

# Improving the Pruning Ability of Dynamic Metric Access Methods with Local Additional Pivots and Anticipation of Information

Paulo H. Oliveira<sup>1</sup>(✉), Caetano Traina Jr.<sup>2</sup>, and Daniel S. Kaster<sup>1</sup>

<sup>1</sup> Department of Computer Science, University of Londrina (UEL), Londrina, Brazil  
oliveiraph17@gmail.com, dskaster@uel.br

<sup>2</sup> Institute of Mathematics and Computer Science, University of São Paulo (USP),  
São Paulo, Brazil  
caetano@icmc.usp.br

**Abstract.** Metric Access Methods (MAMs) have been proved to allow performing similarity queries over complex data more efficiently than other access methods. They can be considered dynamic or static depending on the pivot type used in their construction. Global pivots tend to compromise the dynamicity of MAMs, as eventual pivot-related updates must be propagated through the entire structure, while local pivots allow this maintenance to occur locally. Several applications handle online complex data and, consequently, demand efficient dynamic indexes to be successful. In this context, this work presents two techniques for improving the pruning ability of dynamic MAMs: (i) using cutting local additional pivots to reduce distance calculations and (ii) anticipating information from child nodes to reduce unnecessary disk accesses. The experiments reveal significant improvements in a dynamic MAM, reducing execution time in more than 50 % for similarity queries posed on datasets ranging from moderate to high dimensionality and cardinality.

**Keywords:** Similarity queries · Metric access methods · Cutting local additional pivots · Anticipation of child information

## 1 Introduction

In recent years, it has been noticed a fast-growing volume of complex data. Multimedia data, georeferenced data and time series are examples of such data. Some reasons for the growth are: lower prices of digital cameras and other video capture devices, high-definition cameras embedded in mobile phones, user-friendly tools for processing and editing images and videos, acquisition of data from medical equipment, data capture through sensor networks and high-speed internet connections. The success of multimedia sharing services such as YouTube, Flickr and social media is another evidence of this growth.

---

This research has been supported by scholarship grants from the Brazilian Coordination for the Improvement of Higher Education Personnel (CAPES).

In this work, complex data are considered as information that are not represented by traditional types, such as numbers, characters, dates and short texts. A key observation is that the *order relation* does not apply to most complex domains [8]. The order relation is a property that allows identifying which element precedes the other, according to some criterion, in each pair of elements of the domain. Since traditional index structures are based on this property, they are not suitable for complex data. Nevertheless, there are structures well-suited for complex domains, such as the Metric Access Methods (MAMs).

There are several MAMs related in the literature, categorized in different ways depending on which factors are taken into account to structure the data. The factors *pivot type* and *structure dynamicity* are directly related to each other. Pivots are elements that act as representatives of certain regions of the search space and are used to prune irrelevant elements during the query execution. It is said that a pivot is global when all elements of the dataset are referenced to it, whereas a pivot is local when only a portion of the dataset is referenced to it. Because global pivots are referenced by the whole dataset, they have a high impact in the pruning process of irrelevant elements, once that a single global pivot can be used to discard a large amount of irrelevant elements. However, MAMs based on global pivots may have their dynamicity compromised by the fact that eventual pivot-related updates need to be propagated through the entire structure. Local pivots, on the other hand, allow the maintenance to occur locally at the price of a lower pruning ability. In this context, the challenge addressed in this work is to improve the pruning ability of dynamic MAMs without harming their dynamicity.

This paper presents two new techniques that significantly improve the performance in similarity queries of dynamic MAMs based on local pivots. The first technique is to employ local additional pivots to reduce the uncertainty area in the search space, i.e. the area that may contain elements that are not part of the answer but cannot be pruned without being analyzed. The second technique is to anticipate information from child nodes to their parents to enable pruning irrelevant elements before visiting the disk pages that actually store them. Differently from other approaches regarding multiple pivots to define a space region, our proposal allows reducing both the number of distance calculations and the number of disk accesses as well as it does not impose any constraint in the index dynamicity.

The new techniques presented in the paper have been applied to the dynamic MAM Slim-tree [15] and evaluated through an extensive set of experiments over real datasets, varying the number of elements, the dimensionality and employing distance functions with different computational costs. In the paper, we present results that confirm their efficiency, as they enabled gains of more than 50% in execution time, number of distance calculations and number of disk accesses when compared to the original structure, regarding every evaluated dataset.

The paper is organized as follows: Sect. 2 covers essential concepts regarding similarity queries over complex data and MAMs, as well as it presents the related work; Sects. 3 and 4 describe the two new proposed techniques; Sect. 5 presents how the techniques were applied to Slim-tree; Sect. 6 describes the experiments and discusses the results; and Sect. 7 presents the conclusions and future work.

## 2 Background and Related Work

### 2.1 Similarity Queries

In order to allow performing queries over complex domains, the elements of a given complex dataset usually have features extracted from their content. The extracted features are used in place of the original data to execute the queries. The retrieval of complex data based on this process is known as *content-based retrieval* and the set of extracted features of an element is called its *feature vector* or *signature*. Feature vectors can be, using images for example, shape and texture attributes, color histograms and results from transformations applied to data [8]. Usually, complex data are compared by dissimilarity relations between pairs of feature vectors. This is performed by employing a *distance function* that calculates how dissimilar are the two feature vectors from each other. Those are known as *similarity queries*, as they retrieve the elements from the dataset that satisfy a given similarity-based criterion.

There are several types of similarity queries [18], which range from similarity selections and joins to aggregate similarity queries. The two most common types are the Range and the  $k$ -Nearest Neighbors queries. Given a maximum threshold  $\xi$ , the *Range query* (Rq) retrieves every tuple  $t_i$  from relation  $R$  that has a value  $s_i$  for attribute  $S_j$ , which represents the feature vector of the element, satisfying the condition  $\delta(s_i, s_q) \leq \xi$ , where  $s_q$  is the value of the query element for attribute  $S_j$  and  $\delta$  is the distance function that returns the dissimilarity between  $s_i$  and  $s_q$ . Considering  $R$  as a relation of images, an example of Range query is: “Select the images which are similar to the image Q by up to 5 units”. Given an integer value  $k \geq 1$ , the  *$k$ -Nearest Neighbor query* ( $k$ -NNq) retrieves  $k$  tuples  $t_i$  from relation  $R$  with values  $s_i$  for attribute  $S_j$  that have the lowest distance from  $s_q$  according to  $\delta$ . An example of  $k$ -Nearest Neighbors query is: “Select the 4 most similar images to the image Q”.

### 2.2 Metric Access Methods

There are two main categories of access methods for indexing complex data. The first category is a class of access methods that support data domains represented in dimensional spaces, especially spatial data, known as Spatial Access Methods (SAMs) [9]. However, many complex data types are high-dimensional and SAMs degrade quickly as the number of dimensions grows. Furthermore, some complex data types are *dimensionless*, that is, they cannot be represented by coordinates in orthogonal axes. The second category is given by the Metric Access Methods (MAMs). MAMs rely on the premise that data are immersed in a metric space. A metric space is defined by a pair  $\langle \mathbb{S}, \delta \rangle$ , where  $\mathbb{S}$  is a complex data domain (i.e. feature vectors) and  $\delta$  is a distance function  $\delta : \mathbb{S} \times \mathbb{S} \mapsto \mathbb{R}^+$ , known as *metric*. A metric has properties called *metric postulates* [18]  $\forall x, y, z \in \mathbb{S}$ : (i)  $\delta(x, y) \geq 0$  (non-negativity), (ii)  $\delta(x, y) = \delta(y, x)$  (symmetry), (iii)  $x = y \iff \delta(x, y) = 0$  (identity) and (iv)  $\delta(x, z) \leq \delta(x, y) + \delta(y, z)$  (triangular inequality). An important characteristic of metric spaces is that, in addition to comprehending vector spaces, they include dimensionless spaces. Therefore, almost any data type

can be immersed into a metric space, including geographic coordinates, images, sounds, words and DNA sequences.

The general idea of most MAMs is to choose some elements as representatives of certain subsets of data. Such elements are also known as pivots. Whenever an element  $s_i \in \mathbb{S}$  is inserted, the distance from its representative is calculated and stored in the structure. Afterwards, these distance values are used in similarity queries for pruning elements by using the triangular inequality property. More formally, an element can be pruned, i.e. discarded without calculating its distance from the query element, when one of the following conditions [3] is true, where  $s_{rep}$  is the representative of the subset,  $s_q$  is the query element,  $s_i$  is any of the remaining elements in the subset and  $r_i$  is the covering radius of  $s_i$ :

$$\delta(s_{rep}, s_i) + r_i < \delta(s_{rep}, s_q) - \xi \quad (1)$$

$$\delta(s_{rep}, s_i) - r_i > \delta(s_{rep}, s_q) + \xi \quad (2)$$

### 2.3 Related Work

The problem addressed in this work is about improving the pruning ability of dynamic MAMs without compromising their dynamicity. MAMs which rely on global pivots, such as the OMNI-family [14] and the permutation-based approximate index PP-Index [7], allow updates in the structure and thus are classified as dynamic. However, if it is necessary to perform a pivot-related update, the cost can be close to rebuilding the whole structure. Therefore, we consider truly dynamic the MAMs in which updates are managed locally, being at most propagated through the path between a descendant and its highest ancestor.

In this sense, the pioneer dynamic MAM is M-tree [5], a balanced hierarchic MAM based on local pivots, with ball partitioning and bottom-up construction. Due to the success of M-tree, several access methods sharing similar principles have been proposed with the goal of achieving a better performance, but keeping their structure dynamic. One of them is the Slim-tree [15], an evolution of M-tree with improvements such as the evaluation and minimization of the overlap level between nodes and a new split algorithm. Another example is the DBM-tree [17], which allows a controlled unbalance to better fit the dataset density variations. These structures have a single local pivot per node. There are MAMs that employ multiple pivots per node, such as MM-tree [11] and Onion-tree [4]. However, the primary goal of these structures is to index data in main memory avoiding overlaps among nodes, being subject to end up highly unbalanced after updates. Other example is the M\*-tree [12], a variation of M-tree that stores, for every node, a nearest-neighbors graph containing the nearest neighbor of each element in the node and the distance between them. This turns every element of a node into a kind of local pivot, allowing saving distance calculations.

Another strategy employed in some works is to include additional global pivots in a local pivot-based MAM. An example of a method using this strategy is the DF-tree [16]. Its structure is similar to the Slim-tree, but it uses global pivots in the pruning process, embodying the idea of global pivots of the OMNI-family.

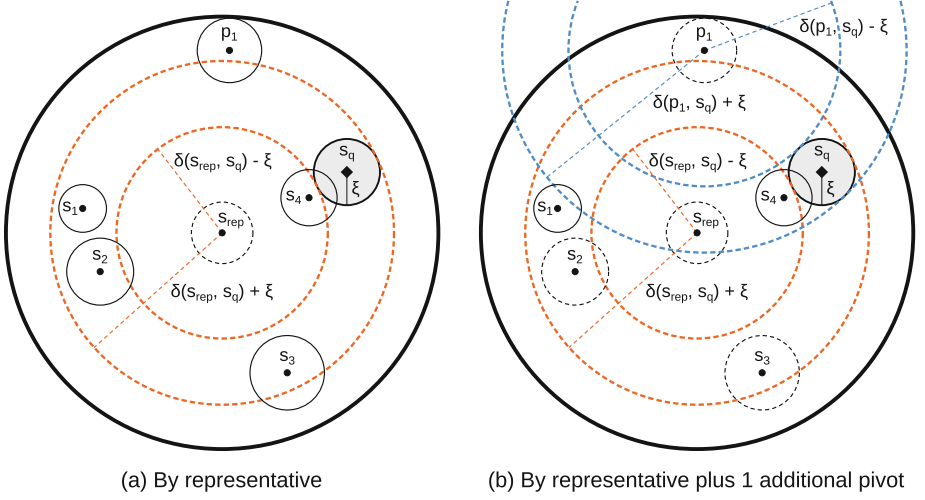
Other examples are PM-tree (*Pivoting M-tree*) [13] and PM\*-tree [12]. The PM-tree is a ball-partitioned structure which restricts the uncertainty region by the intersection of hyper-rings defined by global pivots. The PM\*-tree adds multiple local pivots to the PM-tree in the way M\*-tree does. The main advantage of these MAMs is that the number of distance calculations is significantly reduced due to the improved pruning ability provided by global pivots. However, pivot-related changes require rebuilding the whole index.

Our proposal differs from the existing ones as we include local additional pivots in local pivot-based MAMs and anticipate information from child nodes in such a way that it improves the pruning ability of the structure while maintaining every update locally contained. The first technique consists of adding local pivots to each node and having the distances of their elements from each additional pivot calculated and stored. The target for this technique is to reduce the uncertainty region and, consequently, the number of distance calculations by using the triangular inequality property in each node accessed by a query. The second technique consists of anticipating information from child nodes, such as distance and radius values, and use them to avoid unnecessary disk accesses. Although M\*-tree employs local additional pivots in each node too, the benefits obtained are only in terms of distance calculations. Our techniques, on the other hand, allow reducing both distance calculations and disk accesses. They are presented in the next sections.

### 3 The CLAP Technique

This section describes our new technique to improve the pruning ability of dynamic MAMs, named *Cutting Local Additional Pivots* (CLAP). It aims at reducing the uncertainty region of each node accessed in a similarity query. However, distinctly from other strategies that employ multiple local pivots, in which the pivots define the covering region of a node, as in the MM-tree and the Onion-tree, the CLAP technique is used to cut the region defined by a node representative applying the triangular inequality pruning mechanism using local additional pivots. In spite of changing the node structure, this technique does not change its partitioning, i.e. the covering radius of each node is still defined from  $s_{rep}$  and therefore it does not compromise the dynamicity of the structure.

As described in Sect. 2.2, pruning by the triangular inequality regarding the representative, known as the *main pivot*, consists of discarding every element which satisfies one of the Eqs. 1 and 2. Figure 1a illustrates this property, where the child node whose representative is also  $s_{rep}$  (the black dashed circumference, centered in  $s_{rep}$ ) is pruned, as it does not touch the uncertainty region (the orange dashed ring). What the CLAP technique allows is to extend the use of this property for all local additional pivots. Thus, for every element  $s_i$  which has not been pruned by the triangular inequality involving the main pivot, each additional pivot  $p_j$ ,  $1 \leq j \leq n$ , where  $n$  is the number of additional pivots, is used to prune the elements that satisfy the following additional conditions:



**Fig. 1.** Pruning by triangular inequality with 1 additional pivot. Dashed circumferences centered in  $s_{rep}$ ,  $s_2$ ,  $s_3$  and  $p_1$  represent pruned elements

$$\delta(p_j, s_i) + r_i < \delta(p_j, s_q) - \xi \quad (3)$$

$$\delta(p_j, s_i) - r_i > \delta(p_j, s_q) + \xi \quad (4)$$

Similarly to In Eqs. 1 and 2, these conditions involve two distance calculations. One of them, the value  $\delta(p_j, s_i)$ , was calculated and stored into the node when the MAM was built. The value  $\delta(p_j, s_q)$ , even though it must be calculated for each pivot  $p_j$  when the node is visited, allows reducing even more the number of distance calculations. Figure 1b shows the new uncertainty region as the intersection of two rings: the orange dashed one, centered in the representative  $s_{rep}$ , and the blue dashed one, centered in the cutting local additional pivot  $p_1$  (this figure considers one cutting local additional pivot). It can be verified that the uncertainty region is much smaller than that shown in Fig. 1a, pruning the child nodes which have  $s_{rep}$ ,  $s_2$ ,  $s_3$  and  $p_1$  as their representatives.

## 4 Anticipation of Child Information

This section presents a new approach which aims at avoiding unnecessary disk accesses by modifying the structure of nodes in MAMs in order to get, in advance, information from child nodes that would only be accessed when these nodes were read from disk. Taking the Slim-tree for example, when a node is visited during a query, the first step is to use the triangular inequality pruning mechanism. For every element  $s_i$  not pruned in an index node, the corresponding node must be accessed from disk even if none of its elements touches the search region.

Our second proposed technique is ACIR (*Anticipation of Child Information regarding Representatives*), which consists of anticipating, for each child node of the current node, the array of distances from the representative  $s_i$  plus the array of covering radii (only the distances when the child nodes are leaf nodes, since their entries do not have radii). Consequently, the triangular inequality pruning mechanism regarding the representative is anticipated for each child node which intercepts the search region. With the ACIR technique, the sequence of steps for each node accessed during a similarity query is: (i) to use the triangular inequality pruning mechanism by the main pivot and by the additional pivots; (ii) to calculate the distances between  $s_q$  and all elements not pruned in the previous step; (iii) before reading from disk the elements which intercept the search region defined by  $s_q$ , execute additional steps involving the information anticipated from child nodes. In doing so, it is possible to avoid unnecessary disk accesses by identifying nodes that intercept the search region and, nevertheless, are irrelevant for the result.

This allows avoiding unnecessary disk accesses in situations like the one depicted in Fig. 2, where the child node centered in  $s_1$  intercepts the search region defined by  $s_q$ , but none of its children —  $s_{11}$ ,  $s_{12}$ ,  $s_{13}$  and  $s_{14}$ , which are grandchild nodes of the current node (the biggest one, centered in  $s_{rep}$ ) — intercepts the uncertainty region represented by the dashed ring within the child node centered in  $s_1$ . The evaluation of whether the grandchild nodes intercept or not this uncertainty region can happen in the current level only because their distances and radii have been anticipated. If they had not, the child node centered in  $s_1$  would have to be read from disk (unnecessarily) for this evaluation to be done. Still in Fig. 2, the small circumferences centered in  $s_{11}$ ,  $s_{12}$ ,  $s_{13}$  and  $s_{14}$  are the covering radii of the grandchild nodes. Those regions appear in the figure just to illustrate the anticipated use of the triangular inequality pruning mechanism allowed by the strategy. They are not known when the current node (centered in  $s_{rep}$ ) is in fact processed, once that the representatives of the grandchild nodes are not stored in the current node.

## 5 Application of CLAP and ACIR to Slim-Tree

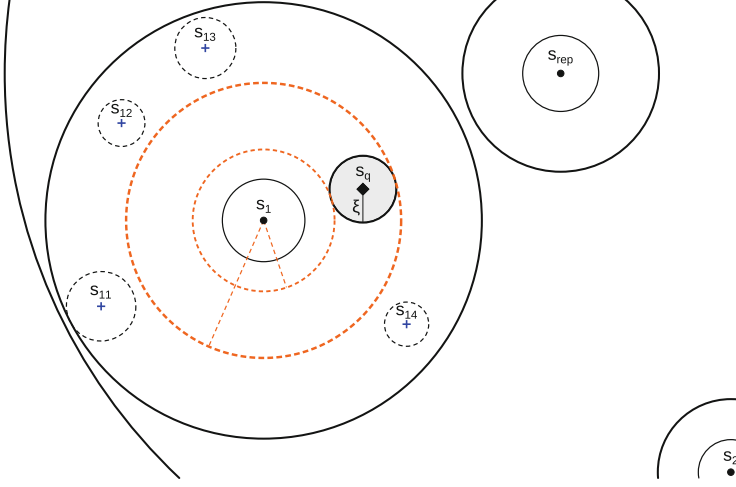
Slim-tree has two types of node: index nodes and leaf nodes. A leaf node presents the following structure (the characters  $\langle$  and  $\rangle$  delimit an array):

$$leaf\ node[\langle OID_i, s_i, \delta(s_i, s_{rep}) \rangle]$$

where  $OID_i$  is the identifier of the element;  $s_i$  is the element itself, stored as a feature vector;  $\delta(s_i, s_{rep})$  is the distance of  $s_i$  from the representative. An index node presents the following structure:

$$index\ node[\langle s_i, r_i, \delta(s_i, s_{rep}), Ptr(T_{s_i}), \#Ent(T_{s_i}) \rangle]$$

where  $s_i$  is the feature vector of the representative of the subtree  $T_{s_i}$ , pointed by  $Ptr(T_{s_i})$ ;  $r_i$  is the covering radius of this subtree, determined by the distance of



**Fig. 2.** Avoiding a disk access by anticipation of information. None of the grandchild nodes ( $s_{11}$ ,  $s_{12}$ ,  $s_{13}$ ,  $s_{14}$  and the representative  $s_1$  itself, which are children of the child node centered in  $s_1$ ) touches the uncertainty region, represented by the dashed ring

$s_i$  from the farthest element in the node of this subtree;  $\delta(s_i, s_{rep})$  is the distance of  $s_i$  from the representative of the current node;  $\#Ent(T_{s_i})$  is the number of entries in  $T_{s_i}$ .

After including the CLAP and ACIR techniques, both index and leaf nodes present new structures. In the following definitions, bold symbols represent the information added by CLAP technique and blue symbols represent the changes promoted by ACIR. A leaf node presents the following new structure:

$$leaf\ node[\langle \mathbf{Pos}_j \rangle, \langle \mathbf{OID}_i, s_i, \langle \delta(s_i, p_j) \rangle \rangle]$$

where  $Pos_j$  is the position of additional pivot  $p_j$  (e.g.  $Pos_j$  equals 2 if  $p_j$  is the second element stored in the node);  $\delta(s_i, p_j)$  is the distance of  $s_i$  from pivot  $p_j$ ; the remaining information are the same as the original structure. The criterion for choosing the local additional pivots used in this implementation is the greater sum of distances from the previous pivots. Nevertheless, other criteria can be analyzed. In the case of the first additional pivot, it is the element which has the greater distance from  $s_{rep}$ ; in the case of the second additional pivot, it is the element which has the greater sum of the distance from  $s_{rep}$  plus the distance from  $p_1$  and so on. In the ACIR technique strategy, once that the distance values from the representative are anticipated one level above in the MAM, these are removed from the structure of the leaf node.

The index nodes, on the other hand, are divided into two types: index nodes which are parents of leaf nodes, called *l-index node*, and index nodes which are parents of index nodes, called *i-index node*. Their structures are the following:



$$\begin{array}{c} l\text{-index node} \\ [\langle \mathbf{Pos}_j \rangle, \langle s_i, r_i, \delta(s_i, s_{rep}), \langle \delta(s_{il}, s_i) \rangle, \mathit{Ptr}(T_{s_i}), \#Ent(T_{s_i}), \langle \delta(s_i, p_j) \rangle \rangle] \end{array}$$

$$\begin{array}{c} i\text{-index node} \\ [\langle \mathbf{Pos}_j \rangle, \langle s_i, r_i, \delta(s_i, s_{rep}), \langle \delta(s_{il}, s_i) \rangle, \langle r_{il} \rangle, \mathit{Ptr}(T_{s_i}), \#Ent(T_{s_i}), \langle \delta(s_i, p_j) \rangle \rangle] \end{array}$$

The difference between those nodes is that, in an l-index node, only the array of distances  $\langle \delta(s_{il}, s_i) \rangle$  are added, where  $s_{il}$  is the l-th entry of the i-th child node and  $s_i$  is its representative. In an i-index node, the covering radius  $r_{il}$  of each  $s_{il}$  is also added.

## 6 Experimental Results

We performed extensive evaluations on both proposed techniques. The first technique has been implemented for only one local additional pivot in order to analyze its impact. We carried out experiments over datasets with different dimensionalities and cardinalities and employed metrics with different computational costs, so that we could evaluate our techniques in varied scenarios. In this section, we present the results achieved through combinations of three datasets and two metrics. For running the experiments, we used a machine with an Intel Core i5 2400@3.1 GHz processor, 4 GB of RAM@1333 MHz and HDD SATA III 6 Gb/s.

The datasets ALOI-T and ALOI-H belong to the *Amsterdam Library of Object Images*<sup>1</sup> (ALOI) [10]. These datasets are based on feature vectors extracted from 108,000 images of objects photographed several times, varying the position, the illumination and the combination of colors. ALOI-T consists of texture feature vectors with 140 dimensions, whereas ALOI-H consists of color histograms with 256 dimensions.

The test-collection CoPhIR<sup>2</sup> [2] contains 106 million images processed from Flickr. For all the images, the standard MPEG-7<sup>3</sup> features have been extracted: Scalable Color, Color Structure, Color Layout, Edge Histogram, Homogeneous Texture. In the experiments, the full feature vector of 282 dimensions was used in datasets of cardinality ranging from 10k to 10M elements, generating the datasets CoPhIR-10k-WL2, CoPhIR-100k-WL2, CoPhIR-1M-WL2, CoPhIR-10M-WL2, in order to evaluate the scalability of the techniques using a weighted euclidean distance. We also built the dataset CoPhIR-1M-M with 1M elements consisting of the Color Structure feature, with 64 dimensions, employing the Mahalanobis metric, also known as *histogram quadratic distance* [6]. This metric is expensive because it considers the correlation between bins of color histograms, which leads to more desirable results.

Since ALOI-H and CoPhIR-1M-M consist of color histograms, they were the chosen datasets for employing the Mahalanobis metric. The L2 metric, which is the euclidean distance, was employed on the rest of the datasets. On the CoPhIR datasets, the weights suggested in [1] were used for the Weighted L2 metric.

<sup>1</sup> Available at: <http://aloi.science.uva.nl>.

<sup>2</sup> Available at: <http://cophir.isti.cnr.it>.

<sup>3</sup> <http://mpeg.chiariglione.org/standards/mpeg-7>.

## 6.1 Performance in Similarity Queries

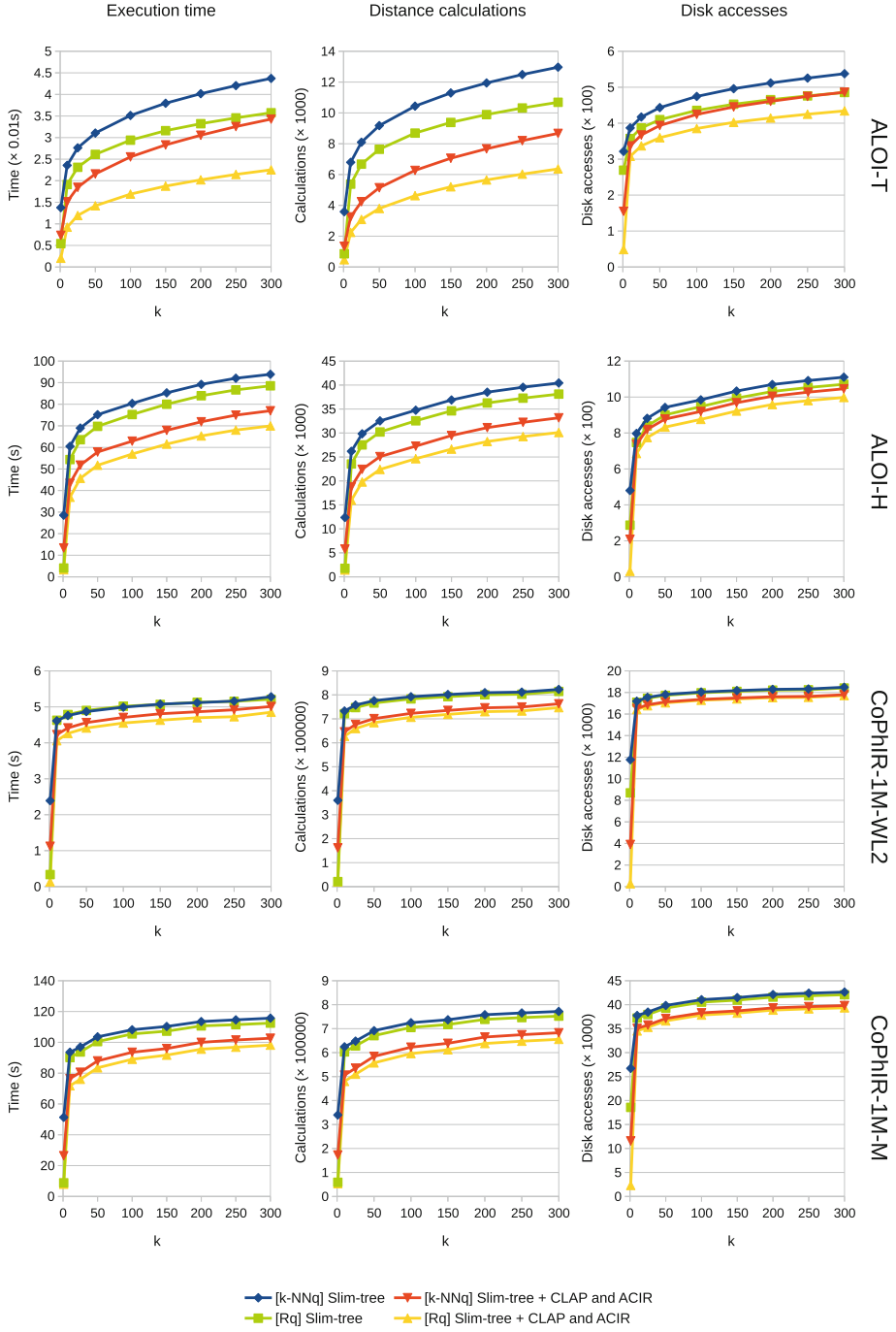
This subsection presents the results comparing the performance of Slim-tree + CLAP and ACIR with the original Slim-tree to execute similarity queries. In these experiments,  $k$ -NN queries and Range queries were performed varying the  $k$  value (1, 10, 25, 50, 100, 150, 200, 250 and 300) and using the corresponding radius values to retrieve  $k$  elements in Range queries. For each  $k$ , the results were obtained by performing queries multiple times (500 when using the L2 metric and 100 when using the Mahalanobis metric), each time with a random query element, and taking the average value.

Figure 3 shows the obtained results. The graphs show that our techniques lead to notable gains when compared to the original Slim-tree. For low selectivities (e.g.  $k = 1$  for  $k$ -NN queries and Range queries returning 1 element) the improvement was very high, being up to 62.51 % regarding execution time, 62.58 % in distance calculations and 96.93 % in disk accesses. When the selectivity was 50 or more, the gains were less expressive. Nevertheless, our techniques consistently outperformed the original structure, regarding every dataset, in execution time, number of distance calculations and number of disk accesses.

The first row of graphs in Fig. 3 corresponds to results over ALOI-T. In  $k$ -NN queries, the gain ranged from 21.59 % to 46.61 % in execution time, from 33.12 % to 62.58 % in distance calculations and from 9.57 % to 51.83 % in disk accesses. In Range queries, the gain ranged from 37.02 % to 62.51 %, from 40.48 % to 58.24 % and from 10.55 % to 82 %, respectively for the same variables.

The second row of graphs in Fig. 3 refers to ALOI-H, which employs the costly Mahalanobis metric. Our proposal presented a noticeable speedup, although the gains, especially in disk accesses, were lower if compared to the previous dataset. In  $k$ -NN queries, the gain ranged from 18 % to 53.2 % in execution time, from 17.96 % to 53.2 % in distance calculations and from 5.8 % to 56.5 % in disk accesses. In Range queries, the gain ranged from 19.77 % to 32.14 %, from 19.68 % to 32.12 % and from 6.85 % to 89.43 %, respectively for the same variables.

Regarding the CoPhIR datasets, the third row in Fig. 3 shows the results of experiments carried out over 1M elements with 282 dimensions by varying  $k$  (dataset CoPhIR-1M-WL2). This is the dataset for which our techniques presented the lowest gains for high values of  $k$ . Nonetheless, its use allowed improving the performance of every evaluated aspect. Regarding execution time, distance calculations and disk accesses, respectively, the gain in  $k$ -NN queries ranged from 5.12 % to 53.05 %, from 7.29 % to 55.09 % and from 3.73 % to 66.82 %, while the gain in Range queries ranged from 6.92 % to 60.36 %, from 5.17 % to 13.05 % and from 3.80 % to 96.93 %. In the set of experiments using CoPhIR-1M-M, the proposed techniques achieved even better results, as the main improvement of the techniques regards distance calculations and the cost of the Mahalanobis metric is much higher than the cost of the L2 metric. In this dataset, the gain in  $k$ -NN queries ranged from 11.25 % to 48.51 % in execution time, from 11.53 % to 49.09 % in distance calculations and from 6.59 % to 56.65 % in disk accesses. Finally, the gain in Range queries ranged from 12.72 % to 20.37 %, from 9.65 % to 20.39 % and from 6.49 % to 87.6 %, respectively for the same variables.

Fig. 3. Results of the experiments varying  $k$

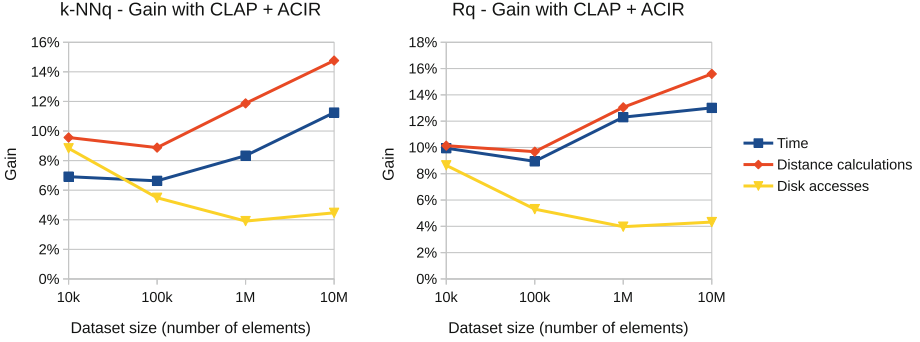
## 6.2 Evaluation of Construction Issues and Scalability of Gain

This section evaluates the impact of the proposed techniques when compared to the original structure in terms of building time, page size and resulting data file size, as well as how the gain promoted by the techniques behaves with the size of the dataset. The information of both structures regarding their construction are presented in Table 1. Note that the Slim-tree with CLAP and ACIR required two page sizes, one for index nodes and one for leaf nodes. This is because we wanted to minimize the overhead of information in index nodes by increasing their page size. The page size of index nodes will be usually a multiple of the page size of leaf nodes, to allow using the same buffer pool for both node types. Although the index nodes in Slim-trees + CLAP and ACIR are the double of the size of the original Slim-trees, taking a longer time to read them from disk, the higher pruning ability of CLAP and ACIR allowed a better performance in our experiments. Also, note that both structures have similar file sizes. Slim-tree + CLAP and ACIR presented files from 2.4 % to 6.91 % smaller than Slim-tree did. It can be explained by the fact that, with the anticipation of information, leaf nodes store less information than before. Since there are many more leaf nodes than index nodes, storing less information within leaf nodes leads to a little smaller file sizes.

The building time for Slim-trees with CLAP and ACIR was from 12.32 % to 34.32 % higher, as expected, because it involves additional computations such as distance calculations regarding the cutting local additional pivots. However, the worst case was on the smallest dataset, CoPhIR-10k-WL2, which resulted in a difference of only 1.528s. Thus, considering the performance gain in queries, the proposed techniques are worth the higher building time.

**Table 1.** Construction information of both structures for all datasets

Dataset	Slim-tree	Time (s)	Page size (KB)	File size
ALOI-T	Original	37.296	32	103 MB
	CLAP and ACIR	43.650	64 (index) — 32 (leaf)	101.2 MB
ALOI-H	Original	29637.4	64	162 MB
	CLAP and ACIR	33289.4	128 (index) — 64 (leaf)	155.9 MB
CoPhIR-10k-WL2	Original	4.452	64	15 MB
	CLAP and ACIR	5.98	128 (index) — 64 (leaf)	14.640 MB
CoPhIR-100k-WL2	Original	58.967	64	139 MB
	CLAP and ACIR	71.684	128 (index) — 64 (leaf)	129.4 MB
CoPhIR-1M-WL2	Original	740.946	64	1.4 GB
	CLAP and ACIR	871.27	128 (index) — 64 (leaf)	1.334 GB
CoPhIR-10M-WL2	Original	9481.47	64	14 GB
	CLAP and ACIR	10954.6	128 (index) — 64 (leaf)	13.329 GB
CoPhIR-1M-M	Original	11763.2	8	449 MB
	CLAP and ACIR	13385.5	16 (index) — 8 (leaf)	434 MB



**Fig. 4.** Results of the scalability experiments over CoPhIR

Finally, we evaluated how the gain promoted by CLAP and ACIR techniques behaves according to the dataset size. In this experiment, we fixed  $k$  to 10 and varied the dataset size from 10k to 10M elements with 282 dimensions, using the Weighted L2 metric and random query elements. Figure 4 presents the obtained results. It can be seen that the improvement raises with the dataset size regarding distance calculations and execution time. The gain in disk accesses drops until 1M elements and afterwards presents a sensible increase, being always positive. These results show that the proposed techniques scale well with increasing dataset size.

## 7 Conclusions

We have proposed new techniques based on local additional pivots and anticipation of information for improving the pruning ability of dynamic MAMs: CLAP and ACIR. Our techniques were extensively tested and achieved better results in all evaluated scenarios, for both  $k$ -NN and Range queries. We also showed that the gain promoted by the techniques scales well with the dataset size. Moreover, the CLAP and ACIR techniques do not affect the dynamicity of the underlying MAM and, just like they were implemented over Slim-tree, other dynamic hierarchic MAMs could be improved by using them as well.

Both contributions of this work opens possibilities for future work. Regarding the CLAP technique, the use of more than one additional pivot per node could be explored, as well as strategies for selecting cutting local additional pivots. In ACIR, the information considered in this work to be anticipated from child nodes are the distances of each element from their representative and the radius values. However, other information could be anticipated due to the CLAP technique, such as the feature vectors of each additional pivot and the distances of each element from the additional pivots. Our insight is that, by having more information anticipated, the improvements can be even better. We are working on these extensions to present them in a next work.

## References

1. Batko, M., Kohoutkova, P., Novak, D.: CoPhIR image collection under the microscope. In: 2nd International Workshop on Similarity Search and Applications, pp. 47–54. IEEE Computer Society, Washington, DC (2009)
2. Bolettieri, P., Esuli, A., Falchi, F., Lucchese, C., Perego, R., Piccioli, T., Rabitti, F.: CoPhIR: A Test Collection for Content-Based Image Retrieval. Computing Research Repository abs/0905.4627v2 (2009)
3. Burkhard, W.A., Keller, R.M.: Some approaches to best-match file searching. *Commun. ACM* **16**(4), 230–236 (1973)
4. Carélo, C.C.M., Pola, I.R.V., Ciferri, R.R., Traina, A.J.M., Traina Jr., C., Ciferri, C.D.A.: Slicing the metric space to provide quick indexing of complex data in the main memory. *Inf. Syst.* **36**(1), 79–98 (2011)
5. Ciaccia, P., Patella, M., Zezula, P.: M-Tree: An efficient access method for similarity search in metric spaces. In: 23rd International Conference on Very Large Data Bases, pp. 426–435. Morgan Kaufmann, San Francisco (1997)
6. Deza, M.M., Deza, E.: *Encyclopedia of Distances*. Springer, Heidelberg (2009)
7. Esuli, A.: Use of permutation prefixes for efficient and scalable approximate similarity search. *Inf. Process. Manage.* **48**(5), 889–902 (2012)
8. Faloutsos, C.: *Searching Multimedia Databases by Content*. Advances in Database Systems, vol. 3. Springer, New York (1996)
9. Gaede, V., Gunther, O.: Multidimensional access methods. *ACM Comput. Surv.* **30**(2), 170–231 (1998)
10. Geusebroek, J.M., Burghouts, G.J., Smeulders, A.W.M.: The amsterdam library of object images. *Int. J. Comput. Vis.* **61**(1), 103–112 (2005)
11. Pola, I.R.V., Traina Jr., C., Traina, A.J.M.: The MM-tree: a memory-based metric tree without overlap between nodes. In: Ioannidis, Y., Novikov, B., Rachev, B. (eds.) ADBIS 2007. LNCS, vol. 4690, pp. 157–171. Springer, Heidelberg (2007)
12. Skopal, T., Hoksza, D.: Improving the performance of M-Tree family by nearest-neighbor graphs. In: Ioannidis, Y., Novikov, B., Rachev, B. (eds.) ADBIS 2007. LNCS, vol. 4690, pp. 172–188. Springer, Heidelberg (2007)
13. Skopal, T., Pokorný, J., Snášel, V.: Nearest neighbours search using the PM-tree. In: Zhou, L., Ooi, B.-C., Meng, X. (eds.) DASFAA 2005. LNCS, vol. 3453, pp. 803–815. Springer, Heidelberg (2005)
14. Traina Jr., C., Filho, R.F.S., Traina, A.J.M., Vieira, M.R., Faloutsos, C.: The omni-family of all-purpose access methods: a simple and effective way to make similarity search more efficient. *VLDB J.* **16**(4), 483–505 (2007)
15. Traina Jr., C., Traina, A.J.M., Faloutsos, C., Seeger, B.: Fast indexing and visualization of metric data sets using slim-trees. *IEEE Trans. Knowl. Data Eng.* **14**(2), 244–260 (2002)
16. Traina Jr., C., Traina, A.J.M., Filho, R.F.S., Faloutsos, C.: How to improve the pruning ability of dynamic metric access methods. In: 11th International Conference on Information and Knowledge Management, pp. 219–226. ACM, New York (2002)
17. Vieira, M.R., Traina Jr., C., Chino, F.J.T., Traina, A.J.M.: DBM-Tree: trading height-balancing for performance in metric access methods. *J. Braz. Comput. Soc.* **11**(3), 37–51 (2005)
18. Zezula, P., Amato, G., Dohnal, V., Batko, M.: *Similarity Search: The Metric Space Approach*. Advances in Database Systems, vol. 32. Springer, New York (2006)

Advances in Databases and Information Systems  
19th East European Conference, ADBIS 2015, Poitiers,  
France, September 8-11, 2015, Proceedings  
Morzy, T.; Valduriez, P.; Bellatreche, L. (Eds.)  
2015, XXVIII, 474 p. 174 illus., Softcover  
ISBN: 978-3-319-23134-1