O(n \operatorname{polylog} n)$ and the total number of distance queries is $O(\operatorname{polylog} n)$.

■

C.3. Querying edge weights

Query Procedure. Given an edge $\{u, v\} \in \mathcal{E}$, let i^* be the minimum level for which one of the following conditions holds:

(1) $\textsc{par}(u, i^*) = \textsc{par}(v, i^*)$,

(2) $\textsc{par}(u, i^*) \neq \textsc{par}(v, i^*)$ and $\textsc{par}(v, i^*) \in \textsc{conflict}(\textsc{par}(u, i^*), i^*)$.

Return 2^{i^*+3}.

Analysis. First, observe that the query procedure does not require any oracle calls. We now prove that the distance returned by the algorithm is a good approximation

Lemma 27 *Given a data structure satisfying the conditions in Section C.1, for every $\mathtt{e} \in \mathcal{E}$, the above procedure returns a distance $\mathtt{d}_{\mathtt{e}} \in \mathbb{R}_{\geq 0}$ such that $d(u, v) \leq \mathtt{d}_{\mathtt{e}} \leq 8 \cdot d(u, v)$.*

Proof Let $\{u, v\} \in \mathcal{E}$ such that $d(u, v) \in [2^{j-1}, 2^j)$ for some integer j. Let the distance returned by the query procedure be 2^{i^*+3}. By definition of conflict-sets, we know that if $\textsc{par}(u, i) \neq \textsc{par}(v, i)$ and $\textsc{par}(v, i) \notin \textsc{conflict}(\textsc{par}(u, i), i)$, then $d(\textsc{par}(u, i), \textsc{par}(v, i)) > 4 \cdot 2^i$. We use this observation to upper bound and lower bound i^*.

We first show that $i^* \leq j - 1$. Equivalently, we show by for every $i \geq j - 1$ at least one of the conditions of the query procedure holds.

Suppose none of the conditions hold for some $i \geq j - 1$. By triangle inequality, we have

$$d(u, v) \geq d(\textsc{par}(v, i), \textsc{par}(u, i)) - d(v, \textsc{par}(v, i)) - d(u, \textsc{par}(u, i)) > 4 \cdot 2^i - 2^i - 2^i = 2 \cdot 2^i \geq 2 \cdot 2^{j-1} = 2^j,$$

which is a contradiction because $d(u, v) \in [2^{j-1}, 2^j)$. Hence, $i^* \leq j - 1$. Next, we show that $i^* \geq j - 3$. Observe the following:

$$d(u, v) \leq d(u, \textsc{par}(u, i)) + d(\textsc{par}(u, i), \textsc{par}(v, i)) + d(\textsc{par}(v, i), v) \leq 6 \cdot 2^i \leq (1.5) \cdot 2^{i+2}.$$

If $i^* \leq j - 4$, we have $d(u, v) \leq (1.5) \cdot 2^{j-2} < 2^{j-1}$, which is also a contradiction.

So the level i^* found by the query procedure must satisfy $j - 3 \leq i^* \leq j - 1$. Consequently, the estimate 2^{i^*+3} is bounded by

$$d(u, v) \leq 2^{i^*+3} \leq 8 \cdot d(u, v).$$

■

Putting Lemma 25, Lemma 26 and Lemma 27 together, we get Theorem 22

Appendix D. k-median and k-means OracleCluster problems

In this section, we develop $O(1)$-approximation algorithms for the k-median/means OracleCluster problem. For metric spaces with bounded doubling dimension, our algorithm uses $O((n + k^2)\,\mathsf{polylog}\, n)$ quadruplet queries and $O(\mathsf{polylog}\, n)$ distance queries. For general metrics, the algorithm requires $O(nk\,\mathsf{polylog}\, n)$ quadruplet queries and $O(\mathsf{polylog}\, n)$ distance queries. An overview of our approach is given in Section 3.3. Our algorithms critically use the ORACLE-CENTER algorithm from Section B.3 and CONSTRUCT-NET from Section C.1. We focus on the k-median algorithm here, noting that the k-means version follows with only minor modifications. The algorithm and analysis for spaces with bounded doubling dimension are presented in Section D.1 and Section D.1.1 respectively. The case for general metric spaces is discussed in Section D.2.

D.1. The procedure ORACLE-MEDIAN

As done in previous sections, we first describe the objective of each step in **bold** and then discuss how to implement it.

Algorithm ORACLE-MEDIAN.

1. **Compute $(\mathcal{C}, \rho^*) =$ ORACLE-CENTER$(\mathcal{V}, k)$.**

   Compute a k-center solution $(\mathcal{C}, \rho^*) =$ ORACLE-CENTER$(\mathcal{V}, k)$, where ORACLE-CENTER is the algorithm described in Section B.3. Theorem 20 implies that $\text{COST}^{\infty}_{\mathcal{C}}(\mathcal{V}) \leq \rho^* \leq O(1) \cdot \mathsf{OPT}^{\infty}(\mathcal{V})$.

   For each $v \in \mathcal{C}$, compute a set $\Psi(v) \subseteq \mathcal{V}$ such that $d(v, \Psi(v)) \leq O(1) \cdot \rho^*$ and $\bigcup_{v \in \mathcal{C}} \Psi(v) = \mathcal{V}$. Note that while ORACLE-CENTER$(\mathcal{V}, k)$ does not explicitly output such a mapping, it can easily be deduced directly from Steps (i), (ii) and (iv) of Phase 1 and Phase 2 of ORACLE-CENTER without additional oracle queries.

2. **Compute a decomposition of $\mathcal{V}$ such that the diameter of each component is $O(n^5\rho^*)$.**

   Let $\bar{\mathcal{E}} = \mathcal{E}(\mathcal{C}, \mathcal{C})$. From Phase 2 of ORACLE-CENTER$(\mathcal{V}, k)$ we get a low-dislocation decomposition $\bar{\Pi}$ of $\bar{\mathcal{E}}$. Let $\bar{\Pi} = \{\pi_{\bar{\mathcal{E}}_1}, \ldots, \pi_{\bar{\mathcal{E}}_{\bar{m}}}\}$ where $\bar{m} = O(\mathsf{polylog}\, n)$ (this follows from the analysis of ORACLE-CENTER).

   For each $i \in [\bar{m}]$, compute $\bar{\mathcal{E}}_i^{(n^3 \cdot \rho^*)} =$ FILTER$(\pi_{\bar{\mathcal{E}}_i}, n^3\rho^*)$ and construct the graph $G = (\mathcal{C}, \bar{\mathcal{E}}^{(n^3 \cdot \rho^*)})$. Let $\mathcal{C}_1, \ldots, \mathcal{C}_p \subseteq \mathcal{C}$ denote a partition of $\mathcal{C}$, where all vertices in the same $\mathcal{C}_i$ belong to the same connected component in G. Observe that $p \leq k$.

   Next, let $\mathcal{S}_\ell^{(1)}$ denote the *first* set of samples obtained in Step (i) of round ℓ of Phase 1 of ORACLE-CENTER$(\mathcal{V})$. Define $\mathcal{S}^{(1)} = \bigcup_{\ell \in [\tau]} \mathcal{S}_\ell^{(1)}$ where $\tau = O(\mathsf{polylog}\, n)$ is the number of rounds in Phase 1. Define

   $$\mathcal{U}_i = \Big(\bigcup_{v \in \mathcal{C}_i} \Psi(v) \Big) \setminus \mathcal{S}^{(1)},$$

   Observe that for each i, $\mathsf{diam}(\mathcal{U}_i) \leq O(n^4\rho^*)$.

3. **Let $\mu = \lceil \log(n^5 \cdot \rho^*) \rceil$ and $\lambda = \lfloor \log\left(\frac{\rho^*}{n^5}\right) \rfloor$. Compute $\mathcal{T}_i =$ CONSTRUCT-NET$(\mathcal{U}_i, \mu, \lambda)$ for each $\mathcal{U}_i$.**

   CONSTRUCT-NET denotes the algorithm for constructing an approximate distance data structure in Section C. Since there can be up to k connected components, we cannot compute each CONSTRUCT-NET$(\mathcal{U}_i, \mu, \lambda)$ independently as that may require $\Omega(k)$ distance oracle queries. Moreover, we also need to make sure we do not reuse any quadruplet query used by ORACLE-CENTER. We proceed as follows.

   (a) **Bucket edges processed directly by ORACLE-CENTER$(\mathcal{V}, k)$.** Let $\mathcal{E}' \subseteq \mathcal{E}$ be the *cluster-processed* edges, i.e. those processed by some invocation of PROB-SORT during the computation ORACLE-CENTER$(\mathcal{V}, k)$ (i.e., the edges in $\pi_{\mathcal{X}_\ell}$, $\pi_{\mathcal{Y}_\ell}$, $\pi_{\hat{\mathcal{N}}_\ell}$, $\pi_{\mathcal{X}_{\ell^*}}$, and $\pi_{\mathcal{Z}_\ell}$ for $\ell \in [\tau]$), and let

   $$\mathcal{E}' = \bigcup_{\ell \in [\tau]} \Big(\pi_{\mathcal{X}_\ell} \cup \pi_{\mathcal{Y}_\ell} \cup \pi_{\hat{\mathcal{N}}_\ell} \cup \pi_{\mathcal{X}_{\ell^*}} \cup \pi_{\mathcal{Z}_\ell} \Big).$$

Bucket the edges in $\mathcal{E}'$ by computing $\textsc{bucket}(\pi, \lambda, \mu, 2)$ for each sequence $\pi \in \{\pi_{\mathcal{X}_\ell}, \pi_{\mathcal{Y}_\ell}, \pi_{\hat{\mathcal{N}}_\ell}, \pi_{\mathcal{X}^*_\ell}, \pi_{\mathcal{Z}_\ell} : \ell \in [\tau]\}$.

(b) **Compute $\mathcal{T}_i = \textsc{Construct-Net}(\mathcal{U}_i)$ for every i simultaneously by dovetailing their executions and sharing oracle queries.**

Observe that each $\mathcal{T}_i$ has the same depth. Consequently, the $\textsc{prob-sort}$ and $\textsc{bucket}$ operations required by each $\textsc{Construct-Net}(\mathcal{U}_i)$ can be shared by dovetailing their executions, ensuring that each round of Phase 1 of every $\textsc{Construct-Net}(\mathcal{U}_i)$ is completed before proceeding to the next round, and similarly in Phase 2.

Suppose level ℓ of each $\mathcal{T}_i$ is being computed. Let $\bar{\mathcal{X}}_{j,i}$ denote the set of unprocessed edges in round j of Phase 1 of $\textsc{Construct-Net}(\mathcal{U}_i)$. The algorithm $\textsc{Construct-Net}(\mathcal{U}_i)$ partitions edges into buckets by first ordering them via $\pi_{\bar{\mathcal{X}}_{j,i}} = \textsc{prob-sort}(\bar{\mathcal{X}}_{j,i})$ and then applying $\textsc{bucket}(\pi_{\bar{\mathcal{X}}_{j,i}}, \lambda, \mu, 2)$. Instead of performing these operations separately for each i, we can directly bucket all edges together by computing $\pi_{\hat{\mathcal{X}}_j} = \textsc{prob-sort}(\hat{\mathcal{X}}_j)$, where $\hat{\mathcal{X}}_j = \bigcup_{i\in[p]} \bar{\mathcal{X}}_{j,i}$, followed by $\textsc{bucket}(\pi_{\hat{\mathcal{X}}_j}, \lambda, \mu, 2)$.

To ensure that the construction of any $\mathcal{T}_i$ does not reuse any query made during the computation of $\textsc{oracle-center}(\mathcal{V}, k)$, we replace $\hat{\mathcal{X}}_j$ with $\hat{\mathcal{X}}_j \setminus \mathcal{E}'$ in the above process. A similar approach applies to Phase 2. Note that whenever any $\textsc{Construct-Net}(\mathcal{U}_i)$ requires information about an edge $\mathfrak{e} \in \mathcal{E}'$, its bucket has already been computed.

4. **Run a standard k-median algorithm on each $\mathcal{U}_i$ independently. Use the the solutions to compute a coreset $\mathcal{M}$.**

Let Alg_M denote the *standard* $O(1)$-approximation algorithm for the k-median problem described in Arya et al. (2001). We run Alg_M on each component $\mathcal{U}_i$ as follows: Whenever the algorithm requires the distance between any pair of vertices $u, v \in \mathcal{U}_i \times \mathcal{U}_i$, it queries the structure $\mathcal{T}_i$ to obtain a constant-stretch estimate of $d(u, v)$. If for a pair of vertices $u, v \in \mathcal{U}_i \times \mathcal{U}_i$, the query procedure on $\mathcal{T}_i$ does not find a parameter i^* satisfying the conditions in Section C.3, we estimate $d(u, v)$ as 2^λ.

Let $\mathcal{A}_i$ denote the k-median solution produced by Alg_M on $\mathcal{U}_i$ (using the approximate distances provided by $\mathcal{T}_i$). For every $u \in \mathcal{A}_i$, let $\bar{n}_u$ be the number of vertices in $\mathcal{U}_i$ that are assigned to u by $\mathsf{Alg}_M(\mathcal{U}_i)$. Define the (weighted) set

$$\mathcal{M} = \Big(\bigcup_{i\in[p]} \mathcal{A}_i\Big) \cup \mathcal{S}^{(1)},$$

assigning each $u \in \mathcal{A}_i$ weight $\bar{n}_u$, and each $u \in \mathcal{S}^{(1)}$ weight 1. We note that $\mathcal{M}$ forms a coreset for the k-median problem on $\mathcal{V}$.

5. **Run a standard k-median algorithm on the coreset $\mathcal{M}$.**

Finally, run Alg_M on the coreset $\mathcal{M}$. Observe that the distance queries made by Alg_M can be categorized into three types:

Type I: $u, v \in \mathcal{A}_i$ for some $i \in [p]$.

Type II: $u \in \mathcal{A}_i$ and $v \in \mathcal{A}_j$, where $i \neq j$ and $i, j \in [p]$.

Type III: $u \in \mathcal{A}_i$ for some $i \in [p]$ and $v \in \mathcal{S}^{(1)}$.

Type IV: $u, v \in \mathcal{S}^{(1)}$.

For Type I, use $\mathcal{T}_i$ to estimate $d(u, v)$, and for Type II, define the distance to be ∞. To answer queries of Type III or IV, let $\mathcal{A}^* = \bigcup_{i \in [p]} \mathcal{A}_i$, compute $\mathcal{E}'' = \mathcal{E}(\mathcal{S}^{(1)}, \mathcal{A}^*) \setminus \mathcal{E}'$, obtain $\pi_{\mathcal{E}''} = \text{PROB-SORT}(\mathcal{E}'')$, and then compute $\text{BUCKET}(\pi_{\mathcal{E}''}, \lambda, \mu, 2)$ to answer all Type III queries; next, define $\hat{\mathcal{E}} = \left(\bigcup_{\ell_1 < \ell_2 \in [\tau]} \mathcal{E}(\mathcal{S}^{(1)}_{\ell_1}, \mathcal{S}^{(1)}_{\ell_2})\right) \setminus \mathcal{E}'$, compute $\pi_{\hat{\mathcal{E}}} = \text{PROB-SORT}(\hat{\mathcal{E}})$, and bucket the edges by computing $\text{BUCKET}(\pi_{\hat{\mathcal{E}}}, \lambda, \mu, 2)$ to answer all Type IV queries, where $u \in \mathcal{S}^{(1)}_{\ell_1}$ and $v \in \mathcal{S}^{(1)}_{\ell_2}$, for $\ell_1 \neq \ell_2$. If $u, v \in \mathcal{S}_\ell$, for $\ell \in [\tau]$, then $\mathcal{E}'$ is used to estimate $d(u, v)$ (step 3 (a)).

Finally, let $\mathcal{P}$ denote the set of k vertices returned by Alg_M on $\mathcal{M}$, and return $\mathcal{P}$ as the final solution.

D.1.1. ANALYSIS OF ORACLE-MEDIAN

Similarly to the correctness proof of the k-center OracleCluster problem, we first show that with high probability all subroutines perform correctly. It suffices to show that no quadruplet query is every repeated. This allows us to invoke the correctness of ORACLE-CENTER, as well as every invocation of CONSTRUCT-NET and PROB-SORT independently. We then complete the analysis under the condition that these routines perform correctly.

Lemma 28 *For any two distinct edges $\mathbb{e}_1, \mathbb{e}_2 \in \mathcal{E}$, the algorithm never invokes $\tilde{\mathcal{Q}}(\mathbb{e}_1, \mathbb{e}_2)$ more than once.*

Proof This is true in Step 1. As stated in Lemma 12, the desired property holds when executing $\text{ORACLE-CENTER}(\mathcal{V}, k)$. In Step 2, note that no quadruplet queries are required.

In Step 3, during the construction of each data structure $\mathcal{T}_i$, no queries are repeated as established in Theorem 22. Moreover, whenever an edge was cluster-processed, it was excluded from the PROB-SORT procedure, which guarantees that the steps remain independent with respect to invocations of PROB-SORT directly by $\text{ORACLE-CENTER}(\mathcal{V}, k)$. Furthermore, observe that the edges provided as input to the probabilistic oracle during the HEAVY-ANN procedure (in the k-center algorithm) do not appear in the construction of any $\mathcal{T}_i$. This follows from the fact that these edges always contain at least one heavy vertex (see Section B.1) from the set $\mathcal{S}^{(1)}$. However, we explicitly remove $\mathcal{S}^{(1)}$ when defining $\mathcal{U}_i$. In Step 4 again, no quadruplet queries are used.

Finally, in Step 5 consider the procedures $\text{PROB-SORT}(\mathcal{E}'')$ and $\text{PROB-SORT}(\hat{\mathcal{E}})$. Since we have already removed the cluster-processed edges, it remains only to ensure that none of the remaining edges were processed in a HEAVY-ANN procedure. Indeed, the queries given as input to the probabilistic oracle during HEAVY-ANN always contain a vertex from CORE, yet no core vertex is included in $\hat{\mathcal{E}}$ by definition (recall that core vertices are subsets of $\mathcal{S}^{(2)}$).

Furthermore, every query to the probabilistic oracle during HEAVY-ANN always contain one edge in $\mathcal{E}(\mathcal{S}^{(1)}, \mathcal{S}^{(1)})$ (specifically, between two heavy vertices), but none of these edges are included in $\mathcal{E}''$. Thus, the result follows. ■

We now prove correctness and bound the query complexity.

Lemma 29 *It holds that* $\textsc{Cost}^1_{\mathcal{P}}(\mathcal{V}) \leq O(1) \cdot \mathsf{OPT}^1(\mathcal{V})$*, with probability at least* $1 - \frac{1}{n^{\Omega(1)}}$.

Proof Since the total number of times ORACLE-CENTER and PROB-SORT are invoked is $O(\mathsf{polylog}\, n)$, by Lemma 28 every procedure invoked by the algorithm works correctly with high probability.

It holds that $\mathsf{OPT}^\infty(\mathcal{V}) \leq \mathsf{OPT}^1(\mathcal{V}) \leq n \cdot \mathsf{OPT}^\infty(\mathcal{V})$. Therefore, $\frac{1}{O(1)\cdot n}\mathsf{OPT}^1(\mathcal{V}) \leq \rho^\star \leq O(1) \cdot \mathsf{OPT}^\infty(\mathcal{V}) \leq O(1) \cdot \mathsf{OPT}^1(\mathcal{V})$. By definition, the diameter of $\mathcal{U}_i$ is bounded by at most $n \cdot n^3 \cdot \rho^* \leq n^5 \mathsf{OPT}^1(\mathcal{U}_i)$; and since $\mu - \lambda = O(\log n)$, the height of the tree structure $\mathcal{T}_i$ is $O(\log n)$.

Note that in any $\mathcal{U}_i$, all edges with distance at most $\frac{1}{n^5}\mathsf{OPT}^1(\mathcal{U}_i)$ can be rounded up to $\frac{8}{n^5}\mathsf{OPT}^1(\mathcal{U}_i)$ because their total contribution to the clustering cost is at most $\frac{8}{n^4}\mathsf{OPT}^1(\mathcal{U}_i)$, preserving the $O(1)$-approximation; consequently, the algorithm Alg_M returns a set $\mathcal{A}_i$ of k vertices satisfying $\textsc{Cost}^1_{\mathcal{A}_i}(\mathcal{U}_i) \leq O(1)\mathsf{OPT}^1(\mathcal{U}_i)$. Moreover, for any vertex $u_i \in \mathcal{U}_i$ and $u_j \in \mathcal{U}_j$ with $i \neq j$, it holds that $d(u_i, u_j) \geq n^3\rho^* \geq n \cdot \mathsf{OPT}^1(\mathcal{V})$, so no vertex from $\mathcal{U}_i$ can be assigned to a center in another connected component in any $O(1)$-approximation solution; thus, $\textsc{Cost}^1_{\mathcal{M}}(\mathcal{V}) \leq O(1) \cdot \mathsf{OPT}^1(\mathcal{V})$ with $|\mathcal{M}| = O(k^2 \mathsf{polylog}\, n)$, making the weighted set $\mathcal{M}$ a coreset for k-median clustering on $\mathcal{V}$ (see Galhotra et al. (2024)).

We need to argue that applying Alg_M on $\mathcal{M}$ with the modified edge weights returns a set $\mathcal{P}$ such that $\textsc{Cost}^1_{\mathcal{P}}(\mathcal{V}) \leq O(1) \cdot \mathsf{OPT}^1(\mathcal{V})$. This follows easily because (i) treating distances less than $\max_{i\in[p]}\left\{\frac{1}{n^5}\mathsf{OPT}^1(\mathcal{U}_i)\right\}$ as $\frac{16}{n^5}\mathsf{OPT}^1(\mathcal{V})$ increases the overall error by at most $\frac{16}{n^4}\mathsf{OPT}^1(\mathcal{V})$, (ii) any distance between vertices from different components can be treated as ∞ since such distances exceed $n \cdot \mathsf{OPT}^1(\mathcal{V})$, and (iii) the remaining edge weights are stretched by at most a factor of 2 (by Theorem 22).

■

Lemma 30 *Our algorithm performs* $O((n + k^2) \cdot \mathsf{polylog}\, n)$ *queries to the quadruplet oracle and* $O(\mathsf{polylog}\, n)$ *queries to the distance oracle, with probability at least* $1 - \frac{1}{n^{O(1)}}$.

Proof In Step 1, the k-center algorithm performs $O((n+k^2)\mathsf{polylog}\, n)$ queries to the quadruplet oracle and $O(\mathsf{polylog}\, n)$ queries to the distance oracle, with high probability. In Step 2, note that $\bar{m} = \mathsf{polylog}\,(n)$, so $\textsc{Filter}(\pi_{\bar{\mathcal{E}}_i}, n^3\rho^\star)$ for every $i \in [\bar{m}]$ performs $O(\mathsf{polylog}\, n)$ distance queries. Next, in Step 3, constructing all data structures $\mathcal{T}_i$ requires $O(n \cdot \mathsf{polylog}\, n)$ queries to the quadruplet oracle and $O(\mathsf{polylog}\, n)$ queries to the distance oracle; for every level at round j, across all the $\mathcal{T}_i$'s, the procedures $\textsc{Prob-Sort}(\hat{\mathcal{X}}'_j)$, $\textsc{Prob-Sort}(\hat{\mathcal{Z}}')$, $\textsc{Bucket}(\hat{\mathcal{X}}'_j, \lambda, \mu, 2)$, and $\textsc{Bucket}(\hat{\mathcal{Z}}', \lambda, \mu, 2)$ are executed, performing $O(n \cdot \mathsf{polylog}\, n)$ quadruplet queries and $O(\mathsf{polylog}\, n)$ distance queries, since $|\hat{\mathcal{X}}'_j|, |\hat{\mathcal{Z}}'| = O(n)$. Furthermore, $\textsc{Bucket}(\pi, \mu, \lambda, 2)$, for all π among $\pi_{\mathcal{X}_\ell}$, $\pi_{\mathcal{Y}_\ell}$, $\pi_{\hat{\mathcal{N}}_\ell}$, $\pi_{\mathcal{X}^*_\ell}$, and $\pi_{\mathcal{Z}_\ell}$ over $\ell \in [\tau]$, where $|\lambda - \mu| = O(\log n)$, performs $O(\mathsf{polylog}\, n)$ distance queries. In Step 5, the subroutines $\textsc{Prob-Sort}(\mathcal{E}'')$ and $\textsc{Bucket}(\pi_{\mathcal{E}''}, \lambda, \mu, 2)$ require $O(k^2\mathsf{polylog}\, n)$ queries to the quadruplet oracle in total (since $|\mathcal{S}^{(1)}| = k$ and $|\mathcal{A}_i| = k$) and $O(\mathsf{polylog}\, n)$ queries to the distance oracle, while the subroutines $\textsc{Prob-Sort}(\hat{\mathcal{E}})$ and $\textsc{Bucket}(\pi_{\hat{\mathcal{E}}}, \lambda, \mu, 2)$ require $O(k^2\mathsf{polylog}\, n)$ quadruplet queries in total because for every $\ell_1 < \ell_2 \in [\tau]$, $|\mathcal{E}(\mathcal{S}^{(1)}_{\ell_1}, \mathcal{S}^{(1)}_{\ell_2})| = O(k^2)$ and there are $O(\mathsf{polylog}\, n)$ phases in the k-center algorithm (i.e., $\tau = O(\mathsf{polylog}\, n)$), and $O(\mathsf{polylog}\, n)$ distance queries.

■

We conclude with the next theorem.

Theorem 31 *There exists an algorithm* ORACLE-MEDIAN *such that, given a metric space* $\Sigma = (\mathcal{V}, d)$ *of size* $|\mathcal{V}| = n$ *with bounded doubling dimension, that supports a probabilistic quadruplet oracle with error* $\varphi \leq \frac{1}{16}$ *and a distance oracle, the algorithm* ORACLE-MEDIAN(Σ) *returns, with high probability, a set* $\mathcal{A} \subseteq \mathcal{V}$ *of size* k *such that* $\textsc{Cost}^1_{\mathcal{A}}(\mathcal{V}) \leq O(1) \cdot \textsf{OPT}^1$*, using* $O((n+k^2) \cdot \textsf{polylog}\, n)$ *queries to the quadruplet oracle and* $O(\textsf{polylog}\, n)$ *queries to the distance oracle. The same guarantees hold for the* k*-means* OracleCluster *problem.*

D.2. Extension to general metric spaces

The result follows by substituting our algorithm for the k-center OracleCluster problem on general metric spaces (Theorem 21) into a modified version of the algorithm for the k-median OracleCluster problem presented in Galhotra et al. (2024) for general metric spaces, which in turn is an adaptation of the approaches in Har-Peled and Mazumdar (2004); Mettu and Plaxton (2002). We provide a rough sketch.

First, compute a k-center clustering to estimate the k-median cost. Let $\rho^\star$ denote the estimated k-center cost. Next, process the vertices in $\mathcal{V}$ in rounds to construct a k-median coreset $\mathcal{M}$. For a round $i \leq O(\textsf{polylog}\, n)$, let $\mathcal{V}_i$ be the set of vertices processed in round i, starting with $\mathcal{V}_1 = \mathcal{V}$. Sample a set of $O(k \cdot \textsf{polylog}\, n)$ vertices $\mathcal{S}_i$.

While Galhotra et al. (2024) execute a subroutine $\textsf{ANN}(\mathcal{S}_i, \mathcal{V}_i)$ to find an approximate nearest neighbor from every vertex in $\mathcal{V}_i$ to those in $\mathcal{S}_i$, we skip this step. Instead, we define $\mathcal{E}' = \mathcal{E}(\mathcal{S}_i, \mathcal{V}_i)$ and compute the ordering $\pi_{\mathcal{E}'} = \textsc{prob-sort}(\mathcal{E}')$. We then use $\textsc{filter}(\pi_{\mathcal{E}'}, 2^j \cdot \frac{\rho^\star}{n^2})$ to partition the edges into buckets, for $j \leq O(\log n)$. These buckets allow us to estimate the nearest neighbor distance of each vertex in $\mathcal{V}_i \setminus \mathcal{S}_i$ within an additive error of $\frac{\rho^\star}{n^2}$.

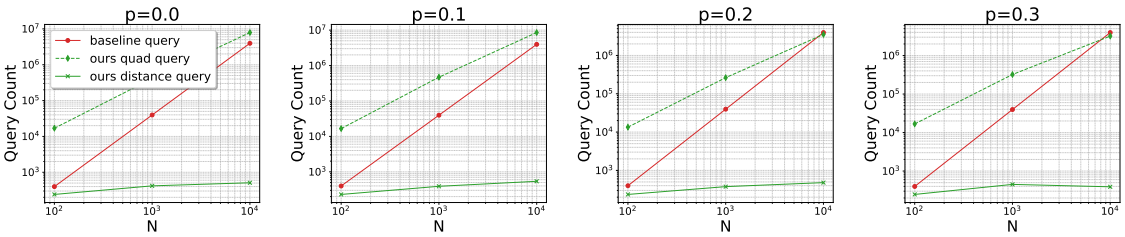
Using these proxy distances, we identify the safe vertices as in Galhotra et al. (2024). At the end of this process, we obtain a set $\mathcal{Q}$ of $O(k\, \textsf{polylog}\, n)$ vertices, which forms a coreset for the k-median OracleCluster problem on $\mathcal{V}$. Let $\bar{\mathcal{E}} = \mathcal{E}(\mathcal{Q}, \mathcal{Q})$. We compute a low-dislocation decomposition of $\bar{\mathcal{E}}$ into $O(\textsf{polylog}\, n)$ components and bucketize each component as above.

Finally, we run a standard k-median algorithm on $\mathcal{Q}$ using the bucketed edge weights. As in the bounded doubling dimension case, we must ensure that no quadruplet queries overlap with the algorithm used to compute the initial k-center solution. We skip the details of this step for brevity. Putting everything together we get the following.

Theorem 32 *There exists an algorithm such that, given a metric space* $\Sigma = (\mathcal{V}, d)$ *of size* $|\mathcal{V}| = n$*, that supports a probabilistic quadruplet oracle with error* $\varphi \leq \frac{1}{16}$ *and a distance oracle, the algorithm returns, with high probability, a set* $\mathcal{A} \subseteq \mathcal{V}$ *of size* k *such that* $\textsc{Cost}^1_{\mathcal{A}}(\mathcal{V}) \leq O(1) \cdot \textsf{OPT}^1$*, using* $O(nk\, \textsf{polylog}\, n)$ *queries to the quadruplet oracle and* $O(\textsf{polylog}\, n)$ *queries to the distance oracle. The same guarantees hold for the* k*-means* OracleCluster *problem.*

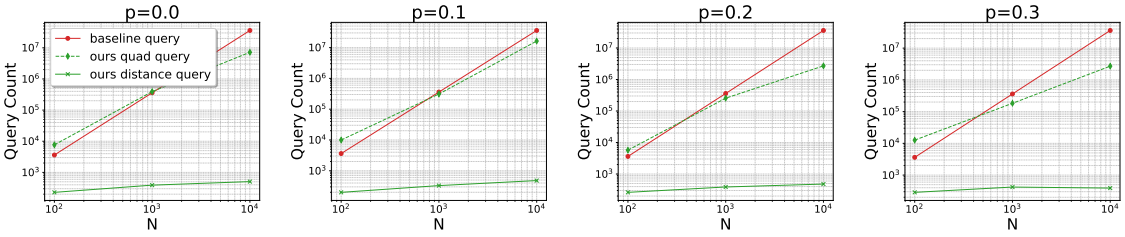
Appendix E. Experiments

In this section, we present preliminary experimental results based on a basic implementation of CONSTRUCT-NET. Specifically, we use CONSTRUCT-NET to construct an approximate distance structure $\mathbf{D}$ (see Section C.1). We then evaluate the quality of solutions to the k-center, k-median, and Minimum Spanning Tree (MST) problems by running standard algorithms on distances from $\mathbf{D}$, and compare them against baselines obtained by applying the same algorithms to ground-truth distances.



(a) (b) (c) (d)

Figure 1: Query count on synthetic dataset with $cluster\_ratio = 0.02$ and baseline k-center.



(a) (b) (c) (d)

Figure 2: Query count on synthetic dataset with $cluster\_ratio = 0.2$ and baseline k-center.

Datasets. We conduct experiments on both synthetic and real-world datasets.

The synthetic dataset consists of N nodes distributed in the 2-dimensional Euclidean space $[0, 1]^2$, where nodes are assigned to k uniformly distributed cluster centers and their positions are perturbed by Gaussian noise ($\sigma = 0.1$). We consider $N = \{10^2, 10^3, 10^4\}$, and define the cluster ratio, $cluster\_ratio$, as the ratio of k to n. In our experiments, we set $cluster\_ratio = \{0.02, 0.2\}$.

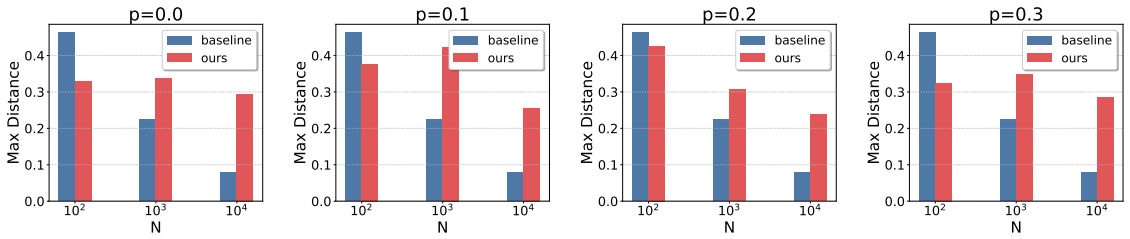
For the real-world dataset, we use the Adult dataset Asuncion et al. (2007), sampling $N = 10{,}000$ entries that describe individuals' income and education attributes. The dimensionality of this dataset is $d = 6$.

Implementation. We do not implement CONSTRUCT-NET exactly and omit steps specifically designed to handle persistence. Furthermore, while the original version uses PROB-SORT and BUCKET to partition edges into buckets, we simplify this step using a sampling-based heuristic.

Specifically, in round i, when identifying edges within a target distance, the approximation process begins by sampling 30 edges from the edge set to be sorted and computing their distances using a distance oracle. Edges with distances less than 2^{i-2} are retained. From these, the top half is selected. Each edge in the original set is then compared against this subset using the quadruplet oracle. An edge passes the filter if it is deemed smaller than at least half of the selected edges.

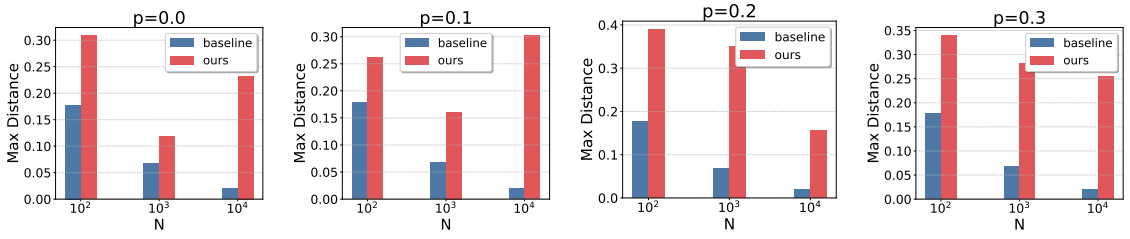
Once $\mathbf{D}$ is built we implement the query procedure as described in the original algorithm.

Results on Synthetic Dataset. We primarily focus on the k-center objective and use GONZALEZ's k-center algorithm as the standard. In this experiment, we vary the number of samples, $N = \{10^2, 10^3, 10^4\}$, and the noise level in the quadruplet oracle, $p = \{0.0, 0.1, 0.2, 0.3\}$. Additionally, we adjust the , $cluster\_ratio = \{0.02, 0.2\}$, to examine how the number of clusters affects query usage.



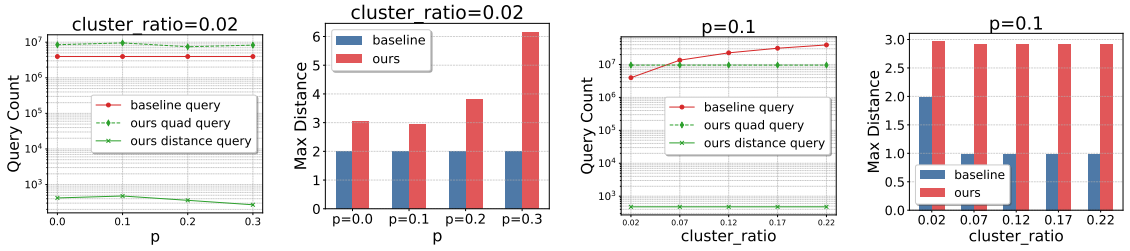
(a) (b) (c) (d)

Figure 3: Clustering result on synthetic dataset with $cluster\_ratio = 0.02$ and baseline k-center.



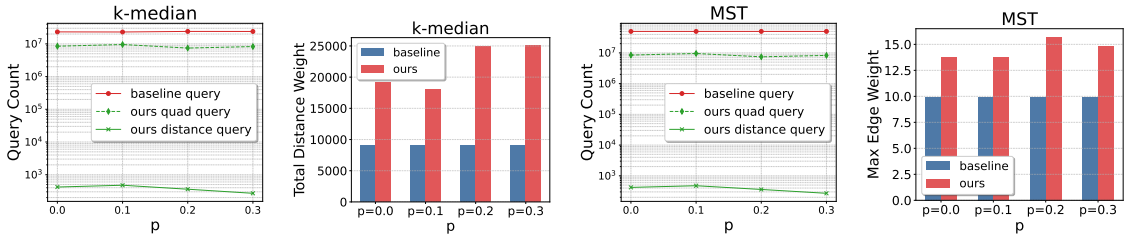
(a) (b) (c) (d)

Figure 4: Clustering result on synthetic dataset with $cluster\_ratio = 0.2$ and baseline k-center.



(a) (b) (c) (d)

Figure 5: k-center on Adult Dataset.



(a) (b) (c) (d)

Figure 6: k-median and MST on Adult dataset.

For query counts, we report the number of distance queries used by the baseline, along with the number of quadruplet and distance queries used to construct $\mathbf{D}$ (Figures 1 and 2). From the

figures, we observe that as N increases, the number of hard queries (i.e., distance oracle calls) used in our method remains stable, while those in the k-center baseline increase rapidly. Although the number of quadruplet queries in our method can sometimes exceed the total queries used by the baseline (especially for small k), these queries are simple comparisons and significantly cheaper, making them more practical in large-scale settings. Moreover, since the number of hard queries in the baseline scales as Nk, we observe a steep increase in its query count as $cluster\_ratio$ grows from 0.02 to 0.2, while both quadruplet and distance queries in our method remain stable. This highlights the scalability advantage of our approach.

To evaluate clustering performance, we compare the maximum distances returned by the baseline versus those using our approximate distances (Figures 3 and 4). Our method achieves results comparable to the baseline and, in some cases, even outperforms it. In summary, our approach is both *efficient*, requiring far fewer hard queries, and *effective*, delivering competitive clustering performance.

Results on real-world dataset. Again, we primarily focus on the k-center objective. We first examine how the noise level p affects performance by fixing $cluster\_ratio = 0.02$ and varying $p = \{0.0, 0.1, 0.2, 0.3\}$ (Figures 5(a), (b)). Next, we investigate the impact of the number of clusters by fixing $p = 0.1$ and varying $cluster\_ratio = \{0.02, 0.07, 0.12, 0.17, 0.22\}$ (Figures 5(c), (d)). In addition to k-center, we also evaluate the performance of k-median and MST objectives (Figure 6), using LLOYD'S algorithm for k-median and PRIM'S algorithm for MST as the standard.

From Figures 5(a), (c), we observe that the query count in our algorithm remains unaffected by both the noise level and the number of clusters, whereas the k-center baseline is sensitive to the number of clusters—consistent with our findings for the synthetic data. For the approximation quality (Figures 5(b), (d)), we observe trends similar to those seen in the synthetic dataset. When combined with other clustering objectives, as shown in Figure 6, our method continues to demonstrate robustness to noise and requires only a small number of hard queries, highlighting its generalizability and scalability across different graph-based clustering tasks.