Differentially Private Clustering in Data Streams

Alessandro Epasto [1] **Tamalika Mukherjee** [2] **Peilin Zhong** [1]

Abstract

Clustering problems (such as k-means and k-median) are fundamental unsupervised machine learning primitives. Recently, these problems have been subject to large interest in the privacy literature. All prior work on private clustering, however, has been devoted to the *offline* case where the entire dataset is known in advance. In this work, we focus on the more challenging private data stream setting where the aim is to design memory-efficient algorithms that process a large stream *incrementally* as points arrive in a private way. Our main contribution is to provide the first differentially private algorithms for k-means and k-median clustering in data streams. In particular, our algorithms are the first to guarantee differential privacy both in the continual release and in the one-shot setting while achieving space sublinear in the stream size. We complement our theoretical results with an empirical analysis of our algorithms on real data.

1. Introduction

Clustering is an essential primitive in unsupervised machine learning, and its geometric formulations, such as k-means and k-median, have been studied extensively, e.g., (Arya et al., 2001; Charikar et al., 2002; Har-Peled & Mazumdar, 2004; Chen, 2006; 2008; Awasthi et al., 2010; Ostrovsky et al., 2012; Li & Svensson, 2016; Ahmadian et al., 2020). In this paper, we focus on the study of clustering in the streaming model under the constraint of data privacy.

Differential privacy (DP) (Dwork et al., 2016) has become the de facto standard for preserving data privacy due to its compelling privacy guarantees and mathematically rigorous definition. There is a rich DP literature for clustering in the polynomial-time setting, e.g., (Nissim et al., 2007; Feldman et al., 2009; 2017; Gupta et al., 2010; Balcan et al., 2017; Huang & Liu, 2018; Nissim & Stemmer, 2018; Stemmer & Kaplan, 2018; Ghazi et al., 2020; Cohen et al., 2021) where the focus has been to improve the approximation ratios and achieve efficient algorithms in high-dimensional Euclidean space. More recent works have studied this problem in other models of computation, such as sublinear-time (Blocki et al., 2021) and massively parallel computing (MPC) (Cohen-Addad et al., 2022a;b). However, the study of DP clustering in the streaming model remains vastly unexplored.

1.1. Our Results

In this paper we address the problem of clustering in the streaming model in which the input $x_1, \ldots, x_T \in \mathbb{R}^d$ arrives in a stream. We the give the first pure DP k-means and k-median clustering algorithms that use space sublinear in the size in T for (1) *continual release setting:* where the algorithm must output a clustering at every timestamp $t \in [T]$, and (2) *one-shot setting:* where the algorithm must output a clustering at the end of the stream. As is standard in DP clustering literature, we assume Λ is an upper bound on the diameter of the space of input points.

In the following two theorems we assume we are given a non-DP algorithm in the offline setting that can compute a ρ-approximation to k-means (or k-median)—many such algorithms exists with constant approximation (e.g. (Ahmadian et al., 2020)).

Theorem 1.1. *There exists an ε-DP algorithm for k-means (or k-median) in the continual release model such that for every timestamp $t \in [T]$ it outputs a clustering with $\Theta(\rho) d^{O(1)}$-multiplicative error and $\tilde{O}(\frac{k\rho\Lambda^2}{\varepsilon} \cdot (d \log T)^{O(1)})$-additive error in $O(k \log^2(\Lambda) \log(k) poly(\log(T\Lambda k)))$ space.*[1]

Theorem 1.2. *There exists an ε-DP algorithm for k-means (or k-median) in the one-shot model such that it outputs a clustering with $\Theta(\rho) d^{O(1)}$-multiplicative error and $\tilde{O}(\frac{k\rho\Lambda^2}{\varepsilon} \cdot (d \log T)^{O(1)})$-additive error in*

*Equal contribution [1]Google Research NYC, USA [2]Purdue University, USA. Work partially done while a Student Researcher at Google. Correspondence to: Alessandro Epasto <aepasto@google.com>, Tamalika Mukherjee <tmukherj@purdue.edu>, Peilin Zhong <peilinz@google.com>.

Presented at the 2^{nd} Annual Workshop on Topology, Algebra, and Geometry in Machine Learning (TAG-ML) at the 40^{th} International Conference on Machine Learning, Honolulu, Hawaii, USA. 2023.

[1]We use the $\tilde{O}(\cdot)$ notation to neglect poly-logarithmic factors in (Λ, k, T).

$O(k\log^2(\Lambda)\log(k)poly(\log(T\Lambda k)))$ space at the end of the stream of length T.

We observe that in both settings the memory used is $\tilde{O}(k)$ (ignoring poly-logarithmic factors in (Λ, k, T)) thus matching the space requirements of non-DP streaming algorithms (Charikar et al., 2003).

1.2. Technical Overview

Our techniques apply to both k-means and k-median clustering, but we assume we are dealing with k-means for simplicity. Before discussing our algorithm in more detail, we first outline the challenges to designing a DP k-means clustering algorithm in the continual release setting.

Natural space-efficient approaches fail. A natural first approach towards this problem in the one-shot setting and one that was employed in the sublinear-time model (Blocki et al., 2021) is to maintain a random sample using the same space as our proposed algorithm, i.e., $\tilde{O}(k)$ and apply a state-of-the-art DP clustering algorithm on this sample at the end of the stream. One can easily show that this algorithm preserves DP and is as space efficient as our method. However, the accuracy of the resulting clustering achieved will be considerably worse than our proposed algorithm. In the worst case, this approach can lead to an additive error (ignoring k, Λ, d dependencies) of $O(\sqrt{T})$ (see Cohen-Addad et al. (2022b) for a detailed exposition). In contrast, our approach leads to an additive error of $O(\text{poly}(\log T))$. We demonstrate this experimentally by showing that our proposed algorithm outperforms the random sampling algorithm we use as a baseline.

Our Approach. For every timestamp $t \in [T]$, our algorithm for both continual release and one-shot settings can be split into two main steps — (1) Compute a weighted DP coreset $\mathcal{F}$ in an online fashion that satisfies a bicriteria approximation to k-means (see Theorem 1.3). (2) Compute a non-DP k-means ρ-approximation algorithm on $\mathcal{F}$ in a postprocessing step.

Theorem 1.3 (Bicriteria approximation). *There exists an ε-DP algorithm that for every timestamp $t \in [T]$, computes a weighted set of $O(k\log(k)\log^2(\Lambda)\log T)$ centers with $d^{O(1)}$-multiplicative error to the best k-means (or k-median) clustering and $\tilde{O}(\frac{k\rho\Lambda^2}{\varepsilon}\cdot(d\log T)^{O(1)})$-additive error in $O(k\log^2\Lambda\log(k)poly(\log(T\Lambda k)))$ space.*

Quadtrees and Heavy Hitters. A quadtree creates a nested series of grids that partitions $\mathbb{R}^d$ and can be used to embed input points into a Hierarchically Separated Tree (HST) metric, which often makes computing k-means cost simpler. We use this embedding to map every input point to the center of a grid (or cell) at every quadtree level. For a fixed level, our goal is to approximately choose the $O(k)$ cells that have the most points, i.e., we want to find the "heaviest" cells in a DP fashion and store them as candidate centers in set $\mathcal{F}$. We achieve this by hashing the cells into $O(k)$ substreams and running a continual release black-box DP heavy hitter algorithm on each hash substream. Since with large enough probability, the heavy cells will not collide, this achieves our goal. Note that since we need to do this over logarithmically many levels of the quadtree, we will end up with a bicriteria approximation.

We stress that we need to run a *continual release* black-box DP heavy hitter algorithm for both our continual release and one-shot setting clustering algorithms. This is because we need to assign x_t to a candidate center in $\mathcal{F}$ (obtained from computing the heavy-hitters) in an online fashion at every timestep $t \in [T]$ in both settings. The main difference in our algorithms for these two settings is that in the continual release setting we release the resulting weighted coreset consisting of candidate centers and their noisy weights at every timestep $t \in [T]$, while in the one-shot setting we release the weighted coreset at the end of the stream. Thus, we keep track of the noisy weights in the continual release setting via multiple instantiations of the binary mechanism (Dwork et al., 2010; Chan et al., 2011) while we can add Laplace noise to release the noisy weights at the end of the stream in the one-shot setting.

2. Preliminaries

An event E occurs *with high probability* if for any $c \geq 1$, there is an appropriate choice of constants for which E occurs with probability at least $1 - O(1/k^c)$ where k is the k-clustering input parameter.

Norms and heavy hitters. Let $p \geq 1$, the ℓ_p-norm of a vector $\mathbf{x} = (x_1, \ldots, x_t)$ is defined as $\|\mathbf{x}\|_p = (\sum_{i=1}^t |x_i|^p)^{1/p}$. Given a multiset $\mathcal{S}$, denote the frequency of an item x appearing in $\mathcal{S}$ as $f(x)$. We say that an item x is an α-heavy hitter (α-HH for short) if $f(x) \geq \alpha\|\mathcal{S}\|_1$.

Differential Privacy. Streams $\mathcal{S} = (x_1, \ldots, x_T)$ and $\mathcal{S}' = (x'_1, \ldots, x'_T)$ are *neighboring* if there exists at most one timestamp $t^* \in [T]$ for which $x_{t^*} \neq x'_{t^*}$ and $x_t = x'_t$ for all $t \neq t^*$.

Definition 2.1 (Differential privacy Dwork et al. (2016)). A randomized algorithm $\mathcal{A}$ is ε-DP if for every pair of neighboring datasets $D \sim D'$, and for all sets $\mathcal{S}$ of possible outputs, we have that $\Pr[\mathcal{A}(D) \in \mathcal{S}] \leq e^{\varepsilon}\Pr[\mathcal{A}(D') \in \mathcal{S}]$

Theorem 2.2 (Binary Mechanism BM Chan et al. (2011); Dwork et al. (2010)). *Let $\varepsilon \geq 0, \gamma \in (0, 0.5)$, there is an ε-DP algorithm for the sum of the stream in the continual release model. With probability $1-\gamma$, the additive error of the output for every timestamp $t \in [T]$ is always at most $O(\frac{1}{\varepsilon}\log^{2.5}(T)\log(\frac{1}{\gamma}))$ and uses $O(\log T)$ space.*

See (Dwork & Roth, 2014) for more foundational concepts in differential privacy.

Clustering. For points $x, y \in \mathbb{R}^d$, we let $d(x,y) = \|x - y\|_2$ be the Euclidean distance between x and y. Given a set $\mathcal{C}$, we define $d(x, \mathcal{C}) := \min_{c \in \mathcal{C}} d(x, c)$.

For a set of centers C, we define the cost of clustering for the set $\mathcal{S}$ wrt C as

$$cost(C, \mathcal{S}) = \sum_{x \in \mathcal{S}} d^z(x, C)$$

where $z = 1$ for k-median, and $z = 2$ for k-means.

Our goal in DP clustering is to produce a set of k centers $C_{\mathcal{S}}$ for input stream $\mathcal{S}$ such that (1) $C_{\mathcal{S}}$ is ε-DP wrt $\mathcal{S}$, and (2) $cost(C_{\mathcal{S}}, \mathcal{S}) \leq \alpha \cdot cost(C_{\mathcal{S}}^{opt}, \mathcal{S}) + \beta$.

3. Bicriteria Approximation in Continual Release Setting

We describe our algorithm in more detail here. We focus on the k-means problem in the sequel, however we stress that our techniques easily extend to the k-median problem and the algorithm and analysis are nearly identical. We refer to the Supplementary Materials for a full version of this paper.

Algorithm. Our main algorithm is given by Algorithm 1 which initializes $\log \Lambda$ parallel instances of randomly shifted quadtrees. At every timestep $t \in [T]$ the input point x_t is assigned to a cell in the $\log \Lambda$ many levels of every quadtree. For a fixed quadtree, the subroutine DPFindCenters (see Algorithm 2) is called on every level. The subroutine DPFindCenters returns a candidate set of centers $\hat{\mathcal{F}}_t$ which is first added to the current set of candidate set of centers $\mathcal{F}$, and x_t is then assigned to the nearest center in $\mathcal{F}$. Finally, the DP counts of all centers in $\mathcal{F}$ are updated via the Binary Mechanism.

The DPFindCenters subroutine (see Algorithm 2) finds the approximate heaviest $O(k)$ cells in a fixed level of a fixed quadtree. It achieves this by first hashing all the cells in that level to $w := O(k)$ many substreams (or buckets) $\mathcal{B}_j$ for all $j \in [w]$ and then runs a continual release α-heavy hitter algorithm DP-HH on each bucket.[2] We use the ℓ_1-heavy hitter algorithm from (Epasto et al., 2023) as DP-HH — it returns a set H of α-heavy hitters and their approximate counts $\hat{f}(\mathbf{c})$ for all $\mathbf{c} \in H$. Since we are storing all the cells marked as heavy hitters as candidate centers over at most T timesteps, we need to ensure that we do not store too many false positives, i.e., cells whose counts are much

[2] Notice that in the pseudo code Algorithm 2, $\perp$ represents an empty update that does not affect the counters of elements of the stream and is ignored. This is needed for technical reasons to ensure DP by avoiding the value of the hash affecting the number of events in the sub-streams.

Algorithm 1 DP Clustering Algorithm in Continual Release Setting

---

Require: Privacy parameter ε, Threshold parameter for heavy hitters α, Time bound T, Binary Mechanism BM, Continual Release DP-HH algorithm, Stream $\mathcal{S}$ of points $x_1, \ldots, x_T \in \mathbb{R}^d$

Ensure: Set of DP centers $\mathcal{F}$ and their noisy weights DP-Coreset at every timestep t

$\varepsilon' := \frac{\varepsilon}{\log^2 \Lambda \log k}$
Initialize hashmap DPCoreset to empty{used to store centers and noisy weights}
Initialize parallel quadtrees $Q_1, \ldots, Q_{\log(\Lambda)}$ as follows: Initialize each quadtree Q_q as $\mathcal{S}_1^{(q)}, \ldots, \mathcal{S}_{\log(\Lambda)}^{(q)}$ parallel streams or levels with the bottom stream/level having grid size $\Theta(1)$
Initialize $\mathcal{F} := \emptyset$
for $t = 1$ to T **do**
  for each $\mathcal{S}_\ell^{(q)}$ where $0 \leq \ell \leq \log(\Lambda)$ and $1 \leq q \leq \log(\Lambda)$ **do**
    Add x_t to $\mathcal{S}_\ell^{(q)}$
    $\hat{\mathcal{F}}_t = \mathsf{DPFindCenters}(\varepsilon', \mathcal{S}_\ell^{(\mathsf{q})})$
    $\mathcal{F} = \mathcal{F} \cup \hat{\mathcal{F}}_t$
    {add new centers to hashmap DPCoreset and initialize their DP weights}
    for $\mathbf{c} \in \hat{\mathcal{F}}_t - \mathcal{F}$ **do**
      Add $\mathbf{c}$ as a key to DPCoreset
      Initialize an instance of $\mathsf{BM}_{\mathbf{c}}(T, \varepsilon', 0)$ for $\mathsf{DPCoreset}(\mathbf{c})$
    end for
    if $\mathcal{F} \neq \emptyset$ **then**
      Let $\mathbf{c}^* := \text{argmin}_{\mathbf{c} \in \mathcal{F}} d(x_t, \mathbf{c})$ {assign x_t to the nearest center; if $\mathcal{F} = \emptyset$ then discard x_t}
      $\mathsf{DPCoreset}(\mathbf{c}^*) = \mathsf{BM}_{\mathbf{c}^*}(T, \varepsilon, 1)$
    end if
    for $\mathbf{c} \neq \mathbf{c}^*$ **do**
      $\mathsf{DPCoreset}(\mathbf{c}) = \mathsf{BM}_{\mathbf{c}}(T, \varepsilon, 0)$
    end for
  end for
  Output $\mathcal{F}$, DPCoreset
end for

---

smaller than $\alpha \|\mathcal{B}_j\|_1$. To address this challenge, we have an additional pruning step that eliminates any cell $\mathbf{c}$ whose approximate count is less than $\Theta(\alpha)\hat{T}_{\mathcal{B}_{h(\mathbf{c})}}$ where $\hat{T}_{\mathcal{B}_{h(\mathbf{c})}}$ denotes the DP size of the hash stream $\mathcal{B}_{h(\mathbf{c})}$ at timestep $t \in [T]$. We keep track of $\hat{T}_{\mathcal{B}_{h(\mathbf{c})}}$ via another instance of the Binary Mechanism. Finally, only the cells that pass this pruning step are added as candidate centers to the set $\hat{\mathcal{F}}_t$.

Theorem 3.1. *Let $\mathcal{S} := \{x_1, \ldots, x_T\}$ be the stream of input points in Euclidean space. For $t = 1, \ldots, T$, let $\mathcal{F}_t$ be the set of centers until time step t. Let $cost(\mathcal{F}, \mathcal{S}) :=$*

Algorithm 2 DPFindCenters

Require: Privacy parameter ε', Stream $\mathcal{S}_\ell$ with 2^ℓ cells (representing the ℓ-th level of quadtree instantiation), Binary Mechanism BM
Ensure: Set of candidate centers $\hat{\mathcal{F}}_t$ at every timestep $t \in [T]$

Initialize $\hat{\mathcal{F}}_t = \emptyset$
Let $w = O(k)$
Initialize $\hat{T}_{\mathcal{B}_1}, \ldots, \hat{T}_{\mathcal{B}_w}$ {DP Count for the size of hash bucket}
for $p = 1, \ldots, L$, where $L := O(\log k)$ run in parallel **do**
  Initialize hash function $h : [2^\ell] \to [w]$ s.t. $\forall \mathbf{c}, \forall j \in [w], \Pr[h(\mathbf{c}) = j] = \frac{1}{w}$
  Initialize empty hash streams $\mathcal{B}_1, \ldots, \mathcal{B}_w$
  for each cell $\mathbf{c}$ at level ℓ **do**
    Append $\mathbf{c}$ to $\mathcal{B}_{h(\mathbf{c})}$ and append $\perp$ to the end of every stream $\mathcal{B}_j$ such that $j \neq h(\mathbf{c})$.
    $\hat{T}_{\mathcal{B}_{h(\mathbf{c})}} = \mathsf{BM}_{h(\mathbf{c})}(T, \varepsilon', 1)$
    for $j \neq h(\mathbf{c})$ **do**
      $\hat{T}_{\mathcal{B}_j} = \mathsf{BM}_j(T, \varepsilon', 0)$
    end for
  end for
  for $j \in [w]$ **do**
    $\hat{f}, H \leftarrow$ DP-HH$(\varepsilon', \mathcal{B}_j)$ $\{\varepsilon' := \frac{\varepsilon}{\log^2 \Lambda \log k}\}$
    for $\mathbf{c} \in H$ **do**
      if $\hat{f}(\mathbf{c}) \geq \frac{\alpha}{1000} \cdot \hat{T}_{\mathcal{B}_{h(\mathbf{c})}}$ **then**
        Append $\mathbf{c}$ to $\hat{\mathcal{F}}_t$ as a center
      end if
    end for
  end for
end for
Return $\hat{\mathcal{F}}_t$

$\sum_{t=1}^{T} cost(\mathcal{F}_t)$ where $cost(\mathcal{F}_t) := \min_{f \in \mathcal{F}_t} dist^2(x_t, f)$. There exists an algorithm $\mathcal{A}$ (see Algorithm 1) that outputs a set of centers $\mathcal{F}$ and their corresponding weights DPCoreset at every timestep $t \in [T]$ such that

1. *(Privacy) $\mathcal{A}$ is 3ε-DP.*
2. *(Accuracy) With high probability,*
$$cost(\mathcal{F}, \mathcal{S}) \leq O(d^3) cost(C_{\mathcal{S}}^{opt}, \mathcal{S}) + \tilde{O}\left(\frac{d^2 \Lambda^2 k}{\varepsilon} \log^C (T \cdot k \cdot \Lambda)\right)$$
where $cost(C_{\mathcal{S}}^{opt}, \mathcal{S})$ is the optimal k-means cost for $\mathcal{S}$.
3. *(Space) $\mathcal{A}$ uses $O(k \log^2(\Lambda) \log(k) poly(\log(T \Lambda k)))$ space.*
4. *(Size of $\mathcal{F}$) $\mathcal{F}$ has at most $O(k \log(k) \log^2(\Lambda) \log T)$ centers.*

Privacy. Since we are outputting the center point of the cells marked as heavy hitters and their respective noisy counts, we only need to show that DP is maintained wrt these centers and noisy counts of centers and hash substreams. An input point is assigned to a specific cell for a specific level of the quadtree, and cells at the same level are disjoint. Since there are $\log \Lambda$ levels per quadtree, a point is a member of $\log \Lambda$ cells. Since there are $\log \Lambda \log k$ parallel processes, a single point participates in $\log^2 \Lambda \log k$ total calls to DP-HH. Note that we do not account for the $O(k)$ buckets that the cells are hashed into, as DP-HH is called on disjoint inputs for each bucket. Thus calling each DP-HH instance with a privacy budget of $\frac{\varepsilon}{\log^2 \Lambda \log k}$ preserves ε-DP. We use the Binary Mechanism to keep track of the size of each hash substream $\mathcal{B}_j \; \forall j \in [w]$. Since the input cells (and corresponding points within cells) are disjoint in each substream due to hashing, this preserves $\frac{\varepsilon}{\log^2 \Lambda \log k}$-DP which over $\log^2 \Lambda \log k$ parallel processes preserves ε-DP. Finally, we release the number of points per center via the Binary Mechanism where each point can only contribute to a single cell count which preserves ε-DP. Therefore by composition, we get 3ε-DP for the entire algorithm.

4. From Bicriteria Approximation to k-Clustering

Suppose we have a non-DP k-means algorithm $\mathcal{A}'$ that gives a ρ-approximation. We run $\mathcal{A}'$(DPCoreset) where DPCoreset is the output of Algorithm 1. Note that by postprocessing, this computation preserves privacy.

For simplicity we denote the centers and their corresponding noisy weights in DPCoreset as a tuple $(\mathcal{F}, \hat{w})$. Let $C_{\mathcal{F},\hat{w}}$ denote the k-clustering obtained as output from $\mathcal{A}'((\mathcal{F}, \hat{w}))$. We show that $cost(C_{\mathcal{F},\hat{w}}, \mathcal{S})$ is reasonably bounded by the optimal cost of clustering of $\mathcal{S}$ denoted as $cost(C_{\mathcal{S}}^{opt}, \mathcal{S})$.

Theorem 4.1. *Let $\mathcal{S} = \{x_1, \ldots, x_T\}$. Suppose $C_{\mathcal{S}}^{opt}$ is the optimal set of centers for $\mathcal{S}$. Then*
$$cost(C_{\mathcal{F},\hat{w}}, \mathcal{S}) \leq (2\rho + 1) O(d^3) cost(C_{\mathcal{S}}^{opt}, \mathcal{S}) + \tilde{O}(\frac{k \rho \Lambda^2}{\varepsilon} \cdot (d \log T)^{O(1)})$$

References

Ahmadian, S., Norouzi-Fard, A., Svensson, O., and Ward, J. Better guarantees for k-means and euclidean k-median by primal-dual algorithms. *SIAM Journal on Computing*, 49, 2020.

Arya, V., Garg, N., Khandekar, R., Meyerson, A., Munagala, K., and Pandit, V. Local search heuristic for k-median and facility location problems. In *STOC*, 2001.

Awasthi, P., Blum, A., and Sheffet, O. Stability yields a ptas for k-median and k-means clustering. In *FOCS*, 2010.

Balcan, M., Dick, T., Liang, Y., Mou, W., and Zhang, H. Differentially private clustering in high-dimensional euclidean spaces. In *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, volume 70 of *Proceedings of Machine Learning Research*, pp. 322–331. PMLR, 2017.

Blocki, J., Grigorescu, E., and Mukherjee, T. Differentially-private sublinear-time clustering. In *2021 IEEE International Symposium on Information Theory (ISIT)*, pp. 332–337. IEEE, 2021.

Chan, T. H., Shi, E., and Song, D. Private and continual release of statistics. *ACM Trans. Inf. Syst. Secur.*, 14(3):26:1–26:24, 2011. doi: 10.1145/2043621.2043626. URL https://doi.org/10.1145/2043621.2043626.

Charikar, M., Guha, S., Tardos, É., and Shmoys, D. B. A constant-factor approximation algorithm for the k-median problem. *Journal of Computer and System Sciences*, 65, 2002.

Charikar, M., O'Callaghan, L., and Panigrahy, R. Better streaming algorithms for clustering problems. In *Proceedings of the thirty-fifth annual ACM symposium on Theory of computing*, pp. 30–39, 2003.

Chen, K. On k-median clustering in high dimensions. In *SODA*, 2006.

Chen, K. A constant factor approximation algorithm for k-median clustering with outliers. In *SODA*, 2008.

Cohen, E., Kaplan, H., Mansour, Y., Stemmer, U., and Tsfadia, E. Differentially-private clustering of easy instances. In Meila, M. and Zhang, T. (eds.), *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pp. 2049–2059. PMLR, 18–24 Jul 2021.

Cohen-Addad, V., Epasto, A., Lattanzi, S., Mirrokni, V., Munoz Medina, A., Saulpic, D., Schwiegelshohn, C., and Vassilvitskii, S. Scalable differentially private clustering via hierarchically separated trees. In *KDD*, pp. 221–230, New York, NY, USA, 2022a. Association for Computing Machinery. ISBN 9781450393850.

Cohen-Addad, V., Epasto, A., Mirrokni, V., Narayanan, S., and Zhong, P. Near-optimal private and scalable k-clustering. In *Advances in Neural Information Processing Systems*, 2022b.

Dwork, C. and Roth, A. The algorithmic foundations of differential privacy. *Found. Trends Theor. Comput. Sci.*, 9(3-4):211–407, 2014. doi: 10.1561/0400000042. URL https://doi.org/10.1561/0400000042.

Dwork, C., Naor, M., Pitassi, T., and Rothblum, G. N. Differential privacy under continual observation. In Schulman, L. J. (ed.), *Proceedings of the 42nd ACM Symposium on Theory of Computing, STOC 2010, Cambridge, Massachusetts, USA, 5-8 June 2010*, pp. 715–724. ACM, 2010. doi: 10.1145/1806689.1806787. URL https://doi.org/10.1145/1806689.1806787.

Dwork, C., McSherry, F., Nissim, K., and Smith, A. D. Calibrating noise to sensitivity in private data analysis. *J. Priv. Confidentiality*, 7, 2016.

Epasto, A., Mao, J., Medina, A. M., Mirrokni, V., Vassilvitskii, S., and Zhong, P. Differentially private continual releases of streaming frequency moment estimations. *CoRR*, abs/2301.05605, 2023. doi: 10.48550/arXiv.2301.05605. URL https://doi.org/10.48550/arXiv.2301.05605.

Feldman, D., Fiat, A., Kaplan, H., and Nissim, K. Private coresets. In *Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009, Bethesda, MD, USA, May 31 - June 2, 2009*, pp. 361–370. ACM, 2009.

Feldman, D., Xiang, C., Zhu, R., and Rus, D. Coresets for differentially private k-means clustering and applications to privacy in mobile sensor networks. In *Proceedings of the 16th ACM/IEEE International Conference on Information Processing in Sensor Networks, IPSN 2017, Pittsburgh, PA, USA, April 18-21, 2017*, 2017.

Ghazi, B., Kumar, R., and Manurangsi, P. Differentially private clustering: Tight approximation ratios. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020.

Gupta, A., Ligett, K., McSherry, F., Roth, A., and Talwar, K. Differentially private combinatorial optimization. In *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2010, Austin, Texas, USA, January 17-19, 2010*, pp. 1106–1125. SIAM, 2010.

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

Huang, Z. and Liu, J. Optimal differentially private algorithms for k-means clustering. In den Bussche, J. V.

and Arenas, M. (eds.), *Proceedings of the 37th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, Houston, TX, USA, June 10-15, 2018*, pp. 395–408. ACM, 2018.

Li, S. and Svensson, O. Approximating k-median via pseudo-approximation. *SIAM Journal on Computing*, 45, 2016.

Nissim, K. and Stemmer, U. Clustering algorithms for the centralized and local models. In *Algorithmic Learning Theory, ALT 2018, 7-9 April 2018, Lanzarote, Canary Islands, Spain*, volume 83 of *Proceedings of Machine Learning Research*, pp. 619–653. PMLR, 2018.

Nissim, K., Raskhodnikova, S., and Smith, A. D. Smooth sensitivity and sampling in private data analysis. In Johnson, D. S. and Feige, U. (eds.), *Proceedings of the 39th Annual ACM Symposium on Theory of Computing, San Diego, California, USA, June 11-13, 2007*, pp. 75–84. ACM, 2007.

Ostrovsky, R., Rabani, Y., Schulman, L. J., and Swamy, C. The effectiveness of lloyd-type methods for the k-means problem. *Journal of the ACM (JACM)*, 59, 2012.

Stemmer, U. and Kaplan, H. Differentially private k-means with constant multiplicative error. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, 2018.