How to Design Robust Algorithms using Noisy Comparison Oracle

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 and farthest 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 a 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 problems, 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.

PVLDB Reference Format:
Raghavendra Addanki, Sainyam Galhotra, Barna Saha. How to Design Robust Algorithms using Noisy Comparison Oracle. PVLDB, 14(10): 1703 - 1716, 2021.
doi:10.14778/3467861.3467862

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 is 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 [26, 40]. 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.

Proceedings of the VLDB Endowment, Vol. 14, No. 10 ISSN 2150-8097.
doi:10.14778/3467861.3467862

Figure 1: Data summarization example

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 [8]. 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 [21, 56, 58]. 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?'[7, 18, 43, 58]. 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 [21, 56]. 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.

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 the 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). We ran a user experiment by querying crowd workers to answer simple Yes/No questions to help summarize the data (Please refer to Section 6.2 for more details).

In this example, we make the following observations.

• **Automated clustering techniques generate noisy clusters.** Consider the greedy approach for k-center clustering [28] 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, thereby leading to poor performance. In general, automated techniques are known to generate erroneous similarity values between records due to missing information or even presence of noise [20, 57, 59]. Even Google's landmark detection API [1] did not identify the locations of images 4 and 5.

• **Answering pairwise optimal cluster query is infeasible.** Answering whether 1 and 3 belong to the same optimal cluster when presented in isolation is impossible unless the crowd worker is aware of other records present in the dataset, and the granularity of the optimum clusters. Using the pair-wise Yes/No answers obtained from the crowd workers for the $\binom{6}{2}$ pairs in this example, the identified clusters achieved 0.40 F-score for $k = 3$. Please refer to Section 6.2 for additional details.

• **Comparing relative distance between the locations is easy.** Answering *relative distance* queries of the form 'Is 1 closer to 3, or is 5 closer to 6?' does not require any extra knowledge about other records in the dataset. For the 6 images in the example, we queried relative distance queries and the final clusters constructed for $k = 3$ achieved an F-score of 1.

In summary, we observe that humans have an innate understanding of the domain knowledge and can answer *relative distance queries* between records easily. Motivated by the aforementioned observations, we consider a *quadruplet* comparison oracle that compares the relative distance between two pairs of points (u_1, u_2) and (v_1, v_2) and outputs the pair with smaller distance between them breaking ties arbitrarily. Such oracle models have been studied extensively in the literature [12, 18, 25, 33, 35, 49, 50]. Even though quadruplet queries are easier than binary optimal queries, some queries maybe harder to answer. In a quadruplet query, if there is a significant gap between the two distances being compared, then they might be easier to answer [10, 16]. However, when the two distances are close, the chances of an error could increase. For e.g., '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 [10, 11, 21, 25, 43, 47] since the error due to oracles often does not change with repetition, and in fact, sometimes increases upon repeated querying [21, 43, 47]. This is in contrast to the noise models studied in [18] where response to every query is independently noisy. Persistent query models are more 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 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 have provable approximation guarantees for both the noise models, are efficient and obtain good query complexity. We empirically evaluate the robustness and efficiency of our techniques on real world datasets.

(i) **Maximum, Farthest and Nearest Neighbor**: Finding maximum has received significant attention under both adversarial and probabilistic model [5, 10, 16, 19, 22–24, 39]. In this paper, we provide the following results.

• **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 $1 - \delta$[1] using $O(n \log^2(1/\delta))$ oracle queries and running time (Theorem 3.6).

• **Maximum under probabilistic model.** We present an algorithm that requires $O(n \log^2(n/\delta))$ queries to identify $O(\log^2(n/\delta))$th rank value with probability $1 - \delta$ (Theorem 3.7). That is, in $O(n \log^2(n))$ time we can identify $O(\log^2(n))$th value in the sorted order with probability $1 - \frac{1}{n^c}$ for any constant c.

To contrast our results with the state of the art, Ajtai et al. [5] study a slightly different additive adversarial error model where the answer of a maximum query is correct if the compared values differ by θ (for some $\theta > 0$) and otherwise the oracle answers adversarially. Under this setting, they give an additive 3θ-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 3.10).

For the probabilistic model, after a long series of works [10, 22, 24, 39], only recently an algorithm is proposed with query complexity $O(n \log n)$ that returns an $O(\log n)$th rank value with probability $1 - \frac{1}{n}$ [23]. Previously, the best query complexity was $O(n^{3/2})$ [24]. While our bounds are slightly worse than [23], our algorithm is significantly simpler.

As discussed earlier, persistent errors are much more difficult to handle than independent errors [16, 19]. In [19], when the answers are independent,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.

• **Nearest Neighbor.** Nearest neighbor queries can be cast as "finding minimum" among a set of distances. We can obtain bounds similar to finding maximum for the nearest neighbor queries. In the

[1] δ is the confidence parameter and is standard in the literature of randomized algorithms.

adversarial model, we obtain an $(1+\mu)^3$-approximation, and in the probabilistic model, we are guaranteed to return an element with rank $O(\log^2(n/\delta))$ with probability $1-\delta$ using $O(n\log^2(1/\delta))$ and $O(n\log^2(n/\delta))$ oracle queries respectively.

Prior techniques have studied nearest neighbor search under noisy distance queries [42], where the oracle returns a noisy estimate of a distance between queried points, and repetitions are allowed. Neither the algorithm of [42], nor other techniques developed for maximum [5, 19] and top-k [16] extend for nearest neighbor under our noise models.

• **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 $O(\log n)$th rank element is insufficient. Since distances on a metric space satisfies triangle inequality, we exploit that to get a constant approximation to the farthest query under the probabilistic model and a mild distribution assumption (Theorem 3.10).

(ii) **k-center Clustering**: k-center clustering is one of the fundamental models of clustering and is very well-studied [53, 60].

• **k-center under adversarial model** We design algorithm that returns a clustering that is a $2+\mu$ approximation for small values of μ with probability $1-\delta$ using $O(nk^2 + nk\log^2(k/\delta))$ queries (Theorem 4.2). In contrast, even when exact distances are known, k-center cannot be approximated better than a 2-factor unless $P = NP$ [53]. Therefore, we achieve near-optimal results.

• **k-center under probabilistic noise model.** For probabilistic noise, when optimal k-center clusters are of size at least $\Omega(\sqrt{n})$, our algorithm returns a clustering that achieves constant approximation with probability $1-\delta$ using $O(nk\log^2(n/\delta))$ queries (Theorem 4.4).

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 $(1+\mu)^3$ in each merge operation and has an overall query complexity of $O(n^2)$. Prior work [25] considers comparison oracle queries to perform average linkage 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 [16]. For the problem of finding nearest neighbor, techniques based on locality sensitive hashing generally work well in practice [6]. Clustering points using k-center objective is NP-hard and there are many well known heuristics and approximation algorithms [60] with the classic 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 [14, 29, 34, 43, 44] to identify ground truth clusters. Recent techniques for distance based clustering objectives, such as k-means [7, 13, 37, 38] and k-median [4] 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 [18, 25].

Distance based comparison oracles have been used to study a wide range of problems and we list a few of them – learning fairness metrics [35], top-down hierarchical clustering with a different objective [12, 18, 25], correlation clustering [50] and classification [33, 49], identify maximum [31, 54], top-k elements [15–17, 39, 41, 46], information retrieval [36], skyline computation [55]. 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 [9, 10]. The work of [16] considers a different probabilistic noise model with error varying as a function of difference in the values but they assume that each query is independent and therefore repetition can help boost the probability of success. Using a quadruplet oracle, [25] studies the problem of recovering a 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 $val(v_i), \forall i \in [1, n]$. 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 values 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 $O : V \times V \to \{\texttt{Yes}, \texttt{No}\}$. Each oracle query considers two values as input and outputs $O(v_1, v_2) =$ `Yes` if $v_1 \leq v_2$ and* `No` *otherwise.*

Note that a comparison oracle is defined for any pair of values. 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 metric space along with a distance $d : V \times V \to \mathbb{R}^+$ 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 [6]. 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* $O : V \times V \times V \times V \rightarrow \{\texttt{Yes}, \texttt{No}\}$. *Each oracle query considers two pairs of records as input and outputs* $O(v_1, v_2, v_3, v_4) = \texttt{Yes}$ *if* $d(v_1, v_2) \leq d(v_3, v_4)$ *and* 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 $\arg\min_{u_i \in V \setminus \{q\}} d(q, u_i)$. 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^* with $C^*(v_i) \subseteq V$ 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 apriori. 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 as:

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

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 Problem 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 $\mu > 0$.

$$O(v_1, v_2) = \begin{cases} \texttt{Yes}, \text{ if } v_1 < \frac{1}{(1+\mu)} v_2 \\ \texttt{No}, \text{ if } v_1 > (1+\mu) v_2 \\ \text{adversarially incorrect if } \frac{1}{(1+\mu)} \leq \frac{v_1}{v_2} \leq (1+\mu) \end{cases}$$

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

$$O(v_1, v_2, v_3, v_4) = \begin{cases} \texttt{Yes}, \text{ if } d(v_1, v_2) < \frac{1}{(1+\mu)} d(v_3, v_4) \\ \texttt{No}, \text{ if } d(v_1, v_2) > (1+\mu) d(v_3, v_4) \\ \text{adversarially incorrect if} \frac{1}{1+\mu} \leq \frac{d(v_1, v_2)}{d(v_3, v_4)} \leq 1 + \mu \end{cases}$$

The second model considers a probabilistic noise model where each comparison query is incorrect independently with a probability $p < \frac{1}{2}$ and asking the same query multiple times yields the same response. We discuss ways to estimate μ and p from real data in Section 6.

3 FINDING MAXIMUM

In this section, we present robust algorithms to identify the record corresponding to the 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. We note that our algorithms for the adversarial model are parameter free (do not depend on μ) and the algorithms for the probabilistic model can use $p = 0.5$ as a worst case estimate of the noise.

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 $v_{max}/(1+\mu)^{n-1}$. To see this, assume $v_1 = 1$, and $v_i = (1 + \mu - \epsilon)^i$ where $\epsilon > 0$ is very close to 0. It is possible that while comparing v_i and v_{i+1}, the oracle returns v_i as the larger element. If this mistake is repeated for every i, then, v_1 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 $\texttt{Count}(v, S)$ that is equal to the number of values smaller than v in S.

$$\texttt{Count}(v, S) = \sum_{x \in S \setminus \{v\}} 1\{O(v, x) == \texttt{No}\}$$

It is easy to observe that when the oracle makes no mistakes, $\texttt{Count}(s_{\max}, S) = |S| - 1$ 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 Count-Max(V) obtained using Algorithm 1 always returns a good approximation of the maximum value in V.

Lemma 3.1. *Given a set of values V with maximum value v_{max}, Count-Max(V) returns a value u_{max} where $u_{max} \geq v_{max}/(1+\mu)^2$ using $O(|V|^2)$ oracle queries.*

Using Example 3.2, when $\mu = 1$, we demonstrate that $(1+\mu)^2 = 4$ approximation ratio is achieved by Algorithm 1.

Example 3.2. *Let S denote a set of four records u, v, w and t with ground truth values 51, 101, 102 and 202, respectively. While identifying the maximum value under adversarial noise with $\mu = 1$, the oracle must return a correct answer to $O(u, t)$ and all other oracle query answers can be incorrect adversarially. If the oracle answers all other queries incorrectly, we have,* Count *values of t, w, u, v are 1, 1, 2, and 2 respectively. Therefore, u and v are equally likely, and when Algorithm 1 returns u, we have a $202/51 \approx 3.96$ approximation.*

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, we use a *tournament* to obtain the maximum value. The idea of using a *tournament* for finding maximum has been studied in the past [16, 19].

Algorithm 2 presents pseudo code of the approach that takes values V as input and outputs an approximate maximum value. It constructs a balanced λ-ary tree $\mathcal{T}$ containing n leaf nodes such that a random permutation of the values V is assigned to the leaves of $\mathcal{T}$. In a *tournament*, the internal nodes of $\mathcal{T}$ 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 $\arg\max_{v \in children(w)}$ Count$(v, children(w))$ at the internal node w, where 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 $\mathcal{T}$ as our output. In Lemma 3.3, we show that Algorithm 2 returns a value that is a $(1+\mu)^{2\log_\lambda n}$ multiplicative approximation of the maximum value.

Algorithm 1 Count-Max(S) : finds the max. value by counting

Input : A set of values S
Output : An approximate maximum value of S
for $v \in S$ **do**
    Calculate Count(v, S)
$u_{\max} \leftarrow \arg\max_{v \in S}$Count$(v, S)$
return $u_{\max}$

Lemma 3.3. *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 3.3, Algorithm 2 identifies a constant approximation when $\lambda = \Theta(n)$, μ is a fixed constant and has a query complexity of $\Theta(n^2)$. By reducing the degree of the tournament tree from λ to 2, we can achieve $\Theta(n)$ query complexity, but with a worse approximation ratio of $(1+\mu)^{\log n}$.

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

Observation 3.4. *At an internal node $w \in \mathcal{T}$, the identified maximum is incorrect only if there exists $x \in children(w)$ that is very close to the true maximum (say w_{max}), i.e. $\frac{w_{\max}}{(1+\mu)} \leq x \leq (1+\mu)w_{\max}$.*

Based on the above observation, our Algorithm 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 Max-Adv, we use a subset $\widetilde{V} \subseteq V$ of size $\sqrt{n}t$ (for a suitable choice of parameter t) obtained using uniform sampling with replacement. We show that using a sufficiently large subset $\widetilde{V}$, obtained by sampling, we ensure that at least one value that is closer to $v_{\max}$ is in $\widetilde{V}$, 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 disjoint parts (for a suitable choice of l) and run the 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 $v_{\max}$, 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 Max-Adv. We repeat this procedure t times and set $l = \sqrt{n}, t = 2\log(2/\delta)$ to achieve the desired success probability $1 - \delta$. We combine the outputs of both the steps, i.e., $\widetilde{V}$ and T and output the maximum among them using Count-Max. This ensures that we get a good approximation as we use the best of both the approaches.

Algorithm 2 Tournament : finds the maximum value using a tournament tree

Input : Set of values V, Degree λ
Output : An approximate maximum value $u_{\max}$
Construct a balanced λ-ary tree $\mathcal{T}$ with $|V|$ nodes as leaves.
Let π_V be a random permutation of V assigned to leaves of $\mathcal{T}$
for $i = 1$ to $\log_\lambda |V|$ **do**
    for internal node w at level $\log_\lambda |V| - i$ **do**
        Let U denote the children of w.
        Set the internal node w to Count-Max(U)
$u_{\max} \leftarrow$value at root of $\mathcal{T}$
return $u_{\max}$

Algorithm 3 Tournament-Partition

Input : Set of values V, number of partitions l
Output : A set of maximum values from each partition
Randomly partition V into l equal parts $V_1, V_2, \cdots V_l$
for $i = 1$ to l **do**
    $p_i \leftarrow$ Tournament$(V_i, 2)$
    $T \leftarrow T \cup \{p_i\}$
return T

Theoretical Guarantees. In order to prove approximation guarantee of Algorithm 4, we first argue that the sample $\widetilde{V}$ 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 $C = \{u : v_{\max}/(1+\mu) \leq u \leq v_{\max}\}$. In Lemma 3.5, we first show that $\widetilde{V}$ contains a value $v_j \in \widetilde{V}$ 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 3.5. *(1) If $|C| > \sqrt{n}/2$, then there exists a value $v_j \in \widetilde{V}$ satisfying $v_j \geq v_{max}/(1+\mu)$ with probability of $1 - \delta/2$.*

Algorithm 4 MAX-ADV : Maximum with Adversarial Noise

Input : Set of values V, number of iterations t, partitions l
Output : An approximate maximum value $u_{\max}$
$i \leftarrow 1, T \leftarrow \phi$
Let $\widetilde{V}$ denote a sample of size $\sqrt{n}t$ selected uniformly at random (with replacement) from V.
for $i \leq t$ **do**
    $T_i \leftarrow$ TOURNAMENT-PARTITION(V, l)
    $T \leftarrow T \cup T_i$
$u_{\max} \leftarrow$ COUNT-MAX$(\widetilde{V} \cup T)$
return $u_{\max}$

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

Now, we briefly provide a sketch of the proof of Lemma 3.5. Consider the first step, where we use a uniformly random sample $\widetilde{V}$ of $\sqrt{n}t$ points from V (obtained with replacement). When $|C| \geq \sqrt{n}/2$, probability that $\widetilde{V}$ contains a value from C is given by $1 - (1 - |C|/n)^{|\widetilde{V}|} = 1 - (1 - \frac{1}{2\sqrt{n}})^{2\sqrt{n}\log(2/\delta)} \approx 1 - \delta/2$.

In the second step, 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 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 $v_{\max}$ is $|C|/l = \sqrt{n}/(2\sqrt{n}) = 1/2$. Thus by the Markov's inequality, the probability that V_i contains a value from C is $\leq 1/2$. With $1/2$ probability, $v_{\max}$ 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 $v_{\max}$ is never compared with any value of C in at least one of the iterations with a probability at least $1 - (1 - 1/2)^{2\log(2/\delta)} \geq 1 - \delta/2$.

In Lemma 3.1, we show that using COUNT-MAX we get a $(1+\mu)^2$ multiplicative approximation. Combining it with Lemma 3.5, we have that $u_{\max}$ returned by Algorithm 4 satisfies $u_{\max} \geq \frac{v_{\max}}{(1+\mu)^3}$ with probability $1 - \delta$. For query complexity, Algorithm 3 identifies $\sqrt{n}t$ samples denoted by $\widetilde{V}$. These identified values, along with T are then processed by COUNT-MAX to identify the maximum u_{max}. This step requires $O(|\widetilde{V} \cup T|^2) = O(n\log^2(1/\delta))$ oracle queries.

THEOREM 3.6. *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 probabilistic noise. Specifically, the theoretical guarantees of Lemma 3.3 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, v_{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 $\texttt{Count}(u, S)$ 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. This process is continued $\Theta(\log n)$ times to identify the maximum value. We present the pseudo code in the full version [3] and prove the following approximation guarantee.

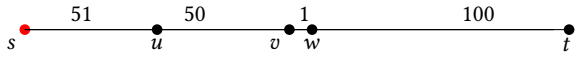


Figure 2: Example for Lemma 3.1 with $\mu = 1$.

THEOREM 3.7. *There is an algorithm that returns $u_{max} \in V$ such that $\text{rank}(u_{max}, V) = O(\log^2(n/\delta))$ with probability $1 - \delta$ and requires $O(n\log^2(n/\delta))$ oracle queries.*

The algorithm to identify the minimum value is same as that of maximum with a modification where Count scores consider the case of Yes (instead of No): $\texttt{Count}(v, S) = \sum_{x \in S \setminus \{v\}} \mathbf{1}\{O(v, x) == \texttt{Yes}\}$

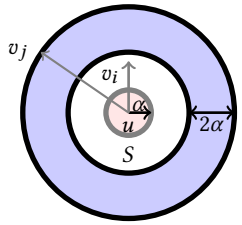
3.3 Extension to Farthest and Nearest Neighbor

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 $D(u) = \{d(u, u') \mid \forall u' \in V\}$ 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.

EXAMPLE 3.8. *Figure 2 shows a worst-case example for the approximation guarantee to identify the farthest point from s (with $\mu = 1$). Similar to Example 3.2, we have,* Count *values of t, w, u, v are $1, 1, 2, 2$ respectively. Therefore, u and v are equally likely, and when Algorithm 1 outputs u, we have a ≈ 3.96 approximation.*

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 3.7). In this section, we show that 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 discussed below. For the simplicity of exposition, we assume $p \leq 0.40$, though our algorithms work for any value of $p < 0.5$.

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 performing pairwise comparisons in a robust manner. Suppose the desired failure probability is δ, we observe that if $\Theta(\log(1/\delta))$ records closest to the query u are known (say S) and $\max_{x \in S}\{d(u, x)\} \leq \alpha$ for some $\alpha > 0$, then each pairwise comparison of the form $O(u, v_i, u, v_j)$ can be replaced by Algorithm 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 $\texttt{FCount}(v_i, v_j) = \sum_{x \in S} \mathbf{1}\{O(v_i, x, v_j, x) == \texttt{Yes}\}$ as a robust estimate where the oracle considers v_i to be closer to x than v_j. If $\texttt{FCount}(v_i, v_j)$ is smaller than $0.3|S| \leq (1-p)|S|/2$ then we output No and Yes otherwise. Therefore, every pairwise comparison query is replaced with $\Theta(\log(1/\delta))$ quadruplet queries using Algorithm 5.



In this example, $O(u, v_i, u, v_j)$ is answered correctly with a probability $1-p$. To boost the correctness probability, FCount uses the queries $O(x, v_i, x, v_j)$, $\forall x$ in the red region around u, denoted by S.

Figure 3: Algorithm 5 returns 'Yes' as $d(u, v_i) < d(u, v_j) - 2\alpha$.

We argue that Algorithm 5 will output the correct answer with a high probability if $|d(u, v_j) - d(u, v_i)| \geq 2\alpha$ (See Fig 3). In Lemma 3.9, we show that, if $d(u, v_j) > d(u, v_i) + 2\alpha$, then, $\texttt{FCount}(v_i, v_j) \geq 0.3|S|$ with probability $1 - \delta$.

Lemma 3.9. *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* $\texttt{FCount}(v_i, v_j) \geq 0.3|S|$ *with a probability of* $1 - \delta$

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 $|d(u, v_i) - d(u, v_j)| < 2\alpha$ and correct otherwise. In the full version [3], 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 PairwiseComp (Algorithm 5).We give an approximation guarantee, that loses an additive 6α following a similar analysis of Theorem 3.6.

Algorithm 5 PairwiseComp (u, v_i, v_j, S)

```
Calculate FCount(v_i, v_j) = Σ_{x∈S} 1{O(x, v_i, x, v_j) == Yes}
if FCount(v_i, v_j) < 0.3|S| then
    return No
else return Yes
```

Theorem 3.10. *Given a query vertex* u *and a set* S *with* $|S| = \Omega(\log(n/\delta))$ *such that* $\max_{v \in S} d(u, v) \leq \alpha$ *then the farthest identified using Algorithm 4 (with* PairwiseComp*), denoted by* u_{max} *is within* 6α *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* $O(n \log^3(n/\delta))$.

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 [28]. The greedy algorithm [28] is initialized 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. Then, it finds the farthest point among the clusters and uses it as the new center. This technique requires $O(nk)$ distance 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 its robustness, we devise new algorithms to perform assignment of points to respective clusters and farthest point identification. Missing Details from this section are discussed in the full version [3].

Algorithm 6 Greedy Algorithm

```
Input : Set of points V
Output : Clusters C
s_1 ← arbitrary point from V, S = {s_1}, C = {{V}}.
for i = 2 to k do
    s_i ← Approx-Farthest(S, C)
    S ← S ∪ {s_i}
    C ← Assign(S)
return C
```

4.1 Adversarial Noise

Now, we describe the two steps (Approx-Farthest and Assign) of the 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.

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. $\arg\max_{v_i \in V} d(v_i, s_j)$, which corresponds to the farthest point. For the parameters, we use $l = \sqrt{n}$, $t = \log(2k/\delta)$ and number of samples $\widetilde{V} = \sqrt{n}t$.

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

Example 4.1. *Suppose we run k-center algorithm with* $k = 2$ *and* $\mu = 1$ *on the points in Example 3.8. The optimal centers are u and t with radius* 51. *On running our algorithm, suppose w is chosen as the first center and* Approx-Farthest *calculates* Count *values similar to Example 3.2. We have,* Count *values of* s, t, u, v *are* $1, 2, 3, 0$ *respectively. Therefore, our algorithm identifies u as the second center, achieving* 3*-approximation.*

Theoretical Guarantees. We now prove the approximation guarantee obtained by Algorithm 6. In each iteration, we show that 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 use MCount scores for assignment. Similarly, we show that Approx-Farthest (Algorithm 4) identifies a close approximation to the true farthest point. Concretely, we show that every point is assigned to a center which is a $(1+\mu)^2$ approximation; Algorithm 4 identifies farthest point w which is a $(1+\mu)^5$ approximation.

In every iteration of the Greedy algorithm, if we identify an α-approximation of the farthest point, and a β-approximation when reassigning the points, then, we show that the clusters output are a

$2\alpha\beta^2$-approximation to the k-center objective. For complete details, please refer the full version [3]. Combining all the claims, for a given error parameter μ, we obtain:

Theorem 4.2. *For $\mu < \frac{1}{18}$, Algorithm 6 achieves a $(2 + O(\mu))$-approximation for the k-center objective using $O(nk^2 + nk \cdot \log^2(k/\delta))$ oracle queries with probability $1 - \delta$.*

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 Approx-Farthest and 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 δ. We assume $p \leq 0.40$, a constant strictly less than $\frac{1}{2}$. Let $\gamma = 450$ be a large constant used in our algorithms which obtains the claimed guarantees.

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 $\gamma \log(n/\delta)/m$ to identify a small sample of $\approx \frac{\gamma n \log(n/\delta)}{m}$ points (denoted by $\widetilde{V}$) 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. The cores are then used in the second phase (line 15) for the assignment of remaining points. Now, we describe

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 $\widetilde{V}$ with probability $\frac{\gamma \log(n/\delta)}{m}$
4: $s_1 \leftarrow$ select an arbitrary point from $\widetilde{V}$, $S \leftarrow \{s_1\}$
5: $C(s_1) \leftarrow \widetilde{V}$
6: $R(s_1) \leftarrow$ Identify-Core$(C(s_1), s_1)$
7: **for** $i = 2$ to k **do**
8: $\quad s_i \leftarrow$ Approx-Farthest(S, C)
9: $\quad C, R \leftarrow$ Assign(S, s_i, R)
10: $\quad S \leftarrow S \cup \{s_i\}$
11: $C \leftarrow$ Assign-Final$(S, R, V \setminus \widetilde{V})$
12: **return** C

the main challenge in extending Approx-Farthest and Assign ideas of Algorithm 6. Given a cluster C containing the center s_i, when we find the Approx-Farthest, the ideas from Section 3.2 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. We argue that such an assumption of set R is justified. For example, consider the case when clusters are of size $\Theta(n)$ and sampling $k \log(n/\delta)$ points gives us $\log(n/\delta)$ points from each optimum cluster; which means that there are $\log(n/\delta)$ points within a distance of 2 OPT from every sampled point where OPT refers to the optimum k-center objective.

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 $C(s_j) \setminus R(s_j)$ centered at s_j. ACount captures the total number of times u is considered to belong to same cluster as that of x for each x in the core $R(s_j)$. Intuitively, points that belong to 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: $\quad$ **for** $u \in C(s_j) \setminus R(s_j)$ **do**
4: $\quad\quad$ ACount$(u, s_i, s_j) = \sum_{v_k \in R(s_j)} \mathbf{1}\{O(u, s_i, u, v_k) ==$ Yes$\}$
5: $\quad\quad$ **if** ACount$(u, s_i, s_j) > 0.3|R(s_j)|$ **then**
6: $\quad\quad\quad C(s_i) \leftarrow C(s_i) \cup \{u\}; C(s_j) \leftarrow C(s_j) \setminus \{u\}$
7: $R(s_i) \leftarrow$ Identify-Core$(C(s_i), s_i)$
8: **return** C, R

Algorithm 9 Identify-Core$(C(s_i), s_i)$

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

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.

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. To do so, we use the ideas developed in section 3.3.

We leverage ClusterComp (Algorithm 10) to compare the distance of two points, say v_i, v_j from their respective centers s_i, s_j. ClusterComp gives a robust answer to a pairwise comparison query to the oracle $O(v_i, s_i, v_j, s_j)$ using the cores $R(s_i)$ and $R(s_j)$. ClusterComp can be used as a pairwise comparison subroutine in place of PairwiseComp for the algorithm in Section 3 to calculate the farthest point. For every $s_i \in S$, let $\widetilde{R}(s_i)$ 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_i) 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 $C(s_i)$, 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 $l = \sqrt{n}$, $t = \log(n/\delta)$.

Example 4.3. *Suppose we run k-center Algorithm 7 with $k = 2$ and $m = 2$ on the points in Example 3.8. Let w denote the first center chosen and Algorithm 7 identifies the core $R(w)$ by calculating* Count *values. If $O(u, w, s, w)$ and $O(s, w, t, w)$ are answered incorrectly (with probability p), we obtain* Count *values of v, s, u, t as $3, 2, 1, 0$ respectively; and v is added to $R(w)$. We identify the second center u by calculating* FCount *for s, u and t (See Figure 3). After assigning (using Assign), the clusters identified are $\{w, v\}, \{u, s, t\}$, achieving 3-approximation.*

Algorithm 10 ClusterComp (v_i, s_i, v_j, s_j)

comparisons $\leftarrow 0$, $\texttt{FCount}(v_i, v_j) \leftarrow 0$
if $s_i = s_j$ **then**
  Let $\texttt{FCount}(v_i, v_j) = \sum_{x \in R(s_i)} \mathbf{1}\{O(v_i, x, v_j, x) == \texttt{Yes}\}$
  comparisons $\leftarrow |R(s_i)|$
else Let $\texttt{FCount}(v_i, v_j) = \sum_{x \in \widetilde{R}(s_i), y \in \widetilde{R}(s_j)} \mathbf{1}\{O(v_i, x, v_j, y) == \texttt{Yes}\}$
  comparisons $\leftarrow |\widetilde{R}(s_i)| \cdot |\widetilde{R}(s_j)|$
if $\texttt{FCount}(v_i, v_j) < 0.3 \cdot$ comparisons **then**
  return No
else return Yes

Assign-Final. After obtaining k clusters on the set of sampled points $\widetilde{V}$, 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 $s_1 \in S$, and if $\texttt{ACount}(u, s_2, s_1) \geq 0.3|R(s_1)|$, 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 $\widetilde{V} \subseteq V$ and runs the 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 of $\gamma \log(n/\delta)/m$ ensures that we have at least $\Theta(\log(n/\delta))$ points from each of the optimal clusters in our sampled set $\widetilde{V}$. 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-Farthest and Assign subroutines. We give the following theorem, which guarantees a constant, i.e., $O(1)$ approximation with high probability.

Theorem 4.4. *Given $p \leq 0.4$, a failure probability δ, and $m = \Omega(\log^3(n/\delta)/\delta)$. Then, Algorithm 7 achieves a $O(1)$-approximation for the k-center objective using $O(nk\log(n/\delta) + \frac{n^2}{m^2}k\log^2(n/\delta))$ oracle queries with probability $1 - O(\delta)$.*

5 HIERARCHICAL CLUSTERING

In this section, we present robust algorithms for agglomerative hierarchical clustering using single linkage and complete linkage objectives. The naive algorithms initialize every record as a singleton cluster and merge the closest pair of clusters iteratively. For a set of clusters $\mathcal{C} = \{C_1, \ldots, C_t\}$, 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, $d_{SL}(C_1, C_2) = \min_{v_1 \in C_1, v_2 \in C_2} d(v_1, v_2)$. 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. The main challenge in implementing 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 $t = 2\log(n/\delta)$) to calculate the minimum over the set containing pairwise distances. For this algorithm, Lemma 5.1 shows that the pair 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 3.6.

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

Algorithm 11 Greedy Algorithm

Input : Set of points V
Output : Hierarchy H
$H \leftarrow \{\{v\} \mid v \in V\}$, $\mathcal{C} \leftarrow \{\{v\} \mid v \in V\}$
for $C_i \in \mathcal{C}$ **do**
  $\widetilde{C_i} \leftarrow$NearestNeighbor of C_i among $\mathcal{C} \setminus \{C_i\}$ using Sec 3.3
while $|\mathcal{C}| > 1$ **do**
  Let $(C_j, \widetilde{C_j})$ be the closest pair among $(C_i, \widetilde{C_i})$, $\forall C_i \in \mathcal{C}$
  $C' \leftarrow C_j \cup \widetilde{C_j}$
  Update Adjacency list of C' with respect to $\mathcal{C}$
  Add C' as parent of $\widetilde{C_j}$ and C_j in H.
  $\mathcal{C} \leftarrow \left(\mathcal{C} \setminus \{C_j, \widetilde{C_j}\}\right) \cup \{C'\}$
  $\widetilde{C'} \leftarrow$NearestNeighbor of C' from its adjacency list
return H

Overview. Agglomerative clustering techniques are inefficient and involve merge operations comparing at most $\binom{n}{2}$ pairs of distance values in each of the n iterations, yielding an overall query complexity of $O(n^3)$. Improving upon this, SLINK algorithm [48] was proposed to construct the hierarchy in $O(n^2)$ comparisons. To implement this algorithm with a comparison oracle, for every cluster $C_i \in \mathcal{C}$, we maintain an adjacency list containing every cluster C_j in $\mathcal{C}$ 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 (v_i, v_j) such that $d(v_i, v_j) = \min_{v_i \in C_i, v_j \in C_j} d(v_i, v_j)$. 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. Then, we identify the closest cluster for every $C_i \in \mathcal{C}$, and denote it by $\widetilde{C_i}$. This step takes n nearest neighbor queries, each requiring $O(n\log^2(n/\delta))$ oracle queries. In every subsequent iteration, we identify the closest pair of clusters (Using section 3.3), say C_j and $\widetilde{C_j}$ from $\mathcal{C}$.

After merging these clusters, in order to update the adjacency list, we need the pair of records with minimum distance between the merged cluster $C' \equiv C_j \cup \widetilde{C_j}$ and every other cluster $C_k \in \mathcal{C}$. In the previous iteration of the algorithm, we already have the minimum distance record pair for (C_j, C_k) and $(\widetilde{C_j}, C_k)$. Therefore a single query between these two pairs of records is sufficient to identify the minimum distance edge between C' and C_k (formally: $d_{SL}(C_j \cup \widetilde{C_j}, C_k) = \min\{d_{SL}(C_j, C_k), d_{SL}(\widetilde{C_j}, C_k)\}$). 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 $O(n\log^2(n/\delta))$ queries. As our Algorithm terminates in at most n iterations, it has an overall query complexity of $O(n^2\log^2(n/\delta))$. In Theorem 5.2, we given an approximation guarantee for every merge operation of Algorithm 11.

Theorem 5.2. *In any iteration, suppose the distance between a cluster $C_j \in \mathcal{C}$ and its identified nearest neighbor $\widetilde{C_j}$ is α-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.

Probabilistic Noise model. The above discussed algorithms do not extend to the probabilistic noise due to constant probability of error for each query. However, when we are given a priori, a partitioning of V into clusters of size $> \log n$ such that the maximum distance between any pair of records in every cluster is smaller than α (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. Note that agglomerative clustering algorithms are known to require $\Omega(n^2)$ queries, which can be infeasible for million scale datasets. However, blocking based techniques present efficient heuristics to prune out low similarity pairs [45]. Devising provable algorithms with better time complexity is outside the scope of this work.

6 EXPERIMENTS

In this section, we evaluate the effectiveness of our techniques on various real world datasets and answer the following questions. **Q1:** Is quadruplet oracle practically feasible? How do the different types of queries compare in terms of quality and time taken by annotators? **Q2:** Are proposed techniques robust to different levels of noise in oracle answers? **Q3:** How does the query complexity and solution quality of proposed techniques compare with optimum for varied levels of noise?

6.1 Experimental Setup

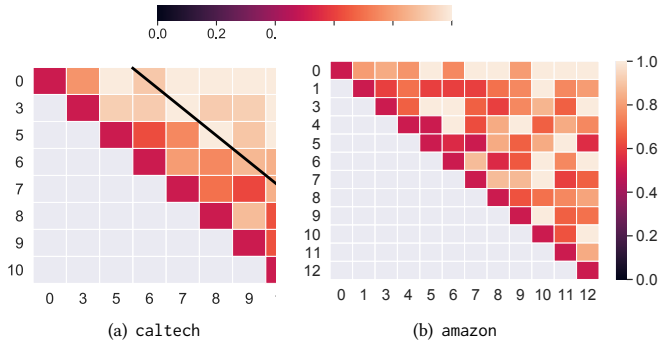
Datasets. We consider the following real-world datasets.
(1) `cities` dataset [2] comprises of 36K cities of the United States. The different features of the cities include state, county, zip code, population, time zone, latitude and longitude.
(2) `caltech` dataset comprises 11.4K images from 20 categories. The ground truth distance between records is calculated using the hierarchical categorization described in Griffin et al. [30].
(3) `amazon` dataset contains 7K images and textual descriptions collected from `amazon.com` [32]. For obtaining the ground truth distances we use Amazon's hierarchical catalog.
(4) `monuments` dataset comprises of 100 images belonging to 10 tourist locations around the world.
(5) `dblp` contains 1.8M titles of computer science papers from different areas [61]. From these titles, noun phrases were extracted and a dictionary of all the phrases was constructed. Euclidean distance in word2vec embedding space is considered as the ground truth distance between concepts.
Baselines. We compare our techniques with the optimal solution (whenever possible) and the following baselines. (a) `Tour2` constructs a binary tournament tree over the entire dataset to compare the values and the root node corresponds to the identified maximum/minimum value (Algorithm 2 with $\lambda = 2$). This approach is an adaptation of the maximum calculation algorithm in [16] 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 6 and closest pair of clusters in hierarchical clustering. (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 the identified centers by querying each record with every pair of center.

Calculating optimal clustering objective for k-center is NP-hard even in the presence of accurate pairwise distance [60]. So, we compare the solution quality with respect to the greedy algorithm on the ground truth distances, denoted by `TDist`. 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. Unless specified, we set $t = 1$ in Algorithm 4 and $\gamma = 2$ 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., maximum radius of the returned clusters as the evaluation metric and compare against the true greedy algorithm (`TDist`) and other baselines. For datasets where ground truth clusters are known (`amazon`, `caltech` and `monuments`), we use F-score over intra-cluster pairs for comparing it with the baselines [21]. 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.
Noise Estimation. For `cities`, `amazon`, `caltech`, and `monuments` datasets, we ran a user study on Amazon Mechanical Turk to estimate the noise in oracle answers over a small sample of the dataset, often referred to as the validation set. Using crowd responses, we trained a classifier (random forest [52] obtained the best results) using active learning to act as the quadruplet oracle, and reduce the number of queries to the crowd. Our active learning algorithm [51] uses a batch of 20 queries and we stop it when the classifier accuracy on the validation set does not improve by more than 0.01 [27]. To efficiently construct a small set of candidates for active learning and pruning low similarity pairs for `dblp`, we employ token based blocking [45] for the datasets. For the synthetic oracle, we simulate quadruplet oracle with different values of the noise parameters.

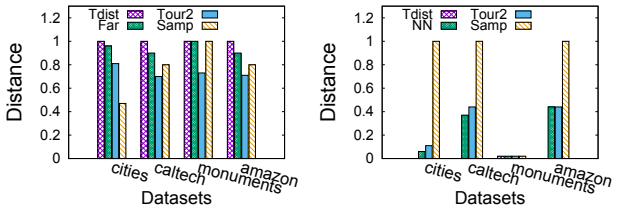
6.2 User study

In this section, we evaluate the users ability to answer quadruplet queries and compare it with other types of queries.
Setup. We ran a user study on Amazon Mechanical Turk platform for four datasets `cities`, `amazon`, `caltech` and `monuments`. We consider the ground truth distance between record pairs and discretize them into buckets, and assign a pair of records to a bucket if the distance falls within its range. For every pair of buckets, we query a random subset of $\log n$ quadruplet oracle queries (where n is size of



(a) caltech (b) amazon

Figure 4: Accuracy values (denoted by the color of a cell) for different distance ranges observed during our user study. The diagonal entries refer to the quadruplets with similar distance between the corresponding pairs and the distance increases as we go further away from the diagonal.



(a) Farthest,higher is better (b) Nearest Neighbor (NN), lower is better

Figure 5: Comparison of farthest and NN techniques for crowdsourced oracle queries.

dataset). Each query is answered by three different crowd workers and a majority vote is taken as the answer to the query.

6.2.1 Qualitative Analysis of Oracle. In Figure 4, for every pair of buckets, using a heat map, we plot the accuracy of answers obtained from the crowd workers for quadruplet queries. For all datasets, average accuracy of quadruplet queries is more than 0.83 and the accuracy is minimum whenever both pairs of records belong to the same bucket (as low as 0.5). However, we observe varied behavior across datasets as the distance between considered pairs increases.

For the caltech dataset, we observe that when the ratio of the distances is more than 1.45 (indicated by a black line in the Figure 4(a)) , there is no noise (or close to zero noise) observed in the query responses. As we observe a sharp decline in noise as the distance between the pairs increases, it suggests that adversarial noise is satisfied for this dataset. We observe a similar pattern for the cities and monuments datasets. For the amazon dataset, we observe that there is substantial noise across all distance ranges (See Figure 4(b)) rather than a sharp decline, suggesting that the probabilistic model is satisfied.

6.2.2 Comparison with pairwise querying mechanisms. To evaluate the benefit of quadruplet queries, we compare the quality of quadruplet comparison oracle answers with the following pairwise oracle query models. (a) Optimal cluster query: This query asks questions of type 'do u and v refer to same/similar type?'. (b) Distance query: How similar are the records x and y? In this query, the annotator scores the similarity of the pair within 1 to 10.

We make the following observations. (i) Optimal cluster queries are answered correctly only if the ground truth clusters refer to different entities (each cluster referring to a distinct entity). Crowd workers tend to answer 'No' if the pair of records refer to different entities. Therefore, we observe high precision (more than 0.90) but low recall (0.50 on amazon and 0.30 on caltech for $k = 10$) of the returned labels. (ii) We observed very high variance in the distance estimation query responses. For all record pairs with identical entities, the users returned distance estimates that were within 20% of the correct distances. In all other cases, we observe the estimates to have errors of upto 50%. We provide more detailed comparison

Table 1: F-score comparison of k-center clustering. Oq is marked with $*$ as it was computed on a sample of 150 pairwise queries to the crowd[3]. All other techniques were run on the complete dataset using a classifier.

Technique	kC	Tour2	Samp	Oq*
caltech ($k = 10$)	**1**	0.88	0.91	0.45
caltech ($k = 15$)	**1**	0.89	0.88	0.49
caltech ($k = 20$)	**0.99**	0.93	0.87	0.58
monuments ($k = 5$)	**1**	0.95	0.97	0.77
amazon ($k = 7$)	**0.96**	0.74	0.57	0.48
amazon ($k = 14$)	**0.92**	0.66	0.54	0.72

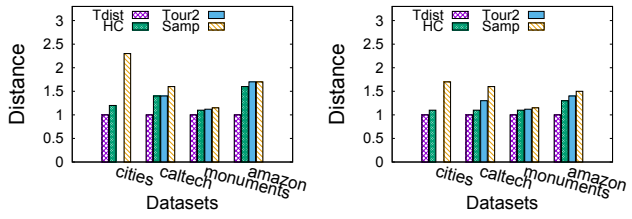
on the quality of clusters identified by pairwise query responses along with quadruplet queries in the next section.

6.3 Crowd Oracle: Solution Quality & Query Complexity

In this section, we compare the quality of our proposed techniques for the datasets on which we performed the user study. Following the findings of Section 6.2, we use probabilistic model based algorithm for amazon (with $p = 0.50$) and adversarial noise model based algorithm for caltech, monuments and cities.

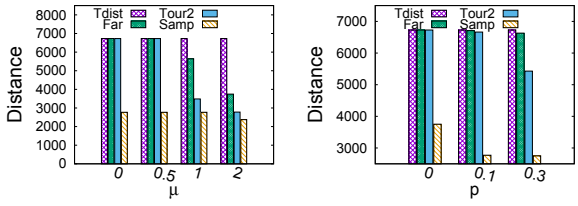
Finding Max and Farthest/Nearest Neighbor. Figure 5 compares the quality of farthest and nearest neighbor (NN) identified by proposed techniques along with other baselines. The values are normalized according to the maximum value to present all datasets on the same scale. Across all datasets, the point identified by Far and NN is closest to the optimal value, TDist. In contrast, the farthest returned by Tour2 is better than that of Samp for cities dataset but not for caltech, monuments and amazon. We found that this difference in quality across datasets is due to varied distance distribution between pairs. The cities dataset has a skewed distribution of distance between record pairs, leading to a unique optimal solution to the farthest/NN problem. Due to this reason, the set of records sampled by Samp does not contain any record that is a good approximation of the optimal farthest. However, ground truth distances between record pairs in amazon, monuments and caltech are less skewed with more than $\log n$ records satisfying the optimal farthest point for all queries. Therefore, Samp performs better than Tour2 on these datasets. We observe Samp performs worse for NN because our sample does not always contain the closest point.

***k*-center Clustering.** We evaluate the F-score of the clusters generated by our techniques along with baselines and techniques for pairwise optimal query mechanism (denoted as Oq) We report the



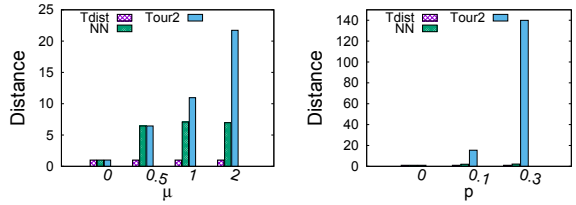
(a) Single Linkage (b) Complete Linkage

Figure 6: Comparison of Hierarchical clustering techniques with crowdsourced oracle.



(a) cities–Adversarial (b) cities–Probabilistic

Figure 7: Comparison of farthest identification techniques for adversarial and probabilistic noise models.



(a) cities–Adversarial (b) cities–Probabilistic

Figure 8: Comparison of nearest neighbor techniques for adversarial and probabilistic noise model (lower is better).

results on the sample of queries asked to the crowd as opposed to training a classifier because the classifier generates noisier results and has poorer F-score than the quality of labels generated by crowdsourcing. Optimal clusters are identified from the original source of the datasets (amazon and caltech) and manually for monuments. Table 1 presents the summary of our results for different values of k. Across all datasets, our technique achieves more than 0.90 F-score. On the other hand, Tour2 and Samp do not identify the ground truth clusters correctly, leading to low F-score. Similarly, Oq achieves poor recall (and hence low F-score) as it labels many record pairs to belong to separate clusters. For example, a *frog* and a *butterfly* belong to the same optimal cluster for caltech (k=10) but the two records are assigned to different clusters by Oq.

Hierarchical Clustering. Figure 6 compares the average distance of the merged clusters across different iterations of the agglomerative clustering algorithm. Tour2 has $O(n^3)$ complexity and does not run for cities dataset in less than 48 hrs. The objective value of different techniques are normalized by the optimal value with Tdist denoting 1. For all datasets, HC performs better than Samp and Tour2. Among datasets, the quality of hierarchies generated for monuments is similar for all techniques due to low noise.

Query Complexity. To ensure scalability, we trained active learning based classifier for all the aforementioned experiments. In total, amazon, cities, and caltech required 540 (cost: $32.40), 220 (cost: $13.20) and 280 (cost: $16.80) queries to the crowd respectively.

6.4 Simulated Oracle: Solution Quality

In this section, we compare the robustness of the techniques where the query response is simulated synthetically for given μ and p.

Finding Max and Farthest/Nearest Neighbor. In Figure 7(a), $\mu = 0$ denotes the setting where the oracle answers all queries correctly. In this case, Far and Tour2 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 (μ), we observe that the farthest is always at a distance within 4 times the optimal distance (See Fig 7(a)). We observe that the quality of farthest identified by Tour2 is close to that of Far for smaller μ 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 Figure 7(b), we compare the true distance of the identified farthest points for the case of probabilistic noise with error probability p. We observe that $\texttt{Far}_p$ 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 than the value returned by $\texttt{Far}_p$ for an error probability of 0.3. Tour2 has a similar performance as that of $\texttt{Far}_p$ for $p \leq 0.1$, but we observe a decline in solution quality for higher noise (p) values.

Figures 8(a), 8(b) compare the true distance of the identified nearest neighbor with different baselines. NN shows superior performance as compared to Tour2 across all error values. The solution quality of NN does not worsen with increase in error. We omit Samp from the plots because the returned points had very poor performance (as bad as 700 even in the absence of error). We observed similar behavior for other datasets. We present a detailed analysis with the simulated oracle in the full version [3].

7 CONCLUSION

In this paper, we show how algorithms for various basic tasks such as finding maximum, nearest neighbor, k-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.

ACKNOWLEDGEMENTS

We thank Anirudh Sabnis for many insightful discussions and help conducting the user study. This work is supported partly by NSF grants 1652303, 1909046, 1908849, 1637536, HDR TRIPODS 1934846, and an Alfred P. Sloan Fellowship.

REFERENCES

[1] [n.d.]. Google Vision API https://cloud.google.com/vision.

[2] https://simplemaps.com/data/us-cities. United States Cities Database. (https://simplemaps.com/data/us-cities).

[3] Raghavendra Addanki, Sainyam Galhotra, and Barna Saha. 2021. How to Design Robust Algorithms using Noisy Comparison Oracles. *arXiv preprint arXiv:2105.05782* (2021).

[4] Nir Ailon, Anup Bhattacharya, Ragesh Jaiswal, and Amit Kumar. 2018. Approximate Clustering with Same-Cluster Queries. In *9th Innovations in Theoretical Computer Science Conference (ITCS 2018)*, Vol. 94. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 40.

[5] Miklós Ajtai, Vitaly Feldman, Avinatan Hassidim, and Jelani Nelson. 2009. Sorting and selection with imprecise comparisons. In *International Colloquium on Automata, Languages, and Programming*. Springer, 37–48.

[6] Akhil Arora, Sakshi Sinha, Piyush Kumar, and Arnab Bhattacharya. 2018. HD-index: pushing the scalability-accuracy boundary for approximate kNN search in high-dimensional spaces. *PVLDB* 11, 8 (2018), 906–919.

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

[8] Shai Ben-David. 2018. Clustering-what both theoreticians and practitioners are doing wrong. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

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

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

[11] Marco Bressan, Nicolò Cesa-Bianchi, Andrea Paudice, and Fabio Vitale. 2019. Correlation Clustering with Adaptive Similarity Queries. In *Advances in Neural Information Processing Systems*. 12510–12519.

[12] Vaggos Chatziafratis, Rad Niazadeh, and Moses Charikar. 2018. Hierarchical clustering with structural constraints. *arXiv preprint arXiv:1805.09476* (2018).

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

[14] Tuhinangshu Choudhury, Dhruti Shah, and Nikhil Karamchandani. 2019. Top-m Clustering with a Noisy Oracle. In *2019 National Conference on Communications (NCC)*. IEEE, 1–6.

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

[16] Susan Davidson, Sanjeev Khanna, Tova Milo, and Sudeepa Roy. 2015. Top-k and Clustering with Noisy Comparisons. *ACM Trans. Database Syst.* 39, 4, Article 35 (Dec. 2015), 39 pages. https://doi.org/10.1145/2684066

[17] Eyal Dushkin and Tova Milo. 2018. Top-k Sorting Under Partial Order Information. In *Proceedings of the 2018 International Conference on Management of Data*. 1007–1019.

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

[19] Uriel Feige, Prabhakar Raghavan, David Peleg, and Eli Upfal. 1994. Computing with noisy information. *SIAM J. Comput.* 23, 5 (1994), 1001–1018.

[20] Donatella Firmani, Barna Saha, and Divesh Srivastava. 2016. Online Entity Resolution Using an Oracle. *PVLDB* 9, 5 (2016), 384–395. https://doi.org/10.14778/2876473.2876474

[21] Sainyam Galhotra, Donatella Firmani, Barna Saha, and Divesh Srivastava. 2018. Robust entity resolution using random graphs. In *Proceedings of the 2018 International Conference on Management of Data*. 3–18.

[22] Barbara Geissmann, Stefano Leucci, Chih-Hung Liu, and Paolo Penna. 2017. Sorting with Recurrent Comparison Errors. In *28th International Symposium on Algorithms and Computation (ISAAC 2017)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.

[23] Barbara Geissmann, Stefano Leucci, Chih-Hung Liu, and Paolo Penna. 2019. Optimal Sorting with Persistent Comparison Errors. In *27th Annual European Symposium on Algorithms (ESA 2019)*, Vol. 144. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 49.

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

[25] Debarghya Ghoshdastidar, Michaël Perrot, and Ulrike von Luxburg. 2019. Foundations of Comparison-Based Hierarchical Clustering. In *Advances in Neural Information Processing Systems*. 7454–7464.

[26] Yogesh Girdhar and Gregory Dudek. 2012. Efficient on-line data summarization using extremum summaries. In *2012 IEEE International Conference on Robotics and Automation*. IEEE, 3490–3496.

[27] Chaitanya Gokhale, Sanjib Das, AnHai Doan, Jeffrey F Naughton, Narasimhan Rampalli, Jude Shavlik, and Xiaojin Zhu. 2014. Corleone: Hands-off crowdsourcing for entity matching. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*. 601–612.

[28] Teofilo F Gonzalez. 1985. Clustering to minimize the maximum intercluster distance. *Theoretical Computer Science* 38 (1985), 293–306.

[29] Kasper Green Larsen, Michael Mitzenmacher, and Charalampos Tsourakakis. 2020. Clustering with a Faulty Oracle. In *Proceedings of The Web Conference 2020 (WWW '20)*. Association for Computing Machinery, New York, NY, USA, 2831–2834. https://doi.org/10.1145/3366423.3380045

[30] Gregory Griffin, Alex Holub, and Pietro Perona. 2007. Caltech-256 object category dataset. (2007).

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

[32] Ruining He and Julian McAuley. 2016. Ups and downs: Modeling the visual evolution of fashion trends with one-class collaborative filtering. In *proceedings of the 25th international conference on world wide web*. 507–517.

[33] Max Hopkins, Daniel Kane, Shachar Lovett, and Gaurav Mahajan. 2020. Noise-tolerant, reliable active classification with comparison queries. *arXiv preprint arXiv:2001.05497* (2020).

[34] Wasim Huleihel, Arya Mazumdar, Muriel Médard, and Soumyabrata Pal. 2019. Same-Cluster Querying for Overlapping Clusters. In *Advances in Neural Information Processing Systems*. 10485–10495.

[35] Christina Ilvento. 2019. Metric learning for individual fairness. *arXiv preprint arXiv:1906.00250* (2019).

[36] Ehsan Kazemi, Lin Chen, Sanjoy Dasgupta, and Amin Karbasi. 2018. Comparison based learning from weak oracles. *arXiv preprint arXiv:1802.06942* (2018).

[37] Taewan Kim and Joydeep Ghosh. 2017. Relaxed oracles for semi-supervised clustering. *arXiv preprint arXiv:1711.07433* (2017).

[38] Taewan Kim and Joydeep Ghosh. 2017. Semi-supervised active clustering with weak oracles. *arXiv preprint arXiv:1709.03202* (2017).

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

[40] Matthäus Kleindessner, Pranjal Awasthi, and Jamie Morgenstern. 2019. Fair k-Center Clustering for Data Summarization. In *International Conference on Machine Learning*. 3448–3457.

[41] Ngai Meng Kou, Yan Li, Hao Wang, Leong Hou U, and Zhiguo Gong. 2017. Crowdsourced Top-k Queries by Confidence-Aware Pairwise Judgments. In *Proceedings of the 2017 ACM International Conference on Management of Data*. 1415–1430.

[42] Blake Mason, Ardhendu Tripathy, and Robert Nowak. 2019. Learning Nearest Neighbor Graphs from Noisy Distance Samples. In *Advances in Neural Information Processing Systems*. 9586–9596.

[43] Arya Mazumdar and Barna Saha. 2017. Clustering with noisy queries. In *Advances in Neural Information Processing Systems*. 5788–5799.

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

[45] George Papadakis, Jonathan Svirsky, Avigdor Gal, and Themis Palpanas. 2016. Comparative analysis of approximate blocking techniques for entity resolution. *PVLDB* 9, 9 (2016), 684–695.

[46] Vassilis Polychronopoulos, Luca De Alfaro, James Davis, Hector Garcia-Molina, and Neoklis Polyzotis. 2013. Human-Powered Top-k Lists.. In *WebDB*. 25–30.

[47] Dražen Prelec, H Sebastian Seung, and John McCoy. 2017. A solution to the single-question crowd wisdom problem. *Nature* 541, 7638 (2017), 532–535.

[48] Robin Sibson. 1973. SLINK: an optimally efficient algorithm for the single-link cluster method. *The computer journal* 16, 1 (1973), 30–34.

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

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

[51] https://modal-python.readthedocs.io/en/latest/. [n.d.]. modAL library.

[52] https://scikit-learn.org/stable/. [n.d.]. Scikit-learn.

[53] Vijay V Vazirani. 2013. *Approximation algorithms*. Springer Science & Business Media.

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

[55] Victor Verdugo. [n.d.]. Skyline Computation with Noisy Comparisons. In *Combinatorial Algorithms: 31st International Workshop, IWOCA 2020, Bordeaux, France, June 8–10, 2020, Proceedings*. Springer, 289.

[56] Vasilis Verroios and Hector Garcia-Molina. 2015. Entity resolution with crowd errors. In *2015 IEEE 31st International Conference on Data Engineering*. IEEE, 219–230.

[57] Norases Vesdapunt, Kedar Bellare, and Nilesh Dalvi. 2014. Crowdsourcing algorithms for entity resolution. *PVLDB* 7, 12 (2014), 1071–1082.

[58] Ramya Korlakai Vinayak and Babak Hassibi. 2016. Crowdsourced clustering: Querying edges vs triangles. In *Advances in Neural Information Processing Systems*. 1316–1324.

[59] Jiannan Wang, Tim Kraska, Michael J Franklin, and Jianhua Feng. 2012. CrowdER: Crowdsourcing Entity Resolution. *PVLDB* 5, 11 (2012).
[60] David P Williamson and David B Shmoys. 2011. *The design of approximation algorithms*. Cambridge university press.
[61] Chao Zhang, Fangbo Tao, Xiusi Chen, Jiaming Shen, Meng Jiang, Brian Sadler, Michelle Vanni, and Jiawei Han. 2018. Taxogen: Unsupervised topic taxonomy construction by adaptive term embedding and clustering. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2701–2709.