A visual framework to understand similarity queries and explore data in Metric Access Methods

Marcos R. Vieira*, Fabio J.T. Chino, Caetano Traina Jr. and Agma J.M. Traina

Computer Science Department,
University of São Paulo at São Carlos,
Av. Trabalhador São-carlense, 400, São Carlos 13566-590, Brazil
E-mail: mvieira@cs.ucr.edu
E-mail: fabio@icmc.usp.br
E-mail: caetano@icmc.usp.br
E-mail: agma@icmc.usp.br
*Corresponding author

Abstract: This paper presents the MAMView framework to help users and developers in understanding the data organisation in Metric Access Methods (MAM). Users and developers can explore and share dynamic and interactively 2- or 3-dimensional representations of a MAM. Such representations can be the steps of a similarity query or the insertion of an object, or the data organisation in a MAM. MAMView was developed as a practical tool that has been successfully applied in studying existing MAM, helping novice users to better understand the behaviour and properties of such structures, as well developers to verify and drill-down their new proposed structures.

Keywords: data visualisation; visual exploration; MAM; metric access methods; similarity queries.

Reference to this paper should be made as follows: Vieira, M.R., Chino, F.J.T., Traina Jr., C. and Traina, A.J.M. (2010) 'A visual framework to understand similarity queries and explore data in Metric Access Methods', *Int. J. Business Intelligence and Data Mining*, Vol. 5, No. 4, pp.370–397.

Biographical notes: Marcos R. Vieira received his BSc Degree in Computer Engineering from Federal University of São Carlos, Brazil, and his MSc Degree from University of São Paulo at São Carlos, Brazil. Currently, he is a PhD candidate at University of California at Riverside, USA.

Fabio J.T. Chino received his BSc and MSc Degrees both in Computer Science from University of São Paulo at São Carlos, Brazil.

Caetano Traina Jr. received his PhD in Computational Physics at the Physics Institute of São Carlos at University of São Paulo, Brazil. He is a Professor with the Department of Computer Science of University of São Paulo at São Carlos, Brazil.

Agma J.M. Traina received her PhD in Computational Physics at the Physics Institute of São Carlos from University of São Paulo, Brazil. She is a Professor with the Department of Computer Science of University of São Paulo at São Carlos, Brazil.

1 Introduction

Human beings have a great ability to grasp information presented graphically (Keim, 2001). Attempts to visually summarising information began with histograms and pie-charts, which aggregate the information of large volumes of data. However, they are limited to present simple statistical measurements. A challenge in the information visualisation field is to deal with the ever increasing volume of data generated by the nowadays computational systems, allied to the expanding diversity of data types. Moreover, data types are much more complex and larger than the numbers and short strings of characters traditionally managed by the Database Management Systems (DBMS). Complex and large data, which are inherently abstract, come from applications of most diverse fields, such as medicine, astronomy, surveillance, business and entertainment, materialised as images, videos, time series, genetic sequences, and so on. Since it is not feasible to use the conventional indexing structures for unidimensional domains, e.g., B-tree (Bayer and McCreight, 1972) and B^+-tree (Comer, 1979), commonly available in commercial DBMS, special structures more suitable to efficiently index, manage and query complex data have been proposed. The reason that conventional structures, like the B-tree, cannot handle complex data is that such structures rely on the total ordering property, which holds for numbers and short strings of characters, but not for complex data. Even Spatial Access Methods (SAM), such as the R-tree (Guttman, 1984) and its variants (Beckmann et al., 1990; Sellis et al., 1987) cannot deal with data that do not have a fixed number of attributes.

Traditional DBMS storing complex data has to follow a similar approach employed by human beings in order to compare complex data, which is mainly done by comparing pair of elements to find those that are the most similar to the intended one. For example, consider a surveillance environment where images containing people faces are stored in a database. When a new image needs to be checked whether it belongs to a given database of images, the new image is compared one by one with the ones in the database. The probability of having *exactly* the same image in the database tends to zero, as people and the acquisition environment changes over time. The same happens with equivalent settings for any complex data types. Thus, complex data are mainly compared by *similarity*, not *equality*. Usually, similarity (or distance) between two complex elements is measured using a distance function (see Section 2) that gives small values for *similar* elements, and larger values for more *dissimilar* elements.

Metric Access Methods (MAM) are index structures suitable to efficiently index and retrieve complex data from large datasets. The only requirement to build a MAM for a particular set of complex elements is defining a proper distance function. MAM are well-suited structures to support answering similarity queries, such as the range and k-nearest neighbour queries (explained in the Section 2).

As MAM can rely only on the distances between elements, the indexes, their internal structure and similarity query algorithms are more complex and far less intuitive than those for unidimensional data or SAM; the later can take advantage of intrinsic spatial representation. This fact makes learning, developing and tuning MAM-enabled databases a burdensome task. This paper presents a visual framework called MAMView that overcomes the spatial limitation of MAM, providing to the user – who can be a student learning a MAM structure, a MAM developer or a database administrator tuning a MAM-based index – an interactive visual presentation of the indexed data. The MAMView allows users/developers to highlight/fade elements of the structure at the users' desire, as well as to show the algorithm traversing over the data when answering similarity queries. This paper makes the following main contributions:

- It presents the MAMView framework and its underpinning techniques, including a complete, practical and easy-to-use tool to generate visualisations of MAM and their operations. The MAMView framework allows users to interactively explore the data indexed by MAM, as well as to follow the paths of execution of similarity queries, driving the developer to examine the structure bottlenecks and to analyse possible optimisation scenarios.
- It describes the MAMView architecture, which is platform independent and takes takes advantage of XML format to store and share the visualisations and the graphical support given by the OpenGL library (Group, 2009)
- It improves the understanding of MAM, through a discussion of users' experiments performed with graduate students of a database class and by experienced MAM developers.

It is important to highlight that, distinctly from other existing works in the literature (Baker et al., 1999; Brabec et al., 2003; Hadjieleftheriou et al., 2005; Livny et al., 1997; Shah et al., 1999), the MAMView framework allows users to manipulate, visualise and browse data indexed by MAM, depicting the indexed data or its structure. The visualisation of high-dimensional or even non-dimensional data is possible due to an extension of the FastMap algorithm (Faloutsos and Lin, 1995), which was employed to produce intuitive low-dimensional spatial representations of MAM-indexed data. Moreover, this paper shows a way to take advantage of the strong ability of human beings to understand graphically presented information (Ware, 2000), fostering the human ability to understand a MAM and helping analysts and database administrators in their implementation and tuning tasks.

The remainder of the paper is structured as follows: Section 2 summarises related concepts required to understand this paper; Section 3 details the proposed MAMView framework and the techniques developed to implement it; in Section 4 we present and discuss some evaluation results performed on the MAMView framework to help users to understand various MAM algorithms; and Section 5 gives the conclusions of this paper.

2 Background

Visualisation techniques greatly improve the understanding of data distributions (Elmqvist et al., 2008; Keim and Kriegel, 1994; Keim, 2001). Therefore, many works have targeted the problem of designing tools to help developing new access methods. The `amdb` tool (Shah et al., 1999) aims at visualising access methods built on top of the Generalised Search Tree (GiST) abstraction (Hellerstein et al., 1995), taking advantage of the spatial (or multidimensional) behaviour of the indexed objects. Many other visualisation tools specifically designed for SAM have been proposed, examples of such tools are the SaIL (Hadjieleftheriou et al., 2005) and VASCO (Brabec et al., 2003). Unfortunately, none of these tools designed for multidimensional domains can be employed to metric domains. Another interesting work is the DEVise system (Livny et al., 1997), aimed at visualising and browsing datasets stored in relational DBMS. The JDSL Visualiser (Baker et al., 1999) was introduced for teaching traditional data structures.

Many works concerning visualisations documents and bibliography information have been proposed. In the Bead system (Chalmers and Chitson, 1992) a set of bibliographies are displayed in a 3D space, thanks to the use of a multidimensional scaling approach (Kruskal and Wish, 1978). The main complaint about Bead is the high computational cost required to produce visualisations. The SPIRE system (Miller et al., 1997) encompasses four types of 2D projections, allowing users to intuitively perceive which documents are more related to others, and possibly browse them. However, such data are not organised by access methods or can handle metric data, which makes them infeasible for our problem.

The techniques presented in this paper are not restricted only to visualise MAM or data having inherent spatial properties. In fact, as human beings capture graphical information in low-dimensional environment (2D and 3D) more intuitively, the visualisation proposed herein maps *any* data in metric domains to 2D or 3D Euclidean spaces, which are the most suitable for an intuitive study.

2.1 Metric spaces and trees

A metric space is defined by a pair $\mathbb{M} = < \mathbb{S}, d() >$, where $\mathbb{S}$ is the metric data domain and $d()$ is the distance function, which measures the similarity between a pair of elements. The distance function $d()$ is required to comply with the following three properties in order to $\mathbb{M}$ be considered a metric space, for any $s_1, s_2, s_3 \in \mathbb{S}$:

- symmetry: $d(s_1, s_2) = d(s_2, s_1)$
- non-negativity: $0 \leq d(s_1, s_2) \leq \infty$ for all $s_1 \neq s_2$, or $d(s_1, s_2) = 0$ for $s_1 = s_2$
- triangle inequality: $d(s_1, s_3) \leq d(s_1, s_2) + d(s_2, s_3)$.

Spatial data following an L_p-metric, such as the Euclidean distance L_2 or the Manhattan distance L_1, are special cases of metric spaces. The two most common types of similarity queries are:

- *Range Query* (*RQ*): Given a query element $s_q \in \mathbb{S}$, a maximum query distance r_q, and a search domain $S \subseteq \mathbb{S}$, $RQ(s_q, r_q)$ returns a set of elements $R \subseteq S$, such that $\forall s_i \in R$, $d(s_i, s_q) \leq r_q$. An example is: "Select all proteins that are similar to the protein s_q by up to 5 purine bases", and it is represented as $RQ(s_q, 5)$
- *k-Nearest Neighbour query* (*kNNQ*): Given a query element $s_q \in \mathbb{S}$ and an integer value $k \geq 1$, $kNNQ(s_q, k)$ returns $R \subseteq S$ containing k elements that have the smallest distance to the query element s_q, according to the distance function $d()$, that is $k = |R|$ and $\forall s_i \in R$, $s_j \in \{S - R\}$, $d(s_q, s_i) \leq d(s_q, s_j)$. An example is: "Select the 3 most similar proteins to protein s_q", and it is represented as $kNNQ(s_q, 3)$.

MAM organise a dataset $S \subseteq \mathbb{S}$ as a metric tree, dealing only with the elements S and the distances between pair of elements defined by $d()$. Basically, a metric tree divides a dataset into regions (or balls) and chooses well-suited elements, called *representatives*, to represent the elements in each region. The elements in a region are stored in a disc register called *node* The M-tree (Ciaccia et al., 1997) was the first dynamic MAM proposed in the literature that efficiently stores elements in disc pages. The Slim-tree (Traina et al., 2000) extended it, introducing faster splitting algorithms and presenting the first technique to estimate the overlap between subtrees, called the *fat-factor*. The Slim-tree also has an optimisation algorithm called Slim-Down, which reorganises elements indexed in a Slim-tree to answer similarity queries more efficiently.

Both the M-tree and Slim-tree have two node types: leaf nodes, where the elements are actually stored, and routing (indexing) nodes that organise the tree structure. Both types of nodes store the node representative and its covering radius. Leaf nodes also include the elements in the covered region and their distances to the representative. Routing nodes also store the representatives of its subtrees and their distances to the node representative. When a query search is evaluated, the query element is compared to the representative stored in the routing node, using the triangle inequality property to prune subtrees that cannot have elements qualifying for the answer. Region overlap is the major drawback of a MAM structure (Traina et al., 2000), as it leads to accessing many subtrees when traversing a tree to answer similarity queries. Figure 1(a) graphically represents a Slim-tree indexing 19 elements ($s_1, \ldots, s_{19}$) using the Euclidean distance, showing for each node, its representative and covering radius. The same Slim-tree is presented hierarchically in Figure 1(b), however this representation does not preserve the notion of the covering region and the distances among the elements. For example, the presentation in Figure 1(b) does not disclose possible overlap occurring among nodes or proximity between elements.

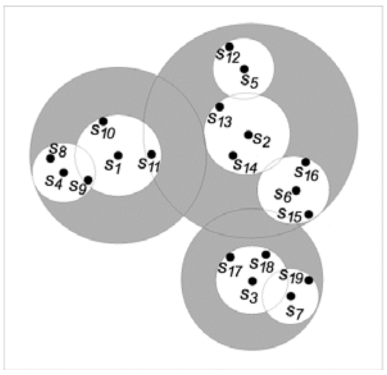
Although the representation in Figure 1(a) is more intuitive in understanding the structure of the Slim-tree than the one shown in Figure 1(b), it has two drawbacks:

- it is hard to scale to large datasets
- it can only be drawn for multidimensional datasets, such as those following the Euclidean distance function.

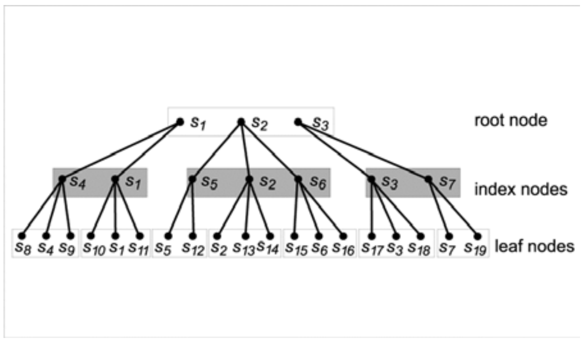
It should be noted that it is always possible to visualise datasets indexed in a MAM that are represented in the 2-dimensional domain. The problem comes when ‘purely’ metric or high dimensional data are indexed in MAM. In these cases, mapping or selection algorithms that produce a lower dimensional representation of the data need to be employed. There are some algorithms in the literature for this purpose, such as: Multidimensional Scale (MDS) (Kruskal and Wish, 1978) and its faster variations (Morrison and Chalmers, 2004), MetricMap (Wang et al., 1999), SparseMap (Hristescu and Farach-Colton, 1999) and FastMap (Faloutsos and Lin, 1995).

In our work we use the FastMap (Faloutsos and Lin, 1995) algorithm since it has some properties that are needed and the other methods do not have. First, the FastMap algorithm has a low computational cost involved on mapping objects from its original domain to a 2-dimensional domain. Second, the FastMap algorithm outputs the mapping data in a *k*-dimensional Euclidean space (in our particular case, 2- or 3-dimensional Euclidean space). And third, it also provides a measurement to verify the amount of mischaracterisation of the mapped dataset. Since all of these three properties are essential in our domain, the FastMap algorithm is the most suitable algorithm for our purposes.

Figure 1 Two possible ways to represent a Slim-tree indexing 19 elements: (a) spatial distribution of nodes and elements and (b) hierarchical view



(a)


s1 s2 s3
root node
s4 s1 s5 s2 s6 s3 s7
index nodes
s8 s4 s9 s10 s1 s11 s5 s12 s2 s13 s14 s15 s6 s16 s17 s3 s18 s7 s19
leaf nodes

(b)

2.2 The FastMap algorithm

FastMap is a mapping algorithm that receives as an input the dataset to be mapped in its original space, the distance function to be employed to measure the distance among elements in the dataset, and the target number of dimension *k*. The output of the algorithm is the mapped dataset represented in the *k* dimension. It executes an iterative process, where the number of iterations is the target number of dimensions *k*. For our problem, *k* is set to 2 or 3 depending on the user choice of the Euclidean space where the dataset should be visualised. At each iteration, two elements with the largest *d*() are chosen as 'pivots' of the target dimension. The pivots define the axis of the dimension (see Figures 2(a)–(c)). Thereafter, the projections of all other elements in this axis are calculated triangulating the element and the two pivots using the Cosine Law. Thus, the algorithm requires only the distances between pairs of elements, as illustrated in Figure 2(d). The error imposed in the positioning of each element due to this projection is measured using the *stress* formulation given in equation (1):

$$stress = \sqrt{\frac{\sum_{s_i, s_j \in \mathbb{S}} (\hat{d}(s_i, s_j) - d(s_i, s_j))^2}{\sum_{s_i, s_j \in \mathbb{S}} d(s_i, s_j)^2}} \tag{1}$$

where $d(s_i, s_j)$ and $\hat{d}(s_i, s_j)$ are the distances in the original and in the mapped spaces, respectively. Small values for *stress* mean that the mapping procedure closely follows the original distances, thus resulting in a mapping with better quality. The k-dimensional points preserve the original distances as much as possible, by spreading the error among the mapped elements. For our intended application, we implemented two versions of the FastMap algorithm:

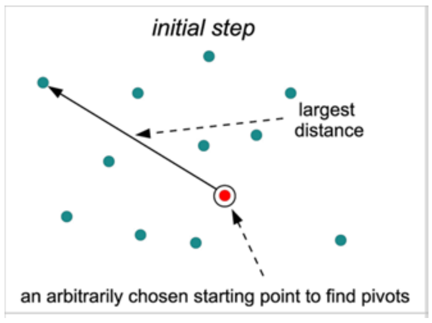
- *FastMap()*: Receives a dataset with N elements, a distance function $d()$ and the target k number of dimensions. It scans the dataset to find k pairs of elements (pivots) that are far apart to each other.
- *CoordinateFastMap()*: Receives an element s_i, a distance function $d()$ and the k pairs of pivots. It maps the given element s_i to a point in the k-dimensional space defined by the k pairs of pivots.

This process exhibits three important properties:

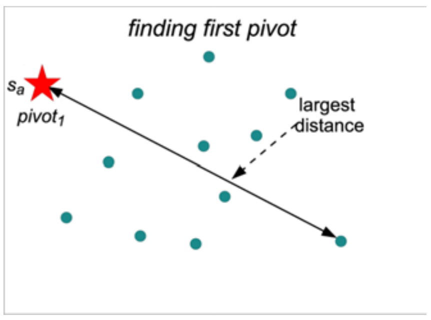
- It is a linear algorithm, $O(N)$, so it is scalable to process very large datasets.
- The k pairs of pivots can be stored to reuse in further executions of the algorithm, which is done by the *CoordinateFastMap()*. Elements can be mapped incrementally, enabling to map the different layers of the tree and different query results, in a unique and consistent target space.
- As it preserves the original distances as much as possible, it enables the representation of geometric structures overlaying the element in a way similar to the visualisation of the hyper-bounding regions defined by traditional access methods.

In this way, we are able to visualise the 'balls' that cover regions in metric spaces.

Figure 2 Steps of the FastMap algorithm (one iteration only): (a) an arbitrary element is chosen to start the iteration, then the most distant element is chosen to be the first pivot; (b) the farthest element to the first pivot is chosen to be the second pivot; (c) the distance between both chosen pivots are the largest in the dataset and (d) all the remaining elements are mapped using the Cosine Law (see online version for colours)

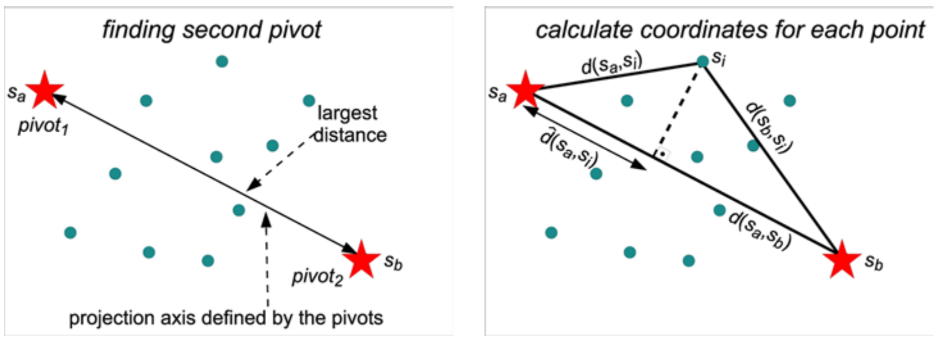


(a)



(b)

Figure 2 Steps of the FastMap algorithm (one iteration only): (a) an arbitrary element is chosen to start the iteration, then the most distant element is chosen to be the first pivot; (b) the farthest element to the first pivot is chosen to be the second pivot; (c) the distance between both chosen pivots are the largest in the dataset and (d) all the remaining elements are mapped using the Cosine Law (see online version for colours) (continued)



(c) (d)

It is important to find pivots in each dimension as farther from each other as possible to achieve good mappings. Finding pairs of elements far apart is the most time-consuming part of FastMap algorithm. However, by construction, the elements stored at the routing nodes contain elements naturally distant from each other. Thus we choose the pivots from the routing elements of the first levels of the that amount to at least 50 elements. In our experiments, 50 pivots showed to be good enough – increasing it does not improve the visualisation, whereas much smaller numbers deteriorates the visualisation. Thus, the selection of pairs of elements far apart is restricted to a much smaller number than *N*, speeding up the process. The algorithm to build visualisations is presented in Figure 3.

Figure 3 Algorithms to visualise a metric tree

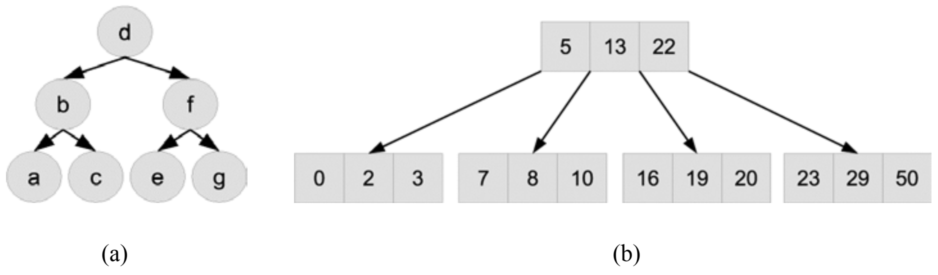
```
Algorithm: VisualizeMT()
   input: a metric tree T, root level of T to be mapped, array of pivots
   output: array of pivots
begin
   1: if array of pivots is null, execute FindPivots()
   2: retrieve objects at root level in T and the corresponding radius associated
      with them, if level is not leaf level
   3: execute CoordinateFastMap() for each object retrieved
   4: for each selected object, plot the point and the corresponding bounding region
   5: return the array of pivots to be reused in further executions of this algorithm
end

Algorithm: FindPivots()
   input: metric tree T
   output: array of pivots
begin
   1: count the number of objects stored in each level of T
   2: identify the lowest level L with at least 50 objects
   3: select all objects S of this level L
   4: execute FastMap() for S
   5: store the pivots found in the array of pivots
end
```

3 The visualisation framework

One of the main challenges in developing a visualisation tool for data structures is on how to create intuitive graphical representations that properly shows the organisation and behaviour of the structure. Traditional data structures, such as binary trees and B-trees, take advantage of the inherent ordering property among elements, which makes them quite simple to represent graphically, as exemplified in Figure 4. However, when dealing with data in metric spaces, one need to answer the question "*How to intuitively represent a MAM when there is no spatial or ordering relationship among the elements, but only distances among them?*" An initial suggestion for such a graphical representation of low dimensional datasets is given in Figure 1, which compares a spatial presentation with a graph-like one. However, when the dataset is high dimensional or metric, a different approach must be employed. The modified FastMap algorithm, discussed in Section 2.2, produces a low-dimensional mapping of any indexed data, revealed to be a valuable resource to solve this problem, enabling the creation of the MAMView visualisation framework.

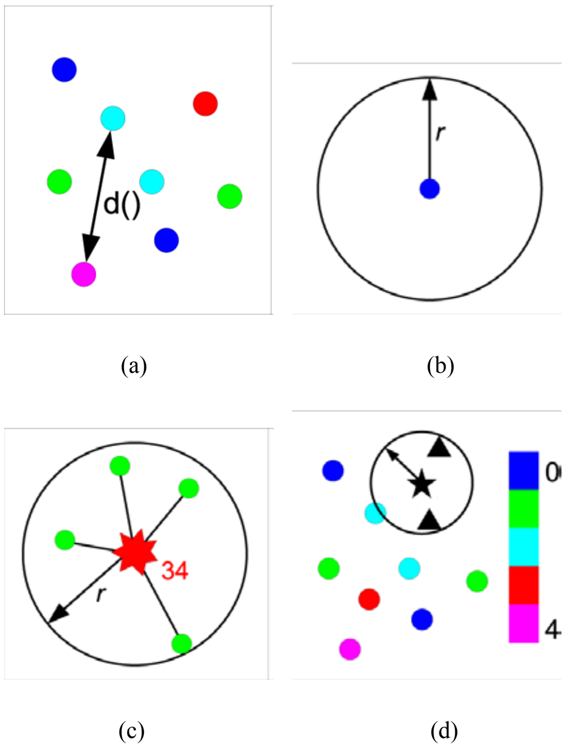
Figure 4 Two examples of graphical representations of: (a) a binary tree indexing a sequence of characters and (b) a B-tree indexing 15 numbers



The main graphical primitives employed by MAMView to show MAM elements are displayed in Figure 5. An element is represented as a coloured point in the space, where colours represent the level where the element is stored in the tree. A metric ball specified by a pair element/radius is represented by a circle entered at the element. The node representatives are represented by a seven-point star. Elements are identified by textual tags representing their identifiers (ids). Elements stored in a node are represented by links between each element and the node's representative. Special shapes and icons represent special cases of elements and connections. For example, a five-point black star represents a query centre and black triangles represent elements in the query result.

Regions in metric space are mapped to the Euclidean space since it is the most intuitive space for users to compare distances. MAMView allows users to interactively traverse a MAM, following its behaviour during queries, insertions, dump operations, and so on. In this way, it helps both the MAM developer to drill-down its internal algorithms whom literally 'see' where to spend more optimising effort, and the database administrator to understand how to tune an index to obtain better behaviour. An interaction mechanism allows manipulating the MAM visualisation, helping to improve the intuitiveness even further, as well as it is a valuable tool aiding to deal with the inherent visual cluttering of large datasets.

Figure 5 Visual representations of a MAM indexing elements in the Euclidean space: (a) the abstraction of distances among elements; (b) a node region defined by an element and a radius r; (c) four elements stored in a node defined by the representative (7-point star) with identifier 34 and (d) different levels are represented using different colours, query results are show as black triangles, and query element as a 5-point black star (see online version for colours)



3.1 *A Generic Model of MAM's Organisation*

As the only information available to a MAM is the elements to be indexed and the distances among them, the great majority of MAM share the same inherent features. However, there is a broad variety of mechanisms and ways to organise data that must be considered when creating a generic model to organise MAM.

In this paper, we describe a model that considers the MAM indexing data stored in a DBMS. The target access methods materialise as multi-via trees and divide the indexed space in metric *balls*. The balls are specified as <element, radius> pairs. A node is a *node ball*, and the centre of a node ball is its *representative*. Notice that every element stored in a node must be covered by the node ball but, due to node overlapping, elements can exist covered by the node ball that is not stored in the node. A typical MAM leads to the hierarchical placement of the nodes. Thus, our generic model requires the following conditions to be met by a MAM:

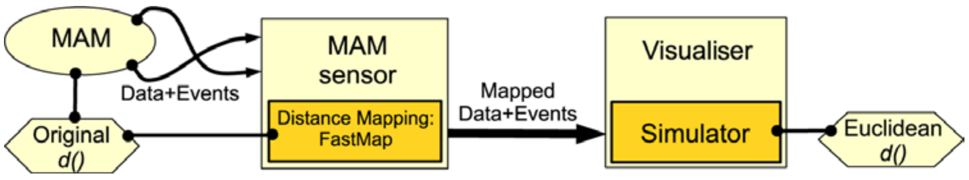
- Each node must have a single identifier
- Nodes should have only one parent, except the root node that has none

- Nodes can have any number of children
- Each node represents a space region defined by a <element, radius> pair
- An element is associated to either a single node or to none (e.g., a query centre)
- An element can occur more than once (a representative is also stored in a leaf node)
- The tree is organised in levels, which indicates how far a node is from the root
- A query requires one or more centres and one or more radii, resulting in a list of elements indexed in the tree.

3.2 *The MAM visualisation steps*

The MAMView framework works as a probe inside the metric-based algorithms. It first collects the necessary information from the MAM algorithms when they are executed. It then processes the information to provide an interactive visualisation of the MAM. The MAMView framework is composed of two logical modules: the MAM *Sensor* and the *Visualiser*. Figure 6 summarises this structure. A vital operation of any metric-based system is the distance calculation, which is performed as part of the Sensor. The Sensor then maps the elements in the original space to the Euclidean space for visualisation using the same metric employed by the MAM. On the other hand, the *Visualiser* employs the Euclidean metric in the mapped space. Besides the distances, the sensor gathers all required information when the MAM algorithms are executed, making them available to the *Visualiser*, which runs after the MAM operations have already finished. When all required information is available to the *Visualiser*, it virtually re-creates the algorithm execution, by simulating its execution. This model allows the user to freely interact with the visualisation, moving backward or forward the visualisation of the execution of the algorithm.

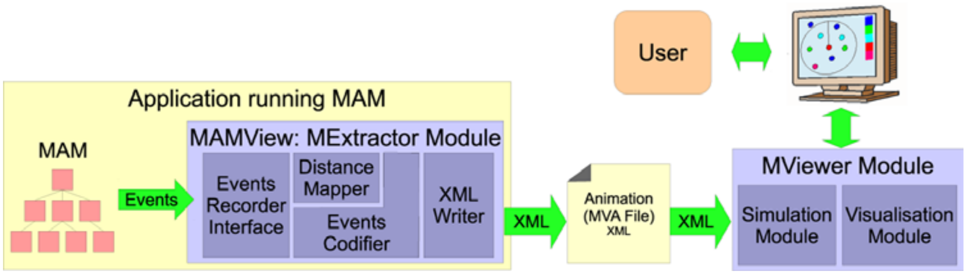
Figure 6 The logical structure to generate a MAM visualisation (see online version for colours)



3.3 *The MAMView architecture*

The MAMView framework has two main modules: the **MExtractor** and the **MViewer**, as is illustrated in Figure 7. The MExtractor is an Application Program Interface (API) for C/C++ environments that extracts data from the MAM itself, tracking the events (local transformations) installed by the developer, and maps the distances from the metric space to a 2- or 3-dimensional Euclidean space. The collected data are then processed and stored in the *MAMView Animation* (MVA) format files. The MVA file is formatted following the XML syntax as a set of frames that composes the animated visualisation. Each state of the MAM (the steps performed by the algorithms being analysed) corresponds to one frame in the visualisation MVA file. Using a MVA file the MViewer builds the visualisation that the user can interacts with it.

Figure 7 The MAMView tool architecture: the MExtractor and MViewer modules internals (see online version for colours)



The MViewer module compromises the **Simulation** and the **Visualisation** components. The Simulation component reads a set of frames in the MVA file. Each frame corresponds to a set of elements, in the Euclidean visualisation space, handled by the algorithm at the corresponding step. In the MViewer module, the user can select the desired portion of the information of each frame, as well as navigates through the algorithm's execution by 'playing' the corresponding frames. The MViewer module uses the OpenGL library (Group, 2009) to manipulate the visualisation as following the user's commands.

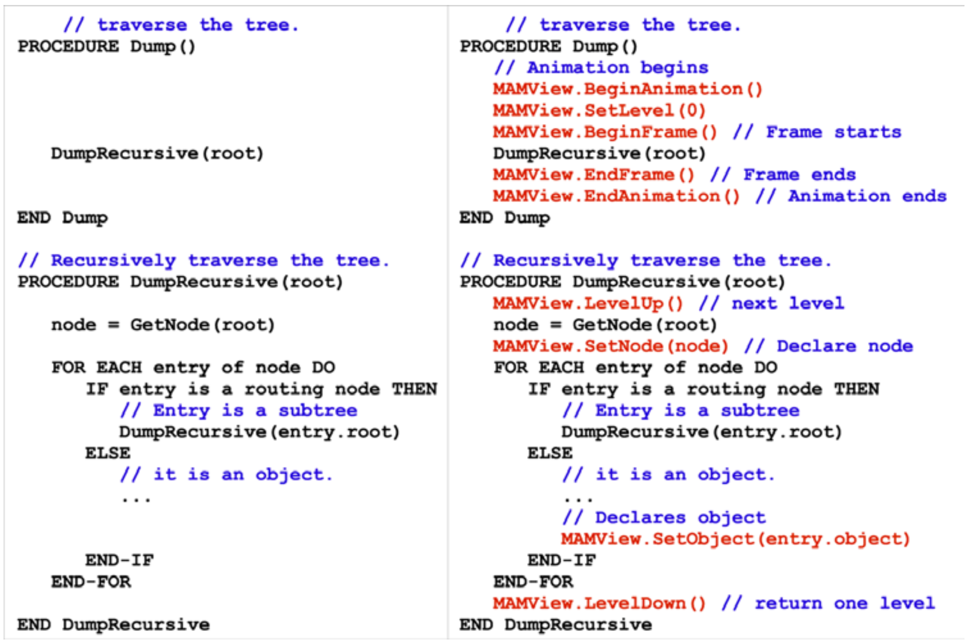
To generate a visualisation for the MViewer module in the MVA format file, it is necessary to install the MAMView API in a MAM in order to gather information about the algorithms of the specific MAM. The API installation is intended to be performed by the MAM developer, since she or he has a good knowledge on how the algorithms being analysed work. Therefore, the MExtractor module is linked as part of the application program that executes the MAM operations. The first step to probe an algorithm in a MAM is to initialise the MAMView environment. In this way, the MExtractor is ready to receive event notifications from the MAM algorithm and to generate frames stored in MVA file(s). The event notifications are generated by the API that takes as input the data (nodes, radii, elements, representatives, etc.). As the animation in an MVA file is composed of frames, calls must start and finish a frame. Therefore, the MAM developer must plan what is the meaning of each frame, defining what composes each frame and what are the events that separate them. For example, to observe the traversing over a tree, the events can be grouped in frames to be visualised as follows:

- *MAMView.BeginAnimation()* is called to initialise a new animation
- *MAMView.BeginFrame()* starts a new frame
- At each step of the loop, call *MAMView.SetFrame()* method to add data to the current frame
- Other methods, e.g., *MAMView.SetLevelDown()* and *MAMView.SetObject()*, can be used to change other status of the animation

- *MAMView.EndFrame()* is called to end a frame
- *MAMView.EndAnimation()* finishes an animation.

The main methods of the MAMView Extractor are shown in the Appendix A. Although data is collected sequentially into the frames from the execution of the algorithm being analysed, each frame treats the corresponding data as an unsorted set of descriptors. Figure 8 shows an example of API installation for the depth-first tree traversal algorithm in a MAM. The code in (a) shows the original traversal algorithm, and in (b) the augmented code with the MAMView Extractor methods. In this example, only a single frame is built with the whole tree structure with the elements indexed.

Figure 8 Installing the MAMView to an algorithm to traverse a metric tree: (a) original code and (b) code with MAMView calls (in red colour) (see online version for colours)



```
   // traverse the tree.
PROCEDURE Dump()




   DumpRecursive(root)


END Dump

// Recursively traverse the tree.
PROCEDURE DumpRecursive(root)

   node = GetNode(root)

   FOR EACH entry of node DO
      IF entry is a routing node THEN
         // Entry is a subtree
         DumpRecursive(entry.root)
      ELSE
         // it is an object.
         ...


      END-IF
   END-FOR

END DumpRecursive
```

(a)

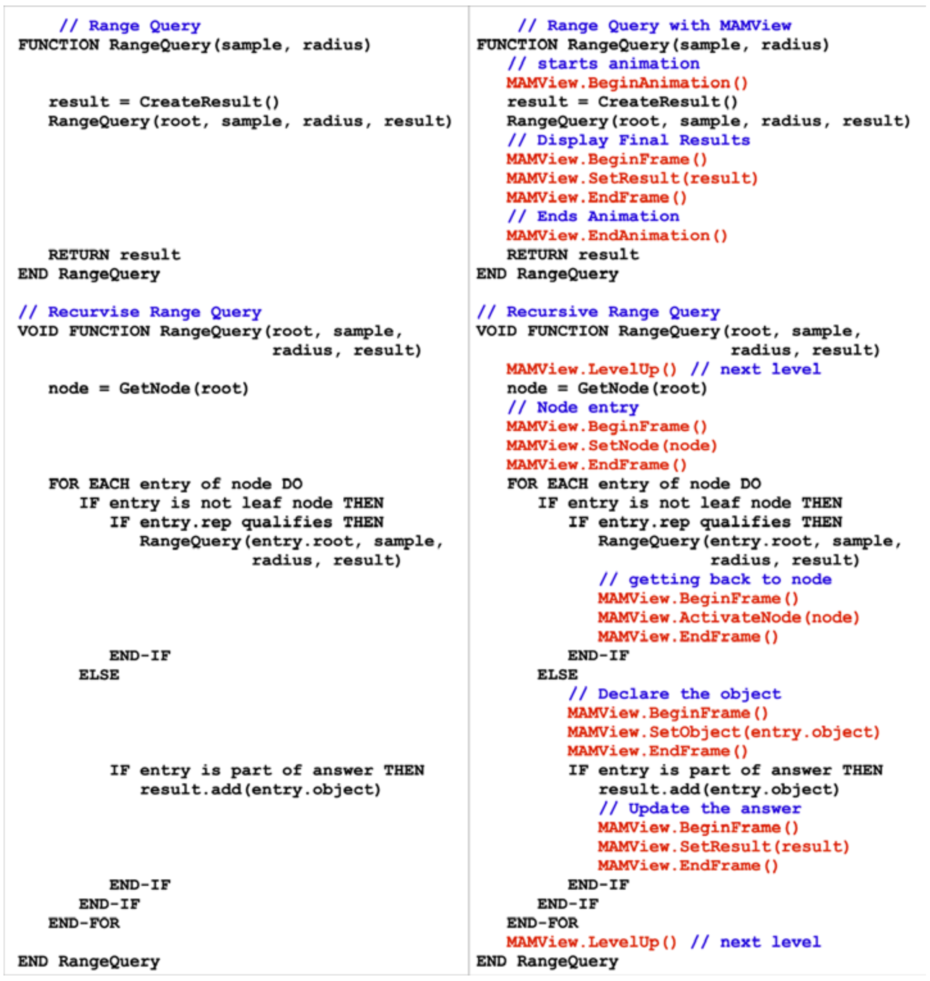
```
   // traverse the tree.
PROCEDURE Dump()
   // Animation begins
   MAMView.BeginAnimation()
   MAMView.SetLevel(0)
   MAMView.BeginFrame() // Frame starts
   DumpRecursive(root)
   MAMView.EndFrame() // Frame ends
   MAMView.EndAnimation() // Animation ends
END Dump

// Recursively traverse the tree.
PROCEDURE DumpRecursive(root)
   MAMView.LevelUp() // next level
   node = GetNode(root)
   MAMView.SetNode(node) // Declare node
   FOR EACH entry of node DO
      IF entry is a routing node THEN
         // Entry is a subtree
         DumpRecursive(entry.root)
      ELSE
         // it is an object.
         ...
         // Declares object
         MAMView.SetObject(entry.object)
      END-IF
   END-FOR
   MAMView.LevelDown() // return one level
END DumpRecursive
```

(b)

A more complex example is shown in Figure 9. In this example, the MAMView Extractor methods were installed in the range search algorithm to construct an animation of a query. In this animation, a frame is created every time a node or elements are accessed in the search process. Also, a new frame is created whenever an element is inserted in the query set. The final sequence of frames allows recording all steps of the range search processing, thus enabling the user to inspect how the nodes and elements are analysed during the query execution. Figure 9(a) and (b) show the range search algorithm before and after, respectively, the installation of the MAMView Extractor methods. This code was used to generate the visualisations for the evaluations discussed in Section 4.

Figure 9 Enabling the visualisation of a Range Query algorithm: (a) original code and (b) MAMView calls to generate an animation of the Range Query (in red) (see online version for colours)



```
   // Range Query
FUNCTION RangeQuery(sample, radius)


   result = CreateResult()
   RangeQuery(root, sample, radius, result)







   RETURN result
END RangeQuery

// Recursive Range Query
VOID FUNCTION RangeQuery(root, sample,
                         radius, result)

   node = GetNode(root)




   FOR EACH entry of node DO
      IF entry is not leaf node THEN
         IF entry.rep qualifies THEN
            RangeQuery(entry.root, sample,
                       radius, result)




         END-IF
      ELSE




         IF entry is part of answer THEN
            result.add(entry.object)




         END-IF
      END-IF
   END-FOR

END RangeQuery
```

```
   // Range Query with MAMView
FUNCTION RangeQuery(sample, radius)
   // starts animation
   MAMView.BeginAnimation()
   result = CreateResult()
   RangeQuery(root, sample, radius, result)
   // Display Final Results
   MAMView.BeginFrame()
   MAMView.SetResult(result)
   MAMView.EndFrame()
   // Ends Animation
   MAMView.EndAnimation()
   RETURN result
END RangeQuery

// Recursive Range Query
VOID FUNCTION RangeQuery(root, sample,
                         radius, result)
   MAMView.LevelUp() // next level
   node = GetNode(root)
   // Node entry
   MAMView.BeginFrame()
   MAMView.SetNode(node)
   MAMView.EndFrame()
   FOR EACH entry of node DO
      IF entry is not leaf node THEN
         IF entry.rep qualifies THEN
            RangeQuery(entry.root, sample,
                       radius, result)
            // getting back to node
            MAMView.BeginFrame()
            MAMView.ActivateNode(node)
            MAMView.EndFrame()
         END-IF
      ELSE
         // Declare the object
         MAMView.BeginFrame()
         MAMView.SetObject(entry.object)
         MAMView.EndFrame()
         IF entry is part of answer THEN
            result.add(entry.object)
            // Update the answer
            MAMView.BeginFrame()
            MAMView.SetResult(result)
            MAMView.EndFrame()
         END-IF
      END-IF
   END-FOR
   MAMView.LevelUp() // next level
END RangeQuery
```

(a) (b)

4 Evaluating the MAMView framework

The MAMView framework was evaluated on two MAM: an already developed Slim-tree, and one under development DBM-tree (Vieira et al., 2004). We evaluated two distinct features of the framework:

- for the already developed Slim-tree, the MAMView was employed to trace the algorithms and to teach new students the internal Slim-tree mechanisms to answer similarity queries, highlighting how it can be tuned to achieve good performance
- for the team developing the DBMtree, the MAMView was employed to help develop and adjust its algorithms.

The MAMView framework was implemented in C++ and OpenGL, as part of the *Arboretum library* (GBDI-ICMC-USP, 2005), a portable platform to implement, test and extend MAM. We evaluated both the Slim-tree and the DBM-tree indexing several datasets. In this paper, we show the results using five real-world datasets, which are representative to illustrate the MAMView usefulness. Figure 10 shows a visualisation of the elements for each datasets using the MViewer. Notice that the visualisation only uses the distances among the dataset elements. The presented spatial datasets are used to validate the techniques, since one can easily understand what is being shown. The four dataset are described below:

- *USCities*: A 2-dimensional dataset representing the longitude and latitude of 25,376 cities in the US and Porto Rico (US-CENSUS, 2002). Although it is a spatial dataset, only the pair wise distances among the elements, measured by the polar distance function, were used to build the visualisation. This dataset has a well known behaviour, thus it is useful to understand the visualisation process. The page size was set to 1 KByte, producing 5 and 12 levels for the Slim-tree and DBM-tree, respectively
- *BRCities*: A 2-dimensional dataset representing 5507 cities in Brazil (IBGE, 2001). This dataset was employed during the user experiments explained in Section 4.3. The data page size is 1 KByte, resulting 3 and 7 levels for the Slim-tree and DBM-tree, respectively
- *English*: 24,893 words representing a subset of words in the English dictionary. The Edit distance (Levenshtein, 1966) was employed to measure the amount of difference between two words.[1] The data page size for this dataset was set for 4 KBytes, producing 3 and 7 levels for the Slim-tree and DBM-tree, respectively
- *Eigenfaces*: A dataset describing 16 distinct features extracted from human faces using the Eigenfaces method (Turk and Pentland, 1991). The data page size is 512 bytes, resulting 6 and 9 levels for the Slim-tree and DBM-tree, respectively
- *Iris*: A 5-dimensional dataset from the UCI-KDD data repository (Hettich and Bay, 1999) describing features (sepal width and length, petal width and length, and flower type) of 3 types of Iris flowers (Setosa, Virginica and Versicolor), each type with 50 elements. The page size was set to 256 bytes to force 3 and 8 levels for the Slim-tree and DBM-tree, respectively.

Figure 11 presents a visualisation of the *USCities* dataset indexed by the DBMtree using the MViewer. The elements in each level of the structure are shown in different colours. The colour pattern is shown at the top right (arrow number 3) of Figure 11, and it is the same for all visualisation frames. The graphical elements employed by the MViewer for each tree level are the ones shown in Figure 5. The navigation and graphical interaction controls are at the frame top left (arrows number 1 and 2). The messages posted by the MExtractor to inform about the settings and events occurred during the MAM algorithm execution are shown at the bottom of the MViewer window (arrow number 4).

Figure 10 Visualisation, using the MViewer, of elements of all five datasets used in the experiments (from left to right): *USCities*, *BRCities* (first row), *English*, *Eigenfaces* and *Iris* (second row) (see online version for colours)

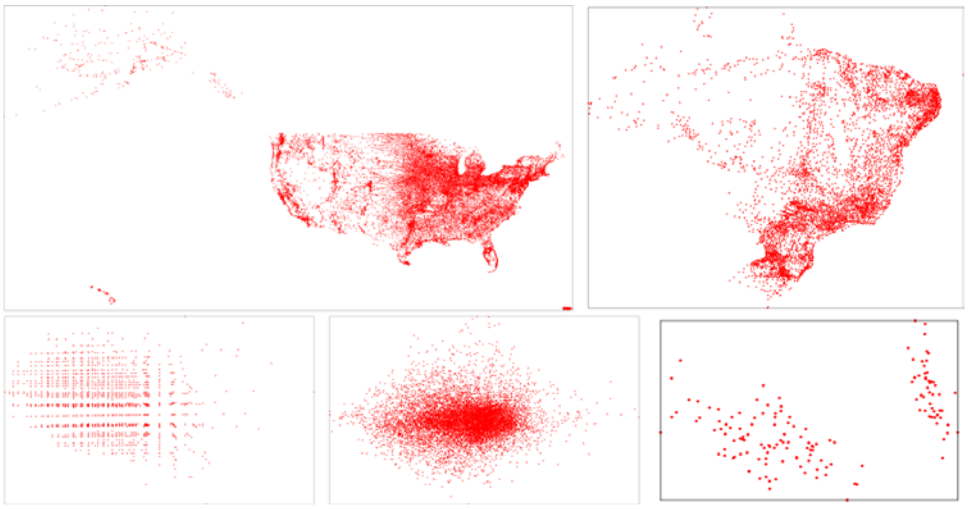
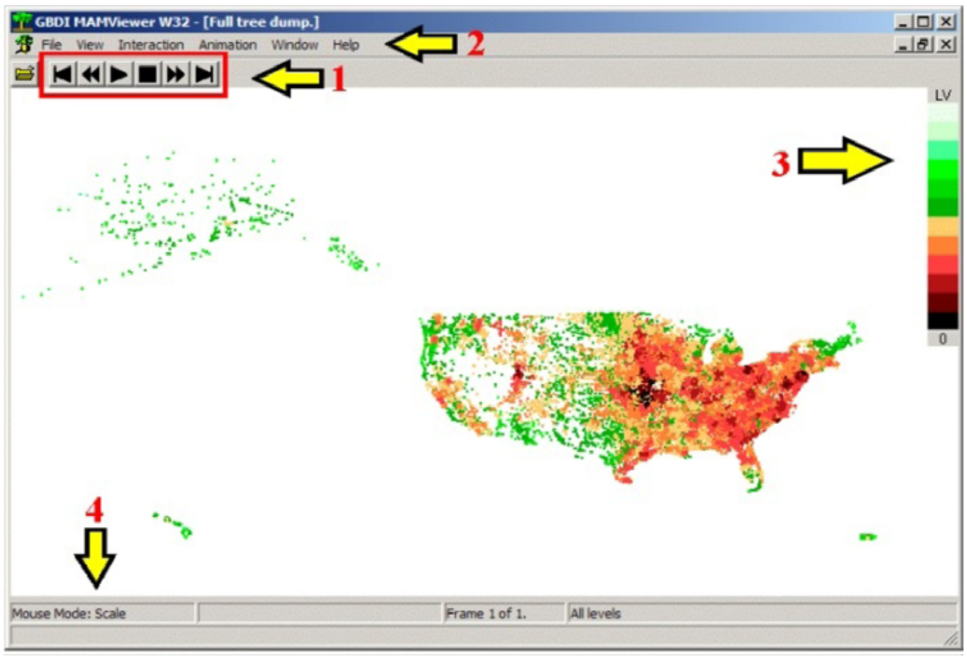
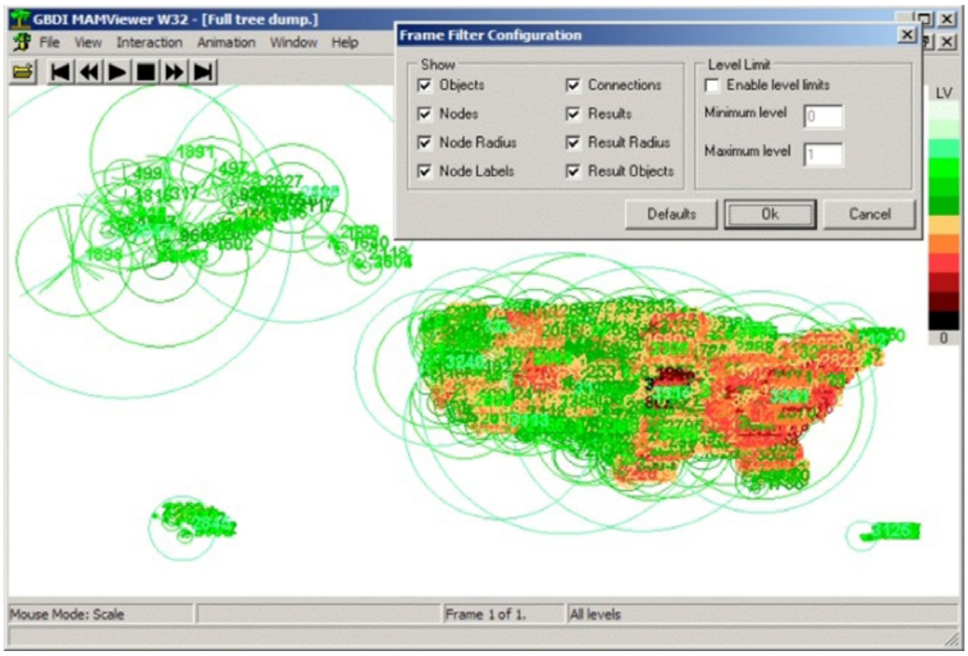


Figure 11 Snapshots of the MViewer presenting the *USCities* dataset indexed by the DBM-tree: (a) only the data elements are shown. Arrow 1 points to the animation interaction controls, arrow 2 points to the visualisation controls, arrow 3 points to the colour pallet, and arrow 4 points to the message fields and (b) the elements and regions covered by each node are presented (see online version for colours)


GBDI MAMViewer W32 - [Full tree dump.]
File View Interaction Animation Window Help
2
1
3
4
LV
0
Mouse Mode: Scale
Frame 1 of 1.
All levels

(a)

Figure 11 Snapshots of the MViewer presenting the *USCities* dataset indexed by the DBM-tree: (a) only the data elements are shown. Arrow 1 points to the animation interaction controls, arrow 2 points to the visualisation controls, arrow 3 points to the colour pallet, and arrow 4 points to the message fields and (b) the elements and regions covered by each node are presented (see online version for colours) (continued)



GBDI MAMViewer W32 - [Full tree dump.]

Frame Filter Configuration

Show
- [✓] Objects
- [✓] Connections
- [✓] Nodes
- [✓] Results
- [✓] Node Radius
- [✓] Result Radius
- [✓] Node Labels
- [✓] Result Objects

Level Limit
- [ ] Enable level limits
- Minimum level 0
- Maximum level 1

Mouse Mode: Scale | Frame 1 of 1. | All levels

(b)

The animated MViewer visualisations allow users to navigate through the execution of the algorithm, continuously or frame by frame, forward or backward. The MViewer tool allows users to select to visualise only the elements indexed in the tree (no representatives, ids or radii), elements in any levels of the tree, node representatives, radii, element identifiers, query object, query answer, or any combination of the previous options. For an animation of a visualisation, composed of a set of frames, users can interact with each frame of the animation as well as a set of frames. In this last animation feature, users can visualise only a specific number of frames while playing the animation, defined here as the 'tail' feature. Another special effect that users can enable while playing an animation is the 'fading' feature, where old frames regarding the current frame being visualised are incrementally faded. This last feature provides users to focus only on the last frames in an animation, superimposing a selected number of frames in each visualisation with different dim effects.

The MViewer tool also allows users to interactively rotate, translate or scale the visualisation to inspect specific regions of the visualisation. All these features are very important to visualise large datasets where there is potential for cluttering and occlusion in the visualisation.

4.1 A visual dump for MAM

Exploring the organisation of elements indexed by a MAM is useful to understand the behaviour of the structure, as well as giving hints on how to improve the structure performance. Hence, users can check whether the algorithms are working properly, the parameters are set correctly, or where the bottlenecks in the structure are located. Figure 12 shows four different snapshots for a visual dump of the Slim-tree indexing the *Iris* dataset. For sake of simplicity, a single frame is used for a visualisation of a dump for a particular MAM. However, depending on the size and complexity of the dataset, different frames can be used allowing observing small details of a particular MAM. In Figure 12(a) only the elements (represented with dots) indexed and the representatives of nodes (7-points stars along with their id labels) are shown. Figure 12(b) adds to Figure 12(a) the links between elements and their respectively representatives. Figure 12(c) adds the radius for each node and Figure 12(d) shows another perspective of the same information as Figure 12(c). Figure 13 shows another visual dump but now for the *English* dataset. Figure 13(a) shows elements, representatives, radius and id for representatives, and Figure 13(b) only the elements indexed by the Slim-tree.

Figure 12 Visual dump of the *Iris* dataset: (a) elements with representatives and ids; (b) connections between elements and representatives; (c) node radii and (d) a different perspective of (c) (see online version for colours)

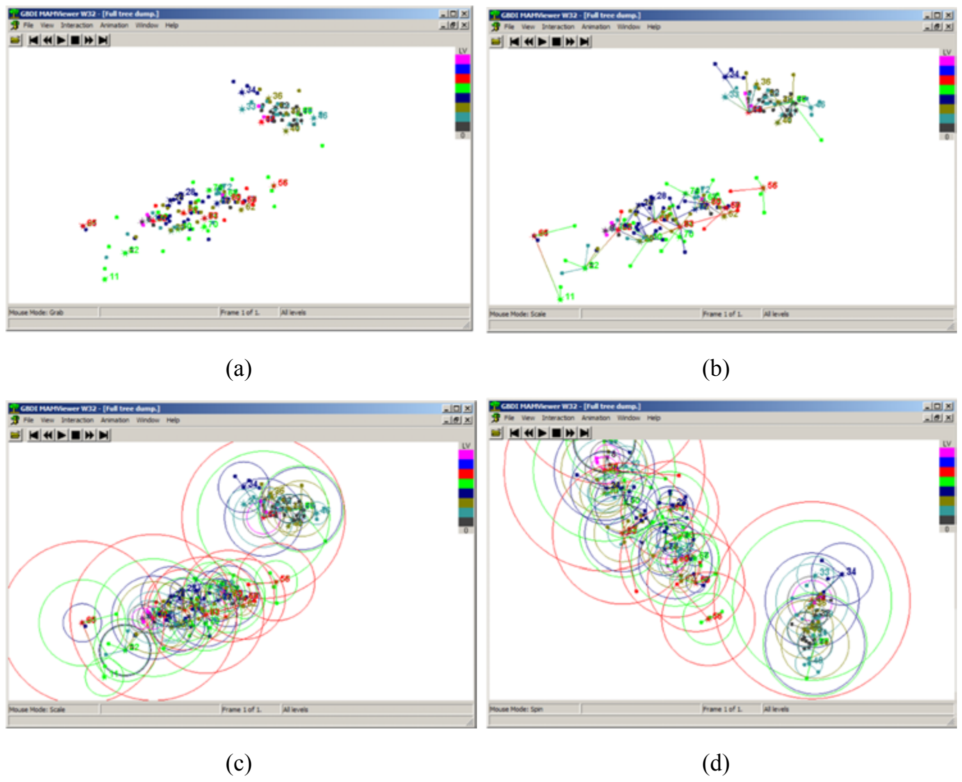
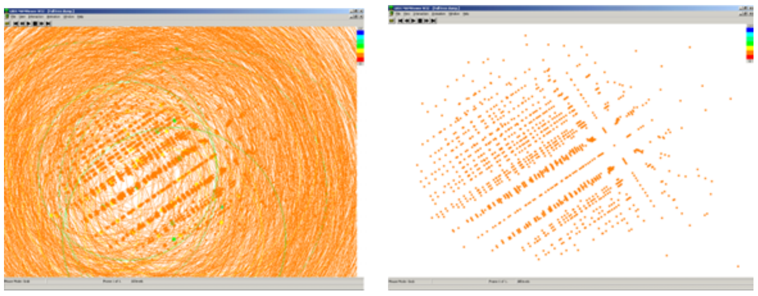
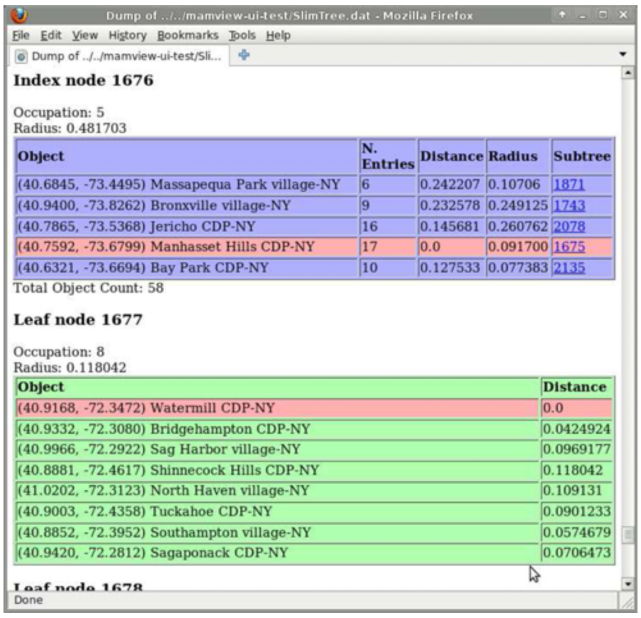


Figure 13 Three snapshots of the MViewer with the English dataset: (a) elements and node representatives and (b) only the elements indexed by a Slim-tree (see online version for colours)



For comparison purposes, we also implemented a *Hypertext Dump* for MAM. This *Hypertext Dump* is improved by hyperlinks to allow users to navigate from a node to its subtrees. Figure 14 presents a screenshot of a *Hypertext Dump* for a Slim-tree indexing the *USCities* dataset. This figure shows both an index (node 1676 with 5 representatives) and leaf (node 1677 with 8 entries) nodes. Index nodes have links to their subtrees, and leaf nodes have the elements and distances to their node representative. In the experiments performed with users (see Section 4.2), we compared the visualisation provided by the MViewer tool against the *Hypertext Dump* for the same MAM (Figure 14).

Figure 14 A *Hypertext Dump* with hyperlinks of the tree shown in Figure 11 for the *USCities* dataset. The index (routing) nodes have links to their subtrees. The elements indexed by the MAM are all stored in the leaf nodes (see online version for colours)


Dump of ../../mamview-ui-test/SlimTree.dat - Mozilla Firefox
File Edit View History Bookmarks Tools Help
Dump of ../../mamview-ui-test/Sli...
Done

Index node 1676

Occupation: 5
Radius: 0.481703

Object	N. Entries	Distance	Radius	Subtree
(40.6845, -73.4495) Massapequa Park village-NY	6	0.242207	0.10706	1871
(40.9400, -73.8262) Bronxville village-NY	9	0.232578	0.249125	1743
(40.7865, -73.5368) Jericho CDP-NY	16	0.145681	0.260762	2078
(40.7592, -73.6799) Manhasset Hills CDP-NY	17	0.0	0.091700	1675
(40.6321, -73.6694) Bay Park CDP-NY	10	0.127533	0.077383	2135

Total Object Count: 58

Leaf node 1677

Occupation: 8
Radius: 0.118042

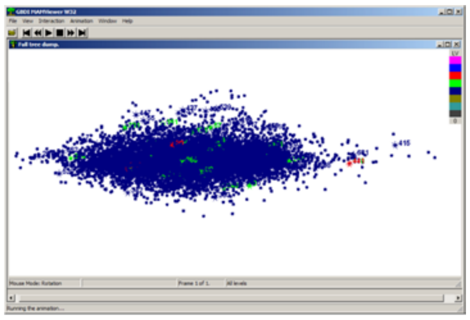
Object	Distance
(40.9168, -72.3472) Watermill CDP-NY	0.0
(40.9332, -72.3080) Bridgehampton CDP-NY	0.0424924
(40.9966, -72.2922) Sag Harbor village-NY	0.0969177
(40.8881, -72.4617) Shinnecock Hills CDP-NY	0.118042
(41.0202, -72.3123) North Haven village-NY	0.109131
(40.9003, -72.4358) Tuckahoe CDP-NY	0.0901233
(40.8852, -72.3952) Southampton village-NY	0.0574679
(40.9420, -72.2812) Sagaponack CDP-NY	0.0706473

Leaf node 1678

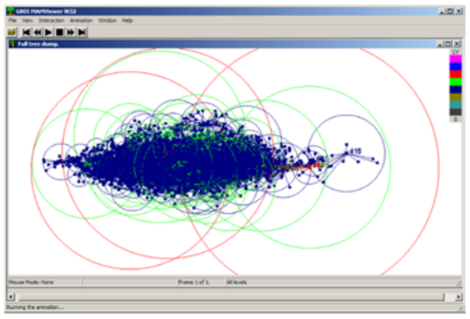
4.2 Evaluating the applicability of the MAMView framework

Processing similarity queries over multimedia data can be very time consuming. Thus, MAM developers strive to optimise the construction and query algorithms as much as possible. As MViewer allows visual inspections and analysis of the changes and navigation steps performed by the algorithms, developers can literally 'see' where to put efforts in order to improve and/or tune the structure or the query algorithms. Figure 15 shows global views of some snapshots of the Slim-tree indexing the *Eigenfaces* dataset, from a 230-frame animation generated by the execution of a RQ. Figure 15(a) shows the dataset with the representatives and their identifiers, and in Figure 15(b) the node radii are included; Figure 15(c) shows the 1st frame, out of 230 frames, for the RQ centre drawn as the 5-point black star and the three subtrees in the root node represented by representatives 585, 30 and 441; Figure 15(d) shows the 2nd frame where subtree represented by the representative 585 is evaluated. Different colours represent different levels in the subtree; Figure 15(e) shows frame 13rd when the range algorithm reaches a leaf node (node 90 at level 3). This particular animation was zoomed and rotated; Figure 15(f) shows frame 18th where index node (node 71) is evaluated; Figure 15(g) shows the same frame as Figure 15(f), but focusing on index node 71. Cluttering in the visualisation makes it difficult to recognise where the processing focus is, but the 'tail' resource can spot the important parts of the visualisation. This resource is fading previous frames to allow the user to focus only on the last 5 frames of the animation; Figure 15(h) shows the 229th frame where the algorithm finished evaluating subtree 30. The result of this particular range query is null.

Figure 15 Some snapshots for a Range Query of a Slim-tree indexing the Eigenfaces dataset: (a) elements and representatives; (b) elements, representatives, radii and ids; (c) shows the 1st frame, out of 230 frames, for a Range Query with query object represented by a 5-point black star (only the tree subtrees, represented by their representatives and radii, in the root node; (d) shows the 2nd frame where one subtree is being explored to evaluate the Range Query; (e) shows the 13rd frame where the leaf node 90 in the third level is explored; (f) shows the 18th frame where the index node 71 in the second level is being evaluated; (g) shows the same frame as (f), but here only the last 5 frames are displayed using the 'tail' feature (tail with size 5) and (h) shows the 229th frame where three subtrees (585, 441 and 30) are being evaluated (see online version for colours)

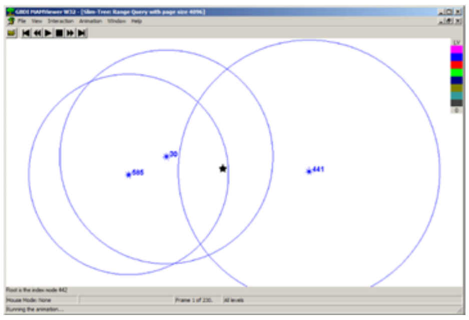


(a)

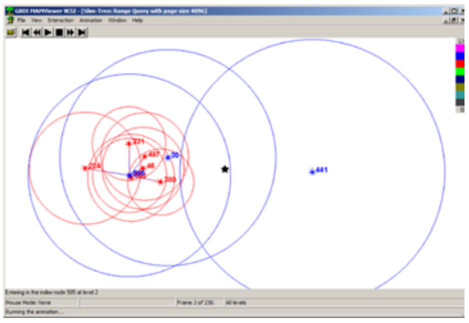


(b)

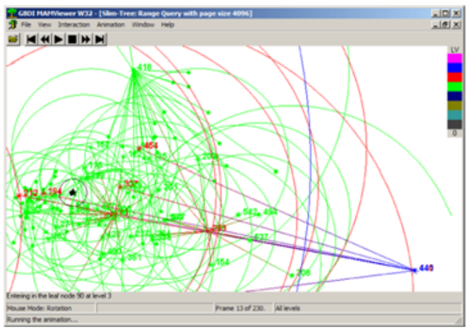
Figure 15 Some snapshots for a Range Query of a Slim-tree indexing the *Eigenfaces* dataset: (a) elements and representatives; (b) elements, representatives, radii and ids; (c) shows the 1st frame, out of 230 frames, for a Range Query with query object represented by a 5-point black star (only the tree subtrees, represented by their representatives and radii, in the root node; (d) shows the 2nd frame where one subtree is being explored to evaluate the Range Query; (e) shows the 13rd frame where the leaf node 90 in the third level is explored; (f) shows the 18th frame where the index node 71 in the second level is being evaluated; (g) shows the same frame as (f), but here only the last 5 frames are displayed using the 'tail' feature (tail with size 5) and (h) shows the 229th frame where three subtrees (585, 441 and 30) are being evaluated (see online version for colours) (continued)


GBDI MAMViewer W32 - [Slim-Tree: Range Query with page size 4096]
585
30
441
Root is the index node 442
Mouse Mode: None
Frame 1 of 230.
All levels
Running the animation...

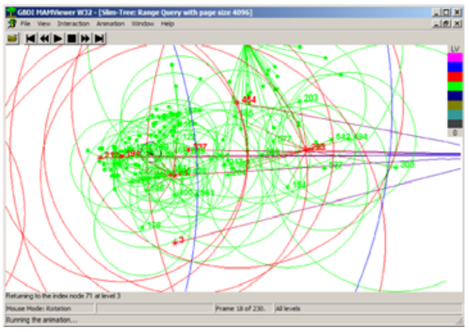
(c)


GBDI MAMViewer W32 - [Slim-Tree: Range Query with page size 4096]
585
30
441
Entering in the index node 585 at level 2
Mouse Mode: None
Frame 2 of 230.
All levels
Running the animation...

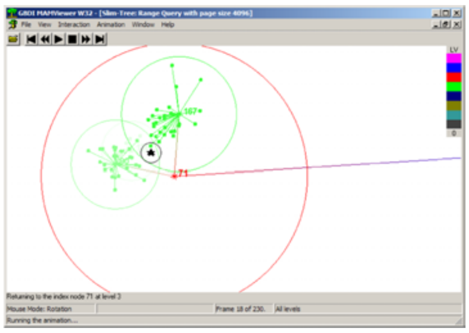
(d)


GBDI MAMViewer W32 - [Slim-Tree: Range Query with page size 4096]
418
454
154
445
Entering in the leaf node 90 at level 3
Mouse Mode: Rotation
Frame 13 of 230.
All levels
Running the animation...

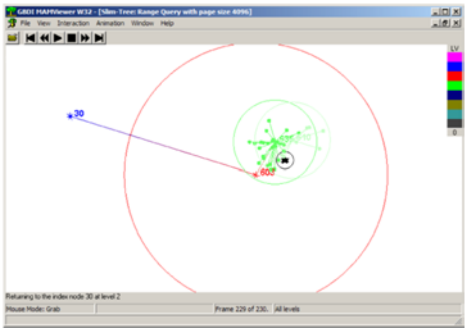
(e)


GBDI MAMViewer W32 - [Slim-Tree: Range Query with page size 4096]
454
203
154
Returning to the index node 71 at level 3
Mouse Mode: Rotation
Frame 18 of 230.
All levels
Running the animation...

(f)


GBDI MAMViewer W32 - [Slim-Tree: Range Query with page size 4096]
71
Returning to the index node 71 at level 3
Mouse Mode: Rotation
Frame 18 of 230.
All levels
Running the animation...

(g)


GBDI MAMViewer W32 - [Slim-Tree: Range Query with page size 4096]
30
Returning to the index node 30 at level 2
Mouse Mode: Grab
Frame 229 of 230.
All levels

(h)

4.3 Evaluating MAMView usefulness to understand similarity queries

MAM are not as easy to understand as the spatial and order-based structures, so students usually have a hard time understanding them. We claim that the MAMView framework can facilitate users to understand MAM, improving the perception of the executed algorithms. Hence, here we present some experiments performed with students to evaluate our claim. We invited 22 graduate students from a database class to participate in the experiments. All those students were introduced to the basic concepts related to MAM and similarity queries. Furthermore, they were given practical exercises to fix these new concepts in class before introduced to the MAMView framework. All students also attended to a presentation where the MAMView framework was introduced, as well as the *Hypertext Dump*.

The experiments aim at comparing the adequacy of the animated visualisation given by the MViewer tool (Figure 11(a)) and the *Hypertext Dump* (Figure 14). The tasks that all the students needed to perform were to trace the execution of a RQ using the Slim-tree with the *BRCities* dataset. Such dataset was chosen in order to make it easier to compute the required distances between elements and the query centre and because all students were more familiar with it (country where all the students were originated from). All participants were asked to trace the same RQ, located in the vicinity of the students' university. In the end, they were asked to answer the following three questions:

1 What is the sequence of nodes accessed to answer the proposed query?

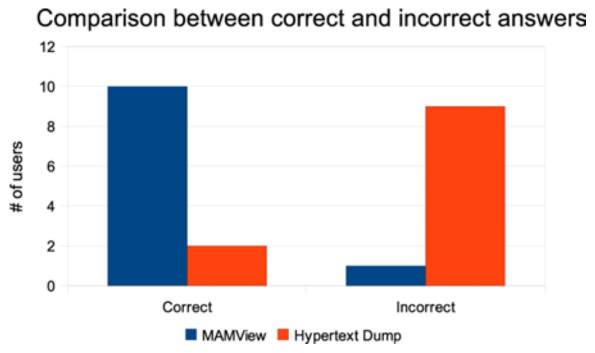
2 How long did it take to answer Question 1?

3 How difficult was to find the answer?

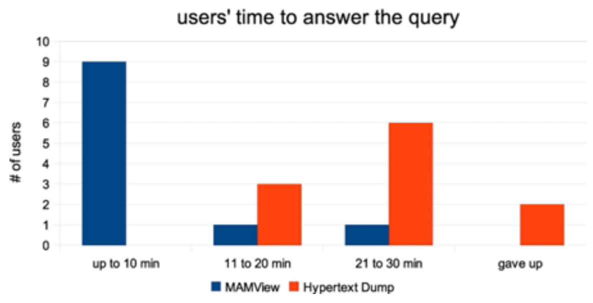
The participants were randomly split into two groups to answer the previous three questions: group A employed only the MViewer tool; and group B employed only the *Hypertext Dump* to evaluate the RQ. The answers of both groups are summarised in Figure 16. The first question spots how many participants properly traced the RQ algorithm. Figure 16(a) shows that 10 out of the 11 users in group A (with MViewer) delivered correct answers, while only two users, out of 11, in group B (with *Hypertext Dump*) achieved the correct answer. Question 2 tells us which tool allows users to get the results faster. Figure 16(b) shows that the majority of users in group A spent 10 min or less to answer the RQ, while users in group B spent more than 10 min. Two users in group B gave up since it took them more than 30 min to answer the first question.

In the next step we asked each group to answer the same previous three questions again, but changing the tool employed, i.e., group A and B employed the *Hypertext Dump* and MViewer, respectively. In this way we could evaluate question 3, measuring the users' satisfaction employing both tools. Figure 16(c) shows the results for question 3. As we can see, the students considered the MViewer tool the easiest to understand and answer the first two questions. We also asked the participants about which tool they think is the most appropriate to learn and understand the similarity query processing. 20 students said that the MViewer was better than the *Hypertext Dump*.

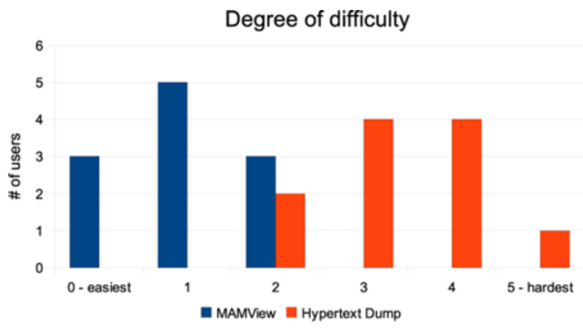
Figure 16 Comparing MViewer and *Hypertext Dump* regarding: (a) number of users that delivered correct answers; (b) total time spent to give the final answer of the proposed Range Query and (c) level of difficulty to follow the execution using each tool (see online version for colours)



(a)



(b)



(c)

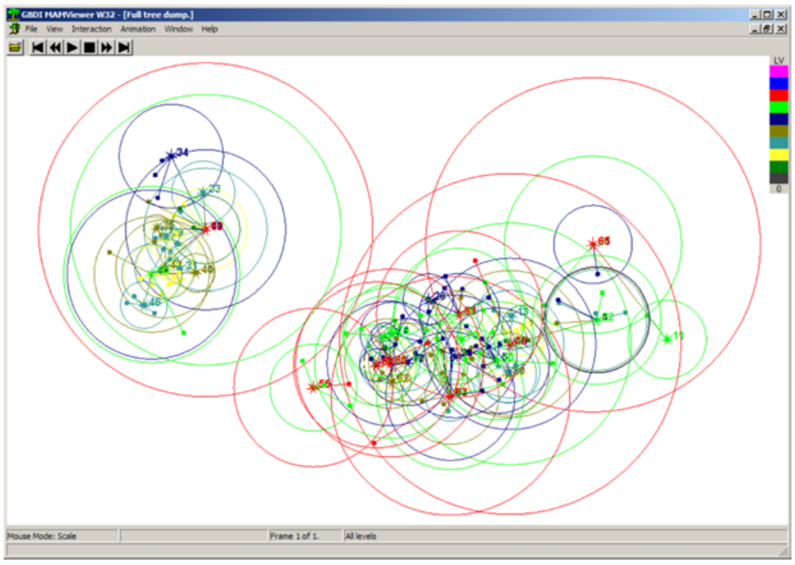
4.4 *Evaluating MAMView usefulness to help developers*

It is quite challenging to setup an experiment to evaluate the MAMView ability to help developers in building new algorithms for MAM. One challenging aspect of this is that we would need several groups of developers to measure the spending time on the development process, as well as to measure the correctness and novelty of the new algorithms. Luckily, we were able to employ MAMView on the development of the DBM-tree. As it turned out, MAMView was fundamental to drill-down the DBM-tree algorithms, including the tree construction and the similarity query algorithms. Therefore, we will briefly discuss, as a case study, the use of MAMView on the development of the DBM-tree.

The *Density-Based Metric tree* (DBM-tree) (Vieira et al., 2004) is a dynamic MAM that differs from the Slim-tree by building deeper subtrees at denser data regions. It achieves better query performance than other height-balanced MAM for datasets with skewed distributions. The total cost in developing the DBM-tree considering time and human effort was *extremely* reduced after introducing the MAMView tool, as related by the developers involved in the project. Each new proposed approach implemented and tested without the use of the MAMView framework took days to understand what was happening with the structure since the only way to measure the new approach was to perform several performance evaluations. After the introduction of the MAMView framework, the developers involved in the project of the DBM-tree related that the design/coding/debug cycle was accelerated from several days to a few hours.

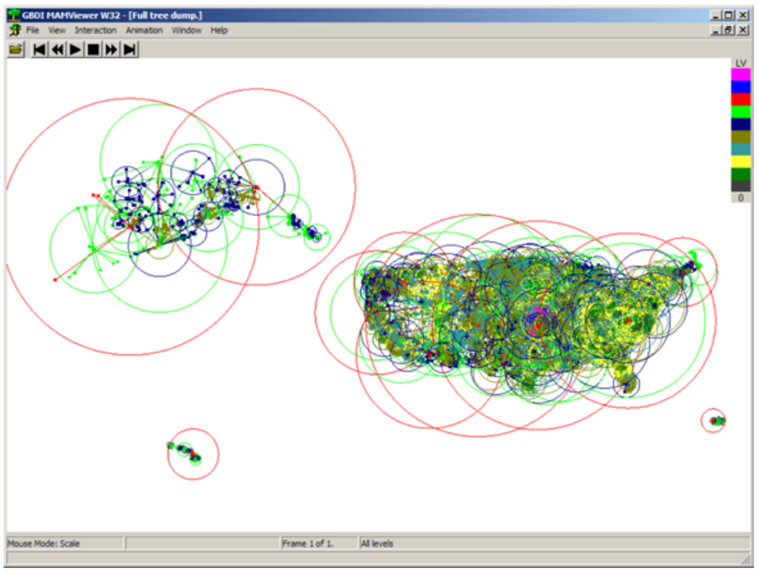
The DBM-tree was conceived after identifying several bottlenecks in the Slim-tree using the MViewer tool. As a result, the DBM-tree spotted several problems not known beforehand, whose solutions enabled improvements of up to twice over the Slim-tree, regarding running time, number of disc accesses and distance calculations. There were many choices on how to define policies to split the nodes of the DBM-tree, to choose subtrees to insert new elements, to choose representatives, among other algorithms. The MAMView was fundamental to narrow down a subset of options to test, highlighting the fact that – as often occurs regarding metric spaces – those intuitively seeming as the most promising, were in fact bad options. Therefore, the visualisation clues provided by the MViewer tool were fundamental in the process of developing the DBM-tree. Figure 17 shows 2 snapshots of the DBM-tree indexing the Iris and the *USCities* datasets.

Figure 17 (a) The DBM-tree indexing the Iris and (b) the *USCities* datasets (see online version for colours)


GBDI MAMViewer W32 - [Full tree dump.]
File View Interaction Animation Window Help
LV
0
Mouse Mode: Scale
Frame 1 of 1.
All levels

(a)

Figure 17 (a) The DBM-tree indexing the Iris and (b) the *USCities* datasets (see online version for colours) (continued)


GBDI MAMViewer W32 - [Full tree dump.]
File View Interaction Animation Window Help
LV
0
Mouse Mode: Scale
Frame 1 of 1.
All levels

(b)

5 Conclusions

This paper presented a framework called MAMView, which assists in the analysis of MAM, including their insertion, querying and optimisation algorithms. This framework enables the visualisation of the data indexed by a MAM to help users in understanding the behaviour of MAM. The MAMView framework is the first visualisation tool allowing users to interactively explore the MAM behaviour during query processing. It is also a powerful tool to help drilling-down new algorithms for MAM. MAMView helps spotting potential bottlenecks and problems in already developed MAM, as well as to help MAM developers to come up with more efficient indexing approaches and optimisation techniques to answer similarity queries.

The visual attributes of the animations can be changed on-the-fly in the MViewer tool. The MViewer is the component of the MAMView framework responsible in presenting the visualisation stored in the MVA format files. This format complies with the XML syntax, allowing users to share the files among other users/developers. Users can interact with the presentation though the MViewer tool. The visualisation can have several frames, each of which containing part of the animation. Users can interact with a single frame or by 'playing' an animation composed by a set of frames.

The MAMView framework is implemented as part of the *Arboretum Library* of MAM. However, the MAMView framework can be employed to visualise operations performed by any MAM.

In the experimental evaluation graduate students were asked to evaluate how easy, or difficult, was to evaluate a range query using the MAMView framework and a *Hypertextual Dump* representation of the same data indexed by the Slim-tree.

The results reported in this paper revels that users understood the concepts of MAM faster and easier using the MAMView framework. We also reported the developers' experiences using the MAMView framework to detect bottlenecks in the Slim-tree and develop a new MAM.

Acknowledgements

This work was supported by FAPESP (São Paulo State Research Foundation) under grants number 01/11987-3, 01/12536-5 and 02/07318-1, and CNPq (Brazilian National Council for Supporting Research) under grants 52.1685/98-6, 52.1267/96-0 and 860.068/00-7.

References

Baker, R., Boilen, M., Goodrich, M.T., Tamassia, R. and Stibel, B.A. (1999) 'Testers and visualizers for teaching data structures', *SIGCSE '99: The Proc. of the thirtieth SIGCSE Technical Symposium on Computer Science Education*, ACM, New Orleans, Louisiana, USA, pp.261–265.

Bayer, R. and McCreight, E.M. (1972) 'Organization and maintenance of large ordered indexes', *Acta Informatica*, Vol. 1, No. 3, pp.173–189.

Beckmann, N., Kriegel, H-P., Schneider, R. and Seeger, B. (1990) 'The R*-tree: an efficient and robust access method for points and rectangles', *Proc. of the ACM SIGMOD Int'l Conf. on Management of Data*, ACM, Atlantic City, New Jersey, USA, pp.322–331.

Brabec, F., Samet, H. and Yilmaz, C. (2003) 'Vasco: visualizing and animating spatial constructs and operations', *Proc. of the ACM Symposium on Computational Geometry*, ACM, San Diego, California, USA, pp.374–375.

Chalmers, M. and Chitson, P. (1992) 'Bead: explorations in information visualization', *15th Annual Intl ACM SIGIR Conf. on Research and Development in Information Retrieval*, ACM, Copenhagen, Denmark, pp.330–337.

Ciaccia, P., Patella, M. and Zezula, P. (1997) 'M-tree: an efficient access method for similarity search in metric spaces', *Proc. of the Int'l Conf. on Very Large Data Bases (VLDB)*, Morgan Kaufmann Publishers Inc., Athens, Greece, pp.426–435.

Comer, D. (1979) 'The ubiquitous B-tree', *ACM Computing Surveys (CSUR)*, Vol. 11, No. 2, pp.121–137.

Elmqvist, N., Dragicevic, P. and Fekete, J-D. (2008) 'Rolling the dice: multidimensional visual exploration using scatterplot matrix navigation', *IEEE Transactions on Visualization and Computer Graphics*, Vol. 14, No. 6, pp.1139–1148.

Faloutsos, C. and Lin, K-I. (1995) 'FastMap: a fast algorithm for indexing, datamining and visualization of traditional and multimedia datasets', *Proc. of the ACM SIGMOD Int'l Conf. on Management of Data*, ACM, San Jose, California, USA, pp.163–174.

GBDI-ICMC-USP (2005) GBDI Arboretum Library, Mathematics and Computer Science Institute, University of São Paulo at São Carlos, São Carlos, SP [gbdi.icmc.usp.br/arboretum].

Group, K. (2009) *OpenGL (Open Graphics Library)* [www.opengl.org].

Guttman, A. (1984) 'R-tree: a dynamic index structure for spatial searching', *Proc. of the ACM SIGMOD Int'l Conf. on Management of Data*, ACM, Boston, Massachusetts, pp.47–57.

Hadjieleftheriou, M., Hoel, E.G. and Tsotras, V.J. (2005) 'Sail: a spatial index library for efficient application integration', *GeoInformatica.*, Vol. 9, No. 4, pp.367–389.

Hellerstein, J.M., Naughton, J.F. and Pfeffer, A. (1995) 'Generalized search trees for database systems', *Proc. of the Int'l Conf. on Very Large Data Bases (VLDB)*, pp.562–573.

Hettich, S. and Bay, S.D. (1999) *The UCI KDD Archive*, University of California, Department of Information and Computer Science, [http://kdd.ics.uci.edu], Irvine, CA.

Hristescu, G. and Farach-Colton, M. (1999) *Cluster-Preserving Embedding of Proteins*, Technical Report 99-50, DIMACS, October, Piscataway, New Jersey, USA, p.20.

IBGE (2001) Brazilian Institute of Geography and Statistics, [www.ibge.gov.br].

Keim, D. and Kriegel, H-P. (1994) 'Visdb: database exploration using multidimensional visualization', *IEEE Computer Graphics and Applications*, Vol. 14, No. 5, pp.40–49.

Keim, D.A. (2001) 'Visual exploration of large data sets', *Communications of the ACM*, Vol. 44, No. 8, pp.38–44.

Kruskal, J.B. and Wish, M. (1978) *Multidimensional Scaling*, Sage Publications, Beverly Hills, USA.

Levenshtein, V.I. (1966) 'Binary codes capable of correcting deletions, insertions, and reversals', *Soviet Physics Doklady*, Vol. 10, No. 8, pp.707–710.

Livny, M., Ramakrishnan, R., Beyer, K.S., Chen, G., Donjerkovic, D., Lawande, S., Myllymaki, J. and Wenger, R.K. (1997) 'Devise: integrated querying and visualization of large datasets', *Proc. of the ACM SIGMOD Int'l Conf. on Management of Data*, pp.301–312.

Miller, N., Hetzler, B., Nakamura, G. and Whitney, P. (1997) 'The need for metrics in visual information analysis', *Workshop on New Paradigms in Information Visualization and Manipulation (NIPV '97)*, ACM Press, pp.24–28.

Morrison, A. and Chalmers, M. (2004) 'A pivot-based routine for improved parent-finding in hybrid MDS', *Information Visualization*, Vol. 3, No. 2, pp.109–122.

Sellis, T.K., Roussopoulos, N. and Faloutsos, C. (1987) 'The R^+-tree: a dynamic index for multi-dimensional objects', *Proc. of the Int'l Conf. on Very Large Data Bases (VLDB)*, Morgan Kaufmann, pp.507–518.

Shah, M.A., Kornacker, M. and Hellerstein, J.M. (1999) 'Amdb: a visual access method development tool', *User Interfaces to Data Intensive Systems*, pp.130–140.

Traina Jr., C., Traina, A.J.M., Seeger, B. and Faloutsos, C. (2000) 'Slim-trees: high performance metric trees minimizing overlap between nodes', *Int'l Conf. on Extending Database Technology (EDBT), volume 1777 of LNCS*, Springer-Verlag, Konstanz, Germany, pp.51–65.

Turk, M. and Pentland, A. (1991) 'Eigenfaces for recognition', *Journal of Cognitive Neuroscience*, Vol. 3, No. 1, pp.71–86.

US-CENSUS (2002) The US Tiger Dataset [http://tiger.census.gov].

Vieira, M.R., Traina Jr., C., Traina, A.J.M. and Chino, F.J.T. (2004) 'DBM-tree: a dynamic metric access method sensitive to local density data', *Brazilian Symposium on Databases (SBBD)*, SBC, Brasília, Distrito Federal, Brazil, pp.163–177.

Wang, J.T-L., Wang, X., Lin, K-I., Shasha, D., Shapiro, B.A. and Zhang, K. (1999) 'Evaluating a class of distance-mapping algorithms for data mining and clustering', *Proc. of the ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining*, ACM, San Diego, California, USA, pp.307–311.

Ware, C. (2000) *Information Visualization: Perception for Design*, The Morgan Kaufmann Series in Interative Technologies, Morgan Kaufmann Publishers, San Francisco, CA, USA.

Note

[1]The Edit distance, or Levenshtein distance, between two strings is defined as the minimum number of edits (insertion, deletion, or substitution of a single character) needed to transform one string into the other.

Appendix A

The main class in the MAMView Extractor is the *stMAMViewExtractor*. It implements all the extractor resources, except the Distance Mapper, which uses the same distance function used by the attached MAM. Its methods are:

Table 1 Main methods of the *stMAMViewExtractor* class

Init(samples)	Initialises the extractor. Parameter *samples* has a set of elements employed to initialise the *stFastMapper*
SetOutputDir(outputDir)	Specifies the directory path where the resulting files will be written to
BeginAnimation(title, comment)	Triggers the extractor to create a new animation file and start the animation. *title* and *comment* are attributes displayed in the MViewer
EndAnimation()	Triggers the end of an animation
BeginFrame(comment)	Starts a new frame. *comment* has the text content displayed in the animation
EndFrame()	Ends a frame
SetNode(nodeID, reps, radii, n, parent, type, active)	Creates a new node or changes the attributes of node *nodeID*
EnableNode(nodeID)	Enables node with identifier *nodeID*
SetObject(obj, parent, active)	Creates a new element or changes the attributes of element with identifier *nodeID*
SetResult(queryObj, result, k, radius)	Creates a result set
SetSample(queryObj)	Set the query object for a query
SetQueryInfo(k, radius)	Set the *k* and *radius* attributes of a query
SetLevel(level)	Creates a new level in the visualisation
GetLevel()	Gets the current level
LevelUp()	Increases the level by one
LevelDown()	Decreases the level by one