Indexing Metric Spaces with Nested Forests of Topological Balls and Hyperplanes

José Martinez, Zineddine Kouahla

To cite this version:

José Martinez, Zineddine Kouahla. Indexing Metric Spaces with Nested Forests of Topological Balls and Hyperplanes. 23rd Int. Conf.on Database and Expert Systems Applications - DEXA 2012, Sep 2012, Wien, Austria. pp.458-465, 2012. <hal-00711604>

HAL Id: hal-00711604

https://hal.archives-ouvertes.fr/hal-00711604

Submitted on 9 Jul 2013

Indexing Metric Spaces with Nested Forests of Topological Balls and Hyperplanes

José Martinez and Zineddine Kouahla

Lunam Université
Laboratoire d'informatique de Nantes-Atlantique (LINA, UMR CNRS 6241)
École polytechnique de l'université de Nantes – BP 50609 – F-44306 Nantes cedex 3
{name.surname}@univ-nantes.fr

Abstract. Searching in a dataset for objects that are similar, with respect to a distance, to a given query object is a fundamental problem for several applications that use complex data, e.g., strings, graphs. The main difficulties are to focus the search on as few elements as possible and to further limit the computationally-extensive distance calculations between them.

Here, we introduce a forest data structure for indexing and querying such data. The efficiency of our proposal is studied through experiments on real-world datasets and a comparison with previous proposals.

1 Introduction

For several decades, indexing techniques have been developed in order to deal with efficient searches over large collections of data, especially associative searches, i.e., searches where part of the information to be retrieved is provided, namely a key. This basic problem has been extended over the years in order to retrieve information based on any subset of its contents as well as taking care of imprecisions. When considering indexing data as vectors, homogeneous or inhomogeneous ones, it turned out that search and indexing become more and more difficult when the dimension of the so-called vectors increases. This has been named the "dimensionality-curse problem." The reader can find several surveys to present and compare existing multidimensional indexing techniques [8] [2] [10].

However, the objects to be indexed are often more complex than mere vectors (e.g., sets, graphs) or cannot be simply and meaningfully concatenated in order to give a larger vector (e.g., colour histograms and keywords for jointly describing images in a multimedia database). Hence, the focus of indexing has partly moved from multidimensional spaces to metric spaces, i.e., from exploiting the data representation itself to working on the similarities that can be computed between objects. Inherently, the difficulties of multidimensional spaces remain, in a generalised version, whereas new difficulties arise due to the lack of information on the objects.

This paper introduces a Metric-space Forest Indexing (MFI) technique. It is organised as follows: Section 2 introduces kNN queries and reviews a short taxonomy of indexing technique in metric spaces. Then, Section 3 introduces our proposal, overviews its main characteristics and the corresponding algorithms. Section 4 discusses the experimental results. Section 5 concludes the paper and introduces research directions.

2 Indexing in Metric Spaces

Metric spaces are becoming more and more useful, for several applications need to compare objects based on a similarity between them that is formalised as a mathematical distance. Let us focus the basic search query and introduce some indexing technique from the literature.

2.1 Similarity Queries in Metric Spaces

There exist numerous measures of similarity applicable to various kinds of "objects," e.g., Minkowski distances, among which the best known are the Manhattan distance and the Euclidean distance, that can be applied to any vector-like data such as colour histograms in multimedia databases.

Formally, a metric space is defined for a family of elements that are comparable through a given distance.

Definition 1 (Metric space). *Let $\mathcal{O}$ be a set of elements. Let $d : \mathcal{O} \times \mathcal{O} \to \mathbb{R}^+$ be a distance function, which verifies:*

1. *non-negativity:* $\forall (x, y) \in \mathcal{O}^2, d(x, y) \geq 0$*;*
2. *reflexivity:* $\forall x \in \mathcal{O}, d(x, x) = 0$*;*
3. *symmetry:* $\forall (x, y) \in \mathcal{O}^2, d(x, y) = d(y, x)$*;*
4. *triangle inequality:* $\forall (x, y, z) \in \mathcal{O}^3, d(x, y) + d(y, z) \leq d(x, z)$*.*

Then $(\mathcal{O}, d)$ is a metric space.

The concept of metric space is rather simple and leads to a limited number of possibilities for querying an actual database of such elements. These are called similarity queries and several variants exist. We consider k nearest neighbour (kNN) searches, i.e., searching for the k closest objects with respect to a query object.

Definition 2 (kNN query). *Let $(\mathcal{O}, d)$ be a metric space. Let $q \in \mathcal{O}$ be a query point and $k \in \mathbb{N}$ be the expected number of answers. Then $(\mathcal{O}, d, q, k)$ defines a kNN query, the value of which is $S \subseteq \mathcal{O}$ such that $|S| = k$ (unless $|\mathcal{O}| < k$) and $\forall (s, o) \in S \times \mathcal{O}, d(q, s) \leq d(q, o)$.*

2.2 Background

Metric spaces introduce the notion of topological ball, which is close to a broad match. It allows to distinguish between inside and external objects.

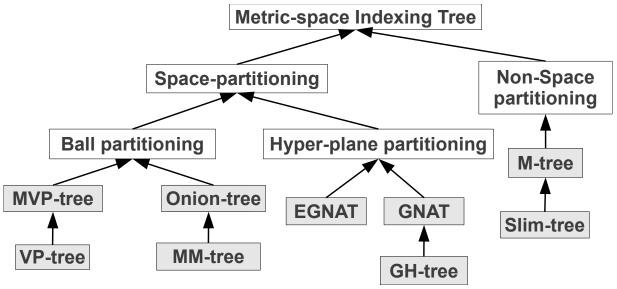


Fig. 1. A simplified taxonomy of indexing techniques in metric spaces

Definition 3 ((Closed) Ball). *Let $(\mathcal{O}, d)$ be a metric space. Let $p \in \mathcal{O}$ be a pivot object and $r \in \mathbb{R}^+$ be a radius. Then $(\mathcal{O}, d, p, r)$ defines a (closed) ball, which can partition inner objects from outer objects:*

$$\begin{aligned} I(\mathcal{O}, d, p, r) &= \{o \in \mathcal{O} : d(p, o) \leq r\} \; ; \\ O(\mathcal{O}, d, p, r) &= \{o \in \mathcal{O} : d(p, o) > r\}. \end{aligned}$$

Another useful partitioning concept is the one of generalised "hyper-plane."

Definition 4 (Generalised hyper-plane). *Let $(\mathcal{O}, d)$ be a metric space. Let $(p_1, p_2) \in \mathcal{O}^2$ be two pivots, with $d(p_1, p_2) > 0$. Then $(\mathcal{O}, d, p_1, p_2)$ defines a generalised hyper-plane:*

$$H(\mathcal{O}, d, p_1, p_2) = \{o \in \mathcal{O} : d(p_1, o) = d(p_2, o)\}$$

which can partition "left-hand" objects from "right-hand" objects:

$$\begin{aligned} L(\mathcal{O}, d, p_1, p_2) &= \{o \in \mathcal{O} : d(p_1, o) \leq d(p_2, o)\} \; ; \\ R(\mathcal{O}, d, p_1, p_2) &= \{o \in \mathcal{O} : d(p_1, o) > d(p_2, oP)\}. \end{aligned}$$

Based on these two partitioning techniques, we can introduce a short taxonomy of some indexing techniques in metric spaces (See Figure 1). There are two main classes:

1. The first class is based on the partitioning of the space. Two sub-approaches are included in this first class:
   (a) One of them uses ball partitioning, like VP-tree [13], mVP-tree [3], etc.
   (b) The other approach uses hyper-plane partitioning such as GH-tree [12], GNAT [4], etc.
2. The second class does enforce a partitioning (non-space-partitioning). There, we find M-tree, Slim-tree, etc.

In the second class, the M-tree [7] builds a balanced index, allows incremental updates, and it performs reasonably well in high dimension. Unfortunately it suffers from the problem of overlapping that increases the number of distance calculations to answer a query. There is an optimised version of it: the Slim-tree [11]. It mainly reorganises the M-tree index in order to reduce overlaps. The used algorithm, called slim-down, has shown good performances on the research algorithm and has reduced its processing time. Its defect is the need for reinserting objects.

The first class, the one base on space-partitioning of the space, either with balls or with hyper-planes, is richer.

The GH-tree is a type of index based on the partitioning through hyper-planes. It has proven its efficiency in some dimensions but it is still inefficient in large dimensions. The principle of this technique is the recursive partitioning of space into two regions. We choose each time two pivots, and each object is associated to the nearest pivot.

The VP-tree is a type of index based on a partitioning thanks to a ball. The VP-tree building process is based on finding the median element of a set of objects. The mVP-tree is an enary generalisation of the VP-tree. The mVP-tree nodes are divided into quantiles. In fact, the operations (insertion and search) on this type of index are very similar to VP-trees. Often, it behaves better; but there is not enough differences to investigate further.

In recent years, a new technique has emerged, the MM-tree [9], which also uses the principle of partitioning by balls, but it is based on the exploitation of regions obtained from the *intersection* between two balls. The general idea of this structure is to select two pivots from the set of objects, in a random way, then to partition the space into four *disjoint* regions using these two balls, their intersection, their respective differences, and their “external space,” i.e., the complementation of their union. The partitioning is done in a recursive way. In order to improve the balancing of the tree, a semi-balancing algorithm is applied near to the leaves, which reorganises the objects in order to gain one level when possible.

An extension of this technique has been developed: the Onion-tree [5]. Its aim is to divide the region external space into successive expansions that improve the search algorithm, because this last region is particularly vast. The goal is to have a wider and shallower index in order to go faster to the right answers to a query. In our opinion, the problem is not totally solved because the construction phase is always slow due to the reinsertion of objects.

3 Metric Forest Indexing

In this section, we introduce a forest of (quadrary) trees, called MFI for short, as an indexing technique in metric spaces. More specifically, it is a family of such trees as we shall see that the framework is generic and allows various implementations. It is a memory-based metric access method that divides recursively the dataset into *disjoint* regions.

3.1 Definition

Let us introduce formally the MFI.

Definition 5 (MFI). *Let $M = (\mathcal{O}, d)$ be a metric space. Let $E \subset \mathcal{O}$ be a subset of objects to be indexed. Let $1 \leq c_{\max} \leq |E|$ be the maximal cardinal that one associates to a leaf node.*

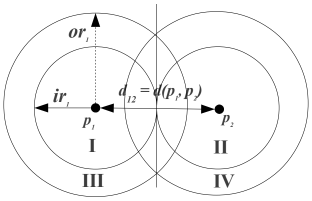


Fig. 2. The partitioning principle of one MFI node

We define $\mathcal{N}$ as the nodes of a so-called MFI in the usual two-fold way: Firstly, a leaf node L consists merely of a subset of the indexed objects, all of them belonging to a closed ball:

$$(p, r, E_L) \in \mathcal{O} \times \mathbb{R}^+ \times 2^{\mathcal{O}}$$

with:

- $E_L \subseteq E$*;*
- $|E_L| \leq c_{\max}$.

Also, the contents of the leaves partition E.
Secondly, an internal node N is a generic set of couples with a recursive component:

$$\{(P_1, N_1), \ldots, (P_n, N_n)\} \in 2^{\mathcal{P} \times \mathcal{N}^{\mathbb{N}}}$$

where:

- $\mathcal{P}$ *is a generic data structure;*
- *each N_i is a sequence of (sub-)nodes;*
- $n \geq 1$. *(Note that the forest becomes* de facto *a tree when all its internal nodes have arity 1.)*

This generic definition is instantiated hereafter based on the heuristic rules, namely: (i) avoiding too large balls (especially near the root of the tree), (ii) intersecting them to further limit their volume, (iii) partitioning them into concentric rings.

Figure 2 illustrates the way a node of the forest is built at a given step in the refining process of recursively splitting the dataset. The structure of the following instance is rather clumsy at first sight. Let us explain it. Two distinct pivots are chosen at random in a given data subset. They serve both as the centres of two balls each and as the base points for an hyper-plane. The two balls create rings that help to reduce the number of searches children searches when then query ball has an intersection either with the ring or with the inner ball but not both.

The first ball, for each pivot, has a radius that is equal to half the distance between the pivots. The second ball has a radius that is equal to a third half of this distance. The goal is to avoid too large balls since in "multidimensional" the volume grows exponentially with respect to the radius.

Thanks to these balls and to the hyper-plane, we can divide the space into four disjoint regions, namely (I) and (II) the inner balls, (III) and (IV) the intersection between the outer balls and their corresponding

"half-plane". There remains a fifth part that is not to be treated as part of the node. The rationale being that idea is again to avoid very large balls, especially near the root of the "tree" because in that case no pruning could occur.
Note that this definition is just one among several others. We shall exemplify this in Section 4 since previously proposed metric trees have been implemented under our framework. In the following definition we provide the formal definition of this proposal, which is slightly more complex that described above.

Definition 6 (An MFI Instance). *Let $M = (\mathcal{O}, d)$ be a metric space. Let $E \subset \mathcal{O}$ be a subset of objects to be indexed. Let $1 \leq c_{\max} \leq |E|$ be the maximal cardinal that one associates to a leaf node.*
Then, we define the following instance through an instantiation of the generic component $\mathcal{P}$ of an MFI:

$$(p_1, p_2, d_{12}, ir_1, ir_2, ior_1, ior_2, sor_1, sor_2) \in \mathcal{O}^2 \times {\mathbb{R}^+}^6$$

where:
- *(p_1, p_2) are two* distinct *objects, i.e., with $d(p_1, p_2) > 0$, called* pivots*;*
- *$d_{12} = d(p_1, p_2)$ is redundantly stored to save computations, from which we also define an inner radius $ir = d_{12}/2$ and an outer radius $or = 3/2d_{12}$;*
- *$ir_1 \leq ir$ and $ir_2 \leq ir$ are the distances to the farthest object in the sub-tree rooted at that node N with respect to the inner balls centred on p_1 and p_2 respectively, i.e., sub-spaces (I) and (II), with $ir_i = \max\{d(p_i, o), \forall o \in N\}$ for $i = 1, 2$ where the set notation $o \in N$ is abusively used for the union of the leaf extensions that are rooted at N;*
- *$ir < ior_1 \leq sor_1 \leq or$ and $ir < ior_2 \leq sor_2 \leq or$ are their more complex counterparts for data belonging to sub-spaces (III) and (IV), for (ior_i, sor_i) delimits a* ring *centred on p_i;*

Then, the children of each couple always consists of four sub-nodes, (N_1, N_2, N_3, N_4), such that:
- *$N_1 = \{o \in N : d(p_1, o) \leq ir \wedge d(p_2, o) > r\}$ for the first inner ball centred on p_1 (region I) that* includes *the single point on the hyper-plane;*
- *$N_2 = \{o \in N : d(p_2, o) \leq ir \wedge d(p_1, o) \geq r\}$ for the partial ball centred on p_2 (region II) that* excludes *the single point on the hyper-plane;*
- *$N_3 = \{o \in N : ir < d(p_1, o) \leq or \wedge d(p_1, o) \leq d(p_2, o)\}$ for the intersection between the ring centred on p_1 and the corresponding hyperplane* including *the points of this plane, i.e., region III;*
- *$N_4 = \{o \in N : ir < d(p_2, o) \geq or \wedge d(p_2, o) < d(p_1, o)\}$ for region IV;*

with the same informal set notation for the extension of an internal node.

3.2 Building an MFI data structure

Building an MFI can be done incrementally or in batch mode. For the sake of clarity, and due to space limitation too, Algorithm 1 presents formally the batch version only, corresponding to the more specific MFI intance of Definition 6.

Algorithm 1 Building an MFI instance in batch-mode

```
Build $\left(E \subseteq \mathcal{O}, d \in \mathcal{O} \times \mathcal{O} \rightarrow \mathbb{R}^{+}, c_{\max} \in \mathbb{N}^{*}\right) \in \mathcal{N}$
with either:
  – $p = \text{any}\{(o \in E\}$;
  – $r = \max\{d(p, o) : o \in E\}$;
or:
  – $(p_1, p_2) = \text{any}\{(o_1, o_2) \in E^2 : d(o_1, o_2) > 0\}$;
  – $d_{12} = d(p_1, p_2)$;
  – $ir = d_{12}/2$;
  – $or = 3/2 d_{12}$;
  – $ir_1 = \max\{d(p_1, e) : \forall e \in E, d(p_1, e) \leq ir\}$;
  – $ir_2 = \max\{d(p_2, e) : \forall e \in E, d(p_1, e) \leq ir\}$;
  – $E_1 = \{o \in E : d(p_1, o) \leq ir \wedge d(p_2, o) \leq r\}$;
  – $E_2 = \{o \in E : d(p_1, o) < r \wedge d(p_2, o) \geq r\}$;
  – $E_3 = \{o \in E : d(p_1, o) \geq r \wedge d(p_2, o) < r\}$;
  – $E_4 = \{o \in E : d(p_1, o) \geq r \wedge d(p_2, o) \geq r\}$;
```

$$\triangleq \begin{cases} \{(\bot, \emptyset)\} \text{ if } E = \emptyset \\ \{((p, r), E)\} \text{ if } |E| \leq c_{\max} \\ \{((p, r), E)\} \text{ if } \forall (o_1, o_2) \in E^2, d(o_1, o_2) = 0 \\ \left\{ \left(\begin{pmatrix} p_1, p_2, d_{12}, \\ ir_1, ir_2, ior_1, sor_1, ior_2, sor_2 \end{pmatrix}, \begin{pmatrix} \text{Build}(E_1, d, c_{\max}), \\ \text{Build}(E_2, d, c_{\max}), \\ \text{Build}(E_3, d, c_{\max}), \\ \text{Build}(E_4, d, c_{\max}) \end{pmatrix} \right) \right\} \cup \text{Build}(E \setminus E_1 \setminus E_2 \setminus E_3 \setminus E_4, d, c_{\max}) \text{ otherwise} \end{cases}$$

For cases where the whole collection of data to index is known before hand, a batch-mode construction can be chosen. However, in most situations, an incremental version would be required.

The incremental version is more intricate, therefore we describe it literately only. The insertions are done in a top-down way.

Initially, the forest is empty, i.e., it is an empty set, that can be translated into a singleton containing a single empty leaf node.

Then, the first insertions in a leaf make it only grow until a maximum number of elements, i.e., $c_{\max}$, is attained. Due to time complexity considerations, its value cannot be larger than $\sqrt{n}$ where $n = |E|$ is the cardinal of the whole population of objects to be inserted in the tree. The ball is updated whenever a new object is inserted in the leaf extension.

When the cardinal limit is reached, a leaf is replaced by an internal node and four new leaves are obtained by splitting the former set of objects into four subsets according to the conditions given in Definition 6 (or another variant).

Here, in order to split the object set, two *distinct* pivots have to be chosen. The selection of these pivots plays an important role in our proposal along with the $c_{\max}$ parameter. The goal is to balance, as much as possible, the tree. In Algorithm 1, without clear guidelines, we "decided" to choose two objects at random.

Inserting a new object into an internal node amounts to selecting the subtree that has to contain it with respect to the conditions given in Definition 6 and applying the insertion recursively. As a side-effect, ir_i, or ior_i and/or sor_i may be modified.

However, the new object does not necessarily belong to regions I to IV. In that case, we create a brother node, i.e., the "tree" becomes an actual forest.

Let us note that, at each internal node, only two distances are calculated in order to insert a new object. Besides, the forest tends to be rather balanced, hence inserting a new object is a logarithmic operation, in amortised cost.

3.3 Similarity queries

Next, let us shortly describe the algorithm for answering kNN queries thanks to an MFI. We develop a "standard" algorithm, i.e., a kNN search that runs a "branch-and-bound" algorithm where the query radius r_q is monotonically decreasing down to the distance to the forthcoming k^{th} answer. The algorithm accepts an additional parameter A, $A = ((d_1, o_1), (d_2, o_2), \ldots, (d_{k'}, o_{k'}))$, i.e., a solution "so-far," with $k' \leq k$ and initialised to an empty answer.

Basically, the forest is traversed in pre-order. When arriving at a leaf node, the currently known sub-solution is merged with the local subsolution. Note that "k-sort" and "k-merge" are variants of the sort and merge algorithms respectively where the size of the answer is limited at most to the first k values. These variants are faster.

When arriving on an internal node, the various children regions are envisaged, the remaining candidates are ordered and hopefully some of them

			Dimensions		
Dataset	#Elements	Distance	Apparent	Intrinsic	Description
French Cities	36,000	L_2	2	2	2D Coordinates
MPEG-7	10,000	L_1	64	7	SCD

Table 1. Real-world datasets

pruned, finally the search is pursued in the elected children in expected order of relevance. In that way, the sub-solution from a previous call is transmitted to the next call, hence improving the knowledge of the next branch on the query upper-bound.

4 Experiments and Analysis

In order to explore the efficiency of our approach, we run some experiments. Firstly, we choose a few real-world datasets and the accompanying queries, then run our indexing structure along previously introduced ones, and finally evaluate some measures on the index structure as well as the kNN searches.

4.1 Datasets

Real-world datasets are summarised in Table 1. The so-called intrinsic dimensional of a dataset is computed thanks to the Chávez et al. formula [6]:

$$d = \frac{1}{2}\frac{\mu^2}{\sigma} \tag{1}$$

where μ is the average value of the observed distances between pairs of objects and σ is the variance. It gives a measure of the complexity of indexing a given dataset.

French cities coordinates provides a base line, since 2D coordinates are easy to index. By contrast, the Scalable Colour Descriptors (SCD) of MPEG-7 are multimedia descriptors that present large dimensions.[1] They are not suitable, in this form, for common multidimensional indexing techniques such as R-trees, X-trees [1], etc. However, their intrinsic dimensionality is not that high, “only” 7.

To run the search algorithms, we used 1,000 different objects as queries and averaged the results.

4.2 Algorithms

Actually, we developed and used five algorithms:

- **Naive.** This is the basic search algorithm, which consists basically of an improved sort limited to the first k^{th} values, the complexity of which is $O(n.D) + O(n.\log_2 k)$ where D is the complexity of the used distance. It serves as a (worst) comparison stallion.

[1] Available from the COPhIR collection at `http://cophir.isti.cnr.it`.

	VP			GH			MM	MFI		
	8	log	sqrt	8	log	sqrt	8	8	log	sqrt
min	61,763	116,563	588,529	271,929	233,864	198,431	152,170	59,093	**47,476**	182,950
average	1,102,326	1,083,944	1,368,951	1,465,409	1,320,001	1,254,771	1,224,772	983,814	**976,551**	1,055,460
stddev	410,149	375,947	**264,875**	511,888	443,818	402,526	449,584	372,059	376,884	363,224
max	1,967,888	1,860,595	1,887,112	2,500,494	2,204,342	1,884,863	2,079,803	1,724,726	**1,716,615**	1,792,628

Table 2. SCD experimental measures

- **FMI.** This is our partitioning proposal.
- **GH, VP, and MM.** These are *not* the original versions but three bucketed implementations of them, developed under our framework. This allows a fair comparison, all the versions being implemented and instrumented with a largely common code.

The $c_{\max}$ parameter was chosen either as a constant, 8, the logarithm (base 2) or the square root of the size of the collection. For our MM-tree-like implementation, only the constant 8 is used; it corresponds to the original semi-balancing algorithm as well as to a tight constraint on the time complexity for building the tree.

In all the experiments, we run kNN searches with k equal to 20.

4.3 Measures on kNN Searches

When running the queries, we extracted several measures. Hereafter, we present only the overall performance, i.e., the sum of (i) the number of accessed objects, (ii) the number and the cost of distance computations, and (iii) the number of distance comparisons including the sort and merge phases.

For the naïve algorithm, this is almost a constant; effectively, it appears as an horizontal line on the various figures.

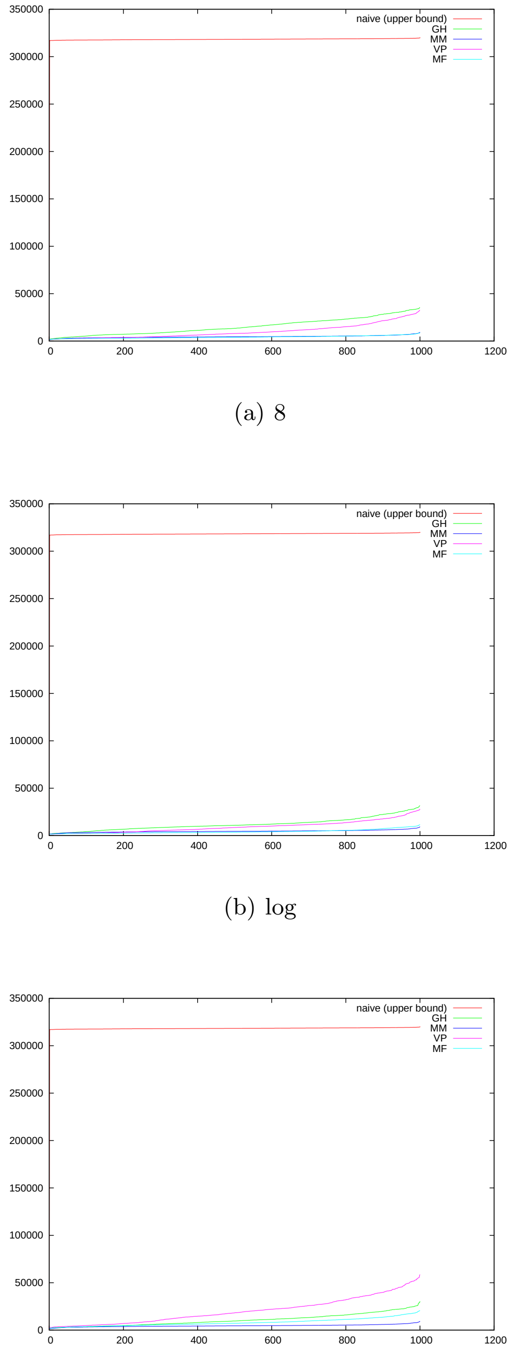
Experiments for the French cities dataset, the easy one, show clearly (See Figure 3) that the performances are largely better than a k-sort. We note that they tend to degrade as the size of the leaves increases, but this degradation is less important for our proposal than for GH- and VP-like trees (and remember that MM-like has a fixed size of 8).

Experiments for the multimedia dataset show clearly two points (See Figure 4): Firstly, the overall performances largely degrade for all the proposals, becoming in some cases worse than the naïve algorithm! This is just another illustration of the so-called "curse of dimensionality" problem. Secondly, our proposal is the one that resists the best, being largely under the other ones for $c_{\max} = \sqrt{|E|}$.

Table 2 provides the numerical values of Figure 4 where we highlighted the smallest values on each line. It turns out that our proposal with a logarithmic size of the leaves is probably the best compromise.

5 Conclusion

In this paper, we have extended the hierarchy of indexing methods in metric spaces with a family of indices consisting of a generalisation of

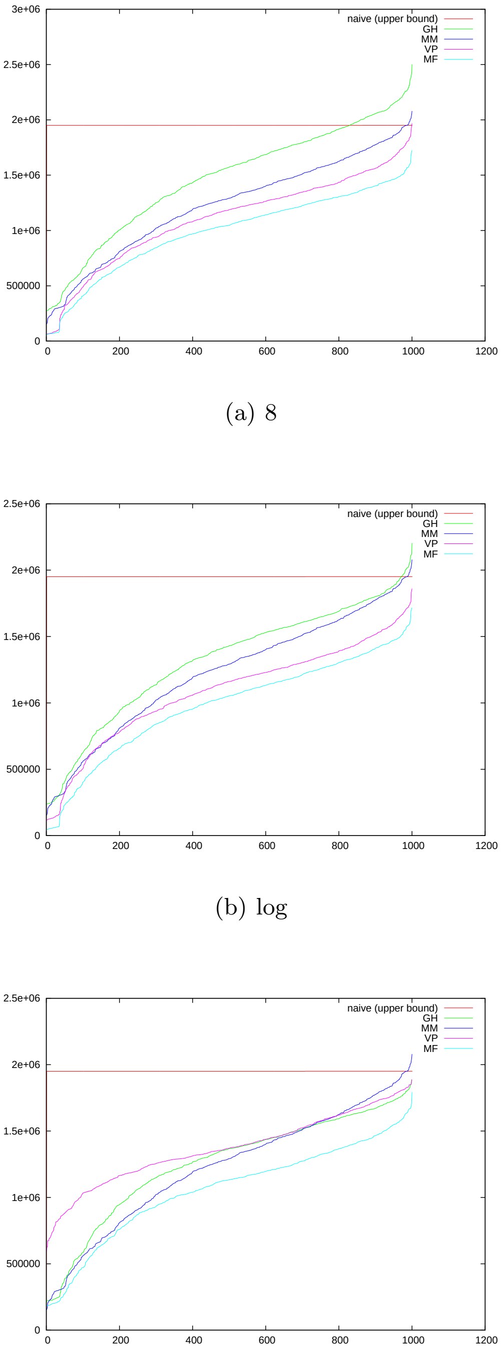


(a) 8

(b) log

(c) sqrt

Fig. 3. French cities dataset performance curves



(a) 8

(b) log

(c) sqrt

Fig. 4. SCD performances curves

metric trees: metric forests. We provided a first instantiation that combines three approaches that seems *a priori* interesting when partitioning a search space and found in the space partitioning part of the literature. In addition, thanks to the concept of forest, we introduce some aspects that are present in M-tree, i.e., several balls are located at the same level which should help to avoid exploring some branches. Experiments are encouraging and we shall work on determining the parameters that best contribute to reducing the search effort.
Next, we notice that the performance curves tend to have close shapes. We do believe that there is a intrinsic limit to the gain that can be achieve through this king of improvements and that parallelism is unavoidable for dealing with large datasets that present inherent "dimensionality" difficulties. Therefore, we shall also investigate parallel search on these structures. Working on a forest rather than a tree certainly offers additional possibilities.

References

1. S. Berchtold, D. A. Keim, and H.-P. Kriegel. The X-tree: An index structure for high-dimensional data. In *Proceedings of the 22nd International Conference on Very Large Data Bases (VLDB)*, pages 28–39, Mumbai (Bombay), India, 1996.
2. Christian Böhm, Stefan Berchtold, and Daniel A. Keim. Searching in high-dimensional spaces: Index structures for improving the performance of multimedia databases. *ACM Computing Surveys*, 33(3):322–373, September 2001.
3. Tolga Bozkaya and Meral Özsoyoglu. Indexing large metric spaces for similarity search queries. *ACM Transactions on Database Systems*, 24:361–404, September 1999.
4. Sergey Brin. Near neighbor search in large metric spaces. *Proceedings VLDB Conference Switzerland, 1995*, pages 574–584, 1995.
5. Caio César Mori Carélo, Ives Rene Venturini Pola, Ricardo Rodrigues Ciferri, Agma Juci Machado Traina, Caetano Traina Jr., and Cristina Dutra de Aguiar Ciferri. Slicing the metric space to provide quick indexing of complex data in the main memory. *Inf. Syst*, 36:79–98, 2011.
6. Edgar Chavez, Gonzalo Navarro, Jose Luis Marroquin, and Ricardo Baeza-Yates. Searching in metric spaces. *ACM Computing Surveys*, 33(3), September 2001.
7. Paolo Ciaccia, Marco Patella, and Pavel Zezula. M-tree: An efficient access method for similarity search in metric spaces. *Proceedings of the 23rd VLDB International Conference*, pages 426–435, 1997.
8. V. Gaede and O. Günther. Multidimensional access methods. *ACM Computing Surveys*, 30(2):170–231, 1998.
9. Ives R. V. Pola, Caetano Traina, Jr, and Agma J. M. Traina. The mm-tree: A memory-based metric tree without overlap between nodes. *ADBIS 2007*, LNCS 4690:157–171, 2007.
10. Hanan Samet. *Foundations of Multidimensional And Metric Data Structures.* Morgan-Kaufmann, September 2006. 993 p.

11. Caetano Traina Jr, Agma Traina, Bernhard Seeger, and Christos Faloutsos. Slim-trees: High performance metric trees minimizing overlap between nodes. *International Conference on Extending Database Technology (EDBT)*, 2000.
12. Jeffrey K. Ulhmann. Satisfying general proximity/similarity queries with metric trees. *Information Processing Letters*, 40:175–179, 1991.
13. Peter N. Yianilos. Data structures and algorithms for nearest neighbor search in general metric spaces. *proceedings of the 4th Annual In ACM-SIAM Symposium on Discrete Algorithms*, pages 311–321, 1993.