

## An efficient approach to scale up *k-medoid* based algorithms in large databases

Maria Camila N. Barioni, Humberto L. Razente,  
Agma J. M. Traina, Caetano Traina Jr.\*

<sup>1</sup>Computer Sciences Department – ICMC/USP  
Caixa Postal 668 – 13560-970 – São Carlos – SP – Brazil

{mcamila, hlr, agma, caetano}@icmc.usp.br

**Abstract.** Scalable data mining algorithms have become crucial to efficiently support KDD processes on large databases. In this paper, we address the task of scaling up *k-medoids* based algorithms through the utilization of metric access methods, allowing clustering algorithms to be executed by database management systems in a fraction of the time usually required by the traditional approaches. Experimental results based on several datasets, including synthetic and real data, show that the proposed algorithm may reduce the number of distance calculations by a factor of more than a thousand times when compared to existing algorithms while producing clusters of comparable quality.

### 1. Introduction

Clustering is one of the key techniques in the KDD (*Knowledge Discovery in Databases*) process. It is usually applied aiming at uncovering hidden structures underlying a collection of objects. Briefly, clustering is the process of dividing the data into groups of similar objects according to a similarity measure. The goal is that each group, or cluster, be composed of objects that are similar to each other and dissimilar to objects of other groups [Han and Kamber 2001].

In the last decades, several clustering algorithms have been developed for a large spectrum of applications. They can be divided into two main groups: partitioning and hierarchical clustering algorithms. Hierarchical algorithms produce a cluster hierarchy consisting of several levels of nested partitions of the dataset. On the other hand, partitioning algorithms try to find the best  $k$  partitions of a dataset, thus creating a single level partition that divides the data into  $k$  clusters. Examples of hierarchical algorithms are *Single-Link* methods [Sibson 1973], *CURE (Clustering Using Representatives)* [Guha et al. 1998] and *BIRCH (Balanced Iterative Reducing and Clustering using Hierarchies)* [Zhang et al. 1996]. Partitioning algorithms include *k-means* [Hartigan and Wong 1979] and *k-medoids* [Kaufman and Rousseeuw 2005]. A description of these and other clustering algorithms can be found at [Jain et al. 1999].

The *k-means* algorithm is the most popular among the algorithms mentioned above due to its simplicity and efficiency. However, the *k-medoids* based algorithms have been shown to be more robust since they are less sensitive to the existence of outliers, do not present limitations on attribute types (*k-means* are restricted to multi-dimensional

---

\*This work has been supported by FAPESP, CNPq and CAPES.

continuous datasets), and also, because the clustering found does not depend on the input order of the dataset. Moreover, they are invariant to translations and orthogonal transformations of objects [Kaufman and Rousseeuw 2005].

The drawback of the *k-medoids* based algorithms is that they are very time consuming and therefore, they cannot be efficiently applied to large datasets. This has motivated the development of several approaches aiming at reducing the computational effort needed to execute these algorithms [Ester et al. 1995, Chu et al. 2002, Zhang and Couloigner 2005]. A common strategy is to extract a sample, or to apply a selection of representative objects before applying the whole or part of the clustering algorithm to the resulting subset of objects. The quality of the resulting clusters is usually dependent on the selection of a relevant subset of objects.

In this paper, we investigate how to improve the efficiency of *k-medoids* based algorithms using a *Metric Access Method (MAM)* in order to select the representative objects over which these algorithms will be applied. Particularly, we illustrate our technique using the *Slim-tree* [Traina-Jr. et al. 2002]. However, any other dynamic *MAM*, based on the use of representative objects to partition the data space, as the *Slim-tree* does, can be equally used. To the best of our knowledge, no *MAM* has yet been applied to reduce the time complexity of *k-medoids* based algorithms.

The remainder of the paper is organized as follows. Section 2 describes the existing *k-medoid* based algorithms and the *Slim-tree*. Section 3 presents our new efficient approach developed to scale up *k-medoid* based algorithms thus allowing fast clustering of large databases. An experimental evaluation of the approach is shown in Section 4. Finally, Section 5 gives the conclusions of this paper and suggestions for future works.

## 2. Basic Concepts

This section presents the existing *k-medoid* based algorithms and a brief description of the main aspects related to the *MAM Slim-tree*. The symbols and definitions presented in Table 1 are used throughout this paper.

**Table 1. Summary of Symbols and Definitions.**

Symbols	Definitions
$k$	number of clusters
$n$	number of objects in the dataset
$S$	set of objects to be clustered
$s_j$	an object $\in S$
$R$	set of objects $\in S$ selected as medoids
$r_j, r_c$	objects $\in R$
$d()$	dissimilarity measure (distance function) between two objects

### 2.1. Clustering Algorithms

The objective of *k-medoids* based algorithms is to find a non-overlapping set of clusters, so that each cluster has one representative object (the medoid), i.e., an object that is the most centrally located in the cluster considering a dissimilarity or distance measure. In order to do so, these algorithms perform two main steps:

- An **initialization** step, where an initial set of  $k$  objects are selected as medoids;
- An **evaluation** step, where they try to minimize an objective function usually based on the sum of the total distance among non-selected objects and their medoids, i.e., the evaluation step tries to minimize:

$$\sum_{j=1}^n d(r_i, s_j) , \quad (1)$$

where  $s_j \in S$  and  $d(r_i, s_j) < d(r_c, s_j), \forall r_i, r_c \in R, r_i \neq r_c$ . The smaller the sum of distances among the medoids and all the other objects of their clusters, the better the clustering.

The three best-known  $k$ -medoid based algorithms are *PAM* (*Partitioning Around Medoids*), *CLARA* (*Clustering LARge Applications*) and *CLARANS* (*Clustering Large Applications based upon RANdomized Search*) [Kaufman and Rousseeuw 2005]. The main aspects related to each one of these algorithms are described in the next subsections.

### 2.1.1. PAM Algorithm

*PAM* [Kaufman and Rousseeuw 1987] is one of the earliest  $k$ -medoids based algorithms. It is based on an iterative process of optimization that evaluates the effect of a swap between a medoid object and a non-medoid object relocating objects between perspective clusters. It can be expressed in the following way [Kaufman and Rousseeuw 2005].

1. **Build Phase:** In this phase, an initial set  $R$  of  $k$  representative objects is selected. The first selected object is the one for which the sum of the dissimilarities to all other objects is as small as possible. Thus, the first selected object is the dataset medoid. The other  $(k - 1)$  medoids are selected subsequently, one at a time, considering the objects that most decrease the objective function.
2. **Swap Phase:** This phase computes the total cost  $T_{ih}$  for all pairs of objects  $r_i$  and  $s_h$ , where  $r_i \in R$  is currently selected and  $s_h \in S$  is not.
3. **Selection Phase:** This phase selects the pair  $(r_i, s_h)$  which minimizes  $T_{ih}$ . If the minimum  $T_{ih}$  is negative, the swap is carried out and the algorithm reiterates Step (2). Otherwise, for each non-selected object, the most similar medoid is found and the algorithm stops.

The guiding principle of the *PAM* clustering process resides in Step (2). As it can be seen, it requires trying all objects that are currently not medoids and thus it presents a very expensive computational cost,  $O(k(n - k)^2)$  in each iteration [Ng and Han 1994]. The *PAM* algorithm results in high quality clusters, as it tries every possible combination, working effectively for small datasets (e.g., 100 objects in 5 clusters). However, due to its computational complexity, it is not practical for clustering large datasets.

### 2.1.2. CLARA Algorithm

The computational complexity of the *PAM* algorithm motivated Kaufman and Rousseeuw to develop *CLARA*, a clustering algorithm based on sampling

[Kaufman and Rousseeuw 2005]. *CLARA* draws multiple samples of the dataset and applies *PAM* on each sample. Then each object of the entire dataset is assigned to the resulting medoids, the objective function is computed, and the best set of medoids is returned as the output. Experiments described in [Kaufman and Rousseeuw 2005] indicated that 5 samples of size  $40 + 2k$  give satisfactory results. The computational complexity of each iteration of *CLARA* is of  $O(ks^2 + k(n - k))$ , where  $s$  is the size of the sample.

### 2.1.3. CLARANS Algorithm

*CLARANS* was developed in the context of spatial data mining. It uses a randomized search strategy in order to improve on both *PAM* and *CLARA* algorithms in terms of efficiency (computation complexity or time) and effectiveness (average distortion over the distances) respectively. When searching for a better medoid in the evaluation step, *CLARANS* randomly chooses objects from the remaining  $(n - k)$  objects. The number of objects tried in this step is restricted by a parameter provided by the user (*maxNeighbor*). If no better solution is found after *maxNeighbor* attempts, the local optimal is assumed to be reached. The procedure continues until *numLocal* local optimals have been found. It was recommended that the parameters *maxNeighbor* and *numLocal* be set to 2 and  $\max(50, 1.25\% \text{ of } k * (n - k))$ , respectively [Ng and Han 2002].

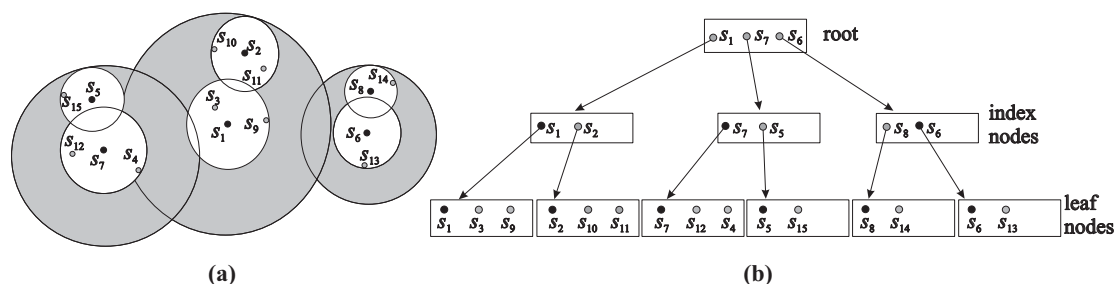
The computational complexity of *CLARANS* is  $O(n^2)$  in terms of the number of objects. Three focusing techniques was proposed in [Ester et al. 1995] employing R\*-trees [Beckmann et al. 1990] in order to make *CLARANS* more efficient for large spatial databases.

## 2.2. The MAM Slim-tree

The basic structure of any metric tree, such as e.g. *M-tree* [Ciaccia et al. 1997], *Slim-tree* [Traina-Jr. et al. 2002] and *DBM-tree* [Vieira et al. 2004], divides the data space into regions using representatives to which the other objects in each region will be associated with. The objects of each region are stored in a node that has a covering radius. Only objects within this radius are associated with the representative. The *Slim-tree* stores the data in the leaves and creates an appropriate hierarchy on top. An important factor that affects the search performance of this kind of tree is related to the degree of overlapping among the nodes. The *Slim-tree* is a dynamic structure that was developed to reduce the overlapping among regions in each level, and to optimize disk accesses on nearest neighbor and range queries.

Like many bottom-up structures (e.g. *B-tree*), its construction is as follows. The objects are inserted in a node up to its capacity. When a new object must be inserted, the node splits and the objects are distributed between the two nodes. One object of each splitted node is elected to go to an upper level, which is done recursively. These objects are the nodes representatives, and they are associated with a radius that covers the respective nodes. Figure 1 presents an example of a *Slim-tree* of 15 objects  $\{s_1, \dots, s_{15}\}$  with maximum node capacity of 3 objects.

The representative objects in a given level and their correspondent radius may be overlapped. This leads to the problem of qualifying more than one node to host a new



**Figure 1. A *Slim-tree* representation with 15 objects (a) and its logic structure (b). The white circles in (a) represent the leaf nodes, while the gray circles represent the index nodes. The objects in black are representatives.**

object. In order to build a *Slim-tree*, it is necessary to have a policy employed when a new object is inserted and more than one node is qualified for the insertion of the new object (*ChooseSubtree* algorithm) [Traina-Jr. et al. 2002]. The *Slim-tree* has three options for the *ChooseSubtree* algorithm:

1. *random*: randomly choose one of the qualifying nodes;
2. *minDist*: choose the node that has the minimum distance from the new object and the center of the node;
3. *minOccup*: choose the node that has the minimum occupancy among the qualifying ones.

It is important to note that the choice of this policy affects the features of the resultant tree regarding the degree of overlapping among nodes. For example, the *minOccup* option generates shallow trees with higher node occupation rates leading to a smaller number of disk accesses on queries. However, this option also leads to a higher overlapping degree of the nodes. On the other hand, the *minDist* option generates higher trees with lower node occupation rates and a lower overlapping degree among nodes.

By definition, a *Slim-tree* indirectly divides a metric space into a number of clusters from the number of objects and the node capacity. Unfortunately, this number of clusters is not a parameter given by the user, so it is not possible to let the *Slim-tree* do the whole clustering. On the other hand, the representative objects stored in the index nodes are, in a sense, approximate cluster centers on each level of the tree. And, although the use of this information does not allow the selection of any number  $k$  of clusters directly, it can be used as a convenient starting point to cluster the data.

### 3. The Proposed Algorithm

The computational complexity of the *k-medoid* based algorithms presented above depend on  $n$  and  $k$ . In order to reduce this complexity, a common approach is to reduce the number of all objects that are submitted to these algorithms making a sample of the original database. In this approach, the quality of the sampling is crucial for the quality of the resulting clustering.

In this paper, we propose to speed up *k-medoid* based algorithms using a *MAM*. Therefore, to improve the scalability of *k-medoid* based algorithms our strategy is to perform sampling operations based on features of a *MAM*, which hierarchically divides the

data space into regions, and assigns a representative to each region. As the *Slim-tree* meets this requirement, we use it as the basis to describe our technique in this paper. This strategy is based on the assumption that by construction, the node centers of the *Slim-tree* are reasonably well distributed over the data domain, as well as they are the natural centers of the regions that they represent. The center of a subtree can be considered as the center of mass of the objects stored in that subtree. Therefore, instead of considering all objects of a dataset to compute the clustering algorithm, only the objects of a chosen tree-level are considered. This sampling strategy considers the distance of an object to the center of a subtree as being approximately the average of the distances for every object stored in that subtree.

By construction, each level of a *Slim-tree* represents a data space division with a granularity that grows from the root to the leaves, eventually storing all the objects of a dataset. One question we need to answer is: which level of the tree has enough information about the data distribution that can lead to a clustering with a lower computational cost while keeping a reasonable quality?

The upper levels of a tree (close to the root level) do not contain much information about the data distribution because the data are grouped by a small number of representatives. On the other hand, the lower levels (close to the leaf level) tend to have too much information slowing down the algorithm. Intuitively, the middle levels of the tree may contain enough information about the data distribution that can lead to suitable clustering. The experiments we have performed indicated that the middle level of the tree really is a good choice.

Although the proposed approach may also be applied to other *k-medoid* based algorithms, we will consider only the *PAM* algorithm in this paper as it is the one presenting the better quality of the clusters identified. The proposed sampling algorithm using the *Slim-tree* applied to *PAM* will be referred to as *PAM-SLIM*, and it can be depicted as follows:

1. **Pre-process phase.** In this phase, the parameters for the construction of the *Slim-tree* are set, and the tree is built.
  - (a) Choose the parameters for the tree construction: the page size and the *ChooseSubtree* algorithm.
  - (b) Build the *Slim-tree*.
2. **Initialization phase.** In this phase, the objects that should be considered in the clustering phase are selected.
  - (a) Find the middle level of the tree as  $h_m$ . If level  $h_m$  does not have at least  $k$  objects, then select the next level.
  - (b) Let  $S_{h_m}$  be the set of objects in the  $h_m$  level.
3. **Clustering phase.** In this phase, the objects  $S_{h_m}$  selected in Step (2) are submitted to *PAM*.
  - (a) Apply the *PAM* algorithm over  $S_{h_m}$ .
  - (b) Assign each object of the entire dataset to the set of medoids returned by *PAM*.

The *PAM-SLIM* algorithm is divided in three main phases. The first phase is responsible for the construction of the *Slim-tree*. This phase allows the specification of some

of the parameters needed for the construction of the tree, such as the *ChooseSubtree* algorithm and the node page size. As mentioned in Section 2.2, the *Slim-trees* built with the different options provided for the *ChooseSubtree* algorithm tend to present different degrees of overlap among their nodes. Thus, the possibility of determining the *ChooseSubtree* option, in the pre-process phase, allows the evaluation of the results obtained from the *PAM-SLIM* algorithms when considering different configurations of the *Slim-tree*. Another parameter that deserves attention in the construction of a *Slim-tree* is the node page size. As the *PAM-SLIM* algorithms choose their medoids based on the node representatives, the node size affects the behavior of the algorithms. Thus, this parameter must be set according to the dataset to be clustered.

Once the *Slim-tree* has been constructed, a subset of objects that must be submitted to PAM are selected in the second phase. This subset of objects is composed of the node representatives stored in the middle level of the tree. The last phase applies the *PAM* algorithm over the objects selected in the previous step and assigns each object of the entire dataset to the set of medoids returned by *PAM*.

#### 4. Experimental Evaluation

In order to show the effectiveness and the efficiency of our approach, we will present three representative sets of experiments chosen from the ones we have performed. To evaluate the clustering obtained by the techniques under analysis (*effectiveness*), we compute the average distance of the resulting clustering, i.e., the average distance of all objects from their medoids (smaller values for average distance indicate better clustering). The *efficiency* was measured by the number of distance calculations.

In the first set of experiments, we aimed at comparing the proposed approach *PAM-SLIM* with the traditional *k-medoid* based algorithms *PAM*, *CLARA* and *CLARANS*. As *PAM* is not suitable for large databases due to its computational complexity (see Section 2.1.1), it was run only in the first set of experiments. The second series of experiments aimed at measuring the scalability of our approach when the database increases. These experiments compared our approach to *CLARANS* and *CLARA*. The third set of experiments showed the performance of our approach using a large, real-world dataset. It is important to note that, in all the experiments, the efficiency graphics are in *log* scale for the number of distance calculation axis, due to the large difference of the results obtained for the tested algorithms.

In every experiment, we have considered two configurations for the proposed approach *PAM-SLIM*: *PAM-SLIM-MD* which uses the *minDist* option of the *Slim-tree* and *PAM-SLIM-MO* which uses the *minOccup* option. This was done mainly, because the *Slim-trees* built with these two options present differences in the overlap degree among their nodes (see Section 2.2). Thus, we have intended to find out which option was more appropriated for the construction of the tree that is employed to sample the objects submitted to *PAM*.

In order to find the ideal page size, the experiments were run varying the page size for the *Slim-tree* methods. The node page sizes were chosen based on the size of the objects of each dataset. It is important to note that the algorithms *PAM*, *CLARA* and *CLARANS* only run in main memory and thus, they are not influenced by the page size.

The five clustering algorithms experimented – *PAM*, *CLARA*, *CLARANS*, *PAM-*

*SLIM-MD* and *PAM-SLIM-MO* – were implemented within the same platform, using the C++ language into the *Arboretum MAM library* [GBDI-ICMC-USP 2006], in order to obtain a fair comparison. *CLARA* and *CLARANS* were configured using their best recommended setup as described in Section 2.1. The experiments were performed in a PC with an Intel P4 2.4 GHz CPU, 1 GB RAM and 60 GB of disk space.

Nine datasets were employed in the experiments. Eight were composed of synthetic data and were built to enable a thoughtful evaluation of the algorithms. The ninth dataset was composed of real adimensional data and was used aiming at analyzing the behavior of our algorithm in real-world. The description of the datasets are presented in Table 2 along with its name, the total number of objects (# Objs.), the object size in bytes, the number of clusters ( $k$ ) used to generate the dataset, the dimensionality of the dataset ( $D$ ), and the distance function used ( $d(\ )$ ).

**Table 2. Description of the synthetic and real-world datasets used in the experiments.**

Name	# Obj.	Object size (Bytes)	$k$	$D$	$d(\ )$	Description
<i>Synt10_5k</i>	10,000	24	5	5	$L_2$	Synthetic vector data with Gaussian distribution in a 5-d unit hypercube. The process to generate these datasets is described in [Ciaccia et al. 1997].
<i>Synt10_10k</i>	10,000	24	10	5	$L_2$	
<i>Synt10_15k</i>	10,000	24	15	5	$L_2$	
<i>Synt10_20k</i>	10,000	24	20	5	$L_2$	
<i>Synt30_10k</i>	30,000	24	10	5	$L_2$	
<i>Synt60_10k</i>	60,000	24	10	5	$L_2$	
<i>Synt90_10k</i>	90,000	24	10	5	$L_2$	
<i>Synt120_10k</i>	120,000	24	10	5	$L_2$	
<i>MedHisto</i>	40,000	44 - 380 (avg. 280)	-	-	$L_M$	Metric histograms of medical gray-level images. This dataset is adimensional and was generated at GBDI-ICMC-USP. For more details on this dataset and the used distance function see [Traina et al. 2002].

#### 4.1. Evaluating *PAM-SLIM* vs *k-medoid* based algorithms

This first set of experiments were carried out in order to compare the two configurations of our method *PAM-SLIM-MD* and *PAM-SLIM-MO* with the existent *k-medoid* based algorithms *PAM*, *CLARA* and *CLARANS*.

Figure 2 shows the efficiency ((a), (b), (c) and (d)) and effectiveness ((e), (f), (g) and (h)) comparison of *PAM*, *CLARANS*, *CLARA*, *PAM-SLIM-MD* and *PAM-SLIM-MO* for the datasets *Synt10\_5k*, *Synt10\_10k*, *Synt10\_15k* and *Synt10\_20k*, where 5, 10, 15 and 20 medoids were selected, respectively. Notice that, as the traditional clustering algorithms – *PAM*, *CLARANS* and *CLARA* – are executed in main memory, their results are shown as the three first bars in the graphs without a page size label. *PAM-SLIM-MD* and *PAM-SLIM-MO* were executed using disk, employing node page sizes of 1,024 and 2,048 bytes.

Considering the node page sizes employed in the tests for both *PAM-SLIM* algorithms, the node page size of 2,048 bytes was the one that presented the best trade-off between efficiency and effectiveness. In some cases, such as in Figure 2(e), the quality of the clusters obtained from the *PAM-SLIM* algorithms, considering the node page size

of 1,024 bytes, was better than the ones considering the node page size of 2,048 bytes. However, if we observe the correspondent graph of efficiency (Figure 2(a)), presented in *log* scale for the vertical axis, the *PAM-SLIM* algorithms execution, with node page size of 2,048 bytes, show an improvement of efficiency of 5.6 times for *PAM-SLIM-MD*, and of 4.5 times for *PAM-SLIM-MO* when compared to the same algorithms executed with node page size of 1,024 bytes. Therefore, node page sizes of 2,048 bytes presented a better overall result.

For the *PAM-SLIM-MD*, we observed improvements of efficiency ranging from 14 to 23 times faster when compared to *CLARANS*, whereas the loss of effectiveness varies only from 0.9% to 4.1%. Compared to *PAM*, the efficiency was improved by a factor of 1,054 to 1,459 times (more than a thousand times faster), although we observed a decrease of effectiveness ranging from only 7.1% to 10.1%. Compared to *CLARA*, our improvement of effectiveness ranged from 10.8% to 27.8%, although the *CLARA* algorithm had presented a smaller number of distance calculations.

Despite the fact that the loss of effectiveness, in terms of the average distance of the resulting clustering, in our approach reaches 10% compared to *PAM* and 4.1% compared to *CLARANS*, if we consider the percentage of objects that are really misgrouped, this loss of effectiveness is smaller. This is due to the fact that the misgrouped objects are near the border of the clusters, so their distances to their medoids are large. Table 3 shows the percentage of objects that were associated with different clusters for the algorithms *CLARANS*, *CLARA*, *PAM-SLIM-MD* and *PAM-SLIM-MO* when compared to *PAM* clustering (the best and exhaustive algorithm). Notice that the *PAM* and *PAM-SLIM* algorithms perform the sampling in a deterministic way, so that every execution always gives the same result. The *CLARANS* and *CLARA* algorithms are based on random samples, so we present the average of 10 executions of each algorithm. Notice that the values for the *PAM-SLIM* algorithms are comparable to the execution of the *CLARANS* algorithm, being better for the datasets *Synt10\_5k* and *Synt10\_10k*, and also that they run in a fraction of the time needed by *CLARANS* (from 14 to 36 times less distance calculations in this set of experiments). It is also important to note that the *CLARA* algorithm degenerated with the increase of the *k* number of clusters asked, reaching 13.5% of the objects associated with different clusters when compared to *PAM*. The quality of the clusters generated by the *CLARA* algorithm also degenerates as the number of objects increases, as it is demonstrated in the next sets of experiments.

**Table 3. Percentage of objects that were associated with different clusters when compared to *PAM* clustering. Considering *PAM-SLIM-MD* and *PAM-SLIM-MO* run with page size of 2,048 bytes.**

Dataset	<i>CLARANS</i>	<i>CLARA</i>	<i>PAM-SLIM-MD</i>	<i>PAM-SLIM-MO</i>
<i>Synt10_5k</i>	0.4%	0.7%	0.2%	0.2%
<i>Synt10_10k</i>	1.2%	3.9%	1.3%	0.8%
<i>Synt10_15k</i>	1.7%	5.9%	1.8%	3.5%
<i>Synt10_20k</i>	2.7%	13.5%	3.7%	5.5%

Table 4 shows the time to execute a clustering on the datasets measured as hours:minutes:seconds. The time comparison of *PAM*, *CLARANS*, *PAM-SLIM-MD* and *PAM-SLIM-MO* considers the page size that presented the best trade-off for *PAM-SLIM-*

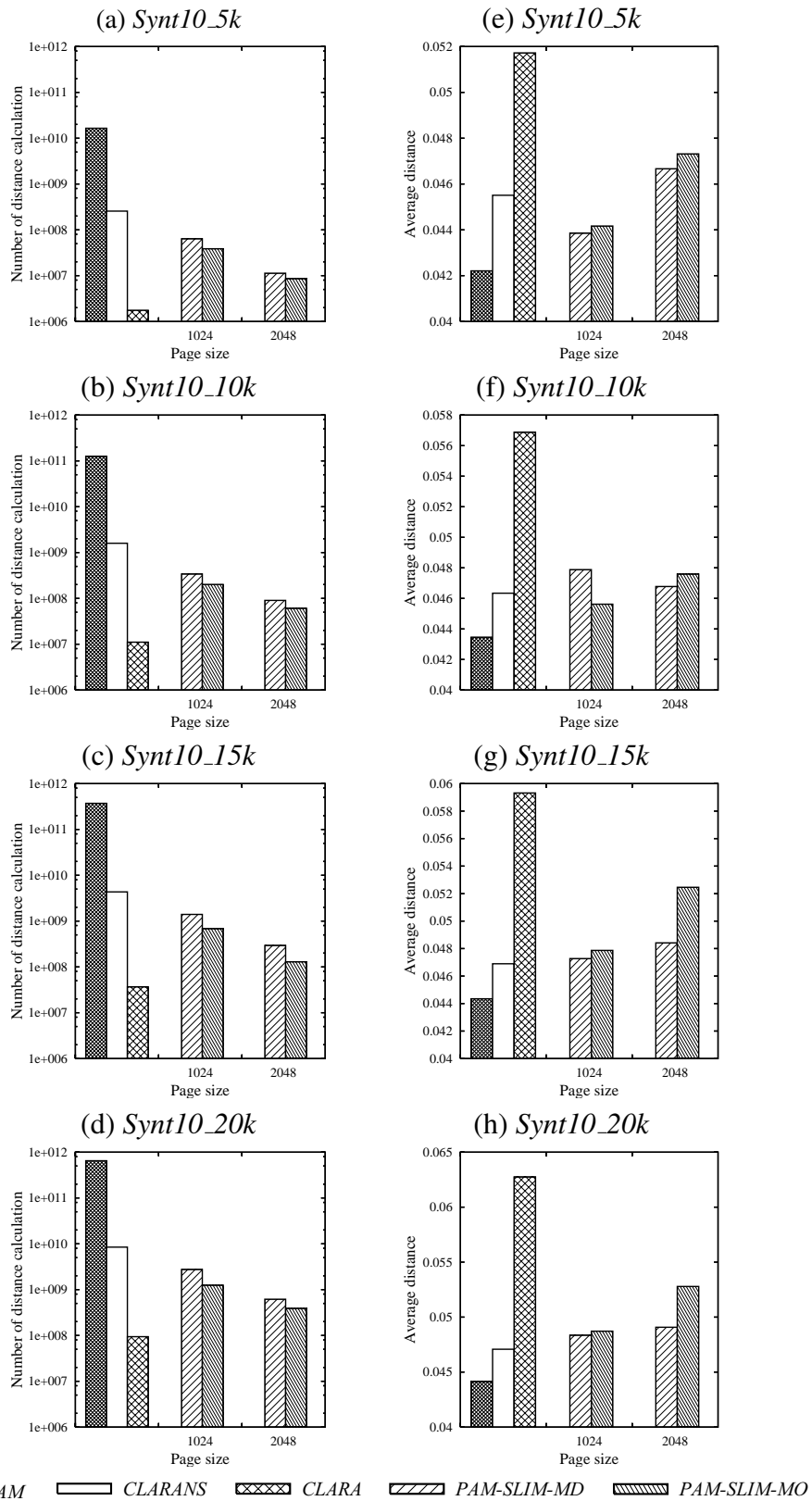


Figure 2. Efficiency ((a), (b), (c) and (d)) and effectiveness ((e), (f), (g) and (h)) comparison of PAM, CLARANS, CLARA, PAM-SLIM-MD and PAM-SLIM-MO for Synt10\_5k, Synt10\_10k, Synt10\_15k and Synt10\_20k datasets.

*MD* and *PAM-SLIM-MO* (page size of 2,048 bytes). As the experiments show, *PAM-SLIM* was more than a thousand times faster than *PAM* while producing clusterings of comparable quality.

**Table 4. Time comparison of *PAM*, *CLARANS*, *PAM-SLIM-MD* and *PAM-SLIM-MO* for *Synt10\_5k*, *Synt10\_10k*, *Synt10\_15k* and *Synt10\_20k* datasets. Considering *PAM-SLIM-MD* and *PAM-SLIM-MO* run with page size of 2,048 bytes. Time measured in hours:minutes:seconds.**

Dataset	PAM	CLARANS	<i>PAM-SLIM-MD</i>	<i>PAM-SLIM-MO</i>
<i>Synt10_5k</i>	01:31:04	00:01:21	00:00:21	00:00:14
<i>Synt10_10k</i>	11:28:31	00:08:22	00:01:52	00:01:10
<i>Synt10_15k</i>	32:28:18	00:22:37	00:07:29	00:03:35
<i>Synt10_20k</i>	57:26:11	00:44:27	00:14:46	00:06:50

Although the *CLARA* algorithm had presented the best efficiency (in terms of the number of distance calculations) in all the tests showed herein, its clustering quality is worse, not being comparable to that of *PAM*, considering its recommended configuration. In order to improve its clustering quality, it is necessary to increase the number of samplings and/or their sizes. However, this results in a large reduction of its efficiency. This makes the algorithm *PAM-SLIM-MD* the one that presents the best trade-off between efficiency and effectiveness for the datasets tested here. In the next experiments, we have not compared the results with *PAM* algorithm, because of its prohibitive computational cost.

#### 4.2. Evaluating the scalability of the *PAM-SLIM* algorithms

The datasets *Synt30\_10k*, *Synt60\_10k*, *Synt90\_10k* and *Synt120\_10k* were used in the second set of experiments aiming at measuring the scalability of our approach regarding the increasing number of data items (see Table 2). In it, 10 medoids were asked for clustering each dataset. Figure 3 shows the efficiency and effectiveness comparison of *CLARANS*, *CLARA*, *PAM-SLIM-MD* and *PAM-SLIM-MO*. The node page sizes employed in these experiments for the *PAM-SLIM-MD* and *PAM-SLIM-MO* algorithms varied from 1,024 to 8,192 bytes.

As it can be seen in Figure 3 ((e), (f), (g) and (h)), the node page size that resulted in the best overall clustering quality for the *PAM-SLIM* algorithms was of 2,048 bytes. Taking this page size into account, we have observed that the *PAM-SLIM-MO* obtained an improvement of efficiency ranging from 7 to 26 times compared to *CLARANS* (the clustering algorithm that presented the best clustering quality) as can be seen from Figure 3 ((a), (b), (c) and (d)), whereas the effectiveness varied from 0.7% of gain to 2.7% of loss. It is important to note that, considering larger page sizes, the improvement of efficiency was even higher while producing a small decrease of the effectiveness. Comparing the configuration of the *PAM-SLIM-MO* with node page size of 2,048 bytes and the configurations of 4,096 and 8,192 bytes, the improvement of efficiency obtained ranged from 4.1 to 7.7 times and from 20.4 to 22.9 times, respectively. In these cases, the loss of effectiveness ranged from 1.6% to 3.0% for the configuration of 4,096 bytes and from 3.1% to 4.6% for the configuration of 8,192 bytes.

In the second set of experiments, although the *CLARA* algorithm presented the best

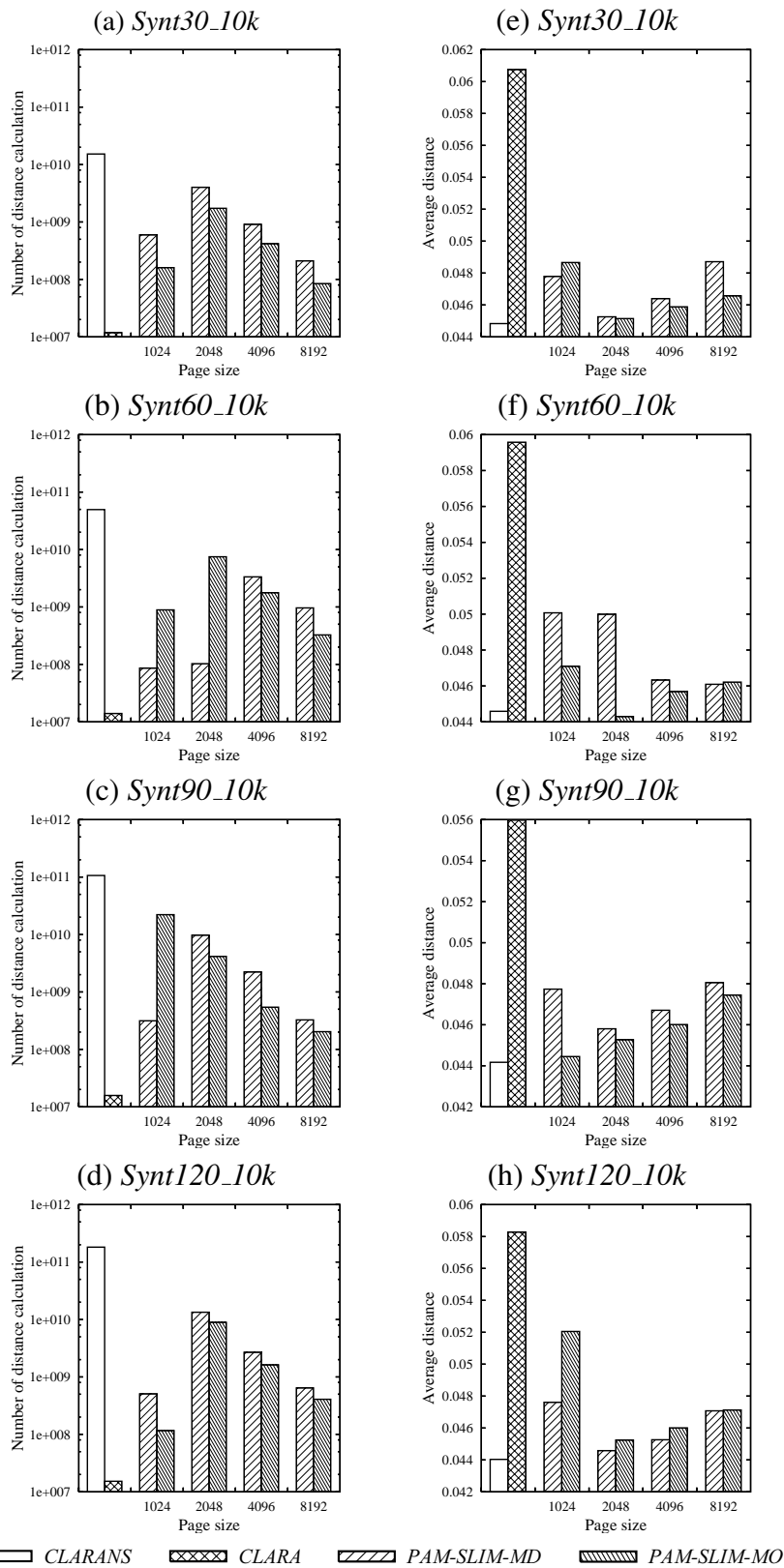


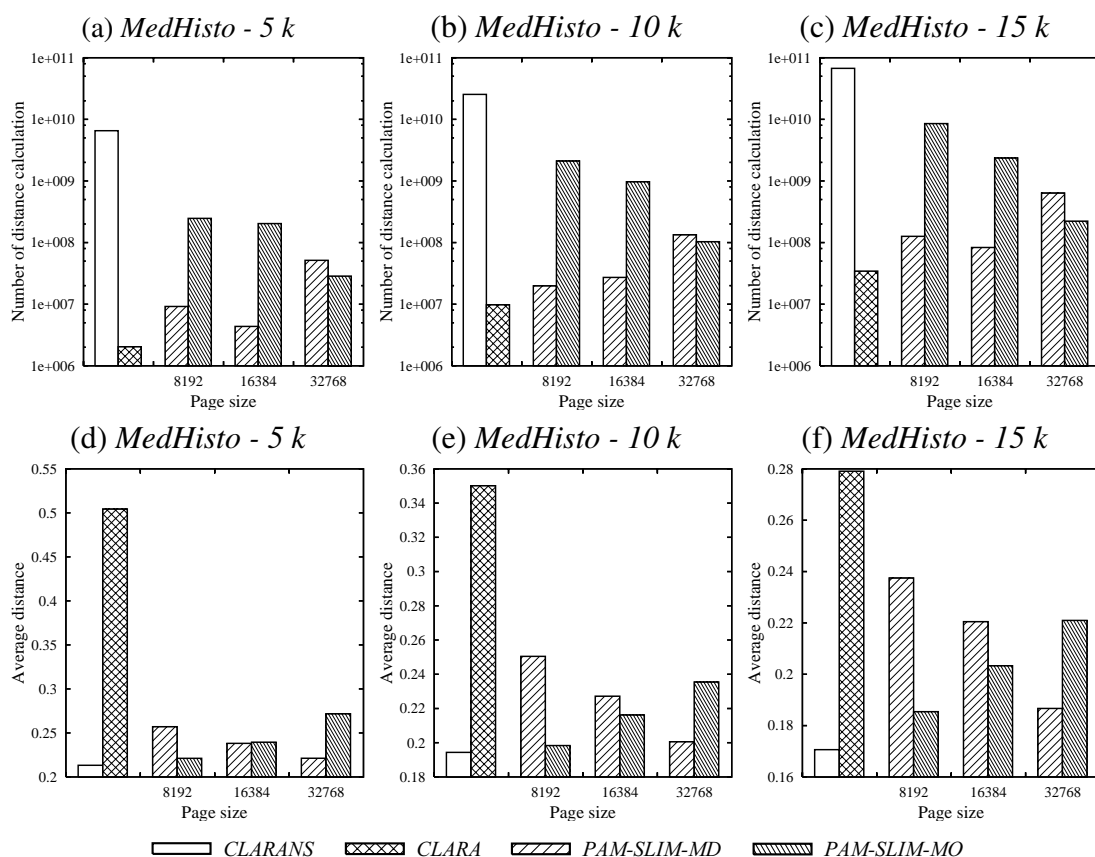
Figure 3. *Efficiency* ((a), (b), (c) and (d)) and *effectiveness* ((e), (f), (g) and (h)) comparison of *CLARANS*, *CLARA*, *PAM-SLIM-MD* and *PAM-SLIM-MO* for *Synt30\_10k*, *Synt60\_10k*, *Synt90\_10k* and *Synt120\_10k* datasets.

efficiency among the tested algorithms, its loss of effectiveness was greater than 21.1% when compared to *CLARANS* and greater than 19.1% when compared to *PAM-SLIM-MO*.

### 4.3. Exploring a real database

The last set of experiments used the real-world adimensional dataset, *MedHisto*, in order to observe the behavior of our algorithms in a real situation. In these experiments, 5, 10 and 15 medoids were selected from 40,000 objects of the *MedHisto* dataset. The node page sizes employed for the *PAM-SLIM* algorithms varied from 8,192 to 32,768 bytes.

Figure 4 shows the efficiency ((a), (b) and (c)) and effectiveness ((d), (e) and (f)) comparison of *CLARANS*, *CLARA*, *PAM-SLIM-MD* and *PAM-SLIM-MO* for this dataset. The best clustering quality obtained for the *PAM-SLIM-MO* employed the node page size of 8,192 bytes, whereas for the *PAM-SLIM-MD* algorithm the best clustering quality was achieved considering the page size configuration of 32,768 bytes. Observing the best configuration for the *PAM-SLIM-MO*, the improvement of efficiency ranged from 8 to 26 times when compared to *CLARANS*, while the loss of effectiveness ranged from 2.0% to 8.0%. For the *PAM-SLIM-MD*, the improvement of efficiency ranged from 106 to 190 times compared to *CLARANS*, whereas the loss of effectiveness ranged from 3.1% to 8.6%.



**Figure 4. Efficiency ((a), (b) and (c)) and effectiveness ((d), (e) and (f)) comparison of *CLARANS*, *CLARA*, *PAM-SLIM-MD* and *PAM-SLIM-MO* for the *MedHisto* dataset.**

It has been noted that, the loss of effectiveness presented by the *PAM-SLIM* algo-

rithms in this set of experiments, was much lower than the one presented by *CLARA* when compared to *CLARANS*. The loss of effectiveness presented by *CLARA* was superior to 39% when compared to *CLARANS* and superior to 33% when compared to *PAM-SLIM-MD*.

## 5. Conclusions

This paper presented a new algorithm, *PAM-SLIM* which employs metric access methods to scale-up *k-medoid* based algorithms. The efficiency of *k-medoid* based algorithms relies on the initial selection of the medoids. Our proposed algorithm assumes that a metric tree tends to naturally choose suitable medoids to be its node representatives. The experiments we performed confirmed that this assumption indeed holds. The strategy employed by this algorithm can be efficiently applied to cluster multi-dimensional as well as adimensional datasets.

Our experiments have shown that the proposed algorithm presented suitable results for several configurations of the *Slim-tree* using different node page sizes and *ChooseSubtree* policies. When compared to *PAM* and *CLARANS*, the *PAM-SLIM* algorithms presented impressive improvement of efficiency (being more than a thousand times faster), whereas maintaining a comparable clustering quality, thus offering a very good trade-off between efficiency and effectiveness. The new algorithm also presented much better clustering quality than *CLARA* for all the tested datasets. Moreover, notice that our proposed algorithm runs on datasets stored on disk, whereas the other algorithms run only in main memory. Even though, our algorithm remains much faster than the ones that presented comparable clustering quality (*PAM* and *CLARANS*). Hence, the efficiency presented by the *PAM-SLIM* algorithm allows the execution of clustering algorithms in database management systems.

There are a few issues that deserve further research. Although the two configurations employed by the *PAM-SLIM* algorithms (*PAM-SLIM-MD* and *PAM-SLIM-MO*) have obtained suitable results, they presented different behaviors in each dataset tested. Future works shall include the identification of parameters to be measured in a dataset to define beforehand where each configuration is the best choice.

## References

- Beckmann, N., Kriegel, H.-P., Schneider, R., and Seeger, B. (1990). The R\*-tree: An efficient and robust access method for points and rectangles. In *ACM SIGMOD International Conference on Management of Data*, pages 322–331, Atlantic City, NJ. ACM.
- Chu, S.-C., Roddick, J. F., and Pan, J. S. (2002). An efficient k-medoids-based algorithm using previous medoid index, triangular inequality elimination criteria, and partial distance search. In *International Conference on Data Warehousing and Knowledge Discovery (DaWaK)*, pages 63–72, London, UK. Springer-Verlag.
- Ciaccia, P., Patella, M., and Zezula, P. (1997). M-tree: An efficient access method for similarity search in metric spaces. In *International Conference on Very Large Data Bases (VLDB)*, pages 426–435, Athens, Greece. Morgan Kaufmann.

- Ester, M., Kriegel, H.-P., and Xu, X. (1995). Knowledge discovery in large spatial databases: focusing techniques for efficient class identification. In *International Symposium on Advances in Spatial Databases*, volume 951, pages 67–82, Portland, ME. Springer.
- GBDI-ICMC-USP (2006). GBDI Arboretum Library. <http://gbdi.icmc.usp.br/arboretum/>.
- Guha, S., Rastogi, R., and Shim, K. (1998). Cure: an efficient clustering algorithm for large databases. In *ACM SIGMOD International Conference on the Management of Data*, pages 73–84, Seattle, WA, USA.
- Han, J. and Kamber, M. (2001). *Data mining: Concepts and techniques*. Academic Press, San Diego, CA.
- Hartigan, J. and Wong, M. (1979). Algorithm as136: A k-means clustering algorithm. *Applied Statistics*, 28:100–108.
- Jain, A., Murty, M., and Flynn, P. (1999). Data clustering: A review. *ACM Computing Surveys*, 31(3):264–323.
- Kaufman, L. and Rousseeuw, P. J. (1987). Clustering by means of medoids. *Statistical Data Analysis based on the L1 Norm, Elsevier*, pages 405–416.
- Kaufman, L. and Rousseeuw, P. J. (2005). *Finding groups in data: An introduction to cluster analysis*. John Wiley and Sons.
- Ng, R. T. and Han, J. (1994). Efficient and effective clustering methods for spatial data mining. In *International Conference on Very Large Data Bases (VLDB)*, pages 144–155, Santiago, Chile. Morgan Kaufmann.
- Ng, R. T. and Han, J. (2002). Clarans: A method for clustering objects for spatial data mining. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 14(5):1003–1016.
- Sibson, R. (1973). Slink: An optimally efficient algorithm for the single link cluster method. *Computer Journal*, 16:30–34.
- Traina, A. J. M., Jr., C. T., Bueno, J. M., and de A. Marques, P. M. (2002). The metric histogram: A new and efficient approach for content-based image retrieval. In *IFIP Working Conference on Visual Database Systems (VDB)*, pages 297–311, Australia.
- Traina-Jr., C., Traina, A. J. M., Faloutsos, C., and Seeger, B. (2002). Fast indexing and visualization of metric datasets using Slim-trees. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 14(2):244–260.
- Vieira, M. R., Chino, C. T.-J. F., and Traina, A. J. M. (2004). Dbm-tree: A metric access method sensitive to local density data. In *Brazilian Symposium on Databases (SBBD)*, pages 163–177, Brasília, DF. SBC.
- Zhang, Q. and Couloigner, I. (2005). A new and efficient k-medoid algorithm for spatial clustering. In *International Conference on Computational Science and Its Applications*, volume 3482 of *LNCS*, pages 181–189, Singapore. Springer-Verlag.
- Zhang, T., Ramakrishnan, R., and Livny, M. (1996). Birch: An efficient data clustering method for very large databases. In *ACM SIGMOD International Conference on Management of Data*, pages 103–114, Montreal, Quebec, Canada. ACM.