Settling Time vs. Accuracy Tradeoffs for Clustering Big Data

ANDREW DRAGANOV*, Aarhus University, Denmark
DAVID SAULPIC, CNRS & Université Paris Cité, France
CHRIS SCHWIEGELSHOHN, Aarhus University, Denmark

We study the theoretical and practical runtime limits of k-means and k-median clustering on large datasets. Since effectively all clustering methods are slower than the time it takes to read the dataset, the fastest approach is to quickly compress the data and perform the clustering on the compressed representation. Unfortunately, there is no universal best choice for compressing the number of points – while random sampling runs in sublinear time and coresets provide theoretical guarantees, the former does not enforce accuracy while the latter is too slow as the numbers of points and clusters grow. Indeed, it has been conjectured that any sensitivity-based coreset construction requires super-linear time in the dataset size.

We examine this relationship by first showing that there does exist an algorithm that obtains coresets via sensitivity sampling in effectively linear time – within log-factors of the time it takes to read the data. Any approach that significantly improves on this must then resort to practical heuristics, leading us to consider the spectrum of sampling strategies across both real and artificial datasets in the static and streaming settings. Through this, we show the conditions in which coresets are necessary for preserving cluster validity as well as the settings in which faster, cruder sampling strategies are sufficient. As a result, we provide a comprehensive theoretical and practical blueprint for effective clustering regardless of data size. Our code is publicly available at source and has scripts to recreate the experiments.

CCS Concepts: • **Theory of computation → Sketching and sampling**; **Unsupervised learning and clustering**; *Facility location and clustering*.

Additional Key Words and Phrases: k-means clustering, k-median clustering, coresets, nearly-linear time

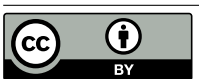
ACM Reference Format:
Andrew Draganov, David Saulpic, and Chris Schwiegelshohn. 2024. Settling Time vs. Accuracy Tradeoffs for Clustering Big Data. *Proc. ACM Manag. Data* 2, 3 (SIGMOD), Article 173 (June 2024), 25 pages. https://doi.org/10.1145/3654976

1 INTRODUCTION

The modern data analyst has no shortage of clustering algorithms to choose from but, given the ever-increasing size of relevant datasets, many are often too slow to be practically useful. This is particularly relevant for big-data pipelines, where clustering algorithms are commonly used for compression. The goal is to replace a very large dataset by a smaller, more manageable one for downstream tasks, with the hope it represents the original input well. Lloyd's algorithm [49] was introduced for precisely this reason and minimizes the *quantization error* – the sum of square distance from each input point to its representative in the compressed dataset. Arguably the most popular clustering algorithm, Lloyd's runs for multiple iterations until convergence with every

*All authors contributed equally to this research.

Authors' Contact Information: Andrew Draganov, draganovandrew@cs.au.dk, Aarhus University, Aarhus, Denmark; David Saulpic, david.saulpic@irif.fr, CNRS & Université Paris Cité, Paris, France; Chris Schwiegelshohn, cschwiegelshohn@gmail.com, Aarhus University, Aarhus, Denmark.



ACM 2836-6573/2024/6-ART173
https://doi.org/10.1145/3654976

iteration requiring $O(ndk)$ time, where n is the number of points, d is the number of features and k is the number of clusters – or the size of the targeted compression. For such applications, the number of points can easily be hundreds of millions and, since the quality of compression increases with k, standard objectives can have k in the thousands [4, 41]. In such settings, any $O(ndk)$ algorithm is prohibitively slow.

Examples like these have prompted the rise of big data algorithms that provide both theoretical and practical runtime improvements. The perspectives of theoretical soundness and practical efficacy are, however, often at odds with one another. On the one hand, theoretical guarantees provide assurance that the algorithm will work regardless of whatever unlucky inputs it receives. On the other hand, it may be difficult to convince oneself to implement the theoretically optimal algorithm when there are cruder methods that are faster to get running and perform well in practice.

Since datasets can be large in the number of points n and/or the number of features d, big-data methods must mitigate the effects of both. With respect to the feature space, the question is effectively closed as random projections are fast (running in effectively linear time), practical to implement [50], and provide tight guarantees on the embedding's size and quality. The outlook is less clear when reducing the number of points n, and there are two separate paradigms that each provide distinct advantages. On the one hand, we have uniform sampling, which runs in sublinear time but may miss important subsets of the data and therefore can only guarantee accuracy under certain strong assumptions on the data [45]. On the other hand, the most accurate sampling strategies provide the *strong coreset* guarantee, wherein the cost of any solution on the compressed data is within an ε-factor of that solution's cost on the original dataset [25].

Our contributions. We study both paradigms (uniform sampling and strong coresets) with respect to a classic problem – compression for the k-means and k-median objectives. Whereas uniform sampling provides optimal speed but no worst-case accuracy guarantee, all available coreset constructions have a running time of at least $\tilde{\Omega}(nd + nk)$ when yielding tight bounds on the minimum number of samples required for accurate compression.

It is easy to show that any algorithm that achieves a compression guarantee must read the entire dataset[1]. Thus a clear open question is what guarantees are achievable in linear or nearly-linear time. Indeed, currently available fast sampling algorithms for clustering [5, 6] cannot achieve strong coreset guarantees. Recently, [31] proposed a method for strong coresets that runs in time $\tilde{O}(nd + nk)$ and conjectured this to be optimal for k-median and k-means.

While this bound is effectively optimal for small values of k, there are many applications such as computer vision [34] or algorithmic fairness [18] where the number of clusters can be larger than the number of features by several orders of magnitude. In such settings, the question of time-optimal coresets remains open. Since the issue of determining a coreset of optimal size has recently been closed [25, 28, 44], this is arguably the main open problem in coreset research for center-based clustering. We resolve this by showing that there exists an easy-to-implement algorithm that constructs coresets in $\tilde{O}(nd)$ time – only logarithmic factors away from the time it takes to read in the dataset.

Nonetheless, this does not fully illuminate the landscape among sampling algorithms for clustering in practice. Although our algorithm achieves both an optimal runtime and an optimal compression, it is certainly possible that other, cruder methods may be just as viable for all practical purposes. We state this formally in the following question: *When are optimal k-means and k-median coresets necessary and what is the practical tradeoff between coreset speed and accuracy?*

[1]A simple example is to consider a data matrix in $\mathbb{R}^{n\times d}$ with one single large entry and the remaining entries being very small. The algorithm must find this large entry to yield a good clustering.

To answer this, we perform a thorough comparison across the full span of sampling algorithms that are faster than our proposed method. Through this we verify that, while these faster methods are sufficiently accurate on many real-world datasets, there exist data distributions that cause catastrophic failure for each of them. Indeed, these cases can only be avoided with a strong-coreset method. Thus, while many practical settings do not require the full coreset guarantee, one cannot cut corners if one wants to be confident in their compression. We verify that this extends to the streaming paradigm and applies to downstream clustering approaches.

In summary, our contributions are as follows:

- We show that one can obtain strong coresets for k-means and k-median in $\tilde{O}(nd)$ time. This resolves a conjecture on the necessary runtime for k-means coresets [31] and is theoretically optimal up to log-factors.
- Through a comprehensive analysis across datasets, tasks, and streaming/non-streaming paradigms, we verify that there is a necessary tradeoff between speed and accuracy among the linear- and sublinear-time sampling methods. This gives the clustering practitioner a blueprint on when to use each compression algorithm for effective clustering results in the fastest possible time.

2 PRELIMINARIES AND RELATED WORK

2.1 On Sampling Strategies.

As discussed, we focus our study on linear- and sublinear-time sampling strategies. Generally speaking, we consider compression algorithms through the lens of three requirements:

- Finding the compression should not take much longer than the time to read in the dataset.
- The size of the compressed data should not depend on the size of the original dataset.
- Candidate solutions on the compressed data are provably good on the original data.

If these requirements are satisfied then, when analyzing runtimes on large datasets, it is always preferable to compress the dataset and then perform the task in question on the compressed representation.

Specifically, given a dataset $P \in \mathbb{R}^{n\times d}$, we concern ourselves with sampling $\Omega \in \mathbb{R}^{m\times d} \subset P$ (such that $m \ll n$) along with a weight vector $w \in \mathbb{R}^m$. The goal is then that for any candidate solution C, Ω provides us with an idea of the solution's quality with respect to the original dataset, i.e. $\sum_{p\in\Omega} w_p \operatorname{cost}(p, C) \approx \sum_{p\in P} \operatorname{cost}(p, C)$ for a problem-specific cost function.

The quickest sampling strategy, running in sublinear time, is uniform sampling. It is clear, however, that this cannot provide any cost-preservation guarantee as missing a single extreme outlier will cause the sampling strategy to fail. Thus, any approach that outperforms uniform sampling must read in the entire dataset and therefore run in at least linear time[2]. Among these more sophisticated sampling strategies, coresets offer the strongest compression guarantee:

Definition 2.1. A *strong ε-coreset* is a pair (Ω, w) such that for *any* candidate solution C,

$$\sum_{p\in\Omega} w_p \operatorname{cost}(p, C) \in (1 \pm \varepsilon) \operatorname{cost}(P, C).$$

Going forward, we will discuss this in the context of the k-median and k-means cost functions: for dataset $P \in \mathbb{R}^d$ with weights $w : P \to \mathbb{R}^+$, and any k-tuple C in $\mathbb{R}^d$,

$$\operatorname{cost}_z(P, C) := \sum_{p\in P} w_p \operatorname{dist}^z(p, C),$$

[2]Indeed, sublinear algorithms always require some assumption on the input to provide guarantees, see [10, 30, 45, 51].

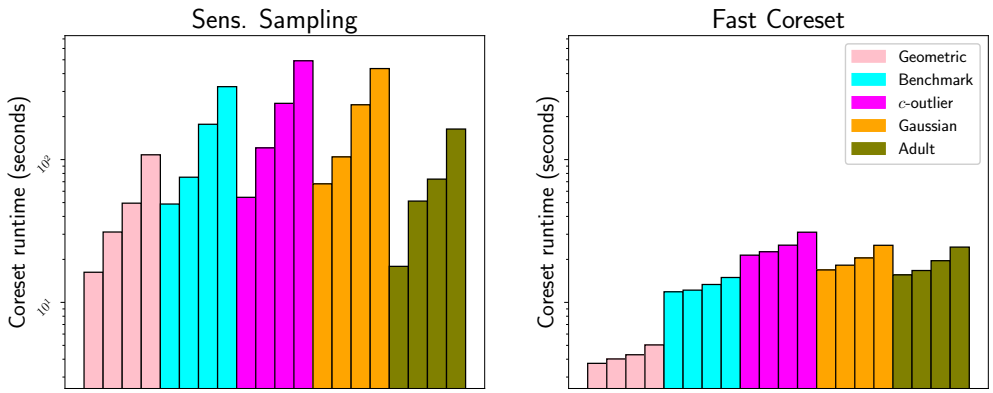


Fig. 1. Mean runtime over five runs as we vary k for sensitivity sampling and Fast-Coresets. Bars are $k = 50, 100, 200, 400$; y-axis is log-scale.

with $z = 1$ for k-median and $z = 2$ for k-means. We use OPT to denote $\min_C \text{cost}_z(P, C)$ and will denote an α-approximation as any candidate solution C such that $\text{cost}(C) \leq \alpha \cdot \text{OPT}$.

Recently, sampling with respect to sensitivity values has grown to prominence due to its simple-to-obtain coreset guarantee. True sensitivity values are defined as $\sup_C \frac{\text{dist}^z(p,C)}{\text{cost}_z(P,C)}$, where the supremum is taken over all possible solutions C. Intuitively, this is a measure of the maximum impact a point can have on a solution and is difficult to evaluate directly. Thus, the approximate sensitivity-sampling algorithm we consider is the following (as introduced in [37]). Given an α-approximate solution C to a clustering problem, importance scores are defined as

$$\sigma_C(p) = \alpha \cdot \left(\frac{\text{cost}(p, C)}{\text{cost}(C_p, C)} + \frac{1}{|C_p|} \right), \tag{1}$$

where C_p is the cluster that p belongs to. This is always an upper-bound on the sensitivity values [37]. In essence, sampling enough points proportionately to these values guarantees an accurate compression. The following paragraph discusses how the points must be re-weighted to guarantee the coreset property:

The coreset Ω consists of m points sampled proportionate to σ with weights defined as follows: for any sampled point p, define $w_p := \frac{1}{\Pr[p \in \Omega]} = \frac{\sum_{p'} \sigma_C(p')}{m \cdot \sigma_C(p)}$. The weights ensure that the cost estimator is unbiaised: in expectation, for any solution C, the cost evaluated on the sample should be equal to the original cost. It was shown in [42] that, when C is a $O(1)$-approximation, sampling $m = \tilde{O}\left(k\varepsilon^{-2z-2}\right)$ many points was enough to ensure concentration, namely, Ω is a coreset with probability at least $2/3$ – the conventional success probability for Monte-Carlo algorithms.

To perform this algorithm, the bottleneck in the running time lies in computing the solution C as well as then obtaining costs of every point to its assigned center in C. This takes $\tilde{O}(nk + nd)$ time when using a bicriteria approximation algorithm[3] such as the standard k-means++ algorithm [2] combined with dimension reduction techniques (see for example [9, 19, 50]). This is precisely what was conjectured as the necessary runtime for obtaining k-means and k-median coresets, as merely assigning points to their centers from the bicriteria seems to require $\Omega(nk)$ running time [31].

[3]For k-means an (α, β) bicriteria approximation is an algorithm that computes a solution C satisfying $\text{cost}(P, C) \leq \alpha \cdot \text{OPT}$ and $|C| \leq \beta \cdot k$.

2.2 Other Coreset Strategies

Many of the advancements regarding coresets have sought the smallest coreset possible across metric spaces and downstream objectives. The most prominent examples are in Euclidean space [7, 16, 27, 40, 42], with much of the focus on obtaining the optimal size in the k-means and k-median setting. Recently, a lower bound [44] showed that the group sampling algorithm developed in [25, 27, 28] is optimal.

Although optimal coresets have size $\tilde{O}(k \cdot \varepsilon^{-2} \min(k^{z/(z+2)}, \varepsilon^{-z}))$ [28] and are theoretically smaller than those obtained by sensitivity sampling, the experiments of [57] showed that the latter is always more efficient in practice.

In terms of other linear-time methods with sensitivity sampling, we are only aware of the lightweight coresets approach [6], wherein one samples with respect to the candidate solution $C = \{\mu\}$, i.e. the mean of the data set, instead of a candidate solution with k centers. This runs in $O(nd)$ time but provides a weaker guarantee – one incurs an additive error of $\varepsilon \cdot \text{cost}(P, \{\mu\})$. We note that this can be generalized to performing sensitivity sampling using a C that has fewer than k centers. We discuss this in more depth in Section 5.2.

Lastly, the BICO coreset algorithm [38] utilizes the SIGMOD test-of-time winning BIRCH [58] clustering method to obtain k-means coresets while the Streamkm++ method [1] uses k-means++ [2] to obtain a coreset whose size depends on n and is exponential in d. While both were developed to perform quickly in a data-stream, we show in Section 5 that they do not provide strong coreset guarantees in practice for reasonable coreset sizes.

All efficient coreset constructions are probabilistic, making coresets difficult to evaluate. For example, it is co-NP-hard to check whether a candidate compression is a weak coreset [57][4]. Therefore, although coreset algorithms succeed with some high probability, it is unclear how to computationally verify this. We refer to [57] for further discussion on this topic and discuss our evaluation metrics in Section 5.

2.3 Coresets for Database Applications

MapReduce[5] is one of the most popular architectures for large scale data analysis (see for example [17, 29, 33] and references therein for MapReduce algorithms for clustering). Within this context, strong coresets are 'embarrassingly parallel' and have a natural synergy with the MapReduce framework due to the following two properties. First, if two coresets are computed on subsets of data then their union is a coreset for the union of the data subsets. Second, many coreset constructions produce a compression with size completely independent of n, allowing the coreset to be stored in memory-constrained environments.

Using these two properties, one can compress the data in a single round of MapReduce as follows [36]. The data is partitioned randomly among the m entities, each of which computes a coreset and sends it to a central host. By the first property, the union of these coresets is a coreset for the full dataset. Thus, the host now possesses a small summary of the full dataset with strong coreset guarantees. By the second property, this summary's size does not depend on the size of the original dataset – in particular, the total size of the messages received by the host is independent of n. Lastly, by the coreset guarantee, any solution that is good w.r.t. the summary is good w.r.t. the original data (and vice versa).

[4]A weak coreset guarantee only requires that a $(1+\varepsilon)$ approximation computed on the coreset yields a $(1+\varepsilon)$ on the entire point set.

[5]The MapReduce computation model is the following: there are several computation entities, each with memory constraints that prevent them from holding the entire dataset. The costly operation is in exchanging data between those entities, implying that the complexity is measured in terms of rounds of communication and size of the memory exchanged.

This idea allows one to compute an approximate solution to k-means and k-median, and do so efficiently, in MapReduce: each computation entity needs to communicate merely $O(k)$ bits, where k is the number of clusters. The computational burden can therefore be completely parallelized up to the time required to compute a coreset in each entity – precisely the focus of this paper. We explore similar aggregation strategies experimentally in Section 5.4.

2.4 Quadtree Embeddings

A common technique for designing fast algorithms is to embed the Euclidean space into a tree metric using *quadtree embeddings*. The central idea is that any hypercube in the input space can be split into 2^d sub-cubes. We can represent this in a tree-structure, where each sub-cube has the original hypercube as its parent. If we apply a random shift to all the points in the original hypercube and appropriately set the weight of each branch, then the expected distance in the tree preserves the distance between points in different sub-cubes within an $O(\sqrt{d}\log\Delta)$ factor. Here, Δ is the *spread* of the point set and is equal to the ratio of the largest distance over the smallest non-zero distance. Given this context, we now introduce quadtree embeddings more formally:

Formal Overview. Given a set of points in $\mathbb{R}^{n\times d}$, we want to return a tree-structure that roughly preserves their pairwise distances. To do this, our first step is to obtain a box enclosing all input points, centered at zero, with all side lengths equal to Δ. This can be done as follows: select an arbitrary input point, and translate the dataset so that this point is at the origin. Then, using $O(nd)$ time, set Δ to be the maximum distance from any point to the origin. Note that, up to rescaling the points so that the smallest distance equals 1, this is equivalent to the *spread* as described in the previous paragraph.

Having obtained this box, add a shift s (picked uniformly at random in $[0,\Delta]$) to all the points' coordinates so that the input is now in the box $[-2\Delta, 2\Delta]^d$. This transformation does not change any distances and therefore preserves the k-median and k-means costs. The i-th level of the tree (for $i \in \{0, ..., \log\Delta\}$) is constructed by centering a grid of side length $2^{-i}\cdot 2\Delta$ at 0, making each grid-cell a node in the tree. The parent of a cell c is simply the cell at level $i-1$ that contains c, and the distance between c and its parent is set to $2^{-i}\cdot 2\Delta\cdot\sqrt{d}$ (the length of the hypercube's diagonal). This embedding takes $O(nd\log\Delta)$ time to construct, where $\log\Delta$ is the depth of the tree[6]. The linearity in the $\log\Delta$ term comes from the fact that this is the maximum depth of the tree.

The distortion of this embedding is at most $O(d\log\Delta)$, as stated in the following lemma:

Lemma 2.2 (Lemma 11.9 in [39])**.** *The distances in the tree metric d_T satisfy $\forall p,q, \|p-q\| \le \mathbb{E}[d_T(p,q)] \le O(d\log\Delta)\|p-q\|$, where the expectation is taken over the random shift s of the decomposition.*

A simple proof (and further intuition on quadtree embeddings) can be found in [39]. The result follows from combining linearity of expectation and the fact that two points p and q are separated at level i with probability at most $\sqrt{d}\|p-q\|\frac{2^i}{\Delta}$ (as in the proof of Lemma 4.3).

3 FAST-CORESETS

Technical Preview. We start by giving an overview of the arguments in this section.

There exists a strong relationship between computing coresets and approximate solutions – one can quickly find a coreset given a reasonable solution and vice-versa. Thus, the general blueprint is as follows: we very quickly find a rough solution which, in turn, facilitates finding a coreset that approximates all solutions. Importantly, the coreset size depends linearly on the quality of the

[6]Crucially, note that our Algorithm 2 does not need to build the full tree. Instead, we only require $\log\log\Delta$ levels of it, resulting in our improved complexity.

Algorithm 1 Fast-Coreset(P, k, ε, m)

1: **Input:** data P, number of clusters k, precision ε and target size m
2: Use a Johnson-Lindenstrauss embedding to embed $\tilde{P}$ of P into $\tilde{d} = O(\log k)$ dimensions
3: Find approx. solution $\tilde{C} = \{\tilde{c}_1, ..., \tilde{c}_k\}$ on $\tilde{P}$ and assignment $\tilde{\sigma} : \tilde{P} \to \tilde{C}$ by Fast-kmeans++ .
4: Let $C_i = \tilde{\sigma}^{-1}(c_i)$. Compute the 1-median (or 1-mean) c_i of each C_i in $\mathbb{R}^d$.
5: For each point $p \in C_i$, define $s(p) = \frac{\text{dist}^z(p,c_i)}{\text{cost}(C_i,c_i)} + \frac{1}{|C_i|}$.
6: Compute a set Ω of m points randomly sampled from P proportionate to s.
7: For each C_i, define $|\hat{C}_i|$ the estimated weight of C_i by Ω, namely $|\hat{C}_i| := \sum_{p \in C_i \cap \Omega} \frac{\sum_{p' \in P} s(p')}{s(p)m}$.
8: **Output:** the coreset Ω, with weights $w(p) = \frac{\sum_{p' \in P} s(p')}{s(p)m} \left((1+\varepsilon)|C_i| - |\hat{C}_i| \right)$

rough solution. Put simply, the coreset can compensate for a bad initial solution by oversampling. Our primary focus is therefore on finding a sufficiently good coarse solution quickly since, once this has been done, sampling the coreset requires linear-time in n. Our theoretical contribution shows how one can find this solution on Euclidean data using dimension reduction and quadtree embeddings.

Formal Results. In this section, we first combine two existing results to produce a strong coreset in time $\tilde{O}(nd \log \Delta)$, where Δ is the spread of the input. We show afterwards how to reduce the dependency in Δ to $\log \log \Delta$, giving the desired nearly-linear runtime.

Our method is based on the following observations about the group sampling [25] and sensitivity sampling [37] coreset construction algorithms. Both start by computing a solution C. When C is a c-approximation, they compute a coreset with distortion $1 \pm c\varepsilon$ of size $\tilde{O}\left(k\varepsilon^{-z-2}\right)$ and $\tilde{O}\left(k\varepsilon^{-2z-2}\right)$, respectively. Hence, by rescaling ε, they provide an ε-coreset with size $\tilde{O}\left(k(\varepsilon/c)^{-z-2}\right)$ and $\tilde{O}\left(k(\varepsilon/c)^{-2z-2}\right)$. This leads to the following fact:

Fact 3.1. *[See Theorem 1 in [25] and 4.8 in [37][7]] Let C be an $O\left(\log^{O(1)} k\right)$ approximation to k-median or k-means. Then group (resp. sensitivity) sampling using solution C computes a coreset of size $\tilde{O}\left(k\varepsilon^{-z-2}\right)$ (resp. $\tilde{O}\left(k\varepsilon^{-2z-2}\right)$) in time $\tilde{O}(nd)$.*

To turn Fact 3.1 into an algorithm, we use the quadtree-based Fast-kmeans++ approximation algorithm from [23], which has two key properties:

1 Fast-kmeans++ runs in $\tilde{O}(nd \log \Delta)$ time (Corollary 4.3 in [23]), and
2 Fast-kmeans++ computes an assignment from input points to centers that is an $O(d^z \log k)$ approximation to k-median ($z = 1$) and k-means ($z = 2$) (see Lemma 3.1 in [23] for $z = 2$ and the discussion above that lemma for $z = 1$). Applying dimension reduction techniques [50], the dimension d may be replaced by a $\log k$ in time $\tilde{O}(nd)$. This results in a $O\left(\log^{z+1} k\right)$ approximation.

The second property is crucial for us: the algorithm does not only compute centers, but also assignments in $\tilde{O}(nd \log \Delta)$ time. Thus, it satisfies the requirements of Fact 3.1 as a sufficiently good initial solution. We describe how to combine Fast-kmeans++ with sensitivity sampling in Algorithm 1 and prove in Section 8.1 that this computes an ε-coreset in time $\tilde{O}(nd \log \Delta)$:

Corollary 3.2. *Algorithm 1 runs in time $\widetilde{O}(nd \log \Delta)$ and computes an ε-coreset for k-means.*

[7]The references use an $O(1)$-approximation, but the proofs work with an $O\left(\log^{O(1)} k\right)$ approximation, with an added $O\left(\log^{O(1)} k\right)$ factor in the coreset size.

Value of r	20	30	40	50
Runtime	13.5 ± 0.16	14.2 ± 0.33	15.5 ± 0.02	16.2 ± 0.16

Table 1. Mean runtime in seconds for Fast-kmeans++ as a function of $r \sim \log \Delta$, taken over five runs.

Furthermore, we generalize Algorithm 1 to other fast k-median approaches in Section 8.4.

Thus, one can combine existing results to obtain an ε-coreset without an $\widetilde{O}(nk)$ time-dependency. However, this has only replaced the $\widetilde{O}(nd + nk)$ runtime by $\widetilde{O}(nd \log \Delta)$. Indeed, the spirit of the issue remains – this is not near-linear in the input size.

We verify that $\log \Delta$ is on the same order as n by devising a dataset that has $n - n'$ points uniformly in the $[-1, 1]^2$ square. Then, for $r \in \mathbb{Z}^+$, we produce a sequence of points at $(0, 1), (0, 0.5), \cdots, (0, 0.5^r)$ and copy this sequence n'/r times, each time with a different x coordinate. The result is a dataset of size n where $\log \Delta$ grows linearly with $r \in o(n)$. The resulting linear time-dependency can be seen in Table 1.

4 REDUCING THE IMPACT OF THE SPREAD

We now show how one can replace the linear time-dependency on $\log \Delta$ with a logarithmic one (i.e., $\log \Delta \to \log(\log \Delta)$).

Overview of the Approach. Without loss of generality, we assume that the smallest pairwise distance is at least 1, and Δ is a (known) upper-bound on the diameter of the input. To remove the $\log \Delta$ dependency, we proceed by producing a substitute dataset P' that has a lower-bound on its minimum distance and an upper-bound on its maximum distance. We then show that, with overwhelming likelihood, reasonable solutions on P' have the same cost as solutions on P up to an additive error.

In order to produce the substitute dataset P', we first find a crude upper-bound on the cost of the optimal solution to the clustering task on P. We then create a grid such that, for every cluster in the optimal solution, the cluster is entirely contained in a grid cell with overwhelming likelihood – in some sense, points in different grid cell do not interact. We can then freely move these grid cells around without affecting the cost of the solution, as long as they stay far enough away so that points in different cells still do not interact. Thus, we can constrain the diameter of P' to be small with respect to the quality of the approximate solution. We show later how to constrain the minimum distance to be comparable to the quality of this approximate solution as well.

We will focus this section on the simpler k-median problem but show how to reduce k-means to this case in Section 8.2.

4.1 Computing a crude upper-bound

As described, we start by computing an approximate solution U such that $\text{OPT} \le U \le \text{poly}(n) \cdot \text{OPT}$. For this, the first step is to embed the input into a quadtree. This embedding has two key properties. First, distances are preserved up to a multiplicative factor $O(d \log \Delta)$, and therefore the k-median cost is preserved up to this factor as well. Second, the metric is a *hierarchically separated tree*: it can be represented with a tree, where points of P are the leafs. The distance between two points is then given by the depth of their lowest common ancestor – if it is at depth ℓ, their distance is $\sqrt{d}\Delta 2^{-\ell}$. Our first lemma shows that finding the first level of the tree for which the input lies in $k + 1$ disjoint subtrees provides us with the desired approximation.

Lemma 4.1. *[Proof in Section 8.3] Let ℓ be the first level of the tree with at least $k+1$ non-empty subtrees. Then, $\sqrt{d}2^{-\ell+1} \cdot \Delta \leq OPT_T \leq n \cdot \sqrt{d}2^{-\ell+4} \cdot \Delta$, where OPT_T is the optimal k-median solution in the tree metric.*

We prove this in Section 8.3. A direct consequence is that the first level of the tree for which at least $k+1$ cells are non empty allows us to compute an $O(n)$-approximation for k-median on the tree metric. Since the tree metric approximates the oringial Euclidean metric up to $O(d \log \Delta)$, this is therefore an $O(nd \log \Delta)$-approximation to k-median in the Euclidean space.

To turn this observation into an algorithm, one needs to count the number of non-empty cells at a given level ℓ: for each point, we identify the cell that contains it using modulo operations. Furthermore, we count the number of distinct non-empty cells using a standard dictionary data structure. This is done in time $\tilde{O}(nd)$, and pseudo-code is given Algorithm 2. Using a binary search on the $O(\log \Delta)$ many levels then gives the following result:

Lemma 4.2. *[Proof in Section 8.3] There is an algorithm running in time $\tilde{O}(nd \log\log \Delta)$ that computes an $O(nd \log \Delta)$-approximation to k-median, and $O(n^3 d^2 \log^2 \Delta)$-approximation to k-means.*

Given this crude approximate solution, it remains to create a substitute dataset P' that satisfies two requirements:

1 First, P' must have spread linear in the quality of the approximate solution. If this holds, Algorithm 1 on P' will take $\tilde{O}(nd \log\log \Delta)$ time.
2 Second, any reasonable solution on P' should be roughly equivalent to a corresponding solution on P. This would mean that running Algorithm 1 on P' gives us a valid coreset for P.

4.2 From Approximate Solution to Reduced Spread

Let U be an upper-bound on the optimal cost, computed via Lemma 4.2. We place a grid with side length $r := \sqrt{d}n^2 \cdot U$, centered at a random point in $\{0, ..., r\}^d$. The following lemma ensures that with high probability (over the randomness of the center of the grid), no cluster of the optimal solution is spread on several grid cells.

Lemma 4.3. *The probability over the center's location that two points p and q are in different grid cells is at most $\frac{\|p-q\|}{n^2 U}$*

Proof. We first bound the probability that there is a grid line along the i-th dimension between p and q. Let p_i, q_i be the i-th coordinate of p and q, assume $p_i \leq q_i$ and let $\ell \in \mathbb{Z}$ such that $p_i - \ell r \in [0, r)$. Then, p and q are separated along dimension i if and only if the i-th coordinate of the center is in $[p_i - \ell r, q_i - \ell r]$. This happens with probability $|p_i - q_i|/r$. Finally, a union-bound over all coordinates shows that p and q are in different grid cells with probability at most $\sum |p_i - q_i|/r \leq \sqrt{d}\|p - q\|/r$. □

Since U is larger than the distance between any input point and its center in the optimal solution, a union-bound ensures that with probability $1 - 1/n$, no cluster of this solution is split among different cells. In particular, there are at most k-non empty cells which we can identify using a dictionary. We call these "boxes".Each box j has a middle point, which we call m_j.

From this input, we build a new set of points P' as follows. For each dimension $i \in \{1, ..., d\}$, sort the k boxes according to their value on dimension i. Then, for each $j \in \{1, ..., k\}$, let m_j^i be the i-th coordinate of the middle-point of the j-th box. If $m_{j+1}^i - m_j^i \geq 2r$, then for all boxes j' with $j' > j$, shift the points of j' by $m_{j+1}^i - m_j^i - 2r$ in the i-th dimension. This can be implemented in near-linear time, as described in Algorithm 3 (presented in Section 8). In essence, we take boxes that

are far apart and bring them closer together. The dataset P' obtained after these transformations has the following properties:

Proposition 4.4. *Let P' be the dataset produced by Algorithm 3. It holds that:*

1. *in P', the diameter of the input is $O(dn^2 \cdot U \cdot k)$ with probability $1 - 1/n$ over the randomness of the grid, and*
2. *two boxes that are adjacent (respectively non-adjacent) in P are still adjacent (resp. non-adjacent) in P'.*

Proof. By construction, the max distance between the middle-points of any two boxes in P' is $2r = 2\sqrt{d}n^2 \cdot U$. Lemma 4.3 ensures that, with probability $1 - 1/n^2$, any two points in the same cluster of the optimal solution (e.g., at distance less than $\text{OPT} \leq U$ from each other) are in the same box. Therefore, a union-bound ensures that there are at most k boxes. It then follows that the total distance along a coordinate is at most $2kr$, and the diameter of the whole point set is $\sqrt{d} \cdot 2kr$.

If two boxes were adjacent in P, then along any dimension their middle-points have distance at most r. Thus, if one of the adjacent boxes moves along any dimension, the other must as well and they will remain adjacent. If they are not adjacent, there is at least one dimension where their middle-points are at distance at least $2r$ from one another. Along each such dimension, their shift cannot bring them closer than to within $2r$ of one another and they will stay non-adjacent.

□

The first property allows us to reduce the spread to $(nd \log \Delta)^{O(1)}$. Indeed, one can round the coordinates of every point in P' to the closest multiple of $g := \frac{U}{n^4 d^2 \log \Delta}$. Combined with the diameter reduction, this ensures that the spread of the dataset obtained is at most $(nd \log \Delta)^{O(1)}$. Furthermore, the second property of Proposition 4.4 combined with the choice of g ensures that the cost of any reasonable solution is the same before and after the transformation, as stated in the following lemma:

Lemma 4.5. *Let P' be the outcome of the diameter reduction and rounding steps. With probability $1 - 1/n$ over the randomness of the grid, P' has spread $(nd \log \Delta)^{O(1)}$.*

Suppose U is such that $\text{OPT} \leq U \leq dn^2 \text{OPT}$, and let $\mathcal{S}'$ be a solution for k-median (resp. k-means) on P' with cost at most $dn^2 \text{OPT}$ (resp. $d^2 n^4 \text{OPT}$ for k-means). Then, one can compute a k-median solution for P, with the same cost as $\mathcal{S}'$ for P' up to an additive error OPT/n, in time $O(nd)$. This also works if we replace P' and P.

Proof. First, in rounding points to the closest multiple of g, the distance between any point and its rounding is at most $g \leq \frac{U}{n^4 d^2 \log \Delta} \leq \frac{\text{OPT}}{n^2}$. Summing over all points, any solution computed on the gridded data has cost within an additive factor $\pm \frac{\text{OPT}}{n}$ of the true cost.

Let $\mathcal{S}$ be the solution obtained from $\mathcal{S}'$ by reversing the construction of P', namely re-adding the shift that was substracted to every box. Since adjacency is preserved by Proposition 4.4, all points that are in the same cluster have the same shift, and therefore all intra-cluster distances are the same in P and P'. Therefore, the costs are equal and, by extension, $\text{cost}(\mathcal{S}) \in \text{cost}(\mathcal{S}') \pm \text{OPT}/n$, where the additive OPT/n comes from the rounding.

Finally, the smallest non-zero distance is $g = \frac{U}{n^4 d^2 \log \Delta}$, and with probability $1 - 1/n$ the diameter is $\sqrt{d} \cdot 3dn^2 \cdot U \cdot k$ (see Proposition 4.4), implying that the spread of P' is $(nd \log \Delta)^{O(1)}$. □

Combining the algorithm of Lemma 4.2, which gives a bound on U, with Lemma 4.5 brings us to the following theorem:

Theorem 4.6. *Given $P \subset \mathbb{R}^d$ with spread Δ, there is an algorithm running in time $O(nd \log\log \Delta)$ that computes a set P' such that (1) with probability $1 - 1/n$, the spread of P' is $\mathrm{poly}(n, d, \log(\Delta))$ and (2) any solution with cost at most $dn^2 OPT$ for k-median (resp. $d^2n^4 OPT$ for k-means) on P' can be converted in time $O(nd)$ into a solution with same cost on P, up to an additive error $O(OPT/n)$.*

To summarize, we have shown that one can, in $\tilde{O}(nd)$ time, find a modified dataset P' that preserves the cost of corresponding k-means and k-median solutions from P. Importantly, this P' has spread that is logarithmic in the spread of P. As a result, one can apply Algorithm 1 onto P' in $\tilde{O}(nd)$ time without compromising the compression quality with respect to P. Lastly, this compression on P' can be re-formatted onto P in $\tilde{O}(nd)$ time.

5 FAST COMPRESSION IN PRACTICE

Despite the near-linear time algorithm described in Sections 3 and 4, the coreset construction of Algorithm 1 nonetheless requires a bounded approximation to the clustering task before the sampling can occur. Although theoretically justified, it is unclear how necessary this is in practice – would a method that cuts corners still obtain good practical compression?

Metrics. To answer this question, we analyze the sampling methods along two metrics – compression accuracy and construction time. Although measuring runtime is standard, it is unclear how to confirm that a subset of points satisfies the coreset property over all solutions. To this end, we use the distortion measure introduced in [57] $\max\left(\frac{\mathrm{cost}(P, C_\Omega)}{\mathrm{cost}(\Omega, C_\Omega)}, \frac{\mathrm{cost}(\Omega, C_\Omega)}{\mathrm{cost}(P, C_\Omega)}\right)$, where C_Ω is a candidate solution computed over the coreset Ω. This will be within $1 + \varepsilon$ if the coreset guarantee is satisfied but may be unbounded otherwise. We refer to this as the *coreset distortion.*

5.1 Goal and Scope of the Empirical Analysis

To motivate the upcoming experiments, we begin by asking "how do other sampling strategies compare to standard sensitivity sampling?" For this preliminary experiment, we focus on the uniform sampling and Fast-Coreset algorithm (Algorithm 1). For each, we evaluate its distortion across the following real datasets: Adult [32], MNIST [48], Taxi [52], Star [55], Song [12], Census [53], and Cover Type [13]. The dataset characteristics are summarized in Table 3.

The resulting comparisons can be found in Table 2. Since sensitivity sampling is the recommended coreset method [57], we use it to obtain a baseline distortion for each dataset[8]. We then compare uniform sampling and Fast-Coresets against this baseline by showing the ratio of their distortion divided by the distortion obtained by sensitivity sampling. As guaranteed by Section 4, we see that Fast-Coresets obtain consistent distortions with standard sensitivity sampling. However, the question is more subtle for uniform sampling – it performs well on most real-world datasets but catastrophically fails on a few (for example, it is $\sim 600\times$ worse on the taxi dataset when compared with standard sensitivity sampling).

This confirms that uniform sampling is not unequivocally reliable as an aggregation method – although it is fast, it has the potential to miss important data points. On the other end of the runtime vs. quality spectrum, Fast-Coresets consistently provide an accurate compression but, despite being the fastest coreset method, are still significantly slower than uniform sampling. Thus, the fundamental question is: when is one safe to use fast, inaccurate sampling methods and when is the full coreset guarantee necessary?

We focus the remainder of the experimental section on this question. Specifically, we define a suite of compression methods that interpolate between uniform sampling and Fast-Coresets

[8]We use $k = 100$ and coreset size $m = 40k$ for these experiments.

and evaluate these methods across a set of synthetic datasets. This allows us to experimentally verify the full spectrum of speed vs. accuracy tradeoffs and provide insight into which dataset characteristics are necessary before one can apply suboptimal sampling methods.

We complement this with a comprehensive study of related topics, such as additional analysis on real datasets, comparing against other state-of-the-art coreset methods, and verifying that these results extend to the streaming setting. Unless stated otherwise, our experimental results focus on the k-means task.

Dataset	Uniform Dist.	Fast-Coreset Dist.
	Sensitivity Dist.	Sensitivity Dist.
Adult	1.00	1.07
MNIST	1.03	1.02
Star	**8.46**	1.09
Song	1.11	1.47
Cover-Type	1.02	1.27
Taxi	**614**	1.90
Census	1.09	1.08

Table 2. Ratio of each algorithms' distortion with respect to sensitivity sampling's distortion. Failure cases are bolded.

Dataset	Points	Dim
Adult	48 842	14
MNIST	60 000	784
Star	138 500	3
Song	515 345	90
Cover Type	581 012	54
Taxi	754 539	2
Census	2 458 285	68

Table 3. Description of real world datasets

5.2 Experimental Setup

Algorithms. We compare Fast-Coresets (Algorithm 1) against 4 different benchmark sampling strategies that span the space between optimal time and optimal accuracy, as well as state-of-the-art competitors BICO [38] and Streamkm++[1].

- *Standard uniform sampling*. Each point is sampled with equal probability and weights are set to n/m, where m is the size of the sample.
- *Lightweight coresets [6].* Obtain a coreset by sampling sensitivity values with respect to the 1-means solution, i.e. sensitivities are given by $\hat{s}(p) = 1/|P| + \text{cost}(p, \mu)/\text{cost}(P, \mu)$, where μ is the dataset mean.
- *Welterweight coresets.* For any $j \in \{1, ..., k\}$, we compute a coreset using sensitivity sampling with respect to a candidate j-means solution.
- *Standard sensitivity sampling [47].* We refer to Section 2.
- *BICO [38].* Utilizes BIRCH [58] to produce a k-means coreset in a stream.
- *Streamkm++ [1].* Uses k-means++ to create large coresets for the streaming k-means task.

We do not compare against group sampling [25] as it uses sensitivity sampling as a subroutine and is merely a preprocessing algorithm designed to facilitate the theoretical analysis. By the authors' own admission, it should not be implemented.

We use j going forward to describe the number of centers in the candidate j-means solution. Thus, lightweight coresets have $j = 1$ while Fast-Coresets have $j = k$.

We take a moment here to motivate the welterweight coreset algorithm. Consider that lightweight coresets use the 1-means solution to obtain the sensitivities that dictate the sampling distribution whereas sensitivity sampling uses the full k-means solution. In effect, as we change the value of j, the cluster sizes $|C_p|$ in our approximate solution change. Referring back to equation 1, one can see that setting j between 1 and k acts as an interpolation between uniform and sensitivity sampling. We default to $j = \log k$ unless stated otherwise. We use the term 'accelerated sampling methods' when referring to uniform, lightweight and welterweight coresets as a group.

	Method							
	Uniform Sampling		Lightweight		Welterweight		Fast Coreset	
	$m = 40k$	$m = 80k$	$m = 40k$	$m = 80k$	$m = 40k$	$m = 80k$	$m = 40k$	$m = 80k$
c-outlier	**405 ± 24K**	**159 ± 20K**	1.07 ± 0.0	**5.63 ± 84.6**	**9.44 ± 105**	1.03 ± 0.0	1.12 ± 0.0	1.05 ± 0.0
Geometric	**86.3 ± 8.4K**	**21.8 ± 652**	1.05 ± 0.0	**11.6 ± 452**	1.11 ± 0.0	1.03 ± 0.0	1.11 ± 0.0	1.05 ± 0.0
Gaussian Mix.	3.17 ± 3.42	1.43 ± 0.25	2.64 ± 1.61	1.81 ± 0.58	2.26 ± 1.55	1.28 ± 0.07	1.24 ± 0.0	1.13 ± 0.0
Benchmark	1.07 ± 0.0	1.03 ± 0.0	1.11 ± 0.0	1.05 ± 0.0	1.10 ± 0.0	1.04 ± 0.0	1.15 ± 0.0	1.06 ± 0.0
–	–	–	–	–	–	–	–	–
MNIST	1.08 ± 0.0	1.03 ± 0.0	1.08 ± 0.0	1.03 ± 0.0	1.08 ± 0.0	1.04 ± 0.0	1.08 ± 0.0	1.04 ± 0.0
Adult	1.08 ± 0.0	1.04 ± 0.0	1.09 ± 0.0	1.04 ± 0.0	1.32 ± 0.0	1.17 ± 0.0	1.17 ± 0.0	1.07 ± 0.0
Star	**5.43 ± 0.99**	2.25 ± 0.12	1.94 ± 0.1	1.68 ± 0.0	1.85 ± 0.03	1.86 ± 0.03	1.13 ± 0.01	1.14 ± 0.01
Song	1.30 ± 0.0	1.14 ± 0.0	1.13 ± 0.0	1.12 ± 0.0	1.18 ± 0.0	1.16 ± 0.0	1.50 ± 0.0	1.29 ± 0.0
Cover-Type	1.12 ± 0.0	1.05 ± 0.0	1.11 ± 0.0	1.06 ± 0.0	1.17 ± 0.0	1.08 ± 0.0	1.11 ± 0.0	1.04 ± 0.0
Taxi	**3.7K ± 43K**	**2.3K ± 12K**	2.4 ± 0.1	2.51 ± 0.05	**5.82 ± 10.8**	**84.3 ± 263**	4.64 ± 0.09	4.90 ± 0.35
Census	1.15 ± 0.0	1.07 ± 0.0	1.12 ± 0.0	1.06 ± 0.0	1.14 ± 0.0	1.06 ± 0.0	1.13 ± 0.0	1.07 ± 0.0

Table 4. Distortion means and variances for different sample sizes across datasets for k-means; taken over 5 runs. Failure cases (distortion > 5) are bolded. Catastrophic failures (distortion > 10) are underlined.

	Method							
	Uniform Sampling		Lightweight		Welterweight		Fast Coreset	
	Streaming	Static	Streaming	Static	Streaming	Static	Streaming	Static
c-outlier	**221 ± 15K**	**261 ± 44K**	1.07 ± 0.0	**5.51 ± 78.9**	1.09 ± 0.0	**12.1 ± 80.1**	1.13 ± 0.0	1.11 ± 0.0
Geometric	**66.5 ± 2.7K**	**140 ± 1.8K**	**85.2 ± 2.8K**	**45.6 ± 4.2K**	1.09 ± 0.0	**10.4 ± 349**	1.15 ± 0.0	1.12 ± 0.0
Gaussian Mix.	1.51 ± 0.07	2.42 ± 2.52	2.35 ± 0.67	2.38 ± 1.78	1.45 ± 0.05	3.65 ± 3.85	1.15 ± 0.0	1.24 ± 0.0
Benchmark	1.10 ± 0.0	1.07 ± 0.0	1.08 ± 0.0	1.11 ± 0.0	1.09 ± 0.0	1.11 ± 0.0	1.18 ± 0.0	1.16 ± 0.0
MNIST	1.42 ± 0.0	1.08 ± 0.0	1.07 ± 0.0	1.07 ± 0.0	1.02 ± 0.0	1.09 ± 0.0	1.12 ± 0.0	1.08 ± 0.0
Adult	1.33 ± 0.0	**895K ± 3.2B**	1.09 ± 0.0	1.09 ± 0.0	1.27 ± 0.01	1.32 ± 0.0	1.14 ± 0.0	1.15 ± 0.0

Table 5. Distortion means and variances in the streaming and non-streaming setting for k-means; taken over 5 runs. Failure cases (distortion > 5) are bolded. Catastrophic failures (distortion > 10) are underlined.

	Static		Streaming
	$m = 40k$	$m = 80k$	
c-outlier	**12.5 ± 0.60**	**6.10 ± 3.90**	**18.8 ± 5.47**
Geometric	2.42 ± 0.01	1.38 ± 0.0	2.91 ± 0.14
Gaussian Mix.	**11.4 ± 0.69**	**7.58 ± 0.33**	**27.0 ± 6.72**
Benchmark	**5.49 ± 0.03**	2.64 ± 0.0	**6.67 ± 0.01**
–	–	–	–
MNIST	**6.41 ± 1.36**	3.83 ± 0.02	**11.7 ± 2.53**
Adult	3.12 ± 0.0	3.39 ± 0.0	**9.03 ± 3.10**
Star	1.07 ± 0.0	1.26 ± 0.0	–
Song	**5.44 ± 1.32**	3.24 ± 0.45	–
Cover-Type	1.49 ± 1.78	1.41 ± 0.0	–
Taxi	1.60 ± 0.0	1.84 ± 0.0	–
Census	3.45 ± 0.03	3.13 ± 0.04	–

Table 6. Distortion values for the BICO algorithm in the static and streaming settings, taken over five runs. Failure cases (distortion > 5) are bolded. Catastrophic failures (distortion > 10) are underlined.

Datasets. We complement our suite of real datasets with the following artificial datasets. We default to $n = 50\,000$ and $d = 50$ unless stated otherwise.

- *c-outlier.* Place $n - c$ points in a single location and c points a large distance away.
- *Geometric.* Place ck points at $(1, 0, 0, \cdots)$, $\frac{ck}{r}$ points at $(0, 1, 0, \cdots)$, $\frac{ck}{r^2}$ points at $(0, 0, 1, \cdots)$, and so on for $\log_r(ck)$ rounds. Thus, the data creates a high-dimensional simplex with uneven weights across the vertices. We default to $c = 100$ and $r = 2$.
- *Gaussian mixture.* A set of scattered Gaussian clusters of varying density.

These clusters are sequentially defined, with the size of the first cluster defined by $\frac{n}{\kappa} \exp(\gamma \cdot \rho_0)$, where κ is the number of Gaussian clusters, ρ_0 is uniformly chosen from $[-0.5, 0.5]$, and γ is a

hyperparameter that affects the distribution of cluster sizes. Then, given clusters $\{c_1, \cdots, c_i\}$, we obtain the size of the $(i+1)$-st cluster by

$$|c_{i+1}| = \frac{n - \sum_i |c_i|}{\kappa - i} \exp\left(\gamma \cdot \rho_{i+1}\right).$$

This has the property that all clusters have size n/k when $\gamma = 0$ and, as γ grows, the cluster sizes diverge at an exponential rate. We note that this is a well-clusterable instance with respect to cost stability conditions, see [3, 22, 46, 54].

- *Benchmark.* A specific distribution of points introduced in [57] as a testbed for coreset algorithms. It has the property that all reasonable k-means solutions are of equal quality but are maximally far apart in the solution space. Thus, the dataset is fully determined by the number of centers k. As suggested, we produce three benchmark datasets of varying size before applying random offsets to each. We choose the sizes by $k_1 = \frac{k}{c_1}$, $k_2 = \frac{k-k_1}{c_2}$, and $k_3 = k - k_1 - k_2$ for $c_1, c_2 \in \mathbb{R}^+$. Note, the benchmark dataset being difficult for sensitivity sampling does not imply that it should be equally difficult for other sampling methods.

The artificial datasets are constructed to emphasize strengths and weaknesses of the various sampling schemas. For example, the c-outlier problem contains very little information and, as such, should be simple for any sampling strategy that builds a reasonable representation of its input. The geometric dataset then increases the difficulty by having more regions of interest that must be sampled. The Gaussian mixture dataset is harder still, as it incorporates uneven inter-cluster distances and inconsistent cluster sizes. Lastly, the benchmark dataset is devised to be a worst-case example for sensitivity sampling.

Data Parameters. In all real and artificial datasets, we add random uniform noise η with $0 \leq \eta_i \leq 0.001$ in each dimension in order to make all points unique. Unless specifically varying these parameters, we default all algorithms in 5.2 to $k = 100$ for the Adult, MNIST, Star, and artificial datasets and $k = 500$ for the Song, Cover Type, Taxi, and Census datasets. Our default coreset size is then $m = 40k$. We refer to the coreset size scalar (the "40" in the previous sentence) as the *m-scalar*. We only run the dimension-reduction step on the MNIST dataset, as the remaining datasets already have sufficiently low dimensionality. We run our experiments on an Intel Core i9 10940X 3.3GHz 14-Core processor.

5.3 Evaluating Sampling Strategies

Theoretically guaranteed methods. We first round out the comparison between the Fast-Coreset algorithm and standard sensitivity sampling. Specifically, the last columns of Tables 4 and 5 show that the Fast-Coreset method produces compressions of consistently low distortion and that this holds across datasets, m-scalar values and in the streaming setting. Despite this, Figure 1 shows that varying k from 50 to 400 causes a linear slowdown in sensitivity sampling but only a logarithmic one for the Fast-Coreset method. This analysis confirms the theory in Section 4 – Fast-Coresets obtain equivalent compression to sensitivity sampling but do not have a linear runtime dependence on k. We therefore do not add traditional sensitivity sampling to the remaining experiments.

Speed vs. Accuracy. We now refer the reader to the remaining columns of Table 4 and to Figure 2, where we show the effect of coreset size across datasets by sweeping over m-scalar values. Despite the suboptimal theoretical guarantees of the accelerated sampling methods, we see that they obtain competitive distortions on most of the real-world datasets while running faster than Fast-Coresets in practice. However, uniform sampling breaks on the Taxi and Star datasets – Taxi corresponds to the 2D start locations of taxi rides in Porto and has many clusters of varied size while Star is the pixel values of an image of a shooting star (most pixels are black except for a small cluster of white

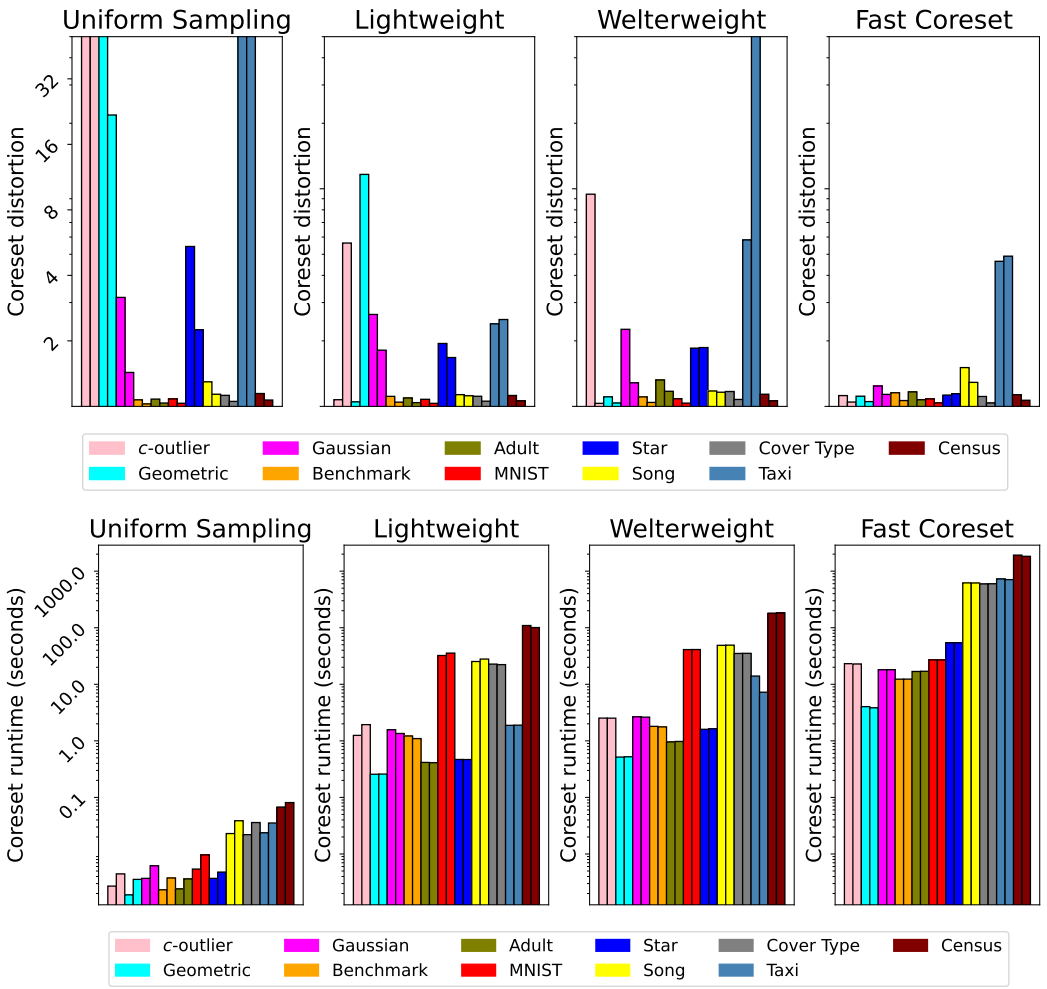


Fig. 2. *Top*: The effect of the m-scalar on coreset distortion for real-world datasets. This is a visualization of the data in Table 4. *Bottom*: The effect of the m-scalar on the algorithm runtime for real-world datasets. All values are the mean over 5 runs. The three bars represent samples of size $m = 40k, 80k$.

pixels). Thus, it seems that uniform sampling requires well-behaved datasets, with few outliers and consistent class sizes.

To verify this, consider the results of these sampling strategies on the artificial datasets in Table 4 and Figure 2: as disparity in cluster sizes and distributions grows, the accelerated sampling methods have difficulty capturing all of the outlying points in the dataset. Thus, Figure 2 shows a clear interplay between runtime and sample quality: *the faster the method, the more brittle its compression.*

While uniform sampling is expected to be brittle, it may be less obvious what causes light- and welterweight coresets to break. The explanation is simple for lightweight coresets: they sample according to the 1-means solution and are therefore biased towards points that are far from the mean. Thus, as a simple counterexample, lightweight coresets are likely to miss a small cluster that is close to the center-of-mass of the dataset. This can be seen in Figure 3, where we show an

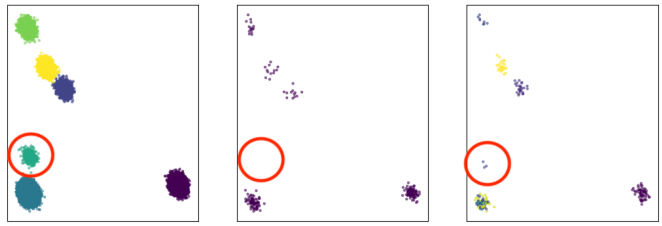


Fig. 3. The results of lightweight and fast-coreset constructions on a 2D Gaussian mixture dataset of $n = 100K$ points with clusters of varying size. The circled cluster has ~ 400 points and coresets have 200 points. *Left*: Original multivariate-Gaussian dataset. *Middle*: Lightweight coresets fail to capture the cluster of ~ 400 points. *Right*: Sensitivity sampling with $j = k$ identifies all of the clusters.

example where the lightweight coreset construction fails on the Gaussian mixture dataset. Since the small circled cluster is close to the center-of-mass of the dataset, it is missed when sampling according to distance from the mean.

Generalizing this reasoning also explains the brittleness of welterweight coresets when $j < k$. To see this, let C_j be the approximation obtained during welterweight coresets and observe that the sum of importance values of the points belonging to center $c_i \in C_j$ is

$$\sum_{p \in c_i} \left[\frac{\text{cost}(p, c_i)}{\text{cost}(c_i, C_j)} + \frac{1}{|c_i|} \right] = \frac{\sum_{p \in c_i} \text{cost}(p, c_i)}{\text{cost}(c_i, C_j)} + 1 = 2.$$

Thus, our probability mass is distributed across the clusters that have been found in the approximate solution. Naturally, if $j < k$ and we missed a cluster from OPT, there is some set of points that have not received an appropriate probability mass and may therefore be missed.

	$\gamma = 0$	$\gamma = 1$	$\gamma = 3$	$\gamma = 5$
LW Coreset	1.03	1.03	1.36	2.17
$j = 2$	1.04	1.04	1.04	1.92
$j = \log k$	1.04	1.04	1.04	1.95
$j = \sqrt{k}$	1.05	1.06	1.04	1.18
Fast Coreset	1.03	1.03	1.04	1.12

Table 7. The effect of γ in the Gaussian mixture dataset on the coreset distortion. We report the means over 5 random dataset generations. Each generation had 50 000 points in 50 dimensions, with 50 Gaussian clusters and coresets of size 4 000. We set $k = 100$.

	Uniform Sampling	Light-weight	Welter-weight	Fast Coreset
MNIST	125	124	125	124
Adult	250	249	249	250
Star	24.6	53.9	25.9	27.3
Song	168	166	170	170
Census	367	356	374	322
Taxi	1.42	0.13	0.10	0.13
Cov. Type	105	103	100	98

Table 8. $cost(P, C_S)$, where P is the whole dataset and C_S is found via k-means++[2, 23] ($k = 50$) and Lloyd's algorithm on the coreset. Sample sizes are $m = 4\,000$ for the first two rows and $m = 20\,000$ for the remaining ones. Initializations are identical within each row. We show the first 3 digits of the cost for readability.

We evaluate the full extent of this relationship in Table 7, where we show the interplay between the welterweight coreset's j parameter (number of centers in the approximate solution) and the Gaussian mixture dataset's γ parameter (higher γ leads to higher class imbalance). We can consider this as answering the question "How good must our approximate solution be before sensitivity sampling can handle class imbalance?" To this end, all the methods have low distortion for small values of γ but, as γ grows, only Fast-Coresets (and, to a lesser extent, welterweight coresets for larger values of j) are guaranteed to have low distortion.

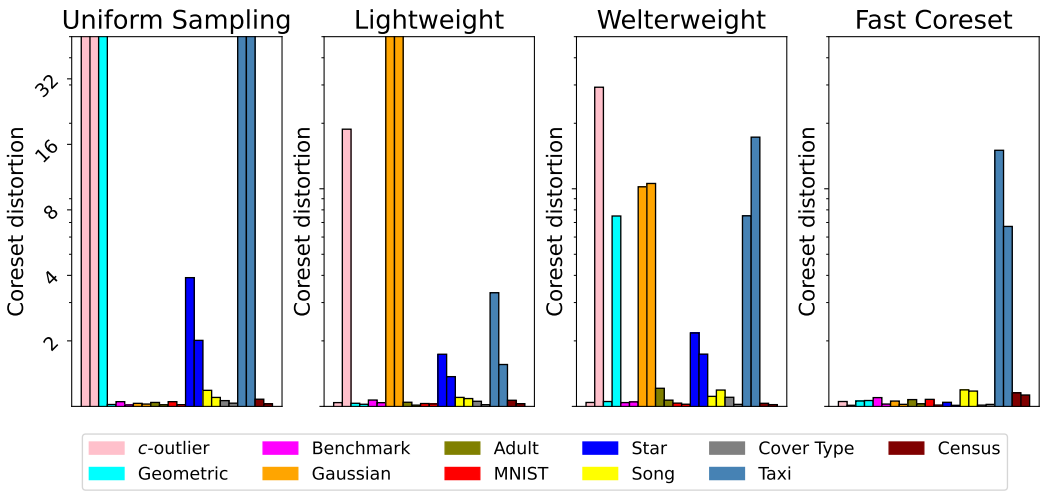


Fig. 4. Sample coreset distortions under k-median for one run on each dataset. Bars within each dataset correspond to $m = 40k, 60k, 80k$.

Dataset	c-outlier	Geometric	Gaussian	Benchmark
Distortion	2.07	1.40	1.54	2.53

Table 9. Distortions for Streamkm++ on artificial datasets.

For completeness, we verify that these results also hold for the k-median task in Figure 4. There, we see that k-median distortions across datasets are consistent with k-means distortions. We show one of five runs to emphasize the random nature of compression quality when using various sampling schemas.

To round out the dataset analysis, we note that BICO performs consistently poorly on the coreset distortion metric[9], as can be seen in Table 6. We also analyze the Streamkm++ method across the artificial datasets in Table 9 with $m = 40k$ and see that it obtains poor distortions compared to sensitivity sampling. This is due to Streamkm++'s required coreset size – logarithmic in n and exponential in d – being much larger than those for sensitivity sampling (sensitivity sampling coresets depend on neither parameter). We did not include Streamkm++ in tables 4, 5 due to its suboptimal coreset size, distortion and runtime.

Lastly, we point out that every sampling method performs well on the benchmark dataset, which is designed to explicitly punish sensitivity sampling's reliance on the initial solution. Thus, we verify that there is no setting that breaks sensitivity sampling.

We lastly show how well these compression schemas facilitate fast clustering on large datasets in Table 8. Consider that a large coreset-distortion means that the centers obtained on the coreset poorly represent the full dataset. However, among sampling methods with small distortion, it may be the case that one consistently leads to the 'best' solutions. To test this, we compare the solution quality across all fast methods on the real-world datasets, where coreset distortions are consistent. Indeed, Table 8 shows that no sampling method leads to solutions with consistently minimal costs.

[9]We do not include BICO or Streamkm++ in Figures 2, 4, 5, as they do not fall into the $\tilde{O}(nd)$ complexity class and are only designed for k-means.

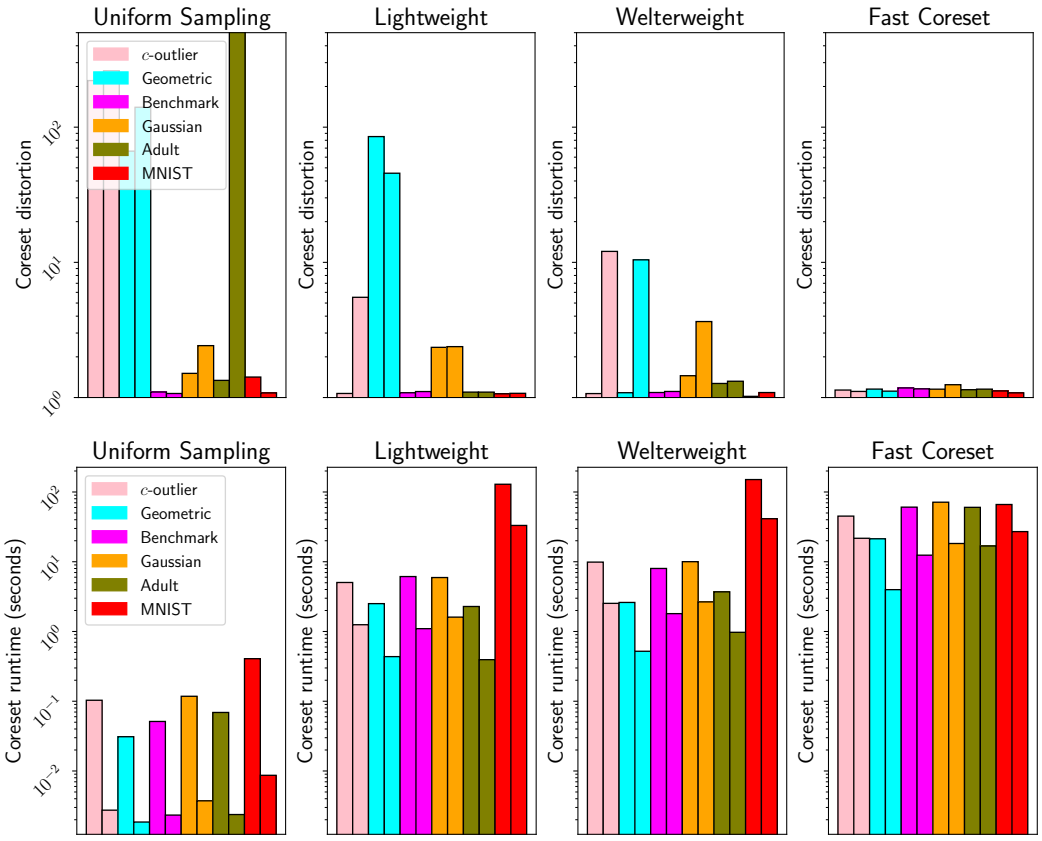


Fig. 5. *Top*: Coreset distortion on the k-means task in the streaming and non-streaming settings. This is a visualization of the data in Table 5. *Bottom*: Coreset construction runtimes in the streaming and non-streaming settings for the linear and sub-linear complexity coreset algorithms. Bars are [Streaming, Non-Streaming].

5.4 Streaming Setting

One of the most common use-cases for big-data algorithms is the streaming setting, where one receives input in batches and must maintain a compression that is representative of the dataset. Although there is a wealth of sampling and coreset methods in the streaming paradigm, we require consistency across algorithms and therefore assume a black-box sampling procedure. Since the coreset property is preserved under composition, we utilize the merge-&-reduce strategy originally proposed by [11] and first applied to maintaining clustering coresets in stream by [40]. The idea is to first partition the input into b blocks and then perform sampling and composition along them until a single compression is obtained. Specifically, we start by obtaining a coreset on each block. Then, combining samples using a complete binary tree, we (1) recursively re-sample from the children until there is at least one coreset for each level[10] in the tree and then (2) concatenate these samples and obtain one final coreset from the composition. Since we are composing coresets from coresets, the errors stack and, in theory, we should require more samples to obtain a similar accuracy.

[10]If there are $b = 8$ blocks, then there would be coresets corresponding to blocks $[[1], [2], [3, 4], [5, 6, 7, 8]]$

Despite this, we see the opposite result for many of our sampling strategies. Surprisingly, Table 5 and Figure 5 show that the accelerated sampling methods all perform *at least as well* under composition on the artificial datasets and do not suffer significant drops in accuracy, variance or runtime on the real datasets. Although inconsistent with the prevailing intuition, we must therefore conclude that the accelerated sampling methods are *equally* feasible in the streaming setting. We suspect that this is due to the non-uniformity imposed by the merge-&-reduce algorithm. To see this, consider uniform sampling on the c-outlier dataset during the final step of the composition, where we are composing the samples corresponding to each layer of the tree. Assume first that our outlier points happened to fall in the first block. Then we have taken a sample of size m from this block and immediately use this for the final composition. Thus, in this case the outliers are *more* likely to be in our final sample than in the non-streaming setting. In the alternate setting where the outliers are less likely, our expected error is already high and missing the outlier 'more' cannot worsen our expected error.

5.5 Takeaways

To summarize the comparisons between sampling strategies and datasets, the practical guideline is that uniform sampling usually works well, so an optimistic user can default to that (while accepting that it might fail). This was evidenced in Sections 5.1 and 5.3. However, it was also shown there that the accelerated sampling methods all have a chance of catastrophically failing. Thus, in accordance with Table 7, a cautious user may wish to verify whether uniform sampling will work by checking how balanced the dataset's clusters are. By our theoretical contribution (Corollary 3.2 and Theorem 4.6), performing this verification is just as expensive as computing a coreset. Thus, the cautious user may as well create a Fast-Coreset in that amount of time. Importantly, we do not claim that there is a 'best' algorithm among the ones that have been discussed.

6 CONCLUSION

In this work, we discussed the theoretical and practical limits of compression algorithms for center-based clustering. We proposed the first nearly-linear time coreset algorithm for k-median and k-means. Moreover, the algorithm can be parameterized to achieve an asymptotically optimal coreset size. Subsequently, we conducted a thorough experimental analysis comparing this algorithm with fast sampling heuristics. In doing so, we find that although the Fast-Coreset algorithm achieves the best compression guarantees among its competitors, naive uniform sampling is already a sufficient compression for downstream clustering tasks in well-behaved datasets. Furthermore, we find that intermediate heuristics interpolating between uniform sampling and coresets play an important role in balancing efficiency and accuracy.

Although this closes the door on the highly-studied problem of optimally small and fast coresets for k-median and k-means, open questions of wider scope still remain. For example, when does sensitivity sampling guarantee accurate compression with optimal space in linear time and can these conditions be formalized? Furthermore, sensitivity sampling is incompatible with paradigms such as fair-clustering [8, 15, 21, 43, 56] and it is unclear whether one can expect that a linear-time method can optimally compress a dataset while adhering to the fairness constraints.

7 ACKNOWLEDGEMENTS

Andrew Draganov and Chris Schwiegelshohn are partially supported by the Independent Research Fund Denmark (DFF) under a Sapere Aude Research Leader grant No 1051-00106B. David Sauplic has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Sklodowska-Curie grant agreement No 101034413.

Algorithm 2 Crude-Approx(P)

procedure Count-Distinct-Cells(P, c, ℓ) ▷ data P, c is the center of the quadtree grid at level ℓ
  Let $\mathcal{D}$ be a dictionary, and count = 0.
  for each point $p \in P$ **do**
    let c^p be the center of the cell containing p. The i-th coordinate of c_p is $\lfloor \frac{p_i - c_i}{2^\ell} \rfloor \cdot 2^\ell + \frac{2^\ell}{2}$.
    if c_p is a not a key of $\mathcal{D}$ **then**
      insert c_p in $\mathcal{D}$ and do count $\leftarrow$ count +1.
  Output: True if count $\geq k+1$, False otherwise.
procedure Crude-Approx(P) ▷ data P, with diameter Δ.
  let s be a u.a.r in $[0, \Delta]^d$, and $c = (s, ..., s) \in \mathbb{R}^d$.
  using a binary search, find the smallest ℓ such that Count-Distinct-Cells(P, c, ℓ) = True. Let ℓ_0 be that level.
  Output: $U = 2^{\ell_0}\Delta$.

Algorithm 3 Reduce-Spread(P, U)

procedure Reduce-Diameter(P, U) ▷ data P, upper bound on OPT U
  Let $\mathcal{D}$ be a dictionary
  Let $r = \sqrt{d}n^2 \cdot U$, s be chosen u.a.r. in $\{0, ..., r\}$, and $c = s, ..., s$ be the center of the grid.
  for each point $p \in P$ **do**
    let c^p be the center of the cell containing p. The i-th coordinate of c_p is $\lfloor \frac{p_i - c_i}{r} \rfloor \cdot r + \frac{r}{2}$. Add p to $\mathcal{D}[c_p]$.
  Identify the non-empty dictionary keys $c^1, ..., c^k$, and let $C_1, ..., C_k$, be the corresponding cells.
  for each coordinate $i = 1, ..., d$ **do**
    Sort the cells according to the i-th coordinate of their center. Let $\delta = 0$.
    for $j = 1, ..., k$ **do**
      if $\frac{c_i^j - c_{i-1}^j}{r} \geq 2$ **then** update $\delta \leftarrow \delta + c_i^j - c_{i-1}^j - 2r$.
      Substract δ from the i-th coordinate of all points in the j-th cell.
  Output: the dataset P' consisting of all shifted points.
procedure Reduce-Min-Distance(P, U) ▷ data P, U such that $U \leq n^2 d \log(\Delta)$OPT.
  Round each coordinates of points to the closest multiple of $g := \frac{U}{n^4 d^2 \log \Delta}$.
  Output: the dataset P' with all point after rounding.

8 PROOFS, PSEUDO-CODE, AND EXTENSIONS

We put the proofs, pseudo-code and algorithmic extensions towards the end of the paper for improved readability of the primary text. Algorithm 2 corresponds to the discussion in Section 4.1 and Algorithm 3 corresponds to the discussion in Section 4.2.

8.1 Proof of Corollary 3.2

Recall that Corollary 3.2 states that Algorithm 1 produces an ε-coreset in time $\tilde{O}(nd \log \Delta)$.

Proof of Corollary 3.2. First, performing the Johnson Lindenstrauss embedding [50] takes time $\tilde{O}(nd)$.

On the projected dataset, the algorithm Fast-kmeans++ runs in time $\tilde{O}(n \log \Delta)$, and its solution has an approximation-ratio $O\left(\tilde{d}^z \log k\right) = O\left(\log^{z+1} k\right)$ for $\tilde{P}$. The guarantee offered by the embedding ensures that the clustering $\{C_1, ..., C_k\}$ still has approximation ratio for P [50].

For k-means, computing the 1-mean solution for each C_i takes time $O(nd)$ (the 1-mean is simply the mean). For k-median, computing the 1-median solution can be done as well in time $O(nd)$ [20]. We note that both may be approximated to a factor 2 in constant time, by sampling uniformly at randm few points from each cluster [26].

Provided the c_i and the partition C_i, computing $|C_i|$ and $\text{cost}(C_i, c_i)$ for all i also takes time $O(nd)$.

Since the solution consisting of assigning each $p \in C_i$ to c_i is a $O\left(\log^{z+1} k\right)$-approximation, the values $s(p)$ defined in Algorithm 1 can be used to perform the coreset construction algorithm, and we conclude from Fact 3.1.

□

8.2 Reduction of k-means to k-median.

In our argument, the only step specific to k-median is computing the upper-bound U on the cost of the solution. Provided such an upper-bound, rounding points and shifting the box would work exactly alike for k-means. Therefore, the next lemma is enough to extend our reduction of the spread to k-means:

Lemma 8.1. *Let $\mathcal{S}$ be a c-approximation for k-median on P. Then, $\mathcal{S}$ is a nc^2-approximation for k-means on P.*

Proof. Let cost_1 (resp. cost_2) be the k-median (resp. k-means) cost, OPT_1 (resp. OPT_2) be the optimal k-median (resp. k-means) solution. We have the following inequalities:

$$\begin{aligned}
\text{cost}_2(\mathcal{S}) = \sum_{p \in P} \text{dist}(p, \mathcal{S})^2 &\leq \left(\sum_{p \in P} \text{dist}(p, \mathcal{S}) \right)^2 \\
&\leq c^2 \cdot \left(\sum_{p \in P} \text{dist}(p, \text{OPT}_1) \right)^2 \\
&\leq c^2 \cdot \left(\sum_{p \in P} \text{dist}(p, \text{OPT}_2) \right)^2 \\
&\leq c^2 \cdot n \cdot \sum_{p \in P} \text{dist}(p, \text{OPT}_2)^2,
\end{aligned}$$

where the last inequality stems from Cauchy-Schwarz. Therefore, $\mathcal{S}$ is an nc^2-approximation to k-means. □

8.3 Estimation of the Optimal Cost in a Tree

proof of Lemma 4.1. If the input is spread in $k+1$ distinct cell at level i, then in any solution there is at least one of those cells with no center. In the tree metric, points lying in this cell have therefore connection cost at least $\sqrt{d} 2^{-i+1} \cdot \Delta$. Thus, the left-hand-side of the inequality holds.

On the other hand, if the input is contained in k cells at level $i-1$, then placing a center arbitrarily in each cell yields a solution with cost at most $n \cdot \sqrt{d} 2^{-i+4} \cdot \Delta$. Indeed, the distance from any point

to its closest center is at most $2 \cdot \sum_{j \geq i-1} \sqrt{d} 2^{-j} \cdot 2\Delta \leq 4\sqrt{d} 2^{-i+2} \cdot \Delta$ (summing the edge length from the point to the cell at level i, and then going down to the center). This concludes the proof.

□

Proof of Lemma 4.2. The running time of Count-Distinct-Cells in Algorithm 2 is $\tilde{O}(nd)$, using a standard dictionary data structure. To compute U, Crude-Approx makes $O(\log \log \Delta)$ calls to Count-Distinct-Cells, as there are $O(\log \Delta)$ levels. This concludes the complexity of the algorithm.

The approximation guarantee for k-median directly stems from Lemma 4.1 and Lemma 2.2. For k-means, it is a consequence of Lemma 8.1.

□

8.4 Extensions to Algorithm 1

Consider that Algorithm 1 only needs to be provided with an assignment to a solution that is a $O(\text{polylog}\, k)$ approximation, implying that one could compute the solution C via any algorithm that satisfies this. Indeed, this initial solution may as well be an $O\left(\text{polylog} \|P\|_0\right)$ approximation, where $\|P\|_0$ is the number of non-zero samples in P. Using the iterative coreset construction from [14], one could then derive a near-optimal coreset size, only suffering a $O(\log^* n)$ loss in the running time.

As an example, we illustrate a different approach for k-median. One could first embed the input into a hierarchically separated tree (HST) with expected distortion $O(\log \|P\|_0)$ [35]. On such tree metrics, solving k-median can be done in linear time using dedicated algorithms (see e.g. [24]). Using the solution from the HST metric, one can compute a coreset, and iterate using the previous argument. This embedding into HST is very similar to what is done by the Fast-kmeans++ algorithm, but can be actually performed in *any* metric space, not only Euclidean. For instance, in a metric described by a graph with m edges, the running time of this construction would be near linear-time $\tilde{O}(m)$.

REFERENCES

[1] Marcel R. Ackermann, Marcus Märtens, Christoph Raupach, Kamil Swierkot, Christiane Lammersen, and Christian Sohler. Streamkm++: A clustering algorithm for data streams. *ACM J. Exp. Algorithmics*, 17(1), 2012. doi: 10.1145/2133803.2184450. URL https://doi.org/10.1145/2133803.2184450.

[2] David Arthur and Sergei Vassilvitskii. k-means++: the advantages of careful seeding. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2007, New Orleans, Louisiana, USA, January 7-9, 2007*, pages 1027–1035, 2007. URL http://dl.acm.org/citation.cfm?id=1283383.1283494.

[3] Pranjal Awasthi and Or Sheffet. Improved spectral-norm bounds for clustering. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques - 15th International Workshop, APPROX 2012, and 16th International Workshop, RANDOM 2012, Cambridge, MA, USA, August 15-17, 2012. Proceedings*, pages 37–49, 2012. doi: 10.1007/978-3-642-32512-0_4. URL http://dx.doi.org/10.1007/978-3-642-32512-0_4.

[4] Olivier Bachem, Mario Lucic, Hamed Hassani, and Andreas Krause. Fast and provably good seedings for k-means. *Advances in neural information processing systems*, 29, 2016.

[5] Olivier Bachem, Mario Lucic, S. Hamed Hassani, and Andreas Krause. Approximate k-means++ in sublinear time. In Dale Schuurmans and Michael P. Wellman, editors, *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12-17, 2016, Phoenix, Arizona, USA*, pages 1459–1467. AAAI Press, 2016. doi: 10.1609/AAAI.V30I1.10259. URL https://doi.org/10.1609/aaai.v30i1.10259.

[6] Olivier Bachem, Mario Lucic, and Andreas Krause. Scalable k-means clustering via lightweight coresets. In Yike Guo and Faisal Farooq, editors, *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD 2018, London, UK, August 19-23, 2018*, pages 1119–1127. ACM, 2018. doi: 10.1145/3219819.3219973. URL https://doi.org/10.1145/3219819.3219973.

[7] Mihai Badoiu, Sariel Har-Peled, and Piotr Indyk. Approximate clustering via core-sets. In *Proceedings on 34th Annual ACM Symposium on Theory of Computing, May 19-21, 2002, Montréal, Québec, Canada*, pages 250–257, 2002. doi: 10.1145/509907.509947. URL https://doi.org/10.1145/509907.509947.

[8] Sayan Bandyapadhyay, Fedor V. Fomin, and Kirill Simonov. On coresets for fair clustering in metric and euclidean spaces and their applications. In Nikhil Bansal, Emanuela Merelli, and James Worrell, editors, *48th International Colloquium on Automata, Languages, and Programming, ICALP 2021, July 12-16, 2021, Glasgow, Scotland (Virtual Conference)*, volume 198 of *LIPIcs*, pages 23:1–23:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. doi: 10.4230/LIPIcs.ICALP.2021.23. URL https://doi.org/10.4230/LIPIcs.ICALP.2021.23.

[9] Luca Becchetti, Marc Bury, Vincent Cohen-Addad, Fabrizio Grandoni, and Chris Schwiegelshohn. Oblivious dimension reduction for *k*-means: beyond subspaces and the johnson-lindenstrauss lemma. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019, Phoenix, AZ, USA, June 23-26, 2019*, pages 1039–1050, 2019. doi: 10.1145/3313276.3316318. URL https://doi.org/10.1145/3313276.3316318.

[10] Shai Ben-David. A framework for statistical clustering with constant time approximation algorithms for *K*-median and *K*-means clustering. *Mach. Learn.*, 66(2-3):243–257, 2007. doi: 10.1007/s10994-006-0587-3. URL https://doi.org/10.1007/s10994-006-0587-3.

[11] Jon Louis Bentley and James B. Saxe. Decomposable searching problems I: static-to-dynamic transformation. *J. Algorithms*, 1(4):301–358, 1980. doi: 10.1016/0196-6774(80)90015-2. URL https://doi.org/10.1016/0196-6774(80)90015-2.

[12] Thierry Bertin-Mahieux, Daniel P. W. Ellis, Brian Whitman, and Paul Lamere. The million song dataset. pages 591–596, 2011. doi: 10.7916/D8NZ8J07. URL http://ismir2011.ismir.net/papers/OS6-1.pdf.

[13] Jock A Blackard and Denis J Dean. Comparative accuracies of artificial neural networks and discriminant analysis in predicting forest cover types from cartographic variables. *Computers and electronics in agriculture*, 24(3):131–151, 1999. doi: 10.1016/S0168-1699(99)00046-0. URL https://doi.org/10.1016/S0168-1699(99)00046-0.

[14] Vladimir Braverman, Shaofeng H.-C. Jiang, Robert Krauthgamer, and Xuan Wu. Coresets for clustering in excluded-minor graphs and beyond. In Dániel Marx, editor, *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10 - 13, 2021*, pages 2679–2696. SIAM, 2021. doi: 10.1137/1.9781611976465.159. URL https://doi.org/10.1137/1.9781611976465.159.

[15] Vladimir Braverman, Vincent Cohen-Addad, Shaofeng H.-C. Jiang, Robert Krauthgamer, Chris Schwiegelshohn, Mads Bech Toftrup, and Xuan Wu. The power of uniform sampling for coresets. In *63rd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2022, Denver, CO, USA, October 31 - November 3, 2022*, pages 462–473. IEEE, 2022. doi: 10.1109/FOCS54457.2022.00051. URL https://doi.org/10.1109/FOCS54457.2022.00051.

[16] Ke Chen. On coresets for k-median and k-means clustering in metric and euclidean spaces and their applications. *SIAM J. Comput.*, 39(3):923–947, 2009. doi: 10.1137/070699007. URL https://doi.org/10.1137/070699007.

[17] Flavio Chierichetti, Nilesh N. Dalvi, and Ravi Kumar. Correlation clustering in mapreduce. In Sofus A. Macskassy, Claudia Perlich, Jure Leskovec, Wei Wang, and Rayid Ghani, editors, *The 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '14, New York, NY, USA - August 24 - 27, 2014*, pages 641–650. ACM, 2014. doi: 10.1145/2623330.2623743. URL https://doi.org/10.1145/2623330.2623743.

[18] Flavio Chierichetti, Ravi Kumar, Silvio Lattanzi, and Sergei Vassilvitskii. Fair clustering through fairlets. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 5029–5037, 2017. URL https://proceedings.neurips.cc/paper/2017/hash/978fce5bcc4eccc88ad48ce3914124a2-Abstract.html.

[19] Michael B. Cohen, Sam Elder, Cameron Musco, Christopher Musco, and Madalina Persu. Dimensionality reduction for k-means clustering and low rank approximation. In Rocco A. Servedio and Ronitt Rubinfeld, editors, *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*, pages 163–172. ACM, 2015. doi: 10.1145/2746539.2746569. URL https://doi.org/10.1145/2746539.2746569.

[20] Michael B. Cohen, Yin Tat Lee, Gary L. Miller, Jakub Pachocki, and Aaron Sidford. Geometric median in nearly linear time. In Daniel Wichs and Yishay Mansour, editors, *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18-21, 2016*, pages 9–21. ACM, 2016. doi: 10.1145/2897518.2897647. URL https://doi.org/10.1145/2897518.2897647.

[21] Vincent Cohen-Addad and Jason Li. On the fixed-parameter tractability of capacitated clustering. In *46th International Colloquium on Automata, Languages, and Programming, ICALP 2019, July 9-12, 2019, Patras, Greece*, pages 41:1–41:14, 2019. doi: 10.4230/LIPIcs.ICALP.2019.41. URL https://doi.org/10.4230/LIPIcs.ICALP.2019.41.

[22] Vincent Cohen-Addad and Chris Schwiegelshohn. On the local structure of stable clustering instances. In *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017, Berkeley, CA, USA, October 15-17, 2017*, pages 49–60, 2017. doi: 10.1109/FOCS.2017.14. URL https://doi.org/10.1109/FOCS.2017.14.

[23] Vincent Cohen-Addad, Silvio Lattanzi, Ashkan Norouzi-Fard, Christian Sohler, and Ola Svensson. Fast and accurate *k*-means++ via rejection sampling. *Advances in Neural Information Processing Systems*, 33:16235–16245, 2020. URL https://proceedings.neurips.cc/paper/2020/hash/babcff88f8be8c4795bd6f0f8cccca61-Abstract.html.

[24] Vincent Cohen-Addad, Silvio Lattanzi, Ashkan Norouzi-Fard, Christian Sohler, and Ola Svensson. Parallel and efficient hierarchical k-median clustering. In Marc'Aurelio Ranzato, Alina Beygelzimer, Yann N. Dauphin, Percy Liang, and

Jennifer Wortman Vaughan, editors, *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pages 20333–20345, 2021. URL https://proceedings.neurips.cc/paper/2021/hash/aa495e18c7e3a21a4e48923b92048a61-Abstract.html.

[25] Vincent Cohen-Addad, David Saulpic, and Chris Schwiegelshohn. A new coreset framework for clustering. In Samir Khuller and Virginia Vassilevska Williams, editors, *STOC '21: 53rd Annual ACM SIGACT Symposium on Theory of Computing, Virtual Event, Italy, June 21-25, 2021*. ACM, 2021. doi: 10.1145/3406325.3451022. URL https://doi.org/10.1145/3406325.3451022.

[26] Vincent Cohen-Addad, David Saulpic, and Chris Schwiegelshohn. Improved coresets and sublinear algorithms for power means in euclidean spaces. In Marc'Aurelio Ranzato, Alina Beygelzimer, Yann N. Dauphin, Percy Liang, and Jennifer Wortman Vaughan, editors, *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pages 21085–21098, 2021. URL https://proceedings.neurips.cc/paper/2021/hash/b035d6563a2adac9f822940c145263ce-Abstract.html.

[27] Vincent Cohen-Addad, Kasper Green Larsen, David Saulpic, and Chris Schwiegelshohn. Towards optimal lower bounds for k-median and k-means coresets. In Stefano Leonardi and Anupam Gupta, editors, *STOC '22: 54th Annual ACM SIGACT Symposium on Theory of Computing, Rome, Italy, June 20 - 24, 2022*, pages 1038–1051. ACM, 2022. doi: 10.1145/3519935.3519946. URL https://doi.org/10.1145/3519935.3519946.

[28] Vincent Cohen-Addad, Kasper Green Larsen, David Saulpic, Chris Schwiegelshohn, and Omar Ali Sheikh-Omar. Improved coresets for euclidean k-means. In *NeurIPS*, 2022. URL http://papers.nips.cc/paper_files/paper/2022/hash/120c9ab5c58ba0fa9dd3a22ace1de245-Abstract-Conference.html.

[29] Robson Leonardo Ferreira Cordeiro, Caetano Traina Jr., Agma Juci Machado Traina, Julio César López-Hernández, U Kang, and Christos Faloutsos. Clustering very large multi-dimensional datasets with mapreduce. In Chid Apté, Joydeep Ghosh, and Padhraic Smyth, editors, *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Diego, CA, USA, August 21-24, 2011*, pages 690–698. ACM, 2011. doi: 10.1145/2020408.2020516. URL https://doi.org/10.1145/2020408.2020516.

[30] Artur Czumaj and Christian Sohler. Sublinear-time approximation algorithms for clustering via random sampling. *Random Struct. Algorithms*, 30(1-2):226–256, 2007. doi: 10.1002/RSA.20157. URL https://doi.org/10.1002/rsa.20157.

[31] Yichuan Deng, Zhao Song, Yitan Wang, and Yuanyuan Yang. A nearly optimal size coreset algorithm with nearly linear time. *CoRR*, abs/2210.08361, 2022. doi: 10.48550/arXiv.2210.08361. URL https://doi.org/10.48550/arXiv.2210.08361.

[32] Dheeru Dua and Casey Graff. UCI machine learning repository, 2017. URL http://archive.ics.uci.edu/ml.

[33] Alina Ene, Sungjin Im, and Benjamin Moseley. Fast clustering using mapreduce. In Chid Apté, Joydeep Ghosh, and Padhraic Smyth, editors, *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Diego, CA, USA, August 21-24, 2011*, pages 681–689. ACM, 2011. doi: 10.1145/2020408.2020515. URL https://doi.org/10.1145/2020408.2020515.

[34] Georgios Exarchakis, Omar Oubari, and Gregor Lenz. A sampling-based approach for efficient clustering in large datasets. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2022, New Orleans, LA, USA, June 18-24, 2022*, pages 12393–12402. IEEE, 2022. doi: 10.1109/CVPR52688.2022.01208. URL https://doi.org/10.1109/CVPR52688.2022.01208.

[35] Jittat Fakcharoenphol, Satish Rao, and Kunal Talwar. A tight bound on approximating arbitrary metrics by tree metrics. In Lawrence L. Larmore and Michel X. Goemans, editors, *Proceedings of the 35th Annual ACM Symposium on Theory of Computing, June 9-11, 2003, San Diego, CA, USA*, pages 448–455. ACM, 2003. doi: 10.1145/780542.780608. URL https://doi.org/10.1145/780542.780608.

[36] Dan Feldman. Introduction to core-sets: an updated survey. *arXiv preprint arXiv:2011.09384*, 2020. doi: 10.1002/widm.1335.

[37] Dan Feldman and Michael Langberg. A unified framework for approximating and clustering data. In Lance Fortnow and Salil P. Vadhan, editors, *Proceedings of the 43rd ACM Symposium on Theory of Computing, STOC 2011, San Jose, CA, USA, 6-8 June 2011*, pages 569–578. ACM, 2011. doi: 10.1145/1993636.1993712. URL https://doi.org/10.1145/1993636.1993712.

[38] Hendrik Fichtenberger, Marc Gillé, Melanie Schmidt, Chris Schwiegelshohn, and Christian Sohler. Bico: Birch meets coresets for k-means clustering. In *European symposium on Algorithms*, pages 481–492. Springer, 2013.

[39] Sariel Har-Peled. *Geometric approximation algorithms*. Number 173. American Mathematical Soc., 2011. doi: 10.1090/surv/173.

[40] Sariel Har-Peled and Soham Mazumdar. On coresets for k-means and k-median clustering. In László Babai, editor, *Proceedings of the 36th Annual ACM Symposium on Theory of Computing, Chicago, IL, USA, June 13-16, 2004*, pages 291–300. ACM, 2004. doi: 10.1145/1007352.1007400. URL https://doi.org/10.1145/1007352.1007400.

[41] Qinghao Hu, Jiaxiang Wu, Lu Bai, Yifan Zhang, and Jian Cheng. Fast k-means for large scale clustering. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, pages 2099–2102, 2017. doi: 10.1145/3132847.3133091.

[42] Lingxiao Huang and Nisheeth K. Vishnoi. Coresets for clustering in euclidean spaces: importance sampling is nearly optimal. In Konstantin Makarychev, Yury Makarychev, Madhur Tulsiani, Gautam Kamath, and Julia Chuzhoy, editors, *Proccedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020, Chicago, IL, USA, June 22-26, 2020*, pages 1416–1429. ACM, 2020. doi: 10.1145/3357713.3384296. URL https://doi.org/10.1145/3357713.3384296.

[43] Lingxiao Huang, Shaofeng H.-C. Jiang, and Nisheeth K. Vishnoi. Coresets for clustering with fairness constraints. In *NeurIPS*, pages 7587–7598, 2019. URL https://proceedings.neurips.cc/paper/2019/hash/810dfbbebb17302018ae903e9cb7a483-Abstract.html.

[44] Lingxiao Huang, Jian Li, and Xuan Wu. Towards optimal coreset construction for (k, z)-clustering: Breaking the quadratic dependency on k. *CoRR*, abs/2211.11923, 2022. doi: 10.48550/arXiv.2211.11923. URL https://doi.org/10.48550/arXiv.2211.11923.

[45] Lingxiao Huang, Shaofeng H.-C. Jiang, and Jianing Lou. The power of uniform sampling for k-median. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett, editors, *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, volume 202 of *Proceedings of Machine Learning Research*, pages 13933–13956. PMLR, 2023. URL https://proceedings.mlr.press/v202/huang23j.html.

[46] Amit Kumar and Ravindran Kannan. Clustering with spectral norm and the k-means algorithm. In *51th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2010, October 23-26, 2010, Las Vegas, Nevada, USA*, pages 299–308, 2010. doi: 10.1109/FOCS.2010.35. URL http://dx.doi.org/10.1109/FOCS.2010.35.

[47] Michael Langberg and Leonard J. Schulman. Universal epsilon-approximators for integrals. In Moses Charikar, editor, *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2010, Austin, Texas, USA, January 17-19, 2010*, pages 598–607. SIAM, 2010. doi: 10.1137/1.9781611973075.50. URL https://doi.org/10.1137/1.9781611973075.50.

[48] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proc. IEEE*, 86(11):2278–2324, 1998. doi: 10.1109/5.726791. URL https://doi.org/10.1109/5.726791.

[49] Stuart P. Lloyd. Least squares quantization in PCM. *IEEE Trans. Inf. Theory*, 28(2):129–136, 1982. doi: 10.1109/TIT.1982.1056489. URL https://doi.org/10.1109/TIT.1982.1056489.

[50] Konstantin Makarychev, Yury Makarychev, and Ilya P. Razenshteyn. Performance of johnson-lindenstrauss transform for k-means and k-medians clustering. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019, Phoenix, AZ, USA, June 23-26, 2019*, pages 1027–1038, 2019. doi: 10.1145/3313276.3316350. URL https://doi.org/10.1145/3313276.3316350.

[51] Adam Meyerson, Liadan O'Callaghan, and Serge A. Plotkin. A k-median algorithm with running time independent of data size. *Mach. Learn.*, 56(1-3):61–87, 2004. doi: 10.1023/B:MACH.0000033115.78247.F0. URL https://doi.org/10.1023/B:MACH.0000033115.78247.f0.

[52] Luis Moreira-Matias, Michel Ferreira, Joao Mendes-Moreira, L. L., and J. J. Taxi Service Trajectory - Prediction Challenge, ECML PKDD 2015. UCI Machine Learning Repository, 2015. DOI: https://doi.org/10.24432/C55W25.

[53] N/A, 1990. URL https://archive.ics.uci.edu/ml/datasets/US+Census+Data+(1990).

[54] R. Ostrovsky, Y. Rabani, L. J. Schulman, and C. Swamy. The effectiveness of Lloyd-type methods for the k-means problem. *J. ACM*, 59(6):28, 2012. doi: 10.1145/2395116.2395117. URL http://doi.acm.org/10.1145/2395116.2395117.

[55] Pixabay. Shooting star sky night royalty-free vector graphic, 2023. URL https://pixabay.com/vectors/shooting-star-sky-night-dark-stars-2024127/. [Online; accessed Jan 8th 2024].

[56] Melanie Schmidt, Chris Schwiegelshohn, and Christian Sohler. Fair coresets and streaming algorithms for fair k-means. In *Approximation and Online Algorithms - 17th International Workshop, WAOA 2019, Munich, Germany, September 12-13, 2019, Revised Selected Papers*, pages 232–251, 2019. doi: 10.1007/978-3-030-39479-0_16. URL https://doi.org/10.1007/978-3-030-39479-0_16.

[57] Chris Schwiegelshohn and Omar Ali Sheikh-Omar. An empirical evaluation of k-means coresets. In Shiri Chechik, Gonzalo Navarro, Eva Rotenberg, and Grzegorz Herman, editors, *30th Annual European Symposium on Algorithms, ESA 2022, September 5-9, 2022, Berlin/Potsdam, Germany*, volume 244 of *LIPIcs*, pages 84:1–84:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. doi: 10.4230/LIPIcs.ESA.2022.84. URL https://doi.org/10.4230/LIPIcs.ESA.2022.84.

[58] Tian Zhang, Raghu Ramakrishnan, and Miron Livny. BIRCH: an efficient data clustering method for very large databases. pages 103–114, 1996. doi: 10.1145/233269.233324. URL https://doi.org/10.1145/233269.233324.

Received October 2023; revised January 2024; accepted February 2024