MAMView: A Framework for Visualization of Metric Trees

Marcos R. Vieira[1], Fabio J. T. Chino[2], Caetano Traina Jr.[2], Agma J. M. Traina[2]

[1]University of California, Riverside, CA – USA

[2]University of São Paulo, São Carlos, SP – Brazil

mvieira@cs.ucr.edu, {chino, caetano, agma}@icmc.usp.br

Abstract. *In this demonstration, we present the MAMView framework for exploring and understanding metric trees. Users and developers of metric trees can use the MAMView framework to visually explore operations and data indexed in metric trees. Understanding how these structures operate and organize the indexed data is an important task for both users and developers. MAMView was developed as a practical tool that has been successfully applied to study new and existing metric trees.*

1. Introduction

Nowadays, many applications use Database Management Systems (DBMS) to store, manage and query complex data types (e.g. multimedia data, time series, spatial data, documents). In general, these complex data types are searched by *similarity*, such as *k*-nearest neighbor (*kNN*), e.g. "*find the 5 most similar images to the query image*", and range queries (*Rq*), e.g. "*select all proteins that are similar to the query protein by up to 5 purine bases*". Metric trees, or Metric Access Methods (MAM), e.g. Slim-tree [Jr et al. 2002] and DBM-tree [Vieira et al. 2004], are indexes that can be employed to make similarity queries more efficient. Indexing and querying operations in MAM are performed using only the complex data (objects) in the domain and the similarity (distance) values between pair of objects. For MAM, the similarity value is computed using a (metric) distance function that have to satisfy the **symmetry**, **positivity**, and **triangular inequality** properties. A dataset that is defined by a metric distance function is also called metric dataset.

A few tools have been proposed targeting the understanding and developing of access methods. The *Amdb* [Kornacker et al. 2003], *Sail* [Hadjieleftheriou et al. 2005], and *Vasco* [Brabec et al. 2003] are examples of tools that construct visual representations of the dataset using spatial properties of objects indexed in access methods. These tools, however, are limited to datasets in 2D or 3D spaces. To the best of our knowledge, *MAMView* is the first framework to generate visualizations of high dimensional or metric datasets indexed in MAM, regardless of the dataset, distance function or dimensionality of the dataset. Furthermore, the *MAMView* framework is not only limited to generate visualizations of the dataset indexed in MAM, but also the execution steps of the algorithms of a MAM (e.g. an animation containing the steps of executing a *kNN* algorithm implemented in a MAM).

While the *MAMView* framework was first detailed in [Chino et al. 2005], our goal here is to demonstrate its main features to assist users in understanding MAM, as well as to help developers in designing and tuning new or existing algorithms. The *MAMView* framework is implemented in C++ and it is part of the *Arboretum* library[1], a portable, easy-to-use, and open-source platform for developing, testing and using MAM.

[1]Both frameworks are available for download at http://gbdi.icmc.usp.br/arboretum

2. The MAMView Framework

In this section, we first describe the model and its graphical representation employed in the *MAMView* framework. We then briefly detail the *MAMView* architecture, following with an example on how to install the *MAMView* API in order to generate visualizations for the *MAMViewer* tool.

In the *MAMView* framework, we use a generic model to visually represent multi-way trees, since most of the previously proposed MAM can be materialized into this category [Chávez et al. 2001]. Since multi-way trees partition the metric space into several *balls*, each one representing a subtree, in our model we symbolize each subtree as a *sphere* defined by a set of *representatives* (selected objects from the dataset) and radii. For simplicity, here we only describe the case where each partition is defined by a single pair of *representative*/radius. Each object in the data domain is associated to only one partition and it has to be "covered" by the *sphere* (partition radius). An object is "covered" by a partition if the distance between the object and the partition *representative* is less or equal than the radius of the partition. Thus, all objects in a partition have to be "covered" by the associated radius of the partition. In this way, every object in the dataset is associated with a single partition in a recursive way, generating a multi-way tree. Node overlapping may exist in MAM, and in our model it is symbolized by overlapping of *spheres*.

Using the above description, our model has the following properties: (1) each object is identified by a set of features, which is used by the distance function; (2) a *sphere* (node) has only one parent node, except for the root node that has none, but can have many children (subtrees); (3) an object is associated with a single partition, but may be covered by many due to node overlapping; (4) objects or *spheres* (subtrees) must have a unique identifier; (5) each *sphere* symbolizes a region in the space defined by the *representative*/radius; (6) *representatives* can be stored in multiple levels in the tree (e.g. an object or representative that is selected as *representative* of a *sphere* is copied in the immediate upper level); (7) the tree is organized in levels, which indicate the height of the tree; (8) a query is defined by a query center and a radius; and (9) a query answer returns a set of objects.

In order to cover all nine of these properties of our model, we employ the following graphical primitives in the *MAMView*: (1) an object is symbolized by a point in the mapped space, which position is defined by the mapping algorithm; (2) an object that belongs to a partition is symbolized by a connection line linking the object to the *sphere representative*; (3) an object is *connected* to only one *representative* (the *sphere representative* that the object belongs to); (4) the identifier of an object or a subtree is specified by a label next to it; (5) a *sphere* is symbolized by a circle (in the 2D projected space) centered at the *sphere representative*; (6) a *representative* is symbolized by a 7 point-star using different colors, which indicate the level that it belongs to; (7) colors are employed to symbolize different levels in the hierarchy; (8) a query is symbolized by a 5-point black star (query center) and a *sphere* centered in the query center (query radius); and (9) an object that belongs to the answer set of a query is symbolized by a black triangle.

The *MAMView* architecture, illustrated in Figure 1, is divided into two modules: the *MAMView Extractor* and the *MAMViewer*. The original dataset is first mapped to the 3D Euclidean space and then to 2D Euclidean space using the *MAMView Extractor* and *MAMViewer* tool, respectively. In this way, distance calculations are performed in the original space using the same metric employed to index object in the MAM. Thereafter, the

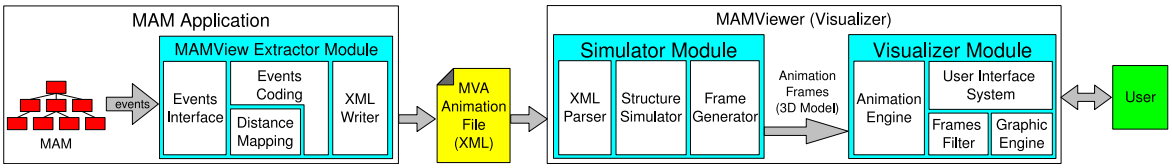

Figure 1. The *MAMView* architecture.

MAMViewer always works on the common Euclidean distance function, regardless of the original distance function. When the MAM algorithms are executed, the *MAMView Extractor* gathers all required information to *MAMViewer* tool. In this way, the *MAMView Extractor* is executed only once to generate the information needed for the *MAMViewer* tool, which can be executed many times. The *MAMView Extractor* can generate very complex visualizations of the organization of the indexed data (here we call it "dump" operation), or an animation containing the several steps (*frames*) of the execution of a MAM algorithm (e.g. steps of inserting a new object in the tree or evaluating a similarity search).

In the *MAMView Extractor*, the *Distance Mapping* module uses the *FastMap* algorithm [Faloutsos and Lin 1995], since it has several advantages for our settings (detailed next). This algorithm executes an iterative process, where the target mapping, in our case, is the 3D Euclidean space. In each iteration, two objects that have the largest distance value from each other are chosen to be the *pivots* of each target dimension. These *pivots* define the axis of the mapped dimension, and all other objects are projected in the space defined by these *pivots*. Thereafter, the projections of all other objects in the axis defined by the *pivots* are calculated, triangulating the object and the two *pivots* using the Cosine Law. In each iteration, the error in the mapping operation is calculated using the *stress* formula, defined as: $stress = \sqrt{\sum_{ij}(\hat{d}_{ij} - d_{ij})^2 / \sum_{ij} d_{ij}^2}$, where d_{ij} and $\hat{d}_{ij}$ are the distances between the objects s_i e s_j in the original and mapped spaces, respectively. Thus, the quality in the mapping process can be measured in each iteration of the *FastMap* algorithm.

In our framework, we chose the *FastMap* algorithm because it has three important characteristics for visualizing very large metric dataset. First, it is a linear algorithm with respect to the number of objects. Second, the set of pivots can be stored for reuse in further executions of the algorithm, which is done by the *CoordinateFastMap()*. Third, it preserves the original distances as much as possible, which can be measured by the *stress*.

We implemented two versions of the *FastMap* algorithm, one that *quickly* browses the index and selects six pivots (one pair for each dimension), called *FastMap()*, and another that maps the objects in the original metric space to the 3D Euclidean space, called *CoordinateFastMap()*. *FastMap()* receives a dataset with N objects and a distance function d, and it scans the index to find three pairs of pivots. Then, for each N objects in the 3D Euclidean space, the *CoordinateFastMap()* maps it using the distance function d and the six pivots.

The *MAMView Extractor* gets the data from the MAM following the events set by the developer (see Figure 2). The gathered data is written in the MVA (*MAMView Animation*) file format, which is an XML-based format. The MVA format specifies the set of frames and information of the objects needed to build the visualization in the 3D Euclidean space in the *MAMViewer* tool. Therefore, the only requirement to generate MVA files is to install the *MAMView Extractor* in the MAM.

Figure 2(a) shows one example of extracting the organization of objects in a MAM.

(a)

```
// Navegate through the index
Dump()
   MAMView.BeginAnimation() // Animation begins
   MAMView.SetLevel(0)      // Report tree level
   MAMView.BeginFrame()     // Frame starts
   DumpRecursive(root)
   MAMView.EndFrame()       // Frame ends
   MAMView.EndAnimation()   // Animation ends
end Dump

// Navegate the structure in pre-order
DumpRecursive(root)
   MAMView.LevelUp()     // Reports next level
   node = GetNode(root)
   MAMView.SetNode(node) // Declares node data
   for each entry of node do
      if entry is a subtree
         // Analize it recursively
         DumpRecursive(entry.root)
      else
         // It is an object. Reports it
         MAMView.SetObject(entry.object)
      else-if
   else-for
   MAMView.LevelDown()   // Finish the level
end DumpRecursive
```

(b)

```
// Range Query.
RangeQuery(q, radius)
   MAMView.BeginAnimation() // Animation begins
   result = CreateResult()  // Create an empty result
   Range(root, q, radius, result)  // Recur. call
   MAMView.BeginFrame()       // Starts frame
   MAMView.SetResult(result)  // Reports result
   MAMView.EndFrame()         // Ends frame
   MAMView.EndAnimation()     // Animation ends
   return result     // Return the answer
end RangeQuery

// Recursive Range Query
Range(subtree, sample, radius, result)
   MAMView.LevelUp()       // Reports next level
   node = GetNode(subtree) // Gets the node
   MAMView.BeginFrame()    // Begins a new frame
   MAMView.SetNode(node)   // Report the node
   MAMView.EndFrame()      // Ends frame
   for each entry of node do   // Analyze each entry
      if entry is subtree
         if entry.rep qualifies // Analyze the subtree
            Range(entry.subtree, q, radius, result)
            MAMView.BeginFrame()   // Begins a new frame
            MAMView.ActivateNode(node)  // Activate it
            MAMView.EndFrame()     // Ends the frame
         end-if
      else           // Entry is an object
         MAMView.BeginFrame()   // Begin a new frame
         MAMView.SetObject(entry.object)//Add the object
         MAMView.EndFrame()     // Ends the frame
         if entry qualifies for answer
            result.add(entry.object) // Updates Answer
            MAMView.BeginFrame() // Begins a new frame
            MAMView.SetResult(result) // Reports result
            MAMView.EndFrame() // Ends the frame
         end-if
      end-if
   end for
   MAMView.LevelUp()   // Reports the end of this level
end Range
```

Figure 2. Examples of the *MAMView Extractor* API (`MAMView` object) to extract information for a tree structure (a) and the execution steps a range query (b).

The `Dump` and `DumpRecursive` procedures traverse a tree in *pre-order* to extract information to generate a MVA file. The *MAMView Extractor* APIs are represented in *red color* and they all start with the `MAMView` prefix. Figure 2(b) shows how to install the *MAMView Extractor* APIs in order to build an animation containing the steps of an execution of a range similarity query, represented by `RangeQuery` and `Range` procedures. In this last example, a sequence of frames is created to emulate each step of the range algorithm.

With the physical separation of *MAMView Extractor* and *MAMViewer* in the *MAMView* architecture, users can freely interact with the visualization, going forward and backward in the algorithms execution, without interact on the index itself. The interaction of the user with the simulation is very important, because it allows users to understand certain properties and behaviors of the MAM algorithms that cannot be displayed all at once. Moreover, this separation has the advantage of enabling user to share common visualizations through MVA files, without the need of performing the mapping several times, or interfering the performance of the index.

The *MAMViewer* tool has two components, the *Simulator* and *Visualizer* modules. The first one reads a MVA file and prepares the set of frames for the second module. The *Visualizer* is the module where the user can select the desired portion of the information, as well as navigate through the algorithm's execution by "playing" the corresponding set of frames (i.e., showing each frame in the proper sequence). The *MAMViewer* tool uses the

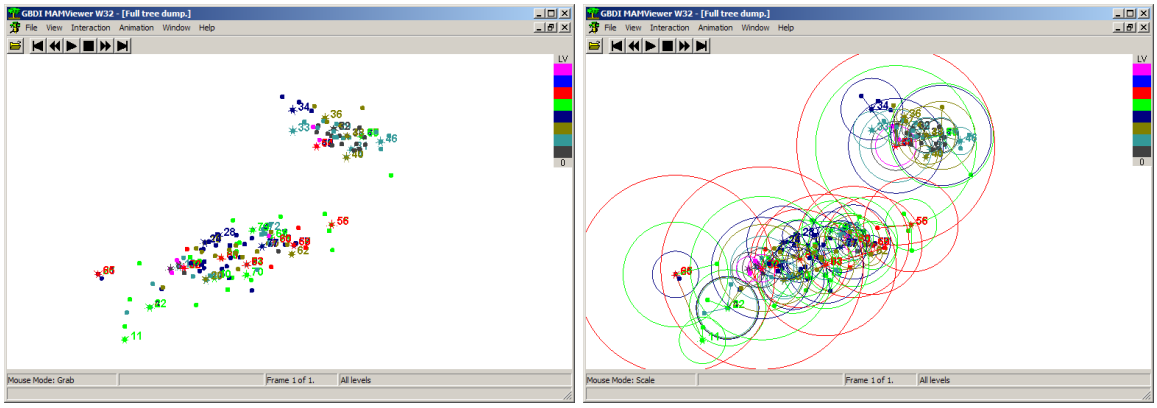

Figure 3. Snapshots of the *MAMViewer* using *iris*: (left) objects, *representatives* and labels only; and (right) with the addition of spheres and connections.

OpenGL[2] library to manipulate the visualization following the user's commands.

3. Demonstration

In this demonstration, we explore several *MAMView* visualizations generated for the Slim-tree with the *faces*[3] and *iris*[4] datasets. The *faces* dataset has 11,900 objects, each with 16 features extracted from images of human faces using the Eigenfaces method. The *iris* dataset has 150 features describing 4 attributes (sepal length, sepal width, petal length, and petal width) of 3 types of Iris flowers (*Setosa*, *Versicolour* and *Virginica*).

Figure 3 presents two snapshots of the *MAMViewer* tool for the *iris* dataset. Both visualizations have a single frame since they represent the organization of the data ("dump") indexed by a Slim-tree. The first snapshot (left) shows only objects, *representatives* and labels, while in the second snapshot (right), radius and connections were also enabled in the previous visualization. In these two snapshots, the navigational controls are displayed on the top left area of the tool, just below the menus for the graphical interactions. On the bottom of the tool, messages are displayed with the settings of the visualization and/or the structure, following the context of the visualization for each step of the MAM algorithm. The color pattern, shown on the top right position of the tool, can be redefined, and each color symbolizes a different level in the hierarchy of the index.

Figure 4 shows the first two frames, out of 230, for an animation containing the execution steps of a range query for the *faces* dataset. Figure 4(left) presents the query center (black star) and the three *spheres* with their *representatives*, symbolizing three subtrees of the root node of the index. Figure 4(right) shows the second frame in the animation where the range query algorithm explores the second level (red color) of *sphere* 585.

The *MAMViewer* tool allows users, in an animated visualization, to "navigate" the algorithm's execution frame by frame, forward or backward, in a similar fashion of "playing" a movie. Additionally, the tool allows users to interactively change any properties presented in a visualization, e.g., display only the indexed objects, *spheres*, radii, *spheres representatives*, identifiers, or any combination of these properties. In the *MAMViewer* tool, users can change the appearance of the graphical elements, increasing/diminishing the size

[2] http://www.opengl.org

[3] http://www.informedia.cs.cmu.edu

[4] http://kdd.ics.uci.edu

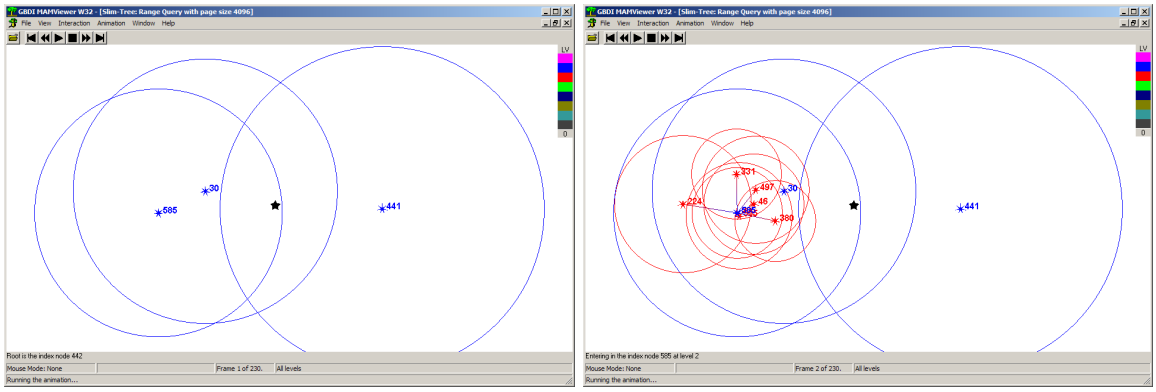

Figure 4. Two frames of an animation for a range query for *faces* (black star symbolizes the query center): (left) 1st frame with the 3 subtrees (30, 441 and 585) in the root *sphere*; and (right) 2nd frame exploring the second level of subtree 585

of the dots symbolizing the objects, or selecting the number of frames that are superimposed in the visualization (*tail length*) of an animation.

The visualization can be rotated, translated or scaled, allowing users to closely inspect certain regions of the visualization. These features are very useful, since one of the main problems in visualizing large datasets is the potential "cluttering" and "occlusion" in the visualization. The visualization depicted in a frame is only the starting point for the user to work on the visual presentation of the data, so the user can graphically manipulate the visualization with the special resources presented in the tool ("fading", "tail", center of observation, etc.). During the analysis, the user can highlight objects, *spheres*, *representatives*, radii, etc., or certain levels of the tree (e.g. the first three levels of the tree). All the manipulations over the visualization are facilitated by mouse interaction, as well as by menu shortcuts.

Acknowledgements: This research was supported by FAPESP, CNPq and CAPES grants.

References

Brabec, F., Samet, H., and Yilmaz, C. (2003). Vasco: visualizing and animating spatial constructs and operations. In *Annual Symp. on Computational Geometry*, pages 374–375.

Chino, F., Vieira, M., Traina, A., and Jr, C. T. (2005). MAMView: A visual tool for exploring and understanding metric access methods. In *ACM Symp. on Applied Computing (SAC)*, pages 1218–1223.

Chávez, E., Navarro, G., Baeza-Yates, R., and Marroquín, J. L. (2001). Searching in metric spaces. *ACM Computing Surveys*, 33(3):273–321.

Faloutsos, C. and Lin, K.-I. (1995). FastMap: A fast algorithm for indexing, data-mining and visualization of traditional and multimedia datasets. In *ACM SIGMOD*, pages 163–174.

Hadjieleftheriou, M., Hoel, E., and Tsotras, V. (2005). Sail: A spatial index library for efficient application integration. *Geoinformatica*, 9(4):367–389.

Jr, C. T., Traina, A., Faloutsos, C., and Seeger, B. (2002). Fast indexing and visualization of metric datasets using slim-trees. *IEEE Trans. Knowl. Data Eng.*, 14(2):244–260.

Kornacker, M., Shah, M., and Hellerstein, J. (2003). Amdb: A design tool for access methods. *IEEE Data Eng. Bull.*, 26(2):3–11.

Vieira, M., Jr, C. T., Traina, A., and Chino, F. (2004). DBM-Tree: A dynamic metric access method sensitive to local density data. In *Brazilian Symp. on Databases (SBBD)*, pages 163–177.