OneBatchPAM: A Fast and Frugal K-Medoids Algorithm

Antoine de Mathelin[1], **Nicolas Enrique Cecchi**[1], **François Deheeger**[2], **Mathilde Mougeot**[1], **Nicolas Vayatis**[1]

[1]Centre Borelli, Université Paris-Saclay, CNRS, ENS Paris-Saclay
[2]Michelin
antoine.de_mathelin@ens-paris-saclay.fr

Abstract

This paper proposes a novel k-medoids approximation algorithm to handle large-scale datasets with reasonable computational time and memory complexity. We develop a local-search algorithm that iteratively improves the medoid selection based on the estimation of the k-medoids objective. A single batch of size $m \ll n$ provides the estimation, which reduces the required memory size and the number of pairwise dissimilarities computations to $\mathcal{O}(mn)$, instead of $\mathcal{O}(n^2)$ compared to most k-medoids baselines. We obtain theoretical results highlighting that a batch of size $m = \mathcal{O}(\log(n))$ is sufficient to guarantee, with strong probability, the same performance as the original local-search algorithm. Multiple experiments conducted on real datasets of various sizes and dimensions show that our algorithm provides similar performances as state-of-the-art methods such as FasterPAM and BanditPAM++ with a drastically reduced running time.

Code — https://github.com/antoinedemathelin/obpam

Introduction

The k-medoids problem consists in choosing k medoids from a set of n points $\mathcal{X}_n$, minimizing the sum of the pairwise dissimilarities between the n points and their nearest medoid. This problem has many uses in machine learning, in particular for clustering, subset selection and active learning (Bhat 2014; Wei, Iyer, and Bilmes 2015; Kaushal et al. 2019; de Mathelin et al. 2021). The k-medoids problem is related to k-medians, k-means and facility location (Schubert and Rousseeuw 2021). One specificity of k-medoids is to consider generic dissimilarities (non-necessarily metric). In machine learning applications, the dissimilarity function can involve heavy computational costs, especially when computed between complex data types such as images, texts, or time series.

The k-medoids problem is a discrete optimization problem known to be NP-hard (Kariv and Hakimi 1979), for which a wide variety of approximation algorithms have been developed. Many k-medoids approximations are greedy or local-search algorithms, which improve a medoid selection sequentially by either adding or removing a medoid or swapping one medoid with another data point (Dohan, Karp, and Matejek 2015). The main local-search approach considered by the operations research communities is called PAM (Partitioning Around Medoid) (Kaufman and Rousseeuw 1987; Kaufman 1990). This algorithm starts from an initial choice of k points (potentially greedily selected) and then performs a series of "swaps". The state-of-the-art PAM algorithms are the FastPAM variants (Schubert and Rousseeuw 2021; Schubert and Lenssen 2022).

A major drawback of these approximation algorithms is the computational burden encountered for large values of n. Indeed, the main algorithms require the computation and in-memory conservation of pairwise dissimilarities between the n points, resulting in a complexity of $\mathcal{O}(n^2)$. Nowadays, with the rise of Big Data, and the focus on reducing computational resources, there is a strong incentive to build algorithms that overcome this $\mathcal{O}(n^2)$ limitation.

Subsampling is a straightforward solution to reduce the number of dissimilarity calculations. The idea is to use an approximation algorithm (like PAM) on a subsample of size $m \ll n$ selected among the n data points, resulting in a reduction of the time and memory complexities from $\mathcal{O}(n^2)$ to $\mathcal{O}(m^2)$. Previous works have proven that this simple approach yields appealing statistical guarantees over the approximation error for relatively small batch size m (Mishra, Oblinger, and Pitt 2001; Thorup 2005; Mettu and Plaxton 2004; Meyerson, O'callaghan, and Plotkin 2004; Huang, Jiang, and Lou 2023; Guha and Mishra 2016; Czumaj and Sohler 2007). In this category of methods, the CLARA algorithm (Clustering LARge Applications) (Kaufman 1986; Kaufman and Rousseeuw 2008) is the most commonly used. The main drawback of the subsampling approach is the loose approximation of considering only the medoid candidates in the m subsampled data points, resulting in worse clustering quality (Tiwari et al. 2020). A recent method, BanditPAM, leverages Bandit algorithms to deal with this limitation (Tiwari et al. 2020, 2023). BanditPAM keeps the n data points as potential medoid candidates but only computes the dissimilarities for data points with high medoid potential, thus reducing the number of pairwise dissimilarity computations to $\mathcal{O}(n \log(n))$ for one medoid selection or one swap step of the PAM algorithm. Although BanditPAM provides a medoid selection close to PAM (in terms of k-medoids objective), the Bandit-based framework requires the computation of new pairwise dissimilarities at each medoid selection,

which then results in computing $\mathcal{O}(Tn\log(n))$ pairwise dissimilarities, with T the number of iterations of the algorithm.

In this paper, we propose an alternative approach to address the $\mathcal{O}(n^2)$ limitations of local-search k-medoids algorithms. To avoid computing new pairwise dissimilarities at each swap, we only compute the dissimilarities between the n data points and a single batch of size m. Our theoretical analysis shows that $m = \mathcal{O}(\log(n))$ is sufficient to guarantee similar performances as FasterPAM with strong probability. Our algorithm called OneBatchPAM provides a $\mathcal{O}(T)$ speedup of time complexity compared to BanditPAM and a $\mathcal{O}(n/\log(n))$ speedup compared to FasterPAM for similar performances. We show through several experiments, conducted on real datasets, that OneBatchPAM proposes an efficient time / objective trade-off compared to multiple k-medoids algorithms.

Related Works

Approximation algorithms for k-medoids

The k-medoids problem is related to facility locations, k-medians (or p-medians) and k-means problems. A detailed comparison of these problems is given in (Schubert and Rousseeuw 2021). In a nutshell, the main k-medoids particularity is to consider generic dissimilarities (non-necessarily metric as in k-medians) and to constrain the k medoids to belong to the dataset $\mathcal{X}_n$ (unlike k-means). k-medoids can then be seen as a special case of the facility location problem, where at most k facilities, belonging to the set of clients, can be opened with cost zero. The metric k-medoids problem is often considered, in which case the problem is similar to k-medians over discrete metric space (Schubert and Rousseeuw 2021).

As solving the k-medoids problem is NP-hard, many algorithms have been developed to provide approximations in polynomial running time[1] (Kaufman 1990; Charikar et al. 1999; Li and Svensson 2013; Bhat 2014). A "naive" greedy approach selects the medoids sequentially by solving a 1-medoid problem at each iteration. This approach is simple to implement and yields relatively good results in practice, but its theoretical approximation error in $\mathcal{O}(n)$ is quite large (Dohan, Karp, and Matejek 2015). It can be improved to $\mathcal{O}(\log(n))$ by the reverse greedy approach that starts with n medoids and removes them one by one until reaching k medoids (Chrobak, Kenyon, and Young 2006). The most notable improvement to the greedy approach is the PAM algorithm (Kaufman and Rousseeuw 1987; Kaufman 1990). It greedily initializes the set of medoids and then performs a series of swaps from one medoid to one non-medoid that improve the total objective. In the metric case, this local search approach provides a constant approximation ratio of 5 which can be reduced to $3 + \epsilon$ when swapping multiple medoids at each iteration (Arya et al. 2001). Assuming pairwise dissimilarities are precomputed, the time complexity of the seminal PAM algorithm is $\mathcal{O}(Tkn^2)$, with T the number of swap steps. A notable recent improvement, called FastPAM (Schubert and Rousseeuw 2021; Schubert and Lenssen 2022), reduces the PAM's time complexity to $\mathcal{O}(Tn^2)$ by using a smart decomposition of the swap evaluation (Schubert and Rousseeuw 2021). We emphasize that the PAM algorithm and its variants are perhaps the most widespread approximation algorithms for k-medoids. It provides an appealing trade-off between approximation error and time complexity. Although its theoretical approximation ratio is 5, the error is often much smaller in practical use-cases (less than 2% (Schubert and Rousseeuw 2021)).

When the dissimilarity evaluation is costly, and/or when the available memory is restricted. All aforementioned algorithms are limited by the $\mathcal{O}(n^2)$ pairwise dissimilarities computation cost and by the $\mathcal{O}(n^2)$ memory requirement to store the computed dissimilarities. Our work then focuses on reducing the time complexity of PAM while keeping similar performance. We therefore do not consider algorithms that propose improvement over the PAM performance at the price of additional computational efforts, such as (Li and Svensson 2013; Byrka et al. 2017; Ren, Hua, and Cao 2022).

Subsampling Methods

Subsampling consists in performing a k-medoids algorithm on a subsample $\mathcal{X}_m$ of the original dataset $\mathcal{X}_n$, of size $m \ll n$. For instance, the CLARA algorithm (Kaufman and Rousseeuw 2008) uses PAM on a subsample $\mathcal{X}_m$. It has been shown that a k-medoids algorithm with constant approximation can be derived, with great probability, using a random uniform subsample of size $m \simeq \mathcal{O}(k\log(n))$ (Mishra, Oblinger, and Pitt 2001). The required size of the subsample has been further reduced to $\mathcal{O}(k\log(k))$ with deeper analysis (Meyerson, O'callaghan, and Plotkin 2004; Czumaj and Sohler 2007), which is independent of n. In this perspective, the CLARA algorithm proposes the heuristic $m = 40 + 2k$ for the subsample's size (Kaufman and Rousseeuw 2008). With such a setting, this subsampling method can drastically reduce the number of pairwise dissimilarity computations from $\mathcal{O}(n^2)$ to $\mathcal{O}(k^2)$. CLARA repeatedly computes a k-medoids approximation over multiple random subsamples drawn uniformly from $\mathcal{X}_n$ and selects the best set of k medoids based on the evaluation over the whole dataset $\mathcal{X}_n$. Although only $\mathcal{O}(k^2)$ dissimilarity computations are needed to perform PAM over the subsample, one evaluation step requires to compute nk dissimilarities, resulting in a $\mathcal{O}(Tpnk)$ time complexity, with T the number of subsamples. The cost of the evaluation step can be mitigated by evaluating the medoid set over another subsample from $\mathcal{X}_n$ (Meyerson, O'callaghan, and Plotkin 2004), in the same spirit as hold-out validation in machine learning.

The primary drawback of the subsampling approach is the approximation error, which is theoretically twice as large as that of performing the same k-medoids approximation on the full dataset of n data points (Mishra, Oblinger, and Pitt 2001; Meyerson, O'callaghan, and Plotkin 2004; Czumaj and Sohler 2007). In practice, this leads to a noticeable decline in performance.

A recent method BanditPAM (Tiwari et al. 2020, 2023) proposes an interesting idea based on Bandit evaluation of the swap and initialization steps in PAM. At each step, the best local improvement is estimated using multi-armed ban-

[1] i.e., a constant polynomial degree independent of k

dit techniques. BanditPAM therefore does not need to compute all pairwise dissimilarities but only the ones useful to find the best swap. The drawback of such an approach is to compute new dissimilarities at each step resulting in $\mathcal{O}((T+k)n\log(n))$ dissimilarity computations, with T the number of swap evaluations. In this work, we propose to instead compute all pairwise distance between the n data points and a batch of size $m = \mathcal{O}(\log(n))$. The same dissimilarities are used to evaluate all swap steps, resulting in $\mathcal{O}(n\log(n))$ dissimilarity computations.

k-means++ as a proxy for k-medoids

k-means++ is first designed as a seeding algorithm for k-means. It iteratively samples data points from $\mathcal{X}_n$ with a probability proportional to the distance raised to the power p to the already sampled points for any ℓ_p distance. As the output of k-means++ is a set of cluster centers in $\mathcal{X}_n$, and since the objective of k-means is the same as k-medoids when considering the Euclidean distance, this algorithm can be used as a natural proxy for k-medoids. The k-means++ algorithm provides a $\mathcal{O}(\log(k))$-approximation for k-means and k-medians (Arthur, Vassilvitskii et al. 2007), which is generally worse than PAM but it only require $\mathcal{O}(kn)$ pairwise dissimilarity computations instead of $\mathcal{O}(n^2)$.

Since the seminal work of (Arthur, Vassilvitskii et al. 2007), two primary directions have been pursued to improve k-means++: enhancing the approximation error and reducing the time complexity. To improve the approximation, local-search algorithms are employed. These algorithms typically involve a random selection process similar to k-means++, followed by a swap if the new selection yields a better clustering outcome. For instance, single-swap local-search methods require $\mathcal{O}((Z+k)n)$ pairwise distance computations, with Z the number of swap steps, and $\mathcal{O}(Zkn)$ additional operations (Lattanzi and Sohler 2019). Multiple-swap approaches can further refine the clustering but at a higher computational cost, involving $\mathcal{O}((Zt+k)n)$ distance computations and $\mathcal{O}(Znk^{2t-1})$ additional operations, with t the number of simultaneous swaps (Beretta et al. 2024; Huang et al. 2024). To accelerate the process, (Bachem et al. 2016) introduced kmc2, which speeds up k-means++ to $\mathcal{O}(Lk^2)$ distance computations, with L a method's specific parameter. Other methods leverage the specificity of Euclidean distance. For example, by projecting the data onto one dimension (Charikar et al. 2023), or leveraging specific nearest neighbor structure (Cohen-Addad et al. 2020) (Pelleg and Moore 1999).

Coreset for k-medians

According to (Feldman 2020), a coreset is a data summarization technique that selects a subsample from a large dataset, preserving the information needed to perform specific tasks such as linear regression or clustering. Specifically, given a dataset $\mathcal{X}_n$, an objective function $\mathcal{L}$, and a set of queries $\mathcal{Q}$, a coreset $\mathcal{X}_m$ is a subsample of $\mathcal{X}_n$ for which any query $q \in \mathcal{Q}$ yields a similar objective value when computed on the coreset as when computed on the entire dataset, i.e., $\mathcal{L}(q, \mathcal{X}_n) \simeq \mathcal{L}(q, \mathcal{X}_m)$. In the context of k-medians clustering, the clustering cost of any k centers computed on a coreset is approximately the same as the cost of these centers on the entire dataset.

While there is no consensual definition of a coreset, it generally refers to a "strong coreset" where the objective computed on the coreset is a $(1+\epsilon)$-approximation of the objective computed over the entire dataset for any query (e.g., any set of k centers). Coreset is sometimes equated with subsampling when the set of queries is restricted to the coreset itself (Huang, Jiang, and Lou 2023; Har-Peled and Mazumdar 2004). When the set of queries includes any combination of k centers from the entire dataset, coresets are similar in spirit to OnebatchPAM. However, the coreset literature focuses on constructing sets that provide a $(1+\epsilon)$-approximation for any query, whereas OnebatchPAM focuses on achieving results comparable to PAM.

Various coresets for k-medians have been proposed, aiming to find the minimal size that guarantees the $(1+\epsilon)$-approximation for any set of k centers (Har-Peled and Mazumdar 2004; Chen 2009). The best-known result for discrete metric k-medians (similar to metric k-medoids) is provided by (Feldman 2020), with $m = \mathcal{O}(k\log(n)\epsilon^{-2})$. However, constructing such a coreset has a running time of $\mathcal{O}(pnk)$, with p the data dimension. This has been improved by (Cohen-Addad, Saulpic, and Schwiegelshohn 2021b), which provides a similar-sized coreset with a running time of $\mathcal{O}(nk)$. The coreset size $m = \mathcal{O}(k\log(n)\epsilon^{-2})$ has been shown to be the minimal size to guarantee the $(1+\epsilon)$-approximation (Cohen-Addad et al. 2022).

This size can be reduced when the constraints are relaxed, leading to what is known as a weak coreset (Feldman and Langberg 2011). A weak coreset guarantees the $(1+\epsilon)$-approximation for only a subset of queries (Feldman 2020; Jaiswal and Kumar 2024). Other definitions of coresets include those with additive and multiplicative error approximations, such as lightweight coresets (Bachem, Lucic, and Krause 2018). Smaller coresets yielding the $(1+\epsilon)$-approximation guarantee can be constructed when considering Euclidean space (Cohen-Addad, Saulpic, and Schwiegelshohn 2021a; Feldman and Langberg 2011), or constrained problems, such as capacitated clustering (uniform distribution between clusters) (Huang, Jiang, and Lou 2023; Braverman et al. 2022) and fair constraint clustering (Schmidt, Schwiegelshohn, and Sohler 2020).

From PAM to OneBatchPAM

Notations

The four parameters $n, k, p, m \in \mathbb{N}^*$ respectively denote the number of data points, the number of medoids, the problem dimension and the batch size. We consider the space $\mathcal{X}$ with $\mathcal{X} \subset \mathbb{R}^p$ and $d : \mathcal{X} \times \mathcal{X} \to \mathbb{R}_+$ a measure of dissimilarity over $\mathcal{X}$. We consider the set $\mathcal{X}_n = \{x_1, ..., x_n\}$ of n data points in $\mathcal{X}$. We denote $\mathcal{P}_k(\mathcal{X}_n)$ the set of all subsets of $\mathcal{X}_n$ of size k. We aim at solving the k-medoids selection problem:

$$\min_{\mathcal{M} \in \mathcal{P}_k(\mathcal{X}_n)} \sum_{i=1}^{n} d(x_i, \mathcal{M}), \tag{1}$$

with $d(x_i, \mathcal{M}) = \min_{\tilde{x} \in \mathcal{M}} d(x_i, \tilde{x})$. We denote by $\mathcal{L}$ the objective function, such that $\mathcal{L}(\mathcal{M}) = \frac{1}{n}\sum_{x \in \mathcal{X}_n} d(x, \mathcal{M})$

for any $\mathcal{M} \in \mathcal{P}_k(\mathcal{X}_n)$. In the following, we consider the common assumption that one dissimilarity computation requires $\mathcal{O}(p)$ time complexity (Tiwari et al. 2020; Schubert and Rousseeuw 2021).

PAM, FastPAM and FasterPAM

The local-search approximation algorithm called PAM, performs several "swap" steps that progressively improve the medoid selection. This is formally described by the following recurrence equation:

$$\mathcal{M}_{t+1} = \underset{\substack{x \in \mathcal{M}_t, \\ x' \in \mathcal{X}_n \setminus \mathcal{M}_t}}{\operatorname{argmin}} \sum_{i=1}^{n} d\left(x_i, (\mathcal{M}_t \setminus \{x\}) \cup \{x'\}\right). \quad (2)$$

For any $t \in \{0, ..., T-1\}$, with T the number of iterations. In the original PAM algorithm, the initial medoid set $\mathcal{M}_0$ is built using a greedy algorithm (Kaufman 1990). The swap step described in Equation (2) consists in removing one medoid from $\mathcal{M}_t$ and adding a non-medoid from $\mathcal{X}_n \setminus \mathcal{M}_t$. This algorithm theoretically provides a 5-approximation (Arya et al. 2001). In practical scenarios, however, the approximation error is often below 2% (Schubert and Rousseeuw 2021).

As highlighted by Equation (2) the "naive" approach to perform a swap step requires computing the sum of dissimilarities for every swap pair $(x, x') \in \mathcal{M}_t \times \mathcal{X}_n \setminus \mathcal{M}_t$, which leads to $\mathcal{O}(kn^2)$ operations to perform one swap. The FastPAM algorithm introduced in (Schubert and Rousseeuw 2021) proposes a modification of PAM that yields a $\mathcal{O}(k)$ speed up. The main idea lies in the fact that for each x_i, only the removal of the nearest medoid will modify the value of $d(x_i, \mathcal{M}_t)$. Therefore, only one pass through $\mathcal{X}_n$ is needed to compute the impact of removing one medoid for all k medoids. The complexity of one swap step then only requires $\mathcal{O}(n^2)$ operations. Moreover, (Schubert and Rousseeuw 2021) shows that random initializations of the medoids lead to similar results as the greedy initialization but save $\mathcal{O}(kn^2)$ operations. Finally, additional speedups are derived by eagerly swapping a medoid with a non-medoid as soon as an improvement is found. In theory, eager swapping still requires $\mathcal{O}(n^2)$ operations for one swap but, in practice, it significantly speeds up the algorithm. The FastPAM algorithm with these additional improvements is called FasterPAM.

As noticed by (Tiwari et al. 2020), the main drawback of FastPAM and FasterPAM is that they require to compute every dissimilarity between each pair of data points in $\mathcal{X}_n$, with complexity $\mathcal{O}(pn^2)$. A solution proposed by (Schubert and Rousseeuw 2021) is FasterCLARA which uses FasterPAM on subsamples of $\mathcal{X}_n$. However, this solution comes with large approximation error in practice. To overcome this issue, we propose the OneBatchPAM algorithm.

OneBatchPAM

The OneBatchPAM idea is the following: for any x, it is not necessary to compute every distance to every x_i to perform the exact same swaps as FasterPAM. An estimation of the objectives on a subsample is sufficient. Theorem 1 will show that only a subsample of size $m = \mathcal{O}(\log(n))$ is needed to find the same series of swaps as FasterPAM with great probability.

Formally, OneBatch involves choosing a subsample $\mathcal{X}_m = \{x_{\sigma(1)}, ..., x_{\sigma(m)}\}$ drawn from $\mathcal{X}_n$, with $\sigma : [\![1, m]\!] \to [\![1, n]\!]$ the mapping indice function. The subsample $\mathcal{X}_m$ is used to estimate the best swap to perform, such that:

$$\mathcal{M}_{t+1} = \underset{\substack{x \in \mathcal{M}_t, \\ x' \in \mathcal{X}_n \setminus \mathcal{M}_t}}{\operatorname{argmin}} \sum_{j=1}^{m} d\left(x_{\sigma(j)}, (\mathcal{M}_t \setminus \{x\}) \cup \{x'\}\right). \quad (3)$$

Compared to Equation (2), the sum is now only computed over $\mathcal{X}_m$. This modification drastically reduces the time complexity while keeping similar performances as FasterPAM with high probability as proven in Theorem 1 and Corollary 2.

It must be underlined that Equation (3) is not equivalent to subsampling as the search space is still $\mathcal{X}_n$. In subsampling methods, such as CLARA, we would have $x' \in \mathcal{X}_m \setminus \mathcal{M}_t$ instead of $x' \in \mathcal{X}_n \setminus \mathcal{M}_t$. This difference has a significant impact on the approximation error. By reducing the search space to $\mathcal{X}_m \setminus \mathcal{M}_t$, subsampling methods multiply by two the theoretical approximation error and, in practice, degraded performances are indeed observed.

Theorem 1. *Let $\mathcal{X}_m$ be a subsample uniformly drawn from $\mathcal{X}_n$. Let $D = \max_{(x,x') \in \mathcal{X}_n} d(x, x')$ and Δ be the smallest difference between two objectives computed by FasterPAM. Then, for any $\delta \in]0, 1]$, the OneBatchPAM algorithm returns the same set of medoid as FasterPAM with probability at least $1 - \delta$ if:*

$$m \geq \frac{4D^2}{\Delta^2} \log\left(\frac{2Tn}{\delta}\right). \quad (4)$$

Where $\Delta = \min_{t \in [\![0,T]\!]} \min_{\substack{x \in \mathcal{M}_t, \\ x' \in \mathcal{X}_n \setminus \mathcal{M}_t}} |\mathcal{L}(\mathcal{M}_t) - \mathcal{L}(\mathcal{M}_t \setminus \{x\} \cup \{x'\})|$

Proof. The proof follows the same framework as the proof of Theorem 1 in (Tiwari et al. 2020). It consists in finding the minimal sample size which guarantees that the statistical error on the objectives remains smaller than the smallest objective difference, Δ, with high probability. Consequently, OneBatchPAM performs the same swaps as FasterPAM. The detailed proof is reported in the supplementary materials. □

As stated by Theorem 1, the dependence of m with respect to n is only $m = \mathcal{O}(\log(n))$. This implies a drastic reduction of the time complexity as formally described in the following corollary.

Corollary 2. *The OneBatch PAM algorithm returns the same set of medoids as FasterPAM with arbitrarily high probability with time complexity:*

$$\mathcal{O}\left((p + T)n \log(n)\right). \quad (5)$$

Table 1 provides a detailed comparison of OneBatchPAM's complexity against other algorithms. OneBatchPAM achieves a complexity gain of $\mathcal{O}(n/\log(n))$ over FasterPAM due to subsampling, and at least a $\mathcal{O}(T)$ improvement

over BanditPAM++, as it avoids computing new dissimilarities at each swap step. While OneBatchPAM may require more computational time compared to subsampling and k-means++, it offers a superior approximation error factor. It is important to note that the values in Table 1 are theoretical; in practical scenarios, the performance comparison between methods can vary. For example, the approximation errors for PAM-based algorithms are often significantly lower than 5.

Algorithm	Complexity	Approximation
FasterPAM	$(p+T)n^2$	5
BanditPAM++	$p(T+k)n\log(n)$	5
OneBatchPAM	$(p+T)n\log(n)$	5
FasterCLARA	$I\left((p+T)k^2+pkn\right)$	10
k-means++	pkn	$\log(k)$

Table 1: Summary of theoretical time complexity and approximation error. T is the number of swaps iterations and I the number of subsamples.

How many iterations T are needed? Generally, the larger the value of T, the better the objective, but this also increases the time complexity. It is important to note that the algorithms may terminate before reaching T swaps if a local minimum is attained. According to (Tiwari et al. 2023) and (Schubert and Rousseeuw 2021), in practice, the required number of swaps is typically $\mathcal{O}(k)$. If a threshold ϵ on the improvement is set instead of a maximum number of iterations, such that the algorithm terminates when no swap is $1-\epsilon$ better than the current medoid selection, then the number of swaps is at most $T = \mathcal{O}(\log(n)/\epsilon)$.

How $\mathcal{X}_m$ should be sampled? Theorem 1 demonstrates that uniform sampling is sufficient to obtain good guarantees with a relatively small subset. However, a natural question arises: can we improve this with a more specific selection method? One initial approach consists in modifying the dissimilarity between the subsampled points and themselves as follows: $d(x_{\sigma(j)}, x_{\sigma(j)}) = +\infty$ for any $j \in \{1,\dots,m\}$. We empirically observed that this adjustment prevents the medoid selection from being biased toward the subsampled data points. A second approach is to reweight the uniform sample to correct any potential sample bias. Since all distances between $\mathcal{X}_n$ and $\mathcal{X}_m$ are computed to perform OneBatchPAM, we recommend using the nearest neighbor sample bias correction method from (Loog 2012). In this method, the importance of the data point $x_{\sigma(j)}$ is proportional to the number of data points in $\mathcal{X}_n$ whose nearest neighbor in $\mathcal{X}_m$ is $x_{\sigma(j)}$. Additionally, specific sampling techniques, such as those used to build coresets, may also be considered (Bachem, Lucic, and Krause 2018).

Discussion and Limitations

Minimum sample size of OneBatchPAM derived in Theorem 1. The factor $1/\Delta$ in the sample size lower bound also appears in the theoretical time complexity of BanditPAM. It is implicitly assumed that the minimum objective difference, Δ, is not null (Tiwari et al. 2020). The inverse proportionality between m and Δ^2 indicates that OneBatchPAM may require a large subsample to perform the exact same swaps as FasterPAM if two objectives are close. This can happen if two data points $x, x' \in \mathcal{X}_n$ are close. In that case, OneBatchPAM may estimate that adding x to the set of medoids instead of x' is more efficient while FatserPAM may do the opposite. However, as the difference between both objectives is small, OneBatchPAM will likely return a set of medoids with close performance to the one of FasterPAM. This is confirmed in our empirical experiments where OneBatchPAM provides close objectives compared to FasterPAM (around 2% error) but not exactly the same. We emphasize that the purpose of Theorem 1 is essentially to highlight the dependence of m relative to n. Indeed, many upper bound approximations are involved in the derivation of the Theorem's result, hence using the exact value of Equation (4) for m may be disproportionate. In practice, we do not estimate the ratio D/Δ to set the sample size, but instead choose a value proportional to $\log(n)$.

It is interesting to notice that the minimum sample size for OneBatchPAM does not directly depend on the number of medoids k. However, this dependence is somehow hidden in the number of swap steps T. As highlighted by (Schubert and Rousseeuw 2021), when starting with a random medoid selection, one can expect at least k swaps before reaching a local minimum.

Comparison to BanditPAM and memory limitations of OneBatchPAM. Both BanditPAM and OneBatchPAM rely on the estimation of the k-medoids objective to determine which swap to perform. However, they consider two different approaches for estimating this objective. BanditPAM gradually improves the objective's estimation of swap pair candidates using mini-batches while reducing the set of candidates as the estimation becomes more accurate. The process is repeated after each swap, as the update of the medoid set modifies the swap pairs' evaluation. This leads to a linear increase in pairwise dissimilarity computations relative to the number of iterations. In contrast, OneBatchPAM computes all pairwise dissimilarities between the entire dataset $\mathcal{X}_n$ and a subsample $\mathcal{X}_m$ only once, using these precomputed values for each swap step. Consequently, it avoids the linear scaling of dissimilarity computations with the number of iterations. It should be noted that this computational load reduction comes with an increase in memory consumption. Indeed, BanditPAM only requires $\mathcal{O}(n)$ memory space while OneBatchPAM needs $\mathcal{O}(n\log(n))$. Nevertheless, this memory usage is significantly more efficient than the $\mathcal{O}(n^2)$ memory requirement of FasterPAM.

Comparison to coresets. As discussed in the related works section, OneBatchPAM is closely associated with coresets used in the context of k-medians. The coresets literature essentially focuses on constructing subsets that provide a $(1+\epsilon)$-approximation for any k-medoids selection. This imposes a stronger constraint compared to OneBatchPAM, which focuses on achieving results similar to those of the PAM algorithm. This explains why the minimal sample size for OneBatchPAM $m = \mathcal{O}(\log(n))$ is smaller than the minimal size for coresets for k-medians clustering with discrete metric spaces, $m = \mathcal{O}(k\log(n)\epsilon^{-2})$ (Cohen-Addad

et al. 2022). It is important to note, however, that the sample size for OneBatchPAM is derived from a uniform sample $\mathcal{X}_m$. Leveraging coreset construction techniques could potentially further reduce the required sample size and, consequently, the time complexity of OneBatchPAM.

Overfitting for highly imbalanced datasets. Overfitting is a potential risk for OneBatchPAM, especially when the batch is not representative of the full dataset. Overfitting issues especially arise in situations involving highly imbalanced datasets. For instance, if a small subset of points are very far from all others. In such a case, there is a low probability that any neighbors of these distant points will be included in the batch, potentially leaving these points "not covered" by any medoid at the end of the OneBatchPAM algorithm. A potential future improvement to our approach could be to construct the batch progressively, leveraging the computed distances to identify imbalances in the dataset and mitigate the issue by selecting data points that improve the "representativeness" of the batch.

Experiments

We conduct several experiments on real datasets to compare OneBatchPAM with state-of-the-art k-medoids algorithms in practical scenarios. Our implementation of OneBatchPAM is coded in Python with the Cython module. The experiments are run on a 8G RAM computer with 4 cores. The source code of the experiments is available on GitHub[2].

Datasets and settings

We conduct the experiments on the MNIST and CIFAR10 image datasets (LeCun, Cortes, and Burges 1994; Krizhevsky, Hinton et al. 2009) and 8 UCI datasets (Dua and Graff 2017), arbitrarily selected, with various sizes and dimensions (cf. Table 2). The ℓ_1 distance is used as the dissimilarity function. Experiments are performed for different values of k in $\{10, 50, 100\}$. Each experiment is repeated 5 times to compute the standard deviations.

We divide the datasets into two categories respectively called "small scale" and "large scale" to account for the fact that some algorithms cannot provide a medoid selection in reasonable time for datasets above $\sim$ 50000 instances. In particular, FasterPAM is not able to handle the size of the MNIST dataset (Schubert and Rousseeuw 2021).

Small Scale			Large Scale		
Dataset	n	p	Dataset	n	p
abalone	4,176	8	CIFAR	50,000	3072
bankruptcy	6,819	96	MNIST	60,000	784
mapping	10,545	28	dota2	92,650	117
drybean	13,611	16	gas	416,153	9
letter	19,999	16	covertype	581,011	55

Table 2: Datasets Summary. n and p are respectively the dataset's size and dimension.

[2]https://github.com/antoinedemathelin/obpam

Competitors and Hyper-parameters

The following two kinds of competitors are considered

- **PAM Algorithms**. we consider the PAM variants: FasterPAM, BanditPAM++ and FasterCLARA, as well as the Alternate approach (Park and Jun 2009) although it is not formally a PAM method. We use the official implementations of BanditPAM++[3] (Tiwari et al. 2023). The other algorithms are found in the Python library `kmedoids`[4], providing the official implementation of FasterPAM (Schubert and Lenssen 2022).
- **k-means++ Algorithms**. We consider the original k-means++ algorithms and the two variants introduced in the related works: kmc2 (Bachem et al. 2016) and k-means++ with local-search (LS-k-means++) (Lattanzi and Sohler 2019).

If nothing else is specified the default hyperparameters are selected for the method. For BanditPAM++, we consider the three different settings of swap iterations: $T \in \{0, 2, 5\}$. We noticed that larger values of this parameter lead to excessive running time. For FasterCLARA we consider two different settings for the number of subsampling repetitions: $I \in \{5, 50\}$. The sample size is set to $m = 80 + 4k$ as suggested in (Schubert and Rousseeuw 2021). Three different chain lengths are considered for kmc2: $L = \{20, 100, 200\}$ and two different number of local search iterations for LS-k-means++: $Z = \{5, 10\}$. When different values of a parameter P are used for an algorithm Alg, we denote the corresponding variants by Alg-P.

For OneBatchPAM, we use a sample size of $m = 100 \log(kn)$. The four following subsampling techniques introduced in Section are considered: **Unif**: uniform sampling; **Debias**: uniform sampling with $d(x_{\sigma(j)}, x_{\sigma(j)}) = +\infty$ for any $j \in \{1, \dots, m\}$; **NNIW**: uniform sampling with nearest-neighbor importance weighting and **LWCS**: sample built through the "lightweight coreset" technique from (Bachem, Lucic, and Krause 2018).

Results

The methods are compared in terms of both objective value and computational time. To provide a normalized measure between datasets, we consider the "delta relative objective" (ΔRO) and "relative time" (RT), defined for any algorithm $\mathcal{A}$ as follows:

$$\Delta\text{RO}(\mathcal{A}) = \frac{\mathcal{L}(\mathcal{M}^{\mathcal{A}})}{\mathcal{L}(\mathcal{M}^{\mathcal{A}^*})} - 1 \; ; \; \text{RT}(\mathcal{A}) = \frac{T(\mathcal{A})}{T(\mathcal{A}^*)}. \quad (6)$$

Where $\mathcal{M}^{\mathcal{A}}$ is the set of medoids selected by algorithm $\mathcal{A}$, $\mathcal{A}^*$ refers to the algorithm providing the best objective.

Evolution of the objective and running time for different values of n and k. Figure 1 shows the objective and running time of five algorithms for different (k, n) settings on the MNIST dataset. In each graph, OneBatchPAM ranks among the best methods both in terms of objective and running time. The time evolution of OneBatchPAM is similar

[3]https://github.com/motiwari/BanditPAM

[4]https://github.com/kno10/python-kmedoids

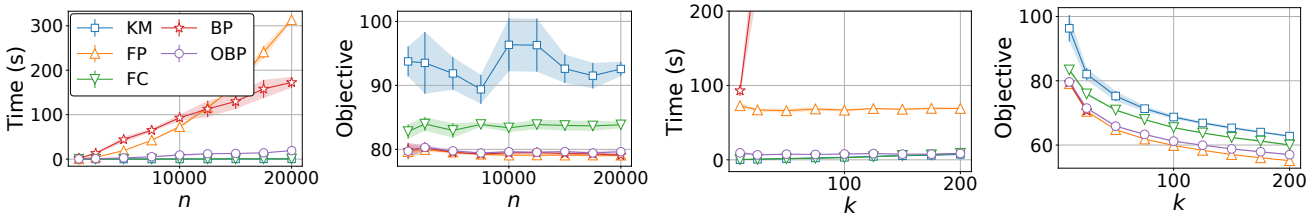

Figure 1: Evolution of the running time and objective on the MNIST dataset. Left: evolution as a function of n for $k = 10$. Right: evolution as a function of k for $n = 10000$. The results for five competitors are reported: k-means++ (KM), FasterPAM (FP), FasterCLARA-5 (FC), BanditPAM++-2 (BP), OneBatchPAM (OBP)

to the one of k-means++ and FasterCLARA-5, and significantly smaller than BanditPAM++ and FasterPAM, especially for large values of n. Additionally, the objective evolution of OneBatchPAM closely matches that of FasterPAM, while FasterCLARA-5 and k-means++ provide larger objective values.

Aggregated Results. Table 3 presents the averaged results over the three values of $k \in \{10, 50, 100\}$, the five repetitions of the experiments and the respective five "small scale" and "large scale" datasets. The detailed results per dataset and value of k are reported in the supplementary materials. As expected, FasterPAM provides the best objective and Random the fastest medoid selection for the small scale experiments. The OneBatchPAM variants reduce the computational burden of FasterPAM by a factor of 7 on average (RT = 15%) for a small penalization of the objective value (1.7% compared to FasterPAM for the NNIW variant). This observation highlights the efficiency of OneBatchPAM to provide a fast and accurate medoid selection. Notice that the time reduction factor increases with the number of samples. The relative time for OneBatchPAM is equal to 8.5% for the *letter* dataset, which corresponds to a reduction factor of around 12 (cf. detailed results in supplementary materials). We observe that k-means++ and FasterCLARA-5 are faster than OneBatchPAM (by a factor of around 7 for FasterCLARA-5). However, the running time reduction comes with a significant penalization of the objective: respectively 13% and 30% for FasterCLARA-5 and k-means++. For large scale datasets, FasterPAM and BanditPAM++ fail to provide medoid selections within reasonable computational times, positioning OneBatchPAM as the method with the best objective (ΔRO = 0). Similar to the small-scale experiments, FasterCLARA-5 is 7 times faster than OneBatch but is 8% worse in terms of objective. kmc2 is even faster, however, its objective is close to the random selection's objective.

Regarding the OneBatchPAM variants, we observe that debiasing offers a modest improvement compared to uniform sampling ($\sim 0.2\%$). The gain is higher for large values of k (around 1%), as highlighted in the detailed results. While the LWCS method degrades performance (likely because LWCS is primarily designed to provide strong theoretical guarantees for k-means++ rather than PAM), the NNIW variant shows significant objective improvements (above 1.2%) over uniform sampling with comparable computational time. This observation supports the systematic use of nearest neighbor importance weighting in OneBatchPAM. Indeed, the pairwise dissimilarities needed to compute the importance weights are also required by the OneBatchPAM core algorithm, which explains why using NNIW has a negligible impact on the running time.

Method	Small Scale		Large Scale	
	RT	ΔRO	RT	ΔRO
Random	**0.0**	62.9	**0.0**	20.3
FasterPAM	100.0	**0.0**	Na	Na
Alternate	161.1	20.0	Na	Na
FasterCLARA-5	2.8	13.0	15.0	8.0
FasterCLARA-50	30.0	10.9	161.7	7.1
kmc2-20	14.5	31.3	0.5	18.2
kmc2-100	72.2	31.9	2.4	17.6
kmc2-200	153.6	33.0	5.2	18.6
k-means++	1.6	30.4	78.8	18.4
LS-k-means++-5	37.2	23.5	97.1	15.3
LS-k-means++-10	73.1	20.1	121.6	13.7
BanditPAM++-0	930.2	3.6	Na	Na
BanditPAM++-2	1670.1	2.8	Na	Na
BanditPAM++-5	2880.7	2.2	Na	Na
OneBatchPAM-lwcs	15.1	12.3	117.9	2.8
OneBatchPAM-unif	15.1	3.9	104.2	1.2
OneBatchPAM-debias	15.7	3.7	100.0	0.8
OneBatchPAM-nniw	15.5	1.7	100.0	**0.0**

Table 3: Results Summary. The scores are averaged over the five repetitions of the experiment, the three values of $k \in \{10, 50, 100\}$ and the five respective "small scale" and "large scale" datasets. RT and ΔRO are given in percentage. Standard deviations are reported in Appendix.

Conclusion and Perspectives

This paper introduces OneBatchPAM, a novel k-medoids algorithm that accelerates FasterPAM by using a single batch of size $m = \mathcal{O}(\log(n))$ to estimate the objective. Our experiments demonstrate that OneBatchPAM is an efficient alternative to subsampling for handling large datasets within a reasonable running time while achieving performance similar to FasterPAM (with less than 2% error). Future work will focus on refining the subsampling process to further improve the running time and the accuracy of the medoid selection.

References

Arthur, D.; Vassilvitskii, S.; et al. 2007. k-means++: The advantages of careful seeding. In *Soda*, volume 7, 1027–1035.

Arya, V.; Garg, N.; Khandekar, R.; Meyerson, A.; Munagala, K.; and Pandit, V. 2001. Local search heuristic for k-median and facility location problems. In *Proceedings of the thirty-third annual ACM symposium on Theory of computing*, 21–29.

Bachem, O.; Lucic, M.; Hassani, S. H.; and Krause, A. 2016. Approximate k-means++ in sublinear time. In *Proceedings of the AAAI conference on artificial intelligence*, volume 30.

Bachem, O.; Lucic, M.; and Krause, A. 2018. Scalable k-means clustering via lightweight coresets. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 1119–1127.

Beretta, L.; Cohen-Addad, V.; Lattanzi, S.; and Parotsidis, N. 2024. Multi-Swap k-Means++. *Advances in Neural Information Processing Systems*, 36.

Bhat, A. 2014. K-medoids clustering using partitioning around medoids for performing face recognition. *International Journal of Soft Computing, Mathematics and Control*, 3(3): 1–12.

Braverman, V.; Cohen-Addad, V.; Jiang, H.-C. S.; Krauthgamer, R.; Schwiegelshohn, C.; Toftrup, M. B.; and Wu, X. 2022. The power of uniform sampling for coresets. In *2022 IEEE 63rd Annual Symposium on Foundations of Computer Science (FOCS)*, 462–473. IEEE.

Byrka, J.; Pensyl, T.; Rybicki, B.; Srinivasan, A.; and Trinh, K. 2017. An improved approximation for k-median and positive correlation in budgeted optimization. *ACM Transactions on Algorithms (TALG)*, 13(2): 1–31.

Charikar, M.; Guha, S.; Tardos, É.; and Shmoys, D. B. 1999. A constant-factor approximation algorithm for the k-median problem. In *Proceedings of the thirty-first annual ACM symposium on Theory of computing*, 1–10.

Charikar, M.; Henzinger, M.; Hu, L.; Vötsch, M.; and Waingarten, E. 2023. Simple, scalable and effective clustering via one-dimensional projections. *Advances in Neural Information Processing Systems*, 36: 64618–64649.

Chen, K. 2009. On coresets for k-median and k-means clustering in metric and euclidean spaces and their applications. *SIAM Journal on Computing*, 39(3): 923–947.

Chrobak, M.; Kenyon, C.; and Young, N. 2006. The reverse greedy algorithm for the metric k-median problem. *Information Processing Letters*, 97(2): 68–72.

Cohen-Addad, V.; Larsen, K. G.; Saulpic, D.; and Schwiegelshohn, C. 2022. Towards optimal lower bounds for k-median and k-means coresets. In *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing*, 1038–1051.

Cohen-Addad, V.; Lattanzi, S.; Norouzi-Fard, A.; Sohler, C.; and Svensson, O. 2020. Fast and accurate k-means++ via rejection sampling. *Advances in Neural Information Processing Systems*, 33: 16235–16245.

Cohen-Addad, V.; Saulpic, D.; and Schwiegelshohn, C. 2021a. Improved coresets and sublinear algorithms for power means in euclidean spaces. *Advances in Neural Information Processing Systems*, 34: 21085–21098.

Cohen-Addad, V.; Saulpic, D.; and Schwiegelshohn, C. 2021b. A new coreset framework for clustering. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, 169–182.

Czumaj, A.; and Sohler, C. 2007. Sublinear-time approximation algorithms for clustering via random sampling. *Random Structures & Algorithms*, 30(1-2): 226–256.

de Mathelin, A.; Deheeger, F.; MOUGEOT, M.; and Vayatis, N. 2021. Discrepancy-Based Active Learning for Domain Adaptation. In *International Conference on Learning Representations*.

Dohan, D.; Karp, S.; and Matejek, B. 2015. K-median algorithms: theory in practice. *Princeton University*.

Dua, D.; and Graff, C. 2017. UCI Machine Learning Repository.

Feldman, D. 2020. Core-sets: Updated survey. *Sampling techniques for supervised or unsupervised tasks*, 23–44.

Feldman, D.; and Langberg, M. 2011. A unified framework for approximating and clustering data. In *Proceedings of the forty-third annual ACM symposium on Theory of computing*, 569–578.

Guha, S.; and Mishra, N. 2016. Clustering data streams. In *Data stream management: processing high-speed data streams*, 169–187. Springer.

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

Huang, J.; Feng, Q.; Huang, Z.; Xu, J.; and Wang, J. 2024. Linear Time Algorithms for k-means with Multi-Swap Local Search. *Advances in Neural Information Processing Systems*, 36.

Huang, L.; Jiang, S. H.-C.; and Lou, J. 2023. The power of uniform sampling for k-median. In *International Conference on Machine Learning*, 13933–13956. PMLR.

Jaiswal, R.; and Kumar, A. 2024. Universal Weak Coreset. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, 12782–12789.

Kariv, S.; and Hakimi, O. 1979. An algorithmic approach to network location problems. II: the p-medians. *SIAM J. Appl. Math*, 37(3): 539.

Kaufman, L. 1986. Clustering large data sets. *Pattern recognition in practice*, 425–437.

Kaufman, L. 1990. Partitioning around medoids (program pam). *Finding groups in data*, 344: 68–125.

Kaufman, L.; and Rousseeuw, P. J. 1987. Clustering by means of Medoids. Statistical data analysis based on the L1–norm and related methods, edited by Y. Dodge.

Kaufman, L.; and Rousseeuw, P. J. 2008. Clustering large applications (Program CLARA). *Finding groups in data: an introduction to cluster analysis*, 126–63.

Kaushal, V.; Iyer, R.; Kothawade, S.; Mahadev, R.; Doctor, K.; and Ramakrishnan, G. 2019. Learning from less data: A unified data subset selection and active learning framework for computer vision. In *2019 IEEE Winter Conference on Applications of Computer Vision (WACV)*, 1289–1299. IEEE.

Krizhevsky, A.; Hinton, G.; et al. 2009. Learning multiple layers of features from tiny images.

Lattanzi, S.; and Sohler, C. 2019. A better k-means++ algorithm via local search. In *International Conference on Machine Learning*, 3662–3671. PMLR.

LeCun, Y.; Cortes, C.; and Burges, C. J. 1994. The MNIST databaseof handwritten digits.

Li, S.; and Svensson, O. 2013. Approximating k-median via pseudo-approximation. In *proceedings of the forty-fifth annual ACM symposium on theory of computing*, 901–910.

Loog, M. 2012. Nearest neighbor-based importance weighting. In *2012 IEEE international workshop on machine learning for signal processing*, 1–6. IEEE.

Mettu, R. R.; and Plaxton, C. G. 2004. Optimal time bounds for approximate clustering. *Machine Learning*, 56(1): 35–60.

Meyerson, A.; O'callaghan, L.; and Plotkin, S. 2004. A k-median algorithm with running time independent of data size. *Machine Learning*, 56(1): 61–87.

Mishra, N.; Oblinger, D.; and Pitt, L. 2001. Sublinear time approximate clustering. In *SODA*, volume 1, 439–447.

Park, H.-S.; and Jun, C.-H. 2009. A simple and fast algorithm for K-medoids clustering. *Expert systems with applications*, 36(2): 3336–3341.

Pelleg, D.; and Moore, A. 1999. Accelerating exact k-means algorithms with geometric reasoning. In *Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining*, 277–281.

Ren, J.; Hua, K.; and Cao, Y. 2022. Global optimal k-medoids clustering of one million samples. *Advances in Neural Information Processing Systems*, 35: 982–994.

Schmidt, M.; Schwiegelshohn, C.; and Sohler, C. 2020. 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 17*, 232–251. Springer.

Schubert, E.; and Lenssen, L. 2022. Fast k-medoids Clustering in Rust and Python. *Journal of Open Source Software*, 7(75): 4183.

Schubert, E.; and Rousseeuw, P. J. 2021. Fast and eager k-medoids clustering: O (k) runtime improvement of the PAM, CLARA, and CLARANS algorithms. *Information Systems*, 101: 101804.

Thorup, M. 2005. Quick k-median, k-center, and facility location for sparse graphs. *SIAM Journal on Computing*, 34(2): 405–432.

Tiwari, M.; Kang, R.; Lee, D.; Thrun, S.; Shomorony, I.; and Zhang, M. J. 2023. BanditPAM++: Faster k-medoids Clustering. *Advances in Neural Information Processing Systems*, 36: 73371–73382.

Tiwari, M.; Zhang, M. J.; Mayclin, J.; Thrun, S.; Piech, C.; and Shomorony, I. 2020. Banditpam: Almost linear time k-medoids clustering via multi-armed bandits. *Advances in Neural Information Processing Systems*, 33: 10211–10222.

Wei, K.; Iyer, R.; and Bilmes, J. 2015. Submodularity in data subset selection and active learning. In *International conference on machine learning*, 1954–1963. PMLR.