Clustering Permutations: New Techniques with Streaming Applications

Diptarka Chakraborty  
National University of Singapore, Singapore

Debarati Das  
Pennsylvania State University, University Park, PA, USA

Robert Krauthgamer  
Weizmann Institute of Science, Rehovot, Israel

Abstract

We study the classical metric $k$-median clustering problem over a set of input rankings (i.e., permutations), which has myriad applications, from social-choice theory to web search and databases. A folklore algorithm provides a 2-approximate solution in polynomial time for all $k = O(1)$, and works irrespective of the underlying distance measure, so long it is a metric; however, going below the 2-factor is a notorious challenge. We consider the Ulam distance, a variant of the well-known edit-distance metric, where strings are restricted to be permutations. For this metric, Chakraborty, Das, and Krauthgamer [SODA, 2021] provided a $(2 - \delta)$-approximation algorithm for $k = 1$, where $\delta \approx 2^{-40}$.

Our primary contribution is a new algorithmic framework for clustering a set of permutations. Our first result is a 1.999-approximation algorithm for the metric $k$-median problem under the Ulam metric, that runs in time $(k \log(nd))^{O(k)}nd^3$ for an input consisting of $n$ permutations over $[d]$. In fact, our framework is powerful enough to extend this result to the streaming model (where the $n$ input permutations arrive one by one) using only polylogarithmic (in $n$) space. Additionally, we show that similar results can be obtained even in the presence of outliers, which is presumably a more difficult problem.

2012 ACM Subject Classification  Theory of computation → Facility location and clustering; Theory of computation → Streaming, sublinear and near linear time algorithms

Keywords and phrases Clustering, Approximation Algorithms, Ulam Distance, Rank Aggregation, Streaming

Digital Object Identifier 10.4230/LIPIcs.ITCS.2023.31


Funding Diptarka Chakraborty: Work partially supported by an MoE AcRF Tier 2 grant (WBS No. A-8000416-00-00) and an NUS ODPRT grant (WBS No. A-0008078-00-00). Robert Krauthgamer: Work partially supported by ONR Award N00014-18-1-2364, the Israel Science Foundation grant #1086/18, and a Minerva Foundation grant, and by the Israeli Council for Higher Education (CHE) via the Weizmann Data Science Research Center.

1 Introduction

Clustering is one of the ubiquitous tasks used in data analysis, which partitions a set of objects into several groups so that similar items lie in the same group. One of the most widely studied variants is the metric $k$-median clustering. In this problem, given an input set $S$ of $n$ data points, the goal is to find a set of $k$ median points from the underlying space (not necessarily from $S$) such that the sum of distances of all the data points in $S$ to its nearest median point is minimized. (See Section 2 for a formal definition.) Throughout this paper, we consider the above variant and refer to it simply by the $k$-median problem.
For \( k = 1 \), the problem is also referred to as the \textit{geometric median} (or simply median) problem. For many applications, it suffices to find an approximate solution to the problem, i.e., find a set of points from the metric space whose objective value approximates the minimum multiplicatively (for a formal definition, see Section 2). This problem has been studied extensively in theory as well as in applied domains, both for \( k = 1 \) and arbitrary \( k \). The complexity of the problem varies with the underlying metric space. For \( k = 1 \), perhaps the most well-studied version is over a Euclidean space (aka the Fermat-Weber problem), for which a near-linear time \((1 + \epsilon)\)-approximation algorithm (for any \( \epsilon > 0 \)) is known \cite{FerWeber}. Other spaces that have been considered for the median problem include Hamming (folklore), the edit metric \cite{Hamming,EditMetric,EditMetric2}, rankings/permutations \cite{PermutationMedian,PermutationMedian2,PermutationMedian3}, Jaccard distance between sets \cite{Jaccard}, and many more \cite{Jaccard2, Jaccard3, Jaccard4}. The problem is clearly more challenging for general \( k \). For points in \( \mathbb{R}^d \), Chen \cite{Chen} gave \((1 + \epsilon)\)-approximation algorithm with running time \( O(nd^k + 2^{(k/\epsilon)\Theta(d^2) \log^{k+2} n}) \) (see references therein for an overview) using coresets.

For arbitrary metric spaces, \( O(1)\)-approximation algorithms (with a trade-off between the approximation factor and the running time) are known, e.g. \cite{ArbitraryMetric1, ArbitraryMetric2, ArbitraryMetric3, ArbitraryMetric4, ArbitraryMetric5}. Better approximation results are known for specific metric spaces, like Hamming \cite{Hamming2}, shortest-path metric in a graph \cite{ShortestPath}, etc.

One of the fundamental metrics (other than Euclidean and Hamming) that finds numerous applications is the \textit{edit metric}. The edit distance is a well-known dissimilarity measure between strings, which counts the minimum number of basic edit operations, like character insertion, deleting, and substitution, required to transform one string into the other. The \( k \)-median clustering problem over the edit metric is of utter importance in various domains, including computational biology \cite{Bioedit, Bioedit2}, DNA storage system \cite{DNAstorage,DNAstorage2}, speech recognition \cite{SpeechRecognition}, and classification \cite{Classification}. For \( k = 1 \), it is also referred to as the \textit{median string} problem \cite{MedianString} (an equivalent formulation is known as multiple sequence alignment \cite{MultipleSequenceAlignment}). Despite being an important problem, we only know that it is NP-hard, even for \( k = 1 \) \cite{NP-hard1,NP-hard2}. Although several heuristics exist \cite{Heuristics1, Heuristics2, Heuristics3, Heuristics4, Heuristics5, Heuristics6}, we do not know any approximation result better than what holds for arbitrary metrics.

We study the \( k \)-median problem over the \textit{Ulam metric}, which is a variant of the edit metric, by restricting strings to be permutations. The Ulam metric of dimension \( d \) is \( (S_d, \Delta) \), where \( S_d \) is the set of all the permutations over \([d]\) and \( \Delta(x, y) \) is the minimum number of character-move operations needed to transform \( x \) into \( y \) \cite{Ulam}.\(^1\) Studying the Ulam metric is beneficial from two facets. First, it is a close variant of the edit metric defined over permutations and thus captures many inherent difficulties of the edit metric. Thus, progress in the Ulam metric may provide insights to the same problem under a more general edit metric. Second, it is a natural dissimilarity measure between rankings that arise in diverse areas ranging from social-choice theory \cite{SocialChoice} to information retrieval \cite{InformationRetrieval}, and indeed has been studied from different algorithmic perspectives \cite{AlgorithmicPerspective1, AlgorithmicPerspective2, AlgorithmicPerspective3, AlgorithmicPerspective4, AlgorithmicPerspective5, AlgorithmicPerspective6}.

There is a folklore \( O(n^{k+1} \cdot f(d))\)-time algorithm (where \( f(d) \) denotes the time to compute the distance between two points) that provides a 2-approximate solution to the \( k \)-median problem for an arbitrary metric space, by simply reporting the best \( k \)-tuple from the input set as the median points (see Procedure 1). Breaking below the 2-factor even in \( n^{O(k)} \) time is one of the most notorious challenges, even for some specific metrics. So far, we do not know any affirmative result for the Ulam metric. Very recently, Chakraborty, Das, and

\(^1\) In a permutation, a character move can be thought of as “picking up” a character and “placing” it in another position. Since it is equivalent to one deletion and one insertion, one can define the distance alternatively using character insertions and deletions \cite{Chakraborty}.
Krauthgamer [15] provided a \((2 - \delta)\)-approximation algorithm only for the special case of \(k = 1\) (aka rank aggregation under the Ulam), where \(\delta \approx 2^{-40}\) is a tiny constant. On the contrary, the median problem concerning Kendall’s tau distance, yet another popular dissimilarity measure over permutations (e.g., [33, 51, 52, 24, 2]), possesses a PTAS [34, 49], which can be extended to \(k\)-median using coresets [22].

We further study this problem (\(k\)-median under the Ulam metric) in the streaming model, i.e., when the input arrives sequentially; more specifically, the \(n\) input permutations \(x_1, x_2, \ldots, x_n \in S_d\) arrive one by one. This is sometimes called an insertion-only stream, because an input permutation cannot be deleted after its arrival. One of the challenges in this model is that the algorithm cannot freely access the input, and the main goal is to devise an algorithm with an amount of space that is sublinear (ideally, logarithmic) in \(n\). The \(k\)-median problem in the insertion-only streaming model has also been studied extensively, e.g., [28, 18, 10, 12]. However, no non-trivial result (other than what holds for an arbitrary metric) is known for the Ulam metric.

1.1 Our Contribution

Our main result is a streaming algorithm for the \(k\)-median problem under the Ulam metric that achieves better than the 2-approximation factor.

\begin{center}
\textbf{Theorem 1.1.} There is a (randomized) streaming algorithm that, given a set of permutations \(x_1, x_2, \ldots, x_n \in S_d\) (arriving in streaming fashion), provides a 1.9999995-approximate solution to the (metric) \(k\)-median problem under the Ulam metric, using only \(k^2d \text{polylog}(nd)\) bits of space, with high probability. Moreover, the algorithm has update time \((k \log n)^{O(1)} d \log^2 d\) and query time \((k \log(\log(\log nd)))^O(k) d^3\).
\end{center}

It is worth mentioning that here the input size is \(O(nd \log d)\) bits (since each permutation of \(S_d\) requires \(O(d \log d)\) bits). In our algorithm, we only need to maintain a subset of \(k^2 \text{polylog}(nd)\) permutations. For \(k = 1\), we can improve the space-bound to only \(O(d \log d \log^2 n)\) (Theorem 5.1). Also, all our algorithms require only a single pass.

To achieve our result, we first develop a new algorithm framework that provides a 1.999-approximate solution to the median problem (i.e., for \(k = 1\)) deterministically (Theorem 3.1). Compared to the previous approximation result of [15], our algorithm is superior in various aspects. First, the approximation factor is 1.999, an improvement over the \(2 - \delta\), where \(\delta \approx 2^{-40}\) in [15]. Second, our analysis is much simpler than that in [15], which was roughly divided into the following two cases: In the first case, a large fraction of the optimum objective value is “concentrated on a small fraction of symbols of \([d]\)”, and in the second is the optimum objective value is “spread out throughout all the symbols of \([d]\)” (see also the Technical Overview below). Analyzing these two cases separately and then combining them makes the entire analysis quite complicated and also affects the approximation guarantee. In contrast, our analysis is pretty simple and argues that for \(k = 1\),

- either there exists five input permutations from which we can reconstruct an approximate median (by essentially solving the feedback vertex set problem over a special type of tournament graph);
- or, there is an input permutation around which there is a large cluster (and thus would also provide a good approximate solution to the entire input);
- or, there is an input permutation that is close to an (unknown) optimal median (and thus, again, would be a good approximate median).

Third, and perhaps most importantly, because the final output is either an input permutation or derived from only five input permutations, we essentially show that there are at most \(n^5\) candidate points, one of which provides a 1.999-approximate solution. For general
k, the same argument is clearly true also for each of the clusters induced by an optimal $k$-median solution, and thus we can easily extend our framework to the $k$-median problem, for arbitrary $k$, with running time $n^{O(k)}d^3$ (Theorem 3.4). This running time can further be improved to $(k \log(nd))^{O(k)}nd^3$ using the sampling technique described in Section 4. Further, we can extend it to the $k$-median problem with outliers (see Section 3.2 for the definition and the details), presumably a more challenging problem. Such extensions were not possible to the algorithm of [15]. Lastly, our algorithm can be implemented in a streaming fashion by storing (with the help of a “clever” sampling) only $k^2 \log(nd)$ input permutations and then running our offline algorithm (with a slight modification) on them. We emphasize that randomization is used only while implementing it in the streaming model.

1.2 Technical Overview

The algorithm of Chakraborty, Das and Krauthgamer [15] computes, given a set $S$ of $n$ permutations over $[d]$, a $(2 - \delta)$-approximate 1-median under the Ulam metric for $\delta \approx 2^{-40}$. Their algorithm starts with the following simple observation: Fix an optimal median permutation $y^*$, and let $\ell$ be the average distance between $y^*$ and the input permutations. Now if $S$ contains a permutation $x$ such that the distance between $y$ and $x$ is $\leq (1 - \delta)\ell$, for some constant $\delta > 0$, then $x$ provides a $(2 - \delta)$-approximation to the optimal objective value. Otherwise, they first considered the case where the average distance is large, i.e., $\ell = \Omega(d)$. In this case, using a counting argument, they show that there exists $x \in S$ such that at least $\Omega(n)$ other input permutations are at a distance at most $(2 - \delta')\ell$. Thus taking $x$ as an approximate median provides better-than-2 approximation to the optimal objective value. Of course, this $x \in S$ is not known, but outputting the input permutation that minimizes the objective serves the purpose. In the other case, where the average distance $\ell$ is small, if the cost is distributed only over a few symbols of $[d]$, then restricting the input permutations only to these symbols gives rise to an instance with a large average objective, and thus one can reuse the large-distance algorithm mentioned above. If, however, the total cost is distributed over many symbols, then for almost all the symbols in $[d]$, the cost is small. Now consider two such small-cost symbols; as both are aligned together in most of the input permutations, their relative order in all these permutations is the same as that in $y^*$, and thus can be computed by examining all the input permutations and taking the majority. Moreover, using random sampling, this majority can be decided by looking at only $O(\log n)$ permutations. However, this dependency on $\log n$ input permutations becomes ineffective when we try to lift these ideas to $k$-median for $k > 1$, and this is where our new framework plays a crucial role.

We start by providing an algorithm that shows that either an input permutation breaks the 2-factor or the input set $S$ contains 5 permutations from which we can derive a permutation whose distance is small from an optimal median $y^*$, and it thus gives better-than-2 approximation to the optimal objective value. This dependency on the number inputs is significantly better than in [15], and it is specifically important for the $k$-median problem (general $k > 1$), where the grouping of the input permutations into $k$ clusters is not known. However, using our framework, one can try all $\binom{n}{5}$ different ways of picking 5 input permutations and use them to derive candidates for approximate median. These can serve as candidates for all the $k$ clusters (simultaneously) without knowing the optimal partitioning of $S$. This approach breaks down for the algorithm of [15], where the candidates are derived from $\Omega(\log n)$ inputs, thus giving $\binom{n}{\log n}$ candidates overall. We proceed to present next our new framework for computing 1-median.
New algorithm for 1-median. Our algorithm (in Section 3) is based on a three-step framework. First, similar to [15], we use the fact that if the input set $S$ contains a permutation $x$ such that the distance between $x, y^*$ is at most $(1 - \delta)\ell$ then $x$ serves as a $(2 - \delta)$-approximate median.

Next, for the sake of analysis, we fix an optimal alignment between each input $x$ and $y^*$. Let $I_x$ denote the symbols that are not aligned in this optimal alignment. In step two, we consider the scenario where all the permutations are at a distance at least $(1 - \delta)\ell$ from $y^*$ and moreover, there is a subset $T \subseteq S$ containing five permutations $x_1, \ldots, x_5$ such that for any pair $x_i, x_j$ their corresponding sets of unaligned characters have a very small overlap, i.e., $|I_{x_i} \cap I_{x_j}| \leq \epsilon\ell$. In this case, we design an algorithm MedianReconstruct that just by using $x_1, \ldots, x_5$ constructs a permutation $\tilde{x}$ such that the distance between $\tilde{x}$ and $y^*$ is at most $(1 - \delta)\ell$. Let $B = \cup_{i,j \in [5]} (I_{x_i} \cap I_{x_j})$. Then $|B| \leq 10\epsilon\ell$. Now for any pair of symbols $a, b \in [d] \setminus B$, as each of $a, b$ can be unaligned in at most one $x_i$, together they are aligned in at least three out of five permutations $x_1, \ldots, x_5$. Thus by looking at the relative order of $a$ and $b$ in these five permutations and taking the majority, we correctly deduce their order in $y^*$. This observation makes our algorithm framework significantly stronger than [15] by reducing the dependency on the number of input permutations that are actually required to construct a good approximate median. However, we might not get the correct relative order for the pair of symbols that come from the set $B$. Instead, they may create conflict with other useful orders. To solve this, following a similar idea as in [15], we create a relative order graph $H$ that contains $d$ vertices corresponding to the $d$ symbols in the input permutations. For every pair of symbols $a, b$, we add an edge from $a$ to $b$ if at least in three of the five permutations from $T$, $a$ appears before $b$; otherwise, add an edge from $b$ to $a$. Next, we generate from $H$ an acyclic graph $H'$ by removing cycles iteratively while deleting the smallest one first. Here, since $H$ is a tournament, the shortest cycle length is always three. Along with this, each cycle should contain at least one vertex from $B$. As $|B| \leq 10\epsilon\ell$, this process removes very few vertices, and thus most of the symbols survive in $H'$. Next, we create a permutation $\tilde{x}$ by taking a topological ordering on $H'$ and appending the missing symbols at the end. Our analysis vastly differs from the one given in [15], where the relative order graph is not a tournament. This simplifies our cycle removal process and provides a better approximation guarantee.

Lastly, we consider the case where there are at most four input permutations whose corresponding sets of unaligned symbols in the optimal alignment with $y$ have a small overlap. Again here, for at least one of them (call it $x$), there are $\Omega(n)$ other inputs $x'$ such that the sets of unaligned symbols for $x$ and $x'$ have a large overlap, and thus the distance between $x$ and $x'$ is strictly smaller than $2\ell$. Thus considering $x$ as an approximate median provides a better-than-2 approximation of the optimal objective value. Our algorithm also finds an input permutation that minimizes the total objective and finally outputs the best among those generated by MedianReconstruct and the best input permutation.

Approximating $k$-median. Next, we show how our 1-median algorithm can be extended to $k$-median for general $k$. For analysis purpose, fix $k$ optimal medians $y_1^*, \ldots, y_k^*$, and let $C_i$ be the set of input permutations served by $y_i^*$. Following the 1-median algorithm, either $C_i$ contains a permutation that provides $< 2$ approximation, or it includes 5 permutations using which we can construct the approximate median for $C_i$. However, to start with, we do not know the optimal $k$-partition of the input set. Nevertheless, we can construct a set $M$ of potential approximate $k$ medians as follows: For each of the $\binom{k}{5}$ different choices of 5 input permutations, create a permutation using the MedianReconstruct algorithm and add it
to the set $M$. Next, add all input permutations to $M$. It is straightforward to see that the set $M$ contains $k$ permutations $\tilde{y}_1, \ldots, \tilde{y}_k$ such that each $\tilde{y}_i$ is a better-than-2 approximate median for $C_i$. To identify these $k$ permutations, we try all possible size-$k$ subsets of $M$ and output the one that minimizes the total objective. We can bound the overall running time of the algorithm by $n^{O(k)}d^3$. The details of this algorithm can be found in Section 3.1. This running time can further be improved to $(k \log(nd))^{O(k)}n^2d^3$ using the sampling technique used in the streaming algorithm described next.

**Streaming Algorithm for Approximating $k$-Median.** For the $k$-median problem, we have shown that for each cluster $C_i$, at most 5 permutations are enough to construct an approximate median. However, without knowing the optimal $k$-partitioning of the input $S$, we need to try all possible $\binom{n}{k}$ choices. Unfortunately, this step becomes infeasible when the $n$ input permutations arrive in a stream, and we can afford to store only a few (preferably polylog) of them. Towards this, we design a streaming algorithm that requires several techniques, including an efficient sampling of the input permutations, coreset construction, etc. This algorithm provides a $(2 - \delta)$-approximation of $k$-median while storing $O(k^2 \log^3 n \log d)$ permutations. Next, we provide a brief overview of this algorithm. The details can be found in Section 4.

**Sampling procedure.** Let $OPT$ be the total optimal objective for all $k$ clusters. The key component of our streaming algorithm is a “clever” sampling technique that selects a set $R_1$ of $O(k \log^4 n \log d)$ permutations from the input stream and ensures that for any cluster $C_i$, if its optimal objective $O_i$ is at least a constant fraction of the average objective, i.e., $O_i = \Omega(\frac{OPT}{n_i})$, then either $R_1$ contains a permutation that serves as a better-than-2 approximate median for $C_i$, or there are 5 permutations in $R_1$ such that applying the $\text{MedianReconstruct}$ algorithm on them we can generate an approximate median. To explain this, we fix a cluster $C_i$ and let $n_i = |C_i|$, $y_i^*$ be an optimal median of $C_i$, and $\ell_i = O_i/n_i$ be the average objective. Next, we argue that if we sample each permutation in the input stream independently, uniformly at random with probability $\log^2 n/n_i$, then the sample set satisfies the above-mentioned properties.

For this, we first consider the case when at least $1/\log n$ fraction of the permutations in $C_i$ are close, i.e., at a distance $<(1 - \delta)\ell_i$ from the optimal median $y_i^*$. Note that all such permutations are good candidates for an approximate median. Also, following our sampling rate, we will sample at least one of them with a high probability.

Thus from now on, we assume even fewer permutations are close to $y_i^*$. Note, there can be at most $\frac{n_i}{1 + \delta}$ permutations in $C_i$ that are at distance $>(1 + \delta)\ell_i$ from $y_i^*$. Hence a constant fraction of permutations is at a distance between $(1 - \delta)\ell_i$ and $(1 + \delta)\ell_i$ from $y_i^*$. Let $C_i^{\text{avg}}$ be the set of all these permutations. Now for every permutation $x \in C_i^{\text{avg}}$, we define a neighboring set $C(x)$ containing all other permutations $z \in C_i^{\text{avg}}$ such that the intersection between the set of unaligned characters in optimal alignments between $x$, $y_i^*$ and $z$, $y_i^*$ is large and thus they are close. Note here $x$ serves as a better-than-2 approximate median for $C(x)$. Now we call $x$ to be in $C_i^{\text{avg,dense}}$ if $|C(x)|$ is large i.e., $|C(x)| = \Omega(|C_i^{\text{avg}}|) = \Omega(n_i)$. This means for each permutation $x$ in $C_i^{\text{avg,dense}}$ there is a large cluster of size $\Omega(n_i)$ around $x$ and as $x$ serves as a better-than-2 approximate median for $C(x)$, overall by considering $x$ to be the approximate median for $C_i$ we get $<2$ approximation of the optimal objective $O_i$. Now, if $C_i^{\text{avg,dense}}$ is large, then again, by our sampling strategy with high probability, we will sample a permutation $x$ from it.

Lastly, we consider the case where $C_i^{\text{avg,dense}}$ is small. Here using an iterative argument, we show that our sampling algorithm will sample at least 5 permutations such that their corresponding set of unaligned symbols has a small overlap. Thus, using algorithm $\text{Median-}$. 


RECONSTRUCT, we can construct an approximate median. To see this notice as $\tilde{C}_3^{\text{avg},\text{dense}}$ is small with high probability, we will sample a permutation $x_1 \in \tilde{C}_3^{\text{avg}} \setminus \tilde{C}_3^{\text{avg},\text{dense}}$ and thus $C(x_1)$ is small. Moreover, as $C(x_1)$ is small, we will sample another permutation $x_2$ such that $C(x_2)$ is small and $x_2 \notin C(x_1)$ and thus $x_1$ and $x_2$ have small overlap between their unaligned character set. We can continue this process and show overall, with a high probability, 5 permutations can be sampled with the small overlap property. Thus we can apply algorithm MEDIANRECONSTRUCT on these 5 strings and compute an approximate median with objective $< 2\Omega_i$.

Here for the sampling to work for cluster $C_i$, we use a sampling rate $\log^2 n/n_i$. However, as we do not know $n_i$ in advance, we try all sampling rates in $\{1/(1+\epsilon), 1/(1+\epsilon)^2, \ldots, 1/n\}$ and we can ensure that one of them is arbitrarily close to $\log^2 n/n_i$. The challenge here is that for a sampling rate much larger than $\log^2 n/n_i$, the sample size can grow beyond our space limit. For this, whenever a sample set grows beyond $k \log^3 n$, we discard that set. Though this limits the space complexity, it can destroy a good sample set that is necessary to keep from computing an approximate median for $C_i$. Next, we need to show for the right sampling rate, the sample set size always stays within the limit, and thus we do not miss the useful permutations. To ensure this in the uniform sampling process, we also incorporate a pruning strategy. Note with a sampling rate $\log^2 n/n_i$ or $(1 + \epsilon) \log^2 n/n_i$, with high probability, we sample at most $10 \log^2 n$ permutations from $C_i$. However, if there is a cluster $C_j$ such that $|C_j| > |C_i|$, then we end up sampling much more permutations from $C_j$ and the sampling set size goes beyond $k \log^3 n$, and we discard it. To upper bound this while adding a new permutation to our sample set, we ensure all previously added permutations are at a distance at least $\beta \ell_i$ (for a small constant $\beta$) from it. Now to start with, as we assumed $O_i = \Omega(p_i^2)$, we can show that the total number of sampled permutations from all clusters $C_j \neq C_i$ is bounded by $k \log^3 n$ with high probability. Also, because of the pruning, we get an extra additive error $\beta \ell_i$. This can be bounded by setting $\beta$ appropriately.

Approximating small objective clusters. Now we focus on the clusters whose objective is much smaller than the average objective $\Omega(k)/k$. Notice that the total objective contributed by the small objective clusters is very small. For these clusters, it suffices just to give a constant approximation of their optimal cost. For this we use the monotone faraway sampling (MFS) from [12] to sample a set $R_2 \subseteq S$ of size $O(k^2 \log k \log kn)$ such that for each cluster $C_i$, $R_2$ contains a permutation $y_i$ such that the objective of $C_i$ w.r.t. $y_i$ is at most $5O_i + \rho \Omega(k)/k$. Thus by keeping $\rho$ small, we achieve the required approximation.

Computing approximate $k$-median using coreset. After sampling set $R_1$ and $R_2$, following the offline $k$-median algorithm, we design a potential $k$-median set $\tilde{M}$ by adding all permutations of $R_1, R_2$ to $\tilde{M}$. Together with this for each subset $T$ of $R_1$ of size 5, run algorithm MEDIANRECONSTRUCT($T$) and add the output $\tilde{x}_T$ to $\tilde{M}$. However, to decide which $k$ permutations from $\tilde{M}$ minimize the total objective, we need access to the actual input set $S$ (which we cannot store using a small space). Now again, to upper bound the space complexity, instead of storing $S$ explicitly, we construct a $(k, \lambda)$-coreset (see Section 4 for the definition) for $S$ with respect to the implicit potential median set $M$. Here $M$ is a set containing all permutations from $S$. Moreover, for each subset $T \subseteq S$ of size 5, the output of MEDIANRECONSTRUCT($T$) is also present in $M$. This coreset ensures that considering any $k$-size subset of $M$ as a potential $k$-median if we compute the corresponding objectives for set $S$ and the coreset then these two objectives are close when $\lambda$ is small. Thus we can use the coreset to decide the best candidate $k$-median from $\tilde{M}$. Moreover, as $|M| \leq O(n^\lambda)$, following Theorem 4.3, we can show that the coreset size is also bounded by $O(k^2 \log^2 n)$. We remark that to make our algorithm space efficient, instead of explicitly storing each $\tilde{x}_T$ to $\tilde{M}$, we recompute it whenever $\tilde{x}_T$ is a part of a candidate $k$-median.
Approximating $k$-median with outliers. We further extend our $k$-median algorithm in the presence of outliers. In the $k$-median with outliers problem, given a parameter $p \in [0, 1)$ and input set $S$, the goal is to find a set of at most $k$ permutations that minimizes the $k$-median objective value of a subset of $S$ of size at least $(1 - p)|S|$. (See Section 3.2 for a formal definition.) Using the argument the same as that for the $k$-median problem (without outlier), we claim that a subset of at most $k$ permutations from a candidate set $M$ (of potential medians) of size at most $O(n^3)$ achieves a 1.999-factor approximation for the outlier variant as well (Theorem 3.5).

1.3 Conclusion

In this paper, we study the (metric) $k$-median problem under the Ulam metric, which is known to be NP-hard even for $k = 2$. The Ulam metric is of utter importance because it is a variant of the more general edit metric and an interesting dissimilarity measure over rankings (or permutations). There is a folklore 2-approximation algorithm that works for any metric space, and breaking this factor-2 barrier is one of the interesting challenges. Despite being an important metric, the problem under the Ulam metric does not possess any better approximation algorithm. For the special case of $k = 1$, there is a $(2 - \delta)$-approximation (where $\delta \approx 2^{-40}$) algorithm known [15]. However, that algorithm does not provide any non-trivial result for arbitrary values of $k$.

We provide a 1.9999995-approximation algorithm for the $k$-median problem, with running time $(k \log(n^d))^{O(k)} n d^3$. Moreover, our algorithm works in the insertion-only streaming model, using only polylogarithmic (in the number of input permutations) space. Further, we can extend our framework to get a similar result even in the presence of outliers, which is presumably a more complex problem. We also would like to highlight that our framework is not very specific to the Ulam metric; in fact, it (with slight modification) also provides a similar result for other metrics like Kendall’s tau defined over rankings/permutations. One exciting direction is to see whether our new framework can give similar results to the other known distance measures involving rankings, such as Spearman’s footrule, Minkowski, Cayley, swap-and-mismatch, etc. Another stimulating future direction is to use this new framework to get a similar result for another essential variant of the clustering problem, namely the $k$-center clustering problem (under the Ulam metric). Finally, extending our result to the more general edit metric (even with certain restrictions on the input) would, of course, be super intriguing.

2 Preliminaries

Notations. We use $[d]$ to denote the set $\{1, 2, \ldots, d\}$. Let $S_d$ denote the set of all permutations over $[d]$. Throughout the paper, we consider a permutation $x$ (over $[d]$) as a sequence $a_1, a_2, \ldots, a_d$ such that $x(i) = a_i$.

Ulam metric and the $k$-median problem. For any two permutations $x, y \in S_d$, the Ulam distance between them, denoted by $\Delta(x, y)$, is the minimum number of character move operations that is needed to transform $x$ into $y$. Equivalently, it can be defined as $d - |LCS(x, y)|$, where LCS$(x, y)$ denotes a longest common subsequence between $x$ and $y$.

---

2 A single move operation in a permutation can be thought of as “picking up” a character from its position and then “inserting” that character in a different position.
For any two permutations (permutations) \( x \) and \( y \) of lengths \( d_x \) and \( d_y \) respectively, an alignment \( g \) is a function that maps \([d_x]\) to \([d_y] \cup \{\perp\}\) such that:

1. \( \forall i \in [d_x], \text{ if } g(i) \neq \perp, \text{ then } x(i) = y(g(i)) \);
2. For any two distinct \( i, j \in [d_x] \) where \( g(i), g(j) \neq \perp \), \( i < j \Leftrightarrow g(i) < g(j) \).

For an alignment \( g \) between two permutations (permutations) \( x \) and \( y \), we say \( g \) aligns a character \( x(i) \) with some character \( y(j) \) if and only if \( j = g(i) \). Thus the alignment \( g \) is essentially a common subsequence between \( x \) and \( y \).

For any permutation \( x \in \mathcal{S}_d \) and a set \( Y \subseteq \mathcal{S}_d \), let us define the distance between \( x \) and \( Y \) as \( \Delta(x,Y) := \min_{y \in Y} \Delta(x,y) \) (i.e., the minimum distance between \( x \) and a permutation from \( Y \)). Given a set \( S \subseteq \mathcal{S}_d \) and a subset \( Y \subseteq \mathcal{S}_d \), we define the \textit{median objective value} of \( S \) with respect to \( Y \) as \( \text{Obj}(S,Y) := \sum_{x \in S} \Delta(x,Y) \).

Given a set \( S \subseteq \mathcal{S}_d \), the \textit{k-median} problem asks to find a subset \( Y \subseteq \mathcal{S}_d \) of size at most\(^3 \) \( k \) such that \( \text{Obj}(S,Y) \) is minimized, i.e., \( Y^* = \arg \min_{Y \subseteq \mathcal{S}_d, |Y| \leq k} \text{Obj}(S,Y) \). We refer to the set \( Y^* \) as \textit{k-median} of \( S \). We refer \( \text{Obj}(S,Y^*) \) as \( \text{OPT}(S) \), or simply \( \text{OPT} \) when \( S \) is clear from the context. Note, for \( k = 1 \), \( y^* = \arg \min_{y \in \mathcal{S}_d} \text{Obj}(S,y) \) is referred to as a \textit{median} (or geometric median or 1-median). We call a set \( Y \) a \textit{c-approximate k-median} of \( S \), for some \( c > 0 \), if \( \text{Obj}(S,Y) \leq c \times \text{OPT}(S) \). Further, note, each set \( \{y_1, \ldots, y_k\} \) induces a partitioning (clustering) of \( S \) into \( k \)-clusters \( C_1, \ldots, C_k \), where \( C_i := \{x \in S \mid \Delta(x,y_i) \leq \Delta(x,y_j) \text{ for all } j \neq i \} \) (if for some \( x \in S \), \( \Delta(x,y_i) \leq \Delta(x,y_j) \) for some \( i \neq j \), then break the ties arbitrarily to form \( C_i \’ \)s).

It is worth emphasizing that in the above definition of the \textit{k-median} problem, the \textit{k-median} set \( Y^* \) need not be a subset of the input \( S \). In the literature, this variant is sometimes referred to as the \textit{continuous} \textit{k-median} problem. On the other hand, the discrete variant asks to find a set \( Y^* \) of size at most \( k \), strictly from \( S \) that minimizes the median objective value (over all the subset of \( S \) of size at most \( k \)). It follows directly from the triangle inequality that any optimum \textit{discrete} \textit{k-median} set is a 2-approximate solution to the \textit{continuous} \textit{k-median} problem.

Since in the discrete version, the median points are necessarily from \( S \), by brute force over all the \( O(n^k) \) (where \( |S| = n \)) possibilities, we can compute an optimum solution. We refer to this algorithm as Procedure \textsc{BestFromInput} (Procedure 1). So, the Procedure \textsc{BestFromInput} provides a 2-approximate solution to the \textit{continuous} \textit{k-median} problem. The running time is \( O(n^k + n^2 d \log d) \), since for any \( x, y \in \mathcal{S}_d \), we can compute \( \Delta(x,y) \) in \( O(d \log d) \) time.

\begin{algorithm}
\setstretch{1.15}
\begin{algorithmic}[1]
\Require \( S \subseteq \mathcal{S}_d \).
\Ensure A subset \( Y \subseteq S \) of size at most \( k \).
1: For all pairs of permutations \( x_i, x_j \in S \), compute \( \Delta(x_i,x_j) \)
2: \Return \( \arg \min_{Y \subseteq S, |Y| \leq k} \text{Obj}(S,Y) \).
\end{algorithmic}
\caption{\textsc{BestFromInput}(\( S, k \)).}
\end{algorithm}

\section{Approximation Algorithm for 1-Median}

\begin{theorem}
There is a deterministic polynomial-time algorithm that, given a set \( S \) of \( n \) permutations over \([d]\), finds a 1.999-approximate median.
\end{theorem}

\footnote{\textit{Note:} These formulations are equivalent.}

\begin{proof}
\end{proof}

\textit{Proof:}

\begin{proof}
\end{proof}
Description of the algorithm. Let us start with the description of our algorithm. Our algorithm consists of two procedures. The first procedure is \textsc{BestFromInput} (by setting \(k = 1\)), which simply outputs a permutation \(\tilde{y} \in S\) with the minimum median objective value among all the inputs, i.e., \(\tilde{y} = \arg \min_{y \in S} \sum_{x \in S} \Delta(x, y)\). The second procedure enumerates all subsets of \(S\) of size five (i.e., 5-tuples of input permutations) and runs the procedure \textsc{MedianReconstruct}. For a subset \(T \subseteq S\), \textsc{MedianReconstruct} works as follows: It constructs a directed graph \(H\) with vertex set \([d]\) and edge set
\[
E(H) := \{(a, b) \mid a \text{ appears before } b \text{ in at least three permutations of } T\}.
\]
Observe, the graph \(H\) is a tournament\(^4\), but may not be acyclic. Next, the procedure iterates over all the vertices and while iterating over a vertex \(v\), it finds a shortest cycle containing \(v\) and deletes all its vertices (along with all the incident edges). Let \(H'\) be the final resulting acyclic graph. Then the procedure performs a topological sorting on the vertices of \(H'\) and let \(\tilde{x}'\) denote the sorted ordering. Finally, it appends the remaining symbols \(([d] \setminus V(H'))\) at the end of \(\tilde{x}'\) in an arbitrary order and outputs the resulting permutation \(\tilde{x}\).

\begin{algorithm}
\caption{\textsc{MedianReconstruct}(\(T\)).}
\begin{algorithmic}[1]
\Require \(T \subseteq S\).
\Ensure A permutation \(\tilde{x}\) over \([d]\).
\State \(H \leftarrow ([d], E)\) where 
\State \(E = \{(a, b) \mid a \text{ appears before } b \text{ in at least three permutations of } T\}\)
\ForAll \(v \in [d]\)
\State \(C_{\text{min}} \leftarrow \text{cycle of minimum length containing } v \text{ in } H\)
\State \(H = H \setminus V(C_{\text{min}})\)
\EndFor
\State \(H' \leftarrow H\)
\State \(\tilde{x}' \leftarrow \text{permutation formed by topological ordering of } V(H')\)
\State \(\tilde{x} \leftarrow \text{permutation formed by appending to } \tilde{x}' \text{ the symbols } [d] \setminus V(H') \text{ in an arbitrary order}\)
\State \Return \(\tilde{x}\).
\end{algorithmic}
\end{algorithm}

For a subset \(T \subseteq S\), let us denote the output of \textsc{MedianReconstruct} by \(\tilde{x}_T\). Consider the set
\[
M = \{\tilde{y} \cup \{\tilde{x}_T\} \mid \text{for all } T \subseteq S \text{ such that } |T| = 5\}.
\]
The final algorithm \textsc{ApproxMedian}(\(S\)) (Algorithm 3) outputs the best permutation \(z\) among the set \(M\) that minimizes the median objective value, i.e., \(z = \arg \min_{y \in M} \sum_{x \in S} \Delta(x, y)\).

Running time analysis. Note, each \(\Delta(x, y)\) computation takes \(O(d \log d)\) time. Then the first procedure \textsc{BestFromInput} takes only \(O(n^2 d \log d)\) time. There are at most \(O(n^5)\) subsets of \(S\) of size exactly five. For each such subset, the \textsc{MedianReconstruct} procedure takes \(O(d^2)\) time to construct the graph \(H\). Then, computing a minimum length cycle passing through a vertex \(v\) at each iteration takes \(O(d^2)\) time. Since it iterates over all the vertices \(v \in [d]\), the running time for the whole cycle removal step is \(O(d^3)\). The topological ordering can be performed in \(O(d^2)\) time. So the running time of \textsc{MedianReconstruct} is \(O(d^3)\). Hence, the overall running time of the final algorithm \textsc{ApproxMedian} is \(O(n^5 d^3)\). Later in Section 4, we will comment on how to reduce the running time to \(\tilde{O}(d^3)\).

\(^4\) A directed graph is called a tournament if between every pair of vertices there is a directed edge.
As a consequence, we get that

\[ \bar{\alpha} \]

Let us consider the set of

\[ \text{Proof.} \]

Otherwise,

\[ \text{Lemma 3.2.} \]

\[ \text{Ensure: A subset } Y \subseteq S \text{ of size at most } k. \]

\[ 1: \text{Initialize an empty set } M \]

\[ 2: \tilde{y} \leftarrow \text{BestFromInput}(S, 1) \]

\[ 3: \text{Add } \tilde{y} \text{ to } M \]

\[ 4: \text{For all the subsets } T \subseteq S \text{ of size } 5, \text{ run } \text{MedianReconstruct}(T) \text{ and add the output to } M \]

\[ 5: \text{return } \arg \min_{y \in M} \text{Obj}(S, y). \]

**Algorithm 3**\textbf{APPROX\textsc{Median}}\((S)\).

\[ \textbf{Require: } S \subseteq S_i. \]

\[ \textbf{Ensure: A subset } Y \subseteq S \text{ of size at most } k. \]

\[ 1: \text{Initialize an empty set } M \]

\[ 2: \tilde{y} \leftarrow \text{BestFromInput}(S, 1) \]

\[ 3: \text{Add } \tilde{y} \text{ to } M \]

\[ 4: \text{For all the subsets } T \subseteq S \text{ of size } 5, \text{ run } \text{MedianReconstruct}(T) \text{ and add the output to } M \]

\[ 5: \text{return } \arg \min_{y \in M} \text{Obj}(S, y). \]

**Analyzing the approximation factor.** Suppose \( S = \{x_1, x_2, \cdots, x_n\} \). Let \( x^* \) be an arbitrary optimal median of \( S \). So, \( \text{OPT}(S) = \sum_{x_i \in S} \Delta(x_i, x^*) \). For each \( x_i \in S \), consider an arbitrary optimal alignment between \( x_i \) and \( x^* \), and let \( I_{x_i} \) (or for brevity, \( I_i \)) denote the set of unaligned symbols (\( \subseteq [d] \)) with respect to this alignment. Recall, by the definition, \( \Delta(x_i, x^*) = |I_i| \) for each \( x_i \in S \). WLOG assume, \( |I_1| \leq |I_2| \leq \cdots \leq |I_n| \).

Consider \( \epsilon = 0.03319 \) and \( \alpha = \epsilon/11 \). WLOG assume,

\[ |I_1| \geq (1 - \alpha) \text{OPT}/n. \] (1)

Otherwise,

\[ \text{Obj}(S, x_1) \leq \sum_{x_i \in S} \Delta(x_i, x_1) \]

\[ \leq \sum_{x_i \in S} (\Delta(x_i, x^*) + \Delta(x^*, x_1)) \] (by triangle inequality)

\[ \leq (2 - \alpha) \text{OPT}. \] (2)

It is straightforward to see that \( \text{Obj}(S, \tilde{y}) \leq \text{Obj}(S, x_1) \), and thus the final output \( z \) satisfies \( \text{Obj}(S, z) \leq (2 - \alpha) \text{OPT} \). So from now, we assume Equation 1.

**Lemma 3.2.** Consider \( \epsilon = 0.03319 \) and \( \alpha = \epsilon/11 \). Then one of the following holds:

1. Either there are five inputs \( x_{i_1}, x_{i_2}, x_{i_3}, x_{i_4}, x_{i_5} \) (with \( i_1 < \cdots < i_5 \)) such that for any two \( r, \ell \in \{i_1, i_2, i_3, i_4, i_5\} \) with \( r > \ell \), \( |I_r \cap I_\ell| \leq \epsilon |I_r| \) and \( |I_{i_4}| \leq (1 + \alpha) \text{OPT}/n; \)
2. Or there exists \( x_j \in S \) such that \( \text{Obj}(S, x_j) \leq 1.999 \cdot \text{OPT} \).

**Proof.** Let us consider the set of far points, defined as

\[ F := \{ x_i \in S \mid |I_i| \geq (1 + \alpha) \text{OPT}/n \}. \]

Let \( F := S \setminus F \). For any subset \( R \subseteq S \), let us define \( \text{OPT}_R := \sum_{x_i \in R} \Delta(x_i, x^*). \)

Recall, we assume that \( |I_1| \leq \cdots \leq |I_n| \). Then it is straightforward to see that for all \( x_i \in S \), \( \Delta(x_i, x_1) \leq \Delta(x_i, x^*) + \Delta(x^*, x_1) = |I_i| + |I_1| \leq 2 |I_1| = 2 \Delta(x_i, x^*). \) Further, observe, by an averaging argument, \( |I_i| \leq \text{OPT}/n \). Then for all \( x_i \in F \),

\[ \Delta(x_i, x_1) \leq |I_i| + |I_1| \] (by the triangle inequality)

\[ \leq |I_i| + \text{OPT}/n \]

\[ \leq |I_i| + |I_1|/(1 + \alpha) \leq (2 - \alpha/2) |I_i|. \]

As a consequence, we get that

\[ \text{Obj}(S, x_1) \leq 2 \text{OPT}_F + (2 - \alpha/2) \text{OPT}_F = 2 \text{OPT} - \frac{\alpha}{2} \text{OPT}_F. \] (3)
So if $\text{OPT}_F \geq \frac{2}{3}\text{OPT}$, we get that $\text{Obj}(S, x_1) \leq 1.999 \cdot \text{OPT}$. So from now, assume

$$\text{OPT}_F < \frac{2}{3}\text{OPT}. \quad (4)$$

Next, consider the following procedure $A$ that processes $x_1, \ldots, x_n \in S$ one by one. Initialize a set $T \leftarrow x_1$. For each $x_i \in S$, if for all $x_j \in T$, $|I_j \cap I_i| \leq \epsilon|I_j|$, add $x_i$ in $T$. Break when $|T| \geq 5$.

It is worth noting that the above procedure is considered only for the sake of analysis. Now when the above procedure terminates, suppose $T = \{x_{i_1}, x_{i_2}, \ldots, x_{i_n}\}$, where $1 = i_1 < i_2 < \cdots < i_5$, and $x_{i_4} \in F$. Then clearly it satisfies Item 1 of the statement of the lemma.

If not, then either $x_{i_4} \in F$ or $|T| \leq 4$. By the procedure $A$, $x_{i_4} \in F$ implies that for all $x_i \in F$, there exists $x_j \in T \cap F$ such that $|I_j \cap I_i| > \epsilon|I_j|$. Then by a simple averaging argument, there exists $j \in T \cap F$ and $R \subseteq F$ such that

$$\text{OPT}_R \geq \frac{\text{OPT}_F}{2(1 - \epsilon)}; \quad \text{and}$$

$$\text{For all } x_i \in R, |I_i \cap I_j| > \epsilon|I_j|. \quad (i)$$

Consider this $j \in T \cap F$ and $R \subseteq F$. It follows from the triangle inequality that

(i) For all $x_i \in F$, $\Delta(x_i, x_j) \leq 2\Delta(x_i, x^*)$;

(ii) For all $x_i \in F$, $\Delta(x_i, x_j) \leq (2 + 3\alpha)\Delta(x_i, x^*)$;

(iii) For all $x_i \in R$, $\Delta(x_i, x_j) \leq (2 - \epsilon)\Delta(x_i, x^*)$.

To see this, observe, for any two $x_i, x_r \in S$, $\Delta(x_i, x_r) \leq |I_j| + |I_i| - |I_i \cap I_r|$. Now, since for all $x_i \in F$, $|I_i| \leq |I_j|$, the first item follows. For the second item, observe, for all $x_i \in F$, $|I_i| \leq (1 + \alpha)|I_j|$ (since $|I_j| \geq (1 - \alpha)\text{OPT}/n$ by assumption Equation 1). For the third item, note, for all $x_i \in R$, $|I_i \cap I_j| > \epsilon|I_j|$, and further $|I_j| \leq |I_i|$ (by the description of procedure $A$).

Thus

$$\text{Obj}(S, x_j) = (2 - \epsilon)\text{OPT}_R + 2\text{OPT}_F + (2 + 3\alpha)\text{OPT}_{F \setminus R}$$

$$\leq 2\text{OPT} - (\frac{\epsilon + 3\alpha}{2} - 3\alpha)\text{OPT}_F \quad \text{ (since } \text{OPT}_R \geq \frac{\text{OPT}_F}{3})$$

$$\leq (2 - 5\alpha/11)\text{OPT} \quad \text{ (by Equation 4 and } \epsilon = 11\alpha)$$

$$\leq 1.999 \cdot \text{OPT} \quad \text{ (for } \alpha = \epsilon/11 = 0.03319/11).$$

When $x_{i_4} \in F$, but $|T| \leq 4$, in a similar way we can argue that there exists $x_j \in T$ and $R \subseteq S$ such that

$$\text{OPT}_R \geq \frac{\text{OPT}}{4}; \quad \text{and}$$

$$\text{For all } x_i \in R, |I_i \cap I_j| > \epsilon|I_j|. \quad (i)$$

Hence, again, we can argue as before that

(i) For all $x_i \in F$, $\Delta(x_i, x_j) \leq 2\Delta(x_i, x^*)$;

(ii) For all $x_i \in F$, $\Delta(x_i, x_j) \leq (2 + 3\alpha)\Delta(x_i, x^*)$;

(iii) For all $x_i \in R$, $\Delta(x_i, x_j) \leq (2 - \epsilon)\Delta(x_i, x^*)$.

Thus

$$\text{Obj}(S, x_j) = (2 - \epsilon)\text{OPT}_R + 2\text{OPT}_{F \setminus R} + (2 + 3\alpha)\text{OPT}_{F \setminus R}$$

$$= 2\text{OPT} - \epsilon\text{OPT}_R + 3\alpha\text{OPT}_{F \setminus R} \leq 2\text{OPT} - \epsilon\text{OPT}_R + 3\alpha(\text{OPT} - \text{OPT}_R) = 2\text{OPT} - (\frac{\epsilon + 3\alpha}{4} - 3\alpha)\text{OPT} \quad \text{ (since } \text{OPT}_R \geq \frac{\text{OPT}}{4})$$

$$\leq (2 - \alpha/2)\text{OPT} \quad \text{ (by setting } \epsilon = 11\alpha)$$

$$\leq 1.999 \cdot \text{OPT} \quad \text{ (for } \alpha = \epsilon/11 = 0.03319/11).$$

This concludes the proof. \(\blacktriangleleft\)
Clearly, if there exists \( x_j \in S \) such that \( \text{Obj}(S, x_j) \leq 1.999 \cdot \text{OPT} \), then
\[
\text{Obj}(S, z) \leq \text{Obj}(S, \tilde{y}) \leq \text{Obj}(S, x_j) \leq 1.999 \cdot \text{OPT}.
\]
So it only remains to show that if there are five inputs \( x_{i_1}, x_{i_2}, x_{i_3}, x_{i_4}, x_{i_5} \) (with \( i_1 < \cdots < i_5 \)) such that
1. For any two \( r, \ell \in \{i_1, i_2, i_3, i_4, i_5\} \) with \( \ell > r \), \( |I_r \cap I_\ell| \leq \epsilon |I_r| \), and
2. \( |I_{i_4}| \leq (1 + \alpha) \text{OPT}/n \),
then \( \text{Obj}(S, z) \leq 1.999 \cdot \text{OPT} \). For that purpose, consider \( T = \{ x_{i_1}, x_{i_2}, x_{i_3}, x_{i_4}, x_{i_5} \} \), and the output \( \tilde{x}_T \) of \text{MEDIANRECONSTRUCT} on input \( T \). We want to claim that \( \text{Obj}(S, \tilde{x}_T) \leq 1.999 \cdot \text{OPT} \), and hence \( \text{Obj}(S, z) \leq \text{Obj}(S, \tilde{x}_T) \leq 1.999 \cdot \text{OPT} \), which will complete the analysis.

\[\triangleright \text{Claim 3.3.} \] For \( T = \{ x_{i_1}, x_{i_2}, x_{i_3}, x_{i_4}, x_{i_5} \} \), \( \text{Obj}(S, \tilde{x}_T) \leq 1.999 \cdot \text{OPT} \).

\textbf{Proof.} Let us define the set of bad symbols as
\[
B := \bigcup_{r \neq \ell \in \{i_1, \cdots, i_5\}} (I_r \cap I_\ell).
\]

Note,
\[
|B| \leq 4\epsilon |I_{i_1}| + 3\epsilon |I_{i_2}| + 2\epsilon |I_{i_3}| + \epsilon |I_{i_4}| \leq 10\epsilon |I_{i_4}|
\tag{5}
\]
(recall, by our assumption \( |I_{i_1}| \leq \cdots \leq |I_{i_4}| \)). Let us define the set of good symbols as \( G := [d] \setminus B \). Observe, a symbol \( a \in G \) if and only if \( a \in I_r \) for at most one \( r \in \{i_1, \cdots, i_5\} \).

Hence, for any two distinct \( a, b \in G \), for at least three \( x \in T \), \( a, b \notin I_x \), in other words, both \( a, b \) are aligned between \( x \) and \( x^* \). Thus, by the construction of \( H \), \( (a, b) \in E(H) \) if \( a \) appears before \( b \) in \( x^* \), for every distinct \( a, b \in G \).

Next, observe that for any subset \( V \subseteq [d] \), for any vertex \( v \in [d] \) and a shortest cycle \( C \) containing \( v \) in the subgraph \( H - V \),
1. \( C \) must contain at least one bad symbol (i.e., from \( B \));
2. \( C \) must be of length 3.

The first condition is straightforward since a set of good symbols cannot form a cycle (because they form a directed path according to their ordering in \( x^* \)). For the second condition, suppose \( C \) is of length strictly greater than 3 and \( v, a, b \) are three consecutive vertices in \( C \). Observe, between any two vertices in the subgraph \( H - V \), there is a directed edge (because for any two symbols \( a_1, a_2 \), either \( a_1 \) appears before \( a_2 \) or \( a_2 \) appears before \( a_1 \) in at least three permutations out of five). So either the edge \( (b, v) \) or \( (v, b) \) must be in the subgraph \( H - V \). In the first case, we get a length 3 cycle consisting of \( v, a, b \), and in the second case, we get a shorter cycle (by taking the edge \( (v, b) \) while bypassing the vertex \( a \) of \( C \)) contradicting the fact that \( C \) is a shortest cycle containing \( v \).

Due to the above observation, after iterative cycle removal in \text{MEDIANRECONSTRUCT}, we get a subgraph \( H' \) with \( |V(H') \cap G| \geq |G| - 2|B| = d - 3|B| \). Since \( \tilde{x}' \) is a topological ordering of the vertices in \( V(H') \), the length of a longest common subsequence between \( x^* \) and \( \tilde{x}' \) must be
\[
|\text{LCS}(\tilde{x}', x^*)| \geq |V(H') \cap G| \geq d - 3|B|.
\]

Hence,
\[
\Delta(\tilde{x}_T, x^*) = d - \text{LCS}(\tilde{x}_T, x^*)
\leq d - \text{LCS}(\tilde{x}', x^*)
\leq 3|B| \leq 30\epsilon |I_{i_4}|
\tag{by Equation 5}.
\]
Recall, $|I_{\alpha}| \leq (1 + \alpha)OPT/n$. So, we get that
\[
\text{Obj}(S, \tilde{x}_T) = \sum_{x_i \in S} \Delta(x_i, \tilde{x}_T) \leq \sum_{x_i \in S} (\Delta(x_i, x^*) + \Delta(x^*, \tilde{x}_T)) \quad \text{(by triangle inequality)} \\
\leq (1 + 30\epsilon(1 + \alpha))OPT \\
\leq 1.999 \cdot OPT
\]
for the choice of $\alpha = \epsilon/11 = 0.03319/11$.

### 3.1 Extension to the $k$-Median

Now we argue that our algorithm framework described so far can be extended to the $k$-median problem. More specifically, we show the following result.

**Theorem 3.4.** There is a deterministic algorithm, that given a set $S$ of $n$ permutations over $[d]$, finds a 1.999-approximate $k$-median in time $n^{O(k)}d \log d + O(n^5d^3)$.

We would like to highlight that the above running time can further be improved to $(k \log(nd))^O(k)nd^3$ by running the algorithm described in this section on a sample set. However, such a modification slightly worsens the approximation factor. We describe the sampling procedure in detail in Section 4.

**Proof.** Here we briefly describe how to extend our median algorithm to the $k$-median problem. We build a set $M$ by adding the output permutations of the procedure $\text{MedianReconstruct}(T)$ for all the subsets $T \subseteq S$ such that $|T| = 5$. Then, we also add the permutations in the input set $S$ to $M$. Finally, we output a subset $\hat{Y}$ of $M$, of size at most $k$ that minimizes the $k$-median objective value, i.e., $\hat{Y} = \arg\min_{Y \subseteq M, |Y| \leq k} \text{Obj}(S, Y)$. We refer to this algorithm as $\text{Approx k-Median}$.

By the construction, $|M| = O(n^5)$, and by the running time analysis of the procedure $\text{MedianReconstruct}$, constructing the set $M$ takes time $O(n^5d^3)$. The final step that outputs a subset of $M$ which minimizes the objective function, takes $n^{5k+1}d \log d$ time. So the overall running time is $n^{O(k)}d \log d + n^5d^3$.

To argue about the approximation guarantee of the above algorithm, let us first consider an arbitrary optimal $k$-median $Y^*$ (which is of size at most $k$). This set $Y^*$ implicitly induces a partitioning of $S$ into at most $k$ clusters $C_1, C_2, \ldots, C_k$ (where some of the $C_i$’s could be empty depending on the size of $Y^*$). WLOG assume, $|Y^*| = k$ and $Y^* = \{y_1^*, \ldots, y_k^*\}$. Then $C_i := \{x \in S \mid \Delta(x, y_i^*) \leq \Delta(x, y_j^*) \text{ for all } j \neq i\}$ (if for some $x \in S$, $\Delta(x, y_i^*) = \Delta(x, y_j^*)$ for two $y_i^* \neq y_j^*$, then break the ties arbitrarily to form $C_i$’s). It is straightforward to see that $y_i^*$ is an optimal median of the set/cluster $C_i$. Then, by the analysis of $\text{ApproxMedian}$ (in the previous section), a permutation $\hat{y}_i$ will be added in $M$ that is a 1.999-approximate median of the cluster $C_i$, for all $i \in [k]$. Since in the final step we output a subset $\hat{Y}$ of $M$, of size at most $k$ that minimizes the $k$-median objective value, clearly it would be a 1.999-approximate $k$-median of the input set $S$. \hfill $\blacktriangle$

### 3.2 Extension to the $k$-median with outliers

The algorithm described in the previous section can produce a 1.999-approximate median even in the presence of outliers. In the $k$-median with outliers problem, we are given a parameter $p \in [0, 1)$. Given an input set $S$, the problem then asks to find a set of size at most $k$ (which is not necessarily a subset of $S$) that minimizes the $k$-median objective value of a subset of $S$ of size at least $(1 - p)|S|$. Formally, we define the objective value of $S$ with
respect to a set \( Y \subseteq S_d \) as \( \text{Obj}_p(S, Y) := \min_{S' \subseteq S : |S'| \geq (1-p)|S|} \text{Obj}_p(S', Y) \). The problem asks to output a set \( Y^* \subseteq S_d \) that minimizes \( \text{Obj}_p(S, Y) \). Note, \( \text{Obj}_{p, 0} \) is the same as the standard \( k \)-median objective function \( \text{Obj}_p \) (as defined in Section 2).

**Theorem 3.5.** There is a deterministic algorithm, that given a set \( S \) of \( n \) permutations over \([d]\) and \( p \in [0, 1] \), finds a 1.999-approximate solution to the \( k \)-median with outliers problem with parameter \( p \), in time \( n^{O(k)} d \log d + n^5 d^3 \).

**Proof.** The algorithm is the same as that without outliers, i.e., that described in Theorem 3.4, with the only exception that now we consider \( \text{Obj}_{p, 0} \) as the objective function. So the running time also remains the same. To argue about the approximation guarantee, let us first consider an arbitrary optimal \( k \)-median \( Y^* \). Let the corresponding subset of \( S \) be \( S^* \) (i.e., \( S^* = \arg \min_{S \subseteq S : |S| \geq (1-p)|S|} \text{Obj}_p(S', Y^*) \)). Then the argument would be exactly the same as that in Theorem 3.4 on the set \( S^* \). ▶

## 4 Streaming Algorithm for Approximating \( k \)-Median

In the streaming model, we are given a set \( S \) of \( n \) permutations \( x_1, x_2, \ldots, x_n \) over \([d]\) that arrive in a stream. Our objective is to design an algorithm that uses space \( O(d \log^{20} n \log^k d) \) and computes \( k \) permutations \( \tilde{y}_1, \ldots, \tilde{y}_k \) over \([d]\) such that \( \sum_{x \in S} \min(\Delta(\tilde{y}_1, x), \ldots, \Delta(\tilde{y}_k, x)) \leq (2 - \delta) \text{OPT} \) for some constant \( \delta > 0 \) in polynomial time. Here \( \text{OPT} \) denotes the optimal objective.

**Theorem 1.1.** There is a (randomized) streaming algorithm that, given a set of permutations \( x_1, x_2, \ldots, x_n \in S_d \) (arriving in streaming fashion), provides a 1.9999995-approximate solution to the (metric) \( k \)-median problem under the Ulam metric, using only \( k^2 d \text{polylog}(nd) \) bits of space, with high probability. Moreover, the algorithm has update time \( (k \log n)^{O(1)} d \log^2 d \) and query time \( (k \log(nd))^{O(k)} d^3 \).

Before proving the above theorem, let us first introduce a few tools which will be critical for our algorithm.

**Coreset and streaming.** One of the important tools to solve the clustering problem is coresets.

**Definition 4.1 ((\( k, \epsilon \))-coreset).** For a set \( S \) of points in an arbitrary metric space \( \mathcal{X} \) and an implicit set \( X \subseteq \mathcal{X} \) (of potential centers/medians), a weighted subset \( P \subseteq S \) (with a weight function \( w : P \rightarrow \mathbb{R} \)) is a \((k, \epsilon)\)-coreset of \( S \) with respect to \( X \) for the \( k \)-median problem if

\[
(1 - \epsilon) \text{Obj}(S, Y) \leq \sum_{x \in P} w(x) \cdot \Delta(x, Y) \leq (1 + \epsilon) \text{Obj}(S, Y)
\]

for all subsets \( Y \subseteq X \) of size at most \( k \).

There are several coreset constructions known in the literature. In this paper, we consider the following coreset construction, which is implied from [25] (and further explained in [7, 11]).

**Theorem 4.2** ([25, 7, 11]). There is an algorithm that, given a set \( S \) of points of an arbitrary metric space \( \mathcal{X} \) and an implicit set \( X \subseteq \mathcal{X} \) (WLOG assume \( S \subseteq X \)), outputs a \((k, \epsilon)\)-coreset of \( S \) with respect to \( X \) for the \( k \)-median problem, of size \( O(\epsilon^{-2} k^2 \log |X|) \).
In this paper, we are interested in solving the $k$-median problem over the Ulam metric in the streaming model. We consider the insertion-only streaming model, where a set of points (in our case, permutations) $x_1, x_2, \ldots, x_n$ arrive one after another in a streaming fashion. By combining Theorem 4.2 and the framework provided by [10], it is possible to build a coreset for the $k$-median problem over an arbitrary metric space using polylogarithmic space. More specifically, we use the following result.

**Theorem 4.3.** There is a streaming algorithm that, given a set $S$ of points of an arbitrary metric space $X$, arriving in an insertion-only stream and an implicit set $X \subseteq X$ (WLOG assume $S \subseteq X$), maintains a $(k, \epsilon)$-coreset of the input with respect to $X$ for the $k$-median problem, by storing at most $O(\epsilon^{-2}k^2 \log |X| \log n)$ points of $S$. Furthermore, the algorithm has worst-case update time of $(\epsilon^{-1}k \log n)^O(1)$.

**Monotone Faraway Sampling (MFS).** Another important tool that we will use to design a streaming algorithm for the $k$-median problem is the monotone faraway sampling (MFS), introduced in [12]. This sampling method allows us to sample “a few” points from an (insertion-only) stream such that the sample set includes a set of candidate medians that achieves $O(1)$-approximation. Although the approximation factor involved is much larger than 2, roughly speaking, it is sufficient for the clusters that contribute a small amount to the overall objective. We use the following result implied from [12].

**Theorem 4.4 ([12]).** There is a streaming algorithm that, given a set $S$ of points of an arbitrary metric space $X$, arriving in an insertion-only stream and parameters $\kappa, \rho \in (0, 1)$, samples a subset $F \subseteq S$ of size $O(k^2(\rho \kappa)^{-1} \log k \log(1 + k\gamma n))$ such that the following holds: Suppose $Y^* = \{y_1^*, \ldots, y_k^*\}$ be an arbitrary optimum $k$-median of $S$ (where $\text{OPT} = \text{Ob}(S, Y^*)$) and let $C_1, \ldots, C_k$ denote the induced clustering of $S$. Then for each $i \in [k]$, there exists a $y_i^* \in F$ such that

$$\sum_{x \in C_i} \Delta(x, y_i^*) \leq 2 \left(1 + \frac{1}{1 - \kappa}\right) \sum_{x \in C_i} \Delta(x, y_i^*) + \rho \frac{\text{OPT}}{k}.$$

Moreover, the algorithm requires both space and update time of $O(k^2(\rho \kappa)^{-1} \log k \log(1 + k\gamma n))$.

All the above algorithms are randomized and err with probability at most $1/10$. Now we are ready to describe our streaming algorithm for the $k$-median problem.

**Algorithm Description.** The algorithm is similar to the $k$-median algorithm described in Section 3.1, and we refer to it as APPROX $k$-MEDIANSTREAMING. However, because of the space limitation, instead of storing all the permutations, we run the algorithm on a sample set instead of the whole input. Let us consider the following set $M$ of (implicit) potential $k$-medians: $M$ contains all the input permutations. Further, it also contains all the output of MEDIANRECONSTRUCT($T$) for all $T \subseteq S$ such that $|T| = 5$. Our algorithm works in two phases. In the first step, it samples a set $R$ of $O((\log d)^4 \log d)$ permutations from the stream $x_1, x_2, \ldots, x_n$. Additionally, it also constructs a coreset $(P, w)$ for the set $M$ of (implicit) $k$-medians on input $S$ in a streaming fashion. Then we show using these sampled permutations and the coreset we can compute $k$ permutations $\tilde{y}_1, \ldots, \tilde{y}_k$ such that $\sum_{x \in S} \min(\Delta(\tilde{y}_1, x), \ldots, \Delta(\tilde{y}_k, x)) \leq (2 - \delta)\text{OPT}$ for some constant $\delta > 0$.

**Step 1 (Sampling Algorithm):** Given set $S$ and parameters $\beta, \gamma > 0$ (the values of which are to be fixed later), we sample a set of permutations $R$ from $S$ as follows.

For each $\ell \in \{1, (1 + \gamma), (1 + \gamma)^2, \ldots, d\}$ and $p \in \{1, \frac{1}{1+\gamma}, \frac{1}{(1+\gamma)^2}, \ldots, \frac{1}{\ell}\}$, we create a set $S_{\ell, p} \subseteq S$ as follows:
Step i) For each $x_i$, discard $x_i$ with probability $1 - p$.

Step ii) If $\forall x_j \in S_{t,p}$, $\Delta(x_i, x_j) \geq \beta \ell$, add $x_i$ to $S_{t,p}$.

Step iii) If $|S_{t,p}| \geq k \log^3 n$, set $S_{t,p} = \emptyset$.

Then set $R = \bigcup_{i \geq 1} S_{t,p}$. 

Step 2 (Monotone Faraway Sampling): Consider parameters $\kappa = 1/3$, $\rho > 0$. Given the input set $S$, we use the monotone faraway sampling (MFS) from Theorem 4.4 due to [12] to get a subset $F \subseteq S$ of size $O(k^2 \log k \log (kn))$ such that the following holds: Suppose $Y^* = \{y_1^*, \ldots, y_k^*\}$ be an arbitrary optimum $k$-median of $S$ (where $\OPT = \OPT(S, Y^*)$) and let $C_1, \ldots, C_k$ denote the induced clustering (to be defined formally later) of $S$. Then for each $i \in [k]$, there exists a $y_i^* \in F$ such that

$$\sum_{x \in C_i} \Delta(x, y_i^*) \leq 2 \left(1 + \frac{1}{1 - \kappa}\right) \sum_{x \in C_i} \Delta(x, y_i^*) + \rho \OPT/k = 5 \sum_{x \in C_i} \Delta(x, y_i^*) + \rho \OPT/k. \ (7)$$

Step 3 (CoreSet Construction): Given a set of input permutations $S$ and a parameter $\lambda > 0$, define a set $M$ containing all input permutations from $S$. Further, it also contains the output of $\text{MedianReconstruct}(T)$ for all $T \subseteq S$ such that $|T| = 5$. Following Theorem 4.3, construct a $(k, \lambda)$-coreset $(P, w)$ for $S$ with respect to the implicit set $M$ of potential medians.

Step 4 (Computing Approximate $k$-median): At the end of the stream, we use $R, F$ and $(P, w)$ to simulate $\text{APPROX k-Median}$. More specifically, we run algorithm $\text{MedianReconstruct()}$ on every subset $T \subseteq R$ of size five, and then add those outputs $\tilde{x}_T$ to a set $\hat{M}$. Next, add all the elements of $R, F$ to $\hat{M}$. Finally, for each $k$-tuple $(y_1, \ldots, y_k) \in \hat{M}^k$, compute $\sum_{x \in P} w(x) \min(\Delta(x, y_1), \ldots, \Delta(x, y_k))$ and output the $k$-tuple that attains the minimum value.

Analyzing the algorithm. For analysis purpose we fix $k$ optimal medians $y_1^*, y_2^*, \ldots, y_k^*$. For $i \in [k]$, define $C_i = \{x \in S | \forall j \in [k], \Delta(x, y_j^*) \leq \Delta(x, y_i^*)\}$. Let $O_i = \OPT(C_i, y_i^*)$. Thus $\OPT = O_1 + O_2 + \ldots, O_k$. We show our sampling algorithm satisfies the following.

Lemma 4.5. Consider $\gamma = 0.1$. For any constant $\zeta > 0$ and every $i \in [k]$, if $O_i \geq \frac{\OPT}{\kappa}$, then $\exists \ell \in \{1, (1 + \gamma), (1 + 2\gamma), \ldots, d\}$ and $p \in \{1, \frac{1}{1 + \gamma}, \frac{1}{1 + 2\gamma}, \ldots, \frac{1}{\ell}\}$ such that $S_{t,p}$ satisfies at least one of the following with high probability.

1. $S_{t,p}$ contains a permutation $y$ where $\OPT(C_i, y) \leq (1.99999 + \beta)O_i$.
2. $S_{t,p}$ contains a subset $T = \{x_1, x_2, x_3, x_4, x_5\}$ such that $\OPT(C_i, x_T) \leq (1.995 + 61\beta)O_i$.

Proof. We start with a few definitions. Let $d_i = \frac{O_i}{|C_i|}$. For some constant $\alpha > 0$, let $C_i^{\text{far}}$ be the set of permutations in $C_i$ whose distance from $y_i^*$ is a fraction more than the average distance $d_i$. Formally we define,

$$C_i^{\text{far}} = \{x | x \in C_i; \Delta(y_i^*, x) > d_i + \alpha d_i\}.$$ 

Let $C_i^{\text{close}}$ be the set of permutations in $C_i$ whose distance from $y_i^*$ is $\alpha$ fraction less than the average distance $d_i$. Formally we define,

$$C_i^{\text{close}} = \{x | x \in C_i; \Delta(y_i^*, x) < d_i - \alpha d_i\}.$$ 

Lastly, let $C_i^{\text{avg}}$ be the set of permutations in $C_i$ whose distance from $y_i^*$ is roughly the average distance $d_i$. Formally we define,

$$C_i^{\text{avg}} = \{x | x \in C_i; d_i - \alpha d_i \leq \Delta(y_i^*, x) \leq d_i + \alpha d_i\}.$$
Case 1: First we consider the case where $|C_i^{\text{close}}| < \frac{|C_i|}{(1+\alpha)\log n}$. Note by definition $|C_i^{\text{far}}| < \frac{|C_i|}{(1+\alpha)\log n}$. Thus $|C_i \setminus C_i^{\text{far}}| \geq \frac{\alpha |C_i|}{(1+\alpha)\log n}$ and $|C_i^{\text{close}}| \geq \frac{\alpha |C_i|}{(1+\alpha)\log n}$. We consider the set $S_{\ell,p}$ where $\ell \leq d_i < (1+\gamma)\ell$ and $\frac{p}{(1+\gamma)^3} < 2\log^2 n \frac{\gamma}{|C_i|} \leq p$. As $|C_i^{\text{close}}| \geq \frac{\alpha |C_i|}{(1+\alpha)\log n}$, and $p \geq \frac{2\log^2 n}{|C_i|}$ using Chernoff bound, with high probability the sampling algorithm samples at least $\frac{\alpha |C_i|}{10(1+\alpha)\log n}$ permutations from $C_i^{\text{close}}$ in Step (i). Let $y$ be such a permutation. If $y$ survives in Step (ii) then it satisfies $O_{\text{bj}}(y, C_i) \leq (2-\alpha)O_i$ by triangle inequality. Otherwise $S_{\ell,p}$ contains a permutation $z$ such that $\Delta(z, y) \leq \beta \ell \leq 3d_i$ (as $\ell \leq d_i$). Thus $O_{\text{bj}}(y, C_i) \leq (2-\alpha + \beta)O_i$. By setting $\alpha = .0005$, we get $O_{\text{bj}}(y, C_i) \leq (1.995 + \beta)O_i$. Next, we show $S_{\ell,p}$ is never modified in Step (iii) and thus $z \in S_{\ell,p}$.

Again with high probability in Step (i) we sample at most $2\log^2 n$ permutations from $C_i$. Lastly, we argue that for each other cluster $C_j \neq C_i$ the following holds. Among the permutation sampled from $C_j$, at most one permutation which is at distance $< \frac{\beta \ell}{2}$ from $y^*_j$ survives at Step (ii). Otherwise let there be two permutations $p_1, p_2 \in C_j$ such that both $p_1, p_2$ are at distance $< \frac{\beta \ell}{2}$ from $y^*_j$ and both of them survive in Step (ii). However, by triangle inequality, their distance is $< \beta \ell$ and we get a contradiction. Thus all but at most one permutation from $C_j$ that survives in Step (ii) will be at a distance of at least $\beta \ell/2$ from $y^*_j$. Let $T_j \subseteq C_j$ be the set of permutations that are at distance $\geq \beta \ell/2$ from $y^*_j$. Thus

$$\sum_{j \in |k|, j \neq i} |T_j| \leq \frac{2}{\beta \ell} \sum_{j \in |k|, j \neq i} O_i \leq \frac{20PT}{\beta \ell} \leq \frac{2kO_i}{\zeta \beta \ell} \leq \frac{2k(1+\gamma)O_i}{\zeta \beta d_i} \leq \frac{4k|C_i|}{\zeta \beta}$$

As $p < \frac{4\log^2 n}{|C_i|}$, with high probability we sample at most $\frac{100k\log^2 n}{\zeta \beta}$ permutations from $S \setminus C_i$. Thus with high probability $|S_{\ell,p}| \leq k \log^3 n$ and it is never modified in Step (iii).

Case 2: Now on, we assume $|C_i^{\text{close}}| < \frac{|C_i|}{(1+\alpha)\log n}$. Again as $|C_i \setminus C_i^{\text{far}}| \geq \frac{\alpha |C_i|}{(1+\alpha)\log n}$ we have $|C_i^{\text{avg}}| \geq \frac{(\log n - 1)|C_i|}{(1+\alpha)\log n} \geq \frac{\alpha (\log n - 1)|C_i|}{(1+\alpha)\log n}$. For the analysis purpose, we fix an optimal alignment between each $x \in C_i$ and $y^*_j$ and let $I_x$ be the set of symbols from $[d]$ that are unaligned in this optimal alignment. For a permutation $x \in C_i^{\text{avg}}$ we define set $C(x) = \{z | z \in C_i^{\text{avg}}, |I_x \cap I_z| \geq \epsilon d_i\}$, where $\epsilon > 0$ is a constant. Let $C_i^{\text{avg, dense}} = \{x | x \in C_i^{\text{avg}}, |C(x)| \geq \frac{|C_i^{\text{avg}}|}{6}\}$. Here we consider the case where $|C_i^{\text{avg, dense}}| \geq \frac{|C_i^{\text{avg}}|}{10}$.

Again we consider the set $S_{\ell,p}$ where $\ell \leq d_i < (1+\gamma)\ell$ and $\frac{p}{(1+\gamma)^3} < 2\log^2 n \frac{\gamma}{|C_i|} \leq p$. As $|C_i^{\text{avg, dense}}| \geq \frac{\alpha (\log n - 1)|C_i|}{(1+\alpha)\log n}$, and $p \geq \frac{2\log^2 n}{|C_i|}$, with high probability the sampling algorithm samples at least $\log n/10$ permutations from $C_i^{\text{avg, dense}}$ in Step (i). Let $y$ be such a permutation. If $y$ survives in Step (ii) then it satisfies $O_{\text{bj}}(y, C_i) \leq 2O_i + (2\alpha + \frac{\log n}{|C_i|} - \frac{\gamma}{\beta}) |C_i \setminus C_i^{\text{far}}| d_i$ by triangle inequality (as in Section 3). Otherwise $S_{\ell,p}$ contains a permutation $z$
such that \( \Delta(z, y) \leq \beta \ell \leq \beta d_i \). Thus \( \mathbf{0} \mathbf{b}(y, C_i) \leq (2 + \beta)O_i + (2\alpha + \frac{2}{2\log n} - \frac{1}{\beta})|C_i| \leq (2 + \beta + 2\alpha + \frac{2}{2\log n} - \frac{1}{\beta})O_i \). By setting \( \epsilon = 0.333 \) and \( \alpha = 0.005 \) we get \( \mathbf{0} \mathbf{b}(y, C_i) \leq (1.999999 + \beta)O_i \).

Following a similar argument as Case 1, we show \( S_{\ell, p} \) is never modified in Step (iii).

**Case 3:** We define set \( C_{i}^{\text{avg}, \text{sparse}} = C_i^{\text{avg}} \setminus C_i^{\text{avg}, \text{dense}} \). Now we consider the case where \( \lambda(C) > 10\log n \). Thus \( |C_{i}^{\text{avg}, \text{sparse}}| \geq \frac{9|C_{i}^{\text{avg}}|}{10} \geq \frac{9\log(n-l)-|C_{i}|}{10(1+\alpha)\log n} \).

Again we consider the set \( S_{\ell, p} \) where \( \ell \leq d_i < (1 + \gamma)\ell \) and \( \frac{p}{(1+\gamma)} < \frac{2\log n}{|C_{i}|} \leq p \). We argue \( S_{\ell, p} \) contains a set of five permutations \( T = \{x_1, \ldots, x_5\} \) such that \( \forall i, j \in [5], |I_x \setminus I_y| \leq (\epsilon + 2\beta)d_i \). For this, we define five events \( e_1, e_2, \ldots, e_5 \). Here \( e_1 \) is the event that at least one permutation is sampled from \( C_{i}^{\text{avg}, \text{sparse}} \). Note as \( |C_{i}^{\text{avg}, \text{sparse}}| \geq \frac{9\log(n-l)-|C_{i}|}{10(1+\alpha)\log n} \)

and \( p \geq \frac{2\log n}{|C_{i}|} \) with high probability we sample at least \( 10\log n \) permutations from \( C_{i}^{\text{avg}, \text{sparse}} \). Let \( x_1 \) be such a permutation. Next given \( e_1 \), let \( e_2 \) be the event that at least one permutation is sampled from \( C_{i}^{\text{avg}, \text{sparse}} \setminus C(x_1) \). As \( |C(x_1)| < \frac{|C_{i}^{\text{avg}}|}{6} \),

\( |C_{i}^{\text{avg}, \text{sparse}} \setminus C(x_1)| \geq \frac{9|C_{i}^{\text{avg}}|}{10} - \frac{|C_{i}^{\text{avg}}|}{6} \geq \frac{1}{10(1+\alpha)\log n} \).

Thus with high probability, we sample at least \( 10\log n \) permutations from \( C_{i}^{\text{avg}, \text{sparse}} \setminus C(x_1) \). Let \( x_2 \) be such a string. In a similar way given \( e_1, e_2, \ldots, e_{j-1} \) (where \( j \leq 5 \)) let \( e_j \) be the event that at least one permutation is sampled from \( C_{i}^{\text{avg}, \text{sparse}} \setminus (C(x_1) \cup \cdots \cup C(x_{j-1})) \). Again using a similar argument as before we can show \( e_j \) is satisfied with high probability and let \( x_j \) be a permutation sampled from \( C_{i}^{\text{avg}, \text{sparse}} \setminus (C(x_1) \cup \cdots \cup C(x_{j-1})) \). Thus all these five events are satisfied together with high probability.

Given \( T = \{x_1, \ldots, x_5\} \), we can construct a permutation \( \tilde{x}_T \) using the MEDIANRECONSTRUCT() algorithm (Algorithm 2), such that \( \mathbf{0} \mathbf{b}(C_i, \tilde{x}_T) \leq (1 + 30(\epsilon + 2\beta)(1 + \alpha))O_i \). By setting \( \epsilon = 0.333 \) and \( \alpha = 0.005 \) we get \( \mathbf{0} \mathbf{b}(C_i, \tilde{x}_T) \leq (1.999999 + 61\beta)O_i \).

Again, following a similar argument as Case 1, we show \( S_{\ell, p} \) is never modified in Step (iii).

---

**Proof of Theorem 1.1.** Given the input set of permutations \( S \) arriving in a stream, we run Algorithm \textsc{Approx}-k-MEDIANSTREAMING with the following parameter setting: \( \beta = 0.00000001, \lambda = 0.00000001 \) and \( \rho = 0.00000001 \). Let the algorithm outputs \( k \) permutations \( y_1, \ldots, y_k \). We show with high probability \( 1 - \sum_{x \in S} \min(\Delta(y_1, x), \ldots, \Delta(y_k, x)) \leq 1.9999995PT \), the algorithm uses \( k^2d\text{polylog}(nd) \) bits space and has update time \( (k\log n)^{O(1)}d^2 \log^2 d \) and query time \( (k\log nd)^{O(k)}d^3 \).

First, we argue the space complexity. In the sampling step (Step 1), there are \( O(\log d) \) different choices for \( \ell \) and \( O(\log n) \) different choices for \( p \). Moreover \( |S_{\ell, p}| < k\log^2 n \). Thus the sampled set \( R \) contains \( O(k^2\log^2 n \log d) \) permutations. Moreover as we consider all subsets \( T \subseteq R \) of size 5, there are at most \( |R|^5 = O(k^5\log^5 n \log^3 d) \) different choices for \( \tilde{x}_T \). However, instead of storing \( \tilde{x}_T \) explicitly, we compute it whenever we use \( \tilde{x}_T \) as a candidate \( k \)-median. Thus storing only set \( R \) is enough. Next, by the MFS algorithm of Theorem 4.4, the sample set \( F \) contains at most \( O(k^2\log k\log (kn)) \) input permutations with high probability. Also, as the size of the implicit set \( M \) can be bounded by \( O(n^5) \), using Theorem 4.3, we can claim that the size of the coreset \( (P, w) \) is \( O(k^2\log^2 n) \). Each permutation can be stored using \( d\log d \) bits. Thus the total space is bounded by \( k^2d\text{polylog}(nd) \).

Next, we show the algorithm has update time \( O(k^2d^2\log^2 (kn) \log^2 d) \) and query time \( (k\log nd)^{O(k)}d^3 \). For a given pair of permutations of length \( d \), their distance can be computed in time \( O(d\log d) \). As \( |R| = O(k^2\log^2 n \log d) \), the update time of the first sampling step is \( O(kd^2\log^2 n \log d^2) \). Next, by the MFS algorithm of Theorem 4.4, the update time to construct set \( F \) and following Theorem 4.3 the update time to construct the coreset
Clustering Permutations: New Techniques with Streaming Applications

(P, w) is \((k \log n)^{O(1)}\). Thus the total update time is \((k \log n)^{O(1)} d \log^2 d\). Next, we argue the query time. As \(|R| = O(k \log^4 n \log d)\), there are at most \(|R|^\epsilon = O(k^4 \log^{20} n \log^5 d)\) different choices for \(\tilde{x}_T\). Thus \(|\hat{M}| = k^{O(1)} \log \log(n)\). Now to bound the space complexity instead of storing \(\tilde{x}_T\) explicitly, we compute it whenever it is used as a candidate k-median.

Using Algorithm MedianReconstruct() it can be constructed in time \(O(d^3)\). Moreover \(|\hat{M}| = k^{O(1)} \log \log(n)\), total number of different candidate k-median is \((k \log(n))^{O(k)}\). Also evaluating the total objective of a candidate k-median using the coreset takes time \(O(k^3 d \log(kn) \log d)\). Thus the total query time can be bounded by \((k \log(n))^O(k)\delta^2\).

Lastly, we argue the approximation guarantee. Let \(S_1 := \{i \in [k] \mid O_i < \frac{\text{OPT}}{10000000}\}\). Thus \(\sum_{i \in S_1} O_i \leq \frac{\text{OPT}}{10000000}\).

\[
\text{Obj}(C, y_i') \leq 5 O_i + \frac{\rho}{k} \text{OPT}.
\]

Thus

\[
\sum_{i \in S_1} \text{Obj}(C, y_i') \leq 5 \sum_{i \in S_1} O_i + \rho \text{OPT} \leq \frac{\text{OPT}}{10000000} \quad \text{(for } \rho = 0.0000001\).
\]

Next following Lemma 4.5 for each \(i \in [k] \setminus S_1\), with high probability, either the sample set \(R\) and thus \(\hat{M}\) contains a permutation \(y\) such that \(\text{Obj}(C_i, y) \leq 1.99999910\) (as \(\beta = 0.0000001\)) or \(R\) contains a tuple \(T = (x_1, \ldots, x_5)\) such that \(\text{Obj}(C_i, \tilde{x}_T) \leq 1.99500610\). In this case in Step (iii) we compute \(\tilde{x}_T\) and add this to \(\hat{M}\). Thus \(\hat{M}\) contains \(k\) permutations with total objective \(\leq 1.99999990\text{OPT} + \frac{\text{OPT}}{10000000} = 1.99999920\text{OPT}\). As we evaluate this using the \((k, \lambda)\) coreset \((P, w)\), the total objective is \(1.9999992(1 + \lambda)\text{OPT} \leq 1.99999950\text{OPT}\) as \(\lambda = 0.0000001\).

5 Improved Space Bound for (1-)Median

In this section, we extend the result of Section 3 in the streaming model. More specifically, suppose a set of input permutations \(x_1, \cdots, x_n \in S_d\) arrive one after another as an insertion-only stream (a permutation, once arrived, cannot be deleted). Here we show a result similar to Theorem 3.1 in the streaming model using \(o(nd)\) bits (note, the input size is \(O(nd \log d)\) bits) of space.

Theorem 5.1. There is a randomized streaming algorithm that, given a set \(S\) of \(n\) permutations over \([d]\) arriving on a stream, finds a 1.9999-approximate median using \(O(d \log d \log^2 n)\) bits of space.

It is worth emphasizing that the space is only needed to maintain at most \(O(\log n)\) permutations at any point of time of the stream.

Description of the algorithm. The algorithm is essentially the same as that described in Section 3, except now we will run the algorithm on a sample set instead of the whole input. Let us consider the following (implicit) set \(M\) (of potential medians): \(M\) contains all the input permutations. Further, it also contains the output of MedianReconstruct\((T)\) for all \(T \subseteq S\) such that \(|T| = 5\). Our algorithm consists of two components. First, it picks each input permutation independently with probability \(\log n/n\). (Alternatively, we can use standard reservoir sampling to sample \(O(\log n)\) permutations uniformly at random.) Let \(R\) denote the sampled set. Second, consider an \(\epsilon \in (0, 1)\) (the value of which is to be fixed
later). Then it constructs a $(1, \epsilon)$-coreset $(P, w)$ (where $P \subseteq S$ and $w : P \to \mathbb{R}$) for $S$ with respect to the (implicit) set $M$, in streaming fashion. At the end of the stream, we use $R$ and $(P, w)$ to simulate $\text{APPROXMEDIAND}. More specifically, we run $\text{MEDIANRECONSTRUCT}$

\begin{align*}
M' = R \cup \{\tilde{x}_T \mid \text{for all } T \subseteq R \text{ such that } |T| = 5\}
\end{align*}

where $\tilde{x}_T$ denotes the output of $\text{MEDIANRECONSTRUCT}(T)$. Finally, for each $y \in M'$, compute $\sum_{x \in P} w(x) \Delta(x, y)$, and output one that attains the minimum value.

Since the analysis of the approximation factor is similar to that for the $k$-median problem (as in Section 4), we omit the details. The only difference is that since now we have only one cluster, we can entirely avoid the “clever” sampling used in Step 1 and the MFS sampling (Step 2) of the algorithm described in Section 4. Instead, we can just use the uniform sampling from the input and then follow a similar argument as used in Lemma 4.5. We focus on analyzing the space usage of our algorithm.

**Space usage.** Clearly, the first sampling step requires $O(\log n)$ input permutations to store (with high probability). By Theorem 4.3, the coreset construction needs to maintain at most $O(\epsilon^{-2} \log |M| \log n) = O(\epsilon^{-2} \log^2 n)$ (since $|M| = O(n^5)$) permutations. So the overall space usage is $O(\epsilon^{-2} d \log d \log^2 n)$ bits of space (since to store each permutation over $[d]$, we need $O(d \log d)$ bits of space).

### References


31:22 Clustering Permutations: New Techniques with Streaming Applications


31:24 Clustering Permutations: New Techniques with Streaming Applications


