How to Design Robust Algorithms using Noisy Comparison Oracles

Raghavendra Addanki  
UMass Amherst  
raddanki@cs.umass.edu

Sainyam Galhotra  
UMass Amherst  
sainyam@cs.umass.edu

Barna Saha  
UC Berkeley  
barnas@berkeley.edu

ABSTRACT
Metric based comparison operations such as finding maximum, nearest neighbor are fundamental to studying various clustering techniques such as $k$-center clustering and agglomerative hierarchical clustering. These techniques crucially rely on accurate estimation of pairwise distance between records. However, computing exact features of the records, and their pairwise distances is often challenging, and sometimes not possible. We circumvent this challenge by leveraging weak supervision in the form of a comparison oracle that compares the relative distance between the queried points such as 'Is point $u$ closer to $v$ or $w$ closer to $x$?'

However, it is possible that some queries are easier to answer than others using our comparison oracle. We capture this by introducing two different noise models called adversarial and probabilistic noise. In this paper, we study various problems that include finding maximum, nearest/farthest neighbor search under these noise models. Building upon the techniques we develop for these comparison operations, we give robust algorithms for $k$-center clustering and agglomerative hierarchical clustering. We prove that our algorithms achieve good approximation guarantees with a high probability and analyze their query complexity. We evaluate the effectiveness and efficiency of our techniques empirically on various real-world datasets.

1 INTRODUCTION
Many real world applications such as data summarization, social network analysis, facility location crucially rely on metric based comparative operations such as finding maximum, nearest neighbor search or ranking. As an example, data summarization aims to identify a small representative subset of the data, where each representative can be considered as a summary of similar records in the dataset. Popular clustering algorithms such as $k$-center clustering and hierarchical clustering are often used for data summarization [23, 34]. In this paper, we study fundamental metric based operations such as finding maximum, nearest neighbor search, and use the developed techniques to study clustering algorithms such as $k$-center clustering and agglomerative hierarchical clustering.

Clustering is often regarded as a challenging task especially due to the absence of domain knowledge, and the final set of clusters identified can be highly inaccurate and noisy [6]. It is often hard to compute the exact features of points and thus pairwise distance computation from these feature vectors could also be highly noisy. This will render the clusters computed based on objectives such as $k$-center unreliable.

To address these challenges, there has been a recent interest to leverage supervision from crowd workers (abstracted as an oracle) which provides significant improvement in accuracy but at an added cost incurred by human intervention [18, 47, 48]. For clustering, the existing literature on oracle based techniques mostly use optimal cluster queries, that ask questions of the form ‘do the points $u$ and $v$ belong to the same optimal cluster?’[5, 16, 37, 48]. The goal is to minimize the number of queries aka query complexity while ensuring high accuracy of clustering output. This model is relevant for applications where the oracle (human expert or a crowd worker) is aware of the optimal clusters such as in entity resolution [18, 47]. However, in most applications, the clustering output highly depends on the required number of clusters and the presence of other records. Without a global view of the entire dataset, answering optimal queries may not be feasible for any realistic oracle. Let us consider an example data summarization task that highlights some of the challenges involved.

**Example 1.1.** Consider a data summarization task over a collection of images (shown in Figure 1). The goal is to identify $k$ images (say $k = 3$) that summarize the different locations in the dataset. The images $1, 2$ refer to Eiffel tower in Paris, $3$ is the Colosseum in Rome, $4$ is the replica of Eiffel tower at Las Vegas, USA, $5$ is Venice and $6$ is the Leaning tower of Pisa. The ground truth output in this case would be $\{(1, 2), (3, 5, 6), (4)\}$. We calculated pairwise similarity between images using the visual features generated from Google Vision API[1]. The pair $(1, 4)$ exhibits the highest similarity of 0.87, while all other pairs have similarity lower than 0.85. Distance between a pair of images $u$ and $v$, denoted as $d(u, v)$, is defined as $1-$similarity between $u$ and $v$.

In this example, we make the following observations.

- **Automated clustering techniques generate noisy clusters.** Consider the greedy approach for $k$-center clustering [24] which sequentially identifies the farthest record as a new cluster center. In this example, records 1 and 4 are placed in the same cluster by the greedy $k$-center clustering technique, thereby leading to poor performance.
We present new algorithms for finding the maximum, nearest and farthest neighbor is easy. As answering queries of the form 'Is 1 closer to 2, or 5 is closer to 6?' does not require any extra knowledge about other records in the dataset and can be answered in isolation.

Motivated by these observations, we consider a quadruplet comparison oracle that compares the relative distance between two different pairs of points $(u_1, u_2)$ and $(v_1, v_2)$ and outputs the pair with smaller distance between them. Such oracle models have been studied extensively in the literature. Even though quadruplet comparison queries are easier than binary optimal queries, some oracle queries maybe harder than the rest. For example, some oracle queries are difficult for humans and can generate erroneous answers. In a comparison query, if there is a significant gap between the two distances being compared, then such queries are easier to answer. However, when the two distances are close, the chances of an error could increase. For example, 'Is location in image 1 closer to 3, or 2 is closer to 6?' maybe difficult to answer.

To capture noise in quadruplet comparison oracle answers, we consider two noise models. In the first noise model, when the pairwise distances are comparable, the oracle can return the pair of points that are farther instead of closer. Moreover, we assume that the oracle has access to all previous queries and can answer queries by acting adversarially. More formally, there is a parameter $\mu > 0$ such that if $\frac{\max d(u_1, u_2), d(v_1, v_2)}{\min d(u_1, u_2), d(v_1, v_2)} \leq (1 + \mu)$, then adversarial error may occur, otherwise the answers are correct. We call this 'Adversarial Noise Model'.

In the second noise model called 'Probabilistic Noise Model', given a pair of distances, we assume that the oracle answers correctly with a probability of $1 - p$ for some fixed constant $p < \frac{1}{2}$. We consider a persistent probabilistic noise model, where our oracle answers are persistent i.e., query responses remain unchanged even upon repeating the same query multiple times. Such noise models have been studied extensively. However, due to oracles often does not change with repetition, and in fact, sometimes increases upon repeated querying. This is in contrast to the noise models studied in [16] where response to every query is independently noisy. Persistent query models are much difficult to handle than independent query models where repeating each query is sufficient to generate the correct answer by majority voting.

1.1 Our Contributions

We present new algorithms for finding the maximum, nearest and farthest neighbors, k-center clustering and hierarchical clustering objectives under the adversarial and probabilistic noise model using comparison oracle. We show that our techniques minimize query complexity, are highly efficient and have provable approximation guarantees for both the noise models. For all the mentioned problems, we empirically evaluate the robustness and efficiency of our techniques on four real world data sets.

(i) **Maximum, Farthest and Nearest Neighbor**: Finding maximum has received significant attention under both adversarial and probabilistic model [3, 8, 14, 17, 19–21, 33]. In this paper, we provide the following results.

(ii) **Maximum under adversarial model**: We present an algorithm that returns a value within $(1 + \mu)^3$ of the maximum among a set of $n$ values $V$, with probability $1 - \delta$ using $O(n \log^2 (1/\delta))$ oracle queries and running time (Theorem 3.5).

(iii) **Maximum under probabilistic model**: We present an algorithm (Algorithm 12) that requires $O(n \log^2 (n/\delta))$ queries to identify $O(\log^2 (n/\delta))$th rank value with probability $1 - \delta$ (Theorem 3.6). That is, in $O(n \log^2 (n))$ time we can identify $O(\log^2 (n))$th value with probability $1 - \frac{1}{n}$ for any constant $c$.

To contrast our results with the state of the art, Ajtai et al. [3] study a slightly different additive adversarial error model, where the answer of a maximum query is correct if the compared values differ by $\theta$ (for some $\theta > 0$) and otherwise the oracle answers adversarially. Under this setting, they give an additive $3\theta$-approximation with $O(n)$ queries. Although, our model cannot be directly compared with theirs, we note that our model is scale invariant, and thus, provides a much stronger bound when distances are small. As a consequence, our algorithm can be used under additive adversarial model as well providing the same approximation guarantees (Theorem 3.8).

For the probabilistic model, after a long series of works [8, 19, 21, 33], only recently an algorithm is proposed with query complexity $O(n \log n)$ that returns an $O(\log n)$th rank value with probability $1 - \frac{1}{n}$ [20]. Previously, the best query complexity was $O(n^{3/2})$ [21]. While our bounds are slightly worse than [20], our algorithm is significantly simpler.

Rest of the work in finding maximum allow repetition of queries and assume the answers are independent [14, 17]. As discussed earlier, persistent errors are much more difficult to handle than independent errors. In [17], the authors present an algorithm that finds maximum using $O(n \log 1/\delta)$ queries and succeeds with probability $1 - \delta$. Therefore, even under persistent errors, we obtain guarantees close to the existing ones which assume independent error. The algorithms of [14, 17] do not extend to our model.

(ii) **Nearest Neighbor**: Nearest neighbor queries can be cast as "finding minimum" among a set of distances. As such, we get the same bound as finding maximum for the nearest neighbor queries. In particular, under the adversarial model, we obtain an $(1 + \mu)^3$-approximation, and in the probabilistic model, we are guaranteed to return an element within top $O(\log^2 n)$ with $O(n \log^2 (1/\delta))$ and $O(n \log^2 n)$ oracle queries respectively.

Prior techniques have studied nearest neighbor search under noisy distance queries [36], where the oracle returns a noisy estimate of a distance between queried points, and repetitions are allowed. Neither the algorithm of [36], nor other techniques developed for maximum [3, 17] and top-$k$ [14] extend for nearest neighbor under our noise models as already discussed.

(iii) **Farthest Neighbor**: Similarly, the farthest neighbor query can be cast as finding maximum among a set of distances, and the results for computing max extends to this setting. However, computing the farthest neighbor is one of the basic primitives for more complex tasks like k-center clustering, and for that the existing bounds under
the probabilistic model that may return an $O((\log n)\text{th rank element})$ is insufficient. We show exploiting triangle inequality that distances on a metric space satisfies that we get a constant approximation to the nearest neighbor under the probabilistic model and under a mild distribution assumption (Theorem 3.8).

(ii) $k$-center Clustering: $k$-center clustering is one of the fundamental models of clustering and is extremely well-studied [44, 49].

- $k$-center under adversarial model We design algorithm that returns a clustering that is a 2 + $\mu$ approximation, for small values of $\mu$ with probability $1 - \delta$ using $O(nk^2 + nk \log^2 (k/\delta))$ queries (Theorem 4.2). In contrast, even when all the distances are known accurately, $k$-center cannot be approximated better than a 2-factor unless $P = NP$ [44]. Therefore, under the adversarial model, we achieve near-optimal results.

- $k$-center under probabilistic noise model For probabilistic noise, when optimal $k$-center clusters are of size at least $O(\sqrt{n})$, our algorithm returns a clustering that achieves constant approximation with probability $1 - \delta$ using $O(nk \log^2 (n/\delta))$ queries (Theorem 4.3).

To the best of our knowledge, even though $k$-center clustering is an extremely popular and basic clustering paradigm, it hasn’t been studied under the comparison oracle model, and we provide the first results in this domain.

(iii) Single Linkage and Complete Linkage—Agglomerative Hierarchical Clustering: Under adversarial noise, we show a clustering technique that loses only a multiplicative factor of $(1 + \mu)$ in each merge operation and has an overall query complexity of $O(n^2)$ (Theorem 5.2). Prior work [22] consider comparison oracle queries to perform average linkage under a different noise model in which the unobserved pairwise similarities are generated according to a normal distribution. These techniques do not extend to our noise models.

1.2 Other Related Work

For finding the maximum among a given set of values, it is known that techniques based on tournament obtain optimal guarantees and are widely used [14]. For the problem of finding nearest neighbor, techniques based on locality sensitive hashing generally perform well in practice and are widely adopted [4]. Clustering points using $k$-center objective is NP-hard and there are many well known heuristics and approximation algorithms [49] with the classical greedy algorithm achieving an approximation ratio of 2. All these techniques are not applicable when pairwise distances are unknown. As distances between points cannot always be accurately estimated, many recent techniques leverage supervision in the form of an oracle. Most oracle based clustering frameworks consider ‘optimal cluster’ queries [12, 25, 28, 37, 38] to identify ground truth clusters. Recent techniques for distance based clustering objectives, such as $k$-means [5, 11, 31, 32] and $k$-median [2] use optimal cluster queries in addition to distance information for obtaining better approximation guarantees. As ‘optimal cluster’ queries can be costly or sometimes infeasible, there has been recent interest in leveraging distance based comparison oracles for other problems, similar to our quadruplet oracles [16, 22].

Distance based comparison oracles have been used to study a wide range of problems and we list a few of them - learning fairness metrics [29], top-down hierarchical clustering with a different objective [10, 16, 22], correlation clustering [43] and classification [27, 42], identify maximum [26, 45], top-$k$ elements [13–15, 33, 35, 39], information retrieval [30], skyline computation [46] and several other machine learning tasks. To the best of our knowledge, there is no work that considers quadruplet comparison oracle queries to perform $k$-center clustering and single/complete linkage based hierarchical clustering.

Closely related to finding maximum, sorting has also been well studied under various comparison oracle based noise models [7, 8]. The work of [14] considers a different probabilistic noise model with error varying as a function of the values but they assume that each oracle query is independent and therefore repetition can help boost the probability of success. Using a quadruplet oracle, [22] study the problem of recovering hierarchical clustering under a planted noise model and is not applicable for single linkage.

2 PRELIMINARIES

Let $V = \{v_1, v_2, \ldots, v_n\}$ be a collection of $n$ records such that each record maybe associated with a value $val(v_i), v_i \in [1, n]$. We assume that there exists a total ordering over the values of elements in $V$. For simplicity we denote the value of record $v_i$ as $v_i$ instead of $val(v_i)$ whenever it is clear from the context.

Given this setting, we consider a comparison oracle that compares the value of any pair of records ($v_i, v_j$) and outputs Yes if $v_i \leq v_j$ and No otherwise.

**Definition 2.1 (Comparison Oracle).** An oracle is a function $O : V \times V \rightarrow \{\text{Yes}, \text{No}\}$. Each oracle query considers two values as input and outputs $O(v_i, v_j) = \text{Yes}$ if $v_i \leq v_j$ and $\text{No}$ otherwise.

Given this oracle setting, we define the problem of identifying the maximum over the records $V$.

**Problem 2.2 (Maximum).** Given a collection of $n$ records $V = \{v_1, \ldots, v_n\}$ and access to a comparison oracle $O$, identify the $\arg\max_{v_i \in V} v_i$ with minimum number of queries to the oracle.

As a natural extension, we can also study the problem of identifying the record corresponding to the smallest value in $V$.

2.1 Quadruplet Oracle Comparison Query

In applications that consider distance based comparison of records like nearest neighbor identification, the records $V = \{v_1, \ldots, v_n\}$ are generally considered to be present in a high-dimensional space along with a distance $d : V \times V \rightarrow \mathbb{R}^d$ defined over pairs of records. We assume that the embedding of records in latent space is not known, but there exists an underlying ground truth [4]. Prior techniques mostly assume complete knowledge of accurate distance metric and are not applicable in our setting. In order to capture the setting where we can compare distances between pair of records, we define quadruplet oracle below.

**Definition 2.3 (Quadruplet Oracle).** An oracle is a function $O : V \times V \times V \times V \rightarrow \{\text{Yes}, \text{No}\}$. Each oracle query considers two pairs of records as input and outputs $O(v_1, v_2, v_3, v_4) = \text{Yes}$ if $d(v_1, v_2) \leq d(v_3, v_4)$ and $\text{No}$ otherwise.
The quadruplet oracle is equivalent to the comparison oracle discussed before with a difference that the two values being compared are associated with pair of records as opposed to individual records.

Given this oracle setting, we define the problem of identifying the farthest record over \( V \) with respect to a query point \( q \) as follows.

**Problem 2.4 (Farthest Point).** Given a collection of \( n \) records \( V = \{ v_1, \ldots, v_n \} \), a query record \( q \) and access to a quadruplet oracle \( O \), identify arg max \( v_i \in V \setminus \{ q \} \) \( d(q, v_i) \).

Similarly, the nearest neighbor query returns a point that satisfies arg min \( v_i \in V \setminus \{ q \} \) \( d(q, v_i) \). Now, we formally define the \( k \)-center clustering problem.

**Problem 2.5 (\( k \)-Center Clustering).** Given a collection of \( n \) records \( V = \{ v_1, \ldots, v_n \} \) and access to a comparison oracle \( O \), identify \( k \) centers (say \( S \subseteq V \)) and a mapping of records to corresponding centers, \( \pi : V \rightarrow S \), such that the maximum distance of any record from its center, i.e., \( \max_{v_i \in V} d(v_i, \pi(v_i)) \) is minimized.

We assume that the points \( v_i \in V \) exist in a metric space and the distance between any pair of points is not known. We denote the unknown distance between any pair of points \( (v_i, v_j) \) where \( v_i, v_j \in V \) as \( d(v_i, v_j) \) and use \( k \) to denote the number of clusters. Optimal clusters are denoted as \( C_k \) with \( C_k(v_i) \subseteq V \) denoting the set of points belonging to the optimal cluster containing \( v_i \). Similarly, \( C(v_i) \subseteq V \) refers to the nodes belonging to the cluster containing \( v_i \) for any clustering given by \( C(\cdot) \).

In addition to the \( k \)-center clustering, we study single linkage and complete linkage—agglomerative clustering techniques, where the distance metric over the records is not known a priori. These techniques initialize each record \( v_i \) in a separate singleton cluster and sequentially merge the pair of clusters having the least distance between them. In case of single linkage, the distance between two clusters \( C_1 \) and \( C_2 \) is characterized by the closest pair of records defined below.

\[
d_{SL}(C_1, C_2) = \min_{v_i \in C_1, v_j \in C_2} d(v_i, v_j)
\]

In complete linkage, the distance between a pair of clusters \( C_1 \) and \( C_2 \) is calculated by identifying the farthest pair of records, \( d_{CL}(C_1, C_2) = \max_{v_i \in C_1, v_j \in C_2} d(v_i, v_j) \).

### 2.2 Noise Models

The oracle models discussed in Problems 2.2, 2.4 and 2.5 assume that the oracle answers every comparison query correctly. In real-world applications, however, the answers can be wrong which can lead to noisy results. To formalize the notion of noise, we consider two different models. First, adversarial noise model considers a setting where a comparison query can be adversarially wrong if the two values being compared are within a multiplicative factor of \( 1 + \mu \) for some constant \( \mu > 0 \).

\[
O(v_1, v_2) = \begin{cases} 
\text{Yes, if } v_1 \leq \frac{1}{1+\mu} v_2 \\
\text{No, if } v_1 \geq (1 + \mu) v_2 \\
\text{adversarially incorrect if } \frac{1}{1+\mu} \leq \frac{v_1}{v_2} \leq (1 + \mu)
\end{cases}
\]

The parameter \( \mu \) corresponds to the degree of error. For example, \( \mu = 0 \) implies a perfect oracle. The model extends to quadruplet oracle as follows.

\[
O(v_1, v_2, v_3, v_4) = \begin{cases} 
\text{Yes, if } d(v_1, v_2) \leq \frac{1}{1+\mu} d(v_3, v_4) \\
\text{No, if } d(v_1, v_2) \geq (1 + \mu) d(v_3, v_4) \\
\text{adversarially incorrect if } \frac{1}{1+\mu} d(v_1, v_2) \leq d(v_3, v_4) \leq (1 + \mu)
\end{cases}
\]

The second model considers a probabilistic noise model where each comparison query is incorrect independently with a probability \( p < 1 \). In this oracle model, asking the same query yields the same response and therefore asking a query multiple times does not help to reduce the chances of error.

### 3 FINDING MAXIMUM

In this section, we present robust algorithms to identify the record corresponding to a maximum value in \( V \) under the adversarial noise model and the probabilistic noise model. Later we extend the algorithms to find the farthest and the nearest neighbor.

#### 3.1 Adversarial Noise

Consider a trivial approach that maintains a running maximum value while sequentially processing the records, i.e., if a larger value is encountered, the current maximum value is updated to the larger value. This approach requires \( n - 1 \) comparisons. However, in the presence of adversarial noise, our output can have a significantly lower value compared to the correct maximum. In general, if \( v_{\max} \) is the true maximum of \( V \), then the above approach can return an approximate maximum whose value could be as low as \( \frac{v_{\max}}{(1+\mu)^{n-1}} \).

To see this, assume \( v_1 = 1 \) and \( v_i = (1 + \mu) - \epsilon \) where \( \epsilon > 0 \) is very close to 0. It is possible that while comparing \( v_1 \) and \( v_{i+1} \), the oracle returns \( v_i \) as the larger element. If this mistake is repeated for every \( i \), then \( v_n \) will be declared as the maximum element whereas the correct answer is \( v_n \approx v_1 (1 + \mu)^{n-1} \).

To improve upon this naive strategy, we introduce a natural **keeping score** based idea where given a set \( S \subseteq V \) of records, we maintain \( \text{Count}(v, S) \) that is equal to the number of values smaller than \( v \) in \( S \).

\[
\text{Count}(v, S) = \sum_{x \in S \setminus \{v\}} 1\{O(v, x) = \text{No}\}
\]

It is easy to observe that when the oracle makes no mistakes, \( \text{Count}(s_{\max}, S) = |S| - 1 \) and obtains the highest score, where \( s_{\max} \) is the maximum value in \( S \). Using this observation, in Algorithm 1, we output the value with the highest Count score. Given a set of records \( V \), we show in Lemma 3.1 that \( \text{Count-Max}(V) \) obtained using Algorithm 1 always returns a good approximation of the maximum value in \( V \).

**Algorithm 1 Count-Max(S) : finds the maximum value by counting in \( S \)**

1. **Input :** A set of values \( S \)
2. **Output :** An approximate maximum value of \( S \)
3. for \( v \in S \) do
4. Calculate \( \text{Count}(v, S) \)
5. \( u_{\max} \leftarrow \arg \max_{v \in S} \text{Count}(v, S) \)
6. **return** \( u_{\max} \)
Lemma 3.1. Given a set of values \( V \) with maximum value \( v_{\max} \), \textsc{Count-Max}(\( V \)) returns a value \( u_{\max} \) where \( u_{\max} \geq v_{\max}/(1 + \mu)^2 \) using \( O(|V|^2) \) oracle queries.

From Lemma 3.1, we have that \( O(n^2) \) oracle queries where \( |V| = n \), are required to get \( (1 + \mu)^2 \) approximation. In order to improve the query complexity from quadratic in \( n \), we use a tournament to obtain the maximum. Algorithm 2 presents pseudo code of the approach that takes values \( V \) as input and outputs an approximate maximum value. It constructs a balanced \( \lambda \)-ary tree \( T \) containing \( |V| \) leaf nodes such that a random permutation of the set of values \( V \) is assigned to the leaves of \( T \). In a tournament, the internal nodes of \( T \) are processed bottom-up such that at every internal node \( w \), we assign the value that is largest among the children of \( w \). To identify the largest value, we calculate \( \arg\max_{v \in \text{children}(w)} \text{Count}(v, \text{children}(w)) \) at the internal node \( w \), where \( \text{Count}(v, X) \) refers to the number of elements in \( X \) that are considered smaller than \( v \). Finally, we return the value at the root of \( T \) as our output.

Algorithm 2 \textsc{Tournament} : finds the maximum value using a tournament tree

1. **Input**: Set of values \( V \), Degree \( \lambda \)
2. **Output**: An approximate maximum value \( u_{\max} \)
3. Construct a balanced \( \lambda \)-ary tree \( T \) with \(|V|\) nodes as leaves.
4. Let \( \pi \) be a random permutation of \( V \) assigned to leaves of \( T \).
5. for \( i = 1 \) to \( \log_\lambda |V| \) do
6. \hspace{1em} for internal node \( w \) at level \( \log_\lambda |V| - i \) do
7. \hspace{2em} Let \( U \) denote the children of \( w \).
8. \hspace{2em} Set the internal node \( w \) to \( \arg\max_{v \in U} \text{Count}(v, U \setminus \{v\}) \)
9. \hspace{1em} \( u_{\max} \leftarrow \) value at root of \( T \)
10. **return** \( u_{\max} \)

In Lemma 3.2, we show that Algorithm 2 returns a value that is a \( (1 + \mu)^2 \log_\lambda n \) multiplicative approximation of the maximum value.

Lemma 3.2. Suppose \( v_{\max} \) is the maximum value among the set of records \( V \). Algorithm 2 outputs a value \( u_{\max} \) such that \( u_{\max} \geq \frac{v_{\max}}{(1 + \mu)^2 \log_\lambda n} \) using \( O(n \lambda) \) oracle queries.

According to Lemma 3.2, Algorithm 2 identifies a constant approximation when \( \lambda = \Theta(n) \), \( \mu \) is a fixed constant and has a query complexity of \( \Theta(n^2) \). By reducing the degree of the tournament tree from \( \lambda \) to 2, we can achieve \( \Theta(n) \) query complexity, but with a worse approximation ratio of \( (1 + \mu)^2 \log_\lambda n \). The idea of using a tournament for finding maximum has been studied in the past [14, 17].

Now, we describe our main algorithm (Algorithm 4) that uses the above observation to improve the overall query complexity.

Observation 3.3. At an internal node \( w \in T \), the identified maximum is incorrect only if there exists some children \( x \in \text{children}(w) \) that is very close to the true maximum (say \( w_{\max} \)), i.e., \( w_{\max}/(1 + \mu) \leq x \leq (1 + \mu)w_{\max} \).

Based on the above observation, our Algorithm \textsc{Max-Adv} uses two steps to identify a good approximation of \( v_{\max} \). Consider the case when there are a lot of values that are close to \( v_{\max} \). In Algorithm \textsc{Max-Adv}, we use a subset \( \bar{V} \subseteq V \) of size \( \sqrt{n} \) obtained using uniform sampling with replacement. We show that using a sufficiently large subset \( \bar{V} \), obtained by sampling, we ensure that at least one value that is closer to \( v_{\max} \) is in \( \bar{V} \), thereby giving a good approximation of \( v_{\max} \).

In order to handle the case when there are only a few values closer to \( v_{\max} \), we divide the entire data set into \( l = \sqrt{n} \) disjoint parts (for a suitable choice of \( l \)) and run the \textsc{Tournament} algorithm with degree \( \lambda = 2 \) on each of these parts separately (Algorithm 3). As there are very few points close to \( v_{\max} \), the probability of comparing any such value with \( v_{\max} \) is small, and this ensures that in the partition containing \( v_{\max} \), \textsc{Tournament} returns \( v_{\max} \). We collect the maximum values returned by Algorithm 2 from all the partitions and include these values in \( T \) in Algorithm \textsc{Max-Adv}. This procedure is repeated \( t = 2 \log(2/\delta) \) times to achieve the desired success probability \( 1 - \delta \). We combine the outputs of both the steps, i.e., \( \bar{V} \) and \( T \) and output the maximum among them using \textsc{Count-Max}. This ensures that we get a good approximation as we use the best of both the approaches.

Algorithm 3 \textsc{Tournament-Partition}

1. **Input**: Set of values \( V \), number of partitions \( l \)
2. **Output**: An approximate maximum value \( u_{\max} \)
3. Randomly partition \( V \) into \( l \) equal parts \( V_1, V_2, \ldots, V_l \)
4. for \( i = 1 \) to \( l \) do
5. \hspace{1em} \( p_i \leftarrow \text{Tournament}(V_i, 2) \)
6. \hspace{1em} \( T \leftarrow T \cup \{p_i\} \)
7. return \( T \)

Algorithm 4 \textsc{Max-Adv} : Maximum with Adversarial Noise

1. **Input**: Set of values \( V \), number of iterations \( t \)
2. **Output**: An approximate maximum value \( u_{\max} \)
3. \( l \leftarrow 1, T \leftarrow \emptyset \)
4. Let \( \bar{V} \) denote a sample of size \( \sqrt{n} \) selected uniformly at random (with replacement) from \( V \).
5. for \( i \leq t \) do
6. \hspace{1em} \( T_i \leftarrow \text{Tournament-Partition}(\bar{V}) \)
7. \hspace{1em} \( T \leftarrow T \cup T_i \)
8. \hspace{1em} \( u_{\max} \leftarrow \text{Count-Max}(\bar{V} \cup T) \)
9. return \( u_{\max} \)

Theoretical Guarantees. In order to prove approximation guarantee of Algorithm 4, we first argue that the sample \( \bar{V} \) contains a good approximation of the maximum value \( v_{\max} \) with a high probability. Let \( C \) denote the set of values that are very close to \( v_{\max} \). Suppose \( C = \{u : v_{\max}/(1 + \mu) \leq u \leq v_{\max}/(1 + \mu) \} \). In Lemma 3.4, we first show that \( \bar{V} \) contains a value \( v_j \in \bar{V} \) such that \( v_j \geq v_{\max}/(1 + \mu) \), whenever the size of \( C \) is large, i.e., \( |C| > \sqrt{n}/2 \). Otherwise, we show that we can recover \( v_{\max} \) correctly with probability \( 1 - \delta/2 \) whenever \( |C| \leq \sqrt{n}/2 \).

Lemma 3.4. (1) If \( |C| > \sqrt{n}/2 \), then there exists a value \( v_j \in \bar{V} \) satisfying \( v_j \geq v_{\max}/(1 + \mu) \) with a probability of \( 1 - \delta/2 \).
(2) Suppose \( |C| \leq \sqrt{n}/2 \). Then, \( T \) contains \( v_{\max} \) with a probability at least \( 1 - \delta/2 \).

Now, we briefly provide a sketch of the proof of Lemma 3.4. Consider the first step, where we use a uniformly random sample \( \bar{V} \)
of $\sqrt{n}$ points from $V$ (obtained with replacement). When $|C| \geq \sqrt{n}$, the probability that $\tilde{V}$ contains a value from $C$ is given by $1 - (1 - |C|/n)^{\sqrt{n}} \approx 1 - \delta/2$.

In the second step, Algorithm 4 uses a modified tournament tree that partitions the set $V$ into $I = \sqrt{n}$ parts of size $\frac{n}{\sqrt{n}} = \sqrt{n}$ each and identifies a maximum $p_i$ from each partition $V_i$ using Algorithm 2. We have that the expected number of elements from $C$ in a partition $V_i$ containing $p_{\max}$ is $\mathbb{E}[p_i] = \frac{\sqrt{n}}{\sqrt{n}} = \frac{1}{2}$. Thus by the Markov’s inequality, the probability that $V_i$ contains a value from $C$ is at most $\frac{1}{2}$. With $1/2$ probability, $p_{\max}$ will never be compared with any point from $C$ in the partition $V_i$. To increase the success probability, we run this procedure $t$ times and obtain all the outputs. Among the $t$ runs of Algorithm 2, we argue that $p_{\max}$ is never compared with any value of $C$ in at least one of the iterations with a probability at least $1 - (1 - 1/2)^{\log(O(n))} \geq 1 - \delta/2$.

In Lemma 3.1, we show that using COUNT-MAX we get a $(1 + \mu)^2$ multiplicative approximation. Combining it with Lemma 3.4, we have that $p_{\max}$ returned by Algorithm 4 satisfies $p_{\max} \geq \frac{1}{\sqrt{n}}(1+\mu)$ with probability $1 - \delta$. For query complexity, Algorithm 3 identifies $\sqrt{n}t$ samples denoted by $\tilde{V}$. These identified values, along with $T$ are then processed by COUNT-MAX to identify the maximum $p_{\max}$. This step requires $O(|V \cup T|^2) = O(n \log^2(1/\delta))$ oracle queries.

**Theorem 3.5.** Given a set of values $V$, Algorithm 4 returns a $(1 + \mu)^3$ approximation of maximum value with probability $1 - \delta$ using $O(n \log^2(1/\delta))$ oracle queries.

### 3.2 Probabilistic Noise

We cannot directly extend the algorithms for the adversarial noise model to the probabilistic noise model. Specifically, the theoretical guarantees presented in Lemma 3.2, do not apply when the noise is probabilistic. In this section, we develop several new ideas to handle probabilistic noise.

Let $\text{rank}(u, V)$ denote the index of $u$ in the non-increasing sorted order of values in $V$. So, $p_{\max}$ will have rank 1 and so on. Our main idea is to use an early stopping approach that uses a sample $S \subseteq V$ of $O(\log(n/\delta))$ values, selected randomly and for every value $u$ that is not in $S$, we calculate $\text{Count}(u, S)$ and discard $u$ using a chosen threshold for the Count scores. We argue that by doing so, it helps us eliminate the values that are far away from the maximum in the sorted ranking. The process of elimination is continued $\Theta(\log n)$ times to identify the maximum value. We present the pseudo code of this algorithm in Appendix A and prove the following approximation guarantee.

**Theorem 3.6.** There is an algorithm that returns $p_{\max} \in V$ such that $\text{rank}(p_{\max}, V) = O(\log^2(n/\delta))$ with probability $1 - \delta$ and requires $O(n \log^2(n/\delta))$ oracle queries.

**Remark 1.** The algorithm to identify the minimum value is same as that of maximum identification with a modification where Count scores now consider the case of Yes (instead of No) : $\text{Count}(u, S) = \sum_{x \in S} \mathbb{1}\{O(v, x) == \text{Yes}\}$.

### 3.3 Extension to Farthest and Nearest Neighbor

Given a set of records $V$, the farthest record from a query $u$ corresponds to the record $u' \in V$ such that $d(u, u')$ is maximum. This query is equivalent to finding maximum in the set of distance values given by $D(u) = \{d(u, u') \mid u' \in V\}$ containing $n$ values for which we already developed algorithms in Section 3. Since the ground truth distance between any pair of records is not known, we require quadruplet oracle (instead of comparison oracle) to identify the maximum element in $D(u)$. Similarly, the nearest neighbor of query record $u$ corresponds to finding the record with minimum distance value in $D(u)$. Algorithms for finding maximum from previous sections, extend for these settings with similar guarantees.

For probabilistic noise, the farthest identified in Section 3.2 is guaranteed to rank within the top-$O(\log^2 n)$ values of set $V$ (Theorem 3.6). In this section, we show it is possible to compute the farthest point within a small additive error under the probabilistic model, if the data set satisfies an additional property. In this section, we assume $p \leq 0.40$, a constant strictly less than $\frac{1}{4}$.

One of the challenges in developing robust algorithms for farthest identification is that every relative distance comparison of records from $u$ ($O(u, v_i, u, v_j)$ for some $v_i, v_j \in V$) may be answered incorrectly with constant error probability $p$ and the success probability cannot be boosted by repetition. We overcome this challenge by designing an algorithm that can perform pairwise comparisons, stated before, in a robust manner. Suppose the desired failure probability is $\delta$, we observe that if $\Theta(\log(1/\delta))$ records closest to the query record $u$ are known apriori (say $S$) and max$_{x \in S}\{d(u, x)\} \leq u$ for some $u > 0$, then each pairwise comparison of the form $O(u, v_i, u, v_j)$ can be replaced by Algorithm $\text{PAIRWISECOMP}$ and use it to execute Algorithm 4. Algorithm 5 takes the two records $v_i$ and $v_j$ as input along with $S$ and outputs Yes or No where Yes denotes that $v_i$ is closer to $u$. We calculate $\text{FCount}(v_i, v_j) = \sum_{x \in S} \mathbb{1}\{O(v_i, x, v_j, x) == \text{Yes}\}$ as a robust estimate where the oracle considers $v_j$ to be closer to $v_i$ than $v_j$. If $\text{FCount}(v_i, v_j)$ is smaller than $0.3|S| \leq (1 - p)|S|/2$ then we output No and Yes otherwise. Therefore, every pairwise comparison query can be replaced with $\Theta(\log(1/\delta))$ quadruplet oracle queries given using Algorithm 5.

We argue that Algorithm 5 will output the correct answer with a high probability if $d(u, v_i) - d(u, v_j) \geq 2\alpha$. In Lemma 3.7, we show that, if $d(u, v_i) > d(u, v_j) + 2\alpha$, then $\text{FCount}(v_i, v_j) \geq 0.3|S|$ with probability $1 - \delta$.

**Lemma 3.7.** Suppose max$_{v_i \in S} d(u, v_i) \leq \alpha$ and $|S| \geq 6 \log(1/\delta)$. Consider two records $v_i$ and $v_j$ such that $d(u, v_i) < d(u, v_j) - 2\alpha$ then $\text{FCount}(v_i, v_j) \geq 0.3|S|$ with a probability of $1 - \delta$.

**Algorithm 5** $\text{PAIRWISECOMP}(u, v_i, v_j, S)$

1: Calculate $\text{FCount}(v_i, v_j) = \sum_{x \in S} \mathbb{1}\{O(v_i, x, v_j, x) == \text{Yes}\}$
2: if $\text{FCount}(v_i, v_j) < 0.3|S|$ then
3: return No
4: else return Yes

With the help of Algorithm 5, relative distance query of any pair of records $v_i, v_j$ from $u$ can be answered correctly with a high probability provided $d(u, v_i) - d(u, v_j) \geq 2\alpha$. Therefore, the output of Algorithm 5 is equivalent to an additive adversarial error model where any quadruplet query can be adversarially incorrect if the distance $d(u, v_i) - d(u, v_j) \leq 2\alpha$ and correct otherwise. In Appendix B, we show that Algorithm 4 can be extended to the additive adversarial error model, such that each comparison $(u, v_i, u, v_j)$
is replaced by \textsc{PairwiseComp} (Algorithm 5) increasing the query complexity by a \(O(\log(1/\delta))\) factor. We give an approximation guarantee, that loses an additive \(6\alpha\) following a similar analysis of Theorem 3.5. We have the following guarantees:

\textbf{Theorem 3.8.} Given a query vertex \(u\) and a set \(S\) with \(|S| = \Omega(\log(1/\delta))\) such that \(\max_{v \in S} d(u,v) \leq \alpha\) then the farthest identified using Algorithm 4 (with \textsc{PairwiseComp}), denoted by \(u_{\max}\) is within \(6\alpha\) distance from the optimal farthest point, i.e., \(d(u,u_{\max}) \geq \max_{v \in V} d(u,v) - 6\alpha\) with a probability of \(1 - \delta\). Further the query complexity is \(O(n \log^3(1/\delta))\).

4 \(k\)-CENTER CLUSTERING

In this section, we present algorithms for \(k\)-center clustering and prove constant approximation guarantees of our algorithm. Our algorithm is an adaptation of the classical greedy algorithm for \(k\)-center [24], which identifies \(k\) centers iteratively. The greedy algorithm [24] initializes with an arbitrary point as the first cluster center and then iteratively identifies the next centers. In each iteration, it assigns all the points to the current set of clusters, by identifying the closest center for each point. After that, it finds the farthest point among the clusters and uses the farthest point as the new center. This technique requires \(O(nk)\) distance or pairwise oracle comparisons in the absence of noise and guarantees 2-approximation of the optimal clustering objective. We provide the pseudo code for this approach in Algorithm 6. If we use Algorithm 6 where we replace every comparison with an oracle query, the generated clusters can be arbitrarily worse even for small error. In order to improve the robustness of greedy algorithm, we devise new algorithms to perform assignment of points to respective clusters and farthest point identification. Missing Details from this section are collected in Appendix C and D.

\begin{algorithm}[h]
\caption{Greedy Algorithm}
\begin{algorithmic}[1]
\State \textbf{Input} : Set of points \(V\)
\State \textbf{Output} : Clusters \(C\)
\State \(s_1 \leftarrow\) arbitrary point from \(V, S = \{s_1\}, C = \{|V|\}\).
\For {\(i = 2\) to \(k\)}
\State \(s_j \leftarrow\) \textsc{Approx-Farthest}(\(S, C\))
\State \(S \leftarrow S \cup \{s_j\}\)
\State \(C \leftarrow\) \textsc{Assign}(\(S\))
\EndFor
\Return \(C\)
\end{algorithmic}
\end{algorithm}

4.1 Adversarial Noise

Now, we describe the two steps (\textsc{Approx-Farthest} and \textsc{Assign}) of Greedy Algorithm that will complete the description of Algorithm 6. To do so, we build upon the results from previous section that give algorithms for obtaining maximum/farthest point.

\textbf{APPROX-FARTHEST.} Given a clustering \(C\), and a set of centers \(S\), we construct the pairs \((v_i, s_j)\) where \(v_i\) is assigned to cluster \(C(s_j)\) centered at \(s_j \in S\). Using Algorithm 4, we identify the point, center pair that have the maximum distance i.e. \(\arg \max_{v \in V} d(v_i, s_j)\), which corresponds to the farthest point. For the parameters, we use \(l = \sqrt{n}, t = \log(2k/\delta)\) and number of samples \(V = \sqrt{n}t\).

\textbf{Assign.} After identifying the farthest point, we reassigned all the points to the centers (now including the farthest point as the new center) closest to them. We calculate a movement score called \(\text{MCount}\) for every point with respect to each center. \(\text{MCount}(u, s_j) = \sum_{u \in S} 1\{O((s_j, u), (s_k, u)) = \text{Yes}\}\) for any record \(u \in V\) and \(s_j \in S\). This step is similar to \textsc{Count-Max} Algorithm. We assign the point \(u\) to the center with the highest \(\text{MCount}\) value.

\textbf{Theoretical Guarantees.} In this section, we show the approximation guarantee obtained by greedy Algorithm 6 for the adversarial noise model. In each iteration, given a set of centers \(S\), we show that \textsc{Assign} reassigns each point to a center with distance approximately similar to the distance from the closest center. This is surprising given that we only make use of \(\text{MCount}\) scores to determine the assignment. Similarly, we argue that Algorithm 4 (used for \textsc{Approx-Farthest}) identifies a close approximation to the true farthest point for \(S\). Concretely, we show that \(u\) is assigned to \(s_j\) by \textsc{Assign} which is a \((1 + \mu)^2\) approximation; Algorithm 4 identifies farthest point \(w\) which is a \((1 + \mu)^5\) approximation (See Appendix C for more details). The proof follows an identical outline as in the case of finding maximum.

\begin{lemma}
\textbf{Lemma 4.1. Given a current set of centers} \(S\),
\begin{enumerate}
\item \textbf{assign assigns a point} \(u\) \textbf{to a cluster} \(C(s_j)\) \textbf{such that} \(d(u, s_j) \leq (1 + \mu)^2 \min_{s \in S} d(u, s)\) using \(O(nk)\) queries additionally.
\item \textbf{Approx-Farthest} \textbf{identifies a point} \(w\) \textbf{in cluster} \(C(s_i)\) \textbf{such that} \(\min_{s \in S} d(w, s) \geq \max_{s \in S} d(u, s) / (1 + \mu)^5\) with probability \(1 - \frac{\delta}{k}\) using \(O(n \log^2(k/\delta))\) oracle queries.
\end{enumerate}
\end{lemma}

In every iteration of the Greedy algorithm, if we identify a \(\alpha\)-approximation of the farthest point, and a \(\beta\)-approximation when reassigning the points, then, we show that the clusters output are a \(2\alpha\beta^2\)-approximation to the \(k\)-center objective (See Appendix C). For a given error parameter \(\mu\), we give the following theorem:

\textbf{Theorem 4.2.} For \(\mu < \frac{1}{12}\), Algorithm 6 achieves a \((2 + \mu)\)-approximation for the \(k\)-center objective using \(O(nk^2 + nk \cdot \log^2(k/\delta))\) oracle queries with probability \(1 - \delta\).

\textbf{Proof.} From the above discussed claim and Lemma 4.1, we have that Algorithm 6 achieves a \(2(1 + \mu)^9\) approximation for \(k\)-center objective. Scaling the value of \(\mu\), we get that when \(\mu \leq \frac{1}{12}\). Algorithm 6 is a \(2 + \mu\) approximation. From Lemma 4.1, we have that in each iteration, we succeed with probability \(1 - \delta/\kappa\). Using union bound, the failure probability is given by \(\delta\). For query complexity, as there are \(k\) iterations, and in each iteration we use \textsc{Assign} and \textsc{Approx-Farthest}, using Lemma 4.1, we have the theorem. □

4.2 Probabilistic Noise

For probabilistic noise, each query can be incorrect with probability \(p\) and therefore, Algorithm 6 may lead to poor approximation guarantees. Here, we build upon the results from section 3.3 and provide \textsc{Approx-Farthest} and \textsc{Assign} algorithms. We denote the size of minimum cluster among optimum clusters \(c^*\) to be \(m\), and total failure probability of our algorithms to be \(\delta\). We assume \(p < 0.40\), a constant strictly less than \(\frac{1}{2}\). Let \(\gamma = 450\) be a large constant used in our algorithms which obtains the claimed guarantees, although we observe a smaller constant empirically for similar performance.

\textbf{Overview.} Algorithm 7 presents the pseudo-code of our algorithm that operates in two phases. In the first phase (lines 3-12), we sample
each point with a probability \( \frac{\log(n)}{m} \) to identify a small sample of \( \frac{\log(n)}{m} \) points (denoted by \( \hat{V} \)) and use Algorithm 7 to identify \( k \) centers iteratively. In this process, we also identify a core for each cluster (denoted by \( R \)). Formally, core is defined as a set of \( \Theta(\log(n)/\delta) \) points that are very close to the center with high probability. These cores are then used in the second phase (line 15) for the assignment of remaining points.

**Algorithm 7 Greedy Clustering**

1. **Input**: Set of points \( V \), smallest cluster size \( m \).
2. **Output**: Clusters \( C \).
3. For every \( u \in V \), include \( u \) in \( \hat{V} \) with probability \( \frac{\log(n)}{m} \).
4. \( s_1 \leftarrow \text{select an arbitrary point from } \hat{V}, S \leftarrow \{s_1\} \)
5. \( C(s_1) \leftarrow \hat{V} \)
6. \( R(s_1) \leftarrow \text{IDENTIFY-CORE}(C(s_1), s_1) \)
7. for \( i = 2 \) to \( k \) do
8. \( s_i \leftarrow \text{APPROX-FARTHEST}(S, C) \)
9. \( C, R \leftarrow \text{ASSIGN}(S, s_i, R) \)
10. \( S \leftarrow S \cup \{s_i\} \)
11. \( C \leftarrow \text{ASSIGN-FINAL}(S, R, V \setminus \hat{V}) \)
12. return \( C \)

Now, we describe the main challenge in extending APPROX-FARTHEST and Assign ideas of Greedy Algorithm 6. Given a cluster \( C \) containing the center \( s_i \), when we find the APPROX-FARTHEST, the ideas from Section 3.2 only give a \( O(\log^2 n) \) rank approximation. As shown in section 3.3, we can improve the approximation guarantee by considering a set of \( \Theta(\log(n)/\delta) \) points closest to \( s_i \), denoted by \( R(s_i) \) and call them core of \( s_i \).

Assign. Consider a point \( s_i \) such that we have to assign points to form the cluster \( C(s_i) \) centered at \( s_i \). We calculate an assignment score (called ACount in line 4) for every point \( u \) of a cluster \( C(s_i) \) \( \setminus R(s_i) \) centered at \( s_i \). ACount captures the total number of times \( u \) is considered to belong to some cluster as that of \( x \) for each \( x \) in the core \( R(s_i) \). Intuitively, points that belong to the same cluster as that of \( s_i \) are expected to have higher ACount score. Based on the scores, we move \( u \) to \( C(s_i) \) or keep it in \( C(s_j) \).

**Algorithm 8 ASSIGN**(\( S, s_i, R \))

1. \( C(s_i) \leftarrow \{s_i\} \)
2. for \( s_j \in S \) do
3. for \( u \in C(s_j) \setminus R(s_j) \) do
4. \( ACount(u, s_i, s_j) = \sum_{s_k \in R(s_j)} I\{O(u, s_i, u, s_k) == \text{Yes}\} \)
5. if \( ACount(u, s_i, s_j) > 0.3|R(s_j)| \) then
6. \( C(s_i) \leftarrow C(s_i) \cup \{u\}; C(s_j) \leftarrow C(s_j) \setminus \{u\} \)
7. \( R(s_i) \leftarrow \text{IDENTIFY-CORE}(C(s_i), s_i) \)
8. return \( C, R \)

**IDENTIFY-CORE.** After forming cluster \( C(s_i) \), we identify the core of \( s_i \). For this, we calculate a score, denoted by Count and captures number of times it is closer to \( s_i \) compared to other points in \( C(s_i) \). Intuitively, we expect points with high values of Count to belong to \( C^*(s_i) \) i.e., optimum cluster containing \( s_i \). Therefore we sort these Count scores and return the highest scored points.

**Algorithm 9 IDENTIFY-CORE**(\( C(s_i), s_i \))

1. for \( u \in C(s_i) \) do
2. \( \text{Count}(u) = \sum_{s_k \in C(s_i)} I\{O(u, x, s_i, u) == \text{No}\} \)
3. \( R(s_i) \) denote set of \( 8y \log(n/\delta)/9 \) points with the highest Count values.
4. return \( R(s_i) \)

**APPROX-FARTHEST.** For a set of clusters \( C \), and a set of centers \( S \), we construct the pairs \((v_i, s_j)\) where \( v_i \) is assigned to cluster \( C(s_j) \) centered at \( s_j \in S \) and each center \( s_j \in S \) has a corresponding core \( R(s_j) \). The farthest point can be found by finding the maximum distance (point, center) pair among all the points considered. So, we use the ideas developed in section 3.3.

We leverage ClusterComp (Algorithm 10) to compare the distance of two points from their respective centers because of constant probability of error in each comparison. ClusterComp gives a robust answer to a pairwise comparison query to the oracle \( O(v_i, v_j, s_j) \) using the cores \( R(s_i) \) and \( R(s_j) \). ClusterComp can be used as a pairwise comparison subroutine in place of PAIRWISE-COMP for the algorithm in Section 3 to calculate the farthest point. For every \( s_j \in S \), let \( R(s_j) \) denote an arbitrary set of \( \sqrt{R(s_i)} \) points from \( R(s_i) \). For a ClusterComp comparison query between the pairs \((v_i, s_j)\) and \((v_j, s_j)\), we use these subsets in Algorithm 10 to ensure that we only make \( \Theta(\log(n)/\delta) \) oracle queries for every comparison. However, when the query is between points of the same cluster, say \( C(s_j) \), we use all the \( \Theta(\log(n)/\delta) \) points from \( R(s_i) \). For the parameters used to find maximum using Algorithm 4, we use \( l = \sqrt{n}, t = \log(4n/\delta) \).

**Algorithm 10 CLUSTERCOMP**(\( v_i, v_j, s_j \))

1. comparisons \( \leftarrow 0, \text{FCOUNT}(v_i, v_j) \leftarrow 0 \)
2. if \( s_j = s_j \) then
3. \( \text{Let FCOUNT}(v_i, v_j) = \sum_{s_k \in R(s_j)} I\{O(v_i, x, v_j, x) == \text{Yes}\} \)
4. comparisons \( \leftarrow |R(s_j)| \)
5. else let \( \text{FCOUNT}(v_i, v_j) = \sum_{s_k \in R(s_j)} \text{FCOUNT}(v_i, v_k) \)
6. comparisons \( \leftarrow |R(s_j)| \cdot |R(s_j)| \)
7. if \( \text{FCOUNT}(v_i, v_j) < 0.3 \cdot \text{comparisons} \) then
8. return No
9. else return Yes

**ASSIGN-FINAL.** After obtaining \( k \) clusters on the set of sampled points \( \hat{V} \), we assign the remaining points using ACount scores, similar to the one described in Assign. For every point that is not sampled, we first assign it to \( s_i \in S \), and if \( \text{ACOUNT}(u, s_2, s_1) \geq 0.3|R(s_i)| \), we re-assign it to \( s_2 \), and continue this process iteratively. After assigning all the points, the clusters are returned as output.

**Theoretical Guarantees.** Our Algorithm first constructs a sample \( \hat{V} \subseteq V \) and runs greedy algorithm on this sampled set of points. Our main idea to ensure that good approximation of the \( k \)-center objective lies in identifying a good core around each center. Using a sampling probability \( \frac{\log(n/\delta)}{m} \) ensures that we have at least \( \Theta(\log(n/\delta)) \) points from each of the optimal \( k \)-center clusters in our sampled set \( \hat{V} \). By finding the closest points using Count scores, we identify \( O(\log(n/\delta)) \) points around every center that are in the
optimal cluster. Essentially, this forms the core of each cluster. These cores are then used for robust pairwise comparison queries (similar to Section 3.3), in our APPROX-FARThIEST and ASSIGN sub-routines. For finding the farthest point, we make use of core of every center $s_i$, i.e., $R(s_i)$ and we know that FCount calculated for ClusterComp ensures a good approximation. As remarked in section 3.3, we can use core of each cluster to once again find out a good approximation for finding closest center. Finally, we use ASSIGN – FINAL to assign all the points to the centers. We give the following theorem, which guarantees a $O(1)$-approximation, i.e., a constant approximation with high probability.

**Theorem 4.3.** Given $p \leq 0.4$, a failure probability $\delta$, and $m = \Omega(\log^3(n/\delta)/\delta)$. Then, Algorithm 7 achieves a $O(1)$-approximation for the $k$-center objective using $O(nk \log(n/\delta) + \Delta^2 k \log^2(n/\delta))$ oracle queries with probability $1 - \delta$.

**Remark.** If the optimal $k$-center clusters are of size at least $O(\sqrt{n})$, our algorithm returns a clustering that achieves constant approximation with a probability $1 - \delta$ using $O(nk \log^2(n/\delta))$ queries.

## 5 HIERARCHICAL CLUSTERING

In this section, we present robust algorithms for agglomerative hierarchical clustering using single linkage and complete linkage objectives. For single and complete linkage clustering, the naïve algorithms initialize every record as a singleton cluster and merge the closest pair of clusters iteratively. For a set of clusters $C = \{C_1, \ldots, C_k\}$, the distance between any pair of clusters $C_i$ and $C_j$, for single linkage clustering, is defined as the minimum distance between any pair of records in the clusters, $d_{\text{SL}}(C_1, C_2) = \min_{v_1 \in C_1, v_2 \in C_2} d(v_1, v_2)$. For complete linkage, cluster distance is defined as the maximum distance between any pair of records. All algorithms discussed in this section can be easily extended for complete linkage, and therefore we study single linkage clustering (See Appendix E for more details). The main challenge to implement single linkage clustering in the presence of adversarial noise is identification of minimum value in a list of at most $\binom{n}{2}$ distance values. In each iteration, the closest pair of clusters can be identified by using Algorithm 4 (with $z = 2 \log(n/\delta)$) to calculate the minimum over the set containing pairwise distances. For this algorithm, Lemma 5.1 shows that the pairs of clusters merged in any iteration are a constant approximation of the optimal merge operation at that iteration. The proof of this lemma follows from Theorem 3.5.

**Lemma 5.1.** Given a collection of clusters $C = \{C_1, \ldots, C_r\}$, our algorithm to calculate the closest pair (using Algorithm 4) identifies $C_1$ and $C_2$ to merge according to single linkage objective if $d_{\text{SL}}(C_1, C_2) \leq (1 + \mu)^3 \min_{C_i, C_j \in C} d(C_i, C_j) + 1 - \delta$ probability and requires $O(r^2 \log^2(n/\delta))$ queries.

**Overview.** Agglomerative clustering techniques are known to be inefficient. Each iteration of merge operation compares at most $\binom{n}{2}$ pairs of distance values and the algorithm operates $n$ times to construct the hierarchy. This yields an overall query complexity of $O(n^3)$. To improve the complexity of these techniques, SLINK algorithm [41] was proposed to construct the hierarchy in $O(n^2)$ comparisons. In order to implement this algorithm with a comparison oracle, for every cluster $C_i \in C$, we maintain an adjacency list containing every cluster $C_j$ in $C$ along with a pair of records with the distance equal to the distance between the clusters. For example, the entry for $C_j$ in the adjacency list of $C_i$ contains the pair of records $(u_i, v_j)$ such that $d(u_i, v_j) = \min_{e \in C_i} d(u_i, v_j)$. Algorithm 11 presents the pseudo code for single linkage clustering under the adversarial noise model. The algorithm is initialized with singleton clusters where every record is a separate cluster (line 3). Then, we identify closest cluster for every $C_i \in C$, and denote it by $\tilde{C}_i$. This step takes $n$ nearest neighbor queries, each requiring $O(n \log^2(n/\delta))$ comparison queries. In every subsequent iteration, we identify the closest pair of clusters (Using section 3.3), say $C_j$ and $\tilde{C}_j$ from $C$.

After merging these clusters, the data structure is updated as follows. To update the adjacency list, we need the pair of records with minimum distance between the merged cluster $C' \equiv C_j \cup \tilde{C}_j$ and every other cluster $C_k \in C$. In the previous iteration of the algorithm, we already have the minimum distance record pair for $(C_j, C_k)$ and $(\tilde{C}_j, C_k)$. Therefore a single query between these two pairs of records is sufficient to identify the minimum distance edge between $C'$ and $C_k$. Formally:

$$d_{\text{SL}}(C_j \cup \tilde{C}_j, C_k) = \min_{u \in C_j \cup \tilde{C}_j, v \in C_k} d(u, v) \quad (1)$$

$$= \min_{u \in C_j \cup \tilde{C}_j} d(u, v) + \min_{v \in C_k} d(u, v) \quad (2)$$

$$= \min_{u \in C_j \cup \tilde{C}_j} \min_{v \in C_k} d(u, v) \quad (3)$$

The nearest neighbor of the merged cluster is identified by running minimum calculation over its adjacency list. In Algorithm 11, as we identify closest pair of clusters, each iteration requires $O(n \log^2(n/\delta))$ comparison queries. As our Algorithm terminates in at most $n$ iteration, it has an overall query complexity of $O(n^2 \log^2(n/\delta))$. In Theorem 5.2, we give an approximation guarantee for every merge operation of Algorithm 11.

**Theorem 5.2.** In any iteration, suppose the distance between a cluster $C_j \in C$ and its identified nearest neighbor $\tilde{C}_j$ is $\alpha$-approximation of its distance from the optimal nearest neighbor, then the distance between pair of clusters merged by Algorithm 11 is $\alpha(1 + \mu)^3$ approximation of the optimal distance between the closest pair of clusters in $C$ with a probability of $1 - \delta$ using $O(n \log^2(n/\delta))$ oracle queries.

### Algorithm 11 Greedy Algorithm

1. **Input**: Set of points $V$
2. **Output**: Hierarchy $H$
3. $H \leftarrow \{\{v\} | v \in V\}, C \leftarrow \{\{v\} | v \in V\}$
4. for $C_i \in C$ do
5.   $\tilde{C}_i \leftarrow \text{NEARESTNEIGHBOR of } C_i$ among $C \setminus \{C_j\}$ using Sec 3.3
6. while $|C| > 1$ do
7.   Let $(C_j, \tilde{C}_j)$ be the closest pair among $(C_i, \tilde{C}_j), \forall C_i \in C$
8.   $C' \leftarrow \tilde{C}_j \cup \tilde{C}_j$
9.   Update Adjacency list of $C'$ with respect to $C$
10. Add $C'$ as parent of $\tilde{C}_j$ and $C_j$ in $H$.
11. $C \leftarrow \{C \setminus \{C_j, \tilde{C}_j\}\} \cup \{C'\}$
12. $\tilde{C'} \leftarrow \text{NEARESTNEIGHBOR of } C'$ from its adjacency list
13. return $H$
Probabilistic Noise model. The above discussed algorithms do not extend to the probabilistic noise due to constant probability of error for each comparison query. However, when we are given apriori, a partitioning of $V$ into clusters of size at least $\log n$ such that the maximum distance between any pair of records in every cluster is smaller than $\alpha$ (a constant), Algorithm 11 can be used to construct the hierarchy correctly. For this case, the algorithm to identify the closest and farthest pair of clusters is same as the one discussed in Section 3.3.

6 EXPERIMENTS

In this section we evaluate the effectiveness of our techniques on various real world datasets and answer the following questions. Q1: Are proposed techniques robust to different levels of noise in oracle answers? Q2: How does the query complexity and solution quality of proposed techniques compare with optimum for varied levels of noise?

6.1 Experimental Setup

Datasets. We consider the following real-world datasets and give results for maximum, farthest and nearest neighbor identification along with $k$-center and hierarchical clustering.

(1) cities dataset comprises of 36K cities of the United States. The different features of the cities include state, county, zip code, population, time zone, latitude and longitude.

(2) covid dataset contains 13K covid cases from Asia-Pacific region along with geo-location of each report.

(3) dblp contains 1.8M titles of computer science papers from different areas. From these titles, noun phrases were extracted and a dictionary of all the topics was constructed. Euclidean distance in word2vec embedding space is considered as the ground truth distance between concepts.

(4) hotels dataset contains descriptions of 15K hotels across India. The descriptions include name, location, latitude, longitude and other facilities.

Baselines. We compare our techniques with the optimal solution (whenever possible) and the following baselines. (a) Tour 2 constructs a binary tournament tree over the entire dataset to compare the values and the root node corresponds to the identified maximum/minimum value. We present the pseudo code in Algorithm 2 ($\lambda = 2$). This approach is an adaptation of the maximum calculation algorithm in [14] with a difference that each query is not repeated multiple times to increase success probability. We also use them to identify the farthest and nearest point in the greedy $k$-center algorithm [49] and closest pair of clusters in hierarchical clustering. Other variations of this baseline consider the tournament tree with degree $\lambda > 2$. By default, all comparisons use $\lambda = 2$ as it has the lowest query complexity and we discuss the effect of $\lambda$ in Figure 3(b).

(b) Samp considers a sample of $\sqrt{n}$ records and identifies the farthest/nearest by performing quadratic number of comparisons over the sampled points using COUNT-MAX. For $k$-center, Samp considers a sample of $k \log n$ points to identify $k$ centers over these samples using the greedy algorithm. It then assigns all the remaining points to these identified centers by querying each record with every pair of center, similar to Algorithm 1.

Calculating optimal clustering objective for $k$-center is NP-hard even in the presence of accurate pairwise distance. Therefore, we compare the solution quality with respect to the greedy algorithm (which obtains a 2-approximation) on the ground truth distances (denoted by TDist) [49]. For farthest, nearest neighbor and hierarchical clustering, TDist denotes the optimal technique that has access to ground truth distance between records.

Our algorithm is labelled Far for farthest identification, NN for nearest neighbor, kC for $k$-center and HC for hierarchical clustering with subscript $a$ denoting the adversarial model and $p$ denoting the probabilistic noise model.

All algorithms are implemented in C++ and run on a server with 64GB RAM. The reported results are averaged over 100 randomly chosen iterations. We simulate the oracle using the ground truth distance between points. Unless specified, we set $t = 1$ in Algorithm 4 and $y = 2$ in Algorithm 7.

Evaluation Metric. For finding maximum and nearest neighbors, we compare different techniques by evaluating the true distance of the returned solution from the queried points. For $k$-center, we use the objective value, i.e., the maximum radius of the returned clusters as the evaluation metric and compare against the true greedy algorithm (TDist) and other baselines. For hierarchical clustering, we compute the pairs of clusters merged in every iteration and compare the average true distance between these clusters. In addition to the quality of returned solution, we compare the query complexity and running time of the proposed techniques with the baselines described above.

6.2 Solution Quality

In this section, we compare the robustness of our proposed techniques for the studied problems.

Finding Max and Farthest/Nearest Neighbor. In Figure 2, $\mu = 0$ denotes the setting where the oracle answers all queries correctly. In this case, Far and Tour 2 identify the optimal solution but Samp does not identify the optimal solution for cities. In both datasets, Far identifies the correct farthest point for $\mu < 1$. Even with an
increase in noise ($\mu$), we observe that farthest is always at a distance within 4 times the optimal distance (See Fig 2).

We observe that the quality of farthest identified by Tour2 is close to that of Far for smaller $\mu$ because the optimal farthest point $v_{\max}$ has only a few points in the confusion region $C$ (See Section 3) that contains the points that are close to $v_{\max}$. For e.g., less than 10% are present in $C$ when $\mu = 1$ for cities dataset, i.e., less than 10% points return erroneous answer when compared with $v_{\max}$.

In terms of query complexity, Tour2 and Samp require $37 \times 10^3$ and $18 \times 10^3$ queries as compared to $100 \times 10^3$ by Far for cities dataset. All these techniques require less than $4n$ (where $n$ is number of records) queries across all settings, validating the linear query complexity claim of our farthest identification algorithm.

To further analyze the quality of these techniques, we consider a synthetic dataset containing $10K$ data points where every point is assigned a value uniformly at random in the range $[50, 100]$ with probability $q$ and $[0, 50]$ with probability $1 - q$ and consider adversarial noise with $\mu = 1$. The ground truth maximum value is 100 and Figure 3(a) compares the quality of the identified maximum value for different values of $q$ averaged over 100 iterations. When $q$ is small, the number of values that are closer to the maximum value are fewer. In this case Tour2 performs similar to Far as the probability of comparing the maximum with one of these values is low. The quality of Tour2 deteriorates with increasing $q$ and Samp achieves better solution for higher values of $q$.

Effect of $\lambda$. In this experiment, we vary the degree of tournament tree to compare the solution quality and query complexity. Figure 3(b) compares the quality of Far with Tour for different values of $\lambda$ under the adversarial noise model with $\mu = 1$. The different variations of tournament are denoted by Tour2 ($\lambda = 2$), Tour4 ($\lambda = 4$), Tour8 and Tour16. With increase in tournament degree, the identified maximum value increases but the number of queries also increase linearly with $\lambda$. We observe that Tour8 and Far have similar query complexity but the maximum value returned by Far is 1.5 times larger than that of Tour8. $\lambda = 8$ requires more than 5× the queries required by Far and achieves poor solution quality. This justifies the effectiveness of Far to identify the maximum value with low query complexity.

Farthest. In Figure 6, we compare the true distance of the identified farthest points for the case of probabilistic noise with error probability $p$. We observe that Far$_p$ identifies points with distance values very close to the farthest distance $TDist$, across all data sets and error values. This shows that Far performs significantly better than the theoretical approximation presented in Section 3. On the other hand, the solution returned by Samp is more than 4× smaller

Figure 4: $k$-center clustering with adversarial noise model. The compared objective denotes the radius of the largest cluster and $k$ denotes the number of clusters.

Figure 5: $k$-center clustering under probabilistic error model with $p = 0.1$.

Figure 6: Comparison of farthest identification techniques for probabilistic noise model for varying error ($p$).

Figure 7: Comparison of nearest neighbor techniques for adversarial and probabilistic noise model.
Our Approach

In conclusion, we observe that our techniques achieve the best quality $\mu$ with higher $\mu$ values. Among baselines, we observe a decline in solution quality for higher noise ($p$) values. We omit $\mu$ uses less than $\mu$ fraction of the queries. As discussed in Section 4, $\mu$ performs worse than HC, demonstrating robustness of HC. In terms of query complexity, HC requires $O(n^3)$ queries. We observe similar behavior across all datasets.

$\mu$-center Clustering. Figure 4 compares the $\mu$-center objective of the returned clusters for varying $\mu$ in the adversarial noise model. $\mu$Dist denotes the best possible clustering objective, which is guaranteed to be a 2-approximation of the optimal objective. The set of clusters returned by $\mu$Cp are consistently very close to $\mu$Dist across all datasets, validating the theory. For higher values of $\mu$, $\mu$Cp approaches closer to $\mu$Dist, thereby improving the approximation guarantees. We observe similar trends for $\mu$Cp under probabilistic noise with $p = 0.1$ (Figure 5).

The quality of clusters identified by $\mu$Cp are similar to that of $\mu$ for adversarial and probabilistic error models. Our $\mu$-center clustering technique requires less than 450 min to identify 50 centers for dblp dataset across different noise models; the running time grows linearly with $\mu$. While the running time of our algorithms are slightly higher than $\mu$2 for farthest, nearest and $\mu$-center, $\mu$2 did not finish in 48 hrs due to $O(n^3)$ running time for single and complete linkage hierarchical clustering. We observe similar performance for the probabilistic noise model.

7 CONCLUSION

In this paper, we show how algorithms for various basic tasks such as finding maximum, nearest neighbor, $\mu$-center clustering, and agglomerative hierarchical clustering can be designed using distance based comparison oracle in presence of noise. We believe our techniques can be useful for other clustering tasks such as $k$-means and $k$-median, and we leave those as future work.
A FINDING MAXIMUM

Lemma A.1. (Hoeffding’s Inequality) If $X_1, X_2, \ldots, X_n$ are independent random variables with $a_i \leq X_i \leq b_i$ for all $i \in [n]$, then

$$\Pr \left( \left| \sum_{i} X_i - E[X_i] \right| \geq ne \right) \leq 2 \exp \left( - \frac{2n^2 \epsilon^2}{\sum_i (b_i - a_i)^2} \right)$$

A.1 Adversarial Noise

Let $V$ be the maximum value among $V$ and the set of records for which the oracle answer can be incorrect is given by

$$C = \{ u \mid u \in V, u \geq \frac{v_{\text{max}}}{1 + \mu} \}$$

Claim A.2. For any partition $V_i$, Tournament($V_i$) uses at most $2|V_i|$ oracle queries.

Proof. Consider the $i$th round in Tournament. We can observe that the number of remaining values is at most $\frac{|V_i|}{2^i}$. So, we make $\frac{|V_i|}{2^i}$ many oracle queries in this round. Total number of oracle queries made is

$$\sum_{i=0}^{\log n} \frac{|V_i|}{2^i} \leq 2|V_i|$$

Lemma A.3. Given a set of values $S$, COUNT-MAX($S$) returns a $(1 + \mu)^2$ approximation of maximum value of $S$ using $O(|S|^2)$ oracle queries.

Proof. Let $v_{\text{max}} = \max\{x \in S\}$. Consider a value $w \in S$ such that $w < \frac{v_{\text{max}}}{1 + \mu}$. We compare the Count values for $v_{\text{max}}$ and $w$ given by, Count($v_{\text{max}}, S$) = $\sum_{x \in S} 1\{O(v_{\text{max}}, x) == \text{No}\}$ and Count($w, S$) = $\sum_{x \in S} 1\{O(w, x) == \text{No}\}$. We argue that $w$ can never be returned by Algorithm 1, i.e., Count($w, S$) < Count($v_{\text{max}}, S$).

$$\begin{align*}
\text{Count}(v_{\text{max}}, S) &= \sum_{x \in S} 1\{O(v_{\text{max}}, x) == \text{No}\} \\
&= \sum_{x \in S \setminus \{v_{\text{max}}\}} 1\{x < v_{\text{max}}/(1 + \mu)\} \\
&= 1 \cdot \sum_{x \in S \setminus \{v_{\text{max}}, w\}} 1\{x < v_{\text{max}}/1 + \mu\} \\
\text{Count}(w, S) &= \sum_{y \in S} 1\{O(w, y) == \text{No}\} \\
&\leq \sum_{y \in S \setminus \{w, v_{\text{max}}\}} 1\{y \leq (1 + \mu)w\} \\
&\leq \sum_{y \in S \setminus \{w, v_{\text{max}}\}} 1\{y \leq v_{\text{max}}/(1 + \mu)\}
\end{align*}$$

Combining the two, we have:

$$\text{Count}(v_{\text{max}}, S) > \text{Count}(w, S)$$

This shows that the Count of $v_{\text{max}}$ is strictly greater than the count of any point $w$ with $w < \frac{v_{\text{max}}}{1 + \mu}$. Therefore, our algorithm would have output $v_{\text{max}}$ instead of $w$. For calculating the Count for all values in $S$, we make at most $|S|^2$ oracle queries as we compare every value with every other value. Finally, we output the maximum value as the value with highest Count. Hence, the claim.

Lemma A.4 (Lemma 3.2 Restated). Suppose $v_{\text{max}}$ is the maximum value among the set of records $V$. Algorithm 2 outputs a value $u_{\text{max}}$ such that $u_{\text{max}} \geq \frac{v_{\text{max}}}{(1 + \mu)^2 \log^2 n}$ using $O(n\lambda)$ oracle queries.

Proof. From Lemma A.3, we have that we lose a factor of $(1 + \mu)^2$ in each level of the tournament tree, we have that after $\log_2 n$ levels, the final output will have an approximation guarantee of $(1 + \mu)^2 \log_2 n$. The total number of queries used is given by $\sum_{i=0}^{\log_2 n} \frac{|V_i|}{\lambda^2} = O(n\lambda)$ where $V_i$ is the number of records at level $i$.

Lemma A.5. Suppose $|C| > \sqrt{n}/2$. Let $\tilde{V}$ denote a set of $2\sqrt{n} \log(1/\delta)$ samples obtained by uniform sampling with replacement from $V$. Then, $\tilde{V}$ contains a $(1 + \mu)$ approximation of the maximum value $v_{\text{max}}$, with probability $1 - \delta$. 

Raghavendra Addanki, Sainyam Galhotra, and Barna Saha
Proof. Consider the first step where we use a uniformly random sample \( \tilde{V} \) of \( \sqrt{n}t = 2\sqrt{n}\log(1/\delta) \) values from \( V \) (obtained by sampling with replacement). Given \( |C| \geq \frac{\sqrt{n}}{2} \), probability that \( \tilde{V} \) contains a value from \( C \) is given by

\[
\Pr[\tilde{V} \cap C \neq \emptyset] = 1 - \left( 1 - \frac{|C|}{n} \right)^{|\tilde{V}|} \geq 1 - \left( 1 - \frac{1}{2\sqrt{n}} \right)^{2\sqrt{n}\log(1/\delta)} > 1 - \delta
\]

So, with probability \( 1 - \delta \), there exists a value \( u \in C \cap \tilde{V} \). Hence, the claim.

Lemma A.6. Suppose the partition \( V_i \) contains the maximum value \( v_{\max} \) of \( V \). If \( |C| \leq \sqrt{n}/2 \), then, Tournament\((V_i)\) returns the \( v_{\max} \) with probability 1/2.

Proof. Algorithm 4 uses a modified tournament tree that partitions the set \( V \) into \( l = \sqrt{n} \) parts of size \( n/l = \sqrt{n} \) each and identifies a maximum \( \rho_i \) from each partition \( V_i \) using Algorithm 2. If \( v_{\max} \in V_i \), then,

\[
E[|C \cap V_i|] = \frac{|C|}{l} = \frac{\sqrt{n}}{2\sqrt{n}} = \frac{1}{2}
\]

Using Markov’s inequality, the probability that \( V_i \) contains a value from \( C \) is given by:

\[
\Pr[|C \cap V_i| \geq 1] \leq E[|C \cap V_i|] \leq \frac{1}{2}
\]

Therefore, with at least a probability of \( \frac{1}{2} \), \( v_{\max} \) will never be compared with any point from \( C \) in the partition \( V_i \) containing \( v_{\max} \). Hence, \( v_{\max} \) is returned by Tournament\((V_i)\) with probability 1/2.

Lemma A.7 (Lemma 3.4 restated). (1) If \( |C| > \sqrt{n}/2 \), then there exists a value \( v_j \in \tilde{V} \) satisfying \( v_j \geq v_{\max}/(1 + \mu) \) with a probability of \( 1 - \delta/2 \).

(2) Suppose \( |C| \leq \sqrt{n}/2 \). Then, \( T \) contains \( v_{\max} \) with a probability at least \( 1 - \delta/2 \).

Proof. Claim (1) follows from Lemma A.5.

In every iteration \( i \leq t \) of Algorithm 4, we have that \( v_{\max} \in T_i \) with probability \( \frac{1}{2} \) (Using Lemma A.6). To increase the success probability, we run this procedure \( t \) times and obtain all the outputs. Among the \( t = 2\log(2/\delta) \) runs of Algorithm 2, we have that \( v_{\max} \) is never compared with any value of \( C \) in at least one of the iterations with a probability at least

\[
1 - (1 - 1/2)^{2\log(2/\delta)} \geq 1 - \frac{\delta}{2}
\]

Hence, \( T = \bigcup T_i \) contains \( v_{\max} \) with a probability \( 1 - \frac{\delta}{2} \).

Theorem A.8 (Theorem 3.5 restated). Given a set of values \( V \), Algorithm 4 returns a \((1 + \mu)^3\) approximation of maximum value with probability \( 1 - \delta \) using \( O(n\log^2(1/\delta)) \) oracle queries.

Proof. In Algorithm 4, we first identify an approximate maximum value using Sampling. If \( |C| \geq \frac{\sqrt{n}}{2} \), then, from Lemma A.5, we have that the value returned is a \((1 + \mu)^3\) approximation of the maximum value of \( V \). Otherwise, from Lemma A.7, \( T \) contains \( v_{\max} \) with a probability \( 1 - \delta/2 \). As we use Count\(\max\) on the set \( \tilde{V} \cup T \), we know that the value returned, i.e., \( u_{\max} \) is a \((1 + \mu)^3\) of the maximum among values from \( \tilde{V} \cup T \). Therefore, \( u_{\max} \geq \frac{\sqrt{n}}{1 + \mu^3} \). Using union bound, the total probability of failure is \( \delta \).

For query complexity, Algorithm 3 obtains a set \( \tilde{V} \) of \( \sqrt{n}t \) sample values. Along with the set \( T \) obtained (where \( |T| = \frac{nt}{2} \)), we use Count\(\max\) on \( \tilde{V} \cup T \) to output the maximum \( u_{\max} \). This step requires \( O(|\tilde{V} \cup T|^2) = O((\sqrt{n}t + \frac{nt}{2})^2) \) oracle queries. In an iteration \( i \), for obtaining \( T_i \), we make \( O(\sum_j |V_j|) = O(n) \) oracle queries (Claim A.2), and for \( t \) iterations, we make \( O(nt) \) queries. Using \( t = 2\log(2/\delta), l = \sqrt{n} \), in total, we make \( O(nt + (\sqrt{n}t + \frac{nt}{2})^2) = O(n\log^2(1/\delta)) \) oracle queries. Hence, the theorem.

A.2 Probabilistic Noise

Lemma A.9. Suppose the maximum value \( u_{\max} \) is returned by Algorithm 2 with parameters \((V, n)\). Then, rank\((u_{\max}, V)\) = \( O(\sqrt{n}\log(1/\delta)) \) with a probability of \( 1 - \delta \).

Proof. We have for the maximum value \( u_{\max} \), expected count value:

\[
\mathbb{E}[\text{Count}(u_{\max}, V)] = \sum_{w \in V} \mathbb{1}\{O(w, u_{\max}) = w\} = (n - 1)(1 - p)
\]
Using Hoeffding’s inequality, with probability $1 - \delta/2$:

$$\text{Count}(v_{\max}, V) \geq (n - 1)(1 - p) - \sqrt{(n - 1) \log(2/\delta)/2}$$

Consider a record $u \in V$ with rank at most $5\sqrt{2n \log(2/\delta)}$. Then,

$$E[\text{Count}(u, V)] = \sum_{w \in V} 1\{O(u, v_{\max}) == w\} = (n - \text{rank}(u))(1 - p) + (\text{rank}(u) - 1)p$$

Using Hoeffding’s inequality, with probability $1 - \delta/2$:

$$\text{Count}(u, V) < (n - 1)(1 - p) - (\text{rank}(u) - 1)(1 - 2p) + \sqrt{0.5(n - 1) \log(2/\delta)}$$

$$< (n - 1)(1 - p) - (5\sqrt{2n \log(2/\delta)} - 1)(1 - 2p) + \sqrt{0.5(n - 1) \log(2/\delta)}$$

$$< \text{Count}(v_{\max}, V)$$

The last inequality is true for a value of $p \leq 0.4$. As Algorithm 2 returns the record $u_{\max}$ with maximum Count value, we have that $\text{rank}(v_{\max}, V) = O(\sqrt{n \log(1/\delta)})$. Using union bound, for the above conditions to be met, we have the claim.

To improve the query complexity, we use an early stopping criteria that discards a value $x$ using the Count $(x, V)$ when it determines that $x$ has no chance of being the maximum. Algorithm 12 presents the pseudocode for this modified count calculation. We sample $100 \log(n/\delta)$ values randomly, denoted by $S_t$ and compare every non-sampled point with $S_t$. We argue that by doing so, it helps us eliminate the values that are far away from the maximum in the sorted ranking. Using Algorithm 12, we compare the Count scores with respect to $S_t$ of a value $u \in V \setminus S_t$ and if $\text{Count}(u, S_t) \geq 50 \log(n/\delta)$, we make it available for the subsequent iterations.

**Algorithm 12 COUNT-MAX-PROB : Maximum with Probabilistic Noise**

1. **Input**: A set $V$ of $n$ values, failure probability $\delta$.
2. **Output**: An approximate maximum value of $V$.
3. $t \leftarrow 1$
4. while $t < \log(n)$ or $|V| > 100 \log(n/\delta)$ do
5. $S_t$ denote a set of $100 \log(n/\delta)$ values obtained by sampling uniformly at random from $V$ with replacement.
6. Set $X \leftarrow \phi$
7. for $u \in V \setminus S_t$ do
8. if $\text{Count}(u, S_t) \geq 50 \log(n/\delta)$ then
9. $X \leftarrow X \cup \{u\}$
10. $V \leftarrow X, t \leftarrow t + 1$
11. $u_{\max} \leftarrow \text{COUNT-MAX}(V)$
12. return $u_{\max}$

As Algorithm 12 considers each value $u \in V \setminus S_t$ by iteratively comparing it with each value $x \in S_t$ and the error probability is less than $p$, the expected count of $v_{\max}$ (if it is available) at any iteration $t$ is $(1 - p)|S_t|$. Accounting for the deviation around the expected value, we have that $\text{Count}(v_{\max}, S_t)$ is at least $50 \log(n/\delta)$ when $p \leq 0.4^2$. If a particular value $u$ has $\text{Count}(u, S_t) < 50 \log(n/\delta)$ in any iteration, i.e., then it can not be the largest value in $V$ and therefore, we remove it from the set of possible candidates for maximum. Therefore, any value that remains in $V$ after an iteration $t$, must have rank closer to that of $v_{\max}$. We argue that after every iteration, the number of candidates remaining is at most $1/60$th of the possible candidates.

**Lemma A.10**. In an iteration $t$ containing $n_t$ remaining records, using Algorithm 5, with probability $1 - \delta/n$, we discard at least $\frac{50}{60} \cdot n_t$ records.

**Proof.** Consider an iteration $t$ which has $n_t$ remaining records. Algorithm 5 and a record $u$ with rank $\alpha \cdot n_t$. Now, we have:

$$E[\text{Count}(u, S_t)] = ((1 - \alpha)(1 - p) + \alpha p)100 \log(n/\delta)$$

For $\alpha = 0$, i.e., we have for maximum value $v_{\max}$.

$$E[\text{Count}(v_{\max}, S_t)] = (1 - p)100 \log(n/\delta)$$

Using $p \leq 0.4$ and Hoeffding’s inequality, with probability $1 - \delta/n^2$, we have:

$$\text{Count}(v_{\max}, S_t) \geq (1 - p)100 \log(n/\delta) - \sqrt{100 \log(n/\delta)} \geq 50 \log(n/\delta)$$

The constants 50, 100 etc. are not optimized and set just to satisfy certain concentration bounds.
For \( u \), we calculate the Count value. Using \( p \leq 0.4 \) and Hoeffding’s inequality, with probability \( 1 - \delta/n^2 \), we have:

\[
\text{Count}(u, S_t) < ((1 - \alpha)(1 - p) + \alpha p)100\log(n/\delta) + \sqrt{100((1 - \alpha)(1 - p) + \alpha p)\log(n/\delta)}
\]

\[
< ((1 - 0.6\alpha)100 + \sqrt{100(1 - 0.6\alpha)\log(n/\delta)}) \log(n/\delta) < 50\log(n/\delta)
\]

Upon calculation, for \( \alpha > \frac{20}{60} \), we have the above expression. Therefore, using union bound, with probability \( 1 - O(\delta/n) \), all records \( u \) with rank at least \( \frac{50n}{60} \) satisfy:

\[
\text{Count}(u, S_t) < \text{Count}(v_{\text{max}}, S_t)
\]

So, all such values can be removed. Hence, the claim. □

In the previous lemma, we argued that in every iteration, at least \( 1/60 \)th fraction is removed and therefore in \( \Theta(\log n) \) iterations, the algorithm will terminate. In each iteration, we discard the sampled values \( S_t \) to ensure that there is no dependency between the Count scores, and our guarantees hold. As we remove at most \( O(t \cdot \log(n/\delta)) = O(\log^2(n/\delta)) \) sampled points, our final statement of the result is:

**Lemma A.11.** Query complexity of Algorithm 5 is \( O(n \cdot \log^2(n/\delta)) \) and \( u_{\text{max}} \) satisfies \( \text{rank}(u_{\text{max}}, V) \leq O(\log^2(n/\delta)) \) with probability \( 1 - \delta \).

**Proof.** From Lemma A.10, we have with probability \( 1 - \delta/n \), after iteration \( t \), at least \( \frac{50n}{60} \) records removed along with the 100 \( \log(n/\delta) \) records that are sampled. Therefore, we have:

\[
n_{t+1} \leq n_t - 100\log(n/\delta)
\]

After \( \log(n/\delta) \) iterations, we have \( n_{t+1} \leq 1 \). As we have removed \( \log_{60} n \cdot 100\log(n/\delta) \) records that were sampled in total, these could records with rank \( \leq 100\log^2(n/\delta) \). So, the rank of \( u_{\text{max}} \) output is at most \( 100\log^2(n/\delta) \). In an iteration \( t \), the number of oracle queries calculating Count values is \( O(n_t \cdot \log(n/\delta)) \). In total, Algorithm 5 makes \( O(n \log^2(n/\delta)) \) oracle queries. Using union bound over \( \log(n/\delta) \) iterations, we get a total failure probability of \( \delta \). □

**Theorem A.12 (Theorem 3.6 restated).** There is an algorithm that returns \( u_{\text{max}} \in V \) such that \( \text{rank}(u_{\text{max}}, V) = O(\log^2(n/\delta)) \) with probability \( 1 - \delta \) and requires \( O(n \log^2(n/\delta)) \) oracle queries.

**Proof.** The proof follows from Lemma A.11 □

**B FARthest AND NEAREST NeighBoR**

**Lemma B.1 (Lemma 3.7 restated).** Suppose \( \max_{v_i \in S} d(u, v_i) = \alpha \) and \( |S| \geq 6 \log(1/\delta) \). Consider two records \( v_i \) and \( v_j \) such that \( d(u, v_i) < d(u, v_j) - 2\alpha \) then \( \text{FCount}(v_i, v_j) \geq 0.3|S| \) with a probability of \( 1 - \delta \).

**Proof.** Since \( d(u, v_i) < d(u, v_j) - 2\alpha \), for a point \( x \in S \),

\[
d(v_j, x) \geq d(v_i, x) + 2\alpha - d(u, x)
\]

\[
\geq d(v_i, x) - d(u, x) + 2\alpha - d(u, x)
\]

\[
\geq d(v_i, x) + 2\alpha - 2d(u, x)
\]

So, \( O(v_i, x, v_j, x) \) is no with a probability \( p \). As \( p \leq 0.4 \), we have:

\[
E[\text{FCount}(v_i, v_j)] = (1 - p)|S|
\]

\[
\Pr[\text{FCount}(v_i, v_j) \leq 0.3|S|] \leq \Pr[\text{FCount}(v_i, v_j) \leq (1 - p)|S|/2]
\]

From Hoeffding’s inequality (with binary random variables), we have with a probability \( \exp(-\frac{|S|(1-p)^2}{2}) \leq \delta \) (using \( |S| \geq 6 \log(1/\delta) \), \( p < 0.4 \)):

\[
\text{FCount}(v_i, v_j) \leq (1 - p)|S|/2.
\]

Therefore, with probability at most \( \delta \), we have, \( \text{FCount}(v_i, v_j) \leq 0.3|S| \). □

For the sake of completeness, we restate the Count definition that is used in Algorithm Count-Max. For every oracle comparison, we replace it with the pairwise comparison query described in Section 3.3. Let \( u \) be a point and \( R(u) \) denote a set of \( \Theta(\log(1/\delta)) \) points within a distance of \( \alpha \) from \( u \). We maintain a Count score for every point \( v \in V \), as given by:

\[
\text{Count}(v, V) = \sum_{v_j \in V} 1\{\text{PAIRWISE-COMP}(u, v_i, v_j, R(u)) = \text{No}\}
\]

**Lemma B.2.** Given a set of values \( V \), and a query \( u \), COUNT-MAX(V) returns a \( 4\alpha \) additive approximation of farthest point from \( u \) in \( V \) using \( O(|V|^2 \log |S|) \) oracle queries.

**Theorem B.3 (Theorem 3.8 restated).** Given a query vertex \( u \) and a set \( S \) with \( |S| = \Omega(\log(1/\delta)) \) such that \( \max_{v \in S} d(u, v) \leq \alpha \) then the farthest identified using Algorithm 4 (with PairwiseComp), denoted by \( u_{\text{max}} \) is within \( 6\alpha \) distance from the optimal farthest point, i.e.,

\[
d(u, u_{\text{max}}) \geq \max_{v \in V} d(u, v) - 6\alpha \text{ with a probability of } 1 - \delta.
\]

Further the query complexity is \( O(n \log^3(1/\delta)) \).
Algorithm 6 be denoted by $S$. Therefore, every point is assigned to a cluster with distance at most $2$.

However, when we output the final clusters and centers, the farthest point after $S$ is a $k$-approximation to the correct assignment. In iteration $t$, consider an optimum clustering $C^*$ with centers $u_1, u_2, \ldots, u_k$ respectively: $C^*(u_1), C^*(u_2), \ldots, C^*(u_k)$. Let the centers obtained by Algorithm 6 be denoted by $S$. If $|S \cap C^*(u_i)| = 1$ for all $i$, then, for some point $x \in C^*(u_i)$ assigned to $s_j \in S$ by Algorithm ASSIGN, we have

$$d(x, S \cap C^*(u_i)) \leq d(x, u_i) + d(u_i, S \cap C^*(u_i)) \leq 2OPT$$

This implies that $S$ is a $2\beta^2$-approximation of the distance to true center. Therefore, every point in $V$ is at a distance of at most $2\beta OPT$ from a center assigned in $S$.

Suppose for some $j$ we have $|S \cap C^*(u_j)| \geq 2$. Let $s_1, s_2 \in S \cap C^*(u_j)$ and $s_2$ appeared after $s_1$ in iteration $t + 1$. As $s_1 \in S_t$, we have $\min_{s \in S_t} d(s, s_2) \leq d(s_1, s_2)$. In iteration $t$, we know that the farthest point $s_2$ is an $\alpha\beta OPT$-approximation of the farthest point (say $f_j$). Moreover, suppose $s_2$ assigned to cluster with center $s_k$ in iteration $t$ that is a $\beta$-approximation of it’s true center. Therefore,

$$\frac{1}{\alpha} \min_{w \in S_t} d(w, f_j) \leq d(s_k, s_2) \leq \beta \min_{w \in S_t} d(w, s_2) \leq \beta \text{dist}(s_1, s_2)$$

Because $s_1$ and $s_2$ are in the same optimum cluster, from triangle inequality we have $d(s_1, s_2) \leq 2OPT$. Combining all the above we get $\min_{w \in S_t} d(w, f_j) \leq 2\alpha\beta OPT$ which means that farthest point of iteration $t$ is at a distance of $2\alpha\beta OPT$ from $S_t$. In the subsequent iterations, the distance of any point to the final set of centers, given by $S$ only gets smaller. Hence,

$$\max \min_{v \in S} d(v, w) \leq \max \min_{v \in S} d(v, w) \leq \min \min_{w \in S_t} d(v, w) \leq 2\alpha\beta OPT$$

However, when we output the final clusters and centers, the farthest point after $k$-iterations (say $f_k$) could be assigned to center $v_j \in S$ that is a $\beta$-approximation of the distance to true center.

$$d(f_k, v_j) \leq \beta \min_{w \in S} d(f_k, w) \leq 2\alpha\beta^2 OPT$$

Therefore, every point is assigned to a cluster with distance at most $2\alpha\beta^2 OPT$. Hence the claim.

**Lemma C.2.** Given a set $S$ of centers, Algorithm ASSIGN assigns a point $u$ to a cluster $s_j \in S$ such that $d(u, s_j) \leq (1 + \mu)^3 \min_{s \in S} \{d(u, s_j)\}$ using $O(nk)$ queries.

**Proof.** The proof is essentially the same as Lemma A.3 and uses MCount instead of Count.

**Lemma C.3.** Given a set of centers $S$, Algorithm 4 identifies a point $v_j$ with probability $1 - \delta/k$, such that

$$\min_{s_j \in S} d(v_j, s_j) \geq \max_{x \in V, s_j \in S} \frac{d(v_j, s_j)}{(1 + \mu)^3}$$

**Proof.** Suppose $v_j$ is the farthest point assigned to center $s_j \in S$. Let $v_j$, assigned to $s_j \in S$ be the point returned by Algorithm 4. From Theorem A.8, we have:

$$d(v_j, s_j) \geq \frac{\max_{s_j \in V} d(v_j, s_j)}{(1 + \mu)^3} \geq \frac{d(v_j, s_j)}{(1 + \mu)^3} \geq \frac{\min_{s_j' \in S} d(v_j, s_j')}{(1 + \mu)^3}$$

Due to error in assignment, using Lemma C.2

$$d(v_j, s_j) \leq (1 + \mu)^3 \min_{s_j' \in S} d(v_j, s_j')$$

Combining the above equations we have

$$\min_{s_j' \in S} d(v_j, s_j') \geq \frac{\min_{s_j' \in S} d(v_j, s_j')}{(1 + \mu)^3}$$

For APPROX-FARthest, we use $l = \sqrt{n}$ and $t = \log(2k/\delta)$ and $\tilde{V} = \sqrt{n}t$. So, following the proof in Theorem 3.5, we succeed with probability $1 - \delta/k$. Hence the lemma.

**Lemma C.4 (Lemma 4.1 restated).** Given a current set of centers $S$,
(1) **ASSIGN** assigns a point $u$ to a cluster $C(s_i)$ such that $d(u, s_i) \leq (1 + \mu)^2 \min_{s_j \in S} (d(u, s_j))$ using $O(nk)$ oracle queries additionally.

(2) **APPROX-FARTHEST** identifies a point $w$ in cluster $C(s_i)$ such that $\min_{s_j \in S} d(w, s_j) \geq \max_{y \in V} \min_{s_j \in S} d(s_j, s_i)/(1 + \mu)^3$ with probability $1 - \frac{\delta}{k}$ using $O(n \log^2 (k/\delta))$ oracle queries.

**Proof.** (1) From Lemma C.2, we have the claim. We assign a point to a cluster based on the scores the cluster center received in comparison to other centers. Except for the newly created center, we have previously queried every center with every other center. Therefore, number of new oracle queries made for every point is $O(k)$; that gives us a total of $O(nk)$ additional new queries used by **ASSIGN**.

(2) From Lemma C.3, we have that $\min_{s_j \in S} d(w, s_j) \geq \max_{y \in V} \min_{s_j \in S} d(s_j, s_i)/(1 + \mu)^3$ with probability $1 - \delta/k$. As the total number of queries made by Algorithm 4 is $O(n(\mu + (\mu^2 N) + \sqrt{n}t)^2)$. For **APPROX-FARTHEST**, we use $l = \sqrt{n}$ and $t = \log(2k/\delta)$ and $\tilde{V} = \sqrt{n}t$, therefore, the query complexity is $O(n \log^2 (k/\delta))$.

\[\square\]

**D. \(k\)-CENTER : PROBABILISTIC NOISE**

**D.1 Sampling**

**Lemma D.1.** Consider the sample $\tilde{V} \subseteq V$ of points obtained by selecting each point with a probability $\frac{450 \log(n/\delta)}{m}$. Then, we have $\frac{400n \log(n/\delta)}{m} \leq |\tilde{V}| \leq \frac{500n \log(n/\delta)}{m}$ and for every $i \in [k]$, $|C^*(s_i) \cap \tilde{V}| \geq 400 \log(n/\delta)$ with probability $1 - O(\delta)$ for sufficiently large $\gamma > 0$.

**Proof.** We include every point in $\tilde{V}$ with a probability $\frac{450 \log(n/\delta)}{m}$ where the size of the smallest cluster is $m$. Using Chernoff bound, with probability $1 - O(\delta)$, we have:

$$\frac{400n \log(n/\delta)}{m} \leq |\tilde{V}| \leq \frac{500n \log(n/\delta)}{m}$$

Consider an optimal cluster $C^*(v_i)$ with center $v_i$. As every point is included with probability $\frac{450 \log(n/\delta)}{m}$:

$$E[|C^*(s_i) \cap \tilde{V}|] = |C^*(s_i)| \cdot \frac{450 \log(n/\delta)}{m} \geq 450 \log(n/\delta)$$

Using Chernoff bound, with probability at least $1 - \delta/n$, we have

$$|C^*(s_i) \cap \tilde{V}| \geq 400 \log(n/\delta)$$

Using union bound for all the $k$ clusters, we have the lemma. \[\square\]

**D.2 Assignment**

$$A\text{Count}(u, s_j, s_i) = \sum_{x \in R(s_i)} 1\{O(u, x, u, s_j) == \text{Yes}\}$$

**Lemma D.2.** Consider a point $u$ and $s_j \neq s_i$ such that $d(u, s_j) \leq 2 \text{OPT} + d(u, s_i)$ and $r(s_i) \geq 12 \log(n/\delta)$, then, $A\text{Count}(u, s_j, s_i) \geq 0.3|r(s_i)|$ with a probability of $1 - \frac{\delta}{n^2}$.

**Proof.** Using triangle inequality, for any $x \in R(s_i)$

$$d(u, x) \leq d(u, s_i) + d(s_i, x) \leq d(u, s_j) - 2 \text{OPT} + d(s_i, x) \leq d(u, s_j)$$

So, $O(u, x, s_i)$ is Yes with a probability at least $1 - p$. We have:

$$E[A\text{Count}(u, s_j, s_i)] = \sum_{x \in R(s_i)} E[1\{O(u, x, u, s_j) == \text{Yes}\}] \geq (1 - p)|R(s_i)|$$

Using Hoeffding’s inequality, with a probability of $\exp(-|R(s_i)|(1 - p)^2/2) \leq \frac{\delta}{n^2}$ (using $p \leq 0.4$), we have

$$A\text{Count}(u, s_i, s_j) \leq (1 - p)|R(s_i)|/2$$

We have $Pr[A\text{Count}(u, s_j, s_i) \leq 0.3|R(s_i)|] \leq Pr[A\text{Count}(u, s_i, s_j) \leq (1 - p)|S_i|/2]$. Therefore, with probability $\frac{\delta}{n^2}$, we have $A\text{Count}(u, s_j, s_i) \leq 0.3|R(s_i)|$. Hence, the lemma.

**Lemma D.3.** Suppose $u \in C^*(s_i)$ and for some $s_j \in S$, if $d(s_i, s_j) \geq 6 \text{OPT}$, then, Algorithm 8 assigns $u$ to center $s_i$ with probability $1 - \frac{\delta}{n^2}$.

**Proof.** As $u \in C^*(s_i)$, we have $d(u, s_i) \leq 2 \text{OPT}$. Therefore,

$$d(s_j, u) - d(s_i, u) \geq d(s_i, s_j) - 2d(s_i, u) \geq 2 \text{OPT}$$

$$d(s_j, u) \geq d(s_j, u) + 2 \text{OPT}$$

From Lemma D.2, we have that if $d(u, s_j) \leq d(u, s_j) - 2 \text{OPT}$, then, we will assign $u$ to $s_i$ with probability $1 - \frac{\delta}{n^2}$. \[\square\]
Lemma D.4. Given a set of centers $S$, every $u \in V$ is assigned to a cluster $s_i$ such that $d(u, s_i) \leq \min_{s_j \in S} d(u, s_j) + 2 \text{OPT}$ with a probability of $1 - 1/n^2$.

Proof. From Lemma D.2, we have that a point $u$ is assigned to $s_j$ from $s_m$ if $d(u, s_j) \leq d(u, s_m) - 2 \text{OPT}$. If $s_j$ is the final assigned center of $u$, then, for every $s_j$, it must be true that $d(u, s_j) \geq d(u, s_j) - 2 \text{OPT}$, which implies $d(u, s_j) \leq \min_{s_j \in S} d(u, s_j) + 2 \text{OPT}$. Using union bound over at most $n$ points, we have with a probability of $1 - \frac{2}{n^2}$, every point $u$ is assigned as claimed.

D.3 Core Calculation

Consider a cluster $C(s_i)$ with center $s_i$. Let $S^d_{u}$ denote the number of points in the set $\{|x : a \leq d(x, s_i) < b\}$.

\[ \text{Count}(u) = \sum_{x \in C(s_i)} 1\{O(s_i, x, s_i, u) == \text{No}\} \]

Lemma D.5. Consider any two points $u_1, u_2 \in C(s_i)$ such that $d(u_1, s_i) \leq d(u_2, s_i)$, then $E[\text{Count}(u_1)] - E[\text{Count}(u_2)] = (1 - 2p)S^d_{d(u_1, s_i)}$

Proof. For a point $u \in C(s_i)$

\[ E[\text{Count}(u)] = E\left[ \sum_{x \in C(s_i)} 1\{O(s_i, x, s_i, u) == \text{No}\} \right] = S^d_{d(u_1, s_i)} p + S^\infty_{d(u_1, s_i)} (1 - p) \]

\[ E[\text{Count}(u_1)] - E[\text{Count}(u_2)] = \left(S^d_{d(u_1, s_i)} p + S^d_{d(u_2, s_i)} (1 - p) + S^\infty_{d(u_1, s_i)} (1 - p)\right) - \left(S^d_{d(u_1, s_i)} p + S^d_{d(u_2, s_i)} (1 - p) + S^\infty_{d(u_2, s_i)} (1 - p)\right) = (1 - 2p)S^d_{d(u_1, s_i)} \]

Lemma D.6. Consider any two points $u_1, u_2 \in C(s_i)$ such that $d(u_1, s_i) \leq d(u_2, s_i)$ and $|S^d_{d(u_1, s_i)}| \geq \sqrt{100|C(s_i)| \log(n/\delta)}$. Then, $\text{Count}(u_1) > \text{Count}(u_2)$ with probability $1 - \delta/n^2$.

Proof. Suppose $u_1, u_2 \in C(s_i)$. We have that $\text{Count}(u_1)$ and $\text{Count}(u_2)$ is a sum of $|C(s_i)|$ binary random variables. Using Hoeffding’s inequality, we have with probability $\exp(-\beta^2/2|C(s_i)|)$ that

\[ \text{Count}(u_1) \leq E[\text{Count}(u_1)] - \frac{\beta}{2} \]

\[ \text{Count}(u_2) \geq E[\text{Count}(u_2)] + \frac{\beta}{2} \]

Using union bound, with probability at least $1 - 2 \exp(-\beta^2/2|C(s_i)|)$, we can conclude that

\[ \text{Count}(u_1) - \text{Count}(u_2) > E[\text{Count}(u_1) - \text{Count}(u_2)] - \beta > (1 - 2p)S^d_{d(u_1, s_i)} - \beta \]

Choosing $\beta = (1 - 2p)S^d_{d(u_1, s_i)}$, we have $\text{Count}(u_1) > \text{Count}(u_2)$ with a probability (for constant $p \leq 0.4$)

\[ 1 - 2 \exp\left(-\left(1 - 2p\right)^2 \left(S^d_{d(u_1, s_i)}\right)^2 / |C(s_i)|\right) \geq 1 - 2 \exp\left(-0.02 \left(S^d_{d(u_1, s_i)}\right)^2 / |C(s_i)|\right) \]

Further, simplifying using $S^d_{d(u_1, s_i)} \geq \sqrt{100|C(s_i)| \log(n/\delta)}$, we get probability of failure is $2 \exp(-2 \log(n/\delta)) = O(\delta/n^2)$.

Lemma D.7. If $|C(s_i)| \geq 400 \log(n/\delta)$, then, $|R(s_i)| \geq 200 \log(n/\delta)$ with probability $1 - |C(s_i)|^2 \delta/n^2$.

Proof. From Lemma D.6, we have that if there are points $u_1, u_2$ with $\sqrt{100|C(s_i)| \log(n/\delta)}$ many points between them, then, we can identify the closer one correctly. When $|C(s_i)| \geq 400 \log(n/\delta)$, we have $\sqrt{100|C(s_i)| \log(n/\delta)} \geq 200 \log(n/\delta)$ points between every point and the point with the rank $200 \log(n/\delta)$. Therefore, $|R(s_i)| \geq 200 \log(n/\delta)$. Using union bound over all pairs of points in the cluster, we get the claim.

Lemma D.8. If $x \in C^*(s_i)$, then, $x \in C(s_i)$ or $x$ is assigned to a cluster $s_j$ such that $d(x, s_j) \leq 8 \text{OPT}$. 

Proof. From Lemma D.7, we have that if there are points $u_1, u_2$ with $\sqrt{100|C(s_i)| \log(n/\delta)}$ many points between them, then, we can identify the closer one correctly. When $|C(s_i)| \geq 400 \log(n/\delta)$, we have $\sqrt{100|C(s_i)| \log(n/\delta)} \geq 200 \log(n/\delta)$ points between every point and the point with the rank $200 \log(n/\delta)$. Therefore, $|R(s_i)| \geq 200 \log(n/\delta)$. Using union bound over all pairs of points in the cluster, we get the claim.
**Proof.** If \( x \in C^*(s_i) \), we argue that it will be assigned to \( C(s_j) \). For the sake of contradiction, suppose \( x \) is assigned to a cluster \( C(s_j) \) for some \( s_j \in S \). We have \( d(x, s_i) \leq 2 \text{OPT} \) and let \( d(s_i, s_j) \geq 6 \text{OPT} \)

\[
d(s_i, s_j) \leq d(s_i, x) + d(s_i, x)
\]

\[
d(s_j, x) \geq 4 \text{OPT}
\]

However, we know that \( d(s_j, x) \leq d(s_i, x) + 2 \text{OPT} \leq 4 \text{OPT} \) from Lemma D.2. We have a contradiction. Therefore, \( x \) is assigned to \( s_j \). If \( d(s_j, s_j) \leq 6 \text{OPT} \), we have \( d(x, s_j) \leq d(x, s_i) + 2 \text{OPT} \leq 8 \text{OPT} \). Hence, the lemma. \( \square \)

### D.4 Farthest point computation

Let \( R(s_j) \) represent the core of the cluster \( C(s_j) \) and contains \( \Theta(\log(n/\delta)) \) points. We define \( \text{FCount} \) for comparing two points \( v_i, v_j \) from their centers \( s_i, s_j \) respectively. If \( s_i \neq s_j \), we let:

\[
\text{FCount}(v_i, v_j) = \sum_{x \in R(s_i), y \in R(s_j)} 1\{O(v_i, x, v_j, y) == \text{Yes}\}
\]

Otherwise, we let \( \text{FCount}(v_i, v_j) = \sum_{x \in R(s_i)} 1\{O(v_i, x, v_j, x) == \text{Yes}\} \). First, we observe that each of the summation is over \(|R(s_i)|\) many terms, because \(|R(s_i)| = \sqrt{|R(s_i)|}\).

**Lemma D.9.** Consider two records \( v_i, v_j \) in different clusters \( C(s_i), C(s_j) \) respectively such that \( d(s_i, v_i) < d(s_j, v_j) - 4 \text{OPT} \) then \( \text{FCount}(v_i, v_j) \geq 0.3|\bar{R}(s_i)||\bar{R}(s_j)| \) with a probability of \( 1 - \frac{\delta}{n^2} \).

**Proof.** We know \( \max_{v_i \in \bar{R}(s_i)} d(u, v_i) \leq 2 \text{OPT} \) and \( \max_{v_j \in \bar{R}(s_j)} d(v_j, s_j) \leq 2 \text{OPT} \). For a point \( x \in R(s_i), y \in \bar{R}(s_j) \)

\[
d(v_j, y) \geq d(s_j, v_j) - d(s_j, y)
\]

\[
> d(v_i, s_j) + 4 \text{OPT} - d(s_j, y)
\]

\[
> d(v_i, x) - d(x, s_j) + 4 \text{OPT} - d(s_j, y)
\]

\[
> d(v_i, x)
\]

So, \( O(v_i, x, v_j, y) \) is No with a probability \( p \). As \( p \leq 0.4 \), we have:

\[
E[\text{FCount}(v_i, v_j)] = (1 - p)|\bar{R}(s_i)||\bar{R}(s_j)|
\]

\[
\Pr[\text{FCount}(v_i, v_j) \geq 0.3|\bar{R}(s_i)||\bar{R}(s_j)|] \leq \Pr[\text{FCount}(v_i, v_j) \leq (1 - p)|\bar{R}(s_i)||\bar{R}(s_j)|/2]
\]

From Hoeffding’s inequality (with binary random variables), we have with a probability \( \exp(-\frac{|\bar{R}(s_i)||\bar{R}(s_j)|(1-p)^2}{2}) \leq \frac{\delta}{n^2} \) (using \(|\bar{R}(s_i)||\bar{R}(s_j)| \geq 12\log(n/\delta), \ p < 0.4 \) : \( \text{FCount}(v_i, v_j) \leq (1 - p)|\bar{R}(s_i)||\bar{R}(s_j)|/2 \). Therefore, with probability at most \( \delta/n^2 \), we have, \( \text{FCount}(v_i, v_j) \geq 0.3|\bar{R}(s_i)||\bar{R}(s_j)| \).

\( \square \)

In order to calculate the farthest point, we use the ideas discussed in Section 3 to identify the point that has the maximum distance from its assigned center. As noted in Section 3.3, our approximation guarantees depend on the maximum distance of points in the core from the center. In the next lemma, we show that assuming a maximum distance of a point in the core (See Lemma D.8), we can obtain a good approximation for the farthest point.

**Lemma D.10.** Let \( \max_{s_j \in S, u \in \bar{R}(s_j)} d(u, s_j) \leq \alpha \). In every iteration, if the farthest point is at a distance more than \((6\alpha + 3 \text{OPT})\), then, \( \text{APPROX-FARTEST} \) outputs a \((6\alpha/\text{OPT} + 3)\)-approximation. Otherwise, the point output is at most \((6\alpha + 3 \text{OPT})\) away.

**Proof.** The farthest point output \( \text{APPROX-FARTEST} \) is a \( 6\alpha \) additive approximation. However, the assignment of points to the cluster also introduces another additive approximation of \( 2 \text{OPT} \), resulting in a total \( 6\alpha + 2 \text{OPT} \) approximation. Suppose in the current iteration, the distance of the farthest point is \( \beta \text{OPT} \), then the point output by \( \text{APPROX-FARTEST} \) is at least \( \beta \text{OPT} - (6\alpha + 2 \text{OPT}) \). So, the approximation ratio is \( \frac{\beta}{\beta - (6\alpha + 2 \text{OPT})} \). If \( \beta \text{OPT} \geq 6\alpha + 3 \text{OPT} \), we have \( \frac{\beta \text{OPT}}{\beta \text{OPT} - (6\alpha + 2 \text{OPT})} \leq \beta \). As we are trying to minimize the approximation ratio, we set \( \beta \text{OPT} = 6\alpha + 3 \text{OPT} \) and get the claimed guarantee. \( \square \)
D.5 Final Guarantees

Throughout this section, we assume that $m = \Omega\left(\frac{\log^3(n/\delta)}{\delta}\right)$ for a given failure probability $\delta > 0$.

**Lemma D.11.** Given a current set of centers $S$, and $\max_{v_j \in S, u \in R(v_j)} d(u, v_j) \leq \alpha$, we have:

1. Every point $u$ is assigned to a cluster $C(s_i)$ such that $d(u, s_i) \leq \min_{v_j \in S} d(u, v_j) + 2\OPT$ using $O(nk \log(n/\delta))$ oracle queries with probability $1 - O(\delta)$.
2. APPROX-Farthest identifies a point $w$ in cluster $C(s_i)$ such that $\min_{v_j \in S} d(w, v_j) \geq \max_{v_j \in V} \min_{s_j \in S} d(v_j, s_j)/(6\alpha/\OPT + 3)$ with probability $1 - O(\delta/k)$ using $O(|V| \log^3(n/\delta))$ oracle queries.

**Proof.** (1) First, we argue that cores are calculated correctly. From Lemma D.3, we have that a point $u \in C^*(s_i)$ is assigned to the center correctly $s_i$. Therefore, all the points from $V \cap C^*(S_i)$ move to $C(S_i)$. As the size of $|C(S_i)| \geq |V \cap C^*(S_i)| \geq 400 \log(n/\delta)$, we have $|R(s_i)| \geq 200 \log(n/\delta)$ with a probability $1 - |C(s_i)|^2/\delta/\log(n/\delta)$ (from Lemma D.6). Using union bound, we have that all the cores are calculated correctly with a failure probability of $\sum_{i} |C(s_i)|^2/\delta^2 = \delta$.

For every point, we compare the distance with every cluster center by maintaining a center that is the current closest. From Lemma D.2, we have that the total number of oracle queries is $O(kn\delta/\delta^2) = O(kn\delta/\delta^2)$. For the calculation of core, the query complexity is $O(|V|^2k)$. For assignment, the query complexity is $O(nk \log(n/\delta))$. Therefore, total query complexity is $O(nk \log(n/\delta) + \frac{n^2k}{m}k \log^2(n/\delta)) = O(nk \log(n/\delta) + \frac{n^2k}{m}k \log^2(n/\delta))$.

**Theorem D.12.** [Theorem 4.3 restated] Given $p \leq 0.4$, a failure probability $\delta$, and $m = \Omega\left(\frac{\log^3(n/\delta)}{\delta}\right)$. Then, Algorithm 7 achieves a $O(1)$-approximation for the $k$-center objective using $O(nk \log(n/\delta) + \frac{n^2k}{m}k \log^2(n/\delta))$ oracle queries with probability $1 - O(\delta)$.

**Proof.** Using similar proof as Lemma D.11, we have that the approximation ratio of Algorithm 7 is $4(6\alpha/\OPT + 3) + 2$. Using $\alpha = 8$ OPT from Lemma D.8, we have that the approximation factor is 206. For the first stage, from Lemma D.11, we have that for all the $k$ iterations, the number of oracle queries is $O(|V| \log^3(n/\delta))$. Using union bound over $k$ iterations, success probability is $1 - O(\delta)$. For the calculation of core, the query complexity is $O(|V|^2k)$. For assignment, the query complexity is $O(nk \log(n/\delta))$. Therefore, total query complexity is $O(nk \log(n/\delta) + \frac{n^2k}{m}k \log^2(n/\delta)) = O(nk \log(n/\delta) + \frac{n^2k}{m}k \log^2(n/\delta))$.

**E HIERARCHICAL CLUSTERING**

**Lemma E.1 (Lemma 5.1 restated).** Given a collection of clusters $C = \{C_1, \ldots, C_r\}$, our algorithm to calculate the closest pair (using Algorithm 4) identifies $C_1$ and $C_2$ to merge according to single linkage objective if $d_{SL}(C_1, C_2) \leq (1 + \mu) \min_{C_i, C_j \in C} d(C_i, C_j)$ with $1 - \delta$ probability and requires $O(r^2 \log^2(n/\delta))$ queries.

**Proof.** In each iteration, our algorithm considers a list of $\binom{r}{2}$ distance values and calculates the closest using Algorithm 4. The claim follows from the proof of Theorem 3.5.

Using the same analysis, we get the following result for complete linkage.

**Lemma E.2.** Given a collection of clusters $C = \{C_1, \ldots, C_r\}$, our algorithm to calculate the closest pair (using Algorithm 4) identifies $C_1$ and $C_2$ to merge according to complete linkage objective if $d_{CL}(C_1, C_2) \leq (1 + \mu) \min_{C_i, C_j \in C} d(C_i, C_j)$ with $1 - \delta$ probability and requires $O(r^2 \log^2(n/\delta))$ queries.

**Theorem E.3 (Theorem 5.2 restated).** In any iteration, suppose the distance between a cluster $C_i \in C$ and its identified nearest neighbor $\tilde{C}_j$ is $\alpha$-approximation of its distance from the optimal nearest neighbor, then the distance between pair of clusters merged by Algorithm 11 is $\alpha(1 + \mu)^3 \alpha$-approximation of the optimal distance between the closest pair of clusters in $C$ with a probability of $1 - \delta$ using $O(n \log^2(n/\delta))$ oracle queries.

**Proof.** Algorithm 11 iterates over the list of pairs $(C_i, \tilde{C}_i), \forall C_i \in C$ and identifies the closest pair using Algorithm 4. The claim follows from the proof of Theorem 3.5.