Journal of
the Brazilian Computer Society
a SpringerOpen Journal

RESEARCH Open Access

Optimizing metric access methods for querying and mining complex data types

Jessica Andressa de Souza[1*], Humberto Luiz Razente[2] and Maria Camila N Barioni[2]

Abstract

Background: There are several application scenarios that can take advantage from the efficient processing of similarity operations in complex data types, such as multimedia data. Among them, it is possible to mention the execution of more complex query types (e.g., similarity queries) and several well-known data mining algorithms (e.g., data clustering) that are directly based on similarity computations. In order to speed up the similarity-based comparisons performed by these approaches, it is possible to store the dataset in specialized data structures known as metric access methods (MAM).

Methods: In this article we present four node split policies that can be employed in the construction of M-tree, the pioneer dynamic MAM, and of Slim-tree, the M-tree successor.

Results: These policies allow faster tree construction, as they result in better distribution of elements on the tree nodes and require less distance calculations when compared with the previously proposed ones. Furthermore, trees built with these policies have shown to be more efficient for techniques that require similarity computations, such as nearest neighbors queries and data clustering algorithms.

Conclusion: The experimental results show that trees built with the proposed policies outperform those built with the original ones with regard to the number of disk accesses, the amount of distance calculations, and the time required to run the queries.

Keywords: Metric access methods; Similarity queries; Multimedia indexing; Data mining

Background

Since the beginning of commercial computer systems half a century ago, the increase in the amount of data demanded the development of systems tailored to efficiently store, represent, and manipulate them. Once the Database Management Systems (DBMS) emerged as a solution to this question, the researchers turned their attention to another important question: how to use the gathered data to get valuable information. In response to this, several data mining approaches were proposed to couple with this issue. Among them it is possible to mention the following: frequent pattern mining, data classification, and data clustering [1].

Nowadays, the rate of data generation is even higher and so is the complexity of the available data [2]. Therefore, the need for more efficient tools to allow both querying [3] and mining [4] large complex datasets is still an open question. Considering the multimedia data types, for example, the traditional querying approach based on attribute matching is not suitable. This fact has motivated the development of querying approaches based on the concept of similarity among complex data elements [5]. There are several examples of querying approaches that dealt with this issue in the scientific literature, such as the similarity selection algorithms (e.g., *k*-nearest neighbor selection and range selection) [6], the similarity join algorithms (e.g., *k*-nearest neighbor join, *k*-closest neighbor join, join around, and range join) [7], and the diversification of similarity selections [8].

The high cost imposed by the similarity-based approaches is related to both the feature vectors employed to represent the data and the distance calculations computed by the algorithms [9]. In order to speed up the similarity-based comparisons performed by these

*Correspondence: jessicasouza@usp.br
[1]Instituto de Ciências Matemáticas e de Computação, USP, Trabalhador Saocarlense, 400, São Carlos, Sao Paulo, Brazil
Full list of author information is available at the end of the article

approaches, it is possible to store the dataset in specialized data structures known as metric access methods (MAM). As many data mining approaches, such as data clustering, are based on similarity comparisons, they can also greatly benefit from the efficient processing of these operations in a MAM.

In the last decades, several MAM were proposed aiming at reducing the number of distance calculations, the number of disk accesses, and the total time spent to compute distance-based queries. At first, these methods were tailored to provide efficient similarity queries processing. There were no concerns about the effectiveness for the data mining approaches based on similarity comparisons.

Among the MAM, the ones based on ball partitioning may result in balanced hierarchies. The hierarchy contains two types of structures: index and leaf nodes. Each index node stores a set of elements and their respective covering radii, where each pair $< element, radius >$ defines a ball that encompasses all the elements in the branch it represents. During traversal, the triangle inequality property may be used to determine if there is an intersection of a ball with a query specification and therefore prunes branches that do not intersect. Finally, a leaf node contains data elements and references to the data they represent.

Many proposed MAM are not dynamic or are not suitable for secondary memory. M-tree [10] is the first balanced dynamic ball partitioning metric access method based on fixed-size disk pages. Its balance is guaranteed by the bottom-up construction, where a page overflow results in a node split, recursively updating the ancestors of the node. No periodic global reorganization is required when dealing with insertions and deletions. An overview about MAM may be found at [9,11].

The major issue related to MAM performance is the overlap among nodes. As far as the overlap increases, the efficiency of these structures decreases, as more nodes may be covered by a query region during a search operation. Although the M-tree, its successor the Slim-tree [12] and other MAM proposed, had dealt with this issue in the proposal of node splitting policies for the tree insertion process, we identified some shortcomings that led us to the proposal of new node split policies. The design of these policies took into account two requirements: MAM efficiency for querying and mining and also MAM effectiveness for mining processes.

Although we started to address this issue in a previous paper [13], there are new aspects that we carried out herein. In [13] we presented three new split policies together with an initial set of experiments that showed their superiority over the original ones when performing similarity queries in ball-partitioning-based MAM such as M-tree and Slim-tree. This article integrates the concepts introduced in our previous paper and extends it presenting: a more detailed description of the node split policies presented in [13], a new node split policy, and the results obtained with an exhaustive set of experiments that was conducted with the aim of evaluating the impact in processing both similarity queries and data clustering processes when employing different node split policies on the construction of the MAM that supports them.

Related work

Over the past two decades, several works have pursued the development of strategies aiming to reduce the overlap among nodes in dynamic MAM construction. Examples of works that dealt with this issue proposing node splitting policies are [10,12,14,15]. The work presented in [10] deals with node splitting by applying MinMax in the construction of the M-tree MAM. This algorithm finds the pair of elements that result in smaller node coverage requiring $O(n^3)$ distance calculations on the number of elements in a node. A more efficient algorithm was later employed by the Slim-tree MAM [12]. Its strategy consists in employing the minimum spanning tree (MST) algorithm as a node split policy, reducing the complexity to $O(n^2 \cdot \log(n))$ distance calculations. In addition to that, the work proposed in [12] also defines an evaluation metric to measure the overlap among nodes (called fat-factor) and presents an algorithm to reduce this overlap (called Slim-down).

Another split policy for the M-Tree MAM restricted to multidimensional data was presented in [14]. It is based on the choice of central elements from partitioned regions to promote as node representatives, which results in smaller overlaps. As it uses the k-means algorithm to compute these representatives, it cannot be employed in generic metric spaces.

CM-tree adopts a different strategy when splitting a node [15]. It employs a clustering approach that splits a node in two or possibly more nodes, recursively propagating the updates or splits up to the root. It also stores a distance matrix in each node with the pairwise distances of each pair of elements. During a search, the algorithm prunes branches based on both the distance matrix values and the triangle inequality. The price paid to store the matrix in each node is compensated by not computing these distances, especially if a data element occupies a large amount of bytes and if the distance function is expensive.

Other approaches that have been explored to build more efficient MAM include [16-18]. The work of [16] presents two new techniques to allow dynamic insertions to M-Tree. The first is a forced reinsertion strategy that avoids

the split of a leaf node that reached its storage capacity. The algorithm removes some elements from the overflowed node and reinserts them, hoping these elements will be placed into more suitable leaves. Finally, if the elements are reinserted in the same node, the node is split. The second strategy works on leaf selection. Instead of a single-way descent to the leaf (that may not find the optimal node to hold an element) and the multi-way leaf selection (that may result in linear scan), the authors propose a hybrid-way algorithm that finds the best candidates at each level, reducing the traversal. The drawback of this strategy is that it increases both the number of distance calculations and disk accesses during the insertion. During query execution, both techniques result in the reduction on the number of disk accesses and distance calculations when compared to M-tree. Among the factors that influenced these reductions are the increase of node occupation and the decrease of node overlap.

The use of a bulk load operation based on a static dataset to build an optimized Slim-tree is proposed in [17]. It builds the MAM hierarchy of a dataset in a top-down approach with reduced node overlapping, resulting in better query performance. The method may be used to recreate a low-performance index. The latter work mentioned above [18], presents EGNAT, a disk-based MAM that indexes data using hyperplanes instead of ball partitioning, resulting in an unbalanced structure with no overlapped regions. Compared to M-tree, EGNAT accesses more disk pages at search time, but on the other hand, computes fewer distances. A summary of the main features of the works described previously is presented in Table 1.

Table 1 Optimization strategies for efficient MAM construction

Work	Strategy	Data domain	Resultant MAM
[10]	Node splitting policy	Generic metric data space	Balanced
[12]	Node splitting policy	Generic metric data space	Balanced
[14]	Node splitting policy	Multidimensional data space	Balanced
[15]	Node splitting policy	Generic metric data space	Balanced
[16]	Dynamic reinsertions	Generic metric data space	Balanced
[17]	Bulk load operation	Generic metric data space	Balanced
[18]	Hyperplane partitioning	Generic metric data space	Unbalanced

The development of approaches to make data clustering algorithms, which are based on similarity comparisons, feasible for large-scale datasets has also been pursued for the data mining research community over the last decades. Among them, the use of sampling techniques has proved to be especially useful for data clustering methods that perform several iterations considering different initializations, such as CLARANS [19] and PAM [2]. Variants of these algorithms have been proposed in [20,21] respectively. In these works the algorithms focus on relevant parts of the dataset as the clustering process is performed on a sample dataset obtained from pages of specialized data structures that allow the indexing of multidimensional data. The former uses the R*-tree [22] and the latter employs the Slim-tree [12] to develop the algorithm called PAM-SLIM.

The strategy adopted by the PAM-SLIM algorithm, for example, is based on the assumption that each level of the Slim-tree indirectly divides the data space into a number of clusters equal to the number of elements stored at each level. The representative elements stored in the index nodes summarizes the information about the elements in the tree low levels and, thus, they can be viewed as approximate cluster centers on each level of the tree. This information is employed by PAM-SLIM to compose the sample dataset to cluster the data. It is important to note that the quality of the sample dataset is affected by the features of the tree regarding the distribution of the elements on the tree nodes.

Fundamental concepts

The time spent to run a query is a basic measure when evaluating an index structure. Considering MAM, time is directly related to the computation of distances and disk page accesses. Depending on the nature of data, distance calculations may impact time as much as randomly accessing disk pages.

In order to provide pruning opportunities when performing similarity queries, MAM employ distance functions that obey the properties defined by the metric space algebra [11]. Formally, a metric space is a pair $< \mathbb{S}, d() >$, where $\mathbb{S}$ is the data domain and $d()$ is a metric distance function that complies with the following properties: *identity*: $d(s_1, s_1) = 0$; *symmetry*: $d(s_1, s_2) = d(s_2, s_1)$; *non-negativity*: $0 < d(s_1, s_2) < \infty$ if $s_1 \neq s_2$; and *triangular inequality*: $d(s_1, s_2) \leq d(s_1, s_3) + d(s_3, s_2)$, $\forall\, s_1, s_2, s_3 \in \mathbb{S}$. It is important to note that vector datasets with any L_p distance function, such as the Euclidean distance (L_2), are special cases of metric spaces.

The basic idea adopted by MAM in order to organize the data elements in its hierarchical structure consists in dividing the data space into regions using representatives

to which the other elements in each region will be associated with. The elements of each region are stored in a node that has a covering radius. Only elements within this radius are associated with the representative.

Briefly, the construction approach employed by bottom-up MAM, such as *M-tree* and *Slim-tree*, is performed as described following. For each new element to be inserted, it is necessary to traverse the tree from the root to the leaves in order to find the leaf node with a coverage radius that encompass this element. If more than one node qualifies to host the new element, a choose subtree policy must be used.

On the other hand, if no node qualifies to host the new element, the node that has the closest representative of the new element is selected. This process is applied recursively to all levels of the tree until it reaches a leaf node where new elements are actually inserted. The elements are inserted in a node up to its capacity. When another element must be inserted in a full node, a new node is created (split) and the elements are distributed between the two nodes according to a node split policy. One element of each split node is promoted to the immediate upper level, which is done recursively. The elected elements are the nodes' representatives, and they are associated with a radius that covers the respective nodes. Figure 1 presents an illustration of a node split process.

The node split is the main step during the construction of MAM, and, as stated previously, it occurs when there is a node overflow. It runs in main memory during the insertion of a new element. Thus, the smaller the complexity on the number of distances needed, the faster the method. Also, a strategy may advance or postpone other splits and certainly determines the overlap among nodes. An illustration of this issue can be seen in Figure 2 in which (b) presents an overlap region greater than (a).

The overlap degree of a MAM can be evaluated using two fat-factor measures defined in [12]: the relative fat-factor and the absolute fat-factor.

- *Absolute fat-factor*: it computes the amount of elements that are inside the intersected regions, which are defined by nodes in the same metric tree level. This measure is calculated as shown in Equation 1:

$$\text{fat}(T) = \frac{I_c - H.N}{N} . \frac{1}{M - H}, \tag{1}$$

  where T is a metric tree with height H, N data elements and M nodes, $M \geq 1$. I_c denotes the total number of node accesses required to answer a point query for each of the N elements stored in the metric tree. The value of the absolute fat-factor will always be in the range $[0, 1]$. The value zero indicates a tree with no overlapping and the value 1 implies a tree with all the nodes overlapped.
- *Relative fat-factor*: it allows the comparison of different MAM built with the same dataset. In order to do so, it takes into consideration the height and the total number of nodes of the minimum tree. Among all possible trees, the minimum tree is the one with the minimum height H_{min} possible and the minimum number of nodes M_{min}. Equation 2 presents the definition of this measure:

$$\text{rfat}(T) = \frac{I_c - H_{\text{min}}.N}{N} . \frac{1}{M_{\text{min}} - H_{\text{min}}}, \tag{2}$$

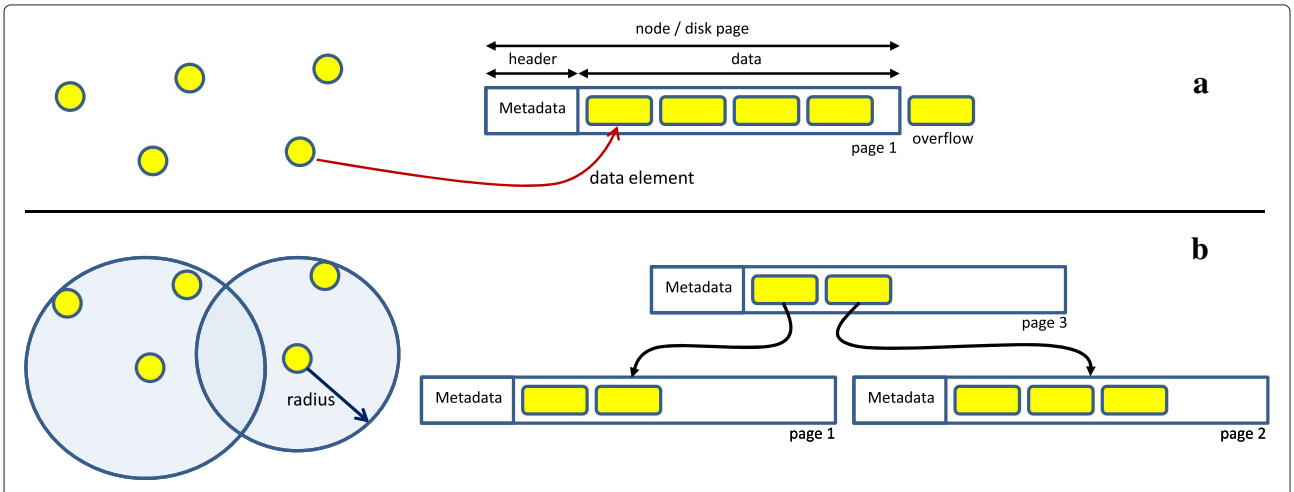

Figure 1 Illustration of a metric access method structure. (a) The insertion of an element in the root page caused overflow. **(b)** Root page was split and page 2 was created. One element of each node was promoted to the upper level (page 3) together with a covering radius.

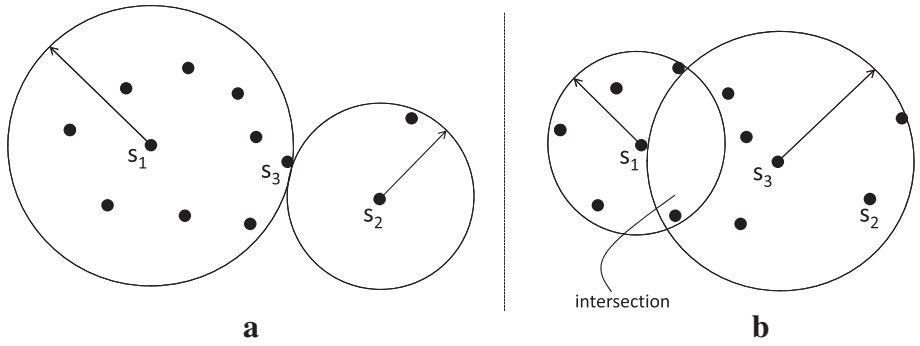

Figure 2 An illustration of different split policies and the resulting overlap among nodes. Both **(a)** and **(b)** present the same data elements.

where T is a metric tree, $H_{\min} = \lceil \log_c N \rceil$ denotes the minimum height of the tree and the minimum number of nodes $M_{\min} = \sum_{i=1}^{H_{\min}} \lceil N/C^i \rceil$, where C is the number of elements that can be stored in a node. The relative fat-factor value will always be greater than zero. The higher the value, the worse is the overlap among the nodes of the metric tree.

During a search, overlapped nodes may not be pruned, leading to the access of more branches of the structure. Thus, the development of strategies that minimize the overlap after node split during the constructions of a MAM is essential to fine-tune the structures and to allow scaling to huge datasets.

An ideal split policy must select two elements as representatives and their respective coverage radii, defining balls, partitioning the data elements into two sets, creating branches with the minimum possible overlap, aiming at efficient queries. A simple split policy (random) selects two random elements as the representatives and then distributes the node elements by minimizing the coverage radii up to the capacity of the nodes. It will be used in the experiments as a baseline.

M-tree MinMax is considered to produce the best set of nodes as it minimizes the node coverage radii. However, it does not guarantee to minimize the global overlap of the structure. It considers all possible pairs of the node elements as potential representatives. For each pair, it assigns the remaining elements of the node to one of the representatives. The pair which minimizes the covering radius is chosen.

Slim-tree MST split consists in separating the node elements in two clusters by removing one of the longest edges of a MST, followed by the selection of the central element in each cluster to be promoted as the node representative. It is important to note that MAM are index structures where each node is stored on a fixed-size disk page. Thus, an unequalized split may lead to a new split after few insertions.

Methods

We present new split policies for M-tree and Slim-tree that consider the distribution of the elements of a full node S into two new nodes, S' and S'', aiming at lowering the global overlap of the structure. We argue that maximizing the free space in both nodes created after a split postpones future splits. A node created after a split with few bytes left for newly promoted elements or new elements has higher probability of being split again, resulting in the increase of the global overlap. Our policies are based on known heuristics and may be employed by several MAM, such as M-tree and its descendants.

Maximum dissimilarity

This policy was proposed due to the hypothesis that it is possible to get smaller overlap by distributing the set of elements $\in S$ with regard to the pair of most dissimilar elements in this set. Thus, it starts by finding a pair of elements such as their distance is maximized argmax $d(s_1, s_2) \in S$. This phase of the algorithm is based on the pivot selection of [23]. The two far apart elements are obtained by randomly selecting an element $s_0 \in S$ (step 1) and finding the element s_1 that is the farthest from s_0 (step 2). Then, s_2 is selected as the farthest from s_1 (step 3). The elements s_1 and s_2 are reported as the desired pair of representative elements (step 4). Although the two middle steps (2 and 3) can be repeated a number of times, our experiments showed no significant increase in the resulting quality when additional steps are performed. Thus, in our experiments we kept three iterations.

The second phase of this policy divides the set S into two groups. The elements closer to s_1 are set to the first group and the elements closer to s_2 are set to the second group. The next phase chooses a central element to be promoted as the node representative. The maximum dissimilarity (MD) policy is depicted in Algorithm 1. Although this policy requires few linear scans on the node

elements, the execution time is $O(n^2)$ on the number of node elements due to the selection of the representatives. The strategy adopted by the MD policy is equivalent to creating a hyperplane that is equidistant to s_1 and s_2, separating the node elements into two groups, as shown in Figure 3.

Algorithm 1 Maximum dissimilarity policy

Require: dataset S
Find two far apart elements s_1 and s_2 in the border of S
Divide the node elements $\in S$ in two groups regarding their distances to s_1 and s_2
Select the medoid of each group as the representatives for S' and S'' respectively
Assign each node element $\in S$ to its proper node (S' or S'') according to the closest medoid
Report S' and S'' as the two new nodes

Path distance sum

This policy aims at avoiding the lack of balance on the number of elements after a MST split [12]. It is based on the sum of distances of a traversal of the MST based on Prim's algorithm to split the set. The idea is to remove the edge after achieving a threshold computed as half of the sum of distances from the whole traversal. The goal is to select two clusters such that the intra-group dissimilarity be minimized, resulting in smaller coverage radii while keeping balance on the number of elements in each cluster. Then the next step chooses a central element to be promoted as the node representative. Figure 4 presents the idea of the path distance sum (PDS) policy, and Algorithm 2 illustrates its main steps. The execution time of this policy is $O(n^2 \cdot \log n)$ on the number of node elements.

Algorithm 2 Path distance sum policy

Require: dataset S and threshold
Build the MST on the elements $\in S$
Divide the node elements $\in S$ in two groups
while the sum of distances in the MST traversal $\leq$ threshold **do**
Assign the elements to the first group
end while
Assign the remaining elements to the second group
Select the medoid of each group as the representatives for S' and S'' respectively
Assign each node element $\in S$ to its proper node (S' or S'') according to the closest medoid
Report S' and S'' as the two new nodes

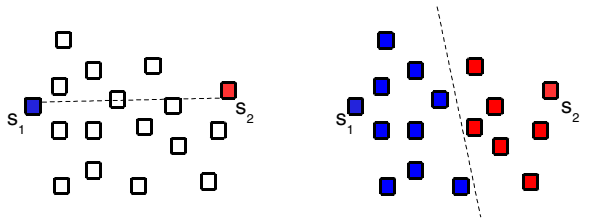

Figure 3 New split strategy where node is split based on a hyperplane. Elements s_1 and s_2 are the selected representatives. Blue and red elements represent different groups.

Reference element

This policy divides the node elements based on a reference element, by sorting the distances from each element to the reference. The goal is to use this ranked list to split the elements in two sets. The policy starts by selecting the farthest element (reference) from a randomly chosen element. The first set is composed of the $n/2$ elements closer to the reference, and the other elements are assigned to the second set. Then, it chooses two new representatives for S' and S'' as the medoids of two sets. After the representatives are selected, the other elements are assigned to their closer representatives. The main steps of the reference element (RE) policy are illustrated in Algorithm 3. The RE policy is equivalent to split the elements with an arc, as shown in Figure 5. Finding the reference element requires a linear scan on the node elements. However, due to the selection of the representatives, the execution time is $O(n^2)$ on the number of node elements.

Algorithm 3 Reference element policy

Require: dataset S and threshold
Select a reference element as the farthest element $s_1 \in S$ from a random element $s_0 \in S$
Build a ranked list R of the elements $\in S$ regarding their distances to s_1
Divide the node elements $\in R$ in two groups
while the amount of elements $\leq$ threshold **do**
Assign the elements to the first group
end while
Assign the remaining elements to the second group
Select the medoid of each group as the representatives for S' and S'' respectively
Assign each node element $\in S$ to its proper node (S' or S'') according to the closest medoid
Report S' and S'' as the two new nodes

Reference element$^+$

This policy is a variation of the RE policy. Unlike the RE policy, the reference element$^+$ (RE$^+$) tries to ensure a

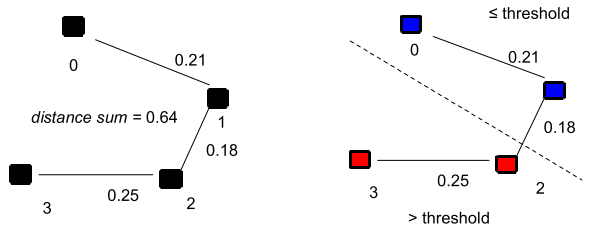

Figure 4 New split strategy. MST split based on the sum of distances obtained during traversal. Blue and red elements represent different groups.

more balanced distribution of the elements between the two nodes S' and S''. In order to do so, it changes the last assignment step of the original RE policy restricting it to the size of the two nodes. Each one of them must have up to half the cardinality of the original node S. Algorithm 4 presents the steps employed by the RE$^+$ policy. Its execution time is also $O(n^2)$ on the number of node elements.

Algorithm 4 Reference element$^+$ policy

Require: dataset S and threshold
  Select a reference element as the farthest element $s_1 \in S$ from a random element $s_0 \in S$
  Build a ranked list R of the elements $\in S$ regarding their distances to s_1
  Divide the node elements $\in R$ in two groups (S' and S'')
  while the amount of elements $\leq$ threshold **do**
    Assign the elements to the first group S'
  end while
  Assign the remaining elements to the second group S''
  Select the medoid of each group as the representatives for S' and S'' respectively
  Report S' and S'' as the two new nodes

Results and discussion

The three sets of experiments presented herein were designed with the intent to evaluate MAM built with the four node split policies regarding both querying and mining complex datasets. Our intention was to discuss node splitting policies that could be employed to build access methods for generic metric data spaces. This feature guided the selection of the baseline approaches employed in the experiments. The results obtained with our policies were compared with the former ones proposed for generic metric data spaces, which are MinMax, MST, and Random (see Table 1). All the policies evaluated were implemented within the same platform, using the C++ language into the Arboretum MAM library in order to obtain a fair comparison [24].

In order to cope with the querying issues, we analyzed the behavior of all the strategies considering the global tree overlap (see 'Analysis of the global tree overlap' section) and the impact on the index performance while running nearest neighbor queries (see 'Analysis of similarity queries' section). In addition to the efficiency evaluation of data clustering processes supported by MAM built with these new policies, we also evaluated their effectiveness (see 'Analysis of data clustering' section).

The experiments were run on an Intel Core i5 (3.2 GHz) with 4 GB of RAM, SATA hard disk of 250 GB (7,200 rpm) and GNU/Linux OS. All the strategies were implemented in C++ using the same framework to allow a fair comparison.

In order to validate the proposed strategies, we employed synthetic and real datasets. We created several synthetic datasets with Gaussian distribution, varying the number of elements from 100,000 to 500,000 and the number of dimensions from 8 to 128. These datasets will be referenced such as this example: S100E16D for the synthetic dataset with 100,000 elements composed of 16 dimensions. The trees were built based on the Euclidean metric (L_2) and with disk page size compatible with data dimensionality.

The real dataset employed is based on the COPhIR dataset [25] due to its size and availability. The dataset is composed of several features extracted from images. We employed the features extracted from 1 million images based on the *Scalable Color* extractor, composed of 64

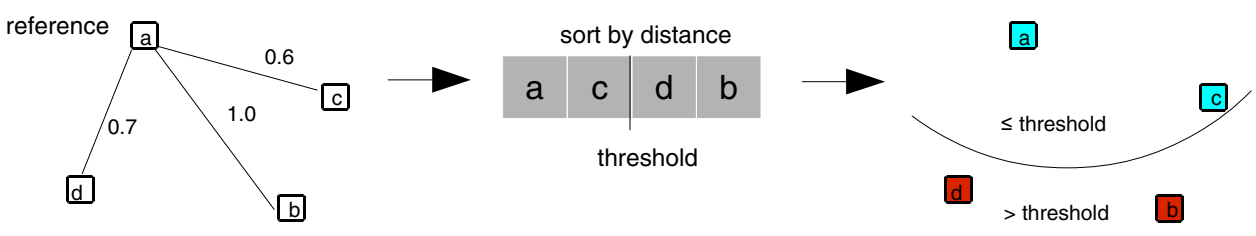

Figure 5 New split strategy. Split based on a reference element.

dimensions. This dataset order was randomly defined prior to the running of the experiments.

Analysis of the global tree overlap

In these experiments, we present the results for the three main parameters: the global overlap, the number of distance calculations, and the time spent to build the structures. The notion of volume is not available in generic metric spaces; thus, it is not possible to compute the volume of an intersection of balls. In order to verify the quality (i.e., the effectiveness) of the resulting indexes, we computed the absolute and relative overlap factors (fat-factors), following the guidelines and measurement definitions from [12] that were presented in 'Fundamental concepts' section. The efficiency was measured by the number of distance calculations and by the total time spent.

Instead of the volume, the fat-factor allows to evaluate the amount of elements that are inside intersected regions defined by nodes in the same level of a MAM. The lower the fat-factor value the lower the overlap. The absolute fat-factor is a measurement in the range $[0, 1]$, where zero indicates a structure without overlap and one indicates the worst possible structure. This measure takes into consideration the total number of nodes of the structure; thus, it is not suitable to compare the overlap of two structures built with different split strategies, as they may be built with different number of nodes. The comparison of trees that store the same elements can be done by the relative fat-factor, which is computed based on the minimum theoretical tree (a tree built with minimum height and minimum number of nodes).

The number of distance calculations is shown in Figure 6. Figure 7 presents the relative fat-factors and Figure 8 the build times of all trees for each split strategy.

In summary, we can draw the following observations from the graphs showed in Figures 6 and 7. When compared to MinMax, it is possible to verify that all the proposed policies required fewer distance computations in order to build the trees, although we observe a slight increase in the overlap factor for some of them. For example, analyzing the results presented in the graph of Figure 7b, we observe that the tree built with the MD policy for the *S500E16D* dataset presented an overlap factor 1.7% inferior requiring only 8.2% of the distance calculations performed by MinMax. Considering the *S100E16D* dataset, we observe that the tree built with the PDS policy required only 6.2% of the distance calculations performed by MinMax, whereas the increase in the overlap factor achieved 5.6%. The RE policy required only 6.2% of the distance calculations performed by MinMax, although the increase in the overlap factor reached 3.9%. The RE^+ policy also showed a gain in efficiency requiring only 5.2% of the distance calculations performed by MinMax, whereas the increase in the overlap factor reached 24.5%.

Particularly with respect to the behavior of the RE^+ policy when compared to the MinMax, it is important to note that as the amount of both data and dimensions of the dataset increases, the difference in the overlap factor for the trees built with these two policies decreases. For example, considering the *S500E128D* (Figures 6e and 7e)

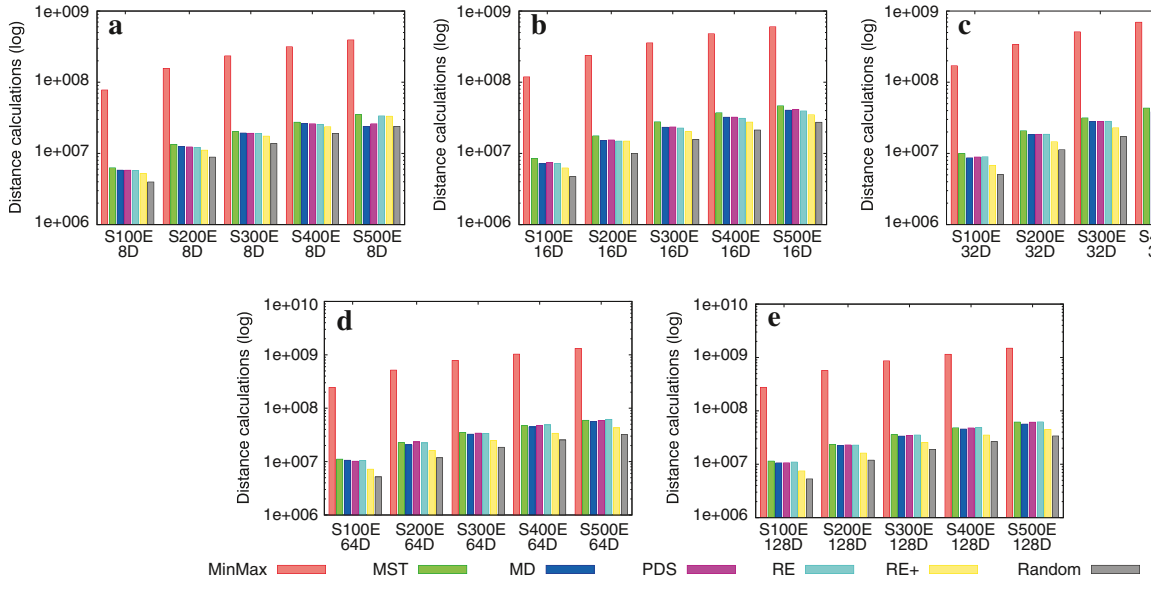

Figure 6 Distance calculations. (a-e) The respective numbers of distance calculations, in log scale. Graphs **(a)** 8 dimensions, **(b)** 16 dimensions, **(c)** 32 dimensions, **(d)** 64 dimensions, and **(e)** 128 dimensions.

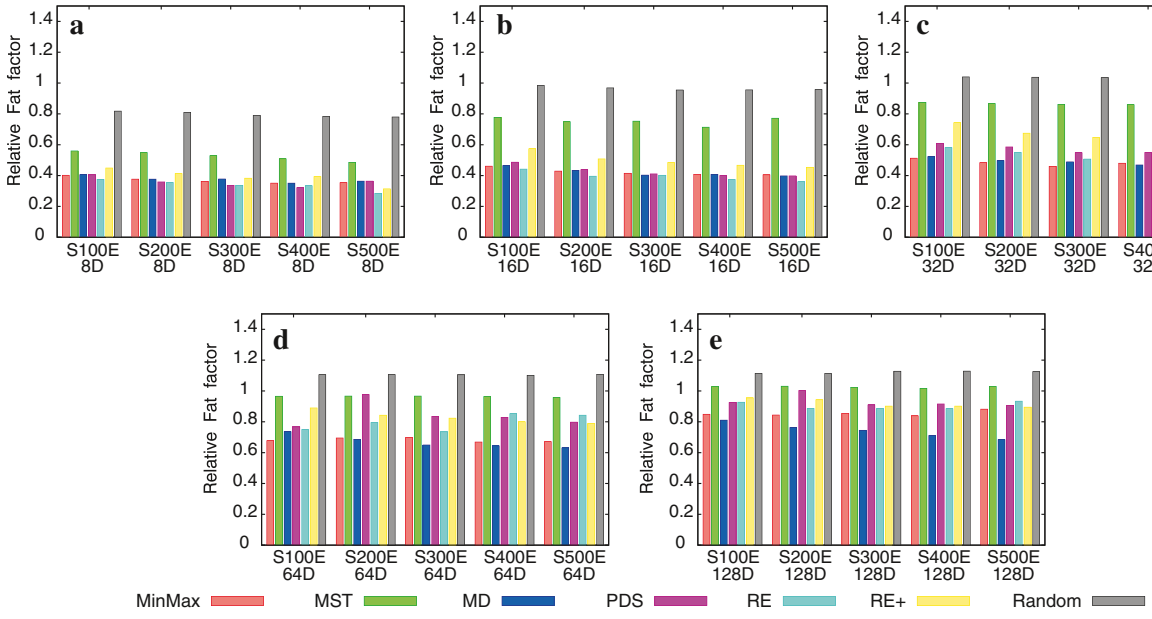

Figure 7 Overlap analysis. (a-e) The relative fat-factors for each split strategy. Graphs **(a)** 8 dimensions; **(b)** 16 dimensions, **(c)** 32 dimensions, **(d)** 64 dimensions, and **(e)** 128 dimensions.

dataset, the RE$^+$ policy showed an increase in the overlap factor of 1.5%, whereas it requires only 3% of the distance calculations performed by MinMax.

Another important remark is that, in general, the trees built with the MD policy were the ones that presented the lowest overlap factor. Moreover, for higher dimensional datasets, this policy outperforms the best classical policy (MinMax). Compared to MinMax, the decrease in the overlap factor was up to 22.4% for the 128 dimensions datasets requiring only 3.8% of the distance calculations

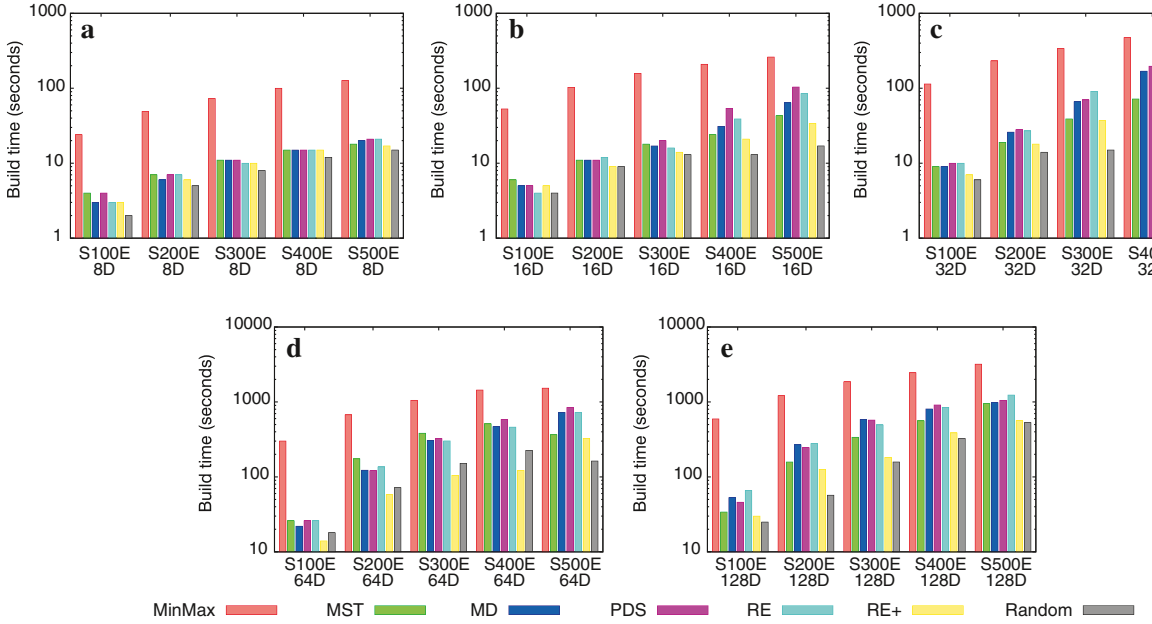

Figure 8 Build times for each tree. Graphs **(a)** 8 dimensions, **(b)** 16 dimensions, **(c)** 32 dimensions, **(d)** 64 dimensions, and **(e)** 128 dimensions. The horizontal axis presents the datasets ranging from 100,000 to 500,000 elements. The results in the time axis are in log scale.

performed by it. These results corroborate the statements made in the background section and also the assumptions made for the development of the work presented herein (see 'Related work' and 'Fundamental concepts' sections).

When compared to MST, the former policy that is considered as the one that led to the best trade-off between efficiency and effectiveness, it is possible to observe that all the trees built with the proposed policies presented a lower overlap factor. For example, for the MD, PDS, RE, and RE^+ policies, the overlap factor was reduced up to 48.4%, 48.1%, 50.7%, and 41.2%, respectively. Regarding the efficiency issue, the number of distance computations required by the proposed strategies was equivalent to the MST. It was observed a slight increase up to 19.9% for the MD strategy, 6.1% for the PDS strategy, and 8.4% for the RE strategy. On the other hand, the RE^+ policy was the one that needed less distance computations requiring only 64.9% of the distance calculations performed by MST for the *S100E64D* dataset (Figures 6d and 7d).

Table 2 shows the results obtained with the experiments performed with the COPhIR dataset. Analyzing the effectiveness issue, it is possible to verify that all the trees built with the proposed strategies presented a lower overlap factor when compared to MST and also that they required a number of distance calculations with the same order of magnitude. It is worth to note that when compared to MinMax, the tree built with the MD strategy presented a lower overlap factor and required much less distance computations. Observing the results achieved with all the experiments performed, it is possible to conclude that all the strategies presented the same behavior with both synthetic and real datasets.

Analysis of similarity queries

The graphs presented in this section show the behavior of the split policies for three main parameters regarding the analysis of the actual cost of similarity queries: the average number of distance calculations, the average number of disk accesses, and the average number of time to execute the queries. The results present the average of 100 k-nearest neighbor queries for each value of k using distinct sets of randomly selected query centers. This set of experiments was performed using the branch-and-bound algorithm of the k-nearest neighbor query defined in [26]. The values of k varied from 10 to 100 elements for the COPhIR dataset.

Table 2 Measures regarding the tree constructions for the COPhIR experiments

Strategy	Absolute fat-factor	Relative fat-factor	Distances	Time (s)
Random	0.126	0.212	67,239,671	1,077
MST	0.047	0.113	98,384,355	2,875
MinMax	0.046	0.078	1,581,570,838	3,272
MD	0.046	0.077	92,460,796	2,232
PDS	0.047	0.080	92,519,725	3,131
RE	0.041	0.069	92,289,401	3,087
RE^+	0.051	0.076	86,895,027	2,235

Figure 9 presents the results obtained for this set of experiments. Regarding the average number of disk accesses (Figure 9a), we can notice that, in general, the results related to the MST policy were the worst ones. This behavior is explained by the fact that the trees constructed with the MST result in a greater number of nodes. Analyzing this figure, it is also possible to verify that only the results obtained for the Random policy were worse than the ones achieved by this policy. As expected, the trees built with the Random policy usually present a high overlap factor, which led to the need to perform a large number of disk accesses and distance computations when performing queries. Therefore, the time spent to execute the queries is also high.

When examining all the parameters tested in conjunction, it is possible to conclude that the results obtained with all the policies proposed outperformed the ones achieved by both MST and MinMax. When compared to MinMax, if we observe the results obtained considering that k equals to 50, the tree built with the PDS policy required 4% fewer disk accesses and 12.7% fewer distance computations, the tree built with the RE policy required 12.2% fewer disk accesses and 15.2% fewer distance computations, the tree built with the MD policy required 2% fewer disk accesses and 4.3% fewer distance computations, and the tree built with the RE^+ policy required 2.9% fewer disk accesses and 3.2% more distance computations. Compared to MST, if we observe the results obtained considering that k equals to 80, the tree built with the PDS strategy required 22.5% fewer disk accesses and 14.9% fewer distance computations, the tree built with the RE strategy required 29% fewer disk accesses and 17.3% fewer distance computations, the tree built with the MD strategy required 20.9% fewer disk accesses and 6.7% fewer distance computations, and the tree built with the RE^+ policy required 21.8% fewer disk accesses and 0.2% more distance computations. Particularly with respect to the RE^+ policy, it is important to note that the gain in efficiency in disk access surpassed the need for a greater number of distance computations. This fact can be observed in Figure 9c that shows that the time required for the RE^+ policy to execute the queries was inferior than the ones required for both MST and MinMax.

Analysis of data clustering

In this set of experiments, we employed the PAM-SLIM algorithm with the aim at analyzing the impact on the behavior of data clustering processes when employing

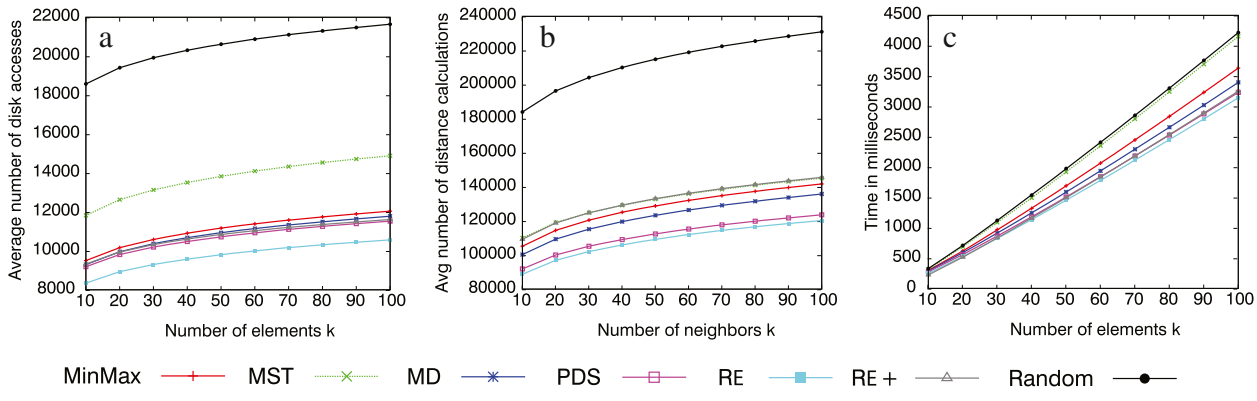

Figure 9 Nearest neighbor queries, COPhIR dataset with 1 million elements. (a) Disk accesses. **(b)** Distance calculations. **(c)** Total time.

different node split policies on the construction of the MAM that supports the clustering algorithm. To evaluate the effectiveness of the clusters obtained by the PAM-SLIM algorithm executed with different MAM configurations, we computed the average distance of the resulting clustering, i.e., the average distance of all elements from their medoids (smaller values for average distance indicate better clustering). This is the standard measure employed in the scientific literature in order to evaluate the quality of the clustering obtained by the *k*-medoid-based algorithms. The efficiency was measured by the number of distance computations. It is worth noting that the efficiency graphs are in log scale for the number of distance calculation axis.

Figures 10, 11, 12 present the effectiveness and efficiency results obtained when applying the PAM-SLIM algorithm with different MAM configurations for the following three datasets: S300E64D, S500E64D, and a subset of the COPhIR dataset with 500,000 elements, respectively. The values for *k* (number of clusters asked) varied from 5 up to 20 and are presented in the *x*-axis of the graphics.

Considering the results presented in Figure 10, the RE$^+$ policy was the one that presented the best trade-off between efficiency and effectiveness. If we observe only the graph related to the clustering quality (Figure 10a), it is possible to verify that besides the Random and the MST policies, the quality of the clusters obtained from the PAM-SLIM algorithm, considering the other node-split policies, was practically the same. However, if we observe the correspondent graphs of efficiency (Figure 10b), the PAM-SLIM algorithm execution with the RE$^+$ policy was the one that presented the smallest number of distance computations, in general. For example, considering the results obtained when *k* was 15, the RE$^+$ policy showed 90.1% less distance computations than the PDS policy, 89.3% less distance computations than the MST policy, 88.7% less distance computations than the RE policy, 85.7% less distance computations than the MinMax policy, 83.6% less distance computations than the MD policy, and 65.7% less distance computations than the Random policy.

Similar results were obtained for the S500E64D (see Figure 11). The RE$^+$ policy was again the one that presented the best trade-off between efficiency and effectiveness. Except the Random and the MST policies, the effectiveness of the clusters obtained from the PAM-SLIM

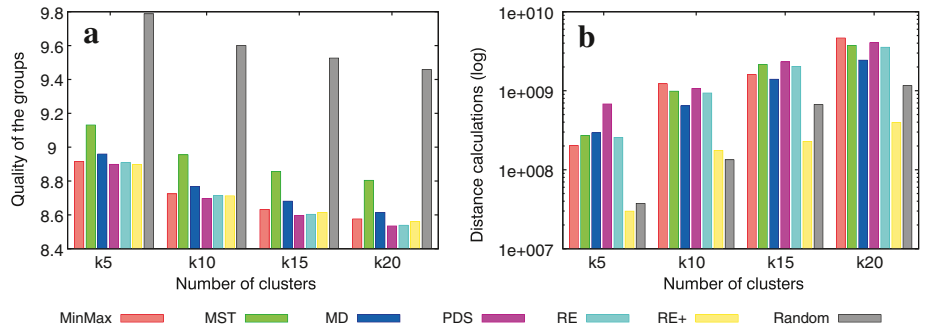

Figure 10 Graphs of the analysis of data clustering for the $S300E64D$. (a) Quality of the groups. **(b)** Distance calculations (the results in the *y*-axis are in log scale).

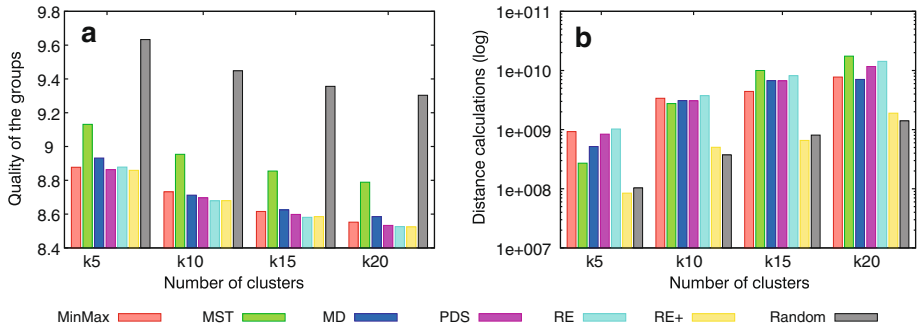

Figure 11 Graphs of the analysis of data clustering for the $S500E64D$. (a) Quality of the groups. **(b)** Distance calculations (the results in the *y*-axis are in log scale).

algorithm, considering the other node-split policies was equivalent (see Figure 11a). In general, the smallest values for the number of distance computations were obtained by the RE$^+$ policy (see Figure 11b). Considering the results obtained when k was 15, this policy showed 93.4% less distance computations than the MST policy, 91.9% less distance computations than the RE policy, 90.2% less distance computations than the MD policy, 90.1% less distance computations than the PDS policy, 85.2% less distance computations than the MinMax policy and 18.8% less distance computations than the Random policy.

When examining the results obtained with a real dataset (see Figure 12), it is possible to verify that the number of distance computations performed by the RE$^+$ policy remained with the same order of magnitude, or even less when compared with other policies. Among the newly proposed node-split policies, this policy showed a higher number of distance computations with respect to the MD and the RE policies just when k was set to 5. Compared to the former policies, the RE$^+$ policy presented a smaller number of distance computations for all values of k when compared to MST and also smaller numbers of distance computations for large values of k (15 and 20) when compared to MinMax. For example, when k was set to 15, this policy required 68.5% and 35.1% less distance computations than MST and MinMax, respectively. Regarding the clustering quality, except from the Random policy, all the other policies presented almost the same effectiveness.

Conclusions

When thinking of supporting both querying and mining large complex datasets in an integrated environment, it is important to provide means not only to built efficient MAM but also MAM that do not degrade the effectiveness of mining processes. The four node split policies presented herein, for M-tree and Slim-tree, meet these requirements.

Our policies are guided by the observation that a node created after a split with few bytes left for newly promoted elements or new elements has higher probability of being split again, resulting in the increase of the overlap among the nodes. The experiments show that the new split policies build more efficient trees in less time. Considering the COPhIR dataset, for example, when compared to MinMax, the RE policy required 6% less build time and 15.2% fewer distance computations to process 50 nearest neighbor queries.

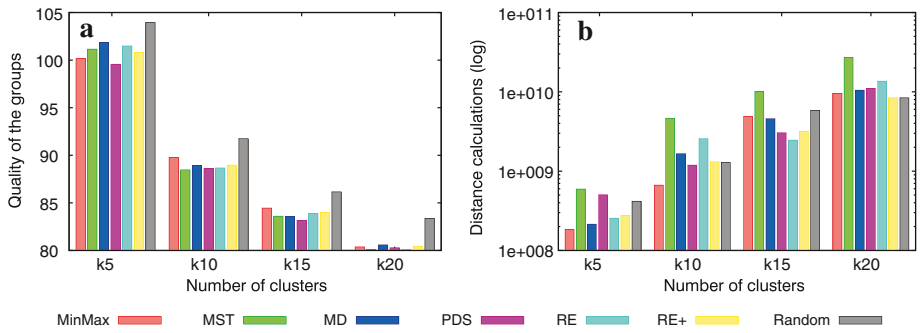

Figure 12 Graphs of the analysis of data clustering for the COPhIR dataset. (a) Quality of the groups. **(b)** Distance calculations (the results in the *y*-axis are in log scale).

Even though reducing the time spent to build the trees, the lower resultant fat-factor may benefit query execution by reducing the number of distance calculations and disk accesses to run, for instance, nearest neighbor queries. In addition to the gain in efficiency also observed for data clustering, the effectiveness of the resulting clustering remained the same for all the evaluated policies. Among future works we intend to combine the proposed policies with dynamic reinsertion and compare with the other optimization strategies for efficient MAM construction mentioned in this article.

Competing interests
The authors declare that they have no competing interests.

Authors' contributions
JAS, HLR, and MCNB participated in the definition of the split policies and in the design of the experiments. JAS was responsible of coding and of executing the experiments. All authors helped to draft the manuscript and also read and approved its final version.

Authors' information
JAS received her B.Sc. degree in Information Systems from the Universidade Federal de Grande Dourados, Brazil in 2010 and the M.Sc. degree in Computer Science from the Universidade Federal do ABC, Brazil in 2012. She is currently a Ph.D. candidate in Computer Science from the Universidade de São Paulo at São Carlos, Brazil. Her research interests include multimedia data mining, indexing structures, and semi-supervised clustering. HLR received his B.Sc. degree in Computer Science from the Universidade Federal do Mato Grosso, Brazil in 2000 and M.Sc. and Ph.D. in Computer Science in 2004 and 2009 at Universidade de São Paulo at Sao Carlos, Brazil. He is currently an assistant professor in the Department of Computing at the Universidade Federal de Uberlândia. His research interests include access methods for complex data, similarity searching, multimedia databases, and information visualization. MCNB received her B.Sc. degree in Computer Science from the Federal University of Uberlandia, Brazil in 2000 and M.Sc. and Ph.D. in Computer Science in 2002 and 2006 at Universidade de São Paulo at Sao Carlos, Brazil. She is currently an assistant professor in the Department of Computing at the Universidade Federal de Uberlândia. Her research interests include multimedia databases, multimedia data mining, indexing methods for multidimensional data, and information visualization.

Acknowledgements
This work has been supported by CNPq (Conselho Nacional de Desenvolvimento Científico e Tecnológico) under grant Universal 479930/2011-2, by FAPEMIG (Fundação de Amparo à Pesquisa de Minas Gerais) under grant CEX-APQ-01290-12, by FAPESP (Fundação de Amparo à Pesquisa do Estado de São Paulo) under grant 2011/16067-1, by CAPES (Coordenação de Aperfeiçoamento de Pessoal de Nível Superior), by NIC.br (Núcleo de Informação do Ponto BR) and by PROPP/UFU (Pró-Reitoria de Pesquisa e Pós-Graduação/Universidade Federal de Uberlândia).

Author details
[1]Instituto de Ciências Matemáticas e de Computação, USP, Trabalhador Saocarlense, 400, São Carlos, Sao Paulo, Brazil. [2]Faculdade de Computação, Universidade Federal de Uberlândia, Av. João Naves de Avila, 2121, Uberlândia, Minas Gerais, Brazil.

Received: 27 January 2014 Accepted: 26 August 2014
Published online: 13 September 2014

References
1. Zaki MJ, Meira W Jr (2014) Data mining and analysis - fundamental concepts and algorithms. 1st edn. Cambridge University Press, New York
2. Han J, Kamber M, Pei J (2011) Data mining: concepts and techniques. 3rd edn. Morgan Kaufmann, San Diego
3. Silva YN, Aref WG, Larson P-Å, Pearson S, Ali MH (2013) Similarity queries: their conceptual evaluation, transformations, and processing. VLDB J 22(3):395–420. doi:10.1007/s00778-012-0296-4
4. Kriegel H-P, Kröger P, Renz M, Schubert M (2010) Metric spaces in data mining: applications to clustering. SIGSPATIAL Special 2(2):36–39. doi:10.1145/1862413.1862423
5. Barioni MCN, Kaster DS, Razente HL, Traina AJM, Traina-Jr. C (2010) Querying multimedia data by similarity in relational DBMS. In: Yan L, Ma Z (eds) Advanced database query systems: techniques, applications and technologies. IGI Global, Hershey, pp 323–359. doi:10.4018/978-1-60960-475-2.ch014
6. Papadopoulos AN, Manolopoulos Y (2005) Nearest neighbor search: a database perspective. Series in computer science. Springer, Heidelberg. p 170
7. Silva YN, Aref WG, Ali MH (2010) The similarity join database operator. In: International conference on data engineering (ICDE). IEEE, Long Beach, pp 892–903. doi:10.1109/ICDE.2010.5447873
8. Vieira MR, Razente HL, Barioni MCN, Hadjieleftheriou M, Srivastava D, Traina C, Tsotras VJ (2011) On query result diversification. In: International conference on data engineering (ICDE). IEEE, Hanover, pp 1163–1174. doi:10.1109/ICDE.2011.5767846
9. Samet H (2006) Foundations of multidimensional and metric data structures. Morgan Kaufmann, San Francisco
10. Ciaccia P, Patella M, Zezula P (1997) M-tree: an efficient access method for similarity search in metric spaces. In: International conference on very large data bases (VLDB). Morgan Kaufmann, Athens, pp 426–435
11. Zezula P, Amato G, Dohnal V, Batko M (2006) Similarity search: the metric space approach. Advances in database systems, vol. 32. Springer, Heidelberg
12. Traina C Jr, Traina AJM, Faloutsos C, Seeger B (2002) Fast indexing and visualization of metric datasets using Slim-trees. IEEE Trans Knowl Data Eng 14(2):244–260. doi:10.1109/69.991715
13. de Souza JA, Razente HL, Barioni MCN (2013) Faster construction of ball-partitioning-based metric access methods. In: Symposium on applied computing (SAC). ACM, Coimbra, pp 8–12. doi:10.1145/2480362.2480365
14. Lim S-H, Ku K-I, Kim K, Kim Y-S (2006) A node split algorithm reducing overlapped index spaces in m-tree index. In: International conference on data engineering workshops (ICDEW). IEEE, Atlanta, pp 15–23. doi:10.1109/ICDEW.2006.14
15. Aronovich L, Spiegler I (2007) CM-tree: a dynamic clustered index for similarity search in metric databases. Data Knowl Eng 63(3):919–946. doi:10.1016/j.datak.2007.06.001
16. Skopal T, Lokoc J (2009) New dynamic construction techniques for M-tree. J Discrete Algorithm 7(1):62–77. doi:10.1016/j.jda.2008.09.013
17. Vespa TG, Traina C Jr, Traina AJ (2010) Efficient bulk-loading on dynamic metric access methods. Inf Syst 35(5):557–569. doi:10.1016/j.is.2009.07.002
18. Navarro G, Uribe-Paredes R (2011) Fully dynamic metric access methods based on hyperplane partitioning. Information Systems 36(4):734–747. doi:10.1016/j.is.2011.01.002
19. Ng RT, Han J (1994) Efficient and effective clustering methods for spatial data mining. In: 20th International conference on very large data bases (VLDB). Morgan Kaufmann, Santiago, pp 144–155
20. Ester M, Kriegel H-P, Xu X (1995) Knowledge discovery in large spatial databases: focusing techniques for efficient class identification. In: International symposium on advances in spatial databases. Lecture notes in computer science, vol. 951. Springer, Portland, pp 67–82. doi:10.1007/3-540-60159-7_5
21. Barioni MCN, Razente H, Traina AJM, Traina C Jr (2008) Accelerating k-medoid-based algorithms through metric access methods. J Syst Software 81(3):343–355. doi:10.1016/j.jss.2007.06.019
22. Beckmann N, Kriegel H-P, Schneider R, 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. ACM, Atlantic City, pp 322–331. doi:10.1145/93597.98741
23. Faloutsos C, Lin K-I (1995) Fastmap: a fast algorithm for indexing, data-mining and visualization of traditional and multimedia datasets. In: ACM international conference on management of data (SIGMOD). ACM, San Jose, pp 163–174. doi:10.1145/568271.223812
24. GBDI-ICMC-USP (2014) GBDI arboretum library. Available on [http://www.gbdi.icmc.usp.br/downloads/arboretum/]. Accessed 01 September 2014

25. Bolettieri P, Esuli A, Falchi F, Lucchese C, Perego R, Piccioli T, Rabitti F (2009) CoPhIR: a test collection for content-based image retrieval. CoRR. abs/0905.4627v2
26. Roussopoulos N, Kelley S, Vincent F (1995) Nearest neighbor queries. In: ACM SIGMOD international conference on management of data. ACM, San Jose, pp 71–79. doi:10.1145/223784.223794

doi:10.1186/s13173-014-0017-5
Cite this article as: Souza *et al.*: Optimizing metric access methods for querying and mining complex data types. *Journal of the Brazilian Computer Society* 2014 **20**:17.