Using Pivots to Speed-Up k-Medoids Clustering

Adriano Arantes Paterlini[1], Mario A. Nascimento[2], Caetano Traina Junior[1]

[1] Department of Computer Science – ICMC – University of São Paulo, Brazil
{paterlini, caetano}@icmc.usp.br
[2] Department of Computing Science – University of Alberta, Edmonton, AB, Canada
mn@cs.ualberta.ca

Abstract. Clustering is a key technique within the KDD process, with k-means, and the more general k-medoids, being well-known incremental partition-based clustering algorithms. A fundamental issue within this class of algorithms is to find an initial set of medians (or medoids) that improves the efficiency of the algorithms (e.g., accelerating its convergence to a solution), at the same time that it improves its effectiveness (e.g., finding more meaningful clusters). Thus, in this article we aim at providing a technique that, given a set of elements, quickly finds a very small number of elements as medoid candidates for this set, allowing to improve both the efficiency and effectiveness of existing clustering algorithms. We target the class of k-medoids algorithms in general, and propose a technique that selects a well-positioned subset of central elements to serve as the initial set of medoids for the clustering process. Our technique leads to a substantially smaller amount of distance calculations, thus improving the algorithm's efficiency when compared to existing methods, without sacrificing effectiveness. A salient feature of our proposed technique is that it is not a new k-medoid clustering algorithm per se, rather, it can be used in conjunction with *any* existing clustering algorithm that is based on the k-medoid paradigm. Experimental results, using both synthetic and real datasets, confirm the efficiency, effectiveness and scalability of the proposed technique.

Categories and Subject Descriptors: I.5.3 [**PATTERN RECOGNITION**]: Clustering

Keywords: Clustering, k-medoids, k-means, Metric space, Pivot-based technique.

1. INTRODUCTION

Clustering is one of the most useful techniques in KDD (Knowledge Discovery in Databases) processes. In a nutshell, it is the process of dividing the data into groups of similar elements according to a similarity measure, so that each group, or cluster, is composed of elements that are similar to each other and dissimilar to elements of other groups [Han 2005].

In the last decades, several clustering algorithms have been developed for a large spectrum of applications. They can be divided into two main groups of algorithms: partitioning and hierarchical. Hierarchical algorithms produce a cluster hierarchy consisting of several levels of nested partitions. Examples of hierarchical algorithms are Single-Link [Sibson 1973], CURE [Guha et al. 1998] and BIRCH [Zhang et al. 1996]. Partitioning algorithms aim at finding the best k partitions in a single level. Examples of partitioning algorithms include k-means [Hartigan and Wong 1979] and k-medoids [Kaufman and Rousseeuw 2005]. A thorough discussion of these and other clustering algorithms can be found in [Jain et al. 1999].

The k-means algorithm is the most popular clustering algorithm, due to its simplicity and efficiency. However, whenever a distance can be evaluated among any pair of elements from the dataset, the k-medoid variation can be employed. Algorithms based on the k-medoid variation have been shown to be more robust, since they are less sensitive to outliers, do not present limitations on attribute types (k-means are restricted to multi-dimensional datasets with continuous-valued attributes), and

This work has been supported by FAPESP, NSERC, GSEP, CNPq, CAPES and STIC-AmSud.

Table I. Summary of Symbols and Definitions.

Symbols	Definitions
S	set of elements where to find the center or set of elements to be clustered
n	cardinality of the dataset
D	dimensionality of the dataset
k	expected number of clusters
$d(.,.)$	distance function between two elements
s_i	an element in S
s'_g	first focus of axis g
s''_g	second focus of axis g
f	number of axes employed to find a medoid
x_i	distance from s'_1 to the projection of element s_i over axis $\overline{s'_1 s''_1}$
X	set of $x_i \forall s_i \in S$
s_{md}	element that has the median distance in X
r_i	a medoid of S
R	subset of elements in S selected as medoids

the found clusters do not depend on the order on which the dataset is considered. Moreover, they are invariant to translations and orthogonal transformations of elements [Kaufman and Rousseeuw 2005].

The main drawback of the k-medoid algorithms is that they are computationally expensive and therefore cannot be efficiently applied to large datasets. Indeed, finding the optimal k-medoid clustering is an NP-hard problem [Mouratidis et al. 2005], and existing algorithms aim at finding approximate answers, i.e., they locate medoids that achieve a good, but not necessarily the best answer. As a consequence, several approaches have been developed aiming at reducing the computational effort needed to execute these algorithms. The three arguably best-known k-medoid based algorithms are PAM [Kaufman and Rousseeuw 1986], CLARA [Kaufman and Rousseeuw 1987] and CLARANS [Ng and Han 1994]. They are detailed in the the next section.

Sample based methods such as CLARA depend on more parameters than just the number of clusters. Deciding the sample size requires solving a compromise: small samples reduce the algorithm effectiveness whereas bigger ones increase the processing time. A fast algorithm is proposed by Park and Jun [Park and Jun 2009], however it is based on the idea of previously calculating the distance between every element pair, i.e., the required memory space is $O(n^2)$. Another algorithm called CLATIN is presented by Zhang and Couloigner [Zhang and Couloigner 2005], but it is only suitable to process small clusters.

In this article, we also offer an approximate solution. It is not a new k-medoid algorithm, but rather a novel technique that can be embedded into *any* existing algorithm to improve its efficiency when selecting would-be medoids for a given subset of elements. Our main idea is to use a pivot-based technique to substantially reduce the number of distance calculations and, hence, the computing time to find the medoids. Unlike methods that use sampling, our technique takes into account all elements, thus increasing the chances of making a very good (if not optimal) choice for the medoid element.

The remainder of this article is organized as follows. Section 2 briefly describes existing k-medoid based algorithms. Section 3 presents how our approach scales up k-medoid based algorithms allowing efficient clustering even for large databases. An experimental evaluation of our approach is shown in Section 4. Finally, Section 5 concludes the article.

2. BACKGROUND

This section presents some well-known existing k-medoid based algorithms. The symbols and definitions used throughout this article are presented in Table I.

The objective of a (clustering) k-medoid-based algorithm is to find a non-overlapping set of clusters, such that each cluster is represented by a medoid, which is the most central element located in the cluster with respect to a distance measure. The most central dataset element minimizes the summation of its distances to every other element. Thus, given a set S of n elements, an integer $k < n$ as the number of desired clusters, and a distance function $d : S \times S \rightarrow \mathbb{R}^+$ that evaluates the distance between any two elements of S, the medoid set $R \subset S$, where $|R| = k$ is the set of elements that minimizes the function

$$f(R) = \sum_{i=1}^{k} \sum_{j=1}^{|S_i|} d(r_i, s_{ij})^2 \quad (1)$$

where $r_i \in R$ is a medoid, $s_{ij} \in S_i$ is an element of S whose nearest medoid is r_i, $S_i \cap S_{i'} = \emptyset, \forall i \neq i'$ and $\bigcup_{i=1}^{k} S_i = S$. Equation 1 is called the objective function of the clustering process.

PAM is the simplest k-medoid algorithm and forms the base for all the others. It is based on an iterative optimization process that evaluates the effect of swapping a medoid with a non-medoid element and relocating the remaining elements among the respective clusters. It is executed in three main steps:

Building Phase: Randomly select an initial set $R = \{s_1, s_2, ..., s_k\} \subset S$ of k medoid elements and evaluate the objective function $f(R)$. Build a set of, initially null, k clusters S_i, and assign each initial medoid s_i to a cluster S_i.

Swapping Phase: For each element $s_j \in (S - R)$, find its closest medoid s_i and assign s_j to S_i.

Selection Phase: From each subset S_i, select a new medoid for that subset, exchanging R for the new medoids. Evaluate the objective function $f(R)$ again and, if it is smaller than the previous evaluation, repeat from step 2. Otherwise, the current subsets S_i and R are the clusters and their corresponding medoids respectively, and the algorithm finishes.

We can see that the computational complexity of step 1 is $O(1)$ and step 2 is $O(k(n-k))$. However, the most expensive part of the PAM algorithm is step 3. It requires evaluating the distances between every pair in each subset S_i. Thus, assuming in the best case that each cluster has approximately the same amount of elements, its computational cost is $O(k(n/k)^2) = O(n^2/k)$ at each iteration [Ng and Han 1994]. The PAM algorithm results in high quality clusters, as it evaluates every possible combination, working well for small datasets. However, due to its computational complexity, it is not practical for large datasets.

The computational complexity of the PAM algorithm motivated the development of CLARA, a k-medoid algorithm based on sampling. CLARA draws multiple samples of the dataset and applies PAM on each one. Then, it selects the clusters resulting from the execution that presented the lowest objective function value and assigns each element of the entire dataset to the corresponding medoids. (Experiments described in [Kaufman and Rousseeuw 2005] indicate that five samples of size $40 + 2k$ give satisfactory results.) The computational complexity of each iteration of CLARA to process each sample is $O(p^2/k + k(p-k))$, where $p > k$ is the size of the sample, thus it is much faster than PAM. However, as the medoids are picked from a sample, they can result in clusters of poor quality.

CLARANS was developed to improve CLARA. It uses a randomized search strategy in order to improve on both PAM and CLARA algorithms in terms of efficiency (computation complexity/time) and effectiveness (average distortion over the distances) respectively. The first element selected is the one having the minimum sum of dissimilarities (distances) to every other element (the objective function). Thus, the first element selected is the dataset medoid. The other $(k-1)$ medoids are subsequently selected, one at a time, considering the elements that most decrease the objective function. When searching for new good medoids, CLARANS randomly chooses elements from the remaining $(n-k)$ elements, searching for the medoids of each group as its group center. The number of elements tried

in this step is restricted by a user-provided parameter ($maxNeighbor$). If no better solution is found after $maxNeighbor$ attempts, a local optimal is assumed to be reached. The procedure continues until $numLocal$ local optimals have been found. It is recommended to set parameters $numLocal$ and $maxNeighbor$ to 2 and $\max\{250, 0.0125 \times k(n-k)\}$, respectively [Ng and Han 1994]. However, the computational complexity of CLARANS is still $O(n^2)$.

Three other techniques were proposed in [Ester et al. 1995] employing R*-trees [Beckmann et al. 1990], in order to make CLARANS more efficient for large spatial databases. However, these techniques cannot be extended to non-dimensional databases. Some improvements to CLARANS are also proposed in [Chu et al. 2002], using pre-processing and approximations.

PAM, CLARA and CLARANS has been the base clustering algorithms for several data domains. They are much costlier than their k-means based counterpart, so special variations have been proposed to handle datasets with specific properties, but no one has been published able to improve generic datasets as our technique does. In fact, our proposed technique is generic and can help improving almost any k-medoid based algorithm, including their more recent variations.

3. THE PROPOSED TECHNIQUE

Typical k-medoid algorithms are based on the notion of finding *optimal* medoids in the datasets provided. Unfortunately, finding a *single* optimal medoid requires $O(n^2)$ distance calculations, making the optimal problem excessively costly for any non-trivial dataset. As we saw in the previous section, this cost is due to the large amount of distance calculations required to find each medoid of the evolving clusters. Existing algorithms relax the notion of optimality and instead seek for the medoid among a small subset randomly selected from the dataset. This approach can indeed speed up the algorithms if small samples are employed, but may also severely hurts the quality of the result as the sample shrinks.

Our approach is to find a good element to be the medoid based on geometric properties that estimate the region of the dataset where the medoid probably is, using the whole dataset in the process. It is not an exact solution, but, as we shall see, the error is usually much smaller than those of the sample-based solutions. In addition, the time complexity is log-linear rather than quadratic on the dataset cardinality. In fact, our technique is linear on the dataset cardinality regarding the number of distance calculations required, and its worst-case scenario is log-linear only because it requires sorting an array of real values of size at most n. We name our technique FAMES (for FAst MEdoid Selection). Notice that we do not claim that FAMES is a new clustering algorithm by itself, instead it is a novel way to find an adequate medoid set, and more importantly it can be used within any clustering algorithm that requires such step.

The intuition of our method is that if we draw an imaginary line connecting two far apart elements s'_1 and s''_1 in the dataset border, the medoid will be found close to that line. More than that, the medoid is close to the point where the line intersects the hyperplane that divides the dataset space in half, so that half of the elements are in each hyperplane side – see Figure 1. Thus, instead of searching for the medoid among a sample of randomly selected elements, FAMES evaluates a small amount of elements that are probably close to that point.

Algorithm 1 describes the FAMES steps required to select a medoid following this intuition. First, we need to find two far apart elements s'_1, s''_1 in the dataset border. This is obtained randomly picking an element $s_z \in S$ (step 1) and finding the element s'_1 that is the farthest from s_z (step 2). Then, s''_1 is selected as the farthest from s'_1 (step 3). Conceptually, for high-dimensional or complex metric datasets, step 3 could be repeated a number of times after reassigning s'_1 as the newly found element s''_1, so really a far apart pair is obtained. However, the experiments we performed over several assorted datasets showed no significant increase in the resulting quality when additional steps are performed. The elements s'_1 and s''_1 are called the pivots of the next steps.

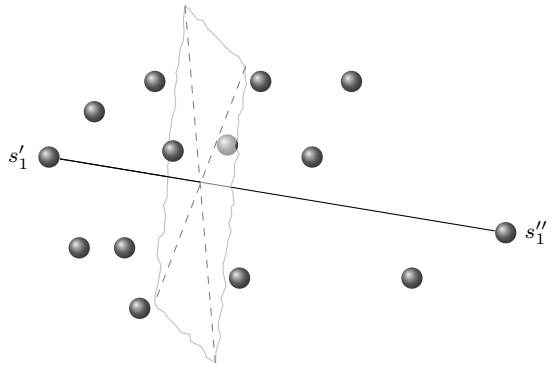


Fig. 1. The intuition of our method is that the medoid shall be close to the point where the line connecting two far apart elements s'_1, s''_1 intersects the hyperplane that divides the dataset space in half.

Algorithm 1 FAMES - Fast Medoid Selection

Require: set of elements S
Ensure: a medoid r

1: Randomly choose an element $s_z \in S$.
2: Find the farthest element s'_1 from s_z.
3: Find the farthest element s''_1 from s'_1.
4: Find the median element s_{md} as the one that splits the elements such that half are nearer to s'_1 e half are nearer to s''_1.
5: Let p_m be the projection of element s_{md} over line $\overline{s'_1 s''_1}$ and $m = x_{md}$ be its distance to s'_1.
6: Select the medoid r as the element $s_i \in S$ that minimize $|d(s'_1, s_i) - m| + |d(s''_1, s_i) - (d(s'_1, s''_1) - m)|$.

Next, we divide the dataset in half, regarding their distances to the pivots, by projecting each element $s_i \in S$ over the line $\overline{s'_1 s''_1}$, using the cosine law and only distances among known elements:

$$x_i = \frac{d(s'_1, s_i)^2 + d(s'_1, s''_1)^2 - d(s''_1, s_i)^2}{2 * d(s'_1, s''_1)} \tag{2}$$

where x_i is the distance of the element s_i projected over the line to s'_1, as illustrated in Figure 2. In this way, we can sort the array X of the projections x_i of all elements $s_i \in S$ and pick up the element s_{md} as the one that has the median value x_{md} in X. This corresponds to step 4 in Algorithm 1.

Let us call the point at distance x_{md} from s'_1 over line $\overline{s'_1 s''_1}$ as p_m. Two things must be noticed about this point. First, p_m does not need to correspond to an element in S, so in a metric space it is generally not possible to evaluate its distance to any element in S other than s'_1 and s''_1. Thus, lets call m the distance from p_m to s'_1, what corresponds to step 5 in Algorithm 1. Second, the distances x_i are exact only if the metric space can be embedded in an Euclidean space – it is a well known fact that for metric spaces that cannot be isometrically embedded in a Euclidean space, Equation 2 can produce an error bounded by the distortion introduced by the embedding process [Hjaltason and Samet 2003]. Thus, although most of the datasets employed in real-world applications can be isometrically embedded in a Euclidean space, and, even though those that cannot present only a small error, the results in X must be treated just as an approximation. However, as our intent is to select a subset of elements expectedly containing a good center, in most cases the error does not kick out the correct element.

Finally, we need to select the medoid as the element r that is the closest to p_m. As we cannot evaluate the real distance from p_m to each element s_i, and considering that every value in X is also an approximation, we choose as r the element that minimizes the sum of differences from s_i and p_m

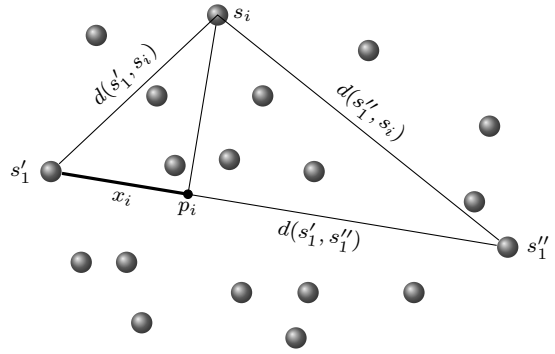


Fig. 2. Using the cosine law to evaluate the projection of element s_i over the line $\overline{s'_1 s''_1}$.

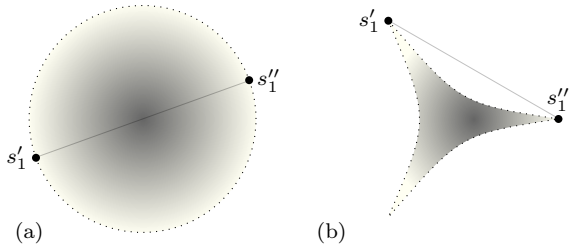


Fig. 3. Distinct distributions of densities over the space. (a) Normal; (b) Spiculated.

to each pivot, evaluated as:

$$Min_{i=1}^{n}(|d(s'_1, s_i) - m| + |d(s''_1, s_i) - (d(s'_1, s''_1) - m)|) \tag{3}$$

This corresponds to step 6 in Algorithm 1.

Regarding the computational complexity of Algorithm 1, steps 1-3 requires two scans over the dataset, comparing each element to a pivot, so their complexity is $O(n)$. Step 4 requires a single scan over the dataset to evaluate X, so this step also has a linear computacional complexity using an appropriate linear algorithm to select the median element from the vector X. Step 5 is independent from the dataset cardinality and step 6 also requires a single scan over the dataset, so their complexity is also $O(n)$. Thus, the overall computacional complexity is $O(n)$.

An important factor for similarity-based algorithms is the number of distance calculations required. At a first look, every step but the fifth requires distance calculations. However, steps 2, 3, 4 and 6 only require calculating distances from the elements in S to s'_1 or s''_1, which are calculated respectively at steps 2 and 3. Thus, if those distances are stored in steps 2 and 3, the remaining steps do not require any new distance calculations. In this way, FAMES requires memory only to store three real values per element: distance to s'_1, distance to s''_1, and x_i, that is, its memory requirement is linear on the dataset cardinality.

3.1 A Multi-axes Variation

The FAMES algorithm works fine to find centers in datasets whose data follow a statistical distribution, such as a Gaussian or even a uniform distribution. Those datasets usually are denser as closer to the center, and the distances from the center to the border are similar regardless of the direction, as shown in Figure 3(a). However, there exist datasets that do not follow that distribution but, for instance, can be rather spiculated as in the case of high dimensional datasets where few dimensions assume higher values at a time, as shown in Figure 3(b).

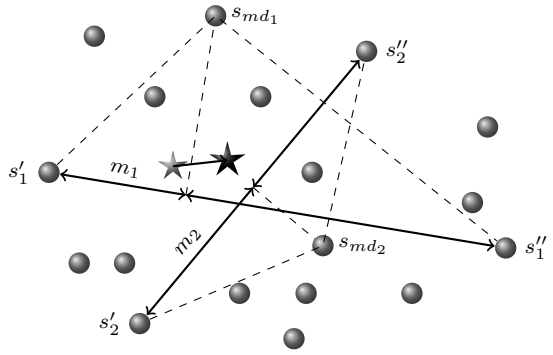


Fig. 4. A set of elements that shows the intuition behind the M-FAMES algorithm representing elements, distances and two axis $\overline{s'_1 s''_1}$ and $\overline{s'_2 s''_2}$.

For complex datasets where only one axis may not be enough to correctly point to the proper region where to search for the medoid, we propose an extension to FAMES that uses multiple axis. This extension is able to improve the medoid selection at the expense of a few extra distance computations. Figure 4 presents the intuition to add a second axis $\overline{s'_2 s''_2}$. The idea is first to find a pair of elements s'_1 and s''_1 far apart, as performed in steps 1-3 of Algorithm 1, then proceed to find another pair s'_2 and s''_2, in such a way that both s'_2 and s''_2 are far apart discounting how far they are in the direction given by the previous axis $\overline{s'_1 s''_1}$. Notice that the second axis displaces the target region to a new region to search for medoid, pulling the medoid from the gray star to the black star, i.e., in the direction of a larger amount of elements. Further axes can be added, always discounting every previous axes. To evaluate the amount to discount, the distance between s'_1 and s''_1 is taken as an approximation of the remaining maximum distance on the dataset, and we call the distances between a pair of pivots as a *chord* (as just the *chord* for the first pivot pair is probably close to the dataset diameter). The subsequent pair of pivots for the next axes are located as far as possible from the previous pivots, taking into account the *chord* value from all the previous axes simultaneously, as formulated by Equations 4 and 5. Thus, taking h as the current pivot pair being evaluated, the first pivot for the current axis is the element in S that minimizes:

$$Min_{i=1}^{n} \sum_{g=1}^{h-1} \{|chord - d(s'_g, s_i)| + |chord - d(s''_g, s_i)|\} \tag{4}$$

The second pivot for the current axis also considers the first element evaluated in Equation 4, so it is the element in S that minimizes:

$$Min_{i=1}^{n} \sum_{g=1}^{h-1} \{|chord - d(s'_g, s_i)| + |chord - d(s''_g, s_i)|\} + |chord - d(s'_h, s_i)| \tag{5}$$

Roughly speaking, the idea of M-FAMES is to create a number of axes in a metric space (or a number of axes smaller than the dimensionality of a multidimensional space) and search for the medoid in the region around the medians of all axis. The main steps to take multiple axes are presented in Algorithm 2, which is called M-FAMES. Besides the set of elements whose medoid must be found, M-FAMES requires also a user-defined number f of axes as input. The first axis is evaluated by Steps 2-5 in Algorithm 2 just as Algorithm 1. Further axes are evaluated by steps 6-11, using Equations 4 and 5 on Steps 7 and 8 to choose new pivots, that are elements far apart each other and far apart previous pivots. Steps 9 e 10 correspond to Steps 4 and 5 in Algorithm 1 for the first axis, which

Algorithm 2 M-FAMES - Multiple Fast Medoid Selection

Require: set of elements S, number of axes f
Ensure: a medoid r

1: Randomly choose an element $s_z \in S$.
2: Find the farthest element s'_1 from s_z.
3: Find the farthest element s''_1 from s'_1.
4: Find the median s_{md} as the element that splits the dataset such that half of the elements are nearer to s'_1 and half are nearer to s''_1.
5: Let p_m be the projection of element s_{md} over line $\overline{s'_1 s''_1}$ and $m_1 = x_{md}$ be its distance to s'_1.
6: **for** g from 2 to f **do**
7: Select elements s'_g using Equation 4
8: Select elements s''_g using Equation 5
9: Find the median s_{md} as the element that splits the dataset such that half of the elements are nearer to s'_g and half are nearer to s''_g.
10: Let p_m be the projection of element s_{md} over line $\overline{s'_g s''_g}$ and $m_g = x_{md}$ be its distance to s'_g.
11: **end for**
12: Select the medoid r as the element $s_i \in S$ such that s_i minimize $\sum_{g=1}^{f}(|d(s'_g, s_i) - m_g| + |d(s''_g, s_i) - (d(s'_g, s''_g) - m_g)|)$.

evaluates the median element projection. Finally, using a single minimization function step 12 selects the medoid element r as the element from S that is simultaneously near the imaginary point located at the median element projection of each axis.

Regarding the computational complexity of the multi-axis variation presented in Algorithm 2, this algorithm basically follows similar steps as Algorithm 1. The difference corresponds essentially to the need to repeat steps 1 to 5 of Algorithm 1 for each axis in Algorithm 2. The number of repetitions is defined by a user parameter, and is independent of the dataset cardinality and dimensionality. Thus, steps 1 to 5 of Algorithm 2 work out the first axis, exactly as in Algorithm 1, whereas steps 7 to 10 work out the remaining axes, pursuing however the same intuition. The computational complexity of Algorithm 2 is the computational complexity of Algorithm 1 multiplied by the required number of axes.

Notice that the number of distance calculations is also twice the cardinality of the dataset multiplied by the required number of axes. Notice that the same seed s_z set at step 1 can be reused in steps 7 and 8 at each iteration beginning at step 6, thus reducing the requirement of computing extra distances. As well, the memory employed to store the intermediary values of the distances x_i can be reused, such as the distances from each s_i to the current pivots being evaluated, so no new memory is required (notice in steps 9 and 10 that variables x_{md}, s_{md} and p_m are not indexed by g, as only the distances m_g to the current pivots must be reserved). Thus the memory requirement for both FAMES and M-FAMES are almost the same.

3.2 A FAMES-based k-medoid Algorithm

The FAMES Algorithm can be easily embedded within algorithms that depend on selecting the medoid from a set of elements. In fact, using the traditional PAM k-medoid-based algorithm, Algorithm 3 can be trivially derived using either FAMES or M-FAMES. As we show in Section 4, this rather simple algorithm is more efficient and no less effective that any of the traditional algorithms reviewed in Section 2.

Algorithm 3 A FAMES-based k-Medoid

Require: set of elements S, number of clusters k
1: Select k elements as the initial medoids set S_m.
2: Assign each element to the nearest medoid.
3: **repeat**
4: Find a new medoid for each cluster, using either FAMES or the M-FAMES algorithm.
5: Reassign each element to its nearest medoid.
6: **until** The medoid set does not change
7: Calculate the sum of distance from all elements to their medoids.

Table II. Description of datasets used in the experiments. 12 Synthetic datasets are generated varying n, k and D using the values presented and 2 real datasets are also presented with its correspondent values.

Name	n	k	D	$d()$
SyntCL($n\|k\|D$)	$5,000 - 10,000 - 20,000$	$5,000 - 10,000 - 20,000$	$5,000 - 10,000 - 20,000 - 50,000$	L_2
Pendigits	$10,992$	10	16	L_2
AloiL	$12,000$	(not defined)	(adimensional)	L_M

4. EXPERIMENTAL EVALUATION

To evaluate our proposed algorithm, we compared it with PAM, CLARA and CLARANS, measuring its effectiveness and efficiency. The average distance from all elements to it nearest medoid is employed to evaluate the effectiveness, as smaller values for the average distance indicate better clustering. The efficiency was measured by the number of distance calculations. All results presented in this article are the average for at least five executions to avoid biased values. Furthermore, to avoid infinite running all algorithms are limited to 20 iterations, as sometimes these algorithms takes too long to converge or do not converge at all. The experiments were performed in a PC with an Intel(R) Core(TM)2 Quad @ 2.83GHz CPU and 4 GB RAM.

We evaluated both FAMES and M-FAMES using several datasets. In this article we present representative results obtained from both real and synthetic datasets, to highlight how those techniques can improve the performance of the well-known PAM, CLARA and CLARANS clustering algorithms.

4.1 Description of the Datasets

Table II presents the description of the datasets along with its name, the total number of elements (n), the number of clusters (k) used to generate the dataset, the dimensionality of the dataset (D), and the distance function ($d()$).

All synthetic clustered datasets were generated with Gaussian distribution in a D-dimensional unit hypercube. We generated the datasets using the same process described in [Ciaccia et al. 1997] with variance $\sigma^2 = 0.1$ and randomly distributed clusters centers. The name of the synthetic datasets are composed by the number of elements, the number of clusters and the dimensionality, for example a synthetic dataset with $5,000$ elements, generated with 10 clusters and dimensionality 50 is denoted as SyntCL($n = 5,000|k = 10|D = 50$).

The Pendigits dataset is avaliable at UCI Machine Learning Repository[1] and is composed of Pen-Based Recognition of Handwritten Digits. The AloiL dataset [Geusebroek et al. 2005] is composed of real adimensional data and it is used aiming at analyzing the behavior of our algorithm in real-world data, using Metric Histograms extracted from each image. The images are from the Amsterdam Library of element Images (ALOI)[2]. It is a collection of color images from one thousand small elements.

[1] `http://archive.ics.uci.edu/ml/`
[2] Avaliable at `www.science.uva.nl/~aloi/`

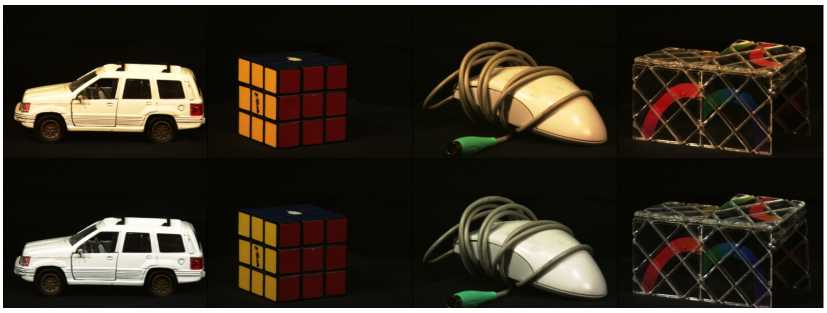


Fig. 5. Small sampre from the AloiL dataset with two different illumination color of four images.

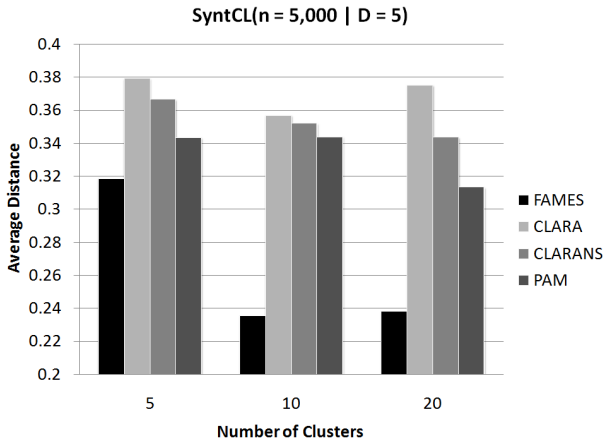


Fig. 6. Average distance from all elements to their corresponding medoid using FAMES, CLARA, CLARANS and PAM on Synthetic Datasets with 5,000 elements and 5 dimensions

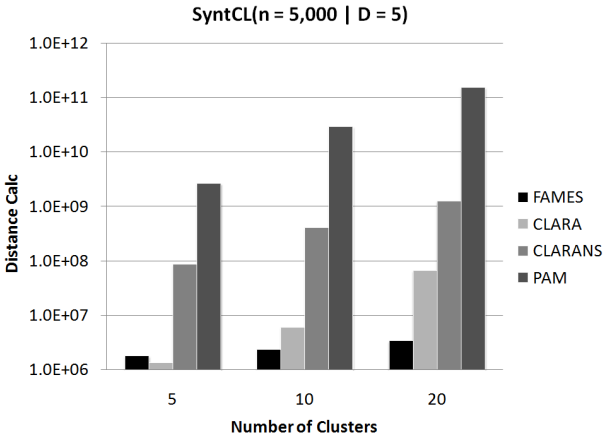


Fig. 7. Number of distance calculations performed by FAMES, PAM, CLARA and CLARANS on synthetic datasets with 5,000 elements and 5 dimensions

Figure 5 shows a small sample of those images. Each element was photographed systematically varying an image parameter. For the experiments, we selected a dataset where each element was photographed with 12 different illumination colors. More details on the adimensional histogram and the used distance

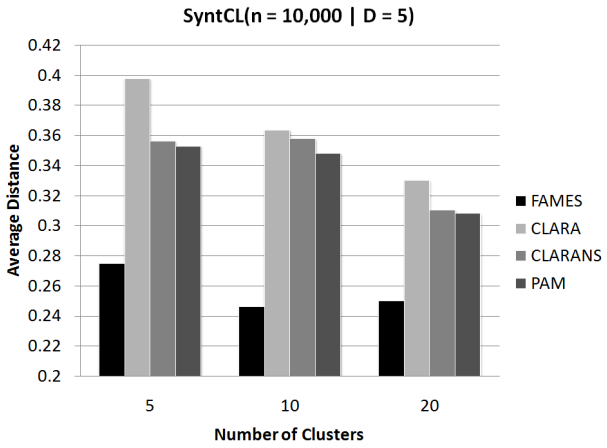


Fig. 8. Average distance from all elements to their corresponding medoid using FAMES, CLARA, CLARANS and PAM on Synthetic Datasets with 10,000 elements and 5 dimensions

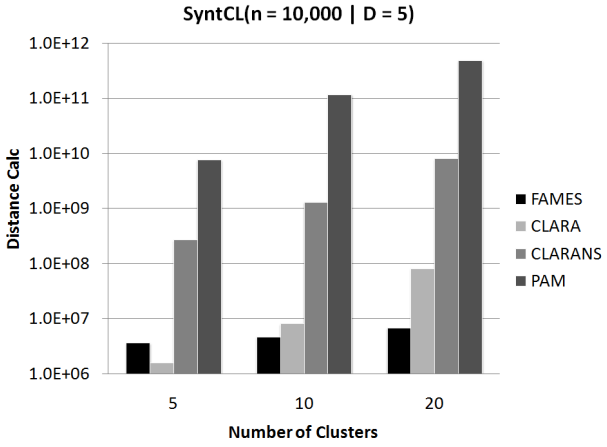


Fig. 9. Number of distance calculations performed by FAMES, PAM, CLARA and CLARANS on synthetic datasets with 10,000 elements and 5 dimensions

function can be found elsewhere [Azevedo-Marques et al. 2002].

Even though the synthetic datasets consist of D-dimensional vectors, we do not exploit the operations available in the multidimensional spaces, and treat the vectors in the dataset as elements in a metric space. The distance between two elements is returned by the Euclidean distance function.

4.2 The Effects of Increasing the Number of Clusters k

Figure 6 shows the effectiveness and Figure 7 shows the efficiency of FAMES, PAM, CLARA and CLARANS for the datasets SyntCL($n = 5,000|k = 5, 10$ and $20|D = 5$). The same number of clusters used to generate the datasets are used as input for each algorithm to process the dataset. Algorithm FAMES always reached a better effectiveness, with the improvements ranging from 7% to 36%. Regarding efficiency, CLARA slightly outperforms FAMES when $k = 5$, but FAMES reached a clearly better efficiency for all other values of k.

Figure 8 shows the effectiveness and Figure 9 shows the efficiency for the datasets with 10,000

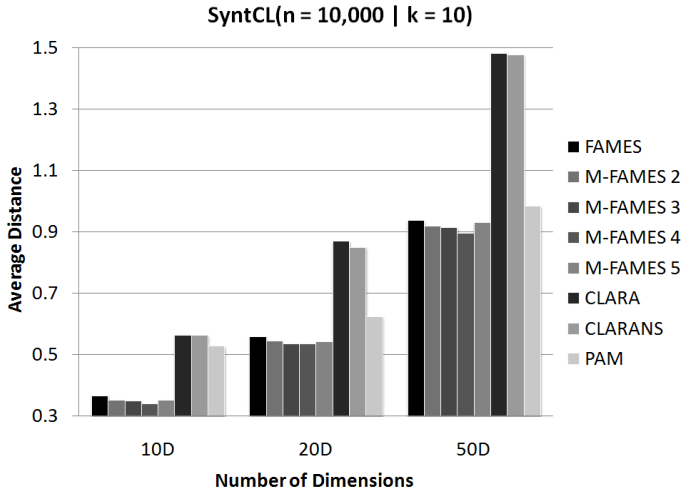


Fig. 10. Average distance from all elements to their corresponding medoid using PAM, CLARA, CLARANS and M-FAMES with projection in 1 to 5 axes on Synthetic Datasets with 10,000 elements and 10 clusters

elements, i.e., SyntCL($n = 5,000|k = 5, 10$ and $20|D = 5$). For all these datasets, FAMES achieved a better effectiveness than PAM, CLARA and CLARANS, and the improvement ranged from 18% to 32%. Again, CLARA beats FAMES when $k = 5$, but FAMES archived a much better performance for all other values of k.

The better efficiency obtained with the CLARA algorithm on datasets SyntCL($n = 5,000|k = 5|D = 5$) and SyntCL($n = 10,000|k = 5|D = 5$) is negligible if we consider that the efficiency improvement obtained provokes a higher decrease on cluster quality. This fact can be explained remembering that CLARA works with fixed sized samples ($40 + 2k$), in this case 5 samples with 50 elements each, resulting to less than 1% chance that the best medoid is inclued on the samples.

4.3 M-FAMES Variant

In order to evaluate the M-FAMES variant and the effect of increasing dataset dimensionality, we performed experiments on three datasets SyntCL($n = 10,000|k = 10|D = 10, 20$ and 50). The results are presented in Figure 10 and Figure 11, showing not only the algorithms CLARA, CLARANS and PAM, but also the M-FAMES algorithm with projections made on 1 (FAMES) to 5 axes (M-FAMES 2–5).

As expected, the dimensionality increase has little effect on the number of distance calculations performed by all the algorithms. However, the difference between M-FAMES and the others is notable. With 20 dimensions M-FAMES is 2.5 times faster than CLARA, 444 times faster than CLARANS and 18950 times faster than PAM. The cluster quality is reduced for all algorithms with the dimensionality increase, due to the effect know as "Curse of Dimensionality", which causes the increase on pair-wise distance between elements. Even so, the quality obtained with M-FAMES is better for all the cases, reaching an effectiveness improvement of more than 30% compared with the competitors considering the dataset with dimensionality 50. Figure 11 shows that increasing the number of axes used by M-FAMES also increases the number of distance calculations. For all these three datasets, using more than three axes results on M-FAMES becoming more costly than CLARA. However, the cluster quality of M-FAMES is better on all cases, compared with any of the competitors.

It is interesting to notice that the cluster quality was always the best with 4 axes M-FAMES, regardless of the dataset dimensionality. Although, this analysis is not enough to generalize to any dataset, the results allow to say that 4 axes could be recommended for the general case.

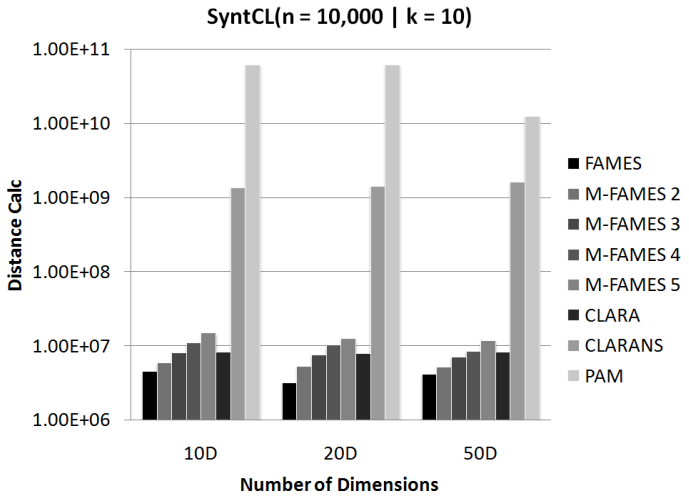


Fig. 11. Number of distance calculations to perform PAM, CLARA, CLARANS and M-FAMES with projection in 1 to 5 axes on synthetic datasets with 10,000 elements and 10 clusters

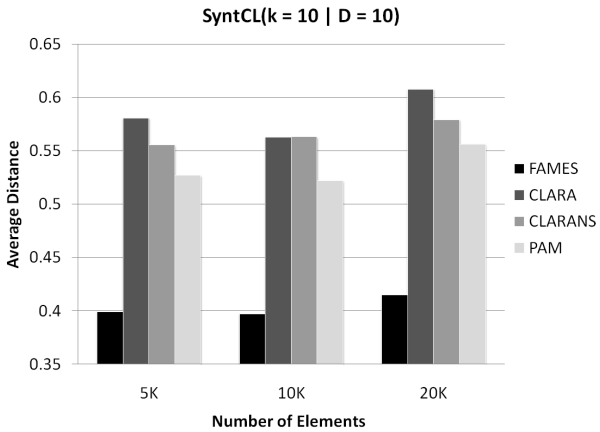
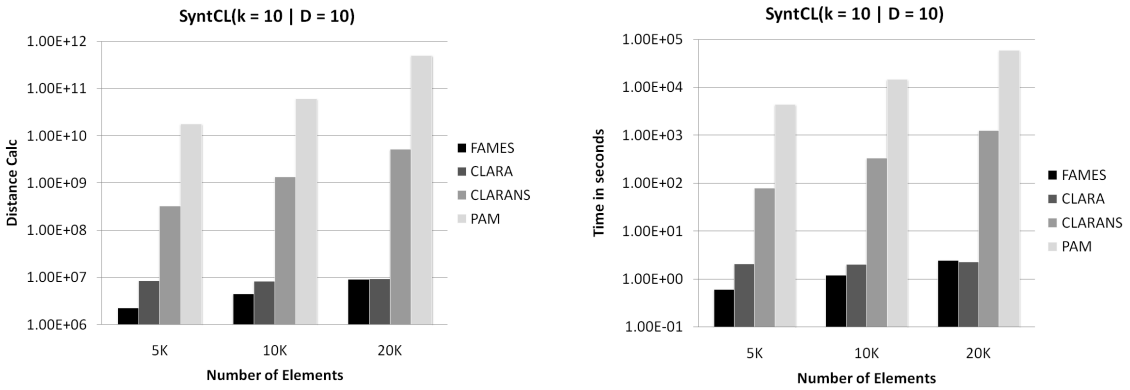


Fig. 12. Average distance from all elements to their corresponding medoid using FAMES, CLARA, CLARANS and PAM on Synthetic Datasets with 10 clusters and 10 dimensions

4.4 Scalability

In order to evaluate the effect of increasing the number of elements on the datasets, we performed experiments on three datasets SyntCL($n = 5,000$ 10,000 and $20,000|k = 10|D = 5$). In this section the time scalability is shown and compared with distance calculations. The results are presented in Figures 12 and 13.a. In order to better illustrate the practicality of our proposed approach we also show data related to actual processing time in Figure 13.b.

The average distance is clearly better for FAMES independently of the number of elements, reaching with the 10,000 elements dataset at least 24% of improvement when compared with PAM and up to 31% of improvement when compared with CLARA. The results show that the increase on the number of elements is followed by a linear increase on the number of distance performed by FAMES while distance calculations for PAM increases 28 times from 5,000 elements to 20,000 elements. Moreover, comparing Figures 13.a and 13.b its is notable that in fact the execution time has the same behavior as the distance calculations used to perform each of the algorithms.



(a) (b)

Fig. 13. (a) Number of distance calculations performed by FAMES, PAM, CLARA and CLARANS on synthetic datasets with 10 clusters and 10 dimensions. (b) Time in seconds to perform FAMES, PAM, CLARA and CLARANS on Synthetic Datasets with 10 clusters and 10 dimensions

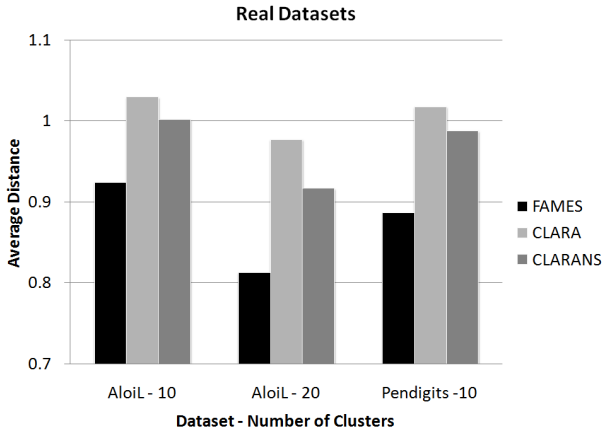


Fig. 14. Average Distance of all elements to correspondent medoid using FAMES, CLARA and CLARANS on Aloi Ilumination Datasets with 12,000 elements of Metric Histograms

4.5 Exploring Real Datasets

To evaluate FAMES over real databases and validate our algorithm, we performed a set of experiments using two real datasets: the Pendigits dataset and the the metric histograms from the AloiL Ilumination dataset. Since the real number of clusters in the AloiL Illumination dataset is not known, in these experiments, 10 and 20 clusters were set as initial parameter. The PAM algorithm was manually stopped after 2 days running on the AloiL dataset without providing a result. Thus, it was disconsidered in this section.

The results in Figure 14 show that FAMES reaches up to 16% clustering quality increase over CLARA and 11% over CLARANS. Futhermore, Figure 15 shows that FAMES is always more efficient than CLARA and CLARANS. For example, FAMES is 1735 times faster than CLARANS and 11 times faster than CLARA for the AloiL dataset with 20 clusters.

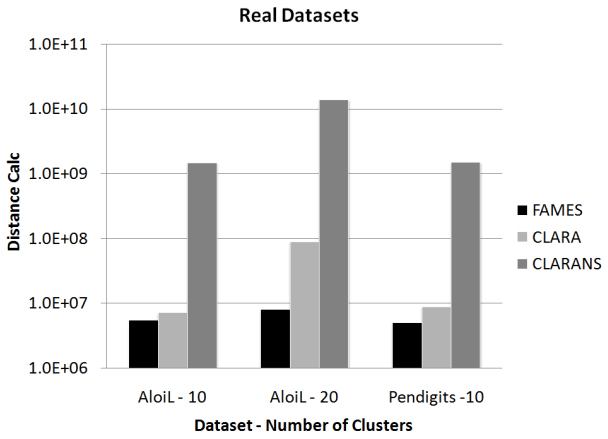


Fig. 15. Number of distance calculations to perform FAMES, CLARA and CLARANS on Aloi Ilumination Datasets with 12,000 elements of Metric Histograms

5. CONCLUSION

In this article we presented the new algorithm FAMES and it extension M-FAMES, which are based on pivoting techniques and can be used to efficiently select a medoid set from a given set of elements. Such a medoid set can be used within any medoid-based clustering algorithm that requires finding medoids, as is the case for the majority of the existing partition-based clustering algorithm, making our contribution of general usefulness. Indeed, the proposed algorithms are suitable to any problem that is commonly solved through medoid selection, helping to improve the most common clustering methods based on k-medoids, and in particular to any of the algorithms derived from PAM, CLARA or CLARANS.

Our experiments on both real and synthetic datasets show that using our techniques for medoid selection, even a fairly simple clustering algorithm can outperform classical algorithms in terms of both effectiveness and efficiency. The gain in efficiency comes from avoiding the need to evaluate all pair-wise distances in each evolving partition during the clustering calculation, and the gain in effectiveness derives from the fact that all elements are considered as potential medoid candidates, i.e., they are not restricted to a sample of the dataset elements as in several previously proposed techniques.

The experiments show that our technique is able to improve the effectiveness of the clustering process in at least 7%, achieving in some cases up to 38%. This improvement is due to the fact that all elements are considered in the search for the medoids, and not just a sample of them, as the sample-based algorithm does. By the other side, it is much faster than the non-sample based algorithms, being on average more than 10 thousand times faster, furthermore the results also reached better effectiveness in most cases. The experiments showed also that the FAMES algorithm is scalable for increasing dataset cardinality, and that the multi-axes M-FAMES variant is scalable also regarding the dataset dimensionality.

Summarizing, we proposed a new way to quickly select the medoid of a set of elements in a domain where a distance function is defined, which is able to improve any clustering algorithm based on finding medoids in partitions of the data. Our technique helps improving both the effectiveness and the efficiency of the base clustering algorithm, and is scalable to both the dimensionality and cardinality of the dataset, as the experiments on both synthetic and real datasets highlighted.

REFERENCES

Azevedo-Marques, P. M., Traina, A. J. M., Traina, Jr., C., and Bueno, J. M. The Metric Histogram: A New and Efficient Approach for Content-based Image Retrieval. In *Proceedings of the IFIP TC2/WG2.6 Sixth Working Conference on Visual Database Systems: Visual and Multimedia Information Management*. Kluwer, B.V., Deventer, The Netherlands, pp. 297–311, 2002.

Beckmann, N., Kriegel, H.-P., Schneider, R., and Seeger, B. The R*-tree: An Efficient and Robust Access Method for Points and Rectangles. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. Atlantic City, NJ, USA, pp. 322–331, 1990.

Chu, S.-C., Roddick, J., and Pan, J. An Efficient K-Medoids-Based Algorithm Using Previous Medoid Index, Triangular Inequality Elimination Criteria, and Partial Distance Search. In *Data Warehousing and Knowledge Discovery*. Lecture Notes in Computer Science, vol. 2454. Springer, Berlin / Heidelberg, pp. 301–311, 2002.

Ciaccia, P., Patella, M., and Zezula, P. M-Tree: An Efficient Access Method for Similarity Search in Metric Spaces. In *Proceedings of International Conference on Very Large Data Bases*. Athens, Greece, pp. 426–435, 1997.

Ester, M., Kriegel, H.-P., and Xu, X. Knowledge Discovery in Large Spatial Databases: Focusing Techniques for Efficient Class Identification. In *Proceedings of the International Symposium on Advances in Spatial Databases*. London, UK, pp. 67–82, 1995.

Geusebroek, J.-M., Burghouts, G. J., and Smeulders, A. W. M. The Amsterdam Library of Object Images. *International Journal of Computer Vision* 61 (1): 103–112, 2005.

Guha, S., Rastogi, R., and Shim, K. CURE: An Efficient Clustering Algorithm for Large Databases. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. Seattle, Washington, USA, pp. 73–84, 1998.

Han, J. *Data Mining: Concepts and Techniques*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2005.

Hartigan, J. A. and Wong, M. A. Algorithm AS 136: A K-Means Clustering Algorithm. *Journal of the Royal Statistical Society. Series C (Applied Statistics)* 28 (1): 100–108, 1979.

Hjaltason, G. and Samet, H. Properties of Embedding Methods for Similarity Searching in Metric Spaces. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 25 (5): 530–549, May, 2003.

Jain, A. K., Murty, M. N., and Flynn, P. J. Data Clustering: A Review. *ACM Computing Surveys* 31 (3): 265–323, 1999.

Kaufman, L. and Rousseeuw, P. J. Clustering Large Data Sets (with discussion). In *Pattern Recognition in Practice II*. North-Holland, Amsterdam, pp. 425 – 437, 1986.

Kaufman, L. and Rousseeuw, P. J. Clustering by Means of Medoids. In Y. Dodge (Ed.), *Statistical Data Analysis based on the L1 Norm*. Elsevier, Berlin, pp. 405–416, 1987.

Kaufman, L. and Rousseeuw, P. J. *Finding Groups in Data: An Introduction to Cluster Analysis*. John Wiley and Sons, 2005.

Mouratidis, K., Papadias, D., and Papadimitriou, S. Medoid Queries in Large Spatial Databases. In *Advances in Spatial and Temporal Databases*. Lecture Notes in Computer Science, vol. 3633. Springer, Berlin / Heidelberg, pp. 55–72, 2005.

Ng, R. T. and Han, J. Efficient and Effective Clustering Methods for Spatial Data Mining. In *Proceedings of International Conference on Very Large Data Bases*. Santiago de Chile, Chile, pp. 144–155, 1994.

Park, H.-S. and Jun, C.-H. A Simple and Fast Algorithm for K-Medoids Clustering. *Expert Systems with Applications* 36 (2, Part 2): 3336 – 3341, 2009.

Sibson, R. SLINK: An Optimally Efficient Algorithm for the Single-Link Cluster Method. *The Computer Journal* 16 (1): 30–34, 1973.

Zhang, Q. and Couloigner, I. A New and Efficient K-Medoid Algorithm for Spatial Clustering. In *Computational Science and Its Applications*. Lecture Notes in Computer Science, vol. 3482. Springer, Berlin / Heidelberg, pp. 207–224, 2005.

Zhang, T., Ramakrishnan, R., and Livny, M. BIRCH: An Efficient Data Clustering Method for Very Large Databases. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. Montreal, Quebec, Canada, pp. 103–114, 1996.