Learning to Cluster via Same-Cluster Queries

Yi Li
Nanyang Technological University
Singapore
yili@ntu.edu.sg

Yan Song*
Indiana University Bloomington
Bloomington, IN, USA
songyan@iu.edu

Qin Zhang†
Indiana University Bloomington
Bloomington, IN, USA
qzhangcs@indiana.edu

ABSTRACT

We study the problem of learning to cluster data points using an oracle which can answer same-cluster queries. Different from previous approaches, we do not assume that the total number of clusters is known at the beginning and do not require that the true clusters are consistent with a predefined objective function such as the K-means. These relaxations are critical from the practical perspective and, meanwhile, make the problem more challenging. We propose two algorithms with provable theoretical guarantees and verify their effectiveness via an extensive set of experiments on both synthetic and real-world data.

CCS CONCEPTS

• **Theory of computation → Semi-supervised learning**; **Facility location and clustering**.

KEYWORDS

clustering; weak supervision; same-cluster oracle

ACM Reference Format:
Yi Li, Yan Song, and Qin Zhang. 2021. Learning to Cluster via Same-Cluster Queries. In *Proceedings of the 30th ACM International Conference on Information and Knowledge Management (CIKM '21), November 1–5, 2021, Virtual Event, QLD, Australia.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3459637.3482365

1 INTRODUCTION

Clustering is a fundamental problem in data analytics and sees numerous applications across many areas in computer science. However, many clustering problems are computationally hard, even for approximate solutions. To alleviate the computational burden, Ashtiani et al. [4] introduced *weak supervision* into the clustering process. More precisely, they allow the algorithm to query an oracle which can answer *same-cluster* queries, that is, whether two elements (points in the Euclidean space) belong to the same cluster. The oracle can, for example, be a domain expert in the setting of crowdsourcing. It was shown in [4] that the K-means clustering can be solved computationally efficiently with the help of a small number of same-cluster queries using the oracle.

The initial work by Ashtiani et al. [4] relies on a strong "data niceness property" called the *γ-margin*, which requires that for each cluster C with center μ, the distance between any point $p \notin C$ and μ needs to be larger than that between any point $p \in C$ and μ by at least a constant multiplicative factor $\gamma > 1$ sufficiently large. This assumption was later removed by Ailon et al. [2], who gave an algorithm which, given a parameter K, outputs a set C of K centers attaining a $(1+\epsilon)$-approximation of the K-means objective function. There has been follow-up work by Chien et al. [6] in the same setting, replacing the γ-margin assumption with a new assumption on the size of the clusters. However, there are still two critical issues in this line of approach:

(1) In [2, 6], it is assumed that the total number of clusters K in the *true clustering* is known to the algorithm, which is unrealistic in many cases.
(2) In [2, 6], it is assumed that the optimal solution to the K-means clustering is *consistent* with the true clustering (i.e., when we have ground truth labels for all points). This assumption is highly problematic, since the ground-truth clustering can be arbitrary and very different from an optimal solution with respect to a fixed objective function such as the K-means.

In this paper, we aim to remove both assumptions. We shall design algorithms that find the approximate centers of the true clusters using same-cluster queries, in the setting that we do *not* know in advance the number of clusters in the true clustering and the true clustering has *no* relationship with the optimal solution of a certain objective function. We obtain our result at a small cost: our sample-based algorithm may not be able to find the centers of the small clusters which are very close to some big identified clusters; we shall elaborate on this shortly.

Problem Setup. We denote the clusters by $X_1, X_2, \ldots$, which are point sets in the Euclidean space. For each X_i, let $\mu_i = \frac{1}{|X_i|}\sum_{x \in X_i} x$ be its centroid. For simplicity we refer to X_i as "cluster i" or "i-th cluster". The number of clusters is unknown at the beginning. Our goal is to find all "big" clusters and their approximate centroids $\tilde{\mu}_i$.

To specify what we mean by finding all big clusters, we need to introduce a concept called *reducibility*. Several previous studies [2, 11, 12] on the K-means/median clustering problem assumed that after finding K main clusters, the residual ones satisfy some reducibility condition, which states that using the K discovered cluster centers to cover the residual ones would only increase the total cost by a small $(1+\epsilon)$ factor. Similarly, we consider the following reducibility condition, based on the usual cost function.

Definition 1 (cost function)**.** Suppose that X and C are the set of data points and centers, respectively. The cost of covering X using

*Supported in part by NSF IIS-1633215 and CCF-1844234.
†Supported in part by NSF IIS-1633215 and CCF-1844234. Corresponding author.

CIKM '21, November 1–5, 2021, Virtual Event, QLD, Australia

ACM ISBN 978-1-4503-8446-9/21/11…$15.00
https://doi.org/10.1145/3459637.3482365

Table 1.1: Notations

X	set of input points
r	index of the round
I	set of indices of recovered clusters so far
k	number of recovered clusters so far
Q	set of indices of newly discovered clusters in the current round
q	size of Q; number of newly discovered clusters in the current round
S	multiset of samples. Each sample has the form of (x, i), where x is the point and i the cluster index.
x_j^*	reference point in cluster j for rejection sampling
S_j	multiset of uniform samples in cluster j returned by rejection sampling. Note that $S \neq \bigcup_j S_j$.
W	set of indices of clusters to recover
K	total number of recovered clusters at the end
L	total number of discovered clusters at the end

C is $\Phi(X, C) = \sum_{x \in X} \min_{c \in C} \|x - c\|^2$, where $\|\cdot\|$ denotes the Euclidean norm. When $C = \{c\}$ is a singleton, we also write $\Phi(X, \{c\})$ as $\Phi(X, c)$. When $C = \emptyset$, we define $\Phi(X, C) = |X| \cdot \sup_{x,y \in X} \|x-y\|^2$.

Definition 2 (ϵ-reducibility)**.** Let $X_1, X_2, \ldots$ be true clusters in X. Let I be a subset of indices of clusters in X. We say the clusters in X are ϵ-reducible w.r.t. I if it holds for each $\ell \notin I$ that

$$\Phi\left(X_\ell, \bigcup_{i \in I} \{\mu_i\}\right) \leq \epsilon \sum_{i \in I} \Phi(X_i, \mu_i).$$

Intuitively, this reducibility condition states that the clusters outside I will be *covered* by the centers of the clusters in I with only a small increase in the total cost.

Our goal is to find a subset of indices I such that the set of true clusters in X is ϵ-reducible w.r.t. I.

Note that in this formulation, we do **not** attempt to optimize any objective function; instead, we just try to recover the approximate centroids of a set of clusters to which all other clusters are reducible. Inevitably we have to adopt a distance function in our definition of irreducibility, and we choose the widely used D^2 (Euclidean-squared) distance function so that we can still use the D^2-sampling method developed and used in the earlier works [2, 3, 11]. The D^2-sampling will be introduced in Definition 3.

To facilitate discussion, we list in Table 1.1 the commonly used notations in our algorithms and analyses. A cluster i is said to be "discovered" if any point in X_i has been sampled and "recovered" when the approximate centroid $\tilde{\mu}_i$ is computed. As discussed in the preceding paragraph, it is possible that the total number of recovered clusters, K, is less than the total number of discovered clusters, L, and that L is less than the true number of clusters.

Our Contributions. We provide two clustering algorithms with theoretically proven bounds on the total number of oracle queries in the circumstance of no prior knowledge on the number of clusters and no predefined objective function. To the best of our knowledge, these are the first algorithms for the clustering problem of its kind. Both our algorithms output $(1 + \epsilon)$-approximate centers for all recovered clusters. Our first algorithm makes $\tilde{O}(\epsilon^{-4}K^2L^2)$ queries (Section 3) and the second algorithm makes $\tilde{O}(\epsilon^{-4}KL^2)$ queries (Section 4).[1]

We also conduct an extensive set of experiments demonstrating the effectiveness of our algorithms on both synthetic and real-world data; see Section 6.

We further extend our algorithms to the case of a noisy same-cluster oracle which errs with a constant probability $p < 1/2$. This extension has been deferred to Appendix in the full version of this paper.[2]

We remark that since our algorithms target *sublinear* (i.e., $o(|X|)$) number of oracle queries, in the general case where the shape of clusters can be arbitrary, it is impossible to classify correctly all points in the datasets. But the approximate centers outputted by the algorithms can be used to *efficiently and correctly* classify any newly inserted points (i.e., database queries) as well as existing database points (when needed), using a natural heuristic (Heuristic 1) that we shall introduce in Section 6. Our experiments show that most points can be classified using only *one* additional oracle query.

Related Work. As mentioned, the semi-supervised active clustering framework was first introduced in [4], where the authors considered the K-means clustering under the γ-margin assumption. Ailon et al. [1, 2] proposed approximation algorithms for the K-means and correlation clustering that compute a $(1 + \epsilon)$-approximation of the optimal solution with the help of same-cluster queries. Chien et al. [6] studied the K-means clustering under the same setting as that in [2], but used uniform sampling instead of D^2-sampling and worked under the assumption that no cluster (in the true clustering) has a small size. Saha and Subramanian [15] gave algorithms for correlation clustering with the same-cluster query complexity bounded by the optimal cost of the clustering. Gamlath et al. [8] extended the K-means problem from Euclidean spaces to general finite metric spaces with a strengthened guarantee that recovered clusters largely overlap with respect to the true clusters. All these works, however, assume that the ground truth clustering (known by the oracle) is consistent with the target objective function, which is unrealistic in most real world applications.

Bressan et al. [5] considered the case where the clusters are separated by ellipsoids, in contrast to the balls as suggested by the usual K-means clustering objective. However, their algorithm still requires the knowledge of K and has a query complexity that depends on n (although logarithmically), which we avoid in this work.

Mazumdar and Saha studied clustering n points into K clusters with a noisy oracle [13] or side information [14]. The noisy oracle gives incorrect answers to same-cluster queries with probability $p < 1/2$ (so that majority voting still works). The side information is the similarity score between each pair of data points, generated from a distribution f_+ if the pair belongs to the same cluster and from another distribution f_- otherwise. Algorithms proposed in these papers guarantee to recover the true clusters of size at least $\Omega(\log n)$, however, with query complexities at least $\Omega(Kn)$, much larger than what we are interested to achieve in this paper.

[1] In $\tilde{O}(\cdot), \tilde{\Omega}(\cdot), \tilde{\Theta}(\cdot)$ we use "~" to hide logarithmic factors.
[2] https://arxiv.org/abs/2108.07383

Algorithm 2.1 CLASSIFY(x). The overall algorithm which maintains the number L of discovered clusters and a representative point z_i for each $i = 1, \ldots, L$.

```
for i = 1 to L do
    if ORACLE(x, z_i) then
        return i
L ← L + 1
z_L ← x
return L
```

Huleihel et al. [9] studied the overlapping clustering with the aid of a same-cluster oracle. Suppose that A is an $n \times K$ clustering matrix whose i-th row is the indicator vector of the cluster membership of the i-th element. The task is to recover A from the similarity matrix AA^T using a small number of oracle queries.

Finally, we note that same-cluster queries have been used extensively for *entity resolution* (or, *de-duplication*) [7, 16–19].

2 PRELIMINARIES

In this paper we consider point sets in the canonical Euclidean space $(\mathbb{R}^d, \|\cdot\|)$. The geometric centroid, or simply centroid, of a finite point set X is defined as $\mu(X) = \frac{1}{|X|}\sum_{x\in X} x$. It is known that $\mu(X)$ is the minimizer of the 1-center problem $\min_c \sum_{x\in X} \|x - c\|^2$. The next lemma provides a guarantee on approximating the centroid of a cluster using uniform samples.

Lemma 1 ([10]). *Let S be a set of points obtained by independently sampling M points uniformly at random with replacement from a point set $X \in \mathbb{R}^d$. Then for any $\delta > 0$, it holds that*

$$\Pr\{\Phi(S, \mu(S)) \le (1 + 1/(\delta M))\, \Phi(X, \mu(X))\} \ge 1 - \delta.$$

We define the D^2-sampling of a point set X with respect to a point set C as follows.

Definition 3 (D^2-sampling)**.** The D^2-sampling of a point set X with respect to a point set C returns a random point $p \in X$ subject to the distribution defined by $\Pr\{p = x\} = \Phi(\{x\}, C)/\Phi(X, C)$ for all $x \in X$.

In this paper we use a same-cluster oracle, that is, given two data points $x, y \in X$, ORACLE(x, y) returns true if x and y belong to the same cluster and false otherwise. For simplicity of the algorithm description, we shall instead invoke the function CLASSIFY(x) to obtain the cluster index of x, which can be easily implemented using the same-cluster oracle, as shown in Algorithm 2.1. This implies that the number of oracle queries is at most L times the number of samples.

3 BASIC ALGORITHM

Algorithm. Despite the fact that the algorithm in [2] cannot be used directly because K is unknown to us, we briefly review the algorithm below. The algorithm gradually "discovers" and then "recovers" more and more clusters by taking D^2-samples. It recovers K clusters in K rounds, one in each round. To recover one cluster, it takes sufficient D^2-samples w.r.t. the approximate centers $\{\tilde{\mu}_i\}_{i\in I}$, where I is the set of the recovered clusters, and finds the largest unrecovered cluster j. For this cluster j, it obtains sufficient uniform samples via rejection sampling and invokes Lemma 1 to compute an approximate centroid $\tilde{\mu}_j$. Then it includes j in the set I of recovered clusters and proceeds to the next round. It is shown that poly(K/ϵ) samples each round can achieve the failure probability of $O(1/K)$ and taking a union bound over all K rounds yields a constant overall failure probability.

There are two major difficulties of adapting this algorithm to our setting where K is unknown.

(1) The number of rounds is unknown. It is not clear when our algorithm should terminate, i.e., when we are confident that there are no more irreducible clusters. The failure probability of each round also needs to be redesigned and cannot be $O(1/K)$.
(2) Since K is unknown, we cannot predetermine the number of samples to use and have to maintain dynamically various stopping criteria. For instance, it is subtle to determine which one the largest cluster is. If we stop too early, we may identify a cluster that is actually small and will need a large number of samples in order to obtain enough uniform samples from this cluster when doing rejection sampling; if we stop too late, we can make a more accurate decision but may have already taken too many samples.

To address the first issue, we observe that $\Omega(\epsilon^{-1}\log K)$ samples is sufficient to test whether there are ϵ-irreducible clusters left with constant probability. We also assign the failure probability of the r-th round to be a_r such that $\sum_{r=1}^{\infty} a_r$ is a small constant.

To address the second issue, observe that the ϵ-irreducibility condition implies that all unrecovered clusters together have a Φ-value of $\Omega(\epsilon)$ and thus the largest unrecovered cluster has a Φ-value of $\Omega(\epsilon/q)$, where q is the number of newly discovered clusters. We can show that $\tilde{O}(q/\epsilon)$ samples suffices to ensure the identification of a cluster with Φ-value at least $\Omega(\epsilon/q)$ and thus we can, since the value of q increases as the number of samples grows, keep sampling until $\tilde{O}(q/\epsilon)$ samples are obtained. Note that this is a dynamic criterion in contrast to the predetermined one in [2]. We then choose the largest cluster j and carry out the rejection sampling and estimate the approximate centroid $\tilde{\mu}_j$ with a careful control over the number of samples.

We present our algorithm in Algorithm 3.1 without attempts to optimize the constants. In our algorithm, each round is divided into four phases. Phase 1 tests whether there are new clusters to recover, using $O(\epsilon^{-1}\ln k)$ samples, where k is the number of the recovered clusters. If no new clusters are found, the algorithm would terminate. Phase 2 samples more points and finds the largest cluster discovered so far, which we shall aim to recover in the two remaining phases. Phase 3 samples more points and finds a reference point x_j^* to be used in the rejection sampling in the next phase. Phase 4 executes rejection sampling using the reference point x_j^* such that the returned samples are uniform from the cluster j. We need to collect $\tilde{O}(k/\epsilon)$ uniform samples before calculating the approximate centroid $\tilde{\mu}_j$ for cluster j. We then declare that cluster j is recovered and start the next round.

We would like to remark that the D^2-sampling which our algorithm uses has a great advantage over uniform sampling [6] when

Algorithm 3.1 The Basic Algorithm.

$I \leftarrow \emptyset$
$k \leftarrow 0, r \leftarrow 0$
repeat
  $r \leftarrow r + 1$
  $Q \leftarrow \emptyset, S \leftarrow \emptyset$
  $T_1 \leftarrow 8\epsilon^{-1} \ln(10(k+1))$
  while $Q = \emptyset$ and $|S| \le T_1$ **do** } Phase 1
    $(x, Q, S) \leftarrow \textsc{Sample}(I, \{\tilde{\mu}_i\}_{i\in I}, Q, S)$ } Phase 1
  end while
  if $Q \neq \emptyset$ **then**
    while $|S| \le 96|Q| \ln(10(k+|Q|))/\epsilon$ **do** } Phase 2
      $(x, Q, S) \leftarrow \textsc{Sample}(I, \{\tilde{\mu}_i\}_{i\in I}, Q, S)$ } Phase 2
    end while } Phase 2
    $j \leftarrow \arg\max_i |\{u \in S : u = (x, i)\}|$ } Phase 2
    $q \leftarrow |Q|$
    $T_2 \leftarrow 2^{12} q \ln(10(k+q))/\epsilon^2$
    while $|S| \le T_2$ **do** } Phase 3
      $(x, Q, S) \leftarrow \textsc{Sample}(I, \{\tilde{\mu}_i\}_{i\in I}, Q, S)$ } Phase 3
    end while } Phase 3
    $x_j^* \leftarrow \arg\min_{x:(x,j)\in S} \Phi(\{x\}, \{\tilde{\mu}_i\}_{i\in I})$ } Phase 3
    $S_j \leftarrow \emptyset$
    $T_3 \leftarrow 20\epsilon^{-1} r \ln^2(10r)$
    $S_j \leftarrow \textsc{RejSamp}(I, \{\tilde{\mu}_i\}_{i\in I}, T_3, \{j\}, \{x_j^*\})$ } Phase 4
    $\tilde{\mu}_j \leftarrow (1/|S_j|) \sum_{x\in S_j} x$ } Phase 4
    $k \leftarrow k + 1$
    $I \leftarrow I \cup \{j\}$
  end if
until $Q = \emptyset$ ▷ no new cluster is discovered

Algorithm 3.2 $\textsc{Sample}(I, \{\tilde{\mu}_i\}_{i\in I}, Q, S)$: Returning a single D^2-sample w.r.t. recovered clusters I

$x \leftarrow$ a point returned by D^2-sampling (w.r.t. $\{\tilde{\mu}_i\}_{i\in I}$)
$j \leftarrow \textsc{Classify}(x)$
if $j \notin I$ **then**
  $Q \leftarrow Q \cup \{j\}$
$S \leftarrow S \cup \{(x, j)\}$
return (x, Q, S)

Algorithm 3.3 $\textsc{RejSamp}(I, \{\tilde{\mu}_i\}_{i\in I}, T, W, \{x_j^*\}_{j\in W})$: Rejection sampling on heavy clusters

repeat
  $x \leftarrow$ a point returned by D^2-sampling w.r.t. $\{\tilde{\mu}_i\}_{i\in I}$
  $j \leftarrow \textsc{Classify}(x)$
  if $j \in W$ **then**
    Add x to S_j with probability $\frac{\epsilon}{128} \cdot \frac{\Phi(\{x_j^*\}, \{\tilde{\mu}_i\}_{i\in I})}{\Phi(\{x\}, \{\tilde{\mu}_i\}_{i\in I})}$
until $|S_j| \ge T$ for all $j \in W$
return $\{S_j\}_{j\in W}$

there are small and faraway clusters. Uniform sampling would require a large number of samples to find small clusters, regardless of their locations. On the other hand, faraway clusters are clearly irreducible and are rightfully expected to be recognized. The D^2-sampling captures the distance information such that those small faraway clusters may have a large Φ-value and can be found with much fewer samples.

Analysis. Throughout the execution of the algorithm, we keep the loop invariant that

$$\Phi(X_i, \tilde{\mu}_i) \le (1+\epsilon)\Phi(X_i, \mu_i) \tag{3.1}$$

for all $i \in I$, i.e., the approximate centers are good for all recovered clusters.

It is clear that the main algorithm will terminate. We prove that the algorithm is correct, i.e., the algorithm will find all clusters with $\tilde{\mu}_i$'s being all good approximate centroids. We first need a lemma showing that the undiscovered clusters are heavy under the assumption of ϵ-reducibility. All proofs are postponed to Appendix in the full version.

Lemma 2. *Suppose the ϵ-reducibility constraint w.r.t. I does not hold. It holds for $\epsilon \in (0, 1]$ that $\sum_{i\notin I} \Phi(X_i, \tilde{C}) / \sum_i \Phi(X_i, \tilde{C}) \ge \epsilon/4$.*

Using the preceding lemma, we are ready to show that Phase 1 of Algorithm 3.1 discovers all heavy clusters with high probability.

Lemma 3. *Upon the termination of Algorithm 3.1, with probability at least 0.95, the ϵ-reducibility condition w.r.t. I is satisfied and an approximate centroid $\tilde{\mu}_i$ that satisfies (3.1) is obtained for every $i \in I$.*

Next we upper bound the number of samples. For each cluster $j \notin I$ define its conditional sample probability

$$p_j = \Phi(X_j, C) / \textstyle\sum_{i\notin I} \Phi(X_i, C), \tag{3.2}$$

where $C = \{\tilde{\mu}_i\}_{i\in I}$ is the set of approximate centers so far. For each p_j, we also define an empirical approximation $\hat{p}_j = s_j / \sum_{i\notin I} s_i$, where s_i denotes the number of samples seen so far which belong to cluster i.

The next lemma shows that the new clusters we find in Phase 2 are heavy.

Lemma 4. *With probability at least $1 - 2/(100(k+q)^2)$, it holds that $p_j \ge 1/(3q)$ for all $j \in W$.*

To analyse Phases 3 and 4, we define an auxiliary point set

$$Y_j = \left\{ y \in X_j : \Phi(\{y\}, C)/\Phi(X_j, C) \le 2/|X_j| \right\}. \tag{3.3}$$

Points in Y_j are good pivot points for the rejection sampling procedure in Phase 4. We first show that we can find a good pivot point x_j^* in Phase 3.

Lemma 5. *Assume that the event in Lemma 4 occurs. Then $x_j^* \in Y_j$ with probability at least $1 - 1/(100(k+q)^2)$.*

We then show that, with a good pivot point x_j^*, the rejection sampling procedure in Phase 4 obtains sufficiently many uniform samples from cluster j so that we can calculate a good approximate center later.

Lemma 6. *Assume that the event in Lemma 5 occurs. By sampling $s = 2^{23}\epsilon^{-4} qr \ln^2(10r)$ points in total, with probability at least $1 - \exp(-8r)$, every cluster $j \in W$ has $T_3 = 20\epsilon^{-1} r \ln^2(10r)$ samples returned by the rejection sampling procedure.*

Combining Lemmata 3, 4, 5 and 6, we arrive at the main conclusion of the basic algorithm.

Theorem 7. *With probability at least* 0.9*, Algorithm 3.1 finds a set I of clusters that satisfies Definition 2, and obtains for every $i \in I$ an approximate centroid $\tilde{\mu}_i$ that satisfies* (3.1)*, using $O(\epsilon^{-4}K^2L^2\log^2 L)$ same-cluster queries in total, where K is the total number of recovered clusters and L is the total number of discovered clusters.*

Remark 1. When the clusters are ϵ-irreducible with respect to any subcollection of the clusters, we shall recover all L clusters using $O(\epsilon^{-4}L^4\log^2 L)$ same-cluster queries.

4 IMPROVED ALGORITHM

In this section, we improve the basic algorithm by allowing the recovery of multiple clusters in each round. A particular case in which the basic algorithm would suffer from a prodigal waste of samples is that there are in total K clusters of approximately the same size and they are ϵ-irreducible with respect to any subset $I \subset [K]$. In such case, our basic algorithm requires a sample of size $\tilde{O}(K/\epsilon)$ from that cluster, which in turn requires a global sample of size $\tilde{O}(K^2/\epsilon)$. Summing over K rounds leads to a total number $O(K^3/\epsilon)$ of samples. However, with $\tilde{O}(K^2/\epsilon)$ global samples, we can obtain $\tilde{O}(K/\epsilon)$ samples from every cluster and recover all clusters in the same round. This reduces a factor of K in the total number of samples. The goal of this section is to improve the sample/query complexity of the basic algorithm by a factor of K even in the *worst* case.

To recover an indefinite number of clusters in each round, it is a natural idea to generalize the basic algorithm to identifying a set W of clusters, instead of the biggest one only, from which we shall obtain uniform samples via rejection sampling and calculate the approximate centroids $\{\tilde{\mu}_j\}_{j\in W}$. Here we face an ever greater challenge as to how to determine W. The naïve idea of including all clusters of Φ-value at least $\Omega(\epsilon/r)$ runs into difficulty: as we sample more points, the value of r may increase and we may need to include more and more points, so when do we stop? We may expect that the value r will stabilize after sufficiently many samples, but it is difficult to quantify "stable" and control the number of samples needed.

Our solution to overcome the above-mentioned difficulty is to split the clusters into bands. To explain this we introduce a few more notations. For each unrecovered cluster $j \notin I$, recall that we defined in (3.2) its conditional sample probability $p_j = \Phi(X_j, C)/\sum_{i\notin I}\Phi(X_i, C)$; we also define its empirical approximation to be $\hat{p}_j = s_j/\sum_{i\notin I} s_i$, where s_i denotes the number of samples seen so far which belong to the cluster i. We split $\{\hat{p}_i\}_{i\notin I}$ into $L+1$ bands for $L = 3\log q$, where the ℓ-th band is defined to be

$$B_\ell = \{i \notin I : 2^{-\ell} < \hat{p}_i \le 2^{-\ell+1}\}$$

for $\ell \le L$ and the last band B_{L+1} consists of all the remaining clusters i (i.e., $\hat{p}_i \le 1/q^3$). We say a band B_ℓ is *heavy* if $\sum_{i\in B_\ell}\hat{p}_i \ge 1/(3L)$. The values $\hat{p}_i$ and the bands are dynamically adjusted as the number of samples grows.

Instead of focusing on individual clusters and identifying the heavy ones, we shall identify the heavy bands at an appropriate moment and recover all the clusters in those heavy bands. Intuitively, it takes much fewer samples for bands to stabilize than for individual clusters. For instance, consider a heavy band which consists of many small clusters. In this case, we will have seen most of the clusters in the band without too many samples, although many of the $\hat{p}_i$'s may be far off from p_i's. The important observation here is that we have *seen* those clusters; in contrast, if we consider individual clusters, we would have missed many of those clusters and will focus on recovering a few of them, which would cause a colossal waste of samples in the rejection sampling stage, for a similar reason explained at the beginning of this section. Therefore such banding approach enables us to control the number of samples needed.

Algorithm 4.1 The Improved Algorithm

$I \leftarrow \emptyset$
$k \leftarrow 0, r \leftarrow 0, K \leftarrow 1/2$
repeat
  $K \leftarrow 2K$
  repeat
    $r \leftarrow r + 1$
    $Q \leftarrow \emptyset, S \leftarrow \emptyset$
    $T_1 \leftarrow 8\epsilon^{-1}\ln(10(k+1))$
    while $Q = \emptyset$ and $|S| \le T_1$ **do** } Phase 1
      $(x, Q, S) \leftarrow \textsc{Sample}(I, \{\tilde{\mu}_i\}_{i\in I}, Q, S)$
    end while
    if $Q \ne \emptyset$ and $k < K$ **then**
      repeat
        $(x, Q, S) \leftarrow \textsc{Sample}(I, \{\tilde{\mu}_i\}_{i\in I}, Q, S)$
        for each $i \in Q$ **do**
          $\hat{p}_i \leftarrow |\{u \in S : u = (x, i)\}|/|S|$
        end for
        Split the clusters in to bands $\{B_\ell\}$
        $W \leftarrow$ all clusters in heavy bands
      until $|S| \ge 1600|W|\log|Q|\ln(10(k+|Q|))/\epsilon$ } Phase 2
      $q \leftarrow |Q|$
      $T_2 \leftarrow 2^{17}|W|\log q\ln(10(k+q))/\epsilon^2$
      while $|S| \le T_2$ **do**
        $(x, Q, S) \leftarrow \textsc{Sample}(I, \{\tilde{\mu}_i\}_{i\in I}, Q, S)$
      end while
      for each $j \in W$ **do**
        $x_j^* \leftarrow \arg\min_{x:(x,j)\in S}\Phi(\{x\}, \{\tilde{\mu}_i\}_{i\in I})$
        $S_j \leftarrow \emptyset$
      end for } Phase 3
      $T_3 \leftarrow 30K/\epsilon$
      $\{S_j\}_{j\in W} \leftarrow \textsc{RejSamp}(I, \{\tilde{\mu}_i\}_{i\in I}, T_3, W, \{x_j^*\}_{j\in W})$
      for each $j \in W$ **do**
        $\tilde{\mu}_j \leftarrow (1/|S_j|)\sum_{x\in S_j} x$
      end for } Phase 4
      $k \leftarrow k + |W|$
      $I \leftarrow I \cup W$
    end if
  until $Q = \emptyset$ or $k \ge K$
until $Q = \emptyset$ and $k \le K$

Our improved algorithm is presented in Algorithm 4.1. Each round remains divided into four phases. Phase 1 remains testing

new clusters. Starting from Phase 2 comes the change that instead of recovering only the largest sampled cluster j, we identify a subset W of indices of the newly discovered clusters and recover all clusters in W. Phase 2 samples more points and identifies this subset W by splitting the discovered clusters into bands and choosing all clusters in the heavy bands. Phase 3 samples more points and finds a reference point x_j^* for each $j \in W$. Phase 4 keeps sample points until each cluster $j \in W$ sees $\Theta(\epsilon^{-1}|W|r)$ points and then calculates the approximate centers $\tilde{\mu}_j$ for each $j \in W$. The clusters in W will then be added to I as recovered clusters before the algorithm starts a new round.

A technical subtlety lies in controlling the failure probability in our new algorithm. In the basic algorithm, the failure probability in the r-th round is $a_r = 1/(r\,\mathrm{poly}(\log r))$. If we recover w_r clusters in r-th round, this failure probability for each of the w_r clusters needs to be $1/(w_r a_r)$, and as a consequence, $\tilde{O}(w_r^2 a_r)$ samples are needed in the r-th round. In the worst case this becomes $\tilde{O}(K^3)$ samples, the same as in the basic algorithm, when both $w_r = \Theta(K)$ and $a_r = \tilde{\Theta}(1/K)$. To resolve this, we guess $K = 1, 2, \ldots$ in powers of 2 and assign a_r to be w_r/K. For each fixed guess K, the number of samples is bounded by $\tilde{O}(K^2)$; iterating over guesses incurs only an additional $\log K$ factor.

The analysis of our improved algorithm follows the same sketch of the basic algorithm and we only present the changes below. All proofs are postponed to Appendix in the full version. The next lemma offers a similar guarantee as Lemma 3.

Lemma 8. *Upon the termination of Algorithm 4.1, with probability at least* 0.95*, the ϵ-reducibility condition w.r.t. I is satisfied and an approximate centroid $\tilde{\mu}_i$ that satisfies* (3.1) *is obtained for every $i \in I$.*

Next we upper bound the number of samples with a lemma analogous to Lemma 4.

Lemma 9. *With probability at least $1 - 1/(50(k+q)^2)$, it holds that $p_j \geq 1/(70|B_{\ell(j)}|\log q)$ for all $j \in W$.*

Recall that Y_j was defined in (3.3) for all $j \in W$. The following two lemmata are the analogue of Lemmata 5 and 6, respectively.

Lemma 10. *Assume that the event in Lemma 9 holds. With probability at least $1 - 1/(25(k+q)^2)$, it holds that $x_j^* \in Y_j$ for all $j \in W$.*

Lemma 11. *Assume that the event in Lemma 10 holds. By sampling $s = 2^{28}\epsilon^{-4}|W|K\log q$ points in total, with probability at least $1 - |W|\exp(-8K)$, every cluster $j \in W$ has $T_3 = 30K/\epsilon$ samples returned by the rejection sampling procedure.*

Now we are ready to prove the main theorem of the improved algorithm.

Theorem 12. *With probability at least* 0.9*, Algorithm 4.1 (the improved algorithm) finds all the clusters and obtain for every i an approximate centroid $\tilde{\mu}_i$ that satisfies* (3.1)*, using $O(\epsilon^{-4}KL^2\log^3 K\log L)$ same-cluster queries in total, where K is the number of recovered clusters and L is the total number of discovered clusters.*

Remark 2. The preceding theorem improves the query complexity of the basic algorithm by about a factor of K, as desired. In particular, when the clusters are ϵ-irreducible with respect to any subcollection of the clusters, we shall recover all L clusters using $O(\epsilon^{-4}L^3\log^3 L)$ same-cluster queries, better than basic algorithm by about a factor of L.

5 NOISY ORACLES

Our algorithm can be extended to the case where the same-cluster oracle errs with a constant probability $p < 1/2$. The full details are postponed to Appendix in the full version.

Theorem 13. *Suppose that the same-cluster oracle errs with a constant probability $p < 1/2$. With probability at least* 0.6*, there is an algorithm that finds all the clusters and obtains for every i an approximate centroid $\tilde{\mu}_i$ that satisfies* (3.1)*, using $\tilde{O}(\epsilon^{-6}L^6K)$ same-cluster queries in total.*

6 EXPERIMENTS

In this section, we conduct experimental studies of our algorithms. As we mentioned before, all previous algorithms need to know K and it is impossible to convert them to the case of unknown K (with the only exception of [6]). In particular, we note the inapplicability of the k-means algorithm and the algorithm in [2].

- The k-means algorithm is for an unsupervised learning task and returns clusters which are always spherical, so it is undesirable for arbitrary shapes of clusters and it makes sense to examine the accuracy in terms of misclassified points. Our problem is semi-supervised with a same-cluster oracle, which can be used to recover clusters of arbitrary shapes and guarantees no misclassification. The assessment of the algorithm is therefore the number of discovered clusters and the number of samples. Owing to the very different nature of the problems, the k-means algorithm should not be used in our problem or compared with algorithms designed specifically for our problem.
- The algorithm in [2] runs in k rounds and take $2^{12}k^3/\epsilon^2$ samples in the first step of each round. With $k = 10$ and $\epsilon = 0.1$, it needs to sample in each round at least 4×10^8 points, usually larger than the size of a dataset. (Our algorithm can be easily simplified to handle such cases, see the subsection titled "Algorithms" below.)

The only exception, the algorithm in [6], is just simple uniform sampling and whether or not K is known is not critical. This uniform sampling algorithm will be referred to as Uniform below.

Datasets. We test our algorithms using both synthetic and real world datasets.

For synthetic datasets, we generate n points that belong to K clusters in the d-dimensional Euclidean space. Since in the real world the cluster sizes typically follow a *power-law* distribution, we assign to each cluster a random size drawn from the widely used *Zipf distribution*, which has the density function $f(x) \propto x^{-\alpha}$, where x is the size of the cluster and α a parameter controlling the skewness of the dataset. We then choose a center μ_i for each cluster $i \in [K]$ in a manner to be specified later and generate the points in the cluster from the multivariate Gaussian distribution $N(\mu_i, \sigma^2 I_d)$, where I_d denotes the $d \times d$ identity matrix.

Now we specify how to choose the centers. In practice, clusters in the dataset are not always well separated; there could be clusters whose centers are close to each other. We thus use an additional parameter p to characterize this phenomenon. In the default setting

of $p = 0$, all centers of the K clusters are drawn uniformly at random from $[0, b]^d$. When $p > 0$, we first partition the clusters into groups as follows: Think of each cluster as a node. For each pair of clusters, with probability p we add an edge between the two nodes. Each connected component of the resulting graph forms a group. Next, for each group of clusters, we pick a random cluster and choose its center μ uniformly at random in $[0, b]^d$. For each of the remaining clusters in the group, we choose its center uniformly at random in the neighborhood of radius ρ centered at μ.

We use the following set of parameters as the default setting in our synthetic datasets: $n = 10^6$, $K = 100$, $\alpha = 2.5$, $\sigma = 0.3$, $b = 5$, $d = 10$ and $\rho = 0.1$.

We also use the following two real-world datasets.

- Shuttle [3]: it describes the radiator positions in a NASA space shuttle. There are 58,000 points with 9 numerical attributes, and 7 clusters in total.
- Kdd [4]: this dataset is taken from the 1999 KDD Cup contest. It contains about 4.9M points classified into 23 clusters. The original dataset has 41 attributes. We retain all numerical attributes except one that contains only zeros, resulting in 33 attributes in total.

Each feature in the real world datasets is normalized to have zero mean and unit standard deviation.

Algorithms. We compare three algorithms listed below. A cluster is said to be *heavy* when the number of (uniform) samples obtained from this cluster is more than a predetermined threshold, which is set to be 10 in our experiments.

- Uniform: This is uniform sampling. That is, we keep getting random samples from the dataset one by one and identify their label by same-cluster queries. We recover a cluster when it becomes heavy.
- Basic: This is based on our basic algorithm (Algorithm 3.1). We recover clusters one at a time when it becomes heavy.
- Batched: This is a simplified version of the improved algorithm (Algorithm 4.1). Instead of partitioning clusters to bands and then recovering all clusters in the heavy bands, in each round we just keep sampling points until the fraction of points in the heavy clusters is more than a half, at which moment we try to recover all heavy clusters in a batch.

Recall that in our basic and improved algorithms, we always "ignore" the samples belonging to the unrecovered clusters in the previous rounds, which will not affect our theoretical analysis. But in practice it is reasonable to reuse these "old" samples in the succeeding rounds so that we are able to recover some clusters earlier. Because of such a sample-reuse procedure, the practical performance difference between the Basic and Batched algorithms becomes less significant. Recall that in theory, the main improvement of Batched over Basic is that we try to avoid wasting too many samples in each round by recovering possibly more than one cluster at a time. But still, as we shall see shortly, Batched outperforms Basic in all metrics, though sometimes the gaps may not seem significant.

[3] https://archive.ics.uci.edu/ml/datasets/Statlog+(Shuttle).
[4] http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html.

Measurements. To measure the same-cluster query complexity, we introduce two assessments. The first is *fixed-budget*, where given a fixed number of same-cluster queries, we compare the number of clusters recovered by different algorithms. The second is *fixed-recovery*, where each algorithm needs to recover a predetermined number of clusters, and we compare the numbers of same-cluster queries the algorithms use.

We also report the error of the approximate centroid of each recovered cluster. For a recovered cluster X_i, let μ_i be its centroid and $\hat{\mu}_i$ be the approximate center. We define the centroid approximation error to be $(\Phi(X_i, \hat{\mu}_i) - \Phi(X_i, \mu_i))\,/\Phi(X_i, \mu_i)$.

For Basic and Batched, we further compare their running time and round usage. We note that a small round usage is very useful if we want to efficiently parallelize the learning process.

Finally, we would like to mention one subtlety when measuring the number of same-cluster queries. Since all algorithms that we are going to test are sampling based, and for each sampled point we need to identify its label by same-cluster queries, which, in the worst case, takes k queries, where k is the number of the discovered clusters so far. In practice, however, a good query order/strategy can save us a lot of same-cluster queries. We apply the following query strategy for all tested algorithms.

Heuristic 1. For each discovered cluster, we maintain its approximate center (using the samples we have obtained). When a new point is sampled, we query the centers of discovered clusters based on their distances to the new point in the *increasing* order as follows. Suppose that the new point is x. We calculate the distances between x and all approximate centers $\hat{\mu}_i$, and sort them in increasing order, say, $d(x, \hat{\mu}_1) \le d(x, \hat{\mu}_2) \le \cdots$. Then for $i = 1, 2, \ldots$ sequentially, we check whether x belongs to cluster i by querying whether x and some sample point from cluster i belong to the same cluster. If x does not belong to any discovered cluster, a new cluster will be created.

We will also use this heuristic to test the effectiveness of approximate centers for classifying new points. That is, for a newly inserted point q, we count the number of same-cluster queries needed to correctly classify q using the approximate centers and Heuristic 1.

Computation Environments. All algorithms are implemented in C++, and all experiments were conducted on Dell PowerEdge T630 server with two Intel Xeon E5-2667 v4 3.2GHz CPU with eight cores each and 256GB memory.

6.1 Results

The three algorithms, Uniform, Basic and Batched, are compared on the same-cluster query complexity, the quality of returned approximate centers, the running time and the number of rounds. All results are the average values of 100 runs of experiments.

Query Complexity. For each of the synthetic and real-world datasets, we measure the query complexities under both fixed-budget and fixed-recovery settings.

The results for synthetic dataset, the Shuttle dataset and the Kdd dataset are plotted in Figure 1, Figure 2 and Figure 3, respectively. In all figures, the left column corresponds to the fixed-budget case and the right column the fixed-recovery case. We note that

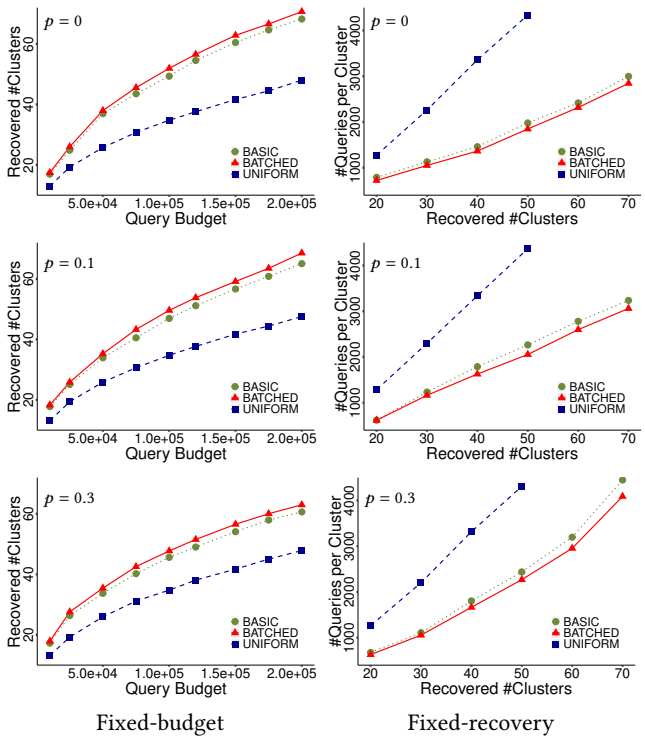

Figure 1: Performance comparison on synthetic datasets.

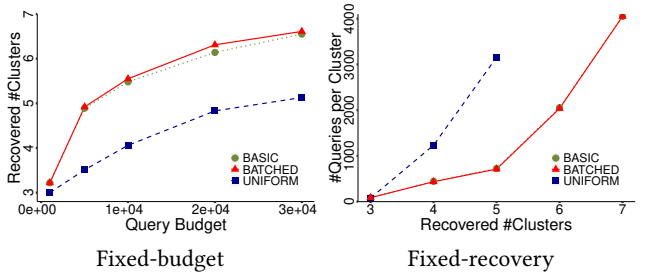

Figure 2: Performance comparison on SHUTTLE dataset. The curves for BASIC and BATCHED almost overlap for fixed-recovery.

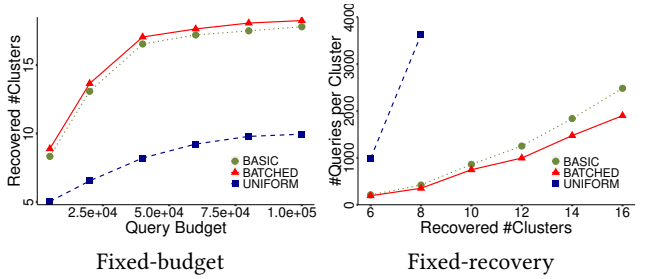

Figure 3: Performance comparison on KDD dataset.

some points for UNIFORM are missing since they are out of the boundary. The exact values of all experiments can be found in Appendix in the full version.

#Clusters	Algorithms	20	30	40	50	60	70
$p = 0$	BASIC	8.70%	8.70%	8.50%	8.41%	8.43%	8.37%
	BATCHED	8.74%	8.60%	8.56%	8.64%	8.43%	8.51%
	UNIFORM	9.90%	9.43%	9.58%	9.22%	9.24%	9.18%
$p = 0.1$	BASIC	8.86%	8.68%	8.72%	8.39%	8.51%	8.45%
	BATCHED	8.76%	8.64%	8.45%	8.49%	8.50%	8.42%
	UNIFORM	9.59%	9.64%	9.39%	9.42%	9.30%	9.18%
$p = 0.3$	BASIC	8.87%	8.68%	8.56%	8.40%	8.34%	8.38%
	BATCHED	8.95%	8.64%	8.57%	8.37%	8.35%	8.39%
	UNIFORM	9.75%	9.44%	9.45%	9.35%	9.25%	9.28%

Table 6.2: Error rates on synthetic datasets

#Clusters	2	3	4	5	6	7
BASIC	9.72%	4.99%	6.28%	5.40%	5.85%	5.91%
BATCHED	8.94%	4.78%	6.33%	5.44%	5.89%	5.66%
UNIFORM	9.32%	4.92%	5.79%	5.67%	5.65%	5.50%

Table 6.3: Error rates on SHUTTLE dataset

#Clusters	6	8	10	12	14	16
BASIC	3.70%	3.83%	4.87%	5.75%	5.95%	5.68%
BATCHED	3.57%	3.68%	4.49%	4.39%	5.04%	5.53%
UNIFORM	3.96%	5.21%	4.92%	4.83%	4.57%	5.43%

Table 6.4: Error rates on KDD dataset

For all datasets, BASIC and BATCHED significantly outperform UNIFORM. Take KDD dataset for example, we observe that with 10^5 query budget UNIFORM recovers 10 clusters while BASIC and BATCHED recover around 18 clusters. In order to recover 16 clusters, UNIFORM requires 200,000 queries per cluster, while BASIC and BATCHED only requires no more than 2,500 queries per cluster. Even on small datasets such as SHUTTLE, UNIFORM needs about twice amount of queries to recover all clusters compared with BASIC and BATCHED. We also observe that BATCHED performs slightly better than BASIC.

On synthetic datasets, one can see that higher collision probability p makes clustering more difficult for all algorithms. Given a query budget of 2×10^5, BASIC and BATCHED can recover about 70 clusters when $p = 0$ (no collision), but only about 60 clusters when $p = 0.3$.

We also observe that BATCHED performs better than BASIC in the KDD dataset in all settings, except the fixed-recovery setting for the SHUTTLE dataset, in which they have almost the same performance.

Center Approximation. We compare the approximation error of each algorithm to show the quality of returned approximate centers. We run all three algorithms in the fixed-recovery setting because we can only measure the error of the recovered clusters. We examine the median error among all recovered clusters and observe that the centroid approximation errors are indeed similar on all datasets when the same number of clusters is recovered. Detailed results are listed in Tables 6.2 to 6.4. We observe that all centroid approximation error rates are below 10%: the error rates are about 9% on the synthetic datasets and are about 5% on the two real-world datasets. There is no significant difference between the error rates of different algorithms; the maximum difference is no more than 2%. These consistent results indicate that the approximate centers computed by all algorithms are of good quality.

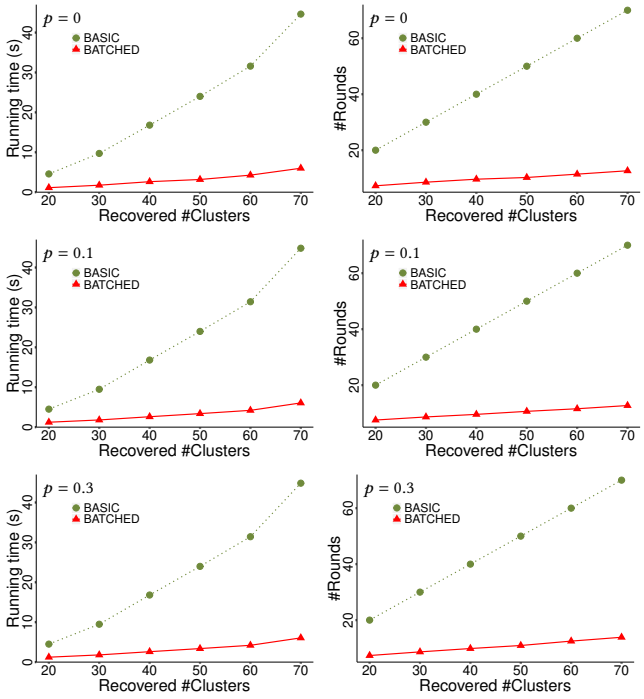

Figure 4: Running time and round usage on synthetic datasets

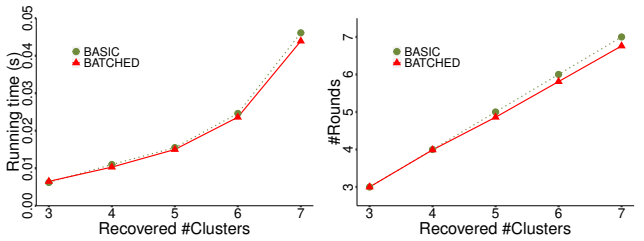

Figure 5: Running time and round usage on Shuttle dataset

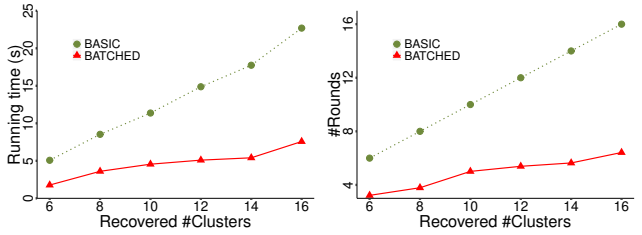

Figure 6: Running time and round usage on Kdd dataset

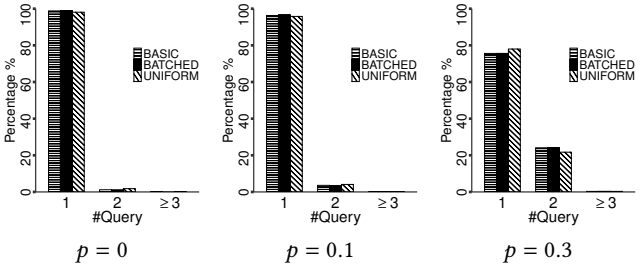

Figure 7: Number of same-cluster queries on synthetic datasets.

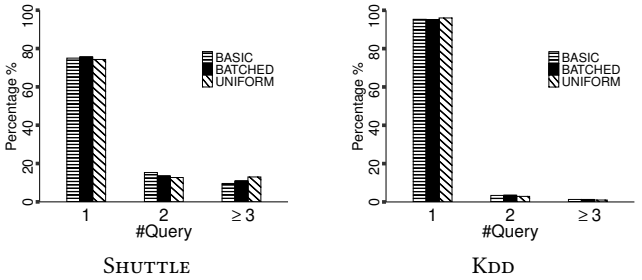

Figure 8: Number of same-cluster queries on real datasets.

Time and Rounds. We compare the running time and the number of rounds between Basic and Batched, with results presented in Figure 4 to Figure 6. Here we present the running time in the sampling stage, showing that the reduced round usage in Batched also leads to a reduced usage of sampling time in Batched. In terms of the time saving in querying stage, we have already discussed it in the query complexity part. The only exception is the Shuttle dataset, where the performance of Basic and Batched are almost the same; this may be because the number of clusters in Shuttle is too small (7 in total) for the algorithms to make any difference. We remark that fewer rounds implies a shorter waiting time for synchronization in the parallel learning setting and a smaller communication cost in the distributed learning setting.

Classification Using Approximate Centers. We now measure the quality of the approximate centers outputted by our algorithms for point classification using Heuristic 1. That is, after the algorithm terminates and outputs the set of approximate centers, we try to cluster new (unclassified) points using Heuristic 1 and count the number of same-cluster queries used for each new point. Since there is only a very small number of points in the dataset that have been sampled/classified during the run of Basic and Batched, we simply use the whole dataset as test points. This should not introduce much bias to our measurement.

Our results for the synthetic and the real-world datasets are plotted in Figures 7 and 8, respectively. We observe that for all three tested algorithms, for the majority of the points in the dataset, only *one* same-cluster query is needed for classification; in other words, we only need to find their nearest approximate centers for determining their cluster labels. Except for the Shuttle dataset for which about 10-15% points need at least three same-cluster queries, in all other datasets the fraction of points which need at least three queries is negligible. Overall, the performance of the three tested algorithms is comparable.

Summary. Briefly summarizing our experimental results, we observe that both Basic and Batched significantly outperform Uniform in terms of query complexities. All three algorithms have similar center approximation quality. Batched outperforms Basic by a large margin in both running time and round usage on the synthetic and Kdd datasets.

REFERENCES

[1] Nir Ailon, Anup Bhattacharya, and Ragesh Jaiswal. 2018. Approximate Correlation Clustering Using Same-Cluster Queries. In *LATIN*. 14–27.

[2] Nir Ailon, Anup Bhattacharya, Ragesh Jaiswal, and Amit Kumar. 2018. Approximate Clustering with Same-Cluster Queries. In *ITCS*. 40:1–40:21.

[3] David Arthur and Sergei Vassilvitskii. 2007. k-means++: the advantages of careful seeding. In *SODA*. 1027–1035.

[4] Hassan Ashtiani, Shrinu Kushagra, and Shai Ben-David. 2016. Clustering with Same-Cluster Queries. In *NIPS*. 3216–3224.

[5] Marco Bressan, Nicolò Cesa-Bianchi, Silvio Lattanzi, and Andrea Paudice. 2020. Exact Recovery of Mangled Clusters with Same-Cluster Queries. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin (Eds.).

[6] I (Eli) Chien, Chao Pan, and Olgica Milenkovic. 2018. Query K-means Clustering and the Double Dixie Cup Problem. In *NeurIPS*. 6650–6659.

[7] Donatella Firmani, Barna Saha, and Divesh Srivastava. 2016. Online Entity Resolution Using an Oracle. *PVLDB* 9, 5 (2016), 384–395.

[8] Buddhima Gamlath, Sangxia Huang, and Ola Svensson. 2018. Semi-Supervised Algorithms for Approximately Optimal and Accurate Clustering. In *ICALP*. 57:1–57:14.

[9] Wasim Huleihel, Arya Mazumdar, Muriel Médard, and Soumyabrata Pal. 2019. Same-Cluster Querying for Overlapping Clusters. In *NeurIPS*. 10485–10495.

[10] Mary Inaba, Naoki Katoh, and Hiroshi Imai. 1994. Applications of Weighted Voronoi Diagrams and Randomization to Variance-Based k-Clustering (Extended Abstract). In *SOCG*. 332–339.

[11] Ragesh Jaiswal, Amit Kumar, and Sandeep Sen. 2014. A Simple D^2-Sampling Based PTAS for k-Means and Other Clustering Problems. *Algorithmica* 70, 1 (2014), 22–46.

[12] Amit Kumar, Yogish Sabharwal, and Sandeep Sen. 2010. Linear-time approximation schemes for clustering problems in any dimensions. *J. ACM* 57, 2 (2010), 5:1–5:32.

[13] Arya Mazumdar and Barna Saha. 2017. Clustering with Noisy Queries. In *NIPS*. 5788–5799.

[14] Arya Mazumdar and Barna Saha. 2017. Query Complexity of Clustering with Side Information. In *NIPS*. 4682–4693.

[15] Barna Saha and Sanjay Subramanian. 2019. Correlation Clustering with Same-Cluster Queries Bounded by Optimal Cost. In *ESA*. 81:1–81:17.

[16] Vasilis Verroios and Hector Garcia-Molina. 2015. Entity Resolution with crowd errors. In *ICDE*. 219–230.

[17] Norases Vesdapunt, Kedar Bellare, and Nilesh N. Dalvi. 2014. Crowdsourcing Algorithms for Entity Resolution. *PVLDB* 7, 12 (2014), 1071–1082.

[18] Jiannan Wang, Tim Kraska, Michael J. Franklin, and Jianhua Feng. 2012. CrowdER: Crowdsourcing Entity Resolution. *PVLDB* 5, 11 (2012), 1483–1494.

[19] Jiannan Wang, Guoliang Li, Tim Kraska, Michael J. Franklin, and Jianhua Feng. 2013. Leveraging transitive relations for crowdsourced joins. In *SIGMOD*. 229–240.