Fast $K$-medoids with the $l_1$-norm

Marco Capó, Aritz Pérez, and Jose A. Lozano, Senior Member, IEEE

Abstract—$K$-medoids clustering is one of the most popular techniques in exploratory data analysis. The most commonly used algorithms to deal with this problem are quadratic on the number of instances, $n$, and usually the quality of the obtained solutions strongly depends upon their initialization phase. In this work, we propose an algorithm for the $K$-medoids problem on the $l_1$-norm/Manhattan distance with a computational complexity of $O(n \cdot \max\{\log n, K\} \cdot d)$, along with theoretical guarantees in terms of the accuracy of the obtained approximation. In addition, we propose a cheap split-merge mechanism that can be used to re-start the proposed algorithm after its convergence to a fixed point. Under some mild assumptions, we prove that such a re-start procedure reduces the error of the given fixed point. The work also includes an extensive experimentation, in which we compare our method to the most popular approaches for the $K$-medoids problem: PAM, CLARA and Park’s $K$-medoids. The obtained empirical results show the proposed algorithm to consistently converge to the solutions with the lowest errors, up to two orders of magnitude of relative error lower than the previously mentioned methods, while also requiring the lowest computational running times among them: up to three orders of magnitude lower.

Impact Statement—Clustering is one of the most commonly used and researched Machine Learning tasks. It has been extensively used for market research and customer segmentation, biological data and medical imaging analysis, pattern recognition and social network analysis. In this work, we exploit the mathematical properties of $l_1$-norm to reduce the computational demands of some of the most commonly used clustering methodologies ($K$-medoids solvers) without losing quality on the obtained solutions. The developments presented in this work facilitates the adoption of these kind of methodologies on massive data applications.

Index Terms—Clustering, $K$-medoids, Manhattan distance.

I. INTRODUCTION

Partitional clustering is a basic task in many areas, such as artificial intelligence and pattern recognition [1], [2]. This data analysis technique tries to unveil the inherent structure of a set of points by partitioning it into a number of disjoint groups, called clusters. This is done in such a way that intra-cluster similarity is high and the inter-cluster similarity is low.

Even when the $K$-means algorithm is probably the most common partitional clustering strategy [3], [4], its use is restricted to the usage of a limited amount of metrics, such as the squared Euclidean distance. For this reason, $K$-medoids clustering has emerged as a standard approach in those scenarios where the $K$-means algorithm can not be applied [5]–[7]. Besides the fact that $K$-medoids clustering can be applied with any dissimilarity measure, it also selects instances from the data set as representatives for each cluster (medoids), which are known to be more robust to outliers than other commonly used choices, such as the arithmetic mean [8], [9].

A. $K$-medoids Problem and benefits of the $l_1$-norm

Given a set of $n$ data points (instances) $D = \{x_1, \ldots, x_n\}$ in $\mathbb{R}^d$, an integer $K$ and a dissimilarity measure, $s : \mathbb{R}^d \times \mathbb{R}^d \rightarrow \mathbb{R}_+$, the $K$-medoids problem is to select a set of $K$ medoids from the dataset $D$, $C = \{c_1, \ldots, c_K\} \subseteq D$, so as to minimize the following error function:

$$E^D(C) = \sum_{x \in D} s(x, c_x), \text{ with } c_x = \arg \min_{c \in C} s(x, c)$$

In this work, we focus our analysis when the dissimilarity measure considered is the $l_1$-norm, also known as the Manhattan distance. The relevance of the use of the $l_1$-norm to carry out clustering can be seen in a wide variety of applications such as [8]–[10]. Moreover, there are several articles where clustering problems with the similarity measures based on the $l_1$-norm are considered [10]–[12]. In fact, the $K$-medoids problem itself was initially proposed with the $l_1$-norm [9]. A large part of its popularity is due to the fact that the $l_1$-norm is less likely to be influenced by outliers than other popular metrics, such as the Euclidean distance [8], [10]. Furthermore, it is well-known that this metric offers a better contrast between points than all the other $l_p$-norms, especially in high-dimensional spaces [10].

Unfortunately, it must be highlighted that regardless of the metric considered, finding the solution to the $K$-medoids problem is NP-hard [13]. For this reason, different algorithms have been developed to approximate the solution of such a problem [7], [9], [14]–[16]. In the following sections, we discuss the most popular among them.

1) Partitioning Around Medoids: The Partitioning Around Medoids algorithm [9], or just PAM, is probably the most commonly used heuristic for the $K$-medoids problem [6], [7], [13]. PAM consists of two steps, an initialization phase and a recursive phase. For the first stage, multiple seeding techniques have been proposed with a competitive trade-off between computational complexity and quality of the obtained approximation [7], [17], see for instance: 1) $D^1$-sampling ($K$-means++) [18], where the first medoid is selected uniformly at random and each subsequent initial medoid is chosen with a probability proportional to the distance with respect to the previously selected set of points, 2) MaxMin [19], where each new medoid is the point that is furthest (max) from its nearest (min) existing. Both of these techniques are $O(n \cdot K \cdot d^4)$.

1Assuming that the cost of computing the distance between two points is $O(d)$, as occurs for several popular distance functions such as the Euclidean distance, Manhattan distance, etc.
In the recursive stage of PAM, each medoid is swapped with each possible non-medoid, and then the swap that leads to the largest error decrease is performed until no error reduction can be achieved, i.e., the cost of this algorithm is $O((n-K)^{2}d)$. For this reason, one of the main disadvantages of PAM refers to its high computational cost regardless of the seeding technique used. In order to deal with such a difficulty, different speed-ups to PAM have been proposed that trade optimality for runtime, out of which both CLARA and CLARANS stand out as the most popular \cite{9,20,21,22,23}.

CLARA \cite{20,21} applies PAM on multiple subsamples, keeping the best clustering. Commonly the size of the subsample selected is $n' = 40 + 2 \cdot K$ \cite{7}. In general, if $n' \in O(K)$, then the algorithm is $O(K^{3}d)$. On the other hand, CLARANS \cite{5} works on the entire data set, but only explores a subset of all the possible swaps between medoids and non-medoids. In particular, CLARANS interprets the search space as a high-dimensional hypergraph, where each edge corresponds to the considered swap between a medoid and a non-medoid. Making use of this graph, CLARANS performs a randomized greedy exploration, where the first edge that reduces the error function is followed until no edge can be found after a predefined number of attempts. Unfortunately, its time complexity is still $O(n^{2}d)$ \cite{22}, \cite{23}, which makes it unsuitable for massive data applications.

2) Voronoi Iteration Methods: A $K$-means algorithm-like approach: Another family of algorithms proposed for the $K$-medoids problem is the Voronoi iteration-based techniques \cite{6,24}. In the same manner as the well-known $K$-means algorithm \cite{25}, at each iteration of this approach, the medoid is chosen to be the object with the smallest distance sum to other members of the cluster (update step), then each point is assigned to the nearest medoid (assignment step) until the $K$-medoids error function, Eq.\ref{eq:1} no longer decreases.

The most popular heuristic among this family, for the $K$-medoids problem, is Park’s approach \cite{6,7}. In \cite{6}, the pairwise distance matrix is initially computed and then used to update the medoid for a given cluster, i.e., Park’s approach has a time and space complexity of $O(n^{2}d)$. In terms of the quality of the obtained approximations for the $K$-medoids problem, it is reported in the literature that Voronoi iteration-based heuristics commonly tends to find uncompetitive results when compared to PAM, CLARA and CLARANS, as they do not allow reassigning points to other clusters while changing medoids and, thus, only explores a smaller search space. Moreover, these techniques are also fairly dependent on the initial medoids selected. \cite{7, 26}.

B. Contribution and Related Work

In this work, we propose an efficient and effective algorithm for the $K$-medoids problem with the $l_1$-norm. Our proposal consists of two steps. We begin by proposing a computationally cheap Voronoi iteration-based algorithm that monotonically decreases the $K$-medoids error function, Eq.\ref{eq:1} called FastVoronoi. Unfortunately, as previously mentioned, this kind of techniques usually fails to improve the quality of the approximations obtained via other methods such as PAM and CLARA \cite{7, 26}. Taking this into consideration, then we developed a second phase for our approximation, called VoronoiCorrection. This algorithm consists of a split-merge medoid re-allocation stage that allows us to exploit the clustering information of the current approximation for the $K$-medoids problem to ideally exit the current basin of attraction and facilitate the convergence of the Voronoi iteration-based heuristic to a more competitive approximation.

The cost of the first step of the algorithm, FastVoronoi, is $O(n \cdot \max\{\log n, K\} \cdot d)$, while the second phase, the split-merge medoid re-allocation phase, VoronoiCorrection, is $O(\max\{n \cdot \log n, K^{2}\} \cdot d)$. Therefore, the entire cost of the proposed method is $O(n \cdot \max\{\log n, K\} \cdot d)$, which is more computationally efficient than the state-of-the-art for the $K$-medoids problem, especially for large number of instances. On top of this, throughout the article, we present theoretical guarantees in terms of the quality of the obtained approximation. The rest of the article is organized as follows: We first comment on the deduction of the proposed algorithm and discuss its computational and theoretical guarantees. Afterwards, we compare the proposed method empirically with respect to the state-of-the-art in terms of quality of the obtained solution and computational load.

II. FAST ALGORITHM FOR THE $K$-MEDOIDS PROBLEM

In this section, we formally introduce our approximation for the $K$-medoids problem with the $l_1$-norm (Manhattan $K$-medoids, Algorithm \ref{alg:1}). Given an initialization, our proposal consists of two steps: first, we develop an algorithm of a similar nature to the $K$-median algorithm \cite{1} (FastVoronoi, Algorithm \ref{alg:2}) that monotonically descends the $K$-medoids error, Eq.\ref{eq:1} without requiring the computation of the pairwise distance matrix. Afterwards, in order to further refine such an approximation, we make use of the clustering information of the approximation obtained by FastVoronoi (set of medoids $C'$ and associated clustering $P$) to construct a new set of medoids, via a cluster split-merge step, that is likely to improve the quality of the previous approximation and/or to exit its basin of attraction (VoronoiCorrection, Algorithm \ref{alg:3}).

**Algorithm 1:** Manhattan $K$-medoids: An algorithm for the $K$-medoids problem

**Input:** Data set $D$, number of clusters $K$ and an initial set of centroids $C'$.

**Output:** Solution to the $K$-medoids problem, $C$.

- $(C', P) = \text{FastVoronoi}(D, K, C')$ - see Algorithm \ref{alg:2}

while not stopping criterion do

- Set $C = C'$.
- $(C'', P'') = \text{VoronoiCorrection}(C, P)$ - see Algorithm \ref{alg:3}
- $(C', P) = \text{FastVoronoi}(D, K, C'')$ - see Algorithm \ref{alg:2}

end

**Return** $C$

We would like to point out that, in the first iteration of Algorithm \ref{alg:1}, the while loop is always executed. Afterwards, the proposed stopping criterion is to run the algorithm as long as...
as the error obtained after the execution of FastVoronoi decreases with respect to this value in the previous iteration, i.e., $E^D(C') < E^D(C)$. In the upcoming sections, we extend on both steps of Manhattan $K$-medoids.

A. FastVoronoi: A Voronoi iteration-based algorithm for the $K$-medoids problem

A Voronoi iteration-based process is an algorithm that repeatedly finds the optimal representative of each set of points in the partition and then re-partitions the input according to which of these representatives is closest. The most popular algorithm among this family is the $K$-means algorithm proposed by Lloyd in [25]. In [25], the clustering problem is to minimize within-cluster variances, i.e., the representative is the centroid of a given set following the squared Euclidean distances, and so, the mean optimizes squared errors of a given cluster. Analogously to the $K$-means problem, the $K$-medians clustering problem [1], [27] consists of minimizing the within-cluster error but with respect to the Manhattan distance. In this case, the coordinate-wise median of the cluster is the corresponding centroid, and therefore the Voronoi iteration-based heuristic has a $O(n \cdot \max \{\log n, K\} \cdot d)$ complexity cost rather than $O(n \cdot K \cdot d)$ of the $K$-means algorithm.

This kind of approach has also been used for the $K$-medoids problem [3], [24]. In [6], the first step of the proposal consists of computing the entire distance matrix and selecting the $K$ instances with the smallest sum of distances with respect to the other elements of the data set. Afterwards, the Voronoi iteration is performed using the pairwise distance matrix to update the medoids of each cluster, and so the overall complexity of the algorithm is $O(n^2 \cdot d)$, which makes it unsuitable for massive data scenarios.

1) An efficient Voronoi iteration-based algorithm for the $K$-medoids problem with the $l_1$-norm: In this section, we develop an efficient Voronoi iteration-based algorithm for the $K$-medoids problem with the $l_1$-norm (FastVoronoi). The main advantage of FastVoronoi is its low computational complexity, which is $O(n \cdot \max \{\log n, K\} \cdot d)$. Such a running time complexity is achieved by avoiding the computation of the distances between every pair of points as in [6], which makes it $O(n^2 \cdot d)$. This is a major improvement since it eases the solution of the $K$-medoids problem for large data sets. Furthermore, FastVoronoi makes use of the following result to find the relation between the centroid (coordinate-wise median in the $l_1$-norm case) and the medoid of a cluster, which ultimately allows us to perform the update step of the algorithm in just $O(n \cdot d)$ time:

**Theorem 1.** Given an increasingly sorted vector $x = (x_1, x_2, \ldots, x_n)$, an instance $y \in [x_{i_y}, x_{i_y+1})$ for $l_y \leq \lfloor \frac{n}{2} \rfloor$, or $y \in (x_{i_y}, x_{i_y+1})$, for $l_y \geq \lfloor \frac{n}{2} \rfloor$, and let $m$ be the median of $x$, then the following equality holds:

$$E^x\{\{y\}\} = E^x\{\{m\}\} + |n - 2 \cdot l_y| \cdot |m - y| - 2 \cdot \sum_{j = \min\{\lfloor \frac{n}{2} \rfloor, l_y\} + 1}^{\max\{\lfloor \frac{n}{2} \rfloor, l_y\}} |m - x_j|, \quad (2)$$

**Theorem 1** allows us to relate the $1$-median and $1$-medoid error functions, $E^x\{\{m\}\}$ and $E^x\{\{y\}\}$ respectively, over a sorted vector, $x$. In order to extend its use on a more general setting, one must observe that, if $\{P_1, \ldots, P_K\}$ represents a set of clusters in $D$ induced by a set of points $C = \{c_1, \ldots, c_K\}$, and $P_k$ is the 1D array of all the entries in the $i^{th}$ dimension of $P_k$, then Eq[1] can be re-written as:

$$E^D(C) = \sum_{x \in D} \|x - c_x\|_1 = \sum_{k=1}^{K} \sum_{x \in P_k} \|x - c_k\|_1$$

$$= \sum_{k=1}^{K} \sum_{i=1}^{d} \sum_{x \in P_k} |x^{(i)} - c_k^{(i)}| = \sum_{k=1}^{K} \sum_{i=1}^{d} E^D_k(\{c_k^{(i)}\})$$

In words, we can apply the result obtained in Theorem 1 within the $K$-medoids error function for any dimension, $d \geq 1$, to rapidly compute the medoid of a given cluster. Before doing so, in order to provide more intuition on the previously deduced results, in Figs[12] we show the within-cluster errors obtained for two data sets for the Manhattan and squared Euclidean distances. In particular, in Fig[1] we have a data set with an odd number of points, represented by the cross symbols. The red cross stands for the medoid of such a cluster and, the cyan circle, represents its centroid. On the other hand, in Fig[2] we have a data set with an even number of points and observe that, for the Manhattan distance, we can select infinitely many possible candidates as centroid (any point in the cyan region).

![Figure 1: Within-cluster errors for an odd number of points, for both Manhattan and squared Euclidean distances.](image1)

![Figure 2: Within-cluster errors for an even number of points, for both Manhattan and squared Euclidean distances.](image2)
and $O$ of the complexity of Algorithm 2, the Sorting until the obtained set of medoids remains invariant. In terms when the number of iterations reaches a predefined limit, or
Criteria could be used for this heuristic, such as stopping ($\text{Sorting}$, see Corollary 1.

Theorem 1. Moreover, in the even case, as previously said, we can select infinitely many instances as centroid, see the cyan region in Fig[3]. The reason behind this event can be verified in Eq[2] where, for $l_y = \frac{y}{2}$, the second and third terms of the equality annul, i.e., all elements in the middle interval of each feature have the same within-cluster error. Furthermore, in this setting, we can select two instances as medoid: not only are both selected candidates located in $l_y = \frac{y}{2} + 1$ for one of the features and, at $l_y = \frac{y}{2} + 2$, for the other, but their Manhattan distance with respect to the box of centroids is the same, 0.10, and so their within-cluster errors are equal. It should be remarked at this point that such a complex behavior of the within-cluster error for the Manhattan distance is modelled, for all possible settings, by the result presented in Theorem[1].

In the light of Eqs[2][5] we can efficiently determine the instance of $P_k$, for all $k \in \{1, \ldots, K\}$, with the smallest sum of errors with respect to the other instances of $P_k$ (update step of the Voronoi iteration-based heuristic), see Corollary[1].

**Corollary 1.** Given an increasingly sorted (on each feature) set of points $P \subset \mathbb{R}^d$, then $\arg\min_{y \in P} E^P\{y\} = \arg\min_{y \in P} s^P\{y\}$ where

$$s^P\{y\} = \frac{d}{\|y - \bar{P}\|^2} \sum_{i=1}^{d} \left(2 \cdot |l_{i}(o) - |P_i\}| \cdot y^{(i)} + (-1)^{1(l_{o}(j) > |P_j|)} \cdot 2 \cdot \sum_{j = \min\left\{\frac{|P_j|}{|P_o|} \cdot l_{o}(j) + 1\right\}}^{\max\left\{\frac{|P_j|}{|P_o|} \cdot l_{o}(j) \right\}} x^{(i)}_j \text{ and } x^{(i)}_j \text{ stands for the } j^{th} \text{ entry of } P^{(i)} \text{ for } i \in \{1, \ldots, d\}.$$

From Corollary[1] we know that the medoid update for FastVoronoi can be re-written as selecting $c_k = \arg\min_{y \in P_k} s^P_k\{\{\}\}$. Observe that this update step can be implemented in $O(|P_k| \cdot d)$ time for all $k \in \{1, \ldots, K\}$ and, therefore, the set of $K$ medoids can be updated in $O(n \cdot d)$ time. Taking this into account, in Algorithm[1] we formally present the proposed Voronoi iteration-based algorithm.

Observe that Algorithm[1] decreases the $K$-medoids error function, Eq[1] monotonically. Furthermore, different Stopping Criteria could be used for this heuristic, such as stopping when the number of iterations reaches a predefined limit, or when the error reduction after an iteration is negligible [6][9]. However, for the experimental setting, we run the algorithm until the obtained set of medoids remains invariant. In terms of the complexity of Algorithm[1] the Sorting phase has a $O(n \cdot \log n \cdot d)$ time cost, while Assignment is $O(n \cdot K \cdot d)$ and Update is $O(n \cdot d)$. Therefore, the overall complexity of

\[ \text{FastVoronoi} \text{ is } \mathcal{O}(n \cdot \max\{\log n, K\} \cdot d). \]

**Algorithm 2:** FastVoronoi: A Voronoi iteration-based heuristic for the $K$-medoids problem

**Input:** Data set $D$, number of cluster $K$, initial medoids $C = \{c_1, \ldots, c_K\}$

**Output:** Set of medoids $C = \{c_1, \ldots, c_K\}$ and associated clusters $P = \{P_1, \ldots, P_K\}$

- **Sorting:** Sort each column of $D$ increasingly, see Corollary[1]

while not Stopping Criterion do

- **Assignment:** Generate clusters $P_k = \{x \in D : c_k = \arg\min_{c \in C} \|x - c\| \}$ for all $k \in \{1, \ldots, K\}$

- **Update:** Taking into account Sorting, compute $s^P_k\{\{\}\}$ for all $y \in P_k$ and take, for all $k \in \{1, \ldots, K\}$, $c_k = \arg\min_{y \in P_k} s^P_k\{\{\}\}$

end

Return $C = \{c_1, \ldots, c_K\}$, $P = \{P_1, \ldots, P_K\}$

**B. VoronoiCorrection: A split-merge medoid re-allocation strategy**

As reported in the literature, one of the main disadvantages of Voronoi iteration-based algorithms, when compared to other methods such as PAM, CLARA and CLARANS, is the fact that they tend to converge to uncompetitive approximations of the $K$-medoids problem, as they do not allow point re-assignment to other clusters while changing medoids, and thus only explore a smaller search space than the previously mentioned algorithms [7][26]. Furthermore, the quality of the obtained approximation for these techniques is usually fairly dependent on their initialization phase [7][26]. Taking this into consideration, in this section we propose a split-merge medoid re-allocation technique for the $K$-medoids problem that only uses information associated to the current approximation to the solution of the $K$-medoids problem. We call this algorithm VoronoiCorrection, see Algorithm[3].

**Algorithm 3:** VoronoiCorrection: A split-merge medoid re-allocation strategy

**Input:** Set of medoids $C$ and associated clustering $P$

**Output:** Approximation to the $K$-medoids problem, $C'$.

- **(C', P')** = Split($C$, $P$) - see Algorithm[4]

- **$C' = Merge(C', P')$$ - see Algorithm[5]

Return Set of medoids $C'$

The goal of VoronoiCorrection is to generate a new initialization to the Voronoi iteration-based algorithm (Algorithm[1]) by placing an additional medoid on a certain cluster $P_s$, from which a large amount of error can be reduced (Split -see Algorithm[3]). Afterwards, two other clusters $P_t$ and $P_v$ are merged. These clusters are ideally close to each other, and so their union does not add a significant error to our approximation (Merge -see Algorithm[5]). Under certain conditions (see for instance Theorem[1]), this process itself already generates a set of medoids $C'$ with an associated error lower than that of the previous approximation. Although such a descent may not always occur, this process still generates

\[ \|c - \bar{P}\|^2, \text{ where } \bar{P} \text{ is the center of mass of } P \text{ and } \|\cdot\|^2 \text{ is the squared Euclidean metric [28], [29]}. \]
a re-initialization that is likely to be located at a different basin of attraction and with a $K$-medoids error close to that of the current local minima. This sort of re-initialization technique has been previously stated and shown competitive performances for other clustering problems and metrics, such as $K$-means and ISODATA\cite{29,30}.

1) Split: Selecting a region of the space to add a medoid: The Split algorithm (Algorithm 3) is the phase of CorrectionVoronoi in which we intend to determine a convenient region of the space to place an additional medoid. To do so, one can re-use the distances computed in the last iteration of FastVoronoi to determine the cluster from where the largest error reduction is attained by placing an extra medoid. In particular, in order to quantify such an error decrease, we propose approximating the solution of the 2-medoids problem on each cluster.

Formally speaking, Split approximates a solution to the 2-medoids problem over each cluster $P_k$ via FastVoronoi. This process generates pairs of medoids $\{c_k^1, c_k^2\}$ for $k \in \{1, \ldots, K\}$ and then the new centroid is placed on the cluster $P_k$ satisfying $s = \arg\max_{k \in \{1, \ldots, K\}} E_{P_k}^s(\{c_k^1, c_k^2\}) - E_{P_k}^s(\{c_k^1, c_k^2\})$.

Algorithm 4: Split step

**Input:** Set of medoids, $C = \{c_1, \ldots, c_K\}$, and its corresponding clustering, $P = \{P_1, \ldots, P_K\}$.

**Output:** Updated clustering $P'$ and set of medoids, $C'$.

- Apply FastVoronoi, with $K = 2$, on $P_k$ for $k \in \{1, \ldots, K\}$.
- Select cluster $P_s$ satisfying $s = \arg\max_{k \in \{1, \ldots, K\}} g_k$.

Return

$P' = P \setminus \{P_s\} \cup \{P_1^s, P_2^s\}$, $C' = C \setminus \{c_s\} \cup \{c_1^s, c_2^s\}$.

It must be noted that in Algorithm 3 FastVoronoi can be initialized in multiple ways. In particular, we recommend the use of a $D^1$-sampling seeding to select a candidate for the new medoid, for which we can re-use the distances computed in the previous run of FastVoronoi. In Theorem 2 we provide a theoretical guarantee for such a choice. Hence, the cost of Split is $O(n \cdot \log n \cdot d)$.

Similar heuristics to Algorithm 4 have been developed for other clustering problems. Among them, probably the most famous is the $X$-means algorithm\cite{30}: an approximation to the $K$-means problem for which the number of clusters $K$ is not fixed. In this case, the local 2-means approximation is initialized via a perturbation of each center of mass along a random direction. Moreover, for $X$-means, no theoretical guarantees, in terms of the quality of the approximation, are provided.

2) Merge: Selecting an over-populated region of the space: In Merge (Algorithm 5), we intend to determine a region of the space that is likely to be over-represented by the current set of medoids. In particular, our proposal is to put together the pair of clusters that minimizes the error increase after being fused. One way of estimating such an error increase, without re-computing all the pairwise medoid-non medoid distances and the new medoids, $c_{i,j}$ for $P_{i,j} = P_i \cup P_j$, consists of merging the clusters $P_i$ and $P_j$ that minimize the measure $f_{i,j} = \min\{|P_i|, |P_j|\} \cdot ||c_i - c_j||_1$, which is an upper-bound of $E_{P_{i,j}}^f(\{c_{i,j}\}) - E_{P_i}^f(\{c_i\}) + E_{P_j}^f(\{c_j\})$.

In order to determine the pair of clusters to be fused, Merge only requires the pairwise distances between each medoid and the size of each cluster, which has a computational cost of $O(K^2 \cdot d)$. Afterwards, we compute the medoid of the pair of clusters that are fused as in Update and so, the final cost of VoronoiCorrection is $O(n \cdot \max\{\log n, K^2\} \cdot d)$ and the time complexity of $K$-medoids Manhattan is the same as that of FastVoronoi, i.e., $O(n \cdot \max\{\log n, K\} \cdot d)$, which substantially reduces the computational demands of most $K$-medoids solvers which, due to the computation of the pairwise distances matrix, tend to have a $O(n^2 \cdot d)$ cost.

Algorithm 5: Merge step

**Input:** Set of medoids, $C = \{c_1, \ldots, c_{K+1}\}$, and its associated clustering, $P = \{P_1, \ldots, P_{K+1}\}$.

**Output:** Set of medoids $C'$.

- Select clusters $P_i$ and $P_j$ such that $i, j = \arg\min_{i < k, j \neq K} f_{i,k}$ and compute $c_{i,j}$.

Return Updated set of medoids:

$C' = C \setminus \{c_i, c_j\} \cup \{c_{i,j}\}$.

C. Error descent for VoronoiCorrection

In comparison to other re-initialization techniques of a similar nature to VoronoiCorrection, such as Hartigan’s heuristic\cite{31} and the “first variation” for the spherical $K$-means\cite{32}, our split-merge procedure does not necessarily generate a set of medoids with a lower error than that of the previous approximation. Instead, its goal is to construct a competitive seed that is likely to be located at a different basin of attraction. In any case, the following inequality holds for the new set of centroids obtained by VoronoiCorrection, $C' = C \setminus \{c_{ij}\}$ and $|\{c_{ij}\}| \leq E_{D'}(C') \leq E_D(C)$. In other words, if our split-merge process determines clusters $P_i$, $P_j$, and $P_{i,j}$ for which $f_{i,j} - g_s < 0$, then already our re-initialization, $C'$, has a lower $K$-medoids error than the departing approximation, $C$. Moreover, in the following result, we provide an error descent condition after VoronoiCorrection, which can be verified using only the information of previous approximation.

Theorem 2. Given a set of $K$ medoids, $C = \{c_1, \ldots, c_K\}$, and their corresponding clusterings $\{P_1, \ldots, P_K\}$, if Split divides a cluster $P_s$, taking the corresponding initialization via $D^1$-sampling and $\min\{\min_{i \neq k} \{|P_i|, |P_k|\} : ||c_i - c_k||_1 \}$, then the set of medoids obtained via VoronoiCorrection, $C'$, satisfies $E_D(C') \leq E_D(C)$, on average.

Theorem 2 implies that the larger the average error is in the cluster to be split and the smaller the closest distance between the different pairs of centroids is, then the more likely it is for our split-merge criterion to generate a set of medoids that, without any FastVoronoi iteration, has a $K$-medoids error smaller than that of the preceding approximation. In the experimental section, we can observe that in practice such a
III. Experiments

In this section, we perform a set of experiments so as to analyze the performance in terms of the quality of the obtained approximation with respect to the K-medoids error function and the computational requirements of Manhattan K-medoids (MK), as well as FastVoronoi with no restart (FV) and different well-known approaches commonly used for the K-medoids problem. In particular, we compare the performance of MK and FV, initialized uniformly at random (MK\textunderscore rand and FV\textunderscore rand), $D^1$-sampling (MK\textunderscore D1 and FV\textunderscore D1) and MaxMin (MK\textunderscore MM and FV\textunderscore MM), to PAM, CLARA and Park's heuristic initialized uniformly at random (PAM\textunderscore rand, CLARA\textunderscore rand and Park\textunderscore rand), $D^1$-sampling (PAM\textunderscore D1, CLARA\textunderscore D1 and Park\textunderscore D1) and MaxMin (PAM\textunderscore MM, CLARA\textunderscore MM and Park\textunderscore MM). For Park's heuristic, we also considered the original initialization proposed in [6](Park).

In order to have a better understanding of Manhattan K-medoids, we analyze its performance on a wide variety of real data sets, see Table I. For each data set we also considered a different number of clusters, $K = \{5, 10, 25, 50\}$. Given the random nature of the algorithms and, in order to observe the stability of the obtained results, each experiment has been repeated ten times and the boxplots of all the measured quantities are presented in the Supplementary Material.

Experiments were run on a 64-bit Intel i7 running at 2.30GHz on Ubuntu 20.04.3 LTS.

Table I: Information of the data sets.

<table>
<thead>
<tr>
<th>Data Set</th>
<th>n</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td>Avila</td>
<td>10,436</td>
<td>11</td>
</tr>
<tr>
<td>USPS LeCun</td>
<td>7,290</td>
<td>513</td>
</tr>
<tr>
<td>Anuran Calls</td>
<td>7,193</td>
<td>22</td>
</tr>
<tr>
<td>Cambridge</td>
<td>5,635</td>
<td>57</td>
</tr>
<tr>
<td>Waveform</td>
<td>4,999</td>
<td>41</td>
</tr>
</tbody>
</table>

In Tables I, II and Fig 3, 4, we compare the different methods in terms of the quality of their approximation (relative error with respect to the approximation with the lowest K-medoids error) and computational load (computational running time proportion with respect to the method that converged the fastest) on different numbers of clusters and groups of numbers of instances.

It can be easily observed in Table II and Fig 3 that, in terms of quality of the approximation and regardless of the number of clusters and number of instances, MK consistently generates the approximations with the lowest K-medoids error among all methods besides the seeding strategy considered. This can be better observed in the case of the MaxMin initialization (MM), where MK reduces, on average, up to two orders of magnitude of relative error with respect to all the considered algorithms ($K = 50$), see Table II.

On the other hand, it should be highlighted that FV is the method that converges the fastest for each clustering setting. In terms of error, FV behaves similarly to the other Voronoi iteration-based algorithm: Park. Nonetheless, since FV does not compute the distance matrix, it takes up to two orders of magnitude of running time less than Park for the largest numbers of instances, see Table III. Furthermore, we would like to remark that MK not only generates the most competitive approximations, but it converges quite rapidly to them, see Fig 4. In fact, MK is commonly the algorithm with the second lowest running time, except in the case with the largest number of instances, since the computational complexity of CLARA is not affected by this factor. This is also the case for the largest number of clusters used, where MK on average is re-started over fourteen times, see Table IV. However, since VoronoiCorrection at most modifies three of the current clusters to generate the new seeds, the convergence of the upcoming FastVoronoi runs is fastened; for $K = 50$, MK converges over three times faster than FV, see Table V.

Table II: (Error, Computational time) - average over all data sets and numbers of clusters.

<table>
<thead>
<tr>
<th>Method</th>
<th>$K=5$</th>
<th>$K=10$</th>
<th>$K=25$</th>
<th>$K=50$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Park</td>
<td>$(5.4 \times 10^{-2}, 3.1 \times 10^1)$</td>
<td>$(1.3 \times 10^{-2}, 1.6 \times 10^2)$</td>
<td>$(2.9 \times 10^{-1}, 1.6 \times 10^2)$</td>
<td>$(4.6 \times 10^{-1}, 1.3 \times 10^2)$</td>
</tr>
<tr>
<td>PAM\textunderscore rand</td>
<td>$(4.5 \times 10^{-2}, 2.9 \times 10^2)$</td>
<td>$(7.0 \times 10^{-2}, 1.6 \times 10^2)$</td>
<td>$(1.1 \times 10^{-1}, 1.5 \times 10^2)$</td>
<td>$(1.3 \times 10^{-1}, 1.2 \times 10^2)$</td>
</tr>
<tr>
<td>CLARA\textunderscore rand</td>
<td>$(6.5 \times 10^{-2}, 9.6 \times 10^2)$</td>
<td>$(6.9 \times 10^{-2}, 4.2 \times 10^2)$</td>
<td>$(1.1 \times 10^{-1}, 1.7 \times 10^2)$</td>
<td>$(1.4 \times 10^{-1}, 8.6 \times 10^2)$</td>
</tr>
<tr>
<td>FV\textunderscore rand</td>
<td>$(4.3 \times 10^{-2}, 2.3 \times 10^2)$</td>
<td>$(8.5 \times 10^{-2}, 1.7 \times 10^2)$</td>
<td>$(1.0 \times 10^{-1}, 1.2 \times 10^2)$</td>
<td>$(1.3 \times 10^{-1}, 1.1 \times 10^2)$</td>
</tr>
<tr>
<td>MK\textunderscore rand</td>
<td>$(1.0 \times 10^{-2}, 3.1 \times 10^2)$</td>
<td>$(1.0 \times 10^{-2}, 4.8 \times 10^2)$</td>
<td>$(1.1 \times 10^{-2}, 8.0 \times 10^2)$</td>
<td>$(1.3 \times 10^{-2}, 1.4 \times 10^2)$</td>
</tr>
<tr>
<td>Park\textunderscore D1</td>
<td>$(4.8 \times 10^{-2}, 2.9 \times 10^2)$</td>
<td>$(5.8 \times 10^{-2}, 1.7 \times 10^2)$</td>
<td>$(7.1 \times 10^{-2}, 1.6 \times 10^2)$</td>
<td>$(7.2 \times 10^{-2}, 1.2 \times 10^2)$</td>
</tr>
<tr>
<td>PAM\textunderscore D1</td>
<td>$(3.7 \times 10^{-2}, 9.7 \times 10^2)$</td>
<td>$(5.0 \times 10^{-2}, 4.4 \times 10^2)$</td>
<td>$(7.4 \times 10^{-2}, 1.7 \times 10^2)$</td>
<td>$(8.6 \times 10^{-2}, 8.7 \times 10^2)$</td>
</tr>
<tr>
<td>CLARA\textunderscore D1</td>
<td>$(6.3 \times 10^{-2}, 9.3 \times 10^2)$</td>
<td>$(1.1 \times 10^{-1}, 7.8 \times 10^2)$</td>
<td>$(1.6 \times 10^{-1}, 1.3 \times 10^2)$</td>
<td>$(1.7 \times 10^{-1}, 3.1 \times 10^2)$</td>
</tr>
<tr>
<td>FV\textunderscore D1</td>
<td>$(4.3 \times 10^{-2}, 2.2 \times 10^2)$</td>
<td>$(5.5 \times 10^{-2}, 1.6 \times 10^2)$</td>
<td>$(7.3 \times 10^{-2}, 1.2 \times 10^2)$</td>
<td>$(7.2 \times 10^{-2}, 1.2 \times 10^2)$</td>
</tr>
<tr>
<td>MK\textunderscore D1</td>
<td>$(1.0 \times 10^{-2}, 3.5 \times 10^2)$</td>
<td>$(8.3 \times 10^{-3}, 5.0 \times 10^2)$</td>
<td>$(8.6 \times 10^{-3}, 7.3 \times 10^2)$</td>
<td>$(1.0 \times 10^{-2}, 1.3 \times 10^2)$</td>
</tr>
<tr>
<td>Park\textunderscore MM</td>
<td>$(9.8 \times 10^{-2}, 2.9 \times 10^2)$</td>
<td>$(1.4 \times 10^{-1}, 1.7 \times 10^2)$</td>
<td>$(1.8 \times 10^{-1}, 1.6 \times 10^2)$</td>
<td>$(1.4 \times 10^{-1}, 1.2 \times 10^2)$</td>
</tr>
<tr>
<td>PAM\textunderscore MM</td>
<td>$(9.0 \times 10^{-2}, 1.4 \times 10^2)$</td>
<td>$(1.5 \times 10^{-1}, 6.2 \times 10^2)$</td>
<td>$(1.5 \times 10^{-1}, 3.9 \times 10^2)$</td>
<td>$(1.7 \times 10^{-1}, 1.7 \times 10^2)$</td>
</tr>
<tr>
<td>CLARA\textunderscore MM</td>
<td>$(7.3 \times 10^{-2}, 9.3 \times 10^2)$</td>
<td>$(1.3 \times 10^{-1}, 9.3 \times 10^2)$</td>
<td>$(1.7 \times 10^{-1}, 1.6 \times 10^2)$</td>
<td>$(1.7 \times 10^{-1}, 3.9 \times 10^2)$</td>
</tr>
<tr>
<td>FV\textunderscore MM</td>
<td>$(1.0 \times 10^{-2}, 3.0 \times 10^2)$</td>
<td>$(1.5 \times 10^{-1}, 2.0 \times 10^2)$</td>
<td>$(1.8 \times 10^{-1}, 1.6 \times 10^2)$</td>
<td>$(1.5 \times 10^{-1}, 1.5 \times 10^2)$</td>
</tr>
<tr>
<td>MK\textunderscore MM</td>
<td>$(8.3 \times 10^{-3}, 6.0 \times 10^2)$</td>
<td>$(6.9 \times 10^{-3}, 7.5 \times 10^2)$</td>
<td>$(5.1 \times 10^{-3}, 1.0 \times 10^2)$</td>
<td>$(2.6 \times 10^{-3}, 1.4 \times 10^2)$</td>
</tr>
</tbody>
</table>
Table III: (Error, Computational time) - average over all data sets and numbers of instances.

<table>
<thead>
<tr>
<th>Method</th>
<th>$n \leq 3000$</th>
<th>$3000 &lt; n \leq 6000$</th>
<th>$n &gt; 6000$</th>
</tr>
</thead>
<tbody>
<tr>
<td>PARK</td>
<td>$(4.1 \times 10^{-1}, 3.5 \times 10^3)$</td>
<td>$(1.6 \times 10^{-1}, 1.7 \times 10^2)$</td>
<td>$(1.3 \times 10^{-1}, 3.8 \times 10^2)$</td>
</tr>
<tr>
<td>PARK_rand</td>
<td>$(1.4 \times 10^{-1}, 3.4 \times 10^4)$</td>
<td>$(7.0 \times 10^{-2}, 1.6 \times 10^2)$</td>
<td>$(5.6 \times 10^{-2}, 3.5 \times 10^2)$</td>
</tr>
<tr>
<td>PAM_rand</td>
<td>$(1.5 \times 10^{-1}, 2.9 \times 10^5)$</td>
<td>$(6.7 \times 10^{-2}, 3.3 \times 10^2)$</td>
<td>$(5.4 \times 10^{-2}, 6.0 \times 10^2)$</td>
</tr>
<tr>
<td>CLARA_rand</td>
<td>$(1.4 \times 10^{-1}, 3.2 \times 10^6)$</td>
<td>$(1.2 \times 10^{-1}, 7.9 \times 10^4)$</td>
<td>$(1.2 \times 10^{-1}, 4.4 \times 10^4)$</td>
</tr>
<tr>
<td>MK_rand</td>
<td>$(1.3 \times 10^{-2}, 5.9 \times 10^6)$</td>
<td>$(7.9 \times 10^{-3}, 1.0 \times 10^4)$</td>
<td>$(1.3 \times 10^{-2}, 6.6 \times 10^2)$</td>
</tr>
<tr>
<td>PARK_D1</td>
<td>$(7.3 \times 10^{-2}, 3.4 \times 10^7)$</td>
<td>$(6.8 \times 10^{-2}, 1.7 \times 10^2)$</td>
<td>$(4.4 \times 10^{-2}, 3.5 \times 10^2)$</td>
</tr>
<tr>
<td>PAM_D1</td>
<td>$(8.0 \times 10^{-2}, 2.4 \times 10^5)$</td>
<td>$(6.4 \times 10^{-2}, 3.6 \times 10^2)$</td>
<td>$(4.0 \times 10^{-2}, 6.5 \times 10^2)$</td>
</tr>
<tr>
<td>CLARA_D1</td>
<td>$(1.4 \times 10^{-1}, 3.1 \times 10^6)$</td>
<td>$(1.2 \times 10^{-1}, 8.8 \times 10^4)$</td>
<td>$(1.2 \times 10^{-1}, 4.9 \times 10^4)$</td>
</tr>
<tr>
<td>FV_D1</td>
<td>$(7.4 \times 10^{-2}, 1.5 \times 10^6)$</td>
<td>$(6.3 \times 10^{-2}, 1.4 \times 10^4)$</td>
<td>$(4.4 \times 10^{-2}, 1.7 \times 10^3)$</td>
</tr>
<tr>
<td>MK_D1</td>
<td>$(1.0 \times 10^{-2}, 5.3 \times 10^6)$</td>
<td>$(9.1 \times 10^{-3}, 1.0 \times 10^4)$</td>
<td>$(8.5 \times 10^{-3}, 6.4 \times 10^2)$</td>
</tr>
<tr>
<td>PARK_MM</td>
<td>$(1.8 \times 10^{-1}, 3.4 \times 10^4)$</td>
<td>$(1.5 \times 10^{-1}, 1.7 \times 10^2)$</td>
<td>$(8.7 \times 10^{-2}, 3.5 \times 10^2)$</td>
</tr>
<tr>
<td>PAM_MM</td>
<td>$(1.9 \times 10^{-1}, 4.5 \times 10^4)$</td>
<td>$(1.7 \times 10^{-1}, 4.9 \times 10^2)$</td>
<td>$(1.0 \times 10^{-1}, 1.9 \times 10^2)$</td>
</tr>
<tr>
<td>CLARA_MM</td>
<td>$(1.2 \times 10^{-1}, 1.0 \times 10^2)$</td>
<td>$(1.6 \times 10^{-1}, 1.1 \times 10^1)$</td>
<td>$(1.3 \times 10^{-1}, 7.4 \times 10^0)$</td>
</tr>
<tr>
<td>FV_MM</td>
<td>$(1.8 \times 10^{-1}, 1.7 \times 10^8)$</td>
<td>$(1.6 \times 10^{-1}, 1.5 \times 10^0)$</td>
<td>$(9.9 \times 10^{-2}, 2.8 \times 10^0)$</td>
</tr>
<tr>
<td>MK_MM</td>
<td>$(6.6 \times 10^{-3}, 7.0 \times 10^0)$</td>
<td>$(5.2 \times 10^{-3}, 1.0 \times 10^1)$</td>
<td>$(6.3 \times 10^{-3}, 1.2 \times 10^1)$</td>
</tr>
</tbody>
</table>

Figure 3: Error and running time of each method (average over all data sets and number of clusters).

Figure 4: Error and running time of each method (average over all data sets and number of instances).

Table IV: VoronoiCorrection error descend (# restarts) - average over all data sets.

<table>
<thead>
<tr>
<th>Method</th>
<th>$K=5$</th>
<th>$K=10$</th>
<th>$K=25$</th>
<th>$K=50$</th>
</tr>
</thead>
<tbody>
<tr>
<td>MK_rand</td>
<td>42.7% (1.8)</td>
<td>69.3% (3.2)</td>
<td>89.4% (6.5)</td>
<td>95.8% (12.5)</td>
</tr>
<tr>
<td>MK_D1</td>
<td>50.0% (1.9)</td>
<td>71.3% (3.2)</td>
<td>85.3% (6.1)</td>
<td>93.4% (11.2)</td>
</tr>
<tr>
<td>MK_MM</td>
<td>66.5% (2.7)</td>
<td>76.1% (4.4)</td>
<td>89.6% (8.9)</td>
<td>94.6% (14.3)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Method</th>
<th>$n \leq 3000$</th>
<th>$3000 &lt; n \leq 6000$</th>
<th>$n &gt; 6000$</th>
</tr>
</thead>
<tbody>
<tr>
<td>MK_rand</td>
<td>68.9% (5.6)</td>
<td>80.0% (7.1)</td>
<td>74.0% (5.4)</td>
</tr>
<tr>
<td>MK_D1</td>
<td>80.2% (4.6)</td>
<td>84.0% (7.0)</td>
<td>72.0% (5.2)</td>
</tr>
<tr>
<td>MK_MM</td>
<td>76.6% (6.0)</td>
<td>91.5% (7.7)</td>
<td>77.0% (9.2)</td>
</tr>
</tbody>
</table>

Table V: VoronoiCorrection basin of attraction exit (FastVoronoi # iteration proportion between MK (after first run) and FV) - average over all data sets.

<table>
<thead>
<tr>
<th>Method</th>
<th>$K=5$</th>
<th>$K=10$</th>
<th>$K=25$</th>
<th>$K=50$</th>
</tr>
</thead>
<tbody>
<tr>
<td>MK_rand</td>
<td>78.0% (0.41)</td>
<td>84.8% (0.36)</td>
<td>92.3% (0.33)</td>
<td>96.8% (0.30)</td>
</tr>
<tr>
<td>MK_D1</td>
<td>80.0% (0.41)</td>
<td>85.2% (0.41)</td>
<td>92.0% (0.34)</td>
<td>96.5% (0.30)</td>
</tr>
<tr>
<td>MK_MM</td>
<td>83.2% (0.55)</td>
<td>92.0% (0.54)</td>
<td>94.0% (0.28)</td>
<td>97.5% (0.27)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Method</th>
<th>$n \leq 3000$</th>
<th>$3000 &lt; n \leq 6000$</th>
<th>$n &gt; 6000$</th>
</tr>
</thead>
<tbody>
<tr>
<td>MK_rand</td>
<td>90.7% (0.34)</td>
<td>92.4% (0.37)</td>
<td>91.0% (0.34)</td>
</tr>
<tr>
<td>MK_D1</td>
<td>90.6% (0.39)</td>
<td>92.1% (0.38)</td>
<td>90.9% (0.34)</td>
</tr>
<tr>
<td>MK_MM</td>
<td>95.1% (0.36)</td>
<td>93.0% (0.35)</td>
<td>95.6% (0.22)</td>
</tr>
</tbody>
</table>

Finally, it can be observed that the split-merge mechanism, VoronoiCorrection, succeeded in exiting the basin of attraction of the given approximation: besides the clustering setting, this occurred in over 80% of the cases, see Table.
More importantly, VoroniCorrection was consistently able to improve the quality of the approximation, e.g., for $K = 50$, this occurred in over 93% of the cases, which shows that the re-start process itself was able to reduce the error of the previous approximation without requiring the execution of the corresponding $K$-medoids heuristic. FastVoroni.

Conclusions

In this work, we propose a two-step approximation for the $K$-medoids problem with the $l_1$-norm. First, we develop a fast algorithm for the $K$-medoids problem that decreases monotonically the $K$-medoids error function in $O(n \cdot \max\{\log n, K\} \cdot d)$ time, see Section II-A. Then, we develop a split-merge mechanism that constructs a competitive re-initialization for the $K$-medoids problem by exploiting the clustering information extracted from a given approximation. This extra step allows our approach to converge to fairly competitive approximations, see Theorem 2 by adding a re-start procedure. In addition to the provided theoretical guarantees, the empirical results show our proposal to consistently outperform the state-of-the-art, i.e., PAM, CLARA and Park’s algorithm, in both quality of the obtained solution and running times. It must be remarked that the obtained results are obtained by exploiting the properties of the $l_1$-norm and, due to the high popularity of the $K$-medoids clustering technique, it could be of interest to develop similar results for other family of metrics as well as to other heuristics in which one can easily relate the centroid and the medoid of a given cluster.

Acknowledgment

This research is supported by Basque Government under grant IT1504-22and Ministry of Science and Innovation under grants PID2019-104933GB-I00/AEI/10.13039/501100011033 and PID2022-137442NB-I00 and also by the Basque Government through the BERC 2022-2025 program and by the Ministry of Science and Innovation: BCAM Severo Ochoa accreditation CEX2021-001142-S / MICIN / AEI / 10.13039/501100011033.

References

Marco Capó received his Ph.D. degree in 2019 from the University of the Basque Country and, in 2015, the MSc. degree in Mathematical Modeling Engineering from Universität Hamburg. He is currently a data scientist at Euskal Oxitas Biotek. His research interests are in machine learning and optimization, with a particular focus on unsupervised learning problems and its application to drug discovery.

Aritz Pérez received his Ph.D. degree in 2010 from the University of Basque Country, department of Computer Science and Artificial Intelligence. Currently, he is a postdoctoral researcher at the Basque Center for Applied Mathematics. His current scientific interests include supervised, unsupervised and weak classification, probabilistic graphical models, model selection and evaluation, time series and crowd learning.

Jose A. Lozano received his Ph.D. degree in 1998 from the University of the Basque Country. He became a full professor at the Department of Computer Science and Artificial Intelligence in 2008. Since 2005 he has led the Intelligent Systems Group (ISG) based in the Computer Science School. His research areas are evolutionary computation, machine learning and probabilistic graphical models. He has published 4 books, more than 100 scientific ISI journal articles and about 150 contributions to national and international conferences. These publications have received more than 12,400 citations. Prof. Lozano is an associate editor of IEEE Trans. on Evolutionary Computation and IEEE Trans. on Neural Network and Learning Systems among other prestigious journals.
Supplementary Material

APPENDIX

This section is divided into three parts in which we show the proofs to all the theoretical results presented throughout the manuscript and present additional comments on the empirical results introduced in Section [III]

A. Proofs

In this section, we present the proofs to Theorem [1] and Theorem [2]

**Theorem 1.** Given an increasingly sorted vector \( x = (x_1, x_2, \ldots, x_n) \), an instance \( y \in [x_{i_l}, x_{i_{l+1}}) \), for \( l_y \leq \lceil \frac{n}{2} \rceil \), or \( y \in (x_{i_l}, x_{i_{l+1}}] \), for \( l_y \geq \lceil \frac{n}{2} \rceil \), and let \( m \) be the median of \( x \), then the following equality holds:

\[
E^x(\{y\}) = E^x(\{m\}) + |n - 2 \cdot l_y| \cdot (m - y) - \left| \sum_{j=\min \{\lceil \frac{n}{2} \rceil, l_y + 1\}}^{m} |m - x_j|, \quad (2)
\]

**Proof.** Since \( m \in [x_{\lceil \frac{n}{2} \rceil}, x_{\lfloor \frac{n}{2} \rfloor + 1}] \), we split the analysis into three cases:

1) Assume \( y \in [x_{i_l}, x_{i_{l+1}}) \) with \( l_y < \lceil \frac{n}{2} \rceil \), hence

\[
E^x(\{y\}) = E^x(\{m\}) + |n - 2 \cdot l_y| \cdot (m - y) + \sum_{i=l_y+1}^{\lfloor \frac{n}{2} \rfloor} (x_i - y) - \sum_{i=l_y+1}^{\lfloor \frac{n}{2} \rfloor} (m - x_i) \quad (4)
\]

Observe that \( \sum_{i=l_y+1}^{\lfloor \frac{n}{2} \rfloor} (x_i - y) = \lfloor \frac{n}{2} \rfloor \cdot (x_l - m) + (\lceil \frac{n}{2} \rceil - l_y) \cdot (m - y) \) and so,

\[
E^x(\{y\}) = E^x(\{m\}) + (n - 2 \cdot l_y) \cdot (m - y) - 2 \cdot \sum_{i=l_y+1}^{\lfloor \frac{n}{2} \rfloor} (m - x_i) \quad (5)
\]

2) Assume \( y \in (x_{i_l}, x_{i_{l+1}}] \), with \( l_y > \lceil \frac{n}{2} \rceil \), therefore in this case

\[
E^x(\{y\}) = E^x(\{m\}) + (\lceil \frac{n}{2} \rceil - n + l_y) \cdot (y - m) - \sum_{i=\lceil \frac{n}{2} \rceil + 1}^{l_y} (x_i - m) + \sum_{i=\lceil \frac{n}{2} \rceil + 1}^{l_y} (y - x_i) \quad (6)
\]

Observe that \( \sum_{i=\lceil \frac{n}{2} \rceil + 1}^{l_y} (y - x_i) = \sum_{i=\lceil \frac{n}{2} \rceil + 1}^{l_y} (m - x_i) + (l_y - \lceil \frac{n}{2} \rceil) \cdot (y - m) \) and so,

\[
E^x(\{y\}) = E^x(\{m\}) + (2 \cdot l_y - n) \cdot (y - m) - 2 \cdot \sum_{i=\lceil \frac{n}{2} \rceil + 1}^{l_y} (x_i - m) \quad (7)
\]

3) Finally, in the case \( y \in [x_{\lceil \frac{n}{2} \rceil}, x_{\lfloor \frac{n}{2} \rfloor + 1}] \), we have

\[
E^x(\{y\}) = E^x(\{m\}) + (n - 2 \cdot \lceil \frac{n}{2} \rceil) \cdot (m - y) \quad (8)
\]

Observe that the third case \( \{y \in [x_{\lceil \frac{n}{2} \rceil}, x_{\lfloor \frac{n}{2} \rfloor + 1}] \) matches the first case \( \{y \in [x_{\lceil \frac{n}{2} \rceil}, x_{\lfloor \frac{n}{2} \rfloor + 1}] \) with \( l_y < \lceil \frac{n}{2} \rceil \), and matches the second case \( \{y \in (x_{\lceil \frac{n}{2} \rceil}, x_{\lfloor \frac{n}{2} \rfloor + 1}] \), with \( l_y > \lceil \frac{n}{2} \rceil \), for \( l_y = \lceil \frac{n}{2} \rceil \). In other words, we can write case 1 and case 2 as \( 2y \leq \lceil \frac{n}{2} \rceil \) and \( l_y \geq \lceil \frac{n}{2} \rceil \), respectively. Therefore, we can express, in general, \( f(y) \), as follows:

\[
E^x(\{y\}) = E^x(\{m\}) + |n - 2 \cdot l_y| \cdot |m - y| - \max_{j=\min \{\lceil \frac{n}{2} \rceil, l_y + 1\}} \sum_{j} |m - x_j| \quad (9)
\]

where \( y \in [x_{i_l}, x_{i_{l+1}}) \), for \( l_y \leq \lceil \frac{n}{2} \rceil \), and \( y \in (x_{i_l}, x_{i_{l+1}}] \), for \( l_y \geq \lceil \frac{n}{2} \rceil \).

\( \square \)

In this work, we made use of Theorem [1] to relate the 1-median and 1-medoid error functions over a sorted vector. Afterwards, we re-write the within-cluster K-medoids error function as Eq [10] in Corollary [1] which ultimately allows us to perform the medoid update step of FastVoronoi in \( O(n \cdot d) \) time.

**Corollary 1.** Given an increasingly sorted (on each feature) set of points \( P \subset \mathbb{R}^d \), then \( \arg \min_{y \in \mathcal{P}} E^P(\{y\}) = \arg \min_{y \in \mathcal{P}} s^P(\{y\}) \), where \( s^P(\{y\}) = \sum_{i=1}^{d} (2 \cdot l_{y(i)} - |P|) \cdot y(i) + (-1)^{l_{y(i)}} \cdot \sum_{j=\min \{\lceil \frac{n}{2} \rceil, l_{y(i)} + 1\}}^{\max \{\lceil \frac{n}{2} \rceil, l_{y(i)}\}} x(j) \) and \( x(j) \) stands for the \( j^{th} \) entry of \( P(i) \) for \( i \in \{1, \ldots, d\} \).

**Proof.** The result is straightforward from Theorem [1] and Eq [3]

If \( m \) is the coordinate-wise median of \( P \), we know that

\[
E^P(\{y\}) = \sum_{i=1}^{d} (E^P(i)(\{m(i)\}) + \|P| - 2 \cdot l_{y(i)} \cdot \|m(i) - y(i)\| - \max_{j=\min \{\lceil \frac{n}{2} \rceil, l_{y(i)} + 1\}} \sum_{j=\min \{\lceil \frac{n}{2} \rceil, l_{y(i)} + 1\}}^{\max \{\lceil \frac{n}{2} \rceil, l_{y(i)}\}} \|m(i) - x(j)\| \quad (10)
\]

where \( s^P = \sum_{i=1}^{d} (2 \cdot l_{y(i)} - |P|) \cdot y(i) + (-1)^{l_{y(i)}} \cdot \sum_{j=\min \{\lceil \frac{n}{2} \rceil, l_{y(i)} + 1\}}^{\max \{\lceil \frac{n}{2} \rceil, l_{y(i)}\}} x(j) \) and \( const \) is a constant value that only depends on \( m \).

**Theorem 2.** Given a set of K medoids, \( C = \{c_1, \ldots, c_K\} \), and their corresponding clusterings \( \{P_1, \ldots, P_K\} \), if Split
divides a cluster $P_s$, taking the corresponding initialization via $D^1$-sampling and \( \min \{ \min_{i \neq k \neq s} \| P_i \|, |P_k| \} \cdot \| c_i - c_k \| \leq E_{P_s}((c_s)) \), then the set of medoids obtained via VoronoiCorrection, $C'$, satisfies $E^P(C') \leq E^P(C)$, on average.

**Proof.** In terms of Split, we propose taking the first centroid of the initialization of FastVoronoi $(c_s^0)$ as the medoid of $P_s$, $c_s$, and the second $(c_s^1)$ selected at random via $D^1$-sampling.

We can now take the following bound $E^{P_s}((c_s, c_s^1)) \leq E^{P_s}((c_s^1)) - \| c_s - c_s^1 \|_1$ (in the worst case, at least $c_s^1$ is the closest instance to itself). Moreover,

$$E(E^{P_s}((c_s, c_s^1))) = \sum_{c_j \in P_s} \sum_{x \in P_s} \| x - c_j \|_1 \cdot E^{P_s}((c_s, c_s^1))$$

$$\leq \sum_{c_j \in P_s} \| c_j^2 - c_s \|_1 \cdot \frac{(E^{P_s}((c_s^1)) - \| c_s - c_s^2 \|_1)}{E^{P_s}((c_s))}$$

$$= E^{P_s}((c_s)) - \frac{1}{E^{P_s}((c_s))} \cdot \sum_{c_j \in P_s} \| c_j^2 - c_s \|_1^2$$

This is $g_s = \frac{1}{E^{P_s}((c_s))} \cdot \sum_{c_j \in P_s} \| c_j^2 - c_s \|_1^2$, using the power-mean inequality.\(^4\) We have

$$g_s = \frac{1}{E^{P_s}((c_s))} \cdot \sum_{c_j \in P_s} \| c_j^2 - c_s \|_1^2$$

$$\geq \frac{1}{E^{P_s}((c_s))} \cdot \frac{1}{|P_s|} \cdot (\sum_{c_j \in P_s} \| c_j^2 - c_s \|_1)^2 = \frac{E^{P_s}((c_s))}{|P_s|}$$

After applying Split, the orginal set of clusters \( \{ P_1, \ldots, P_K \} \) is transformed into \( \{ P'_1, \ldots, P'_K \} \), where $P_s = P'_s \cup P'_{s+1}$. Observe that, if, in Merge, we fuse the clusters $P'_i$ and $P'_k$ and $c'_{iK}$ stands for the medoid of $P'_i \cup P'_k$, then

$$E^{P'_i \cup P'_k}((c'_{iK})) \leq \min \{ E^{P'_i \cup P'_k}((c'_i)), E^{P'_i \cup P'_k}((c'_K)) \}$$

$$\leq E^{P'_i}((c'_i)) + E^{P'_k}((c'_K)) + \min \{ |P'_i|, |P'_k| \} \cdot \| c'_i - c'_k \|_1,$$

hence, the error increase can be upper-bounded by $\min \{ |P'_i|, |P'_k| \} \cdot \| c'_i - c'_k \|_1 \leq \min \{ |P'_i|, |P'_k| \} \cdot \| c_i - c_k \|_1$. Then,

$$E(E^D(C')) \leq E^D(C) + \sum_{i \neq k \neq s} \min \{ |P_i|, |P_k| \} \cdot \| c_i - c_k \|_1 - \frac{E^{P_s}((c_s))}{|P_s|},$$

and so, if $\min \{ |P_i|, |P_k| \} \cdot \| c_i - c_k \|_1 \leq \frac{E^{P_s}((c_s))}{|P_s|}$, then $E(E^D(C')) \leq E^D(C)$.

\(\square\)

\(^4\)Given any real numbers $a_1, \ldots, a_n$, then $\sum_{i=1}^n a_i^2 \geq \frac{1}{n} \cdot (\sum_{i=1}^n a_i)^2$. 

In Theorem 2 we provide a $K$-medoids error descent condition after VoronoiCorrection, which can be verified using only information obtained after the previous FastVoronoi run. This result shows that the larger the average error is in the cluster to be split and the smaller the closest distance between the different pairs of centroids is, then the more likely it is for VoronoiCorrection to generate a set of medoids with an error smaller than that of the precedent approximation. As can be seen in Table VII we observe that in practice such a descent is fairly likely to occur, especially when the number of clusters is large.

**B. Experiments.**

In the experimental section, we analyzed the effect of the number of clusters over the performance of Manhattan $K$-medoids (MK_rand, MK_D1 and MK_MM), FastVoronoi (FV_rand, FV_D1 and FV_MM), PAM algorithm (PAM_rand, PAM_D1 and PAM_MM), CLARA algorithm (CLARA_rand, CLARA_D1 and CLARA_MM) and Park’s heuristic (Park, Park_rand, Park_D1 and Park_MM) on a wide variety of data sets. In this section, we show that MK has a competitive performance regardless of the size of the data sets and number of clusters. In Figs.5-13, each point represents the average relative computational running time and error obtained at each iteration of all the considered methods.
Figure 5: Average error and running time of each method for Avila \((n = 10,436, d = 11)\).

Figure 6: Average error and running time of each method for USPS LeCun \((n = 7,290, d = 513)\).

Figure 7: Average error and running time of each method for Anuran Calls \((n = 7,193, d = 22)\).

Figure 8: Average error and running time of each method for Cambridge \((n = 5,635, d = 57)\).
Figure 9: Average error and running time of each method for Waveform (n = 4.999, d = 41).

Figure 10: Average error and running time of each method for Turkiye (n = 3.976, d = 33).

Figure 11: Average error and running time of each method for Elbow (n = 2.396, d = 28).

Figure 12: Average error and running time of each method for Digits (n = 1.797, d = 64).
As can be seen in Figs. 5-13, MK_rand, MK_D1 and MK_MM have a similar performance in general. All these variants of MK systematically converged at a very fast rate to approximations of similar or better quality than those generated by PAM, CLARA, FastVoronoi and Park. We would also like to highlight that, as for MK, the total number of re-starts may vary at different executions, the curves associated to the relative average error of MK_rand, MK_D1 and MK_MM, in Figs. 5-13 are not necessarily monotonically descendant, as it is for each run of the experiment.

C. Analyzing the overall performance for all the data sets

In the experimental section, we just provided the average performance of each method in terms of quality of the obtained solution and computational time in Tables II-III. In Figs. 14-15, we show the boxplots of both the relative errors and running times for the different numbers of clusters and groups of numbers of instances:

Figs. 14-15 show MK to converge systematically to the most competitive approximations of the K-medoids problem regardless of the seeding technique used to initialize it. In these figures we can also observe that the variability of such results is fairly small. Such an improvement in terms of the quality of the approximation is more evident as we increase the number of clusters as, in this case, it is easier for the VoronoiCorrection step to exit the current basin of attraction towards a more competitive one, see Tables IV-V. Furthermore, as can be observed in Fig. 16, as the number of clusters is larger it is also quite likely for the set of centroids generated via VoronoiCorrection to reduce the K-medoids error of the precedent approximation without requiring any run of FastVoronoi.

As for the running times, we can observe that FV and MK...
are the approaches with the fastest convergence, along with CLARA in the case of largest group of instances, running, at least, one order of magnitude faster than PAM and Park’s approach. It must also be specified that CLARA, regardless of the clustering setting, usually converges to the approximations with the largest $K$-medoids errors. On the other hand, in terms of the number of clusters, $M_K$ still behaves competitively in terms of computational load, regardless of the fact that, for $K = 50$, VoronoiCorrection is applied on average over fourteen times, see Fig.17. However, since the proposed

<table>
<thead>
<tr>
<th>Method</th>
<th>Error</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Park</td>
<td>1e−04</td>
<td></td>
</tr>
<tr>
<td>Park_rand</td>
<td>1e−03</td>
<td></td>
</tr>
<tr>
<td>PAM</td>
<td>1e−02</td>
<td></td>
</tr>
<tr>
<td>CLARA</td>
<td>1e−01</td>
<td></td>
</tr>
<tr>
<td>FV</td>
<td>1e−00</td>
<td></td>
</tr>
</tbody>
</table>

Figure 15: Error and computational time comparison for the different numbers of instances (boxplot).

Figure 16: Error descend after VoronoiCorrection for the different numbers of clusters and instances (boxplot).

Figure 17: VoronoiCorrection runs for the different numbers of clusters (boxplot).

Figure 18: Relative number of FastVoronoi iterations after its first run with respect to the first one, for the different numbers of clusters (boxplot).

split-merge mechanism modifies, at most, three of the current clusters, then the upcoming FastVoronoi tends to converge quite rapidly: after the first execution, for $K = 50$, it required over three times less the number of iterations of the first run of FastVoronoi, see Fig.18.