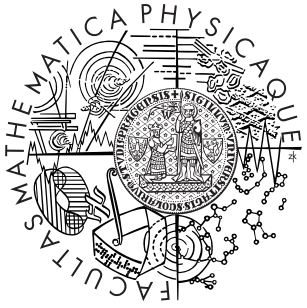
Charles University in Prague
Faculty of Mathematics and Physics

Similarity Search In Multimedia Databases

Habilitation thesis

Tomáš Skopal



Prague 2006

Similarity Search in Multimedia Databases

Habilitation thesis

Tomáš Skopal
November 2006

tomas.skopal@mff.cuni.cz
urtax.ms.mff.cuni.cz/skopal

Charles University in Prague
Faculty of Mathematics and Physics
Department of Software Engineering
Malostranské nám. 25
118 00, Prague 1
Czech Republic

Typeset in pdfLaTeX

Contents

Preface

The proposed thesis presents selected results of the author's research in the area of similarity search in multimedia databases (and related ones). The research has been carried out at VŠB-Technical University of Ostrava, Faculty of Electrical Engineering and Computer Science (2001–2004), and at the Charles University in Prague, Faculty of Mathematics and Physics (2004–2006).

The area of similarity search in multimedia databases could be identified as an important and quickly emerging research topic in the modern database technologies. In a broader meaning, as a multimedia database we understand a collection of data instances which have no unique structure and semantics, like audio/video/image documents, document-centric XML and full-text documents, biometric databases, DNA/protein databases, databases of 3D models, time series, and many others. Since we usually want to retrieve the mentioned data based on its content, it cannot be processed by conventional dabatase technologies, like the (object-)relational DBMS. Thus, there is a need to process the data by specific access methods, in a way that queries can be evaluated *efficiently* (quickly) and *effectively* (following the human's expectations with respect to the quality of query result).

This thesis addresses mainly the efficiency issues and to some extent also the effectiveness issues. We present the results as a collection of 8 selected papers fit into a single framework, where each paper is focused on a particular problem. The papers are presented in separate chapters (2–9) in their camera-ready forms (6 in LNCS-Springer proceedings, 1 in ACM proceedings, 1 in local proceedings), while the unifying commentary is included in Chapter 1. Prior to summarizing the papers, the commentary briefly surveys selected state-of-the-art results. In order to provide quick navigation, the references to the author's original contributions are marked with a bulb (see on the left). Every bulb occurrence refers to a publication included as a single chapter in this thesis. In Chapter 10 we conclude the thesis and outline some directions of current and future research.

The selected papers have been chosen in order to highlight the author's main achievements in the area of similarity search. The modifications of M-tree and PM-tree index structures have been proved to significantly speedup the retrieval

of objects from a multimedia database, based solely on their content (i.e. we consider the content-based similarity retrieval). Furthermore, the author has proposed an approach to *exact* indexing of non-metric data. To the best of author's knowledge, there was not proposed a solution of this problem before (omitting the trivial sequential search). Moreover, the proposed non-metric approach reuses the metric access methods (e.g. the M-tree or PM-tree), thus an integration of the non-metric search into the existing metric retrieval systems can be accomplished simply by adding a preprocessing module. The approximate metric search by semi-metric transformations is another result, allowing to trade the precision of metric similarity search for a gain in performance.

The papers and the respective related work served as a foundation for a brand new course (*DBI030 - Similarity search in multimedia databases*) lectured at the Department of Software Engineering, FMP (started by the author in 2005). Furthermore, since 2006 the author supervises a Ph.D. student whose thesis topic is focused into the area of similarity search in biological databases.

The research included in the selected papers has been supported by several grants – GAČR 201/05/P036 (author's post-doc grant), GAČR 201/06/0756, "Information society" 1ET100300419, GAČR 201/03/0912, GAČR 201/03/1318, GAČR 201/00/1031. The author is a member of DISG research group at the Department of Software Engineering, FMP, where he carries out research in the area of similarity search and database indexing.

Acknowledgments

I would like to thank Jaroslav Pokorný, František Plášil, Peter Vojtáš and Antonín Říha, who have facilitated excellent conditions for my research, gave me valuable advices, professional and pedagogical support. Also, I have to thank Václav Snášel, Michal Krátký and Pavel Moravec from VŠB-Technical university of Ostrava for fruitful cooperation in our joint research activities.

Prague, November 2006 Tomáš Skopal

Chapter 1

The Commentary

1.1 Introduction

In recent years, the volume of available multimedia data has grown rapidly, so the multimedia retrieval systems and multimedia databases are becoming more important than ever. As we see the progress in the fields of acquisition, storage, and dissemination of various multimedia formats, the application of effective and efficient multimedia management systems becomes indispensable in order to handle all these formats. The application domains for multimedia retrieval include image/audio/video databases, CAD databases, but also molecular-biologic and medicine databases, geographical information systems, biometric databases and many others. In particular, more than 95% of web space is considered to store multimedia content, other multimedia data is stored in corporate and scientific databases, personal archives and digital libraries.

Due to the quick growth of multimedia data volumes, the *text-based* multimedia retrieval systems become useless, since the requirements on textual annotation (often manual) exceed human possibilities and resources. The *metadata-based* search systems are of a similar kind, we need an additional explicit information to effectively describe multimedia objects (e.g. structured semantic description, as class hierarchies or ontologies), which is not available in most cases.[1]

The only practicable way how to process and retrieve the vast volumes of raw multimedia data is the *content-based similarity search*[2], i.e. we consider the real content of each particular DB object. Because the multimedia objects have no universal syntactic and semantic structure (unlike traditional strong-typed rows in relational database tables or XML with a schema), the most general and feasible abstraction used in multimedia retrieval is the *query-by-example* concept,

[1] The image search provided by Google is a successful example of text/metadata-based search engine, where the metadata is extracted from web pages wherein the images are encapsulated.

[2] Actually, there exist more models for unstructured search, like probabilistic models or simply a ranking, however, all these approaches perform a kind of *aggregation* that is used when offering query results to the user.

where the database objects are ranked according to similarity to a given query object (the example). Only such database objects are retrieved by the system, which have been ranked as sufficiently similar to the query object. The *similarity measure* returns a real-valued similarity score for any two *models* of multimedia objects on the input.

1.1.1 Dissimilarity spaces

The models of similarity retrieval depend on simplifying dissimilarity abstraction. Let a multimedia object $\mathcal{O}$ be modeled by a *model object* $O \in \mathbb{U}$, where $\mathbb{U}$ is a model universe. The universe can be a cartesian product of attribute sets, a domain of various structures (polygons, graphs, other sets, etc.), string closure, sequence closure, etc. A multimedia database $\mathcal{S}$ is then represented by a dataset $\mathbb{S} \subset \mathbb{U}$.

The *similarity measure* is defined as $s : \mathbb{U} \times \mathbb{U} \mapsto \mathbb{R}$, where $s(O_i, O_j)$ is considered as a similarity score of multimedia objects $\mathcal{O}_i$ and $\mathcal{O}_j$. In many cases it is more suitable to use a *dissimilarity measure* $\delta : \mathbb{U} \times \mathbb{U} \mapsto \mathbb{R}$ equivalent to a similarity measure $s(\cdot, \cdot)$ as $s(Q, O_i) > s(Q, O_j) \Leftrightarrow \delta(Q, O_i) < \delta(Q, O_j)$. A dissimilarity measure (or *distance*) assigns a higher score to less similar objects, and vice versa. The pair $\mathcal{D} = (\mathbb{U}, \delta)$ is called a *dissimilarity space*.

1.1.2 Metric distances

The distance measures often satisfy some of the metric properties ($\forall O_i, O_j, O_k \in \mathbb{U}$):

$$\begin{aligned}
\delta(O_i, O_j) &= 0 \quad \Leftrightarrow O_i = O_j && \text{reflexivity} \\
\delta(O_i, O_j) &> 0 \quad \Leftrightarrow O_i \neq O_j && \text{non-negativity} \\
\delta(O_i, O_j) &= \delta(O_j, O_i) && \text{symmetry} \\
\delta(O_i, O_j) + \delta(O_j, O_k) &\geq \delta(O_i, O_k) && \text{triangle inequality}
\end{aligned}$$

The *reflexivity* permits the zero distance just for identical objects. Both reflexivity and *non-negativity* guarantee every two distinct objects are somehow positively dissimilar. If δ satisfies reflexivity, non-negativity and *symmetry*, we call δ a *semimetric*. Finally, if a semimetric δ satisfies also the *triangle inequality*, we call δ a *metric* (or metric distance). The triangle inequality is a kind of transitivity property; it says if O_i, O_j and O_j, O_k are similar, then also O_i, O_k are similar. If there is an upper bound d^+ such that $\delta : \mathbb{U} \times \mathbb{U} \mapsto \langle 0, d^+ \rangle$, we call δ a *bounded metric*. In such case $\mathcal{M} = (\mathbb{U}, \delta)$ is called a (bounded) *metric space*.

To complete the enumeration, we also distinguish *pseudometrics* (not satisfying the reflexivity), *quasimetrics* (not satisfying symmetry) and *ultrametrics* (a stronger type of metric, where the triangle inequality is restricted to ultrametric inequality – $max\{\delta(O_i, O_j), \delta(O_j, O_k)\} \geq \delta(O_i, O_k)$).

1.1.3 Non-metric distances

The metric properties have been argued against by some theories in psychology and computer vision as restrictive in similarity modeling [40, 53]. In particular, the reflexivity and non-negativity have been refuted [32, 53] by claiming that different objects could be differently self-similar. For instance, in Figure 1.1a the image of a leaf on trunk can be viewed as positively self-dissimilar if we consider a distance which measures the less similar parts of the objects (here the trunk and the leaf). Conversely, in Figure 1.1b the leaf-on-trunk and leaf are treated as identical if we consider a distance which measures the most similar parts of the objects (the leaves). Nevertheless, the reflexivity and non-negativity are the less problematic properties.

The symmetry was questioned by showing that a prototypical object can be less similar to an indistinct one than vice versa [37, 38]. In Figure 1.1c, the more prototypical "Great Britain and Ireland" image is more distant to "Ireland" image than vice versa.

The triangle inequality is the most attacked property. Some theories point out the similarity has not to be transitive [3, 52]. Demonstrated by the well-known example, a man is similar to a centaur, the centaur is similar to a horse, but the man is completely dissimilar to the horse (see Figure 1.1d).

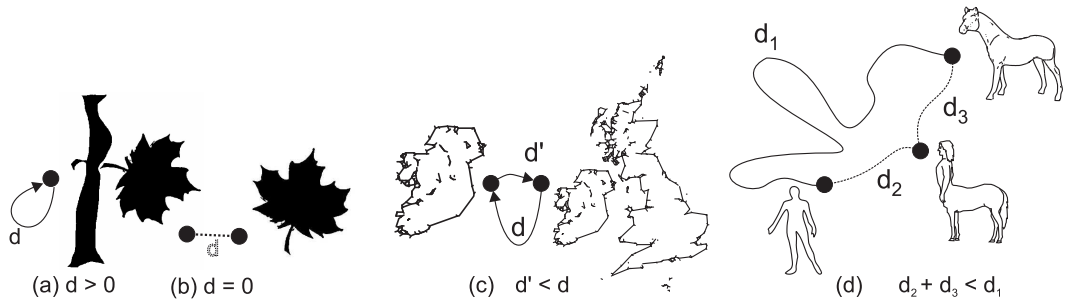


Figure 1.1: Objections against metric properties in similarity measuring: (a) reflexivity (b) non-negativity (c) symmetry (d) triangle inequality

1.1.4 Learning & Dynamic distances

We can identify also a kind of *dynamics preference* declaring whether the similarity can or cannot evolve over the time (and also during the process of retrieval) [29, 9]. The reason could be either learning (the similarity learns the human's cognition by e.g. relevance feedback) or just evolving due to the dynamic nature of similarity (some objects appear more or less similar in different time periods; we can also consider user profiles which adjust the similarity for each user).

When related to the process of retrieval, some approaches consider the query object as one of the factors which modifies the actual semantic of similarity in

given query context. In particular, in [13] the authors suggest dynamic combinations of metrics for more effective 3D retrieval. We can observe that such "multi-metric" approach improves the flexibility of similarity measuring, however, in a different way than the rich but "static" non-metric measuring. Unlike the static similarity measures, the topological properties of learning and dynamic distances can vary over the time.

1.1.5 Similarity Queries

In the following we consider the *query-by-example* concept; we look for objects similar to a query object $Q \in \mathbb{U}$ (Q is derived from an example multimedia object). Necessary to the query-by-example retrieval is a notion of *similarity ordering*, where the objects $O_i \in \mathbb{S}$ are ordered according to the distances to Q. For a particular type of query there is specified a portion of the ordering returned as the query result. The *range query* and the *k nearest neighbors (kNN) query* are the most popular ones[3]. A range query (Q, r_Q) selects all objects from the similarity ordering for which $\delta(Q, O_i) \leq r_Q$, where $r_Q \geq 0$ is a distance threshold (or query radius). A kNN query (Q, k) selects the k most similar objects (first k objects in the ordering).

Each particular query region is represented by a *ball* in the dissimilarity space, centered in Q and of radius r_Q. In a kNN query the r_Q radius is not known in advance, so it must be incrementally refined during the kNN query processing. The simplest implementation of similarity query evaluation is the *sequential search* over the entire dataset. The query object is compared against every object in the dataset, resulting in a similarity ordering which is used for the query evaluation. The sequential search often provides a baseline for other search methods.

1.2 Exact Metric Search

When considering (static) metric distances, the *metric access methods* (MAMs) provide data structures and algorithms by use of which the objects relevant to a similarity query can be efficiently (i.e. quickly) retrieved [57]. The MAMs build an auxiliary data structure, called *metric index*, so we also talk about metric indexing. The main principle behind all MAMs is a utilization of the triangle inequality property (satisfied by any metric), due to which MAMs can organize/index the objects of $\mathbb{S}$ into distinct classes. When a query is to be processed, only the candidate classes are searched (such classes which overlap the query), so the searching becomes more efficient (see Figure 1.2).

[3]There are more types of similarity queries – like reverse kNN query, (k-)closest pairs, similarity join, etc. – however, the range and kNN queries serve as primitives used to compose more complex query types.

The efficiency of a MAM depends not only on *I/O costs* (like spatial access methods, e.g. R-tree, do), the second important (and often the dominant) component are the *computation costs* – the number of distance computations needed to answer a query. The reason for focusing just on the computation costs is due to time complexities of the algorithms implementing dissimilarity measures. Although some distances are quite cheap, say of linear complexity according to the size of compared objects (as e.g. Minkowski L_p distances), some other distances are expensive. The sequence alignment distances (which cover also string-matching distances), e.g. dynamic time warping distance, edit distance, longest common subsequence, are typically implemented by dynamic programming, which exhibits quadratic time complexity. Some other distances are even extremely expensive, such as the earth mover's distance [39], which can be computed in exponential time by linear programming.

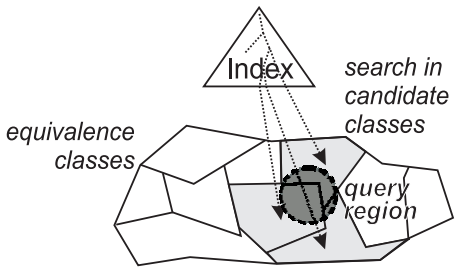


Figure 1.2: Classes of similar objects indexed by a metric access method.

There were developed many MAMs for different scenarios (e.g. designed to either secondary storage or main memory management). Besides others we name the *M-tree* [22], *vp-tree* [56], *(m)vp-tree* [8], *gh-tree* [54], *GNAT* [10], *SAT* [36], *LAESA* [35], or *D-index* [25]. The MAM-based similarity search is accomplished by applying metric properties to quickly prune the search space. Basically, the MAM classes are represented by data regions in the metric space which are described either by *ball regions* (or their compositions, e.g. rings), which is the most used representation (M-tree family, (m)vp-tree, D-index) or by *hyper-plane partitioning* (gh-tree, GNAT). During query processing, a candidate data region is checked whether it is overlapped by the query ball. In case of an overlap, the region has to be searched – this means either filtering of data objects (if the region contains already the data objects e.g. a tree leaf) or filtering of nested regions (when considering hierarchical MAMs, e.g. trees or D-index). In Figure 1.3 see several examples of MAMs – the M-tree, PM-tree, GNAT, mvp-tree, D-index.

Mapping Methods

An indirect way how to accomplish metric search is a mapping of the dataset into a low-dimensional vector space. There have been proposed various *mapping (or embedding) methods* [26, 30], e.g. MDS, FastMap, MetricMap, SparseMap,

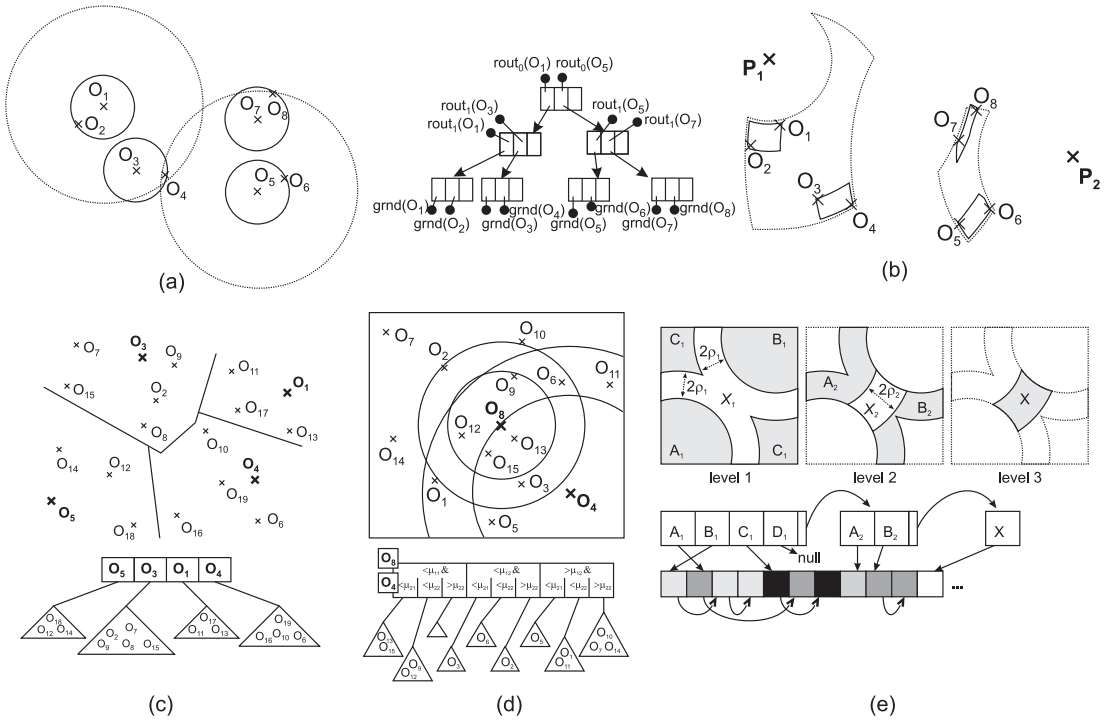


Figure 1.3: Several MAMs: (a) M-tree (b) PM-tree (c) GNAT (d) mvp-tree (e) D-index

to name a few. The dataset $\mathbb{S}$ is embedded into a vector space $(\mathbb{R}^k, \delta_V)$ by a mapping $F : \mathbb{S} \mapsto \mathbb{R}^k$, where the distances $\delta(\cdot,\cdot)$ are (approximately) preserved by a cheap vector metric δ_V (often the L_2 distance). In many cases the mapping F is *contractive*, i.e. $\delta_V(F(O_i), F(O_j)) \leq \delta(O_i, O_j)$, which allows to filter out some irrelevant objects using δ_V, but some other irrelevant objects, called *false hits*, must be re-filtered by δ (see e.g. [27]). The mapped vectors can be indexed/searched by any MAM, however, since the data is mapped to vector space, we can utilize also *spatial access methods* [7], like R-tree, X-tree or VA-file.

A particular method based on mapping is LAESA, where a contractive mapping of the metric space to $(\mathbb{R}^k, L_\infty)$ is constructed using k *pivots* $P_i \in \mathbb{S}$. The mapping function F turns an object O_i to a vector $(\delta(P_1, O_i), \delta(P_2, O_i), \ldots, \delta(P_k, O_i))$. When searching, a range query (Q, r_Q) is mapped to the target space as $(F(Q), r_Q)$, see Figure 1.4 (kNN queries are processed in a similar way). The retrieved candidate objects (here O_1, O_4) have to be refiltered to eliminate possible false hits (here O_4). There have been proposed many heuristics to choose the optimal set of pivots, in general, the good pivots are far from each other and tend to be outliers (outside the dataset) [15]. To say the drawbacks, mapping methods are expensive, while the distances are preserved only approximately, which leads to *false dismissals* (i.e. to relevant objects being not retrieved). The contractive methods eliminate the false dismissals but suffer from a great number of false

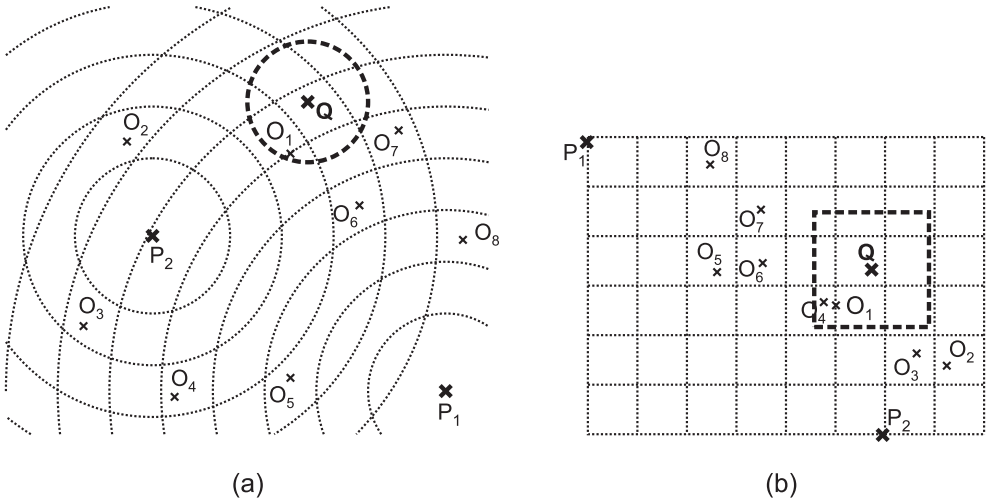


Figure 1.4: Mapping from (a) source metric space to (b) target vector space

hits (especially when k is low), which leads to lower retrieval efficiency. In most cases the methods need to process the dataset (or choose the pivots) in a batch, so they are suitable for static MAMs only.

1.3 The M-tree Family

The M-tree [22] (and its variants) is a popular MAM designed for database environments. As based on B$^+$-tree, the M-tree is a paged, dynamic and balanced index structure (see Figure 1.3a). Its inner nodes contain *routing entries* which describe ball-shaped metric regions which (recursively) bound the underlying data objects in leaves. The leaf nodes consist of *ground entries* – the indexed data objects themselves. In addition to the B-tree-inherited invariants (minimal node utilization and balance), a correct M-tree hierarchy must satisfy the *nesting condition*. The nesting condition says every region ball (in a routing entry) must spatially bound all the data objects stored in leaves of the respective subtree, i.e. all the data in subtree must fall into that ball. During query processing, all such nodes must be visited the region balls of which overlap the query ball.

In recent years, the M-tree has been modified or improved either to achieve better performance, or to extend the query model. The former case modifications include the *Slim-tree* [49] (cheaper splitting of nodes and redistribution of ground entries to obtain more compact regions), and the *M$^+$-tree* [59] (employs twin-nodes to better partition the dataset when using an Euclidean space). The latter case includes the *QIC-M-tree* [21] (support of user-defined query distance lower-bounding the indexing metric), and the *M^2-tree* [20] (support of multiple metrics within a single index).

1.3.1 Compact Hierarchy of M-tree

Since the M-tree's nesting condition is very weak, the efficiency of search in a given dataset is significantly affected by particular M-tree hierarchy, even though the correctness and the logic of search are guaranteed for all M-tree hierarchies satisfying the nesting condition. The key problem of M-tree search efficiency resides in:

1. the overall volume[4] of M-tree regions defined by routing entries. The larger volume, the higher probability of an overlap with query region and, consequently, the higher search costs.

2. the quantity of overlaps among metric regions. We must realize the query processing has to access all nodes the parent metric regions of which overlap the query region. If the query region lies (even partially) in an overlap of two or more regions, all the appropriate nodes must be accessed, thus the search costs grow.

Originally, the algorithms on M-tree have been developed to achieve a trade-off, an efficient construction and a (relatively) efficient searching. Consequently, the M-tree construction techniques incorporate decision moments, that regard only a partial knowledge about the distance distribution in a given dataset. To obtain quick insertion of a new object, the original algorithm guides the insertion along just a single path in the M-tree (the single-way insertion). Using this single-way insertion, the M-tree hierarchy is constructed locally – at a moment when the nodes are about to split. On the other side, the bulk loading algorithm [19] on M-tree works with the entire dataset, however, it also works locally – according to several sample objects. The local construction methods cause the M-tree hierarchies are not compact enough, which increases the overall volume of metric regions as well as the quantity of overlaps among them.

In our approach, we wanted to utilize also global techniques of (re)building M-tree, so that the M-tree hierarchy becomes reasonably optimized. In order to improve the search efficiency at the expense of construction costs, in Chapter 2 we propose two global methods of constructing more compact M-tree hierarchies [46] – the *generalized slim-down algorithm* and the *multi-way object insertion*. The motivation for such efforts has been well-founded by a common DBMS scenario, in which the database (the dataset $\mathbb{S}$, respectively) is updated only occasionally (the dynamic insertions/deletions are not frequent) but, on the other hand, there are many queries issued at a moment. In such a scenario, we rather favor to speedup the search process, while the costs of index updating are not so important. Following this idea, the two proposed methods decrease both the overlaps

[4] Actually, in metric spaces we cannot speak about volume in the vector-spatial meaning, nevertheless, without the loss of generality, we can assume larger covering radius implies larger volume and vice versa.

among metric regions as well as the overall volume, which leads to a higher search efficiency.

In the former case, the slim-down algorithm is a post-processing technique which tries to move entries (both ground and routing entries) from their source nodes to "better" nodes located at the same level of the M-tree. A "better" node is that one, the region ball of which has not to be enlarged due to the movement and, moreover, the region ball of the source node can be (spatially) reduced. In the latter case, the multi-way insertion extends the searching for a target leaf in a way that multiple paths of the M-tree are traversed in order to find the globally optimal leaf for insertion of a new object. This leads to compact M-tree hierarchy as well as to higher utilization of nodes (we prefer insertion into non-full nodes).

1.3.2 Compact Region Shape: PM-tree

As we have discussed previously, the efficiency of search in M-tree is dependent on the overall volume of metric regions. The higher volume, the lower search efficiency. In the previous section, we have presented two ways of reducing the overall volume using *object redistribution* but, however, the redistribution alone is not an ultimate solution and, moreover, it is computationally expensive.

In order to achieve even higher volume reduction and to keep the construction costs low, we consider also another reduction of region volume – a modification of metric region *shape*. Each metric region of M-tree is described by a bounding ball (defined by a local pivot and a covering radius). However, the shape of ball region is far from optimal, because it does not bound the data objects tightly together, thus the region volume is too large. In other words, relatively to the ball volume, there is only "few" objects spread inside the ball, while a huge proportion of empty space[5] is covered. Consequently, for ball regions of large volumes the probability of overlap with a query region is high, thus query processing becomes less efficient.

On the other side, the tightest possible boundary for a set of objects (i.e. a boundary for which the proportion of dead space is zero) is the set of objects themselves. Unfortunately, the simple description of such a "grain region" is useless, since storage of all the objects is too large, and an overlap check with a query region would take many distance computations. In fact, checking a "grain region" for an overlap is equivalent to sequential search over all the objects stored in the appropriate covering subtree.

Keeping the previous observations in mind, we can formulate four requirements on a compact metric region shape (a trade-off between region volume and storage/computation costs), bounding a given set of objects:

- The representation of a region stored in a routing entry should be as small as possible, so that storage of all inner nodes is (by far) smaller than storage

[5]The uselessly indexed empty space is often referred to as the "dead space" [7].

of all leaves.

- The shape of region should be easy to check for an overlap with the query region (query ball respectively).
- The shape should be *compact*, it should bound the objects tightly together, so that probability of an empty overlap with the query region (i.e. a case that no indexed objects are located in the overlap) is minimal.
- Given a set of regions, it should be easy to create a super-region which bounds all the (data in the) regions. This requirement is tree-specific – it ensures that creating a super-region (when splitting an inner node) can be automatically handled. Moreover, the requirement guarantees the nesting condition (introduced for M-tree) is still preserved.

As a rise to the challenge described above, we have proposed an extended variant of M-tree – the *PM-tree* [43, 47], where the shape of ball regions is further cut off by a combination of *rings* (see Figure 1.3b and Chapter 3). The rings share a single set of p global pivots, so the PM-tree can be regarded as a hybrid structure combining the local pivot hierarchies with global pivot-based methods. In more detail, each of the p rings belonging to a given routing/ground entry is stored as two real numbers – the smaller and the larger radius (coded to a two-byte approximation in case of routing entry and one-byte approximation in case of ground entry). The pivots themselves are stored separately, while the numbers of pivots used for routing entries and ground entries are chosen separately. We present theoretical cost model for range queries performed on the PM-tree. It has been experimentally shown that the PM-tree can outperform the M-tree significantly (by up to an order of magnitude).

Besides the range query algorithm, we have introduced also the kNN algorithm for PM-tree [48] (see Chapter 4). In addition to the filtering extensions used in PM-tree's range query, for the kNN query we have proposed modifications to the distance lower and upper bounds being used by the branch-and-bound kNN algorithm (the lower bounds are used in the priority queue of pending requests, while the upper bounds are used in the array of kNN candidates). We have proved that the modified kNN algorithm is optimal in terms of I/O costs (i.e. that I/O costs of equivalent range query are the same). The cost model for kNN search in PM-tree was presented in [42].

1.4 Search in Multi-metric Spaces

A recent proposal aiming to improve the effectiveness of similarity search (i.e., the quality of the retrieved answer) resorts to the use of *combinations of metrics*

[11, 12]. Instead of using a single metric to compare two objects, the search system uses a *linear combination of metrics* to compute the (dis)similarity between two objects. The Figure 1.5 shows an example of the benefits obtained by using combinations of metrics. The first two rows show objects retrieved by a 3D similarity search system using two different single-feature vectors. In both queries, the result includes some non-relevant objects (false hits). The third row shows the result of the search when using a combination of both feature vectors – only relevant objects are retrieved in this case.

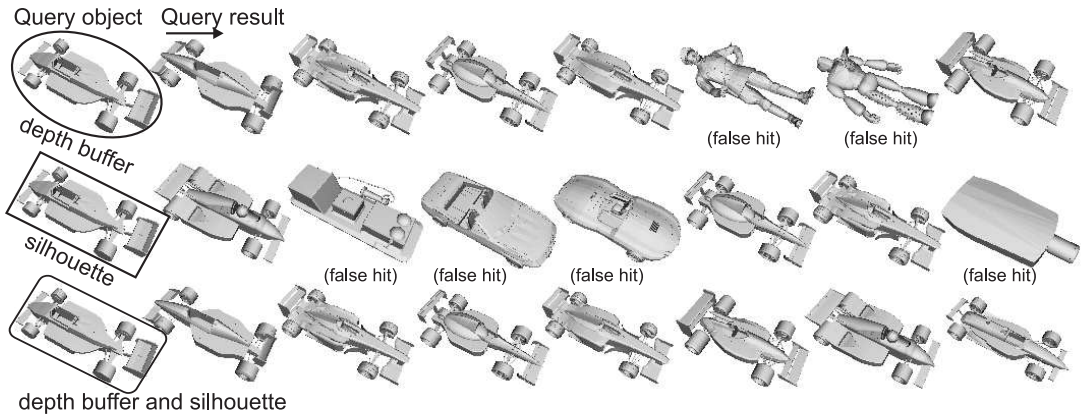


Figure 1.5: Improving effectiveness of 3D similarity search by combining two 3D feature vectors.

To further improve the effectiveness of the search system, *methods for dynamic combinations of metrics* have been proposed [13], where the query processor weighs the contribution of each metric *depending on the query object* (as mentioned in Section 1.1.4). Therefore, instead of a single metric, to perform a given similarity query the system uses a *dynamic metric function* (multi-metric) – a *query-weighted* linear combination of the partial metrics. The weights for a particular query object can be computed arbitrarily (they have to be in $\langle 0, 1 \rangle$), while as a successful technique for query-dependent weights construction the entropy impurity has been used in 3D retrieval [12].

In Chapter 5 the *Multi-Metric M-tree (M^3-tree)* is presented, a dynamic index structure that extends the M-tree to support multi-metric similarity queries [16]. We first describe how to adapt the search algorithms of the original M-tree to directly support multi-metric queries. The idea is to index the dataset by an upper-bounding metric, which is a linear combination (of the underlying partial metrics) where all the weights are set to 1. Then, any query-dependent combination is a lower bound to the index metric and can be thus utilized in filtering non-relevant M-tree subtrees. The disadvantage of this approach arises at a moment when the weights span a substantial part of the interval $\langle 0, 1 \rangle$ (we suppose at least one weight is always set to 1). In such case the indexing metric is a very

loose upper bound to the respective query metric, so the filtering effectiveness deteriorates.

To overcome this drawback, we describe the M^3-tree data structure and new similarity search algorithms. The radii/distances stored in the M^3-tree routing/ground entries are extended by a compact signature which approximates the partial distances aggregated within the radii/distance values. Due to this extension we can create a tight upper bound to a query metric, regardless of which weights have been used. We show experimentally that the M^3-tree outperforms the adapted M-tree, and that its efficiency is very close to having multiple M-trees, one for each used multi-metric, which is the optimal achievable efficiency regarding to this index structure.

1.5 Non-metric Search

As mentioned in Section 1.1.3, the metric properties can be viewed as a serious limitation in similarity modeling. The similarity search, therefore, should allow also non-metric measures. The non-metric measures have already been used in multimedia databases and in information retrieval. A common rationale for their usage is the robustness – a *robust* measure is resistant to outliers, i.e. to anomalous or "noisy" objects. In an "intra-object" meaning, a robust measure can neglect some portions of the measured objects which appear as the most dissimilar.

Another "vote" for non-metric measures is the complexity of similarity modeling. In addition to distance measures based on simple description (like the L_p distances), some measures are very complex and, therefore, for them a "manual" enforcement of metric properties is nearly impossible. As an example, the COSIMIR model [34] consists of three-layer backpropagation network, which can be trained to model an arbitrary user-defined similarity measure (but hardly a metric one).

The third reason for non-metric measuring is the fact we have often insufficient information about a dissimilarity measure provided by the user. Besides the analytical descriptions of various measures (even the very complex ones like the COSIMIR), we can design a similarity measure which can be described solely by an algorithm written in context-free language – as a black box returning a real-value output on a two-object input. The topological properties (the metric axioms, in our case) of an algorithmically described similarity measure are generally undecidable, so we have to treat such a measure as a non-metric. Due to the black-box abstraction, we can even consider hardware-supported similarity measures (e.g. the FPGA devices) [28].

In our recent research [41], we have proposed a general method of non-metric search by metric access methods (see Chapter 6). We show the triangle inequality can be enforced for any *semimetric* (reflexive, non-negative and symmetric dissim-

ilarity measure), resulting in a metric that preserves the original similarity orderings (and so the retrieval effectiveness). The idea is to apply a concave increasing function (so-called *triangle-generating modifier*) on the semimetric. When considering all triplets of the dataset's objects and the appropriate distances among them, the distance triplets generated by a semimetric are not triangular, i.e. they represent the direct effect of a triangle inequality violation. However, the concave modifiers have an interesting property – they turn the non-triangular distance triplets into triangular ones, hence, when given a suitable modifier, the triangle inequality becomes valid for the modified semimetric (making it a metric). Naturally, among the infinitely many triangle-generating modifiers, only some of them are suitable to be used for metric indexing. This is due to the "declustering" effect of triangle-violating modifiers – the modified distances "inflate" the space so that clusters become more or less indistinct. From another point of view, the "inflating" modifications lead to a kind of analogy to the curse of dimensionality, however, in metric spaces we rather speak about a high *intrinsic dimensionality* [18]. A high intrinsic dimensionality implies more overlapped data regions maintained by a MAM, so the intrinsically high-dimensional datasets are hard to be indexed. Keeping these observations in mind, we have designed the *TriGen* algorithm for turning any black-box semimetric into (approximated) metric, just by use of distance distribution in a fraction of the database. The algorithm finds such a modification for which the intrinsic dimensionality is minimized (so the retrieval efficiency is maximized), considering any metric access method. Furthermore, since some semimetrics can be turned into exact metrics only at the cost of very inefficient search (deteriorating to almost sequential scan), we could prefer a modification into an approximated metric where the triangle inequality is preserved only partially. This allows us to trade the retrieval performance for a certain level of retrieval imprecision.

1.6 Approximate Search

Unlike exact-match queries in traditional databases, the similarity measuring and retrieval in multimedia databases is inherently imprecise, subjective and changing over time. Thus, we might prefer faster but approximate methods which could retrieve some non-relevant objects (false hits) and miss some relevant ones (false dismissals). In many cases, the efficiency gain can be traded for an acceptable loss in effectiveness. Nevertheless, in some cases the similarity is precisely defined and then we require the search to be as exact as possible (e.g. biometric identification tasks). The problem of efficient search is especially hard when considering high-dimensional databases. Nowadays, an efficient search in (intrinsically) high-dimensional datasets is feasible solely by usage of approximate methods.

Many of the (exact) metric access methods have been modified to accomplish

also the approximate search. In particular, in [58] authors suggest three heuristics for *approximately correct search* (AC), where the objects in the answer are guaranteed to be close to the desired results. However, the AC search is still quite exact and the gain in efficiency is not very high when considering high-dimensional data. Hence, to obtain a method that is by an order of magnitude faster than the exact ones, we have to resort to *probabilistic search* [18, 14, 2]. Unlike the AC methods where all the objects in the query result are more or less close to our expectations, the probabilistic methods mostly cannot guarantee any level of "result goodness". They rather guarantee an answer will contain the desired objects with a certain probability. A hybrid approach to AC and probabilistic search are the *probably approximately correct* (PAC) methods, which even more reduce the search costs, but also even more back off the precision requirements [20]. Besides exact methods adjusted to be usable also for the approximate case, there were special indexing structures developed, e.g. the *Clindex* [33], the *VQ-file* [51], or the *buoy indexing* [55].

1.6.1 Semimetric Modifications

A way to an approximate search in metric spaces can be a transformation of the metric space into another space. However, unlike the mapping methods which perform a mapping into a vector space (see Section 1.2), in our approach [45] (see Chapter 7) we have proposed a transformation into a semimetric space. The mapping is achieved by so-called *triangle-violating* functions (convex increasing functions), which preserve the original similarity orderings but violate the triangle inequality of the metric being modified. Thus, we obtain a semimetric which is used instead of the metric.

In fact, this is an opposite approach to the non-metric triangle-generation modifications (as presented in the previous section), hence, also the effects of the modifications are inverse. In particular, the distance distribution (according to the modified semimetric) exhibits increased variance and lower mean, so the intrinsic dimensionality is lower than that of the original non-modified metric. From another point of view, some of the triangular triplets generated by the original metric are turned into non-triangular ones, so the metric becomes only a semimetric, while usage of such a measure by MAMs leads to only approximate retrieval (e.g. some subtrees in M-tree are filtered incorrectly). Nevertheless, the loss in retrieval precision (the effectiveness, actually) is traded for a significant gain in retrieval efficiency. The efficiency improvement can reach up to an order of magnitude, while the loss in retrieval precision could be less than a few percent. The level of retrieval precision (relative precision and recall) can be controlled by a *convexity weight* of the modifying triangle-violating function.

The experimental results performed on extremely high-dimensional datasets (240,000-dimensional vectors representing text documents) have shown that semimetric search can successfully fight the curse of dimensionality, while the loss in

effectiveness can be low or moderate (a few percent).

1.6.2 Modified LSI for Efficient Indexing

We have reused the concept of triangle-violating modifiers in another area related to similarity search – in the *latent semantic indexing* (LSI). The classic LSI model applies the singular-value decomposition (SVD) to the vector model in Text retrieval [5].

Basically, a text collection consists of m unique terms and each of the n documents in the collection is represented by an m-dimensional vector of frequencies (or weights) of terms in that document. The entire collection is represented by a matrix A. Using the singular-value decomposition (SVD) of the matrix A

$$A = U\Sigma V^T$$

we obtain so-called *concept vectors* (left-singular vectors – the columns in U), which can be interpreted as individual (semantic) topics hidden in the collection. The concept vectors form a basis in the original high-dimensional vector space, while they are actually linear combinations of terms (the terms are supposed to be independent). An important property of SVD is a fact that concept vectors are ordered according to their "significance", which is determined by values of the singular values σ_i stored in descending order in the diagonal matrix Σ. Informally, the concept significance says in what quantity is the appropriate concept globally present (or missing) in the collection. It also says which concepts are semantically important and which are not (that is where the "latent semantics" came from) – such unimportant concepts are, in fact, a "semantic noise". The columns of ΣV^T contain document vectors $O_i \in \mathbb{S}$ (the *pseudo-document vectors*), but these are now represented in the basis U, i.e. in the *concept basis* (unlike the original term basis). Every pseudo-document vector describes a linear combination of the concept vectors, i.e. the appropriate document consists somehow (positively or negatively) of every concept found. The pseudo-document vectors are then used to perform similarity search on the text collection, where the *cosine measure* is widely used as similarity measure (in both LSI and classic vector model).

Moreover, because of varying significance of the concept vectors, the less significant concepts can be omitted, so we get a kind of dimensionality reduction – the *k-reduced SVD* (we consider just the k most significant concepts; in practice this means a reduction from 10^5 to a few hundred dimensions/concepts). Interestingly, it was experimentally shown that the k-reduced SVD does not worsen the precision of similarity search [24, 6]. In fact, the k-reduced SVD can perform even better that the classic vector model – in particular, it can partially eliminate some negative aspects, like the problems of synonymy and homonymy.

Since the values in columns of V^T are distributed uniformly, the "descend rate" of singular values σ_i in the matrix Σ determines the amount of correlation

between individual coordinates of vectors in ΣV^T. The higher descend rate, the greater correlations and also the lower *intrinsic* dimensionality of the vectors (considering any L_p metric and also the cosine measure). In [44] (see Chapter 8) we have proposed a variant of LSI (*the σ-LSI*) where the descend rate of singular values is increased by application of a suitable triangle-violating modifier. We can understand the modification of Σ as an additional dimensionality reduction, in this case a reduction of intrinsic dimensionality. Although the σ-LSI leads to an approximation of the original decomposition (thus we get data representations which lead to only an approximate search with respect to the original LSI), the gains in search efficiency can be considerable. Note that, unlike the previous approach to semimetric search where the modifiers have been used directly on the metric employed, here the modifiers are applied on the data, i.e. we perform a kind of data transformation rather than a dissimilarity transformation.

1.7 Similarity search in XML databases

During the last decade, the XML (eXtensible Markup Language) has flooded many branches of computer science [1]. The XML structure became a basis for many communication protocols (like SOAP, XMLP), document and presentation formats (e.g. DocBook, XHTML, OpenOffice and MS Word 2003 documents) as well as various data exchange formats (e.g. WSDL used by web services). The XML phenomenon has penetrated even into the stronghold of relational databases – many DBMSs use a kind of XML-based format as an exchange medium for migration or export/import of data. Besides classic DBMSs supporting XML, there arise new database systems designed to store XML data in its native form (so-called *native XML databases*). The management of native XML databases cannot be efficiently performed by traditional methods of data management used in (object-)relational DBMSs, hence, specialized techniques have been developed during last years [23, 17].

Unlike the relational data, the XML data (or documents) can be fully structured (data-oriented XML), semi-structured (document-oriented XML), but also completely unstructured (also document-oriented XML). The former two cases assume a kind of database schema which the data must conform to, e.g. a DTD or XML Schema. Such a schema provides syntactic and also semantic information about the content of a particular XML document; in a similar way as a relational schema describes a particular table. In the unstructured case, however, the XML documents are completely unrestricted, so we are not able to exactly interpret individual XML elements and so we have to treat such documents in a different way. In particular, due to the absence of schema, the user cannot issue well-formed queries (written in XPath or XQuery languages), he or she rather has to use some other means of querying; similarly as a full-text querying is performed. A possible approach to querying unstructured XML data is similarity search,

where the (parts of) documents are matched against a similarity query. Unlike the full-text search, there is an additional information that should be taken into account, the document hierarchy (an XML tree or graph)[6].

In [31] (see Chapter 9) we have proposed an approach to similarity search in XML databases, where all the XML paths extracted from all the documents in a database are indexed (the paths are labeled with the particular XML document id they belong to). The paths are indexed using a cumulated metric (a linear combination of metrics on individual elements), while the similarity can be measured either on the names of path elements/attributes, on the content of elements/attributes, or on both.

[6]The mentioned query languages (XPath and XQuery) have been recently extended to support full-text-like similarity search [4, 50].

Chapter 2

Revisiting M-tree Building Principles

Tomáš Skopal
Jaroslav Pokorný
Michal Krátký
Václav Snášel

Revisiting M-tree Building Principles [46]

Regular paper at the 7th East European Conference on Advances in Databases and Information Systems (ADBIS 2003), Dresden, Germany, September 2003

Published in the *Lecture Notes in Computer Science* (LNCS), vol. 2798, pages 148–162, *Springer-Verlag*, ISSN 0302-9743, ISBN 978-3-540-20047-5

Revisiting M-tree Building Principles

Tomáš Skopal[1], Jaroslav Pokorný[2], Michal Krátký[1], and Václav Snášel[1]

[1] Department of Computer Science,
VŠB–Technical University of Ostrava, Czech Republic
{tomas.skopal, michal.kratky, vaclav.snasel}@vsb.cz
[2] Department of Software Engineering,
Charles University, Prague, Czech Republic
jaroslav.pokorny@ksi.ms.mff.cuni.cz

Abstract. The M-tree is a dynamic data structure designed to index metric datasets. In this paper we introduce two dynamic techniques of building the M-tree. The first one incorporates a multi-way object insertion while the second one exploits the generalized slim-down algorithm. Usage of these techniques or even combination of them significantly increases the querying performance of the M-tree. We also present comparative experimental results on large datasets showing that the new techniques outperform by far even the static bulk loading algorithm.

Keywords: M-tree, bulk loading, multi-way insertion, slim-down algorithm

1 Introduction

Multidimensional and spatial databases have become more and more important for different industries and research areas in the past decade. In the areas of CAD/CAM, geography, or conceptual information management, it is often to have applications involving spatial or multimedia data. Consequently, data management in such databases is still a hot topic of research. Efficient indexing and querying spatial databases is a key necessity to many interesting applications in information retrieval and related disciplines.

In general, the objects of our interests are spatial data objects. Spatial data objects can be points, lines, rectangles, polygons, surfaces, or even objects in higher dimensions. Spatial operations are defined according to the functionality of the spatial database to support efficient querying and data management. A spatial access method (SAM) organizes spatial data objects according to their position in space. As the structure of how the spatial data objects are organized can greatly affect performance of spatial databases, SAM is an essential part in spatial database systems (see e.g. [12] for a survey of various SAM).

So far, many SAM were developed. We usually distinguish them according to which type of space is a particular SAM related. One class of SAM is based on vector spaces, the second one uses metric spaces. For example, well-known data structures like kd-tree [2], quad-tree [11], and R-tree [8], or more recent ones like UB-tree [1], X-tree [3], etc. are based on a form of vector space. Methods for

indexing metric spaces include e.g. metric tree [14], vp-tree [15], mvp-tree [5], Slim-tree [13], and the M-tree [7].

Searching for objects in multimedia databases is based on the concept of similarity search. In many disciplines, similarity is modelled using a distance function. If the well-known triangular inequality is fulfilled by this function, we obtain metric spaces. Authors of [9] remind that if the elements of the metric space are tuples of real numbers then we get a finite dimensional vector space.

For spatial and multimedia databases there are three interesting types of queries in metric spaces: range queries, nearest neighbours queries, and k-nearest neighbours queries. A performance of these queries differs in vector and metric spaces. For example, the existing vector space techniques are very sensitive to the space dimensionality. Closest point search algorithms have an exponential dependency on the dimensionality of the space (this is called the curse of dimensionality, see [4] or [16]).

On the other hand, metric space techniques seem to be more attractive for a large class of applications of spatial and multimedia databases due to their advantages in querying possibilities. In the paper, we focus particularly on improvement of the dynamic data structure M-tree. The reason for M-tree lies in the fact that, except Slim-trees, it is still the only persistent metric index. In existing approaches to M-tree algorithms there is a static bulk loading algorithm with a small construction complexity. Unfortunately, a querying performance of above-mentioned types of queries is not too high on such tree.

We introduce two dynamic techniques of building the M-tree. The first one incorporates a multi-way object insertion while the second one exploits the generalized slim-down algorithm. Usage of these techniques or even combination of them significantly increases the querying performance of the M-tree. We also present comparative experimental results on large datasets showing that the new techniques outperform by far even the static bulk loading algorithm. By the way, the experiments have shown that the querying performance of the improved M-tree has grown by more than 300%.

In Section 2 we introduce shortly general concepts of the M-tree, discuss the quality of the M-tree structure, and introduce the multi-way insertion method. In Section 3 we repeat the slim-down algorithm and we also introduce here a generalization of this algorithm. Experimental results and their discussion are presented in Section 4. Section 5 concludes the results.

2 General Concepts of the M-tree

M-tree, introduced in [7] and elaborated in [10], is a dynamic data structure for indexing objects of metric datasets. The structure of M-tree was primarily designed for multimedia databases to natively support the similarity queries.

Let us have a metric space $\mathcal{M} = (D, d)$ where D is a domain of feature objects and d is a function measuring distance between two feature objects. A feature object $O_i \in D$ is a sequence of features extracted from the original database

object. The function d must be a metric, i.e. d must satisfy the following metric axioms:

$$\begin{aligned} d(O_i, O_i) &= 0 && && \text{reflexivity} \\ d(O_i, O_j) &> 0 && (O_i \neq O_j) && \text{positivity} \\ d(O_i, O_j) &= d(O_j, O_i) && && \text{symmetry} \\ d(O_i, O_j) + d(O_j, O_k) &\geq d(O_i, O_k) && && \text{triangular inequality} \end{aligned}$$

The M-tree is based on a hierarchical organization of feature objects according to a given metric d. Like other dynamic and persistent trees, the M-tree structure is a balanced hierarchy of nodes. As usually, the nodes have a fixed capacity and a utilization threshold. Within the M-tree hierarchy, the objects are clustered into metric regions. The leaf nodes contain entries of objects themselves (here called the ground objects) while entries representing the metric regions are stored in the inner nodes (the objects here are called the routing objects). For a *ground object* O_i, the entry in a leaf has a format:

$$grnd(O_i) = [O_i, oid(O_i), d(O_i, P(O_i))]$$

where $O_i \in D$ is the feature object, $oid(O_i)$ is an identifier of the original DB object (stored externally), and $d(O_i, P(O_i))$ is a precomputed distance between O_i and its parent routing object.

For a *routing object* O_j, the entry in an inner node has a format:

$$rout(O_j) = [O_j, ptr(T(O_j)), r(O_j), d(O_j, P(O_j))]$$

where $O_j \in D$ is the feature object, $ptr(T(O_j))$ is pointer to a covering subtree, $r(O_j)$ is a covering radius, and $d(O_j, P(O_j))$ is a precomputed distance between O_j and its parent routing object (this value is zero for the routing objects stored in the root). The entry of a routing object determines a metric region in space $\mathcal{M}$ where the object O_j is a center of that region and $r(O_j)$ is a radius bounding the region. The precomputed value $d(O_j, P(O_j))$ is redundant and serves for optimizing the algorithms upon the M-tree. In Figure 1, a metric region and

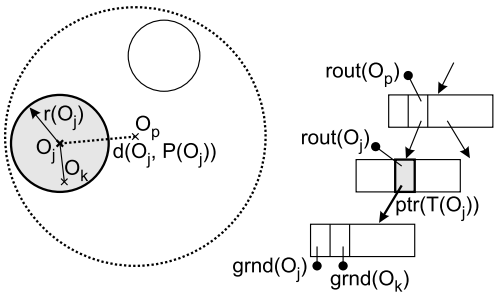


Fig. 1. A metric region and its routing object in the M-tree structure.

its appropriate entry $rout(O_j)$ in the M-tree is presented. For the hierarchy of metric regions (routing objects $rout(O)$ respectively) in the M-tree, only one invariant must be satisfied. The invariant can be formulated as follows:

• All the ground objects stored in the leafs of the covering subtree of $rout(O_j)$ must be spatially located inside the region defined by $rout(O_j)$. •

Formally, having a $rout(O_j)$ then $\forall O \in T(O_j), d(O, O_j) \leq r(O_j)$. If we realize, this invariant is very weak since there can be constructed many M-trees of the same object content but of different structure. The most important consequence is that many regions on the same M-tree level may overlap. An example

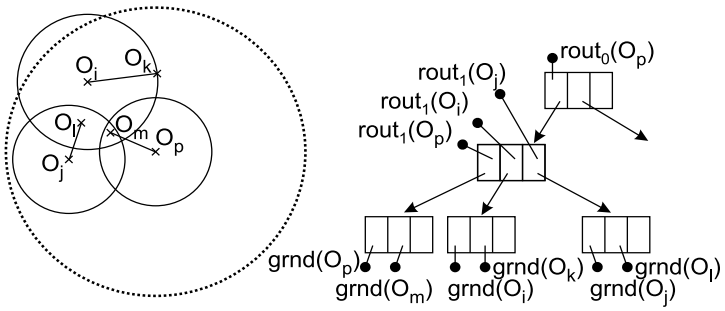


Fig. 2. Hierarchy of metric regions and the appropriate M-tree.

in Figure 2 shows several objects partitioned into metric regions and the appropriate M-tree. We can see that the regions defined by $rout_1(O_p)$, $rout_1(O_i)$, $rout_1(O_j)$ overlap. Moreover, object O_l is located inside the regions of $rout(O_i)$ and $rout(O_j)$ but it is stored just in the subtree of $rout_1(O_j)$. Similarly, the object O_m is located even in three regions but it is stored just in the subtree of $rout_1(O_p)$.

2.1 Similarity Queries

The structure of M-tree natively supports similarity queries. A similarity measure is here represented by the metric function d. Given a query object O_q, a similarity query returns (in general) objects close to O_q. The similarity queries are of two basic kinds: a *range query* and a *k-nearest neighbour query*.

Range Queries. A range query is specified as a *query region* given by a query object O_q and a query radius $r(O_q)$. The purpose of a range query is to return all the objects O satisfying $d(O_q, O) \leq r(O_q)$. A query with $r(O_q) = 0$ is called a *point query*.

***k*-Nearest Neighbours Queries.** A k-nearest neighbours query (k-NN query) is specified by a query object O_q and a number k. A k-NN query returns the first k nearest objects to O_q. Technically, the k-NN query can be implemented using the range query with a dynamic query radius. In practice, the k-NN query is used more often than the range query since the size of the k-NN query result is known in advance.

By the processing of a range query (k-NN query respectively), the M-tree hierarchy is being passed down. Only if a routing object $rout(O_j)$ (its metric region respectively) intersects the query region, the covering subtree of $rout(O_j)$ is relevant to the query and thus further processed.

2.2 Quality of the M-tree

As of many other indexing structures, the main purpose of the M-tree is its ability to efficiently process the queries. In other words, when processing a similarity query, a minimum of disk accesses as well as computations of d should be performed. The need of minimizing the disk access costs[3] (DAC) is a requirement well-known from other index structures (B-trees, R-trees, etc.). Minimization of the computation costs (CC), i.e. the number of the d function executions, is also desirable since the function d can be very complex and its execution can be computationally expensive. In the M-tree algorithms, the DAC and CC are highly correlated, hence in the following we will talk just about "costs".

The key problem of the M-tree's efficiency resides in a quantity of overlaps between the metric regions defined by the routing objects. If we realize, the query processing must examine all the nodes the parent routing objects of which intersect the query region. If the query region lies (even partially) in an overlap of two or more regions, all the appropriate nodes must be examined and thus the costs grow.

In generic metric spaces, we cannot quantify the volume of two regions overlap and we even cannot compute the volume of a whole metric region. Thus we cannot measure the goodness of an M-tree as a sum of overlap volumes. In [13], a *fat-factor* was introduced as a way to classify the goodness of the Slim-tree, but we can adopt it for the M-tree as well. The fat-factor is tightly related to the M-tree's query efficiency since it informs about the number of objects in overlaps using a sequence of point queries.

For the fat-factor computation, a point query for each ground object in the M-tree is performed. Let h be the height of an M-tree T, n be the number of ground objects in T, m be the number of nodes, and I_c be the total DAC of all the n point queries. Then,

$$fat(T) = \frac{I_c - h \cdot n}{n} \cdot \frac{1}{(m - h)}$$

[3] considering all logical disk accesses, i.e. disk cache is not taken into account

is the fat-factor of T, a number from interval $\langle 0, 1 \rangle$. For an ideal tree, the $fat(T)$ is zero. On the other side, for the worst possible M-tree the $fat(T)$ is equal to one. For an M-tree with $fat(T) = 0$, every performed point query costs h disk accesses while for an M-tree with $fat(T) = 1$, every performed point query costs m disk accesses, i.e. the whole M-tree structure must be passed.

2.3 Building the M-tree

By revisiting the M-tree building principles, our objective was to propose an M-tree construction technique keeping the fat-factor minimal even if the building efforts would increase.

First, we will discuss the dynamic insertion of a single object. The insertion of an object into the M-tree has two general steps:

1. Find the "most suitable" leaf node where the object O will be inserted as a ground object. Insert the object into that node.
2. If the node overflows, split the node (partition its content among two new nodes), create two new routing objects and promote them into the parent node. If now the parent node overflows, repeat step 2 for the parent node. If a root is split the M-tree grows by one level.

Single-Way Insertion. In the original approach presented in [7], the basic motivation used to find the "most suitable" leaf node is to follow a path in the M-tree which would avoid any enlargement of the covering radius, i.e. at each level of the tree, a covering subtree of $rout(O_j)$ is chosen, for which $d(O_j, O) \leq r(O_j)$. If multiple paths with this property exist, the one for which object O is closest to the routing object $rout(O_j)$ is chosen.

If no routing object for which $d(O_j, O) \leq r(O_j)$ exists, an enlargement of a covering radius is necessary. In this case, the choice is to minimize the increase of the covering radius. This choice is thightly related to the heuristic criterion that suggests to minimize the overall "volume" covered by routing objects in the current node.

The single-way leaf choice will access only h nodes, one node on each level, as depicted in Figure 3a.

Multi-Way Insertion. The single-way heuristic was designed to keep the building costs as low as possible and simultaneously to choose a leaf node for which the insertion of the object O will not increase the overall "volume". However, this heuristic behaves very locally (only one path in the M-tree is examined) and thus the most suitable leaf may be not chosen.

In our approach, the priority was to choose the most suitable leaf node at all. In principle, a point query defined by the inserted object O is performed. For all the relevant leafs (their routing objects $rout(O_j)$ respectively) visited during the point query, the distances $d(O_j, O)$ are computed and the leaf for which the

distance is minimal is chosen. If no such leaf is found, i.e. no region containing the O exists, the single-way insertion is performed.

This heuristic behaves more globally since multiple paths in the M-tree are examined. In fact, all the leafs the regions of which spatially contain the object O are examined. Naturally, the multi-way leaf choice will access more nodes than h as depicted in Figure 3b.

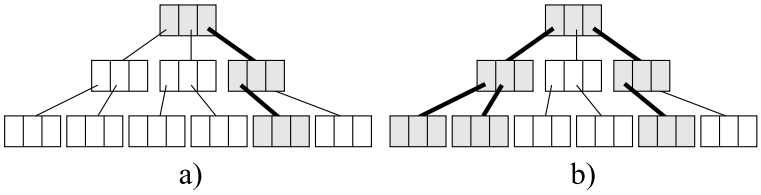


Fig. 3. a) Single path of the M-tree is passed during the single-way insertion. b) Multiple leafs are examined during the multi-way insertion.

Node Splitting. When a node overflows it must be split. According to keep the minimal overlap, a suitable splitting policy must be applied. Splitting policy determines how to split a given node, i.e. which objects to choose as the new routing objects and how to partition the objects into the new nodes.

As the experiments in [10] have shown, the `minMAX_RAD` method of choosing the routing objects causes the best querying performance of the M-tree. The `minMAX_RAD` method examines all of the $\frac{n(n-1)}{2}$ pairs of objects candidating to the two new routing objects. For every such a pair, the remaining objects in the node are partitioned according to the objects of the pair. For the two candidate routing objects a maximal radius is determined. Finally, such a pair $(rout(O_i), rout(O_j))$ for which is the maximal radius (the greater of the two radii $r(O_i), r(O_j)$) minimal is chosen as the two new routing objects.

For the object partition, a distribution according to *general hyperplane* is used as the beneficial method. An object is simply assigned to the routing object that is closer. For preservation of the minimal node utilization a fixed amount of objects is distributed according to the balanced distribution.

2.4 Bulk Loading the M-tree

In [6] a static algorithm of the M-tree construction was proposed. On a given dataset a hierarchy is built resulting into a complete M-tree.

The basic bulk loading algorithm can be described as follows: Given the set of objects $\mathcal{S}$ of a dataset, we first perform an initial clustering by producing k sets of objects $\mathcal{F}_1, \ldots, \mathcal{F}_k$. The k-way clustering is achieved by sampling k objects $O_{f_1}, \ldots, O_{f_k}$ from the $\mathcal{S}$ set, inserting them in the sample set $\mathcal{F}$, and then

assigning each object in $\mathcal{S}$ to its nearest sample, thus computing $k \cdot n$ distance matrix. In this way, we obtain k sets of relatively "close" objects. Now, we invoke the bulk loading algorithm recursively on each of these k sets, obtaining k sub-trees $\mathcal{T}_1, \ldots, \mathcal{T}_k$. Then, we have to invoke the bulk loading algorithm one more time on the set $\mathcal{F}$, obtaining a super-tree $\mathcal{T}_{sup}$. Finally, we append each sub-tree $\mathcal{T}_i$ to the leaf of $\mathcal{T}_{sup}$ corresponding to the sample object O_{f_i}, and obtain the final tree $\mathcal{T}$.

The algorithm, as presented, would produce a non-balanced tree. To resolve this problem we use two different techniques:

- Reassign the objects in underfull sets $\mathcal{F}_i$ to other sets and delete corresponding sample object from $\mathcal{F}$.
- Split the taller sub-trees, obtaining a shorter sub-trees. The roots of the sub-trees will be inserted in the sample set $\mathcal{F}$, replacing the original sample object.

A more precise description of the bulk loading algorithm can be found in [6] or [10].

3 The Slim-Down Algorithm

Presented construction mechanisms incorporate decision moments that regard only a partial knowledge about the data distribution. By the dynamic insertion, the M-tree hierarchy is constructed in a moment when the nodes are about to split. However, splitting a node is only a local redistribution of objects. From this point of view, the dynamic insertion of the whole dataset will raise a sequence of node splits – local redistributions – which may lead to a hierarchy that is not ideal.

On the other side, the bulk loading algorithm works statically with the whole dataset, but it also works locally – according to a randomly chosen sample of objects.

In our approach we wanted to utilize a global mechanism of (re)building the M-tree. In [13] a post-construction method was proposed for the Slim-tree, called as *slim-down* algorithm. The slim-down algorithm was used for an improvement of a Slim-tree already built by dynamic insertions. The basic idea of the slim-down algorithm was an assumption that a more suitable leaf exists for a ground object stored in a leaf. The task was to examine the most distant objects (from the routing object) in the leaf and try to find a better leaf. If such a leaf existed the object was inserted to the new leaf (without the need of its covering radius enlargement) and deleted from the old leaf together with a decrease of its covering radius. This algorithm was repeatedly applied for all the ground objects as long as the object movements occured.

However, the experiments have shown that the original (and also cheaper) version of the slim-down algorithm presented in [13] improves the querying performace of the Slim-tree only by 35%.

3.1 Generalized Slim-Down Algorithm

We have generalized the slim-down algorithm and applied it for the M-tree as follows:

The algorithm separately traverses each level of the M-tree, starting on the leaf level. For each node N on a given level, a better location for each of the objects in the node N is tried to find. For a ground object O in a leaf N, a set of relevant leafs is retrieved, similarly like the point query does it by the multi-way insertion. For a routing object O in a node N, a set of relevant nodes (on the appropriate level) is retrieved. This is achieved by a modified range query, where the query radius is $r(O)$ and only such nodes are processed the routing objects of which *entirely* contain $rout(O)$. From the relevant retrieved nodes a node is chosen the parent routing object $rout(O_i)$ of which is closest to the object O. If the object O is closer to $rout(O_i)$ more than to the routing object of N (i.e. $d(O, rout(O_i)) < d(O, rout(N))$, the object O is moved from N to the new node. If O was the most distant object in N, the covering radius of its routing object $rout(N)$ is decreased. Processing of a given level is repeated as long as any object movements are occuring. When a level is finished the algorithm for the next higher level starts.

The slim-down algorithm reduces the fat-factor of the M-tree via decreasing the covering radii of routing objects. The number of nodes on each M-tree level is preserved since only redistribution of objects on the same level is performed during the algorithm and no node overflows or underflows (and thus node splitting or merging) by the object movements are allowed.

Example (generalized slim-down algorithm):
Figure 4 shows an M-tree before and after the slim-down algorithm application.

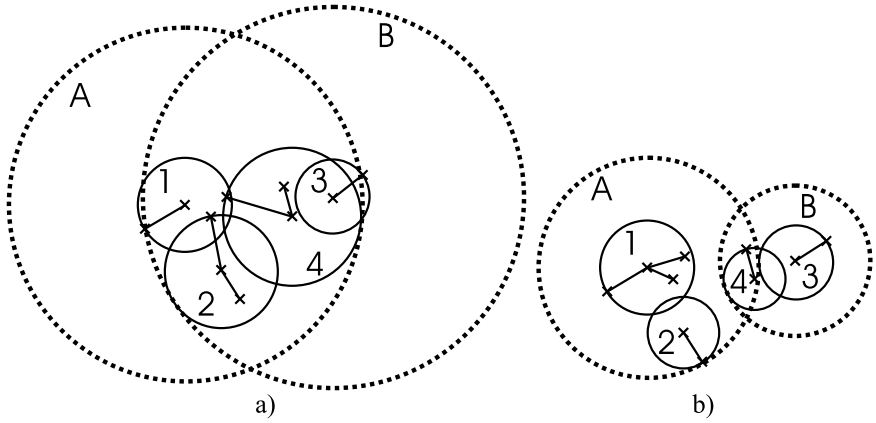


Fig. 4. a) M-tree before slimming down. b) M-tree after slimming down.

Routing objects stored in the root of the M-tree are denoted as A, B while the routing objects stored in the nodes of first level are denoted as 1, 2, 3, 4. In the leafs are stored the ground objects (denoted as crosses). Before slimming down, the subtree of A contains 1 and 4 while the subtree of B contains 3 and 2. After slimming down the leaf level, one object was moved from 2 to 1 and one object was moved from 4 to 1. Covering radii of 2 and 4 were decreased. After slimming down the first level, 4 was moved from A to B, and 2 was moved from B to A. Covering radii of A and B were decreased.

4 Experimental Results

We have completely reimplemented the M-tree in C++, i.e. we have not used the original GiST implementation (our implementation is stable and about 15-times faster than the original one). The experiments ran on an Intel Pentium®4 2.5GHz, 512MB DDR333, under Windows XP.

The experiments were performed on synthetic vector datasets of clustered multidimensional tuples. The datasets were of variable dimensionality, from 2 to 50. The size of dataset was increasing with the dimensionality, from 20,000 2D tuples to 1 million 50D tuples. The integer coordinates of the tuples were ranged from 0 to 1,000,000.

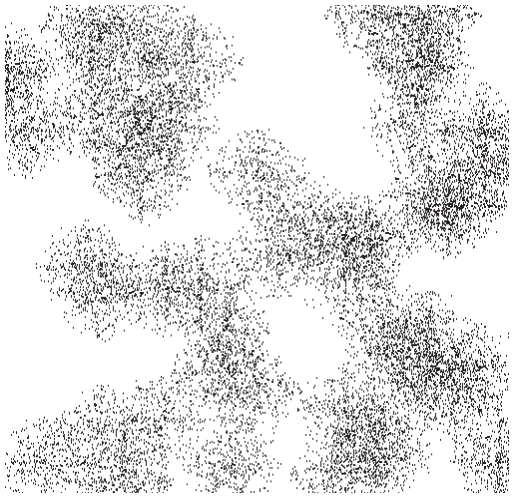


Fig. 5. Two-dimensional dataset distribution.

The data were randomly distributed inside hyper-spherical (L_2) clusters (the number of clusters was increasing with the increasing dimensionality – 50 to 1,000 clusters) with radii increasing from 100,000 (10% of the domain extent)

for 2D tuples to 800,000 (80% of the domain extent) for 50D tuples. In such distributed datasets, the hyper-spherical clusters were highly overlapping due to their quantity and large radii. For the 2D dataset distribution, see Figure 5.

4.1 Building the M-tree

The datasets were indexed in five ways. The single-way insertion method and the bulk loading algorithm (in the graphs denoted as SingleWay and Bulk Loading) represent the original methods of the M-tree construction. In addition to these methods, the multi-way insertion method (denoted as MultiWay) and the generalized slim-down algorithm represent the new building techniques introduced in this article. The slim-down algorithm, as a post-processing technique, was applied on both SingleWay and MultiWay indexes which resulted into indexes denoted as SingleWay+SlimDown and MultiWay+SlimDown. Some general M-tree statistics are presented in Table 1.

Table 1. M-tree statistics.

Metric: L_2 (euclidean)	Node capacity: 20	Dimensionality: 2 – 50
Tuples: 20,000 – 1,000,000	Tree height: 3 – 5	Index size: 1 – 400 MB

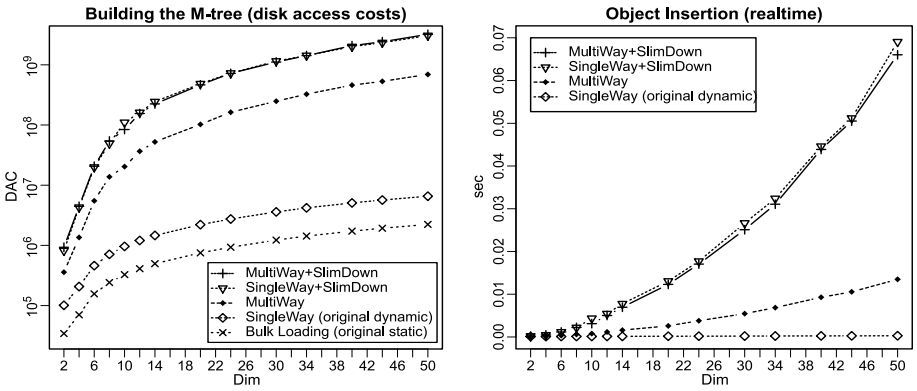


Fig. 6. Building the M-tree: a) Disk access costs. b) Realtime costs per one object.

The first experiment shows the M-tree building costs. In Figure 6a, the disk access costs are presented. We can see that the SingleWay and Bulk Loading indexes were built much cheaply than the other ones, but the construction costs were not the primary objective of our approach. Figure 6b illustrates the average realtime costs per one inserted object. In Figure 7a, the fat-factor characteristics of the indexes are depicted. The fat-factor of SingleWay+SlimDown and MultiWay+SlimDown indexes is very low, which indicates that these indexes contain relatively few overlapping regions. An interesting fact can be observed from the Figure 7b showing the average node utilization.

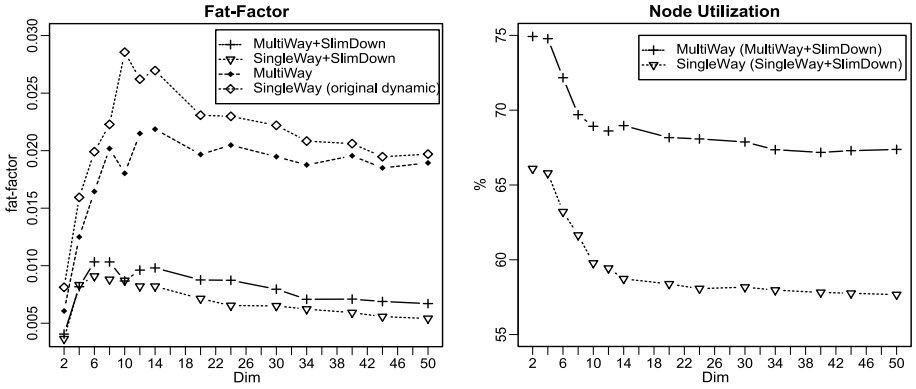


Fig. 7. Building the M-tree: a) Fat-factor. b) Node utilization.

The MultiWay index utilization is by more than 10% better than the utilization of the SingleWay index. Studying this value is not relevant for the SingleWay+SlimDown and MultiWay+SlimDown indexes since the "slimming-down" does not change the average node utilization, thus the results are the same as those achieved for SingleWay and MultiWay.

4.2 Range Queries

The objective of our approach was to increase the querying performace of the M-tree. For the query experiments, sets of query objects were randomly selected from the datasets. Each query test consisted from 100 to 750 queries (according to the dimensionality and dataset size). The results were averaged.

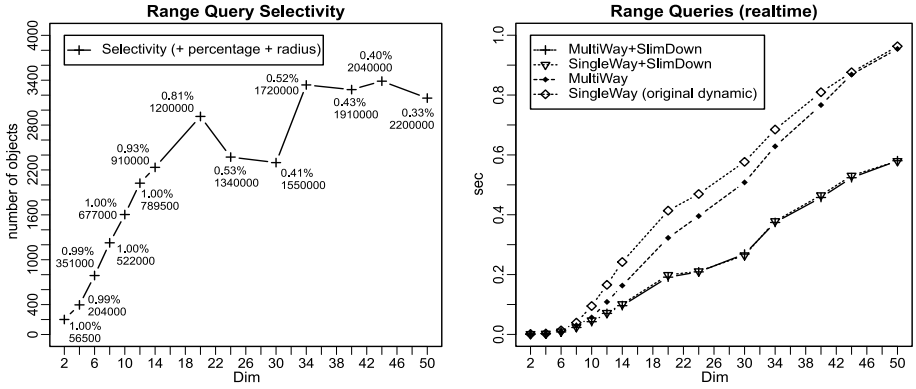


Fig. 8. Range queries: a) Range query selectivity. b) Range query realtimes.

In Figure 8a, the average range query selectivity is presented for each dataset. The selectivity was kept under 1% of all the objects in the dataset. For an interest, we also present the average query radii. In Figure 8b, the realtime costs are presented for the range queries. We can see that the query processing of the SingleWay+SlimDown and MultiWay+SlimDown indexes is almost twice faster when compared with the SingleWay index.

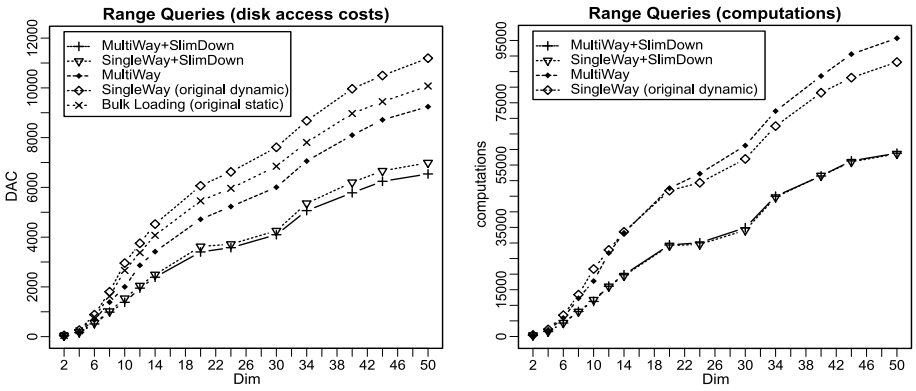


Fig. 9. Range queries: a) Disk access costs. b) Computation costs.

The disk access costs and the computation costs for the range queries are presented in Figure 9. The computation costs comprise the total number of the d function executions.

4.3 k-NN Queries

The performace gain is even more noticeable by the k-NN queries processing. In Figure 10a, the disk access costs are presented for 10-NN queries.

As the results show, querying the SingleWay+SlimDown index consumes 3.5-times less disk accesses than querying the SingleWay index. Similar behaviour can be observed also for the computation costs presented in Figure 10b. The most promising results are presented in Figure 11 where the 100-NN queries were tested. The querying performance of the SingleWay+SlimDown index is here better by more than 300% than the performance of the SingleWay index.

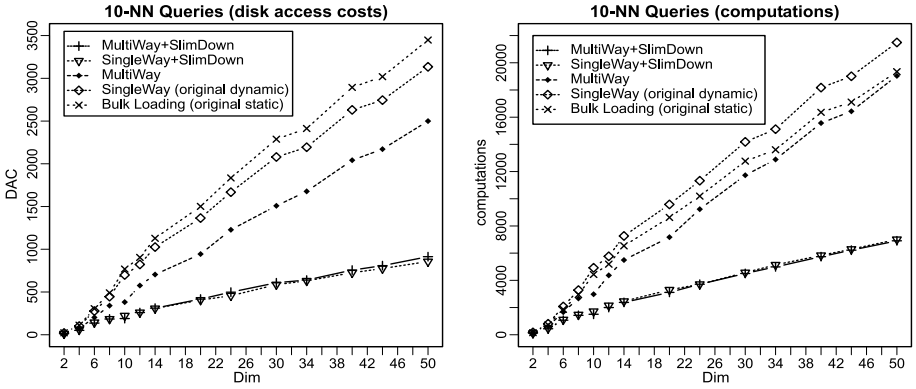


Fig. 10. 10-NN queries: a) Disk access costs. b) Computation costs.

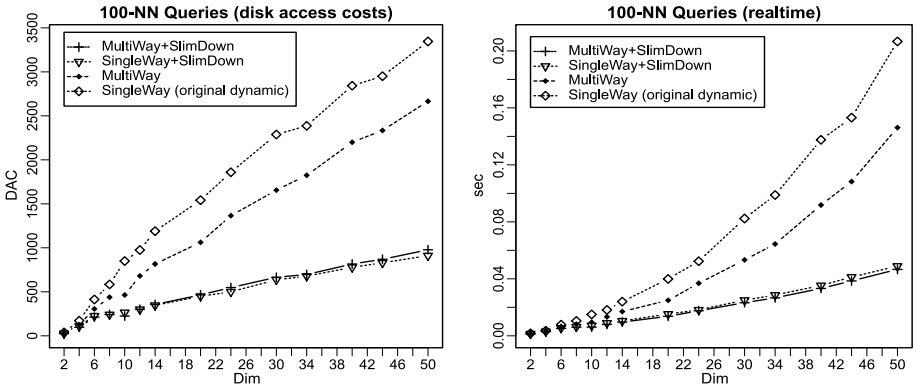


Fig. 11. 100-NN queries: a) Disk access costs. b) Realtime costs.

5 Conclusions

In this paper we have introduced two dynamic techniques of building the M-tree. The cheaper multi-way insertion causes superior node utilization and thus smaller indexes, while the querying performance for the k-NN queries is improved by up to 50%. The more expensive generalized slim-down algorithm causes superior querying performance for both the range and the k-NN queries, for the 100-NN queries even by more than 300%.

Since the M-tree construction costs used by the multi-way insertion and mainly by the generalized slim-down algorithm are considerable, the methods proposed in this paper are suited for DBMS scenarios where relatively few insertions to a database are requested and, on the other hand, many similarity queries must be quickly answered at a moment.

From the DBMS point of view, the static bulk loading algorithm can be considered as a transaction, hence the database is not usable during the bulk loading algorithm run. However, the slim-down algorithm, as a dynamic post-

processing method, is not a transaction. Moreover, it can operate continuously in a processor idle time and it can be, whenever, interrupted without any problem. Thus the construction costs can be spread over the time.

References

1. R. Bayer. The Universal B-Tree for multidimensional indexing: General Concepts. In *Proceedings of World-Wide Computing and its Applications'97, WWCA'97, Tsukuba, Japan*, 1997.
2. J. Bentley. Multidimensional Binary Search Trees Used for Associative Searching. *Communication of the ACM*, 18(9):508–517, 1975.
3. S. Berchtold, D. Keim, and H.-P. Kriegel. The X-tree: An Index Structure for High-Dimensional Data. In *Proceedings of the 22nd Intern. Conf. on VLDB, Mumbai (Bombay), India*, pages 28–39. Morgan Kaufmann, 1996.
4. C. Böhm, S. Berchtold, and D. Keim. Searching in High-Dimensional Spaces – Index Structures for Improving the Performance of Multimedia Databases. *ACM Computing Surveys*, 33(3):322–373, 2001.
5. T. Bozkaya and Z. M. Ozsoyoglu. Indexing large metric spaces for similarity search queries. *ACM Transactions on Database Systems*, 24(3):361–404, 1999.
6. P. Ciaccia and M. Patella. Bulk loading the M-tree. In *Proceedings of the 9th Australian Conference (ADC'98)*, pages 15–26, 1998.
7. P. Ciaccia, M. Patella, and P. Zezula. M-tree: An Efficient Access Method for Similarity Search in Metric Spaces. In *Proceedings of the 23rd Athens Intern. Conf. on VLDB*, pages 426–435. Morgan Kaufmann, 1997.
8. A. Guttman. R-Trees: A Dynamic Index Structure for Spatial Searching. In *Proceedings of ACM SIGMOD 1984, Annual Meeting, Boston, USA*, pages 47–57. ACM Press, June 1984.
9. E. Navarro, R. Baeza-Yates, and J. Marroquin. Searching in Metric Spaces. *ACM Computing Surveys*, 33(3):273–321, 2001.
10. M. Patella. *Similarity Search in Multimedia Databases*. Dipartmento di Elettronica Informatica e Sistemistica, Bologna, 1999.
11. H. Samet. The Quadtree and Related Hierarchical Data Structures. *ACM Computing Surveys*, 16(3):184–260, 1984.
12. H. Samet. *Spatial data structures in Modern Database Systems: The Object Model, Interoperability, and Beyond*, pages 361–385. Addison-Wesley/ACM Press, 1995.
13. C. Traina Jr., A. Traina, B. Seeger, and C. Faloutsos. Slim-Trees: High performance metric trees minimizing overlap between nodes. *Lecture Notes in Computer Science*, 1777, 2000.
14. J. Uhlmann. Satisfying general proximity/similarity queries with metric trees. *Information Processing Letters*, 40(4):175–179, 1991.
15. P. N. Yanilos. Data Structures and Algorithms for Nearest Neighbor Search in General Metric Spaces. In *Proceedings of Fourth Annual ACM/SIGACT-SIAM Symposium on Discrete Algorithms - SODA*, pages 311–321, 1993.
16. C. Yu. *High-Dimensional Indexing*. Springer–Verlag, LNCS 2341, 2002.

Chapter 3

PM-tree: Pivoting metric tree for similarity search in multimedia databases

Tomáš Skopal
Jaroslav Pokorný
Václav Snášel

PM-tree: Pivoting Metric Tree for Similarity Search in Multimedia Databases [47]

Regular paper at the 8th East European Conference on Advances in Databases and Information Systems (ADBIS 2004), Budapest, Hungary, September 2004

Published in the proceedings of ADBIS 2004, pages 99–114, *Computer and Automation Reasearch Institute (CARI) of the Hungarian Academy of Sciences*, ISBN 963-311-358-X

MTA SZTAKI
COMPUTER AND AUTOMATION RESEARCH INSTITUTE
HUNGARIAN ACADEMY OF SCIENCES

PM-tree: Pivoting Metric Tree for Similarity Search in Multimedia Databases

Tomáš Skopal[1], Jaroslav Pokorný[2], and Václav Snášel[1]

[1] Department of Computer Science, VŠB–Technical University of Ostrava, Czech Republic {tomas.skopal, vaclav.snasel}@vsb.cz
[2] Department of Software Engineering, Charles University in Prague, Czech Republic pokorny@ksi.ms.mff.cuni.cz

Abstract. In this paper we introduce the Pivoting M-tree (PM-tree), a metric access method combining M-tree with the pivot-based approach. While in M-tree a metric region is represented by a hyper-sphere, in PM-tree the shape of a metric region is determined by intersection of the hyper-sphere and a set of hyper-rings. The set of hyper-rings for each metric region is related to a fixed set of pivot objects. As a consequence, the shape of a metric region bounds the indexed objects more tightly which, in turn, significantly improves the overall efficiency of similarity search. We present basic algorithms on PM-tree and two cost models for range query processing. Finally, the PM-tree efficiency is experimentally evaluated on large synthetic as well as real-world datasets.

Keywords: PM-tree, M-tree, pivot-based methods, efficient similarity search

1 Introduction

The volume of various multimedia collections worldwide rapidly increases and the need for an efficient content-based similarity search in large multimedia databases becomes stronger. Since a multimedia document is modelled by an object (usually a vector) in a feature space $\mathcal{U}$, the whole collection of documents (the multimedia database) can be represented as a dataset $S \subset \mathcal{U}$. A similarity function is often modelled using a *metric*, i.e. a distance function d satisfying reflexivity, positivity, symmetry, and triangular inequality.

Given a metric space $\mathcal{M} = (\mathcal{U}, d)$, the *metric access methods* (MAMs) [4] organize (or index) objects in dataset S just using the metric d. The MAMs try to recognize a metric structure hidden in S and exploit it for an efficient search. Common to all MAMs is that during search process the triangular inequality of d allows to discard some irrelevant subparts of the metric structure.

2 M-tree

Among many of metric access methods developed so far, the M-tree [5, 8] (and its modifications Slim-tree [11], M$^+$-tree [14]) is still the only indexing technique suitable for an efficient similarity search in large multimedia databases.

The M-tree is based on a hierarchical organization of data objects $O_i \in S$ according to a given metric d. Like other dynamic and paged trees, the M-tree structure consists of a balanced hierarchy of nodes. The nodes have a fixed capacity and a utilization threshold. Within M-tree hierarchy the objects are clustered into *metric regions*. The leaf nodes contain *ground entries* of the indexed data objects while *routing entries* (stored in the inner nodes) describe the metric regions. A ground entry looks like:

$$grnd(O_i) = [O_i, oid(O_i), d(O_i, \mathrm{Par}(O_i))]$$

where $O_i \in S$ is an indexed data object, $oid(O_i)$ is an identifier of the original DB object (stored externally), and $d(O_i, \mathrm{Par}(O_i))$ is a precomputed distance between O_i and the data object of its parent routing entry. A routing entry looks like:

$$rout(O_i) = [O_i, ptr(T(O_i)), r_{O_i}, d(O_i, \mathrm{Par}(O_i))]$$

where $O_i \in S$ is a data object, $ptr(T(O_i))$ is pointer to the covering subtree, and r_{O_i} is the covering radius. The routing entry determines a hyper-spherical metric region in $\mathcal{M}$ where the object O_i is a center of that region and r_{O_i} is a radius bounding the region. The precomputed value $d(O_i, \mathrm{Par}(O_i))$ is used for optimizing most of the M-tree algorithms. In Figure 1 a metric region and

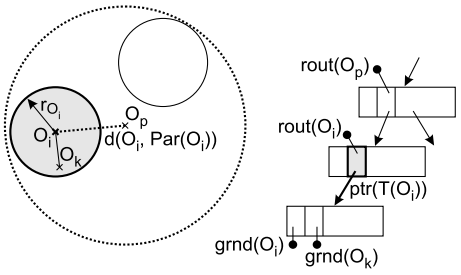


Fig. 1. A routing entry and its metric region in the M-tree structure

its appropriate routing entry $rout(O_i)$ in an inner node are presented. For a hierarchy of metric regions (routing entries $rout(O_i)$ respectively) the following condition must be satisfied:

All data objects stored in leaves of covering subtree $T(O_i)$ of $rout(O_i)$ must be spatially located inside the region defined by $rout(O_i)$.

Formally, having a $rout(O_i)$ then $\forall O_j \in T(O_i), d(O_i, O_j) \leq r_{O_i}$. If we realize, such a condition is very weak since there can be constructed many M-trees of the same object content but of different hierarchy. The most important consequence is that many regions on the same M-tree level may overlap. An example in Figure 2 shows several data objects partitioned among (possibly overlapping) metric regions and the appropriate M-tree.

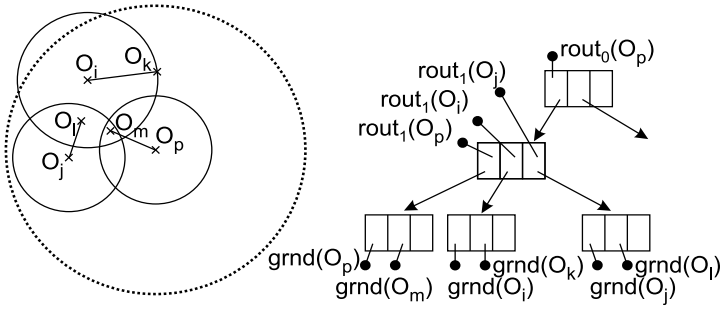


Fig. 2. Hierarchy of metric regions and the appropriate M-tree

2.1 Similarity Queries

The structure of M-tree was designed to natively support similarity queries (proximity queries actually). Given a query object Q, a similarity/proximity query returns objects $O_i \in S$ close to Q.

In the context of similarity search we distinguish two main types of queries. A *range query* $rq(Q, r_Q, S)$ is specified as a hyper-spherical *query region* defined by a query object Q and a query radius r_Q. The purpose of a range query is to return all the objects $O_i \in S$ satisfying $d(Q, O_i) \leq r_Q$. A *k-nearest neighbours query* (k-NN query) $knn(Q, k, S)$ is specified by a query object Q and a number k. A k-NN query returns the first k nearest objects to Q. Technically, a k-NN query can be implemented using a range query with dynamic query radius [8].

During a similarity query processing the M-tree hierarchy is being traversed down. Only if a routing entry $rout(O_i)$ (its metric region respectively) overlaps the query region, the covering subtree $T(O_i)$ of $rout(O_i)$ is relevant to the query and thus further processed.

2.2 Retrieval Efficiency

The retrieval efficiency of an M-tree (i.e. the performance of a query evaluation) is highly dependent on the overall volume[3] of the metric regions described by routing entries. The larger metric region volumes the higher probability of overlap with a query region.

Recently, we have introduced two algorithms [10] leading to reduction of the overall volume of metric regions. The first method, the *multi-way dynamic insertion*, finds the most suitable leaf for each object to be inserted. The second post-processing method, the *generalized slim-down algorithm*, tries to "horizontally" (i.e. separately for each M-tree level) redistribute all entries among more suitable nodes.

[3] We consider only an imaginary volume since there exists no universal notion of volume in general metric spaces. However, without loss of generality, we can say that a hyper-sphere volume grows if its covering radius increases.

3 Pivoting M-tree

Each metric region of M-tree is described by a bounding hyper-sphere (defined by a center object and a covering radius). However, the shape of hyper-spherical region is far from optimal since it does not bound the data objects tightly together thus the region volume is too large. In other words, relatively to the hyper-sphere volume, there is only "few" objects spread inside the hyper-sphere and a huge proportion of an empty space[4] is covered. Consequently, for hyper-spherical regions of large volumes the query processing becomes less efficient.

In this section we introduce an extension of M-tree, called *Pivoting M-tree* (PM-tree), exploiting pivot-based ideas for metric region volume reduction.

3.1 Pivot-based Methods

Similarity search realized by pivot-based methods (e.g. AESA, LAESA) [4, 7] follows a single general idea. A set of p objects $\{P_1, ..., P_t, ..., P_p\} \subset S$ is selected, called *pivots* (or vantage points). The dataset S (of size n) is preprocessed so as to build a table of $n \cdot p$ entries, where all the distances $d(O_i, P_t)$ are stored for every $O_i \in S$ and every pivot P_t. When a range query $rq(Q, r_Q, S)$ is processing, we compute $d(Q, P_t)$ for every pivot P_t and then try to discard such O_i that $|d(O_i, P_t) - d(Q, P_t)| > r_Q$. The objects O_i which cannot be eliminated by this rule have to be directly compared against Q.

The simple sequential pivot-based approach is suitable especially for applications where the distance d is considered expensive to compute. However, it is obvious that the whole table of $n \cdot p$ entries must be sequentially loaded during a query processing which significantly increases the disk access costs. Moreover, each non-discarded object (i.e. an object required to be directly compared) must be processed which means further disk access as well as computation costs.

There were developed also hierarchical pivot-based structures, e.g. the *vp-tree* [13] (vantage point tree) or the *mvp-tree* [2] (multi vp-tree). Unfortunately, these structures are not suitable for similarity search in large multimedia databases since they are static (i.e. they are built in top-down manner while the whole dataset must be available at construction time) and they are not paged (i.e. a secondary memory management is rather complicated for them).

3.2 Structure of PM-tree

Since PM-tree is an extension of M-tree we just describe the new facts instead of a comprehensive definition. To exploit advantages of both, the M-tree and the pivot-based approach, we have enhanced the routing and ground entries by a pivot-based information.

First of all, a set of p pivots $P_t \in S$ must be selected. This set is fixed for all the lifetime of a particular PM-tree index. Furthermore, a routing entry in a

[4] The uselessly indexed empty space is sometimes referred as the "dead space" [1].

PM-tree inner node is defined as:

$$rout_{PM}(O_i) = [O_i, ptr(T(O_i)), r_{O_i}, d(O_i, \mathrm{Par}(O_i)), \mathrm{HR}]$$

The HR attribute is an array of p_{hr} *hyper-rings* ($p_{hr} \leq p$) where the t-th hyper-ring HR[t] is the smallest interval covering distances between the pivot P_t and each of the objects stored in leaves of $T(O_i)$, i.e. HR[t] = $\langle$HR[t].min, HR[t].max$\rangle$ where HR[t].min = $min(\{d(O_j, P_t)\})$ and HR[t].max = $max(\{d(O_j, P_t)\})$, for $\forall O_j \in T(O_i)$. Similarly, for a PM-tree leaf we define a ground entry as:

$$grnd_{PM}(O_i) = [O_i, oid(O_i), d(O_i, \mathrm{Par}(O_i)), \mathrm{PD}]$$

The PD attribute stands for an array of p_{pd} *pivot distances* ($p_{pd} \leq p$) where the t-th distance PD[t] = $d(O_i, P_t)$.

Since each hyper-ring region (P_t, HR[t]) defines a metric region containing *all* the objects stored in $T(O_i)$, an intersection of all the hyper-rings and the hyper-sphere forms a metric region bounding all the objects in $T(O_i)$ as well. Due to the intersection with hyper-sphere the PM-tree metric region is always smaller than the original M-tree region defined just by a hyper-sphere. For a comparison of an M-tree region and an equivalent PM-tree region see Figure 3. The numbers p_{hr} and p_{pd} (both fixed for a PM-tree index lifetime) allow us to specify the "amount of pivoting". Obviously, using a suitable $p_{hr} > 0$ and $p_{pd} > 0$ the PM-tree can be tuned to achieve an optimal performance (see Section 5).

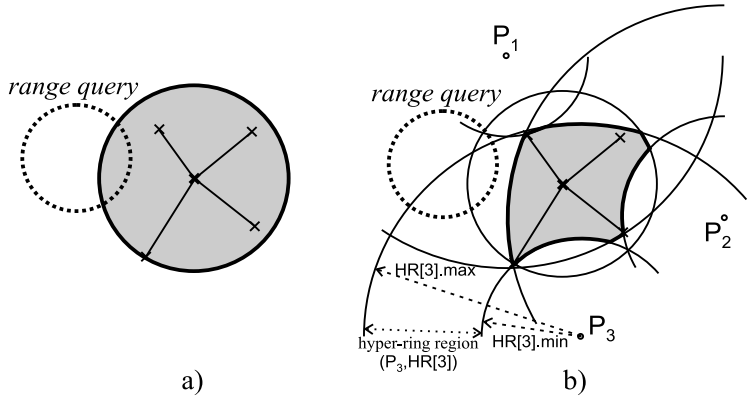


Fig. 3. (a) Region of M-tree (b) Reduced region of PM-tree (using three pivots)

3.3 Building the PM-tree

In order to keep HR and PD arrays up-to-date, the original M-tree construction algorithms [8, 10] must be adjusted. We have to mention the adjusted algorithms still preserve the logarithmic time complexity.

Object Insertion. After a data object O_i is inserted into a leaf the HR arrays of all routing entries in the insertion path must be updated by values $d(O_i, P_t)$, $\forall t \leq p_{hr}$. For the leaf node in insertion path the PD array of the new ground entry must be updated by values $d(O_i, P_t), \forall t \leq p_{pd}$.

Node Splitting. After a node is split a new HR array of the left new routing entry is created by merging all appropriate intervals HR[t] (or computing HR in case of a leaf split) stored in routing entries (ground entries respectively) of the left new node. A new HR array of the right new routing entry is created similarly.

3.4 Query Processing

Before processing a similarity query the distances $d(Q, P_t)$, $\forall t \leq max(p_{hr}, p_{pd})$ have to be computed. During a query processing the PM-tree hierarchy is being traversed down. Only if the metric region of a routing entry $rout(O_i)$ is overlapping the query region (Q, r_Q), the covering subtree $T(O_i)$ is relevant to the query and thus further processed. A routing entry is relevant to the query in case that the query region overlaps *all* the hyper-rings stored in HR. Hence, prior to the standard hyper-sphere overlap check (used by M-tree), the overlap of hyper-rings HR[t] against the query region is checked as follows (note that no additional distance computation is needed):

$$\bigwedge_{t=1}^{p_{hr}} d(Q, P_t) - r_Q \leq \mathrm{HR}[t].\mathrm{max} \ \wedge \ d(Q, P_t) + r_Q \geq \mathrm{HR}[t].\mathrm{min}$$

If the above condition is false, the subtree $T(O_i)$ is not relevant to the query and thus can be discarded from further processing. On the leaf level an irrelevant ground entry is determined such that the following condition is not satisfied:

$$\bigwedge_{t=1}^{p_{pd}} |d(Q, P_t) - \mathrm{PD}[t]| \leq r_Q$$

In Figure 3 a range query situation is illustrated. Although the M-tree metric region cannot be discarded (see Figure 3a), the PM-tree region can be safely ignored since the hyper-ring HR[2] is not overlapped (see Figure 3b).

The hyper-ring overlap condition can be integrated into the original M-tree range query as well as k-NN query algorithms. In case of range query the adjustment is straightforward – the hyper-ring overlap condition is combined with the original hyper-sphere overlap condition. However, the optimal M-tree k-NN query algorithm (based on priority queue heuristics) must be redesigned which is a subject of our future research.

3.5 Object-to-pivot Distance Representation

In order to minimize storage volume of HR and PD arrays in PM-tree nodes, a short representation of object-to-pivot distance is required. We can represent

interval HR[t] by two 4-byte reals and a pivot distance PD[t] by one 4-byte real. However, when (a part of) the dataset is known in advance we can approximate the 4-byte representation by a 1-byte code. For this reason a distance distribution histogram for each pivot is created by random sampling of objects from the dataset and comparing them against the pivot. Then a distance interval $\langle d_{min}, d_{max} \rangle$ is computed so that most of the histogram distances fall into the interval, see an example in Figure 4 (the d^+ value is an (estimated) maximum distance of a bounded metric space $\mathcal{M}$).

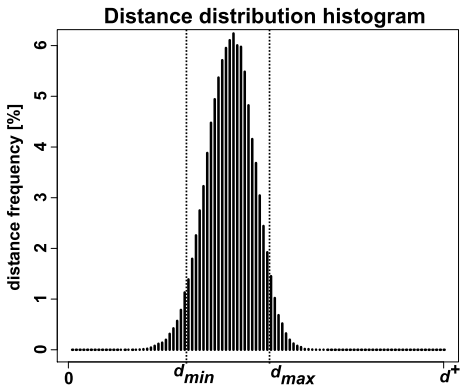


Fig. 4. Distance distribution histogram, 90% distances in interval $\langle d_{min}, d_{max} \rangle$

Distance values in HR and PD are scaled into interval $\langle d_{min}, d_{max} \rangle$ as 1-byte codes. Using 1-byte codes the storage savings are considerable. As an example, for $p_{hr} = 50$ together with using 4-byte distances, the hyper-rings stored in an inner node having capacity 30 entries will consume $30 \cdot 50 \cdot 2 \cdot 4 = 12000$ bytes while by using 1-byte codes the hyper-rings will take $30 \cdot 50 \cdot 2 \cdot 1 = 3000$ bytes.

3.6 Selecting the Pivots

The methods of selecting an optimal set of pivots have been intensively studied [9, 3] while, in general, we can say that a set of pivots is optimal such that distances among pivots are maximal (close pivots give almost the same information) and the pivots are located outside the data clusters.

In the context of PM-tree the optimal set of pivots causes that the M-tree hyper-spherical region is effectively "chopped off" by hyper-rings so that the smallest overall volume of PM-tree regions (considering the volume of intersection of hyper-rings and the hyper-sphere) is obtained.

In experiments presented in Section 5 we have used a cheap but effective method which samples N groups of p pivots from the dataset S at random. The group is selected for which the sum of distances among pivots is maximal.

4 Range Query Cost Models

In this section we present a node-based and a level-based cost models for range query processing in PM-tree, allowing to predict the PM-tree retrieval performance. Since PM-tree is an extension of M-tree, we have extended the original cost models developed for M-tree [6]. Like the M-tree cost models, the PM-tree cost models are conditioned by the following assumptions:

- The only information used is (an estimate of) the *distance distribution* of objects in a given dataset since no information about *data distribution* is known.
- A *biased* query model is considered, i.e. the distribution of query objects is equal to that of data objects.
- The dataset is supposed to have high "homogeneity of viewpoints" (for details we refer to [6, 8]).

The basic tool used in the cost models is a probability estimation that two hyper-spheres overlap, i.e. (using triangular inequality of d)

$$\mathbf{Pr}\{\text{spheres } (O_1, r_{O_1}) \text{ and } (O_2, r_{O_2}) \text{ overlapped}\} = \mathbf{Pr}\{d(O_1, O_2) \leq r_{O_1} + r_{O_2}\}$$

where O_1, O_2 are center objects and r_{O_1}, r_{O_2} are radii of the hyper-spheres. For this purpose the *overall distance distribution* function is used, defined as:

$$F(x) = \mathbf{Pr}\{d(O_i, O_j) \leq x\}, \forall O_i, O_j \in \mathcal{U}$$

and also the *relative distance distribution* function is used, defined as:

$$F_{O_k}(x) = \mathbf{Pr}\{d(O_k, O_i) \leq x\}, O_k \in \mathcal{U}, \forall O_i \in \mathcal{U}$$

For an approximate $F_{(O_k)}$ evaluation a set $\mathcal{O}$ of s objects $\mathcal{O}_i \in S$ is sampled. The F is computed using the $s \times s$ matrix of pairwise distances between objects in $\mathcal{O}$. For the F_{O_k} evaluation only the vector of s distances $d(\mathcal{O}_i, O_k)$ is needed.

4.1 Node-based Cost Model

In the node-based cost model (NB-CM) a probability of access to each PM-tree node is predicted. Basically, a node N is accessed if its metric region (described by the parent routing entry of N) is overlapping the query hyper-sphere (Q, r_Q):

$$\mathbf{Pr}\{\text{node N is accessed}\} = \mathbf{Pr}\{\text{metric region of N is overlapped by } (Q, r_Q)\}$$

Specifically, a PM-tree node N is accessed if its metric region (defined by a hyper-sphere and p_{hr} hyper-rings) overlaps the query hyper-sphere:

$$\mathbf{Pr}\{\text{N is accessed}\} = \mathbf{Pr}\{\text{hyper-sphere is inter.}\} \cdot \prod_{t=1}^{p_{hr}} \mathbf{Pr}\{t\text{-th hyper-ring is inter.}\}$$

and finally (for query radius r_Q and the parent routing entry of N)

$$\mathbf{Pr}\{\text{N accessed}\} \approx F(r_N+r_Q)\cdot\prod_{t=1}^{p_{hr}} F_{P_t}(\text{HR}_N[t].\text{max}+r_Q)\cdot(1-F_{P_t}(r_Q-\text{HR}_N[t].\text{min}))$$

To determine the estimated disk access costs (DAC) for a range query, it is sufficient to sum the above probabilities over all the m nodes in the PM-tree:

$$\mathbf{DAC} = \sum_{i=1}^{m} F(r_{N_i}+r_Q)\cdot\prod_{t=1}^{p_{hr}} F_{P_t}(\text{HR}_{N_i}[t].\text{max}+r_Q)\cdot(1-F_{P_t}(r_Q-\text{HR}_{N_i}[t].\text{min}))$$

The computation costs (CC) are estimated considering the probability that a node is accessed multiplied by the number of its entries, $e(N_i)$, thus obtaining

$$\mathbf{CC} = \sum_{i=1}^{m} e(N_i)\cdot F(r_{N_i}+r_Q)\cdot\prod_{t=1}^{p_{hr}} F_{P_t}(\text{HR}_{N_i}[t].\text{max}+r_Q)\cdot(1-F_{P_t}(r_Q-\text{HR}_{N_i}[t].\text{min}))$$

4.2 Level-based Cost Model

The problem with NB-CM is that maintaining statistics for every node is very time consuming when the PM-tree index is large. To overcome this, we consider a simplified level-based cost model (LB-CM) which uses only average information collected for each level of the PM-tree. For each level l of the tree ($l = 1$ for root level, $l = L$ for leaf level), LB-CM uses this information: m_l (the number of nodes at level l), $\overline{r_l}$ (the average value of covering radius considering all the nodes at level l), $\overline{\text{HR}_l[i].\text{min}}$ and $\overline{\text{HR}_l[i].\text{max}}$ (the average information about hyper-rings considering all the nodes at level l). Given these statistics, the number of nodes accessed by a range query can be estimated as

$$\mathbf{DAC} \approx \sum_{l=1}^{L} m_l\cdot F(\overline{r_l}+r_Q)\cdot\prod_{t=1}^{p_{hr}} F_{P_t}(\overline{\text{HR}_l[t].\text{max}}+r_Q)\cdot(1-F_{P_t}(r_Q-\overline{\text{HR}_l[t].\text{min}}))$$

Similarly, we can estimate computation costs as

$$\mathbf{CC} \approx \sum_{l=1}^{L} m_{l+1}\cdot F(\overline{r_l}+r_Q)\cdot\prod_{t=1}^{p_{hr}} F_{P_t}(\overline{\text{HR}_l[t].\text{max}}+r_Q)\cdot(1-F_{P_t}(r_Q-\overline{\text{HR}_l[t].\text{min}}))$$

where $m_{L+1} \stackrel{\text{def}}{=} n$ is the number of indexed objects.

4.3 Experimental Evaluation

In order to evaluate accuracy of the presented cost models we have made several experiments on a synthetic dataset. The dataset consisted of 10,000 10-dimensional tuples (embedded inside unitary hyper-cube) uniformly distributed among 100 L_2-spherical clusters of diameter $\frac{d^+}{10}$ (where $d^+ = \sqrt{10}$). The labels "`PM-tree(x,y)`" in the graphs below are described in Section 5.

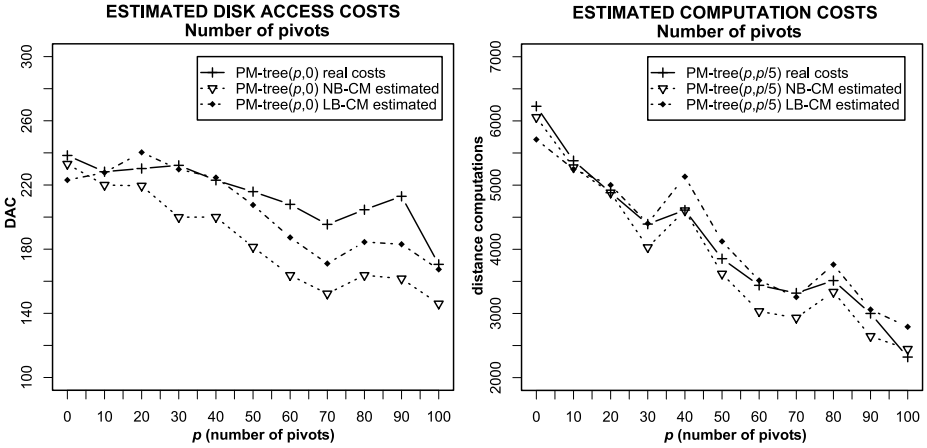


Fig. 5. Number of pivots, query sel. 200 objs. (a) Disk access costs (b) Computation costs

The first set of experiments investigated the accuracy of estimates according to the increasing number of pivots used by the PM-tree. The range query selectivity (the average number of objects in the query result) was set to 200. In Figure 5a the estimated DAC as well as the real DAC are presented. The relative error of NB-CM estimates is below 0.2. Surprisingly, the relative error of LB-CM estimates is smaller than for NB-CM, below 0.15. The estimates of computation costs, presented in Figure 5b, are even more accurate than for the DAC estimates, below 0.05 (for NB-CM) and 0.04 (for LB-CM).

The second set of experiments was focused on the accuracy of estimates according to the increasing query selectivity. The relative error of NB-CM DAC estimates (see Figure 6a) is below 0.1. Again, the relative error of LB-CM estimates is very small, below 0.02. The error of computation costs (see Figure 6b) is below 0.07 (for NB-CM) and 0.05 (for LB-CM).

5 Experimental Results

In order to evaluate the overall PM-tree performance we present some results of experiments made on large synthetic as well as real-world vector datasets. In most of the experiments the retrieval efficiency of range query processing was examined. The query objects were randomly selected from the respective dataset while each particular query test consisted of 1000 range queries of the same query selectivity. The results were averaged. Euclidean (L_2) metric was used. The experiments were aimed to compare PM-tree with M-tree – a comparison with other MAMs was out of scope of this paper.

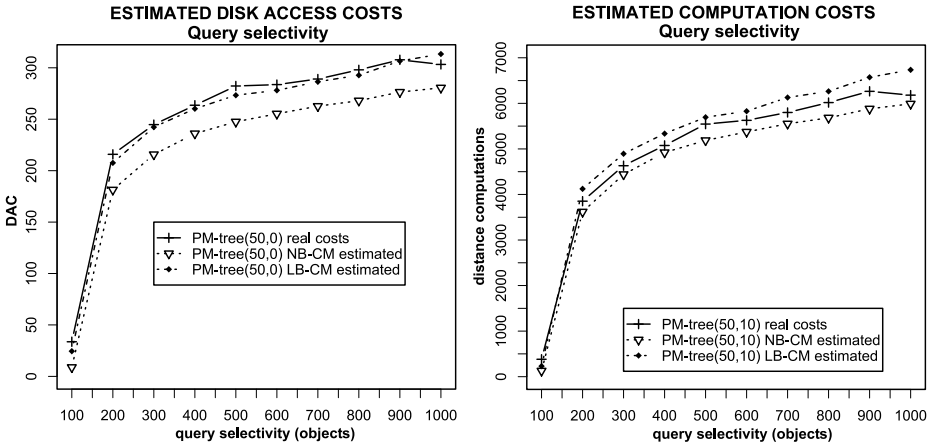


Fig. 6. Query selectivity: (a) Disk access costs (b) Computation costs

Abbreviations in Figures. Each label of form "`PM-tree(x,y)`" stands for a PM-tree index where p_{hr} = `x` and p_{pd} = `y`. A label "<*index*> `+ SlimDown`" denotes an index subsequently post-processed using the slim-down algorithm (for details about the slim-down algorithm we refer to [10]).

5.1 Synthetic Datasets

For the first set of experiments a collection of 8 synthetic vector datasets of increasing dimensionality (from $D = 4$ to $D = 60$) was generated. Each dataset (embedded inside unitary hyper-cube) consisted of 100,000 D-dimensional tuples uniformly distributed within 1000 L_2-spherical uniformly distributed clusters. The diameter of each cluster was $\frac{d^+}{10}$ where $d^+ = \sqrt{D}$. These datasets were indexed by PM-tree (for various p_{hr} and p_{pd}) as well as by M-tree. Some statistics about the created indices are described in Table 1 (for explanation see [10]).

Table 1. PM-tree index statistics (synthetic datasets)

Construction methods: `SingleWay + MinMax (+ SlimDown)`	
Dimensionalities: 4,8,16,20,30,40,50,60	Inner node capacities: 10 – 28
Index file sizes: 4.5 MB – 55 MB	Leaf node capacities: 16 – 36
Pivot file sizes[5]: 2 KB – 17 KB	Avg. node utilization: 66%
Node (disk page) sizes: 1 KB ($D = 4, 8$), 2 KB ($D = 16, 20$), 4 KB ($D \geq 30$)	

[5] Access costs to the pivot files, storing pivots P_t and the scaling intervals for all pivots (see Section 3.5), were not considered because of their negligible sizes.

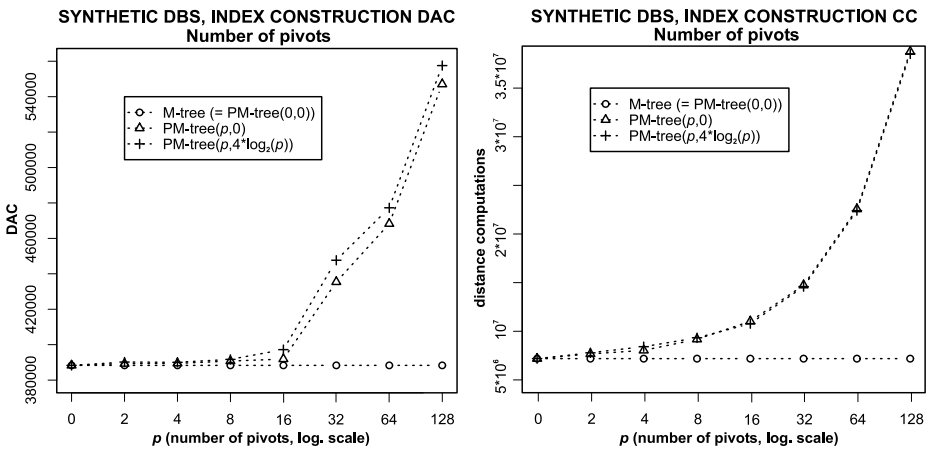


Fig. 7. Construction costs (30D indices): (a) Disk access costs (b) Computation costs

Index construction costs (for 30-dimensional indices) according to the increasing number of pivots are presented in Figure 7. The disk access costs for PM-tree indices with up to 8 pivots are similar to those of M-tree index (see Figure 7a). For `PM-tree`$(128, 0)$ and `PM-tree`$(128, 28)$ indices the DAC are about 1.4 times higher than for the M-tree index. The increasing trend of computation costs (see Figure 7b) depends mainly on the p object-to-pivot distance computations made during each object insertion – additional computations are needed after leaf splitting in order to create HR arrays of the new routing entries.

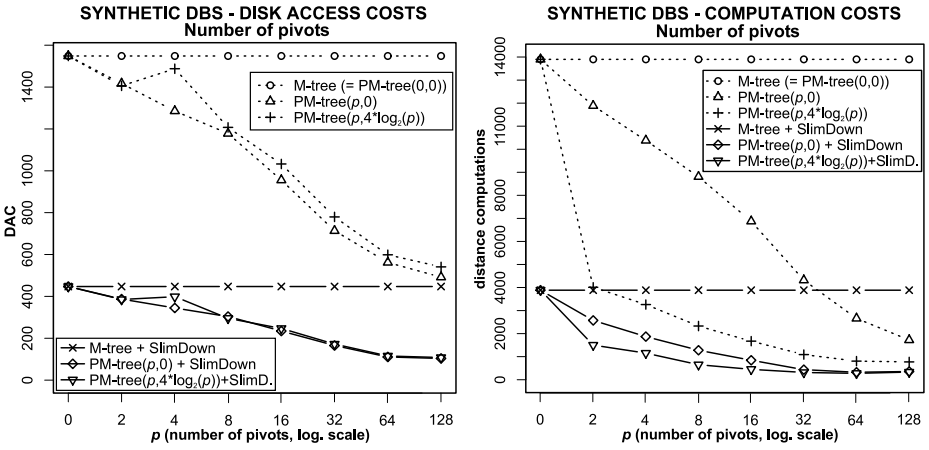


Fig. 8. Number of pivots (30-dim. indices, query selectivity 50 objs.): (a) DAC (b) CC

In Figure 8 the range query costs (for 30-dimensional indices and query selectivity 50 objects) according to the number of pivots are presented. The DAC rapidly decrease with the increasing number of pivots. The `PM-tree(128, 0)` and `PM-tree(128, 28)` indices need only 27% of DAC spent by the M-tree index. Moreover, the PM-tree is superior even after the slim-down algorithm post-processing, e.g. the "slimmed" `PM-tree(128, 0)` index needs only 23% of DAC spent by the "slimmed" M-tree index (and only 6.7% of DAC spent by the ordinary M-tree). The decreasing trend of computation costs is even more steep than for DAC, the `PM-tree(128, 28)` index needs only 5.5% of the M-tree CC.

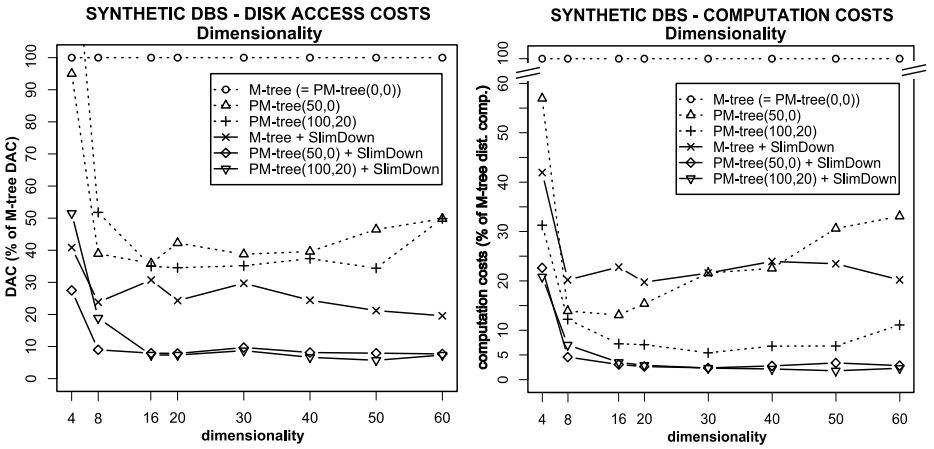


Fig. 9. Dimensionality (query selectivity 50 objects): (a) Disk access costs (b) Computation costs

The influence of increasing dimensionality D is depicted in Figure 9. Since the disk page sizes for different indices vary, the DAC as well as the CC are related (in percent) to the DAC (CC resp.) of M-tree indices. For $8 \leq D \leq 40$ the DAC stay approximately fixed, for $D > 40$ the DAC slightly increase.

5.2 Image Database

For the second set of experiments a collection of about 10,000 web-crawled images [12] was used. Each image was converted into a 256-level gray scale and a frequency histogram was extracted. For indexing the histograms (256-dimensional vectors actually) were used together with the euclidean metric. The statistics about image indices are described in Table 2.

Table 2. PM-tree index statistics (image database)

Construction methods: `SingleWay + MinMax (+ SlimDown)`	
Dimensionality: 256	Inner node capacities: 10 – 31
Index file sizes: 16 MB – 20 MB	Leaf node capacities: 29 – 31
Pivot file sizes: 4 KB – 1 MB	Avg. node utilization: 67%
Node (disk page) size: 32 KB	

In Figure 10a the DAC for increasing number of pivots are presented. We can see that e.g. the "slimmed" `PM-tree(1024,50)` index consumes only 42% of DAC spent by the "slimmed" M-tree index. The computation costs (see Figure 10b) for $p \leq 64$ decrease (down to 36% of M-tree CC). However, for $p > 64$ the overall computation costs grow since the number of necessarily computed query-to-pivot distances (i.e. p distance computations for each query) is proportionally too large. Nevertheless, this fact is dependent on the database size – obviously, for 100,000 objects (images) the proportion of p query-to-pivot distance computations would be smaller when compared with the overall computation costs.

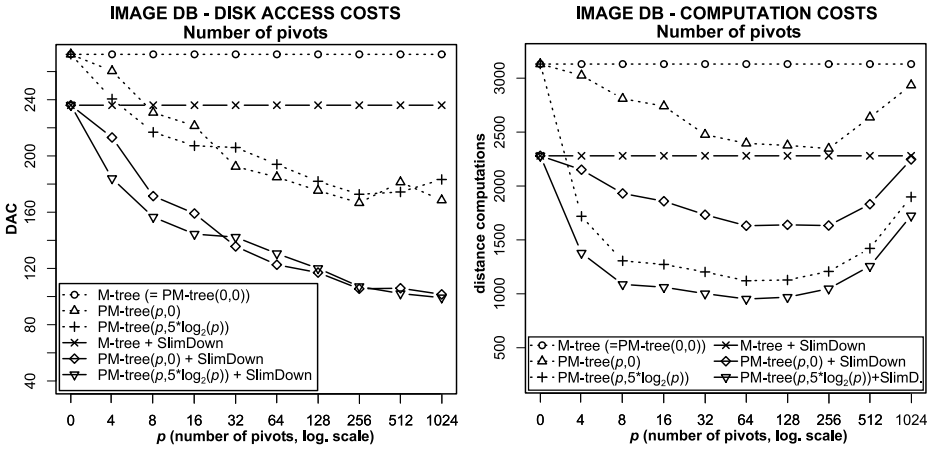


Fig. 10. Number of pivots (query selectivity 50 objects): (a) DAC (b) CC

Finally, the costs according to the increasing range query selectivity are presented in Figure 11. The disk access costs stay below 73% of M-tree DAC (below 58% in case of "slimmed" indices) while the computation costs stay below 43% (49% respectively).

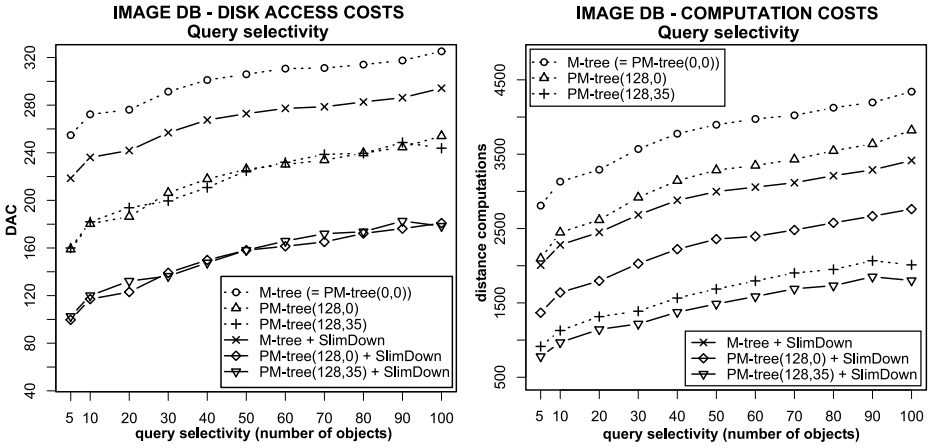


Fig. 11. Query selectivity: (a) Disk access costs (b) Computation costs

5.3 Summary

The experiments on synthetic datasets and mainly on the real dataset have demonstrated the general benefits of PM-tree. The index construction (object insertion respectively) is dynamic and still preserves the logarithmic time complexity. For suitably high p_{hr} and p_{pd} the index size growth is minor. This is true especially for high-dimensional datasets (e.g. the 256-dimensional image dataset) where the size of pivoting information stored in ground/routing entries is negligible when compared with the size of the data object (i.e. vector) itself. A particular (but not serious) limitation of the PM-tree is that a part of the dataset must be known in advance (for the choice of pivots and when used object-to-pivot distance distribution histograms).

Furthermore, the PM-tree could serve as a constructionally much cheaper alternative to the slim-down algorithm on M-tree – the above presented experimental results have shown that the retrieval performance of PM-tree (with sufficiently high p_{hr}, p_{pd}) is comparable or even better than an equivalent "slimmed" M-tree. Finally, a combination of PM-tree and the slim-down algorithm makes the PM-tree a very efficient metric access method.

6 Conclusions and Outlook

In this paper the Pivoting M-tree (PM-tree) was introduced. The PM-tree combines M-tree hierarchy of metric regions with the idea of pivot-based methods. The result is a flexible metric access method providing even more efficient similarity search than the M-tree. Two cost models for range query processing were proposed and evaluated. Experimental results on synthetic as well as real-world datasets have shown that PM-tree is more efficient when compared with the M-tree.

In the future we plan to develop new PM-tree construction algorithms exploiting the pivot-based information. Second, an optimal PM-tree k-NN query algorithm has to be designed and a cost model formulated. Finally, we would like to modify several spatial access methods by utilizing the pivoting information, in particular the R-tree family.

This research has been partially supported by grant Nr. GAČR 201/00/1031 of the Grant Agency of the Czech Republic.

References

1. C. Böhm, S. Berchtold, and D. Keim. Searching in High-Dimensional Spaces – Index Structures for Improving the Performance of Multimedia Databases. *ACM Computing Surveys*, 33(3):322–373, 2001.
2. T. Bozkaya and M. Özgsoyoglu. Distance-based indexing for high-dimensional metric spaces. In *Proceedings of the 1997 ACM SIGMOD International Conference on Management of Data, Tuscon, AZ*, pages 357–368, 1997.
3. B. Bustos, G. Navarro, and E. Chávez. Pivot selection techniques for proximity searching in metric spaces. *Pattern Recognition Letters*, 24(14):2357–2366, 2003.
4. E. Chávez, G. Navarro, R. Baeza-Yates, and J. Marroquín. Searching in Metric Spaces. *ACM Computing Surveys*, 33(3):273–321, 2001.
5. P. Ciaccia, M. Patella, and P. Zezula. M-tree: An Efficient Access Method for Similarity Search in Metric Spaces. In *Proceedings of the 23rd Athens Intern. Conf. on VLDB*, pages 426–435. Morgan Kaufmann, 1997.
6. P. Ciaccia, M. Patella, and P. Zezula. A cost model for similarity queries in metric spaces. In *Proceedings of the Seventeenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 1-3, 1998, Seattle, Washington*, pages 59–68. ACM Press, 1998.
7. L. Mico, J. Oncina, and E. Vidal. A new version of the nearest-neighbor approximating and eliminating search (aesa) with linear preprocessing-time and memory requirements. *Pattern Recognition Letters*, 15:9–17, 1994.
8. M. Patella. *Similarity Search in Multimedia Databases*. PhD thesis, Dipartmento di Elettronica Informatica e Sistemistica, Bologna, 1999.
9. M. Shapiro. The choice of reference points in best-match file searching. *Communications of the ACM*, 20(5):339–343, 1977.
10. T. Skopal, J. Pokorný, M. Krátký, and V. Snášel. Revisiting M-tree Building Principles. In *ADBIS 2003, LNCS 2798, Springer-Verlag, Dresden, Germany*, 2003.
11. C. Traina Jr., A. Traina, B. Seeger, and C. Faloutsos. Slim-Trees: High performance metric trees minimizing overlap between nodes. *Lecture Notes in Computer Science*, 1777, 2000.
12. WBIIS project: Wavelet-based Image Indexing and Searching, Stanford University, http://wang.ist.psu.edu/.
13. P. N. Yanilos. Data Structures and Algorithms for Nearest Neighbor Search in General Metric Spaces. In *Proceedings of Fourth Annual ACM/SIGACT-SIAM Symposium on Discrete Algorithms - SODA*, pages 311–321, 1993.
14. X. Zhou, G. Wang, J. Y. Xu, and G. Yu. M+-tree: A New Dynamical Multidimensional Index for Metric Spaces. In *Proceedings of the Fourteenth Australasian Database Conference - ADC'03, Adelaide, Australia*, 2003.

Chapter 4

Nearest neighbours search using the PM-tree

Tomáš Skopal
Jaroslav Pokorný
Václav Snášel

Nearest Neighbours Search Using the PM-Tree [48]

Regular paper at the 10th International Conference on Database Systems for Advanced Applications (DASFAA 2005), Beijing, China, April 2005

Published in the *Lecture Notes in Computer Science* (LNCS), vol. 3453, pages 803–815, *Springer-Verlag*, ISSN 0302-9743, ISBN 978-3-540-25334-1

Nearest Neighbours Search using the PM-tree

Tomáš Skopal[1], Jaroslav Pokorný[1], and Václav Snášel[2]

[1] Charles University in Prague, FMP, Department of Software Engineering
Malostranské nám. 25, 118 00 Prague, Czech Republic, EU
tomas@skopal.net, jaroslav.pokorny@mff.cuni.cz
[2] VŠB–Technical University of Ostrava, FECS, Dept. of Computer Science
tř. 17. listopadu 15, 708 33 Ostrava, Czech Republic, EU
vaclav.snasel@vsb.cz

Abstract. We introduce a method of searching the k nearest neighbours (k-NN) using PM-tree. The PM-tree is a metric access method for similarity search in large multimedia databases. As an extension of M-tree, the structure of PM-tree exploits local dynamic pivots (like M-tree does it) as well as global static pivots (used by LAESA-like methods). While in M-tree a metric region is represented by a hyper-sphere, in PM-tree the "volume" of metric region is further reduced by a set of hyper-rings. As a consequence, the shape of PM-tree's metric region bounds the indexed objects more tightly which, in turn, improves the overall search efficiency. Besides the description of PM-tree, we propose an optimal k-NN search algorithm. Finally, the efficiency of k-NN search is experimentally evaluated on large synthetic as well as real-world datasets.

1 Introduction

The volume of multimedia databases rapidly increases and the need for efficient content-based search in large multimedia databases becomes stronger. In particular, there is a need for searching for the k most similar documents (called the k nearest neighbours – k-NN) to a given query document.

Since multimedia documents are modelled by objects (usually vectors) in a feature space $\mathbb{U}$, the multimedia database can be represented by a dataset $\mathbb{S} \subset \mathbb{U}$, where $n = |\mathbb{S}|$ is size of the dataset. The search in $\mathbb{S}$ is accomplished by an access method, which retrieves objects relevant to a given similarity query. The similarity measure is often modelled by a *metric*, i.e. a distance d satisfying properties of reflexivity, positivity, symmetry, and triangular inequality. Given a metric space $\mathcal{M} = (\mathbb{U}, d)$, the *metric access methods* (MAMs) [4] organize objects in $\mathbb{S}$ such that a structure in $\mathbb{S}$ is recognized (i.e. a kind of *metric index* is constructed) and exploited for efficient (i.e. quick) search in $\mathbb{S}$. To keep the search as efficient as possible, the MAMs should minimize the *computation costs* (CC) and the *I/O costs*. The computation costs represent the number of (computationally expensive) distance computations spent by the query evaluation. The I/O costs are related to the volume of data needed to be transfered from secondary memory (also referred to as the disk access costs).

In this paper we propose a method of k-NN searching using PM-tree, which is a metric access method for similarity search in large multimedia databases.

2 M-tree

Among the MAMs developed so far, the M-tree [5, 7] (and its modifications) is still the only dynamic MAM suitable for efficient similarity search in large multimedia databases. Like other dynamic and paged trees, the M-tree is a balanced hierarchy of nodes. Given a metric d, the data objects $O_i \in \mathbb{S}$ are organized in a hierarchy of nested clusters, called *metric regions*. The leaf nodes contain *ground entries* of the indexed data objects, while the *routing entries* (stored in the inner nodes) describe the metric regions. A ground entry is denoted as:

$$grnd(O_i) = [O_i, oid(O_i), d(O_i, \mathrm{Par}(O_i))]$$

where $O_i \in \mathbb{S}$ is the data object, $oid(O_i)$ is identifier of the original DB object (stored externally), and $d(O_i, \mathrm{Par}(O_i))$ is precomputed distance between O_i and the data object of its parent routing entry. A routing entry is denoted as:

$$rout(O_i) = [O_i, ptr(T(O_i)), r_{O_i}, d(O_i, \mathrm{Par}(O_i))]$$

where $O_i \in \mathbb{S}$ is a *routing object* (local pivot), $ptr(T(O_i))$ is pointer to the covering subtree, and r_{O_i} is the covering radius. The routing entry determines a hyper-spherical metric region (O_i, r_{O_i}) in $\mathcal{M}$, for which routing object O_i is the center and r_{O_i} is the radius bounding the region. In Figure 1 see several data objects partitioned among (possibly overlapping) metric regions of M-tree.

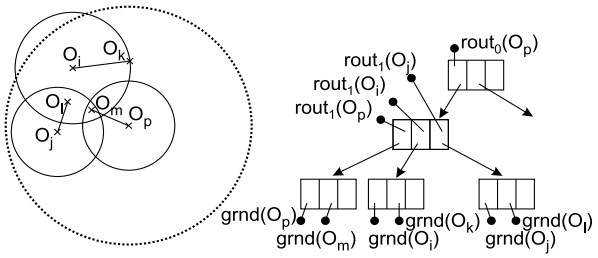


Fig. 1. Hierarchy of metric regions and the appropriate M-tree.

2.1 Similarity Queries in M-tree

The structure of M-tree was designed to support similarity queries (proximity queries actually). We distinguish two basic kinds of queries. The *range query* is specified as a hyper-spherical *query region* (Q, r_Q), defined by a query object Q and a covering query radius r_Q. The purpose of range query is to select all objects $O_i \in \mathbb{S}$ satisfying $d(Q, O_i) \leq r_Q$ (i.e. located inside the query region). The *k nearest neighbours query* (k-NN query) is specified by a query object Q and a number k. A k-NN query selects the first k nearest (most similar) objects to Q. Technically, the k-NN query can be formulated as a range query $(Q, d(Q, O_k))$, where O_k is the k-th nearest neighbour. During query processing, the M-tree hierarchy is traversed down. Given a routing entry $rout(O_i)$, the subtree $T(O_i)$ is processed only if the region defined by $rout(O_i)$ overlaps the query region.

Range Search. The range query algorithm [5, 7] has to follow all M-tree paths leading to data objects O_j inside the query region, i.e. satisfying $d(Q, O_j) \leq r_Q$. In fact, the range query algorithm recursively accesses nodes the metric regions of which (described by the parent routing entries $rout(O_i)$) overlap the query region, i.e. such that $d(O_i, Q) \leq r_{O_i} + r_Q$ is satisfied.

2.2 Nearest Neighbours Search

In fact, the k-NN query algorithm for M-tree is a more complicated range query algorithm. Since the query radius r_Q is not known in advance, it must be determined dynamically (during the query processing). For this purpose a *branch-and-bound* heuristic algorithm has been introduced [5], quite similar to that one for R-trees [8]. The k-NN query algorithm utilizes a priority queue PR of pending requests, and a k-elements array NN used to store the k-NN candidates and which, at the end of the processing, contains the result. At the beginning, the *dynamic radius* r_Q is set to ∞, while during query processing r_Q is consecutively reduced down to the "true" distance between Q and the k-th nearest neighbour.

PR queue. The priority queue PR of pending requests $[ptr(T(O_i)), d_{min}(T(O_i))]$ is used to keep (pointers to) such subtrees $T(O_i)$, which (still) cannot be excluded from the search, due to overlap of their metric regions (O_i, r_{O_i}) with the *dynamic query region* (Q, r_Q). The priority order of each such request is given by $d_{min}(T(O_i))$, which is the smallest possible distance between an object stored in $T(O_i)$ and the query object Q. The smallest distance is denoted as the lower-bound distance between Q and the metric region (O_i, r_{O_i}):

$$d_{min}(T(O_i)) = max\{0, d(O_i, Q) - r_{O_i}\}$$

During k-NN query execution, requests from PR are being processed in the priority order, i.e. the request with smallest lower-bound distance goes first.

NN array. The NN array contains k entries of form either $[oid(O_i), d(Q, O_i)]$ or $[-, d_{max}(T(O_i))]$. The array is sorted according to ascending distance values. Entry of form $[oid(O_i), d(Q, O_i)]$ on the j-th position in NN represents a candidate object O_i for the j-th nearest neighbour. In the second case (i.e. entry of form $[-, d_{max}(T(O_i))]$), the value $d_{max}(T(O_i))$ represents upper-bound distance between Q and objects in subtree $T(O_i)$ (in which some k-NN candidates could be stored). The upper-bound distance $d_{max}(T(O_i))$ is defined as:

$$d_{max}(T(O_i)) = d(O_i, Q) + r_{O_i}$$

Since NN is a sorted array containing the k nearest neighbours candidates (or at least upper-bound distances of the still relevant subtrees), the dynamic query radius r_Q can be determined as the current distance stored in the last entry NN[k]. During the query processing, only the closer candidates (or smaller upper-bound distances) are inserted into NN array, i.e. such candidates, which are currently located inside the dynamic query region (Q, r_Q).

After insertion into NN, the query radius r_Q is decreased (because NN[k] entry was replaced). The priority queue PR must contain only the (still) relevant subtrees, i.e. such subtrees the regions of which overlap the dynamic query region (Q, r_Q). Hence, after the dynamic radius r_Q is decreased, all irrelevant requests (for which $d_{min}(T(O_i)) > r_Q$) must be deleted from PR.

At the beginning of k-NN search, the NN candidates are unknown, thus all entries in the NN array are set to $[-, \infty]$. The query processing starts at the root level, so that $[ptr(root), \infty]$ is the first and only request in PR. For a more detailed description of the k-NN query algorithm we refer to [7, 10].

Note: The k-NN query algorithm is optimal in I/O costs, since it only accesses nodes, the metric regions of which overlap the query region $(Q, d(Q,\text{NN}[k].d_{max}))$. In other words, the I/O costs of a k-NN query (Q, k) and I/O costs of the equivalent range query $(Q, d(Q,\text{NN}[k].d_{max}))$ are equal.

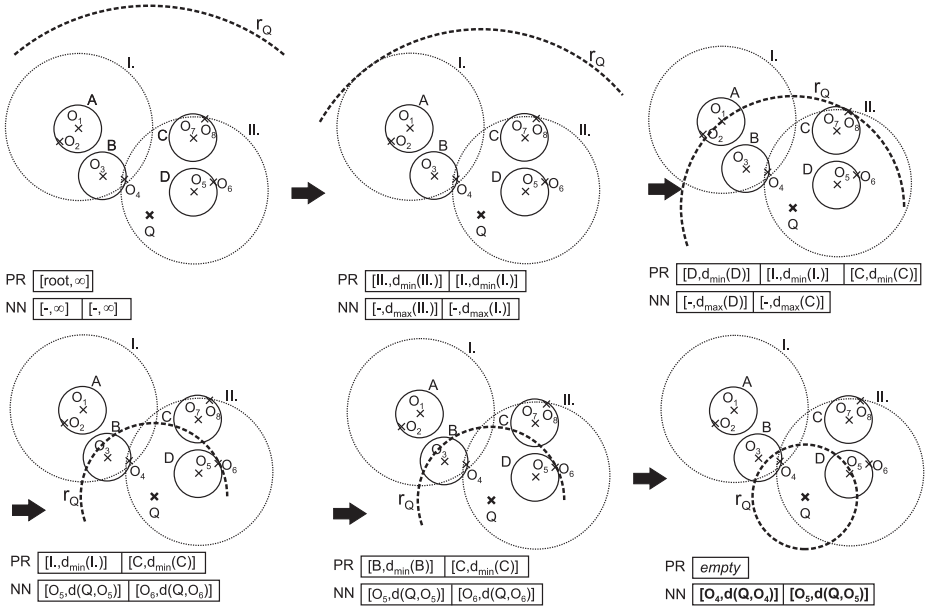


Fig. 2. An example of 2-NN search in M-tree.

Example 1

In Figure 2 see an example of 2-NN query processing. Each of the depicted phases shows the content of PR queue and NN array, right before processing a request from PR. Due to the decreasing query radius r_Q, the dynamic query region (Q, r_Q) (represented by bold-dashed line) is reduced down to $(Q, d(Q, O_5))$. Note the algorithm accesses 5 nodes (processing of single request in PR involves a single node access), while the equivalent range query takes also 5 node accesses.

3 PM-tree

Each metric region in M-tree is described by a bounding hyper-sphere. However, the shape of hyper-sphere is far from optimal, since it does not bound the data objects tightly together and the region "volume" is too large. Relatively to the hyper-sphere volume, there are only "few" objects spread inside the hyper-sphere – a huge proportion of dead space [1] is covered. Consequently, for hyper-spherical regions the probability of overlap with query region grows, thus query processing becomes less efficient. This observation was the major motivation for introduction of the *Pivoting M-tree* (PM-tree) [12, 10], an extension of M-tree.

3.1 Structure of PM-tree

Some metric access methods (e.g. AESA, LAESA [4, 6]) exploit *global static pivots*, i.e. objects to which all objects of the dataset $\mathbb{S}$ (all parts of the index structure respectively) are related. The global pivots actually represent "anchors" or "viewpoints", due to which better filtering of irrelevant data objects is possible.

In PM-tree, the original M-tree hierarchy of hyper-spherical regions (driven by local pivots) is combined with so-called *hyper-ring regions*, centered in global pivots. Since PM-tree is a generalization of M-tree, we just describe the new facts instead of a comprehensive definition. First of all, a set of p global pivots $P_t \in \mathbb{S}$ must be chosen. This set is fixed for all the lifetime of a particular PM-tree index. A routing entry in PM-tree inner node is defined as:

$$rout_{PM}(O_i) = [O_i, ptr(T(O_i)), r_{O_i}, d(O_i, \mathrm{Par}(O_i)), \mathrm{HR}]$$

The new HR attribute is an array of p_{hr} intervals ($p_{hr} \leq p$), where the t-th interval HR[t] is the smallest interval covering distances between the pivot P_t and each of the objects stored in leaves of $T(O_i)$, i.e. HR[t] = $\langle$HR[t].min, HR[t].max$\rangle$, HR[t].min = $min\{d(O_j, P_t)\}$, HR[t].max = $max\{d(O_j, P_t)\}$, $\forall O_j \in T(O_i)$. The interval HR[t] together with pivot P_t define a hyper-ring region (P_t,HR[t]); a hyper-spherical region (P_t,HR[t].max) reduced by a "hole" (P_t,HR[t].min).

Since each hyper-ring region (P_t, HR[t]) defines a metric region bounding *all* the objects stored in $T(O_i)$, the intersection of all the hyper-rings and the hyper-sphere forms a metric region bounding *all* the objects in $T(O_i)$ as well. Due to the intersection with hyper-sphere, the PM-tree metric region is always smaller than the original hyper-spherical region. The probability of overlap between PM-tree region and query region is smaller, thus the search becomes more efficient (see Figure 3). A ground entry in PM-tree leaf is defined as:

$$grnd_{PM}(O_i) = [O_i, oid(O_i), d(O_i, \mathrm{Par}(O_i)), \mathrm{PD}]$$

The new PD attribute stands for an array of p_{pd} pivot distances ($p_{pd} \leq p$) where the t-th distance PD[t] = $d(O_i, P_t)$. The distances PD[t] between data objects and the global pivots are used for simple sequential filtering in leaves, as it is accomplished in LAESA-like methods. For details concerning PM-tree construction as well as representation and storage of the hyper-ring intervals (HR and PD arrays) we refer to [12, 10].

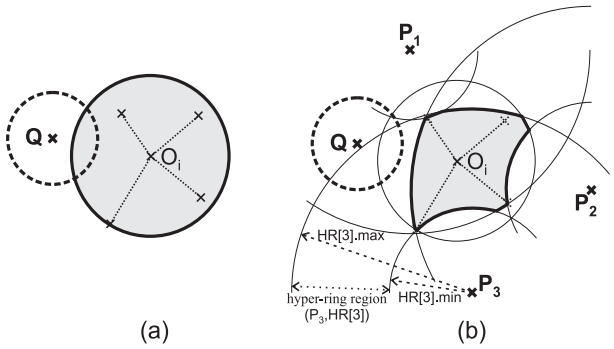


Fig. 3. (a) Region of M-tree. (b) Region of PM-tree (sphere reduced by 3 hyper-rings).

3.2 Choosing the Global Pivots

Problems about choosing the global pivots have been intensively studied for a long time [9, 3, 2]. In general, we can say that pivots should be far from each other (close pivots give almost the same information) and outside data clusters. Distant pivots cause increased variance in distance distribution [4] (the dataset is "viewed" from different "sides"), which is reflected in better filtering properties.

We use a cheap but effective method of pivots choice, described as follows. First, m groups of p objects are randomly sampled from the dataset $\mathbb{S}$, each group representing a candidate set of pivots. Second, such group of pivots is chosen, for which the sum of distances between objects is maximal.

3.3 Similarity Queries in PM-tree

The distances $d(Q, P_t), \forall t \leq max(p_{hr}, p_{pd})$ have to be computed before the query processing itself is started. The query is processed by accessing nodes, the regions of which are overlapped by the query region (similarly as M-tree is queried, see Section 2.1). A PM-tree node is accessed if the query region overlaps *all* the hyper-rings stored in the parent routing entry. Hence, prior to the standard hyper-sphere overlap check (used by M-tree), the overlap of hyper-rings HR[t] against the query region is tested as follows (no additional distance is computed):

$$\bigwedge_{t=1}^{p_{hr}} d(Q, P_t) - r_Q \leq \mathrm{HR}[t].\mathrm{max} \ \wedge \ d(Q, P_t) + r_Q \geq \mathrm{HR}[t].\mathrm{min} \tag{1}$$

If the above condition is false, the subtree $T(O_i)$ is not relevant to the query, and can be excluded from further processing. At the leaf level, an irrelevant ground entry is determined such that the following condition is not satisfied:

$$\bigwedge_{t=1}^{p_{pd}} |d(Q, P_t) - \mathrm{PD}[t]| \leq r_Q \tag{2}$$

In Figure 3 see that M-tree region cannot be filtered out, but PM-tree region can be excluded from the search, since the hyper-ring HR[2] is not overlapped.

4 Nearest Neighbours Search in PM-tree

The hyper-ring overlap condition (1) can be integrated into the original M-tree's range query as well as into k-NN query algorithms. In case of range query the adjustment is straightforward – the hyper-ring overlap condition is combined with the original hyper-sphere overlap condition (we refer to [12]).

The M-tree's k-NN algorithm can be modified for the PM-tree, we only need to respect the changed region shape. As in the range query algorithm, the check for overlap between the query region and a PM-tree region is combined with the hyper-ring overlap condition (1). Furthermore, to obtain an *optimal* k-NN algorithm, there must be adjusted the lower-bound distance d_{min} (used by PR queue) and the upper-bound distance d_{max} (used by NN array), as follows.

The requests $[ptr(T(O_i)), d_{min}(T(O_i))]$ in PR represent the relevant subtrees $T(O_i)$ to be examined, i.e. such subtrees, the parent metric regions of which overlap the dynamic query region (Q, r_Q). Taking the hyper-rings HR$[t]$ of a PM-tree region into account, the lower-bound distance is possibly increased, as:

$$d_{min}(T(O_i)) = max\{0, d(O_i, Q) - r_{O_i}, d^{low}_{HRmax}, d^{low}_{HRmin}\}$$

$$d^{low}_{HRmax} = max \bigcup_{t=1}^{p_{hr}} \{d(P_t, Q) - \text{HR}[t].\text{max}\} \quad d^{low}_{HRmin} = max \bigcup_{t=1}^{p_{hr}} \{\text{HR}[t].\text{min} - d(P_t, Q)\}$$

where $max\{d^{low}_{HRmax}, d^{low}_{HRmin}\}$ determines the lower-bound distance between the query object Q and objects located in the farthest hyper-ring. Comparing to M-tree's k-NN algorithm, the lower-bound distance $d_{min}(T(O_i))$ for a PM-tree region can be additionally increased, since the farthest hyper-ring contains all the objects stored in $T(O_i)$.

The entries $[oid(O_i), d(Q, O_i)]$ or $[-, d_{max}(T(O_i))]$ in NN represent the current k candidates for nearest neighbours (or at least the still relevant subtrees). Taking the hyper-rings HR$[t]$ into account, the upper-bound distance $d_{max}(T(O_i))$ is possibly decreased, as:

$$d_{max}(T(O_i)) = min\{d(O_i, Q) + r_{O_i}, d^{up}_{HR}\} \quad d^{up}_{HR} = min \bigcup_{t=1}^{p_{hr}} \{d(P_t, Q) + \text{HR}[t].\text{max}\}$$

where d^{up}_{HR} determines the upper-bound distance between the query object Q and objects located in the nearest hyper-ring.

In summary, the modification of M-tree's k-NN algorithm for the PM-tree differs in the overlap condition, which has to be additionally combined with the hyper-ring overlap check (1) and (2), respectively. Another difference is in the construction of $d_{max}(T(O_i))$ and $d_{min}(T(O_i))$ bounds.

Example 2

In Figure 4 see an example of 2-NN query processing. The PM-tree hierarchy is the same as the M-tree hierarchy presented in Example 1, but the query processing runs a bit differently. Although in this particular example both the M-tree's and the PM-tree's k-NN query algorithms access 4 nodes, searching the PM-tree saves one insertion into the PR queue.

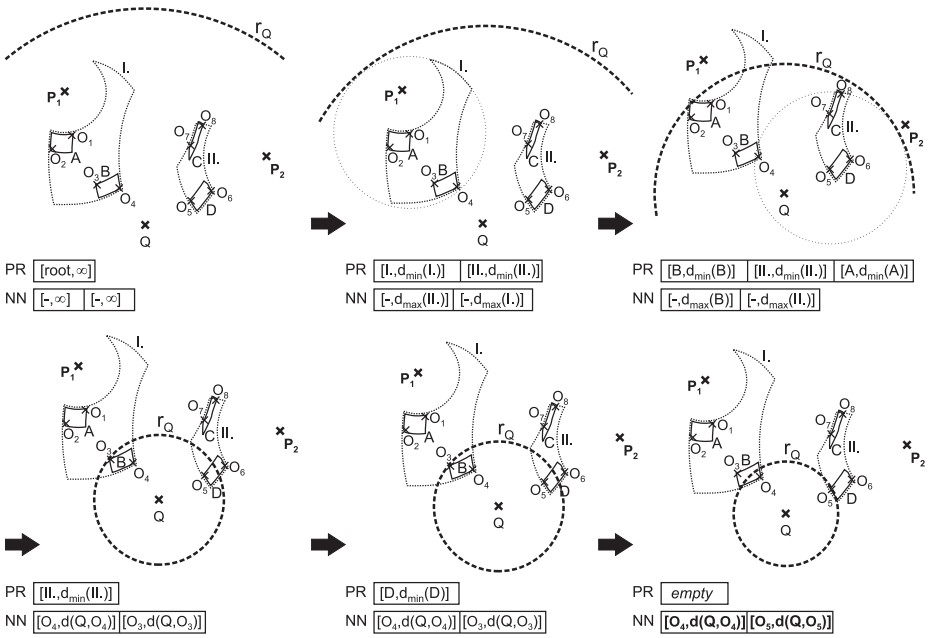


Fig. 4. An example of 2-NN search in PM-tree.

Note: Like the M-tree's k-NN query algorithm, also the PM-tree's k-NN query algorithm is optimal in I/O costs, since it only accesses those PM-tree nodes, the metric regions of which overlap the query region $(Q, d(Q,\text{NN}[k].d_{max}))$. This is guaranteed (besides usage of the hyper-ring overlap check) by correct modification of lower/upper distance bounds stored in PR queue and NN array.

5 Experimental Results

In order to evaluate the performance of k-NN search, we present some experiments made on large synthetic as well as real-world vector datasets. The query objects were selected randomly from each respective dataset, while each particular test consisted of 1000 queries (the results were averaged). Euclidean (L_2) metric was used in all tests. The I/O costs were measured as the number of logic disk page retrievals. The experiments were aimed to compare PM-tree with M-tree – a comparison with other MAMs was out of scope of this paper.

Abbreviations in Figures. Each label of form "`PM-tree(x,y)`" stands for a PM-tree index where p_{hr} = `x` and p_{pd} = `y`. A label "<*index*> `+ SlimDown`" denotes an index subsequently post-processed by the slim-down algorithm [11, 10].

5.1 Synthetic Datasets

For the first set of experiments, a collection of 8 synthetic vector datasets of increasing dimensionality (from $D = 4$ to $D = 60$) was generated. Each dataset

(embedded inside unitary hyper-cube) consisted of 100,000 D-dimensional tuples distributed uniformly among 1000 L_2-spherical uniformly distributed clusters. The diameter of each cluster was $\frac{d^+}{10}$ (where $d^+ = \sqrt{D}$). These datasets were indexed by PM-tree (for various p_{hr} and p_{pd}) as well as by M-tree. Some statistics about the created indices are shown in Table 1 (for details see [11]).

Table 1. PM-tree index statistics (synthetic datasets).

Construction methods: `SingleWay + MinMax (+ SlimDown)`	
Dimensionalities: 4,8,16,20,30,40,50,60	Inner node capacities: 10 – 28
Index file sizes: 4.5 MB – 55 MB	Leaf node capacities: 16 – 36
Pivot file sizes: 2 KB – 17 KB	Avg. node utilization: 66%
Node (disk page) sizes: 1 KB ($D = 4, 8$), 2 KB ($D = 16, 20$), 4 KB ($D \geq 30$)	

Prior to k-NN experiments, in Figure 5 we present index construction costs (for 30-dimensional indices), according to the increasing number of pivots. The increasing I/O costs depend on the hyper-ring storage overhead (the storage ratio of PD or HR arrays to the data vectors becomes higher), while the increasing computation costs depend on the object-to-pivot distance computations performed before each object insertion.

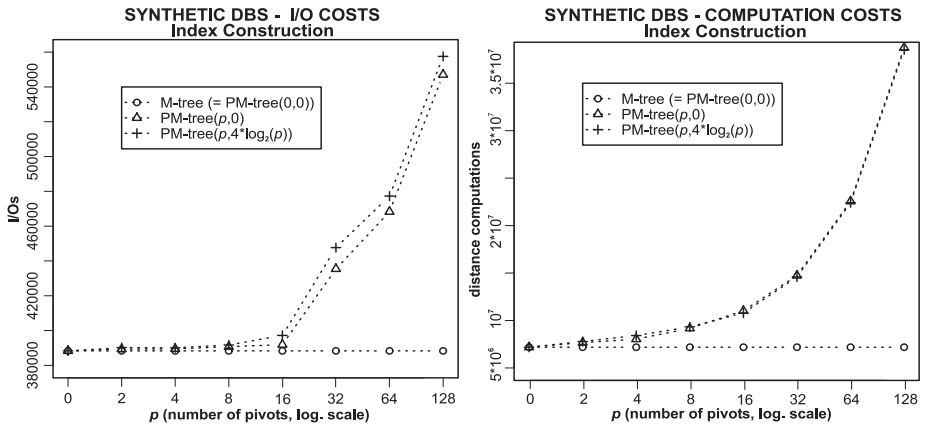


Fig. 5. Number of pivots: (a) I/O costs. (b) Computation costs.

In Figure 6 the 20-NN search costs (for 30-dimensional indices) according to the number of pivots are presented. The I/O costs rapidly decrease with the increasing number of pivots. Moreover, the PM-tree is superior even after post-processing by the slim-down algorithm. The decreasing trend of computation costs is even quicker than of I/O costs, see Figure 6b.

The influence of increasing dimensionality D is depicted in Figure 7. Since the disk pages for different (P)M-tree indices were not of the same size, the I/O costs as well as the computation costs are related (in percent) to the I/O costs (CC resp.) of M-tree indices. For $8 \leq D \leq 40$ the I/O costs stay approximately fixed, for $D > 40$ they slightly increase. In case of $D = 4$, the higher PM-tree I/O costs are caused by higher hyper-ring storage overhead.

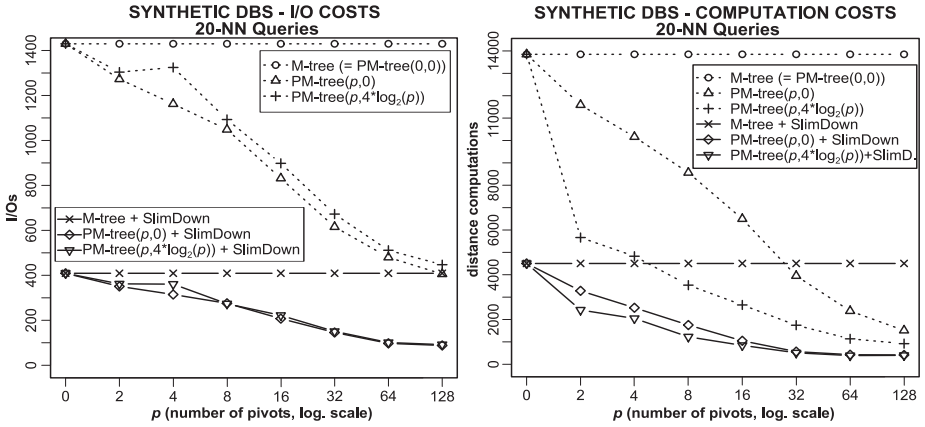


Fig. 6. Number of pivots: (a) I/O costs. (b) Computation costs.

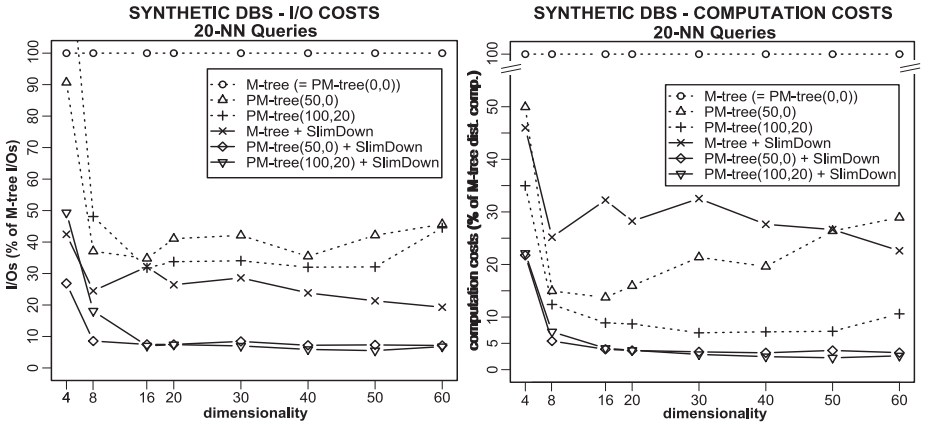


Fig. 7. Dimensionality: (a) I/O costs. (b) Computation costs.

5.2 Image Database

For the second set of experiments, a collection of approx. 10,000 web-crawled images [13] was used. Each image was converted into 256-level gray scale and a frequency histogram was extracted. As indexed objects the histograms (256-dimensional vectors) were used. The index statistics are presented in Table 2.

Table 2. PM-tree index statistics (image database).

Construction methods: `SingleWay + MinMax (+ SlimDown)`	
Dimensionality: 256	Inner node capacities: 10 – 31
Index file sizes: 16 MB – 20 MB	Leaf node capacities: 29 – 31
Pivot file sizes: 4 KB – 1 MB	Avg. node utilization: 67%
Node (disk page) size: 32 KB	

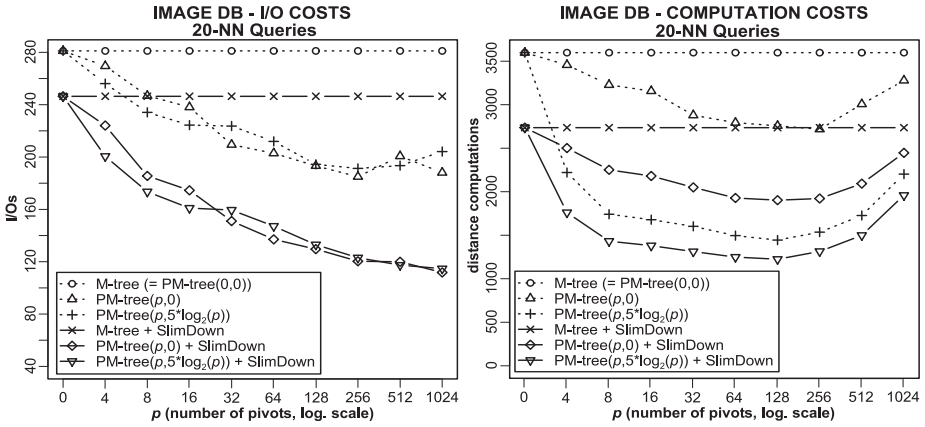


Fig. 8. Number of pivots: (a) I/O costs. (b) Computation costs.

In Figure 8a the I/O search costs for increasing number of pivots are presented. The computation costs (see Figure 8b) for $p \leq 64$ decrease. However, for $p > 64$ the overall computation costs grow, since the number of necessarily computed query-to-pivot distances (i.e. p distance computations for each query) is proportionally too large. Nevertheless, this observation is dependent on the database size – obviously, for million of images the proportion of p query-to-pivot distance computations would be smaller, when compared with the overall computation costs. Finally, the costs according to the increasing number of nearest neighbours are presented in Figure 9.

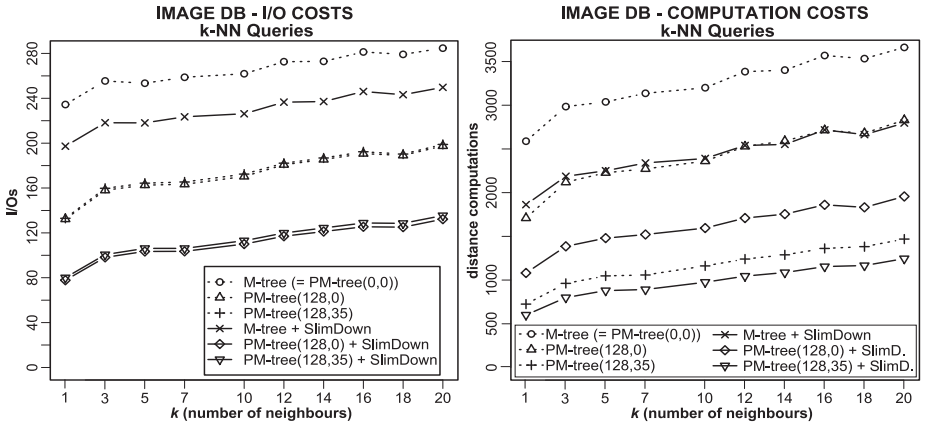


Fig. 9. Number of neighbours: (a) I/O costs. (b) Computation costs.

6 Conclusions

We have proposed an optimal k-NN search algorithm for the PM-tree. Experimental results on synthetic and real-world datasets have shown that searching in PM-tree is significantly more efficient, when compared with the M-tree.

This research has been partially supported by grant 201/05/P036 of the Czech Science Foundation (GAČR) and the National programme of research (Information society project 1ET100300419).

References

1. C. Böhm, S. Berchtold, and D. Keim. Searching in High-Dimensional Spaces – Index Structures for Improving the Performance of Multimedia Databases. *ACM Computing Surveys*, 33(3):322–373, 2001.
2. B. Bustos, G. Navarro, and E. Chávez. Pivot selection techniques for proximity searching in metric spaces. *Pattern Recognition Letters*, 24(14):2357–2366, 2003.
3. E. Chávez. Optimal discretization for pivot based algorithms. Manuscript. `ftp://garota.fismat.umich.mx/pub/users/elchavez/minimax.ps.gz`, 1999.
4. E. Chávez, G. Navarro, R. Baeza-Yates, and J. Marroquín. Searching in Metric Spaces. *ACM Computing Surveys*, 33(3):273–321, 2001.
5. P. Ciaccia, M. Patella, and P. Zezula. M-tree: An Efficient Access Method for Similarity Search in Metric Spaces. In *Proceedings of the 23rd Athens Intern. Conf. on VLDB*, pages 426–435. Morgan Kaufmann, 1997.
6. M. L. Micó, J. Oncina, and E. Vidal. A new version of the nearest-neighbour approximating and eliminating search algorithm (aesa) with linear preprocessing time and memory requirements. *Pattern Recognition Letters*, 15(1):9–17, 1994.
7. M. Patella. *Similarity Search in Multimedia Databases*. PhD thesis, University of Bologna, 1999.
8. N. Roussopoulos, S. Kelley, and F. Vincent. Nearest neighbor queries. In *Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data, San Jose, CA*, pages 71–79, 1995.
9. M. Shapiro. The choice of reference points in best-match file searching. *Commun. ACM*, 20(5):339–343, 1977.
10. T. Skopal. *Metric Indexing in Information Retrieval*. PhD thesis, Technical University of Ostrava, `http://urtax.ms.mff.cuni.cz/~skopal/phd/thesis.pdf`, 2004.
11. T. Skopal, J. Pokorný, M. Krátký, and V. Snášel. Revisiting M-tree Building Principles. In *Proceedings of the 7th East-European Conference on Advances in Databases and Information Systems (ADBIS), Dresden, Germany, LNCS 2798, Springer-Verlag*, pages 148–162, 2003.
12. T. Skopal, J. Pokorný, and V. Snášel. PM-tree: Pivoting Metric Tree for Similarity Search in Multimedia Databases. In *Local proceedings of the 8th East-European Conference on Advances in Databases and Information Systems (ADBIS), Budapest, Hungary*, pages 99–114, 2004.
13. WBIIS project: Wavelet-based Image Indexing and Searching, Stanford University, `http://wang.ist.psu.edu/`.

Chapter 5

Dynamic Similarity Search in Multi-Metric Spaces

Benjamin Bustos
Tomáš Skopal

Dynamic Similarity Search in Multi-Metric Spaces [16]

Poster paper at the 8th ACM international workshop on Multimedia information retrieval (a part of ACM Multimedia conference), Santa Barbara, CA, USA, October 2006

Published in ACM proceedings, pages 137–146, *ACM Press*, ISBN 1-59593-495-2

Dynamic Similarity Search in Multi-Metric Spaces

Benjamin Bustos
Department of Computer & Information Science
University of Konstanz, Germany
bustos@informatik.uni-konstanz.de

Tomáš Skopal
Department of Software Engineering, FMP
Charles University in Prague, Czech Republic
tomas.skopal@mff.cuni.cz

ABSTRACT

An important research issue in multimedia databases is the *retrieval of similar objects*. For most applications in multimedia databases, an exact search is not meaningful. Thus, much effort has been devoted to develop efficient and effective similarity search techniques. A recent approach, that has been shown to improve the effectiveness of similarity search in multimedia databases, resorts to the usage of combinations of metrics where the desirable contribution (weight) of each metric is chosen at query time. This paper presents the *Multi-Metric M-tree* (M^3-tree), a metric access method that supports similarity queries with dynamic combinations of metric functions. The M^3-tree, an extension of the M-tree, stores partial distances to better estimate the weighed distances between routing/ground entries and each query, where a single distance function is used to build the whole index. An experimental evaluation shows that the M^3-tree may be as efficient as having multiple M-trees (one for each combination of metrics).

Categories and Subject Descriptors

H.3.1 [**Information Storage and Retrieval**]: Content analysis and indexing—*indexing methods*

General Terms

Algorithms, performance, design

Keywords

Content-based indexing and retrieval, combination of metric functions, nearest neighbor queries

1. INTRODUCTION

Similarity search in multimedia database systems is becoming increasingly important, due to a rapidly growing amount of available multimedia data like images, audio files, video clips, 3D objects, time series, and text documents. As we see progress in the fields of acquisition, storage, and dissemination of various multimedia formats, the application of effective and efficient database management systems becomes indispensable in order to handle these formats. The application domains for multimedia databases include molecular biology, medicine, geographical information systems, Computer Aided Design/Computer Aided Manufacturing (CAD/CAM), virtual reality, and many others:

a) In medicine, the detection of similar organ deformations can be used for diagnostic purposes [11].

b) Biometric devices (e.g., fingerprint scanners) read a physical characteristic from an individual and then search in a database to verify if the individual is registered or not. The search cannot be exact, as the probability that two fingerprint scans, even from the same person, are exactly equal (bit-to-bit) is very low.

c) A 3D object database can be used to support CAD tools. For example, standard parts in a manufacturing company can be modeled as 3D objects. When a new product is designed, it can be composed of many small parts that fit together to form the product. If some of these parts are similar to one of the standard parts already designed, then the possible replacement of the original part with the standard part can lead to a reduction of production costs.

d) In text databases, a typical query consists of a set of keywords or a whole document. The search system looks in the database for documents that are relevant to the given keywords or that are similar to the query document. A certain tolerance on the search may be allowed in case, e.g., that some of the given keywords were mistyped or an optical character recognition (OCR) system was used to scan the documents (thus they may contain some misspelled words).

1.1 Preliminaries

Many of these practical applications have in common that the objects of the database are modeled in a *metric space* [6, 15], i.e., it is possible to define a positive real-valued function δ among the objects, called *metric*, that satisfies the properties of *strict positiveness* ($\delta(x,y) \geq 0$ and $\delta(x,y) = 0 \Leftrightarrow x = y$), *symmetry* ($\delta(x,y) = \delta(y,x)$), and the *triangle inequality* ($\delta(x,z) \leq \delta(x,y) + \delta(y,z)$). The main motivation for using metric spaces is the fact that they are easily indexable by metric access methods (described later).

An important particular case of metric spaces are *vector spaces*, where the objects are tuples of d real values, i.e., they are vectors in $\mathbb{R}^d$. There are many metric functions defined on vector spaces, e.g., the *Minkowski distances*, defined as $L_p(x,y) = \left(\sum_{1 \leq i \leq d} |x_i - y_i|^p\right)^{1/p},\ p \geq 1,\ x, y \in \mathbb{R}^d$.

MIR'06, October 26–27, 2006, Santa Barbara, California, USA.

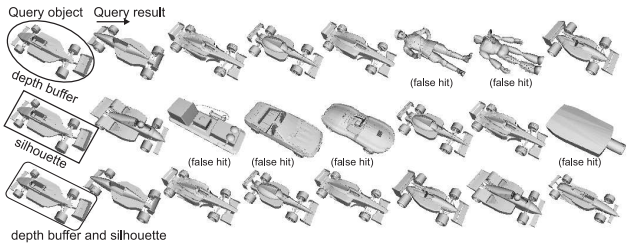


Figure 1: Improving effectiveness of 3D similarity search by combining two 3D feature vectors.

1.2 Simple vs. Combined Metrics

A recent proposal to improve the effectiveness (i.e., the quality of the retrieved answer) of similarity search resorts to the use of *combinations of metrics* [2, 3]. Instead of using a single metric to compare two objects, the search system uses a *linear combination of metrics* to compute the (dis)similarity between two objects. Figure 1 shows an example of the benefits obtained by using such a combined metric. The first two rows show the similar objects retrieved by a 3D similarity search system using two different single-feature vectors (depth buffer *or* silhouette) – a single metric works with the entire particular vector. In both queries, the result includes some non-relevant objects (false hits). The third row shows the result of the search when using both features for each 3D object description (depth buffer *and* silhouette). In this case a combination of the two metrics is used on the double-feature vector, while only relevant objects are retrieved for this time.

The problem with a static combination of metrics (i.e., where the weights of the linear combination are fixed) is that usually not all metrics are well-suited for performing similarity search with all query objects. Moreover, a bad-suited metric may "spoil" the final result of the query. Thus, to further improve the effectiveness of the search system, *methods for dynamic combinations of metrics* have been proposed, where the query processor weighs the contribution of each metric *depending on the query object* (i.e., big weights are assigned to the "good" metrics for that query object, and low weights are assigned to the "bad metrics", according to some quality criteria). This means that, instead of a single metric, the system uses a *dynamic metric function* (multi-metric), where a different metric is computed to perform each similarity query.

1.3 Paper Contributions

This paper presents the *Multi-Metric M-tree (M^3-tree)*, a dynamic index structure that extends the M-tree [8] to support multi-metric similarity queries. We first describe how to adapt the search algorithms of the original M-tree to directly support multi-metric queries. Then, we describe the M^3-tree data structure and the new similarity search algorithms. We show experimentally that the M^3-tree outperforms the adapted M-tree for multi-metrics, and that its efficiency is very close to having multiple M-trees, one for each used multi-metric, which is the optimal achievable efficiency regarding to this index structure.

Note that in this paper we only deal with the efficiency issues of similarity search in multi-metric spaces. For a discussion on the effectiveness of this approach, see [2, 3].

Table 1: Notation used in this paper.

Symbol	Definition
$\mathbb{U}$	set of valid objects (the universe)
$\mathbb{S} \subset \mathbb{U}$	database
$n = \lvert\mathbb{S}\rvert$	database size
$\delta(x, y)$	A metric function
$\mathbb{M} = \langle\delta_i\rangle$	vector of metric functions
$\mathbb{W} = \langle w_i\rangle$	vector of weights
$\lvert\mathbb{M}\rvert = \lvert\mathbb{W}\rvert = m$	number of weights and metrics
$\Delta_{\mathbb{W}}(x, y)$	linear multi-metric
$\Delta_{1.0}(x, y)$	linear multi-metric where $w_i = 1$
$r_{\mathbb{W}}$	$\Delta_{\mathbb{W}}$-based covering radius
$r_{1.0}$	$\Delta_{1.0}$-based covering radius
$Q \in \mathbb{U}$	query object
$\varepsilon_{\mathbb{W}}$	tolerance of a range query (query radius, $\Delta_{\mathbb{W}}$-based)

2. SIMILARITY SEARCH IN METRIC AND MULTI-METRIC SPACES

Table 1 shows the notation used through this paper. Let $(\mathbb{U}, \delta)$ be a metric space and let $\mathbb{S} \subset \mathbb{U}$ be a set of objects (i.e., an instance of a database). There are two typical similarity queries in metric spaces:

- *Range query.* A range query (Q, ε), $Q \in \mathbb{U}$, $\varepsilon \in \mathbb{R}^+$, reports all database objects that are within a tolerance distance ε to Q, that is $(Q, \varepsilon) = \{O_i \in \mathbb{S}, \delta(O_i, Q) \leq \varepsilon\}$. The subspace $\mathbb{V} \subset \mathbb{U}$ defined by Q and ε (i.e., $\forall v \in \mathbb{V}$ $\delta(v, Q) \leq \varepsilon$ and $\forall x \in \mathbb{U} - \mathbb{V}$ $\delta(x, Q) > \varepsilon$) is called the *query ball*.

- *k nearest neighbors query (k-NN).* It reports the k objects from $\mathbb{S}$ closest to Q. That is, it returns the set $\mathbb{C} \subseteq \mathbb{S}$ such that $\lvert\mathbb{C}\rvert = k$ and $\forall O_i \in \mathbb{C}$, $O_j \in \mathbb{S} - \mathbb{C}$, $\delta(O_i, Q) \leq \delta(O_j, Q)$.

Metric access methods (MAMs) [6] are index structures designed to perform efficiently similarity queries in metric spaces. They only use the metric properties of δ, especially the triangle inequality, to filter out objects or entire regions of the space during the search, thus avoiding the sequential (or linear) scan over the database.

MAMs can be classified into two main groups: (1) *Pivot-based MAMs* select from the database a number of *pivot* objects, and classify all the other objects according to their distance from the pivots (2) *MAMs based on compact partitions* divide the space into *regions* as compact as possible. Each region stores a representative point (*local pivot*) and data that can be used to discard the entire region at query time, without computing the actual distance from the region objects to the query object. Each region can be partitioned recursively into more regions, inducing a *search hierarchy*.

2.1 M-tree

The *M-tree* [8] is a dynamic (meaning easily updatable) index structure that provides good performance in secondary memory. The M-tree is a hierarchical index, where some of the data points are selected as centers (local pivots) of regions and the rest of the objects are assigned to suitable regions in order to build up a balanced and compact hierarchy of data regions. Each region (branch of the tree) is indexed recursively. The data is stored in the leaves of the M-tree, where each leaf contains *ground entries* ($grnd(O_i)$, $O_i \in \mathbb{S}$). The internal nodes store *routing entries* ($rout(O_i)$, $O_i \in \mathbb{S}$).

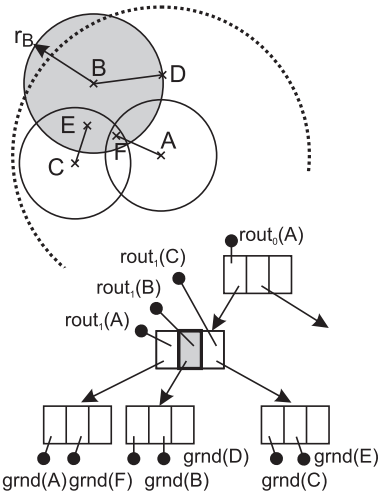


Figure 2: Example of an M-tree.

Starting at the root level, a new object O_i is recursively inserted into the best subtree $T(O_j)$, which is defined as the one where the *covering radius* r_{O_j} must increase the least in order to cover the new object. In case of ties, the subtree whose center is closest to O_i is selected. The insertion algorithm proceeds recursively until a leaf is reached and O_i is inserted into that leaf, at each level storing the distance to the routing object of its parent node (so-called *to-parent distance*). Node overflows are managed in a similar way as in the B-tree. If an insertion produces an overflow, two objects from the node are selected as new centers, the node is split, and the two new centers are promoted to the parent node. If the parent node overflows, the same split procedure is applied. If the root overflows, it is split and a new root is created. Thus, the M-tree is a balanced tree (see Figure 2).

Range queries are implemented by traversing the tree, starting from the root. The nodes which parent region (described by the routing entry) is overlapped by the query ball are accessed (this requires a distance computation). As each node in the tree (except for the root) contains the distances from the routing/ground entries to the center of its parent node (the to-parent distances), some of the non-relevant branches can be further filtered out, without the need of a distance computation, thus avoiding the "more expensive" basic overlap check.

2.2 Searching in Multi-Metric Spaces

Usually, a single metric function is used to compute the similarity between two objects in the metric space. However, a recent trend to improve the effectiveness of the similarity search resorts to use *several metric functions*. The (dis)similarity function is computed as a linear combination of some selected metrics. It follows (from metric spaces theory) that the combined distance function is also a metric.

DEFINITION 1. (linear multi-metric)
Let $\mathbb{M} = \langle \delta_i \rangle$ be a vector of metric functions, and let $\mathbb{W} = \langle w_i \rangle$ be a vector of weights, with $|\mathbb{M}| = |\mathbb{W}| = m$ and $\forall i\ w_i \in [0, 1]$. The *linear multi-metric* (or *linear combined metric function*) is defined as

$$\Delta_{\mathbb{W}}(O_1, O_2) = \sum_{i=1}^{m} w_i \cdot \delta_i(O_1, O_2).$$

A *linear multi-metric space* is defined as $\mathcal{MM} = (\mathbb{U}, \Delta_{\mathbb{W}})$. □

Some notes:

- The multi-metric (space) is denoted as "linear" (in the rest of the paper implicitly assumed), but some other combinations of metrics can be considered in the future, e.g., maximal, multiplicative, etc.
- $\Delta_{1.0}(\cdot) = \Delta_{\mathbb{W}}(\cdot)$ where $\forall i\ w_i = 1$.
- As a consequence, $\Delta_{1.0}(\cdot)$ is an upper-bounding metric to $\Delta_{\mathbb{W}}(\cdot)$ (considering shared $\mathbb{M}$ and any $\mathbb{W}$).
- The vector of weights $\mathbb{W}$ is not included in the definition of multi-metric (space), in fact, it is a parameter of Δ. Consequently, we can view a single multi-metric space as a space covering an infinite number of metric spaces $\mathcal{M}_i = (\mathbb{U}, \Delta_{\mathbb{W}_i})$, where $\mathbb{M}$ is fixed for all the spaces but $\mathbb{W}_i$ is unique for each metric defined on $\mathcal{M}_i$.
- The structure of the universe $\mathbb{U}$ can be either a cartesian product of various domains (even a mix of vector/metric space domains) where each domain is assigned to the respective partial metric δ_i, or a single "flat" domain allowing the δ_is to share some portions of $\mathbb{U}$ (even all being defined on entire $\mathbb{U}$). Nevertheless, in the following we do not need to specify the structure of $\mathbb{U}$ and we assume each partial metric function δ_i "knows" its sub-domain within $\mathbb{U}$.

If the weights of the combination are fixed, the multi-metric space becomes an ordinary metric space and we can use any standard MAM as an index structure. In our framework, however, the weights are *dynamic* – computed at query time – and therefore the metric function is dynamic and depends on the query objects. This has been shown to provide the best effectiveness results [2, 3]. Thus, our problem is to develop a metric index structure that returns the correct answer to the similarity query, even if the *query distance function* is not the same as the distance function used to build the index (*index distance function*). The optimal solution would be to have an index structure for each "fixed multi-metric", but this is not practical because it would imply to build an index for each query, which would be more expensive than performing a sequential scan of the database.

In Section 3, we will describe modifications to the search algorithms of the standard M-tree, that allow us to use it with multi-metrics. Then, in Section 4 we will present our proposed index structure, the M^3-tree, which stores partial distances to dynamically estimate an upper bound of the covering radius with respect to a query-specified metric function, and to estimate the to-parent distances between routing objects and child nodes. These estimations will be used to improve the filtering capability of the index structure, thus improving the efficiency of the similarity search.

2.3 Related Work

Many indexing methods and algorithms have been proposed for implementing similarity queries in metric and vector spaces [6, 1]. However, basically all these index structures have been designed for single metrics, and they do not support dynamic combinations of metrics at query time. One exception is the *branch-and-bound on decomposed data* (BOND) technique [9], which is a spatial access method

(SAM) that can support queries with combinations of feature vectors. The BOND index maintains tables with the coefficients of each dimension for all vectors of the database. These tables are scanned sequentially at query time, computing lower and upper bounds to the distance from the query to the stored vectors and discarding those that cannot belong to the k-NN. The efficiency of the search is improved by scanning on each iteration only the non-discarded objects, thus at the last stages of the algorithm only a small part of the database has to be checked. To compute the lower and upper bound distances, it is necessary to store an auxiliary table with the partial results. In the worst case, the auxiliary table has size $O(n)$, thus the scalability in database size of this technique is limited. Drawbacks of this technique are that the similarity measure must be bounded and it only works in vector spaces.

A MAM specially designed for dynamically weighed combinations of metrics is presented in [5]. This index consists of a set of pivot-based indices, one for each metric, which can be used to compute the *combined pivot table* (i.e., the pivot-based index for the combination of metrics) at query time, when the weights for the dynamic combination are known. The main disadvantage of this index is that it is a main-memory index, and it is not clear how to implement it efficiently in secondary storage.

The *QIC-M-tree* [7] is a MAM designed to support user-defined distance functions. The index is built like a normal M-tree using an *index distance*, and queries may be performed using any distance function that is *lower bounded* by the index distance. While this index structure may be used to perform similarity queries in multi-metric spaces, it is a different approach compared with our proposed index:

- The index distance is an "underscaled" (i.e. not very tight) lower-bounding distance function of the query distance in the QIC-M-tree. In our case, the query distance is a non-scaled lower-bounding distance of the index distance.
- The QIC-M-tree uses lower bounds of the query distance to filter out branches of the tree. The M^3-tree computes a tight approximation of the real query distance (at the cost of a little higher index size), thus providing a better filtering of the space.

3. ADAPTING M-TREE FOR SEARCH IN MULTI-METRIC SPACES

The original M-tree needs to be adapted in order to provide support for multi-metric spaces. The key idea for adapting the M-tree is the use of $\Delta_{1.0}$ for indexing all objects in the index (see Figure 3a). Since $\Delta_{1.0}$ is an upper-bound to any $\Delta_{\mathbb{W}}$, the covering radii $r_{1.0}$ as well as the distances $\Delta_{1.0}(R,P)$ (distance from a routing object to its parent, the to-parent distance) stored in the M-tree nodes can be viewed as upper bounds to the appropriate radii $r_{\mathbb{W}}$ (distances $\Delta_{\mathbb{W}}(R,P)$, respectively), considering any other "index distance" $\Delta_{\mathbb{W}}$. We start proving some lemmas for the adapted discarding criteria.

LEMMA 1. *(basic filtering)*

Let $(Q,\varepsilon_{\mathbb{W}})$ be a range query, where $\varepsilon_{\mathbb{W}}$ is a weighed query radius. Let $(R,r_{1.0})$ represents a routing entry in M-tree, i.e., a data region (note that for $\Delta_{\mathbb{W}}$ we have defined the

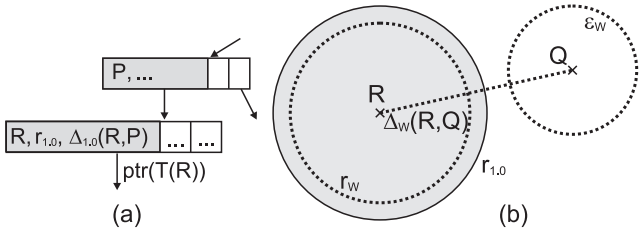


Figure 3: (a) Non-leaf node entries in M-tree. (b) Basic filtering in M-tree.

"real" covering radius as $r_{\mathbb{W}} = \max_{O_i \in T(R)}\{\Delta_{\mathbb{W}}(O_i,R)\}$). If $\Delta_{\mathbb{W}}(R,Q) > \varepsilon_{\mathbb{W}} + r_{1.0}$, the data region is not relevant to the query and can be filtered out.

Proof: For $r_{\mathbb{W}} = r_{1.0}$ it follows (by triangle inequality) that no object from $(R,r_{\mathbb{W}})$ can be located in $(Q,\varepsilon_{\mathbb{W}})$. This property can be extended to all $r_{\mathbb{W}} < r_{1.0}$, since $\Delta_{\mathbb{W}}$ is lower-bounding to $\Delta_{1.0}$, thus objects in $(R,r_{\mathbb{W}})$ are always more (or equally) distant to Q that in case of $\Delta_{1.0}$ (see Figure 3b). ■

Lemma 1 can be used for basic filtering in M-tree, when a data region (covering some subtree) is needed to check against a range query. For this check, the $\Delta_{\mathbb{W}}(R,Q)$ distance must be computed.

LEMMA 2. *(outer parent filtering)*

Let P be the parent object of a data region $(R,r_{1.0})$. If

$$\Delta_{\mathbb{W}}(P,Q) - \Delta_{1.0}(R,P) > r_{1.0} + \varepsilon_{\mathbb{W}}$$

the data region is not relevant to the query and can be filtered out.

Proof: The query object is outside the sphere defined by parent object and radius $\Delta_{1.0}(R,P) + r_{1.0}$ (see Figure 4a). This sphere can be directly used for check with the query (by means of Lemma 1), because the sphere surely covers the data region $(R,r_{1.0})$. This property is guaranteed by the use of the upper bound distance from P to R and by R's covering radius upper bound $r_{1.0}$, so the sphere is always more (or equally) distant to the query than any object in $(R,r_{\mathbb{W}})$. ■

LEMMA 3. *(inner parent filtering)*

Let P be the parent object of a data region $(R,r_{1.0})$. Let $\Delta^{lb}_{\mathbb{W}}(\cdot)$ be a lower-bounding distance to $\Delta_{\mathbb{W}}(\cdot)$. If

$$\Delta^{lb}_{\mathbb{W}}(R,P) - \Delta_{\mathbb{W}}(P,Q) > r_{1.0} + \varepsilon_{\mathbb{W}}$$

the data region is not relevant to the query and can be filtered.

Proof: The query is entirely inside the sphere defined by parent object and radius $\Delta^{lb}_{1.0}(R,P) - r_{1.0}$ (see Figure 4b). Because the actual $\Delta_{1.0}(R,P)$ is upper bound of $\Delta_{\mathbb{W}}(R,P)$, the object R is "artificially shifted" from the parent (i.e., more than by using $\Delta_{\mathbb{W}}$), so we cannot check whether the query does not overlap $(R,r_{\mathbb{W}})$ by directly using $\Delta_{1.0}(R,P)$. However, if we use some distance $\Delta^{lb}_{\mathbb{W}}$ lower-bounding $\Delta_{\mathbb{W}}$ (instead of $\Delta_{1.0}$), we are sure that the "inner border" separating query and the data region is a lower bound of the actual border. ■

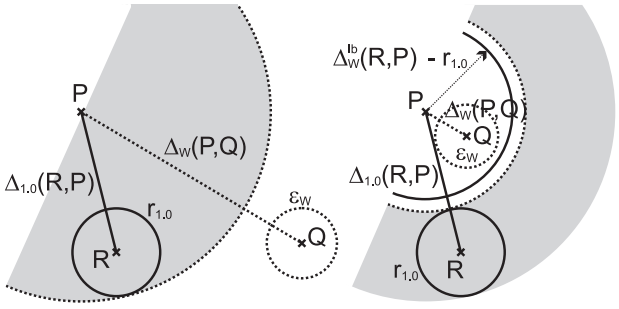


Figure 4: (a) Outer parent filtering in M-tree. (b) Inner parent filtering.

Lemmas 2 and 3 can be used to avoid the basic check (provided by Lemma 1). The advantage is that no extra computation is needed to evaluate the condition in the lemmas, so in many cases the data region is filtered out even without the need of using Lemma 1 (and so without any distance computation).

Up to now, the approach is generally applicable for any index distance $\Delta_{1.0}$ and any lower-bounding query distance $\Delta_{\mathbb{W}}$ (regardless of what the metrics $\Delta_{1.0}$ and $\Delta_{\mathbb{W}}$ really mean), in a similar way as in the QIC-M-tree [7].

However, to construct the lower bound to $\Delta_{\mathbb{W}}$ (needed in Lemma 3), we can exploit the definition of $\Delta_{\mathbb{W}}$ (see Section 2.2). To efficiently compute the lower bound, it is preferable to use some distance already precomputed during the query evaluation, so that no additional distance computation or an explicitly specified lower bound distance (passed as a query parameter) is needed. In the following, we construct such a lower bound just by using the weights vector $\mathbb{W}$.

LEMMA 4. *(lower bound to $\Delta_{\mathbb{W}}$, optimal scaling constant)*
(a) $\Delta^{lb}(\cdot) = \min_{i=1}^{m}(w_i) \cdot \Delta_{1.0}(\cdot)$ is lower bound to $\Delta_{\mathbb{W}}(\cdot)$.
(b) The scaling constant $s = \min_{i=1}^{m}(w_i)$ is the maximal factor for which $\Delta^{lb}(\cdot)$ is still a lower bound of $\Delta_{\mathbb{W}}(\cdot)$ (i.e., such Δ^{lb} is the tightest lower bound of $\Delta_{\mathbb{W}}(\cdot)$ when used $s \cdot \Delta_{1.0}(\cdot)$).

Proof: (a) Obviously,

$$s_1\delta_1(O_1,O_2) + s_2\delta_2(O_1,O_2) + \cdots + s_m\delta_m(O_1,O_2)$$
$$\leq w_1\delta_1(O_1,O_2) + w_2\delta_2(O_1,O_2) + \cdots + w_m\delta_m(O_1,O_2),$$

where $s_i \leq w_i, \forall w_i \in \mathbb{W}$. Since $\min_{j=1}^{m}(w_j) \leq w_i, \forall w_i \in \mathbb{W}$, we get

$$\sum_{i=1}^{m} \min_{j=1}^{m}(w_j)\delta_i(\cdot) \leq \sum_{i=1}^{m} w_i\delta_i(\cdot),$$

hence $\min_{j=1}^{m}(w_j) \sum_{i=1}^{m} \delta_i(\cdot) \leq \sum_{i=1}^{m} w_i\delta_i(\cdot)$.
(b) Consider a greater scaling constant s, i.e., $\exists w_{i_1}, s > w_{i_1}$. However, there can arise a situation where $\delta_{i_1}(O_1,O_2) \gg \delta_{i_j}(O_1,O_2), \delta_{i_j} \neq \delta_{i_1}, \forall j$, so multiplying by s could violate the lower-bounding property even if $s \ll w_{i_j}, \forall w_{i_j} \neq w_{i_1}$. ■

It is possible that tighter lower bounds may be found, but, on the other side, this one can be easily computed just by multiplying a (precomputed) distance $\Delta_{1.0}(\cdot)$ by s, so we avoid an evaluation of an expensive (even though possibly better) lower bound distance. Moreover, this would lost its meaning because in such case we can apply directly the basic filtering, since the parent filtering (which is always less effective) becomes equally (or more) expensive.

3.1 Similarity Queries

Lemmas 1 to 4 are directly applicable to range queries in M-tree, because the range query processing is provided by all the distances needed in conditions of the lemmas. In case of k-NN queries, the M-tree's branch-and-bound algorithm uses a heuristics which treats the k-NN search as a range search with the extension that the unknown query radius is determined dynamically during the query processing (it is continuously decreasing, such that it is in every moment an upper bound of the distance to the k-th neighbor). Thus, also in k-NN processing the lemmas are directly applicable.

Due to the lack of space we present just the modified range query algorithm (see Listing 1), however, the k-NN algorithm can be modified the same way (for both original query algorithms on M-tree we refer to [8]).

LISTING 1. *(modified range query algorithm in M-tree)*

```
QueryResult RangeQuery(Node N, RQuery (Q, ε_W), W)
{
  // if N is root then Δ_x(R,P)=Δ_x(P,Q)=0
  let P be the parent routing object of N
  let's denote Δ^lb_W(R,P) = min{W} · Δ_1.0(R,P)        // lemma 4
  if N is not a leaf then {
    for each rout(R) in N do {
      if Δ_W(P,Q) − Δ_1.0(R,P) ≤ r_1.0 + ε_W And        // lemma 2
        Δ^lb_W(R,P) − Δ_W(P,Q) ≤ r_1.0 + ε_W then {    // lemma 3
          compute Δ_W(R,Q)
          if Δ_W(R,Q) ≤ ε_W + r_1.0 then               // lemma 1
            RangeQuery(ptr(T(R)), (Q, ε_W), W)
        }
    } /* for each ... */
  } else {
    for each grnd(R) in N do {
      if Δ_W(P,Q) − Δ_1.0(R,P) ≤ ε_W And               // lemma 2
        Δ^lb_W(R,P) − Δ_W(P,Q) ≤ ε_W then {            // lemma 3
          compute Δ_W(R,Q)
          if Δ_W(R,Q) ≤ ε_W then
            add R to the query result
        }
    } /* for each ... */
  }
} /* RangeQuery */
```

4. M³-TREE

The tightness of upper/lower bounds of data region radii (and also to-parent distances) stored in the M-tree is heavily dependent on the actual weights vector $\mathbb{W}$. Obviously, if the weights are far from 1.0, the upper/lower bounds will be not very tight, reflecting in larger "volume" of data regions and leading to worse query performance.

In order to keep the search efficiency weight-independent, we introduce the *Multi-Metric M-tree* (M³-tree). The M³-tree extends the M-tree structure by storing the components of $\Delta_{1.0}$, i.e., the δ_i-based components of radii as well as of the to-parent distances are stored separately.

DEFINITION 2. (component-based distance notation)
Let $\Delta_{1.0}(\cdot,\cdot).comp(j)$ stands for the δ_j partial distance aggregated in $\Delta_{1.0}(\cdot,\cdot)$. Similarly, $r_{1.0}.comp(j)$ stands for the δ_j partial distance aggregated in $r_{1.0}$. When making arithmetic operations with component-based distances or radii,

the components are treated separately (for example, $9_{\langle 2,3,4\rangle} + 21_{\langle 6,7,8\rangle} = 30_{\langle 8,10,12\rangle}$). □

Having stored the individual distance components, we can construct a tighter covering radius upper bound to $r_{\mathbb{W}}$, and so reduce the volume of regions which delimit the data objects stored in subtrees of the M^3-tree. The following two lemmas show how the tighter radius upper bound can be constructed using the distance components.

LEMMA 5. *(component-based covering radius upper bound)* Let $O_i \in N$ be a set of objects, R be a center object. Then r_{cub} is an upper bound to $r_{\mathbb{W}}$, i.e.,

$$\max_{i=1}^{|N|} \{\Delta_{\mathbb{W}}(O_i, R)\} \leq \sum_{j=1}^{m} w_j \cdot \max_{i=1}^{|N|} \{\Delta_{1.0}(O_i, R).comp(j)\}\,.$$

$(= r_{\mathbb{W}}$ over $N)$ $\qquad$ $(= r_{cub}$ over $N)$

Proof: By expanding the statement of covering radius $r_{\mathbb{W}}$, together with propagating the w_j in r_{cub}, we obtain

$$\max_{i=1}^{|N|} \left\{ \sum_{j=1}^{m} w_j \cdot \Delta_{1.0}(O_i, R).comp(j) \right\} \leq$$

$$\leq \sum_{j=1}^{m} \max_{i=1}^{|N|} \{w_j \cdot \Delta_{1.0}(O_i, R).comp(j)\}$$

If we denote $w_j \cdot \Delta_{1.0}(O_i, R).comp(j)$ as $f(i,j)$, we get

$$\max_{i=1}^{|N|} \left\{ \sum_{j=1}^{m} f(i,j) \right\} \leq \sum_{j=1}^{m} \max_{i=1}^{|N|} \{f(i,j)\}\,,$$

which holds for any f, thus the proof is complete. ■

Note that a set N of objects $O_i \in \mathbb{S}$ is considered in Lemma 5 (objects in leaf nodes of M^3-tree). However, the lemma can be generalized also for set of regions (routing entries in non-leaf nodes) as follows.

LEMMA 6. *(recursive comp.-based covering radius upper bound)* Let $(R_i, r^i_{1.0}) \in \mathcal{N}$ be a set of regions (where $r^i_{1.0}$ is a covering radius upper bound of region centered in R_i), and P be a center object (of a super-region covering $\mathcal{N}$). Then

$$\max_{i=1}^{|\mathcal{N}|} \left\{ \Delta_{\mathbb{W}}(R_i, P) + r^i_{\mathbb{W}} \right\} \leq$$

$$\leq \sum_{j=1}^{m} w_j \cdot \max_{i=1}^{|\mathcal{N}|} \left\{ \Delta_{1.0}(R_i, P).comp(j) + r^i_{1.0}.comp(j) \right\}$$

$(= r_{\mathbb{W}}$ over $\mathcal{N})$ $\qquad$ $(= r_{cub}$ over $\mathcal{N})$

Proof: Follows from Lemma 5 and from the fact that $r^i_{1.0}$ is an upper bound to $r^i_{\mathbb{W}}$. ■

In most cases, r_{cub} is a tighter upper bound to $r_{\mathbb{W}}$ than $r_{1.0} = \max_{i=1}^{|N|}\{\Delta_{1.0}(O_i, R)\}$ (see Figure 5a). However, in some cases $r_{1.0}$ may be tighter than r_{cub} (see Figure 5b), and so we will use the smaller one, as defined below.

DEFINITION 3. (minimum comp.-based cov. rad. upper bound) The upper bound of the covering radius is defined as

$$r_u = \min\{r_{cub}, r_{1.0}\},$$

which is always a tighter upper bound than $r_{1.0}$. □

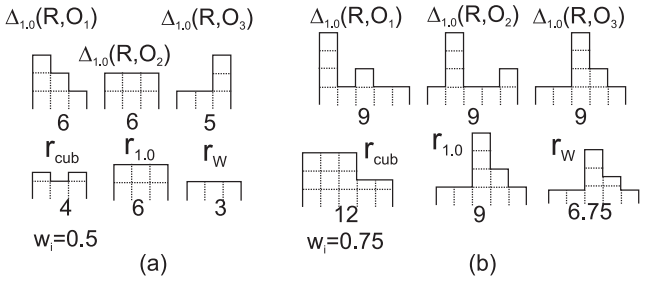


Figure 5: (a) $r_{\mathbb{W}} < r_{cub} < r_{1.0}$ **(b)** $r_{\mathbb{W}} < r_{1.0} < r_{cub}$.

With the covering radii upper bound r_u, we can reformulate the basic filtering into the context of M^3-tree.

LEMMA 7. *(component-wise basic filtering)* Let $(Q, \varepsilon_{\mathbb{W}})$ be a range query, where $\varepsilon_{\mathbb{W}}$ is a weighed query radius. Let (R, r_u) represents a data region (for r_u see Def. 3). If $\Delta_{\mathbb{W}}(R, Q) > \varepsilon_{\mathbb{W}} + r_u$, the data region is not relevant to the query and can be filtered out.

Proof: Follows immediately from Lemma 1 and the definition of r_u. ■

Like the covering radii upper bound, we can use the to-parent distance components to improve the parent filtering.

DEFINITION 4. (comp.-based to-parent dist. lower/upper bound) Let any $d^{ub}_P \geq \Delta_{\mathbb{W}}(R, P) = \sum_{i=1}^{m} w_i \cdot \delta_i(R, P)$ be called a *component-based to-parent distance upper bound*. Similarly, let any $d^{lb}_P \leq \Delta_{\mathbb{W}}(R, P) = \ldots$ be called a *component-based to-parent distance lower bound*. □

Definition 4 is not required for the following lemma (we can think about $\Delta_{\mathbb{W}}(R, P)$ instead of d^{ub}_P or d^{lb}_P), but we will find it useful in the subsequent structural description of the M^3-tree.

LEMMA 8. *(component-wise parent outer/inner filtering)* Let P be the parent object of a region (R, r_u). Then if

$$\Delta_{\mathbb{W}}(P, Q) - d^{ub}_P > r_u + \varepsilon_{\mathbb{W}} \quad \vee \quad d^{lb}_P - \Delta_{\mathbb{W}}(P, Q) > r_u + \varepsilon_{\mathbb{W}}$$

the region can be filtered out as non-relevant to the query $(Q, \varepsilon_{\mathbb{W}})$.

Proof: The proof is similar as in Lemmas 2, 3 – the only difference is the usage of r_u instead of $r_{1.0}$, but this is correct since r_u is (tighter but still) an upper bound to $r_{\mathbb{W}}$. ■

4.1 M^3-tree Structure

The structure of leaf/non-leaf node in M^3-tree is presented in Figure 6. In addition to the standard M-tree content of routing/ground entries, in entries of M^3-tree there are stored the components of covering radii and of the to-parent distances.

To keep the storage of radii/to-parent components as small as possible, these are not stored as floats, but as signatures (bitstrings of user-defined size). The value of each signature is interpreted as a scalar proportion of the respective partial radius (to-parent distance) with respect to the aggregate radius $r_{1.0}$ ($\Delta_{1.0}(R, P)$, resp.). In such a way, we can store each component by, e.g., 4, 8, 16, or another number of bits.

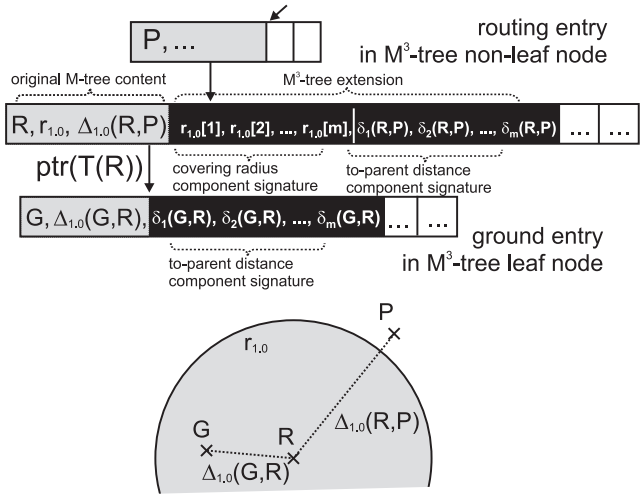


Figure 6: Structure of M^3-tree nodes.

The compact signature representation of radius/to-parent components is imprecise. Thus, in order to keep the query evaluation correct when using upper bound of a radius, we have to overestimate the value by usage of the largest possible float value represented by the respective partial signature. Similarly, in case of to-parent distances, the upper/lower bound is constructed by over/under-estimating the value (considering the largest/smallest possible value represented by signature).

In Lemma 8, we have distinguished between the upper bound d_P^{ub} and lower bound d_P^{lb} to $\Delta_{\mathbb{W}}(R,P)$, these were assumed ahead just with respect to the signature representation of $\Delta_{\mathbb{W}}(R,P)$.

4.2 M^3-tree Construction

The M^3-tree is constructed the same way as M-tree is, i.e., no weights are considered and the $\Delta_{1.0}$ is used for indexing as an ordinary metric. In addition, along with the aggregate value $\Delta_{1.0}(\cdot)$, the distance components $\Delta_{1.0}(\cdot).comp(i)$ are used to update the radii/to-parent distance representations.

When inserting an object, the covering radii components in routing entries must be updated after the aggregate covering radius $r_{1.0}$ is updated. When splitting a node (or inserting a ground entry into a leaf), the to-parent components are stored along with the aggregate to-parent distance $\Delta_{1.0}(R,P)$. When splitting, covering radii components of the two new routing entries are assembled by taking the maximum of covering radii components + the to-parent components of the entries being split.

It should be emphasized that no extra distance computations are needed for M^3-tree construction, the distance components are obtained as a "by-product" when computing $\Delta_{1.0}$. There is just a space overhead needed for storage of the component signatures.

4.3 Similarity Queries in M^3-tree

The M^3-tree-specific lemmas are used (in addition to the "old" lemmas) to discard more non-relevant subtrees when searching. In Listing 2 see the modified algorithm for range query processing. The k-NN algorithm can be adjusted in a similar way.

LISTING 2. *(range query algorithm in M^3-tree)*

```
QueryResult RangeQuery(Node N, RQuery (Q, ε_W), W)
{
  // if N is root then Δ_x(R,P)=Δ_x(P,Q)=0
  let P be the parent routing object of N
  let's denote Δ^lb_W(R,P) = min{W} · Δ_1.0(R,P)        // lemma 4
  if N is not a leaf then {
    for each rout(R) in N do {
      if Δ_W(P,Q) − Δ_1.0(R,P) ≤ r_1.0 + ε_W And        // lemma 2
        Δ^lb_W(R,P) − Δ_W(P,Q) ≤ r_1.0 + ε_W then {    // lemma 3
        if Δ_W(P,Q) − d^ub_P ≤ r_u + ε_W And
          d^lb_P − Δ_W(P,Q) ≤ r_u + ε_W then {         // lemma 8
            compute Δ_W(R,Q)
            if Δ_W(R,Q) ≤ ε_W + r_u then               // lemma 7
              RangeQuery(ptr(T(R)), (Q, ε_W), W)
        }
      }
    } /* for each ... */
  } else {
    for each grnd(R) in N do {
      if Δ_W(P,Q) − Δ_1.0(R,P) ≤ ε_W And               // lemma 2
        Δ^lb_W(R,P) − Δ_W(P,Q) ≤ ε_W then {           // lemma 3
        if Δ_W(P,Q) − d^ub_P ≤ ε_W And
          d^lb_P − Δ_W(P,Q) ≤ ε_W then {               // lemma 8
            compute Δ_W(R,Q)
            if Δ_W(R,Q) ≤ ε_W then
              add R to the query result
        }
      }
    } /* for each ... */
  }
} /* RangeQuery */
```

5. EXPERIMENTAL EVALUATION

We performed an experimental evaluation of the efficiency of the M^3-tree using two real datasets.

5.1 The Testbed

The first dataset is the *Corel image features*, available at the *UCI KDD Archive* [10]. This database consists of 89-D feature vectors representing 65,615 Corel images and 1,000 query images (not included in the dataset). Each feature vector consisted of 4 subvectors (of dimensions 32, 9, 16, 32), representing color histogram, color moments, texture, and layout histogram. As partial distances aggregated in $\Delta_{\mathbb{W}}$, the L_1 distance was used, i.e., $\delta_i = L_1,\ i \in \{1,2,3,4\}$.

A set of query weight vectors (*weights interval*) was independently constructed as vectors of random values from 0.2-wide intervals, starting at $w = 0.1$, increasing by 0.1. Only one such set of query weight vectors was constructed:

$$\{W_{0.1} = \langle 0.21, 0.21, 0.27, 0.11\rangle,\ W_{0.2} = \langle 0.40, 0.33, 0.40, 0.39\rangle,\\ W_{0.3} = \langle 0.46, 0.40, 0.40, 0.42\rangle,\ W_{0.4} = \langle 0.53, 0.42, 0.58, 0.45\rangle,\\ W_{0.5} = \langle 0.55, 0.53, 0.67, 0.60\rangle,\ W_{0.6} = \langle 0.75, 0.76, 0.66, 0.61\rangle,\\ W_{0.7} = \langle 0.88, 0.86, 0.70, 0.83\rangle,\ W_{0.8} = \langle 0.85, 0.82, 0.95, 0.88\rangle\}.$$

Another set of query weight vectors (*weights group*) was created, consisting of 20 generated weight vectors such that: (a) one of the weights is always 1.0 (b) the lowest weight is a random number in $[w, w+0.1]$ (c) the rest of weights (i.e., the last two) are random numbers in $[w, 1.0]$.

The second dataset is a *3D models database*, which contains 1,838 3D objects that we collected from the Internet[1].

[1]Konstanz 3D model search engine.
`http://merkur01.inf.uni-konstanz.de/CCCC/`

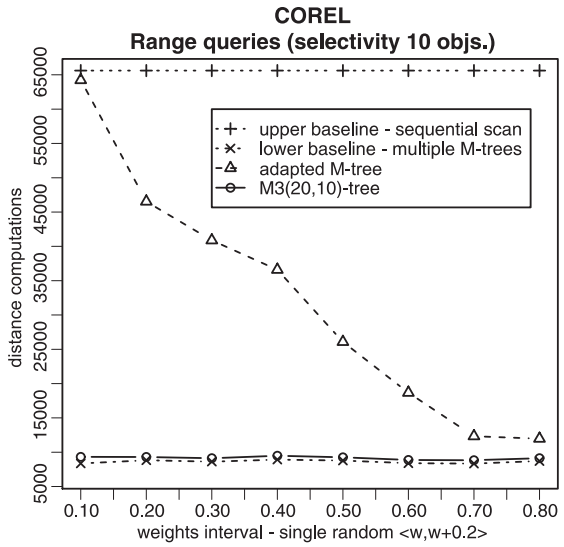


Figure 7: Corel image features: Range queries varying weights interval.

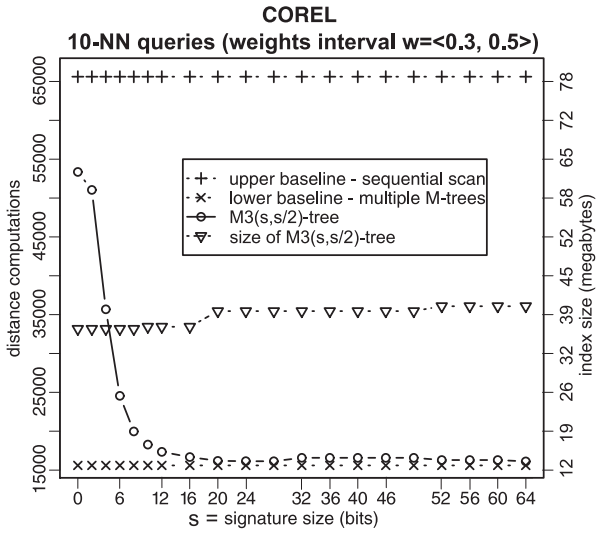


Figure 8: Corel image features: 10-NN queries varying signature size.

From this set, 472 objects were used as a query objects and the rest of 1,366 objects were indexed.

For this dataset, we computed 8 different feature vectors for 3D models, which include volumetric descriptors (16-D voxel, 8-D 3DDFT) and image-based descriptors (16-D depth buffer, 12-D complex, 12-D rays with spherical harmonics, 8-D silhouette, 6-D shading, and 6-D ray-based). For a detailed explanation of the implemented 3D feature vectors, see [4]. We performed a PCA-based dimensionality reduction of the original 3D feature vectors [4] and we kept between 6 and 16 principal axes for each feature vector, resulting in an aggregate dimensionality of 84-D. For this dataset, we also used the L_1 distance as metric function for all 3D feature vectors.

5.1.1 Weights for 3D Models

We implemented a query processor based on the *entropy impurity method* [3] to compute the dynamic weights for each 3D feature vector. This method uses a reference dataset that is classified in object classes (in our case, we used the classified subset of the 3D models database). For each feature vector, a similarity query is performed on the reference dataset. Then, the entropy impurity is computed looking at the model classes of the first t retrieved objects: It is equal to zero if all the first t retrieved objects belong to the same model class, and it has a maximum value if each of the t object belongs to a different model class. Let P_{ω_j} denote the fraction of the first t retrieved objects that belong to model class ω_j. The entropy impurity of feature vector i

$$impurity(i) = -\sum_{j=1}^{|\#classes|} \begin{cases} P_{\omega_j} \cdot \log_2(P_{\omega_j}) & \text{if } P_{\omega_j} > 0 \\ 0 & \text{otherwise} \end{cases}$$

The weight value for feature vector i (i.e., the weight for the i^{th} metric in the combination) is computed as the inverse of the entropy impurity plus one (to avoid dividing by zero), i.e., $w_i = \frac{1}{1+entropyImpurity(i)}$. (We used $t = 3$ for our experiments [3].)

5.1.2 Indexing

Besides the adapted M-tree index and the M^3-tree used in all experiments (which were the subjects of evaluation), we have used the sequential search as the *upper baseline*. We have also created multiple M-tree indexes using the query distance as the index distance, i.e., for each particular $\mathbb{W}$ a standard M-tree was created using the query distance $\Delta_{\mathbb{W}}$. These $\mathbb{W}$-dependent M-trees served us as a *lower baseline*, i.e., they show the most efficient query processing (related to M-trees).

In the figures, we use "M3(x,y)-tree" to denote a single M^3-tree index, where the routing entries consist of m x-bit signatures for covering radii components and m x-bit signatures for the to-parent distance components (i.e., $2m \cdot x$ bits in each routing entry), and the ground entry consists of m y-bit signatures for the to-parent distance components (i.e., $m \cdot y$ bits on each ground entry). It follows that "M3(0,0)-tree" is an ordinary (but adapted) M-tree index.

5.2 Experimental Results

Figure 7 presents range query processing on the Corel image features, where the M^3-tree and M-tree indices were slimmed [13] (the rest of Corel experiments was performed on non-slimmed indices). The figure shows the number of distance computations needed to perform range queries (query radius calculated to have an average selectivity of 10 objects) for the different weight intervals. It clearly shows that the M^3-tree outperforms the adapted M-tree in the whole range of weight intervals, especially if the weights are low. This indicates that the lower bound to $\Delta_{\mathbb{W}}$ proposed in Lemma 4 is too loose if there is a weight with a value close to 0.

Figure 8 shows the influence of the signature size (meaning size of distance/radius components) of the M^3-tree on the efficiency of 10-NN queries. The curve denoted as "size of M^3-index" belongs to the right-hand y-axis, and shows the increase of M^3-index filesize with growing signature size (for a comparison, the sequential file size was 22.3 MB). We found that, even by using a small amount of bits per partial

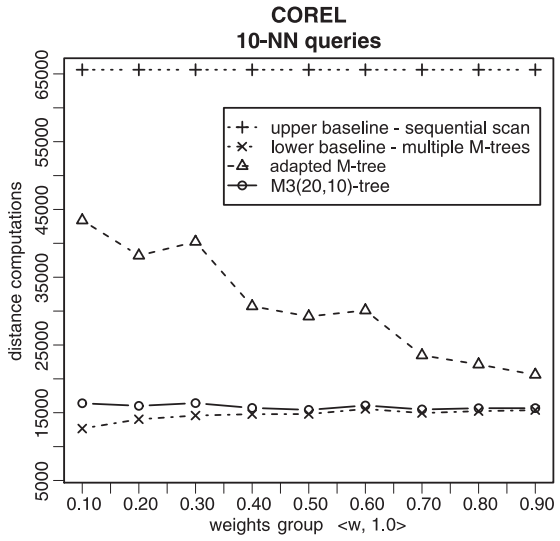


Figure 9: Corel image features: 10-NN queries varying weights group.

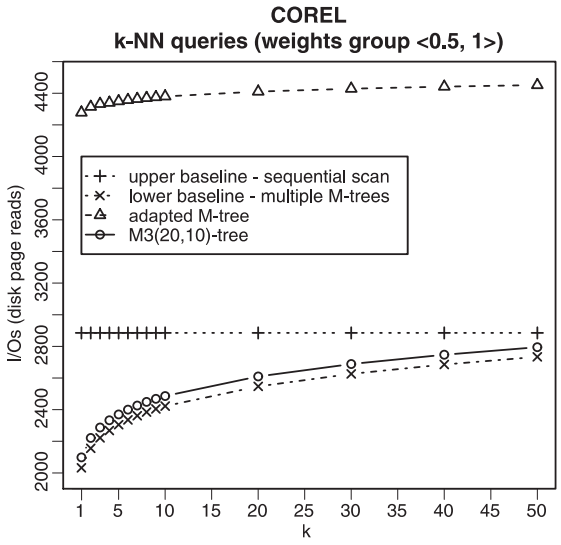


Figure 10: Corel image features: k-NN queries with fixed weights group.

distance, the proposed index structure can achieve a very good efficiency performance. Indeed, the efficiency of the M^3-tree quickly approaches the efficiency of having multiples M-trees, one for each possible combination of metrics.

Figure 9 presents distance computations needed to perform 10-NN queries, but now using the weights groups. Figure 10 shows the I/O cost (the unit of I/O was a single 8kB page read) while performing k-NN queries ($1 \leq k \leq 50$) with a single fixed weights group. The results are similar to those previously presented (M^3-tree outperforms the adapted M-tree in distance computations and disk page accesses).

Figure 11 presents the efficiency of k-NN querying (varying k) for the 3D models database (we have used slimmed indices for all "3D experiments"). In Figure 12 see the effect of increasing signature size with 10-NN queries on retrieval efficiency as well as on the index size (the sequential file size was 450kB). With this database, the experimental results also show that the M^3-tree is more close in efficiency to the lower baseline than the adapted M-tree. Moreover, the adapted M-tree turned out to be slower than a sequential scan. On the other side, we must realize the available 3D database was very small – we expect that by using a larger database the M^3-tree as well as the adapted M-tree will achieve a considerably better efficiency.

6. CONCLUSIONS

In this paper, we presented two index structures specially designed for dynamic multi-metric spaces. In these spaces, the metric function used to perform the similarity query (the so-called query distance) corresponds to a dynamic combination of metrics, thus the metric function may change on each performed query. The index is built using a fixed combined metric (the index distance) that is an upper-bounding distance function of the query distance.

Firstly, we described an adapted M-tree for multi-metric spaces. We formally proved that the usual filtering criteria holds on the adapted M-tree, independently of the used query distance. Secondly, we depicted the M^3-tree, a further adaption of the original M-tree with considerably better performance than the adapted M-tree. The M^3-tree store partial distances (one for each metric function belonging to the combination) to dynamically estimate, for each performed query, the new covering radius of the space regions and the new distances from parent to children nodes.

Our work differs to previous related work in the sense that: (a) We provide a dynamic index structure for multi-metric spaces (b) The adapted M-tree use a lower bound of the query distance to apply some of the discarding criteria. The M^3-tree computes a tight approximation of this distance (using the stored partial distances), thus providing a better filtering.

The experimental results clearly show that a single M^3-tree index is almost as good as if we have infinitely many M-trees indexes at our disposal (M-trees built for every possible vector of query weights).

6.1 Future Work

We plan to adapt the *PM-tree* [14], a MAM that combines the M-tree with the pivot-based approach, for the multi-metric space case. For this purpose, we will merge the techniques presented in this paper and the ones described in [5] (pivot-based index for multi-metrics). We expect that, by combining all these technique in one index structure, we will be able to further improve the efficiency of the M^3-tree.

Although we do not expect that the QIC-M-tree outperforms the M^3-tree, considering that the experimental performance of our proposed index was very close to the lower baseline (multiple standard M-trees), we also plan to perform an experimental comparison of the efficiency of both index structures.

An important subject for future research is the *"number of metrics curse"* (in comparison with the "dimensionality curse" in multi-dimensional spaces [1]). We do not know at the moment whether it is a curse or not, but we expect that with increasing number of metrics the efficiency of the M^3-tree will decrease.

We would also like to compare the effectiveness of multi-metric approach with various non-metric approaches [12].

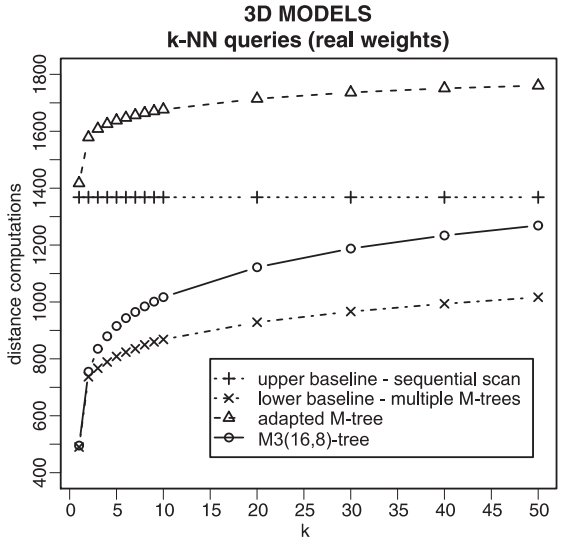


Figure 11: 3D models: k-NN queries.

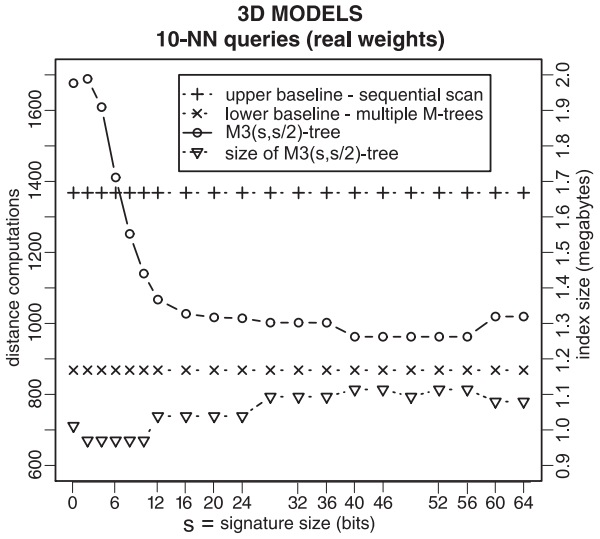


Figure 12: 3D models: 10-NN varying signature size.

Because the multi-metrics allow dynamic weights at query time, there is a possibility of much rich similarity measuring and retrieval, which is currently provided by non-metric measures (especially in multimedia retrieval).

Acknowledgments

This research has been partially supported by Czech grants GAČR 201/05/P036 and Information Society 1ET100300419 (second author). The first author is on leave from the Department of Computer Science, University of Chile.

7. REFERENCES

[1] C. Böhm, S. Berchtold, and D. Keim. Searching in high-dimensional spaces: Index structures for improving the performance of multimedia databases. *ACM Computing Surveys*, 33(3):322–373, 2001.

[2] B. Bustos, D. Keim, D. Saupe, T. Schreck, and D. Vranić. Automatic selection and combination of descriptors for effective 3D similarity search. In *Proc. IEEE International Workshop on Multimedia Content-based Analysis and Retrieval (MCBAR'04)*, pages 514–521. IEEE Computer Society, 2004.

[3] B. Bustos, D. Keim, D. Saupe, T. Schreck, and D. Vranić. Using entropy impurity for improved 3D object similarity search. In *Proc. IEEE International Conference on Multimedia and Expo (ICME'04)*, pages 1303–1306. IEEE, 2004.

[4] B. Bustos, D. Keim, D. Saupe, T. Schreck, and D. Vranić. An experimental effectiveness comparison of methods for 3D similarity search. *Intl. Journal on Digital Libraries*, 6(1):39–54, 2006.

[5] B. Bustos, D. Keim, and T. Schreck. A pivot-based index structure for combination of feature vectors. In *Proc. 20th Annual ACM Symposium on Applied Computing, Multimedia and Visualization Track (SAC-MV'05)*, pages 1180–1184. ACM Press, 2005.

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

[7] P. Ciaccia and M. Patella. Searching in metric spaces with user-defined and approximate distances. *ACM Transactions on Database Systems*, 27(4):398–437, 2002.

[8] P. Ciaccia, M. Patella, and P. Zezula. M-tree: An efficient access method for similarity search in metric spaces. In *Proc. 23rd Conference on Very Large Databases (VLDB'97)*, pages 426–435. Morgan Kaufmann, 1997.

[9] A. de Vries, N. Mamoulis, N. Nes, and M. Kersten. Efficient k-NN search on vertically decomposed data. In *Proc. ACM International Conference on Management of Data (SIGMOD'02)*, pages 322–333. ACM Press, 2002.

[10] S. Hettich and S. Bay. The UCI KDD archive [http://kdd.ics.uci.edu], 1999.

[11] D. Keim. Efficient geometry-based similarity search of 3D spatial databases. In *Proc. ACM International Conference on Management of Data (SIGMOD'99)*, pages 419–430. ACM Press, 1999.

[12] T. Skopal. On fast non-metric similarity search by metric access methods. In *Proc. 10th International Conference on Extending Database Technology (EDBT'06)*, LNCS 3896, pages 718–736. Springer, 2006.

[13] T. Skopal, J. Pokorný, M. Krátký, and V. Snášel. Revisiting M-tree building principles. In *Proc. 7th East European Conference on Advances in Databases and Information Systems (ADBIS'03)*, LNCS 2798, pages 148–162. Springer, 2003.

[14] T. Skopal, J. Pokorný, and V. Snášel. Nearest neighbours search using the PM-tree. In *Proc. 10th International Conference on Database Systems for Advanced Applications (DASFAA'05)*, LNCS 3453, pages 803–815. Springer, 2005.

[15] P. Zezula, G. Amato, V. Dohnal, and M. Batko. *Similarity Search: The Metric Space Approach (Advances in Database Systems)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2005.

Chapter 6

On Fast Non-Metric Similarity Search by Metric Access Methods

Tomáš Skopal

On Fast Non-Metric Similarity Search by Metric Access Methods [41]

Regular paper at the 10th International Conference on Extending Database Technology (EDBT 2006), Munich, Germany, March 2006

Published in the *Lecture Notes in Computer Science* (LNCS), vol. 3896, pages 718–736, *Springer-Verlag*, ISSN 0302-9743, ISBN 978-3-540-32960-2

On Fast Non-Metric Similarity Search by Metric Access Methods

Tomáš Skopal

Charles University in Prague, FMP, Department of Software Engineering,
Malostranské nám. 25, 118 00 Prague 1, Czech Republic
tomas@skopal.net

Abstract. The retrieval of objects from a multimedia database employs a measure which defines a similarity score for every pair of objects. The measure should *effectively* follow the nature of similarity, hence, it should not be limited by the triangular inequality, regarded as a restriction in similarity modeling. On the other hand, the retrieval should be as *efficient* (or fast) as possible. The measure is thus often restricted to a *metric*, because then the search can be handled by *metric access methods* (MAMs). In this paper we propose a general method of non-metric search by MAMs. We show the triangular inequality can be enforced for any *semimetric* (reflexive, non-negative and symmetric measure), resulting in a metric that preserves the original similarity orderings (retrieval effectiveness). We propose the *TriGen* algorithm for turning any black-box semimetric into (approximated) metric, just by use of distance distribution in a fraction of the database. The algorithm finds such a metric for which the retrieval efficiency is maximized, considering any MAM.

1 Introduction

In multimedia databases the semantics of data objects is defined loosely, while for querying such objects we usually need a similarity measure standing for a judging mechanism of how much are two objects similar. We can observe two particular research directions in the area of content-based multimedia retrieval, however, both are essential. The first one follows the subject of retrieval *effectiveness*, where the goal is to achieve query results complying with the user's expectations (measured by the *precision* and *recall* scores). As the effectiveness is obviously dependent on the semantics of similarity measure, we require the possibilities of similarity measuring as rich as possible, thus, the measure should not be limited by properties regarded as restrictive for similarity modeling.

Following the second direction, the retrieval should be as *efficient* (or fast) as possible, because the number of objects in a database can be large and the similarity scores are often expensive to compute. Therefore, the similarity measure is often restricted by metric properties, so that retrieval can be realized by metric access methods. Here we have reached the point. The "effectiveness researchers" claim the metric properties, especially the triangular inequality, are too restrictive. However, the "efficiency researchers" reply the triangular inequality is the most powerful tool to keep the search in a database efficient.

In this paper we show the triangular inequality is not restrictive for similarity search, since every semimetric can be modified into a suitable metric and used for the search instead. Such a metric can be constructed even automatically, just with a partial information about distance distribution in the database.

1.1 Preliminaries

Let a multimedia object $\mathcal{O}$ be modeled by a *model object* $O \in \mathbb{U}$, where $\mathbb{U}$ is a model universe. A multimedia database is then represented by a dataset $\mathbb{S} \subset \mathbb{U}$.

Definition 1 (similarity & dissimilarity measure)
Let $s : \mathbb{U} \times \mathbb{U} \mapsto \mathbb{R}$ be a *similarity measure*, where $s(O_i, O_j)$ is considered as a similarity score of objects $\mathcal{O}_i$ and $\mathcal{O}_j$. In many cases it is more suitable to use a *dissimilarity measure* $d : \mathbb{U} \times \mathbb{U} \mapsto \mathbb{R}$ equivalent to a similarity measure s as $s(Q, O_i) > s(Q, O_j) \Leftrightarrow d(Q, O_i) < d(Q, O_j)$. A dissimilarity measure assigns a higher score (or *distance*) to less similar objects, and vice versa.

The measures often satisfy some of the metric properties. The *reflexivity* ($d(O_i, O_j) = 0 \Leftrightarrow O_i = O_j$) permits the zero distance just for identical objects. Both reflexivity and *non-negativity* ($d(O_i, O_j) \geq 0$) guarantee every two distinct objects are somehow positively dissimilar. If d satisfies reflexivity, non-negativity and *symmetry* ($d(O_i, O_j) = d(O_j, O_i)$), we call d a *semimetric*. Finally, if a semimetric d satisfies also the *triangular inequality* ($d(O_i, O_j) + d(O_j, O_k) \geq d(O_i, O_k)$), we call d a *metric* (or metric distance). This inequality is a kind of transitivity property; it says if O_i, O_j and O_j, O_k are similar, then also O_i, O_k are similar. If there is an upper bound d^+ such that $d : \mathbb{U} \times \mathbb{U} \mapsto \langle 0, d^+ \rangle$, we call d a *bounded metric*. The pair $\mathcal{M} = (\mathbb{U}, d)$ is called a (bounded) *metric space*. □

Definition 2 (triangular triplet)
A triplet (a, b, c), $a, b, c \geq 0$, $a + b \geq c$, $b + c \geq a$, $a + c \geq b$, is called a *triangular triplet*. Let (a, b, c) be ordered as $a \leq b \leq c$, then (a, b, c) is an *ordered triplet*. If $a \leq b \leq c$ and $a + b \geq c$, then (a, b, c) is called an *ordered triangular triplet*. □

A metric d generates just the (ordered) triangular triplets, i.e. $\forall O_i, O_j, O_k \in \mathbb{U}$, $(d(O_i, O_j),\ d(O_j, O_k),\ d(O_i, O_k))$ is triangular triplet. Conversely, if a measure generates just the triangular triplets, then it satisfies the triangular inequality.

1.2 Similarity Queries

In the following we consider the *query-by-example* concept; we look for objects similar to a query object $Q \in \mathbb{U}$ (Q is derived from an example object). Necessary to the query-by-example retrieval is a notion of *similarity ordering*, where the objects $O_i \in \mathbb{S}$ are ordered according to the distances to Q. For a particular query there is specified a portion of the ordering returned as the query result. The *range query* and the *k nearest neighbors (k-NN) query* are the most popular ones. A range query (Q, r_Q) selects objects from the similarity ordering for which $d(Q, O_i) \leq r_Q$, where $r_Q \geq 0$ is a distance threshold (or query radius). A k-NN query (Q, k) selects the k most similar objects (first k objects in the ordering).

1.3 Metric Access Methods

Once we have to search according to a metric d, we can use the *metric access methods* (MAMs) [5], which organize (or index) a given dataset $\mathbb{S}$ in a way that similarity queries can be processed efficiently by use of a *metric index*, hence, without the need of searching the entire dataset $\mathbb{S}$. The main principle behind all MAMs is a utilization of the triangular inequality (satisfied by any metric), due to which MAMs can organize the objects of $\mathbb{S}$ in distinct classes. When a query is processed, only the candidate classes are searched (such classes which overlap the query), so the searching becomes more efficient (see Figure 1a).

In addition to the number of distance computations $d(\cdot, \cdot)$ needed (the *computation costs*), the retrieval efficiency is affected also by the *I/O costs*. To minimize the search costs, i.e. to increase the retrieval efficiency, there were developed many MAMs for different scenarios (e.g. designed to secondary storage or main memory management). Besides others we name *M-tree*, *vp-tree*, *LAESA* (we refer to a survey [5]), or more recent ones, *D-index* [9] and *PM-tree* [27].

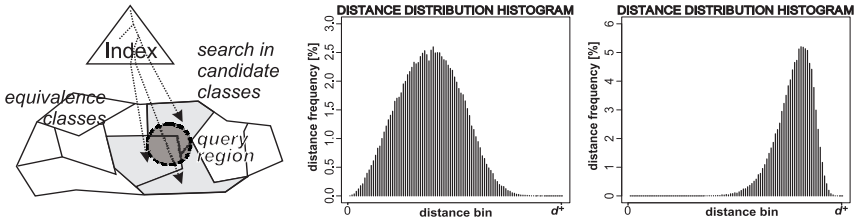


Fig. 1. Search by MAMs (a), DDHs indicating low (b) and high (c) intrinsic dim.

1.4 Intrinsic Dimensionality

The metric access methods are not successful for all datasets and all metrics; the retrieval efficiency is heavily affected by *distance distribution* in the dataset. Given a dataset $\mathbb{S}$ and a metric d, the efficiency limits of any MAM are indicated by the *intrinsic dimensionality*, defined as $\rho(\mathbb{S}, d) = \frac{\mu^2}{2\sigma^2}$, where μ and σ^2 are the mean and the variance of the distance distribution in $\mathbb{S}$ (proposed in [4]). In Figures 1b,c see an example of distance distribution histograms (DDHs) indicating low ($\rho = 3.61$) and high ($\rho = 42.35$) intrinsic dimensionalities.

The intrinsic dimensionality is low if there exist tight clusters of objects. Conversely, if all the indexed objects are almost equally distant, then intrinsic dimensionality is high, which means the dataset is poorly intrinsically structured. A high ρ value says that many (even all) of MAM's classes created on $\mathbb{S}$ are overlapped by every possible query, so that processing deteriorates to sequential search in all the classes. The problem of high intrinsic dimensionality is, in fact, a generalization of the *curse of dimensionality* [31, 4] into metric spaces.

1.5 Theories of Similarity Modeling

The metric properties have been argued against as restrictive in similarity modeling [25, 28]. In particular, the reflexivity and non-negativity have been refuted

[21, 28] by claiming that different objects could be differently self-similar. Nevertheless, these are the less problematic properties. The symmetry was questioned by showing that a prototypical object can be less similar to an indistinct one than vice versa [23, 24]. The triangular inequality is the most attacked property [2, 29]. Some theories point out the similarity has not to be transitive. Demonstrated by the well-known example, a man is similar to a centaur, the centaur is similar to a horse, but the man is completely dissimilar to the horse.

1.6 Examples of Non-Metric Measures

In the following we name several dissimilarity measures of two kinds, proved to be effective in similarity search, but which violate the triangular inequality.

Robust Measures. A *robust* measure is resistant to outliers – anomalous or "noisy" objects. For example, various *k-median distances* measure the kth most similar portion of the compared objects. Generally, a k-median distance d is of form $d(O_1, O_2) = k\text{–}med(\delta_1(O_1, O_2), \delta_2(O_1, O_2), \ldots, \delta_n(O_1, O_2))$, where $\delta_i(O_1, O_2)$ is a distance between O_1 and O_2, considering the ith portion of the objects. Among the partial distances δ_i the *k–med* operator returns the kth smallest value. As a special k-median distance derived from the Hausdorff metric, the *partial Hausdorff distance* (pHD) has been proposed for shape-based image retrieval [17]. Given two sets $\mathcal{S}_1, \mathcal{S}_2$ of points (e.g. two polygons), the partial Hausdorff distance uses $\delta_i(\mathcal{S}_1, \mathcal{S}_2) = dNP(\mathcal{S}_1^i, \mathcal{S}_2)$, where dNP is the Euclidean (L_2) distance of the ith point in $\mathcal{S}_1$ to the nearest point in $\mathcal{S}_2$. To keep the distance symmetric, pHD is the maximum, i.e. $pHD(\mathcal{S}_1, \mathcal{S}_2) = max(d(\mathcal{S}_1, \mathcal{S}_2), d(\mathcal{S}_2, \mathcal{S}_1))$. Similar to pHD is another modification of Hausdorff metric, used for face detection [20], where the average of dNP distances is considered, instead of k-median.

The *time warping distance* for sequence aligning has been used in time series retrieval [33], and even in shape retrieval [3]. The *fractional L_p distances* [1] have been suggested for robust image matching [10] and retrieval [16]. Unlike classic L_p metrics ($L_p(u, v) = (\sum_{i=1}^{n} |u_i - v_i|^p)^{\frac{1}{p}}$, $p \geq 1$), the fractional L_p distances use $0 < p < 1$, which allows us to inhibit extreme differences in coordinate values.

Complex Measures. In the real world, the algorithms for similarity measuring are often complex, even adaptive or learning. Moreover, they are often implemented by heuristic algorithms which combine several measuring strategies. Obviously, an analytic enforcement of triangular inequality for such measures can be simply too difficult. The COSIMIR method [22] uses a back-propagation neural network for supervised similarity modeling and retrieval. Given two vectors $u, v \in \mathbb{S}$, the distance between u and v is computed by activation of three-layer network. This approach allows to train the similarity measure by means of user-assessed pairs of objects. Another example of complex measure can be the *matching by deformable templates* [19], utilized in handwritten digits recognition. Two digits are compared by deforming the contour of one to fit the edges of the other. The distance is derived from the amount of deformation needed, the goodness of edges fit, and the interior overlap between the deformed shapes.

1.7 Paper Contributions

In this paper we present a general approach to efficient and effective non-metric search by metric access methods. First, we show that every semimetric can be non-trivially turned into metric and used for similarity search by MAMs. To achieve this goal, we modify the semimetric by a suitable *triangle-generating modifier*. In consequence, we also claim the triangular inequality is completely unrestrictive with respect to the effectiveness of similarity search. Second, we propose the *TriGen* algorithm for automatic conversion of any "black-box" semimetric (i.e. semimetric given in a non-analytic form) into (approximated) metric, such that intrinsic dimensionality of the indexed dataset is kept as low as possible. The optimal triangle-generating modifier is found by use of predefined base modifiers and by use of distance distribution in a (small) portion of the dataset.

2 Related Work

The simplest approach to non-metric similarity search is the *sequential search* of the entire dataset. The query object is compared against every object in the dataset, resulting in a similarity ordering which is used for the query evaluation. The sequential search often provides a baseline for other retrieval methods.

2.1 Mapping Methods

The non-metric search can be indirectly carried out by various *mapping methods* [11, 15] (e.g. MDS, FastMap, MetricMap, SparseMap). The dataset $\mathbb{S}$ is embedded into a vector space $(\mathbb{R}^k, \delta)$ by a mapping $F : \mathbb{S} \mapsto \mathbb{R}^k$, where the distances $d(\cdot, \cdot)$ are (approximately) preserved by a cheap vector metric δ (often the L_2 distance). Sometimes the mapping F is required to be *contractive*, i.e. $\delta(F(O_i), F(O_j)) \leq d(O_i, O_j)$, which allows to filter out some irrelevant objects using δ, but some other irrelevant objects, called *false hits*, must be re-filtered by d (see e.g. [12]). The mapped vectors can be indexed/retrieved by any MAM.

To say the drawbacks, the mapping methods are expensive, while the distances are preserved only approximately, which leads to *false dismissals* (i.e. to relevant objects being not retrieved). The contractive methods eliminate the false dismissals but suffer from a great number of false hits (especially when k is low), which leads to lower retrieval efficiency. In most cases the methods need to process the dataset in a batch, so they are suitable for static MAMs only.

2.2 Lower-Bounding Metrics

To support similarity search by a non-metric distance d_Q, the QIC-M-tree [6] has been proposed as an extension of the M-tree (the key idea is applicable also to other MAMs). The M-tree index is built by use of an index distance d_I, which is a metric *lower-bounding* the query distance d_Q (up to a scaling constant $S_{I \to Q}$), i.e. $d_I(O_i, O_j) \leq S_{I \to Q}\ d_Q(O_i, O_j), \forall O_i, O_j \in \mathbb{U}$. As d_I lower-bounds d_Q, a query

can be partially processed by d_I (which, moreover, could be much cheaper than d_Q), such that many irrelevant classes of objects (subtrees in M-tree) are filtered out. All objects in the non-filtered classes are compared against Q using d_Q. Actually, this approach is similar to the usage of contractive mapping methods (d_I is an analogy to δ), but here the objects generally need not to be mapped into a vector space. However, this approach has two major limitations. First, for a given non-metric distance d_Q there was not proposed a general way how to find the metric d_I. Although d_I could be found "manually" for a particular d_Q (as in [3]), this is not easy for d_Q given as a black box (an algorithmically described one). Second, the lower-bounding metric should be as tight approximation of d_Q as possible, because this "tightness" heavily affects the intrinsic dimensionality, the number of MAMs' filtered classes, and so the retrieval efficiency.

2.3 Classification

Quite many attempts to non-metric nearest neighbor (NN) search have been tried out in the classification area. Let us recall the basic three steps of classification. First, the dataset is organized in classes of similar objects (by user annotation or clustering). Then, for each class a description consisting of the most representative object(s) is created; this is achieved by *condensing* [14] or *editing* [32] algorithms. Third, the NN search is accomplished as a classification of the query object. Such a class is searched, to which the query object is "nearest", since there is an assumption the nearest neighbor is located in the "nearest class". For non-metric classification there have been proposed methods enhancing the description of classes (step 2). In particular, condensing algorithms producing *atypical points* [13] or *correlated points* [18] have been successfully applied.

The drawbacks of classification-based methods reside in static indexing and limited scalability, while the querying is restricted just to approximate (k-)NN.

3 Turning Semimetric into Metric

In our approach, a given dissimilarity measure is turned into a metric, so that MAMs can be directly used for the search. This idea could seem to disclaim the results of similarity theories (mentioned in Section 1.5), however, we must realize the task of **similarity search** employs only a limited modality of **similarity modeling**. In fact, in similarity search we just need to order the dataset objects according to a single query object and pick the most similar ones. Clearly, if we find a metric for which such similarity orderings are the same as for the original dissimilarity measure, we can safely use the metric instead of the measure.

3.1 Assumptions

We assume d satisfies reflexivity and non-negativity but, as we have mentioned in Section 1.5, these are the less restrictive properties and can be handled easily; e.g. the *non-negativity* is satisfied by a shift of the distances, while for the *reflexivity*

property we require every two non-identical objects are at least d^--distant (d^- is some positive distance lower bound). Furthermore, searching by an *asymmetric measure* δ could be partially provided by a symmetric measure d, e.g. $d(O_i, O_j) = min\{\delta(O_i, O_j), \delta(O_j, O_i)\}$. Using the symmetric measure some irrelevant objects can be filtered out, while the original asymmetric measure δ is then used to rank the remaining non-filtered objects. In the following we assume the measure d is a bounded semimetric, nevertheless, this assumption is introduced just for clarity of the following presentation. Finally, as d is bounded by d^+, we can further simplify the semimetric such that it assigns distances from $\langle 0, 1\rangle$. This can be achieved simply by scaling the original value $d(O_i, O_j)$ to $d(O_i, O_j)/d^+$. The same way a range query radius r_Q must be scaled to r_Q/d^+, when searching.

3.2 Similarity-Preserving Modifications

Based on the assumptions, the only property we have to solve is the triangular inequality. To do so, we apply some special modifying function on the semimetric, such that the original similarity orderings are preserved.

Definition 3 (similarity-preserving modification)
Given a measure d, we call $d^f(O_i, O_j) = f(d(O_i, O_j))$ a *similarity-preserving modification of* d (or *SP-modification*), where f, called the *similarity-preserving modifier* (or *SP-modifier*), is a strictly increasing function for which $f(0) = 0$. Again, for clarity reasons we assume f is bounded, i.e. $f : \langle 0, 1\rangle \mapsto \langle 0, 1\rangle$. □

Definition 4 (similarity ordering)
We define $SimOrder_d : \mathbb{U} \mapsto 2^{\mathbb{U}\times\mathbb{U}}, \forall O_i, O_j, Q \in \mathbb{U}$ as $\langle O_i, O_j\rangle \in SimOrder_d(Q) \Leftrightarrow d(Q, O_i) < d(Q, O_j)$, i.e. $SimOrder_d$ orders objects by their distances to Q. □

Lemma 1
Given a metric d and any d^f, then $SimOrder_d(Q) = SimOrder_{d^f}(Q), \forall Q \in \mathbb{U}$.

Proof: As f is increasing, then $\forall Q, O_i, O_j \in \mathbb{U}$ it follows that $d(Q, O_i) > d(Q, O_j) \Leftrightarrow f(d(Q, O_i)) > f(d(Q, O_j))$. ■

In other words, every SP-modification d^f preserves the similarity orderings generated by d. Consequently, if a query is processed sequentially (by comparing all objects in $\mathbb{S}$ to the query object Q), then it does not matter if we use either d or any d^f, because both ways induce the same similarity orderings. Naturally, the radius r_Q of a range query must be modified to $f(r_Q)$, when searching by d^f.

3.3 Triangle-Generating Modifiers

To obtain a modification forcing a semimetric to satisfy the triangular inequality, we have to use some special SP-modifiers based on metric-preserving functions.

Definition 5 (metric-preserving SP-modifier)

A SP-modifier f is *metric-preserving* if for every metric d the SP-modification d^f preserves the triangular inequality, i.e. d^f is also metric. Such a SP-modifier must be additionally *subadditive* $(f(x) + f(y) \geq f(x + y), \forall x, y)$. □

Lemma 2

(a) Every concave SP-modifier f is metric-preserving.
(b) Let (a, b, c) be a triangular triplet and f be metric-preserving, then $(f(a), f(b), f(c))$ is a triangular triplet as well.

Proof: For the proof and for more about metric-preserving functions see [8]. ■

To modify a semimetric into metric, we have utilized a class of metric-preserving SP-modifiers, denoted as the triangle-generating modifiers.

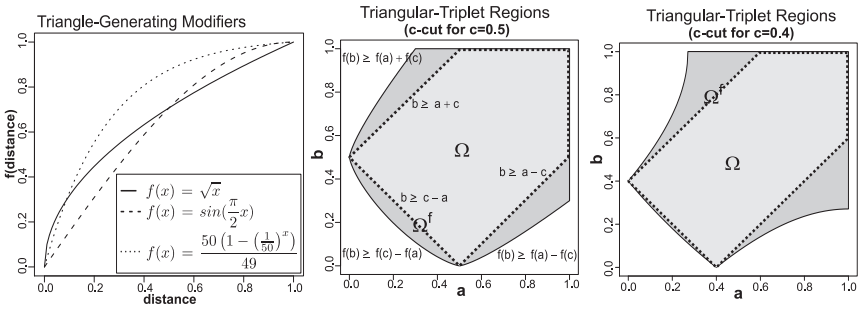


Fig. 2. (a) Several TG-modifiers. Regions Ω, Ω^f; (b) $f(x) = x^{\frac{3}{4}}$ (c) $f(x) = sin(\frac{\pi}{2}x)$

Definition 6 (triangle-generating modifier)

Let a strictly concave SP-modifier f be called a *triangle-generating modifier* (or *TG-modifier*). Having a TG-modifier f, let a d^f be called a *TG-modification*. □

The TG-modifiers (see examples in Figure 2a) not only preserve the triangular inequality, they can even enforce it, as follows.

Theorem 1

Given a semimetric d, then there always exists a TG-modifier f, such that the SP-modification d^f is a metric.

Proof: We show that every ordered triplet (a, b, c) generated by d can be turned by a single TG-modifier f into an ordered triangular triplet.

1. As every semimetric is reflexive and non-negative, it generates ordered triplets just of forms $(0, 0, 0)$, $(0, c, c)$, and (a, b, c), where $a, b, c > 0$. Among these, just the triplets (a, b, c), $0 < a \leq b < c$, can be non-triangular. Hence, it is sufficient to show how to turn such triplets by a TG-modifier into triangular ones.

2. Suppose an arbitrary TG-modifier f_1. From TG-modifiers' properties it follows that $\frac{f_1(a)}{f_1(c)} > \frac{a}{c}$, $\frac{f_1(b)}{f_1(c)} > \frac{b}{c}$, hence $\frac{f_1(a)+f_1(b)}{f_1(c)} > \frac{a+b}{c}$ (theory of concave functions). If $(f_1(a) + f_1(b))/f_1(c) \geq 1$, the triplet $(f_1(a), f_1(b), f_1(c))$ becomes triangular

(i.e. $f_1(a) + f_1(b) \geq f_1(c)$ is true). In case there still exist triplets which have not become triangular after application of f_1, we take another TG-modifier f_2 and compose f_1 and f_2 into $f^*(x) = f_2(f_1(x))$. The compositions (or nestings) $f^*(x) = f_i(\ldots f_2(f_1(x))\ldots)$ are repeated until f^* turns all triplets generated by d into triangular ones – then f^* is the single TG-modifier f we are looking for. ■

The proof shows the more concave TG-modifier we apply, the more triplets become triangular. This effect can be visualized by 3D regions in the space $\langle 0, 1\rangle^3$ of all possible distance triplets, where the three dimensions represent the distance values a,b,c, respectively. In Figures 2b,c see examples of region[1] Ω of all triangular triplets as the dotted-line area. The super-region Ω^f (the solid-line area) represents all the triplets which become (or remain) triangular after the application of TG-modifier $f(x) = x^{\frac{3}{4}}$ and $f(x) = sin(\frac{\pi}{2}x)$, respectively.

3.4 TG-Modifiers Suitable for Metric Search

Although there exist infinitely many TG-modifiers which turn a semimetric d into a metric d^f, their properties can be quite different with respect to the efficiency of search by MAMs. For example, $f(x) = \begin{cases} 0 & (\text{for } x = 0) \\ \frac{x+d^+}{2} & (\text{otherwise}) \end{cases}$ turns every d^+-bounded semimetric d into a metric d^f. However, such a metric is useless for searching, since all classes of objects maintained by a MAM are overlapped by every query, so the retrieval deteriorates to sequential search. This behavior is also reflected in high intrinsic dimensionality of $\mathbb{S}$ with respect to d^f.

In fact, we look for an *optimal* TG-modifier, i.e. a TG-modifier which turns only such non-triangular triplets into triangular ones, which are generated by d. The non-triangular triplets which are not generated by d should remain non-triangular (the white areas in Figures 2b,c), since such triplets represent the "decisions" used by MAMs for filtering of irrelevant objects or classes. The more often such decisions occur, the more efficient the search is (and the lower the intrinsic dimensionality of $\mathbb{S}$ is). As an example, given two vectors u, v of dimensionality n, the optimal TG-modifier for semimetric $d(u, v) = \sum_{i=1}^{n} |u_i - v_i|^2$ is $f(x) = \sqrt{x}$, turning d into the Euclidean (L_2) distance.

From another point of view, the concavity of f determines how much the object clusters (MAMs' classes respectively) become indistinct (overlapped by other clusters/classes). This can be observed indirectly in Figure 2a, where the concave modifiers make the small distances greater, while the great distances remain great; i.e. the mean of distances increases, whereas the variance decreases. To illustrate this fact, we can reuse the example back in Figures 1b,c, where the first DDH was sampled for $d_1 = L_2$, while the second one was sampled for a modification $d_2 = L_2^f$, $f(x) = x^{\frac{1}{4}}$.

In summary, given a dataset $\mathbb{S}$, a semimetric d, and a TG-modifier f, the intrinsic dimensionality is always higher for the modification d^f than for d, i.e. $\rho(\mathbb{S}, d^f) > \rho(\mathbb{S}, d)$. Therefore, an optimal TG-modifier should minimize the increase of intrinsic dimensionality, yet generate the necessary triangular triplets.

[1] The 2D representations of Ω and Ω^f regions are c-cuts of the real 3D regions.

4 The TriGen Algorithm

The question is how to find the optimal TG-modifier f. Had we known an analytical form of d, we could find the TG-modifier "manually". However, if d is implemented by an algorithm, or if the analytical form of d is too complex (e.g. the neural network representation used by COSIMIR), it could be very hard to determine f analytically. Instead, our intention is to find f automatically, regardless of analytical form of d. In other words, we consider a given semimetric d generally as a black box that returns a distance value from a two-object input.

The idea of automatic determination of f makes use of the distance distribution in a sample $\mathbb{S}^*$ of the dataset $\mathbb{S}$. We take m ordered triplets, where each triplet (a, b, c) stores distances between some objects $O_i, O_j, O_k \in \mathbb{S}^* \subseteq \mathbb{S}$, i.e. $(a{=}d(O_i, O_j), b{=}d(O_j, O_k), c{=}d(O_i, O_k))$. Some predefined *base TG-modifiers* f_i (or *TG-bases*) are then applied on the triplets; for each triplet (a, b, c) a modified triplet $(f_i(a), f_i(b), f_i(c))$ is obtained. The *triangle-generating error* ε_Δ (or *TG-error*) is computed as the fraction of triplets remaining non-triangular, $\varepsilon_\Delta = \frac{m_{nt}}{m}$, where m_{nt} is the number of modified triplets remaining non-triangular. Finally, such f_i are selected as *candidates* for the optimal TG-modifier, for which $\varepsilon_\Delta = 0$ or, possibly, $\varepsilon_\Delta \leq \theta$ (where θ is a *TG-error tolerance*). To control the degree of concavity, the TG-bases f_i are parameterizable by a *concavity weight* $w \geq 0$, where $w = 0$ makes every f_i the identity, i.e. $f_i(x, 0) = x$, while with increasing w the concavity of f_i increases as well (a more concave f_i decreases m_{nt}; it turns more triplets into triangular ones). In such a way any TG-base can be forced by an increase of w to minimize the TG-error ε_Δ (possibly to zero).

Among the TG-base candidates the optimal TG-modifier (f_i, w) is chosen such that $\rho(\mathbb{S}^*, d^{f^*(x,w^*)})$ is as low as possible. The TriGen algorithm (see Listing 1) takes advantage of halving the concavity interval $\langle w_{\mathrm{LB}}, w_{\mathrm{UB}} \rangle$ or doubling the upper bound w_{UB}, in order to quickly find the optimal concavity weight w for a TG-base f^*. To keep the computation scalable, the number of iterations (in each iteration w is improved) is limited to e.g. 24 (the iterLimit constant).

Listing 1 (the TriGen algorithm)

Input: semimetric d, set $\mathcal{F}$ of TG-bases, sample $\mathbb{S}^*$, TG-error tolerance θ, iteration limit iterLimit
Output: optimal f, w

```
f = w = null; minIDim = ∞                                                          1
sample m distance triplets into a set T (from S* using d)                          2
for each f* in F                                                                   3
    w_LB = 0; w_UB = ∞; w* = 1; w_best = -1; i = 0                                 4
    while i < iterLimit                                                            5
        if TGError(f*,w*,T) ≤ θ then w_UB = w_best = w* else w_LB = w*             6
        if w_UB ≠ ∞ then w* = (w_LB + w_UB)/2 else w* = 2 * w*                     7
        i = i + 1;                                                                 8
    end while                                                                      9
    if w_best ≥ 0 then                                                             10
        idim = IDim(f*,w_best,T)                                                   11
        if idim < minIDim then f = f*; w = w_best; minIDim = idim                  12
    end if                                                                         13
end for                                                                            14
```

In Listing 2 the TGError function is described. The TG-error ε_Δ is computed by taking m distance triplets from the dataset sample $\mathbb{S}^*$ onto which the examined

TG-base f^* together with the current weight w^* is applied. The distance triplets are sampled only once – at the beginning of the TriGen's run – whereas the modified triplets are recomputed for each particular f^*, w^*.

The not-listed function IDim (computing $\rho(\mathbb{S}^*, d^{f^*(x,w^*)})$ makes use of the previously obtained modified triplets as well, however, the values in the triplets are used independently; just for evaluation of the intrinsic dimensionality.

Listing 2 (the TGError function)

Input: TG-base f^*, concavity weight w^*, set $\mathcal{T}$ of m sampled distance triplets
Output: TG-error ε_Δ

```
m_nt = 0                                                                          1
for each ot in T // "ot" stands for "ordered triplet"                             2
    if f*(ot.a, w*) + f*(ot.b, w*) < f*(ot.c, w*) then m_nt = m_nt + 1            3
end for                                                                           4
ε_Δ = m_nt / m                                                                    5
```

4.1 Sampling the Distance Triplets

Initially, we have n objects in the dataset sample $\mathbb{S}^*$. Then we create an $n \times n$ distance matrix for storage of pairwise distances $d_{ij} = d(O_i, O_j)$ between the sampled objects. In such a way we are able to obtain up to $m = \binom{n}{3}$ distance triplets for at most $\frac{n(n-1)}{2}$ distance computations. Thus, to obtain a sufficiently large number of distance triplets, the dataset sample $\mathbb{S}^*$ needs to be quite small. Each of the m distance triplets is sampled by a random choice of three among the n objects, while the respective distances are retrieved from the matrix. Naturally, the values in the matrix could be computed "on-demand", just in the moment a distance retrieval is requested. Since d is symmetric, the sub-diagonal half of the matrix can be used for storage of the modified distances $d^f_{ji} = f^*(d_{ij}, w^*)$, however, these are recomputed for each particular f^*, w^*. As in case of distances, also the modified distances can be computed "on-demand".

4.2 Time Complexity Analysis (simplified)

Let $|\mathbb{S}^*|$ be the number of objects in the sample $\mathbb{S}^*$, m be the number of sampled triplets, and $O(d)$ be the complexity of single distance computation. The complexity of $f(\cdot)$ computation is supposed $O(1)$. The overall complexity of TriGen is then $O(|\mathbb{S}^*|^2 * O(d) + \mathsf{iterLimit} * |\mathcal{F}| * m)$, i.e. the distance matrix computation plus the main algorithm. The number of TG-bases $|\mathcal{F}|$ as well as the number of iterations (variable iterLimit) are assumed as (small) constants, hence we get $O(|\mathbb{S}^*|^2 * O(d) + m)$. The size of $\mathbb{S}^*$ and the number m affect the precision of TGError and IDim values, so we can trade off the TriGen's complexity and the precision by choosing $|\mathbb{S}^*| = O(1), O(|\mathbb{S}|)$ and $m = O(1), O(|\mathbb{S}^*|)$, or e.g. $O(|\mathbb{S}^*|^2)$.

4.3 Default TG-Bases

We propose two general-purpose TG-bases for the TriGen algorithm. The simpler one, the *Fractional-Power TG-base* (or *FP-base*), is defined as $\mathrm{FP}(x, w) = x^{\frac{1}{1+w}}$,

see Figure 3a. The advantage of FP-base is there always exists a concavity weight w for which the modified semimetric becomes metric, i.e. the TriGen will always find a solution (after a number of iterations). Furthermore, when using the FP-base, the semimetric d needs not to be bounded. A particular disadvantage of the FP-base is that its concavity is controlled globally, just by the weight w.

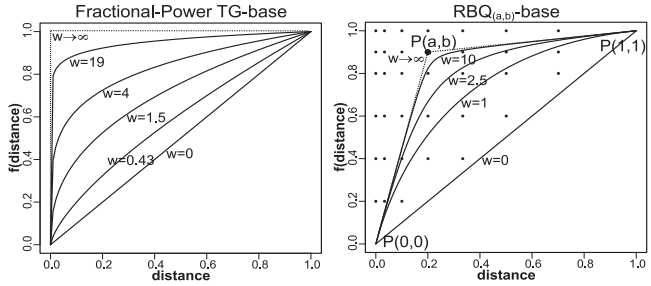


Fig. 3. (a) FP-base (b) $\mathrm{RBQ}_{(a,b)}$-base

As a more flexible TG-base, we have utilized the Rational Bézier Quadratic curve. To derive a proper TG-base from the curve, the three Bézier points are specified as $(0,0),(a,b),(1,1)$, where $0 \leq a < b \leq 1$, see Figure 3b. The *Rational Bézier Quadratic TG-base* (simply RBQ-base) is defined as $\mathrm{RBQ}_{(a,b)}(x,w) = -(\Psi - x + wx - aw) \cdot (-2bwx + 2bw^2x - 2abw^2 + 2bw - x + wx - aw + \Psi(1 - 2bw))/(-1 + 2aw - 4awx - 4a^2w^2 + 2aw^2 + 4aw^2x + 2wx - 2w^2x + 2\Psi(1-w))$, where $\Psi = \sqrt{-x^2 + x^2w^2 - 2aw^2x + a^2w^2 + x}$. The additional RBQ parameters a, b (the second Bézier point) are treated as constants, i.e. for various a, b values (see the dots in Figure 3b) we get multiple RBQ-bases, which are all individually inserted into the set $\mathcal{F}$ of TriGen's input. To keep the RBQ evaluation correct, a possible division by zero or $\Psi^2 < 0$ is prevented by a slight shift of a or w. The advantage of RBQ-bases is the place of maximal concavity can be controlled locally by a choice of (a,b), hence, for a given concavity weight w^* we can achieve lower value of either $\rho(\mathbb{S}^*, d^{f^*(x,w^*)})$ or ε_Δ just by choosing different a, b.

As a particular limitation, for usage of RBQ-bases the semimetric d must be bounded (due to the third Bézier point (1,1)). Furthermore, for an RBQ-base with $(a,b) \neq (0,1)$ the TG-error ε_Δ could be generally greater than the TG-error tolerance θ, even in case $w \to \infty$. Nevertheless, having the FP-base or the $\mathrm{RBQ}_{(0,1)}$-base in $\mathcal{F}$, the TriGen will always find a TG-modifier such that $\varepsilon_\Delta \leq \theta$.

4.4 Notes on the Triangular Inequality

As we have shown, the TriGen algorithm produces a TG-modifier which generates the triangular inequality property for a particular semimetric d. However, we have to realize the triangular inequality is generated just according to the dataset sample $\mathbb{S}^*$ (to the sampled distance triplets, actually). A TG-modification d^f being metric according to $\mathbb{S}^*$ has not to be a "full metric" according to the entire dataset $\mathbb{S}$ (or even to $\mathbb{U}$), so that searching in $\mathbb{S}$ by a MAM could become only

approximate, even in case $\theta = 0$. Nevertheless, in most applications a (random) dataset sample $\mathbb{S}^*$ is supposed to have the distance distribution similar to that of $\mathbb{S} \cup \{Q\}$, and also the sampled distance triplets are expected to be representative.

Moreover, the construction of such a TG-modifier f, for which $(\mathbb{S}, d^f)$ is metric space but $(\mathbb{U}, d^f)$ is not, can be beneficial for the efficiency of search, since the intrinsic dimensionality of $(\mathbb{S}, d^f)$ can be significantly lower than that of $(\mathbb{U}, d^f)$. The above claims are verified experimentally in the following section, where the retrieval error (besides pure ε_Δ) and the retrieval efficiency (besides pure $\rho(\mathbb{S}, d^f)$) are evaluated. Nonetheless, to keep the terminology correct let us read a metric d^f created by the TriGen as a *TriGen-approximated metric*.

5 Experimental Results

To examine the proposed method, we have performed extensive testing of the TriGen algorithm as well as evaluation of the generated distances with respect to the effectiveness and efficiency of retrieval by two MAMs (M-tree and PM-tree).

5.1 The Testbed

We have examined 10 non-metric distance measures (all described in Section 1.6) on two datasets (images and polygons). The dataset of images consisted of 10,000 web-crawled images [30] transformed into 64-level gray-scale histograms. We have tested 6 semimetrics on the images: the COSIMIR measure (denoted **COSIMIR**), the 5-median L_2 distance (**5-medL2**), the squared L_2 distance (**L2square**), and three fractional L_p distances ($p = 0.25, 0.5, 0.75$, denoted **FracLp**p). The **COSIMIR** network was trained by 28 user-assessed pairs of images.

The synthetic dataset of polygons consisted of 1,000,000 2D polygons, each consisting of 5 to 10 vertices. We have tested 4 semimetrics on the polygons: the 3-median and 5-median Hausdorff distances (denoted **3-medHausdorff**, **5-medHausdorff**), and the time warping distance with δ chosen as L_2 and L_∞, respectively (denoted **TimeWarpL2**, **TimeWarpLmax**). The **COSIMIR**, **5-medL2** and k-**medHausdorff** measures were adjusted to be semimetrics, as described in Section 3.1. All the semimetrics were normed to return distances from $\langle 0, 1 \rangle$.

5.2 The TriGen Setup

The TriGen algorithm was used to generate the optimal TG-modifier for each semimetric (considering the respective dataset). To examine the relation between retrieval error of MAMs and the TG-error, we have constructed several TG-modifiers for each semimetric, considering different values of TG-error tolerance $\theta \geq 0$. The TriGen's set of bases $\mathcal{F}$ was populated by the FP-base and 116 RBQ-bases parametrized by all such pairs (a, b) that $a \in \{0, 0.005, 0.015, 0.035, 0.075, 0.155\}$, where for a value of a the values of b were multiples of 0.05 limited by $a < b \leq 1$. The dataset sample $\mathbb{S}^*$ used by TriGen consisted of $n = 1000$ randomly selected objects in case of images (10% of the dataset), and $n = 5000$ in case of polygons (0.5% of the dataset). The distance matrix built from the respective dataset sample $\mathbb{S}^*$ was used to form $m = 10^6$ distance triplets.

In Table 1 see the optimal TG-modifiers found for the semimetrics by TriGen, considering $\theta = 0$ and $\theta = 0.05$, respectively. In the first column, best RBQ modifier parameters (best in sense of lowest ρ depending on a, b) are presented. In the second column, the achieved ρ for a concavity weight w of the FP-base is presented, in order to make a comparison with the best RBQ modifier. Among RBQ- and FP-bases, the winning modifier (with respect to lowest ρ) is printed in bold. When considering $\theta = 0.05$, **FracLp0.5**, **3-medHausdorff**, **5-medHausdorff** even need not to be modified (see the zero weights by the FP-base), since the TG-error is already below θ. Also note that for **L2square** and $\theta = 0$ the weight of FP-base modifier is $w = 0.99$, instead of $w = 1.0$ (which would turn **L2square** into L_2 distance). That is because the intrinsic dimensionality of the dataset sample $\mathbb{S}^*$ is lower than that of the universe $\mathbb{U}$ (64-dimensional vector space).

Table 1. TG-modifiers found by TriGen.

	$\theta = 0.00$				$\theta = 0.05$			
	best RBQ-base		FP-base		best RBQ-base		FP-base	
semimetric	(a, b)	ρ	ρ	w	(a, b)	ρ	ρ	w
L2square	**(0, 0.15)**	**3.74**	4.22	0.99	**(0, 0.05)**	**2.82**	3.02	0.59
COSIMIR	**(0, 0.45)**	**12.2**	27.2	4.33	**(0.005, 0.15)**	**3.19**	3.80	0.63
5-medL2	(0, 0.1)	37.7	**19.8**	**16.5**	(0, 0.05)	4.28	**3.17**	**3.88**
FracLp0.25	**(0, 0.45)**	**12.7**	15.2	2.29	(0.035, 0.05)	3.50	**3.30**	**0.30**
FracLp0.5	**(0, 0.05)**	**7.57**	8.37	0.87	**(0, 0.2)**	**3.28**	3.34	0.06
FracLp0.75	**(0, 0.75)**	**5.13**	5.69	0.30	**any**	**3.77**	**3.77**	**0**
3-medHausdorff	**(0, 0.05)**	**3.77**	5.11	0.60	**any**	**2.28**	**2.28**	**0**
5-medHausdorff	**(0, 0.05)**	**3.42**	4.12	0.35	**any**	**2.45**	**2.45**	**0**
TimeWarpL2	(0, 0.55)	10.0	**9.48**	**1.48**	**(0.035, 0.1)**	**2.72**	2.76	0.23
TimeWarpLmax	**(0.005, 0.3)**	**8.75**	9.69	1.52	**(0, 0.1)**	**2.83**	2.86	0.26

In Figure 4 see the intrinsic dimensionalities $\rho(\mathbb{S}^*, d^f)$ with respect to the growing TG-error tolerance θ (f is the optimal TG-modifier found by TriGen).

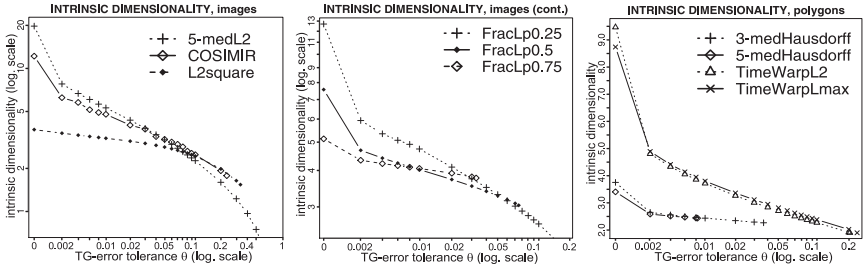


Fig. 4. Intrinsic dimensionality of images and polygons

The rightmost point $[\theta, \rho]$ of a particular curve in each figure means θ is the maximum ε_Δ value that can be reached; for such a value (and all greater) the concavity weight w becomes zero. Similar "endpoints" on curves appear also in other following curves that depend on the TG-error tolerance.

The Figure 5a shows the impact of m sampled triplets (used by TGError) on the intrinsic dimensionality, considering $\theta = 0$ and only the FP-base in $\mathcal{F}$. The more triplets, the more accurate value of ε_Δ and the more concave TG-modifier is needed to keep $\varepsilon_\Delta = 0$, so the concavity weight and the intrinsic dimensionality

grow. However, except for **5-medHausdorff**, the growth of intrinsic dimensionality is quite slow for $m > 10^6$ (and even slower if we set $\theta > 0$).

For the future we plan to improve the simple random selection of triplets from the distance matrix, in order to obtain more representative triplets, and thus more accurate values of ε_Δ together with keeping m low.

5.3 Indexing & Querying

In order to evaluate the efficiency and effectiveness of search when using TriGen-approximated metrics, we have utilized the M-tree [7] and the PM-tree [27].

For either of the datasets several M-tree and PM-tree indices were built, differed in the metric d^f employed – for each semimetric and each θ value a d^f was found by TriGen, and an index created. The setup of (P)M-tree indices is summarized in Table 2 (for technical details see [7, 26, 27]).

Table 2. M-tree and PM-tree setup

disk page size:	4 kB | avg. page utilization: 41%–68%
PM-tree pivots:	64 inner node pivots, 0 leaf pivots
image indices size:	1–2 MB (M-tree) 1.2–2.2 MB (PM-tree)
polygon indices size:	140–150 MB (both M-tree and PM-tree)
construction method:	MinMax + SingleWay (+ slim-down)

To achieve more compact MAM classes, the indices (both M-tree and PM-tree) built on the image dataset were post-processed by the *generalized slim-down algorithm* [26]. The 64 global pivot objects used by PM-tree indices were sampled among the n objects already used for the TriGen's distance matrix construction.

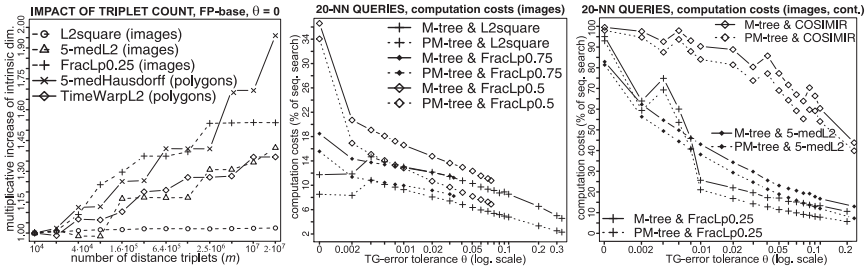


Fig. 5. Impact of triplet count; 20-NN queries on images (costs)

All the (P)M-tree indices were used to process k-NN queries. Since the TriGen-generated modifications are generally metric approximations (especially when $\theta > 0$), the filtration of (P)M-tree branches was affected by a *retrieval error* (the relative error in precision and recall). The retrieval error was computed as the *Jaccard distance* E_{NO} (or normed overlap distance) between the query result QR_{MAM} returned by a (P)M-tree index and the correct query result QR_{SEQ} (obtained by sequential search of the dataset), i.e. $E_{NO} = 1 - \frac{|QR_{MAM} \cap QR_{SEQ}|}{|QR_{MAM} \cup QR_{SEQ}|}$.

To examine retrieval efficiency, the computation costs needed for query evaluation were compared to the costs spent by sequential search. Every query was repeated for 200 randomly selected query objects, and the results were averaged.

In Figures 5b,c see the costs of 20-NN queries processed on image indices, depending on growing θ. The intrinsic dimensionalities decrease, and so the searching becomes more efficient (e.g. down to 2% of costs spent by sequential search for $\theta = 0.4$ and the TG-modification of **L2square**). On the other hand, for $\theta = 0$ the TG-modifications of **COSIMIR** and **FracLp0.25** imply high intrinsic dimensionality, so the retrieval deteriorates to almost sequential search.

In Figures 6a,b the retrieval error E_{NO} is presented for growing θ. In Figures 6c and 7a see the retrieval efficiency and error for 20-NN querying on the polygon indices. As supposed, the error grows with growing TG-error tolerance θ. Interestingly, the values of θ tend to be the upper bounds to the values of E_{NO}, so we could utilize θ in an *error model* for prediction of E_{NO}.

In case of **5-medL2**, **3-medHausdorff** (and partly **COSIMIR**, **5-medHausdorff**) indices, the retrieval error was non-zero even for $\theta = 0$. This was caused by neglecting some "pathological" distance triplets when computing the TGError function (see Section 4), so the triangular inequality was not preserved for all triplets, and the filtering performed by (P)M-tree was sometimes (but rarely) incorrect. In other cases (where $\theta = 0$) the retrieval error was zero.

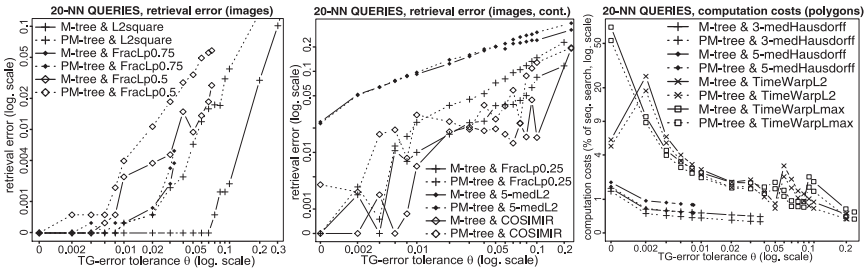


Fig. 6. 20-NN queries on images and polygons (retrieval error, costs)

The costs and the error for k-NN querying are presented in Figures 7b,c – with respect to the increasing number of nearest neighbors k.

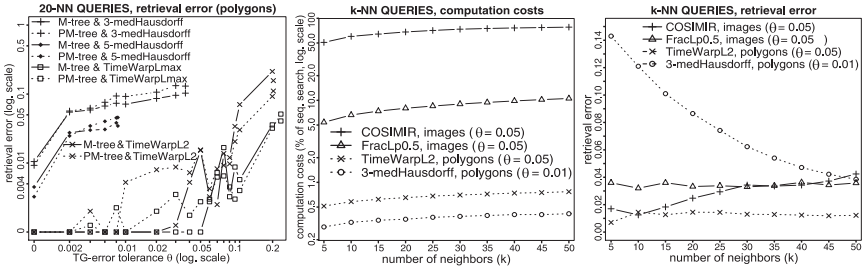


Fig. 7. 20-NN queries on polygons (retrieval error); k-NN queries (costs, retrieval error)

Summary. Based on the above presented experimental results, we can observe that non-metric searching by MAMs, together with usage of the TriGen algorithm as the first step of the indexing, can successfully merge both aspects, the

retrieval efficiency as well as the effectiveness. The efficiency achieved is by far higher than simple sequential search (even for $\theta = 0$), whereas the retrieval error is kept very low for reasonable values of θ. Moreover, by choosing different values of θ we get a trade-off between the effectiveness and efficiency thus, the TriGen algorithm provides a scalability mechanism for non-metric search by MAMs.

On the other hand, some non-metric measures are very hard to use for efficient *exact* search by MAMs (i.e. keeping $E_{NO} = 0$), in particular the **COSIMIR** and the **FracLp0.25** measures. Nevertheless, for *approximate* search ($E_{NO} > 0$) also these measures can be utilized efficiently.

6 Conclusions

In this paper we have proposed a general approach to non-metric similarity search in multimedia databases by use of metric access methods (MAMs). We have shown the triangular inequality property is not restrictive for similarity search and can be enforced for every semimetric (modifying it to a metric). Furthermore, we have introduced the TriGen algorithm for automatic turning of any black-box semimetric into metric (or at least approximation of a metric) just by use of distance distribution in a fraction of the database. Such a "TriGen-approximated metric" can be safely used to search the database by any MAM, while the similarity orderings with respect to a query object (the retrieval effectiveness) are correctly preserved. The main result of the paper is a fact that we can quickly search a multimedia database when using unknown non-metric similarity measures, while the retrieval error achieved can be very low.

Acknowledgements. This research has been supported by grants 201/05/P036 of the Czech Science Foundation (GAČR) and "Information Society" 1ET100300419 – National Research Programme of the Czech Republic. I also thank Július Štroffek for his implementation of backpropagation network (used for the COSIMIR experiments).

References

1. C. C. Aggarwal, A. Hinneburg, and D. A. Keim. On the surprising behavior of distance metrics in high dimensional spaces. In *ICDT*. LNCS, Springer, 2001.
2. F. Ashby and N. Perrin. Toward a unified theory of similarity and recognition. *Psychological Review*, 95(1):124–150, 1988.
3. I. Bartolini, P. Ciaccia, and M. Patella. WARP: Accurate Retrieval of Shapes Using Phase of Fourier Descriptors and Time Warping Distance. *IEEE Pattern Analysis and Machine Intelligence*, 27(1):142–147, 2005.
4. E. Chávez and G. Navarro. A Probabilistic Spell for the Curse of Dimensionality. In *ALENEX'01, LNCS 2153*, pages 147–160. Springer, 2001.
5. E. Chávez, G. Navarro, R. Baeza-Yates, and J. L. Marroquín. Searching in metric spaces. *ACM Computing Surveys*, 33(3):273–321, 2001.
6. P. Ciaccia and M. Patella. Searching in metric spaces with user-defined and approximate distances. *ACM Database Systems*, 27(4):398–437, 2002.
7. P. Ciaccia, M. Patella, and P. Zezula. M-tree: An Efficient Access Method for Similarity Search in Metric Spaces. In *VLDB'97*, pages 426–435, 1997.

8. P. Corazza. Introduction to metric-preserving functions. *American Mathematical Monthly*, 104(4):309–23, 1999.
9. V. Dohnal, C. Gennaro, P. Savino, and P. Zezula. D-index: Distance searching index for metric data sets. *Multimedia Tools and Applications*, 21(1):9–33, 2003.
10. M. Donahue, D. Geiger, T. Liu, and R. Hummel. Sparse representations for image decomposition with occlusions. In *CVPR*, pages 7–12, 1996.
11. C. Faloutsos and K. Lin. Fastmap: A Fast Algorithm for Indexing, Data-Mining and Visualization of Traditional and Multimedia Datasets. In *SIGMOD*, 1995.
12. R. F. S. Filho, A. J. M. Traina, C. Traina, and C. Faloutsos. Similarity search without tears: The OMNI family of all-purpose access methods. In *ICDE*, 2001.
13. K.-S. Goh, B. Li, and E. Chang. DynDex: a dynamic and non-metric space indexer. In *ACM Multimedia*, 2002.
14. P. Hart. The condensed nearest neighbour rule. *IEEE Transactions on Information Theory*, 14(3):515–516, 1968.
15. G. R. Hjaltason and H. Samet. Properties of embedding methods for similarity searching in metric spaces. *IEEE Patt.Anal. and Mach.Intell.*, 25(5):530–549, 2003.
16. P. Howarth and S. Ruger. Fractional distance measures for content-based image retrieval. In *ECIR 2005*, pages 447–456. LNCS 3408, Springer-Verlag, 2005.
17. D. Huttenlocher, G. Klanderman, and W. Rucklidge. Comparing images using the hausdorff distance. *IEEE Patt. Anal. and Mach. Intell.*, 15(9):850–863, 1993.
18. D. Jacobs, D. Weinshall, and Y. Gdalyahu. Classification with nonmetric distances: Image retrieval and class representation. *IEEE Pattern Analysis and Machine Intelligence*, 22(6):583–600, 2000.
19. A. K. Jain and D. E. Zongker. Representation and recognition of handwritten digits using deformable templates. *IEEE Patt.Anal.Mach.Intell.*, 19(12):1386–1391, 1997.
20. O. Jesorsky, K. J. Kirchberg, and R. Frischholz. Robust face detection using the hausdorff distance. In *AVBPA*, pages 90–95. LNCS 2091, Springer-Verlag, 2001.
21. C. L. Krumhansl. Concerning the applicability of geometric models to similar data: The interrelationship between similarity and spatial density. *Psychological Review*, 85(5):445–463, 1978.
22. T. Mandl. Learning similarity functions in information retrieval. In *EUFIT*, 1998.
23. E. Rosch. Cognitive reference points. *Cognitive Psychology*, 7:532–47, 1975.
24. E. Rothkopf. A measure of stimulus similarity and errors in some paired-associate learning tasks. *J. of Experimental Psychology*, 53(2):94–101, 1957.
25. S. Santini and R. Jain. Similarity measures. *IEEE Pattern Analysis and Machine Intelligence*, 21(9):871–883, 1999.
26. T. Skopal, J. Pokorný, M. Krátký, and V. Snášel. Revisiting M-tree Building Principles. In *ADBIS, Dresden*, pages 148–162. LNCS 2798, Springer, 2003.
27. T. Skopal, J. Pokorný, and V. Snášel. Nearest Neighbours Search using the PM-tree. In *DASFAA '05, Beijing, China*, pages 803–815. LNCS 3453, Springer, 2005.
28. A. Tversky. Features of similarity. *Psychological review*, 84(4):327–352, 1977.
29. A. Tversky and I. Gati. Similarity, separability, and the triangle inequality. *Psychological Review*, 89(2):123–154, 1982.
30. Wavelet-based Image Indexing and Searching, Stanford University, `wang.ist.psu.edu`.
31. R. Weber, H.-J. Schek, and S. Blott. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In *VLDB*, 1998.
32. D. L. Wilson. Asymptotic properties of nearest neighbor rules using edited data. *IEEE Transactions on Systems, Man, and Cybernetics*, 2(3):408–421, 1972.
33. B.-K. Yi, H. V. Jagadish, and C. Faloutsos. Efficient retrieval of similar time sequences under time warping. In *ICDE '98*, pages 201–208, 1998.

Chapter 7

Metric Indexing for the Vector Model in Text Retrieval

Tomáš Skopal
Pavel Moravec
Jaroslav Pokorný
Václav Snášel

Metric Indexing for the Vector Model in Text Retrieval [45]

Regular paper at the 11th International Conference on String Processing and Information Retrieval (SPIRE 2004), Padova, Italy, October 2004

Published in the *Lecture Notes in Computer Science* (LNCS), vol. 3246, pages 183–195, *Springer-Verlag*, ISSN 0302-9743, ISBN 978-3-540-23210-0

Metric Indexing for the Vector Model in Text Retrieval

Tomáš Skopal[1], Pavel Moravec[2], Jaroslav Pokorný[1], and Václav Snášel[2]

[1] Charles University in Prague, Department of Software Engineering,
Malostranské nám. 25, 118 00 Prague, Czech Republic, EU
tomas@skopal.net, jaroslav.pokorny@mff.cuni.cz
[2] VŠB – Technical University of Ostrava, Department of Computer Science,
17. listopadu 15, 708 33 Ostrava, Czech Republic, EU
{pavel.moravec, vaclav.snasel}@vsb.cz

Abstract. In the area of Text Retrieval, processing a query in the vector model has been verified to be qualitatively more effective than searching in the boolean model. However, in case of the classic vector model the current methods of processing many-term queries are inefficient, in case of LSI model there does not exist an efficient method for processing even the few-term queries. In this paper we propose a method of vector query processing based on metric indexing, which is efficient especially for the LSI model. In addition, we propose a concept of approximate semi-metric search, which can further improve the efficiency of retrieval process. Results of experiments made on moderate text collection are included.

1 Introduction

The Text Retrieval (TR) models [4, 3] provide a formal framework for retrieval methods aimed to search huge collections of text documents. The classic vector model as well as its algebraic extension LSI have been proved to be more effective (according to precision/recall measures) than the other existing models[1]. However, current methods of vector query processing are not much efficient for many-term queries, while in the LSI model they are inefficient at all. In this paper we propose a method of vector query processing based on metric indexing, which is highly efficient especially for searching in the LSI model.

1.1 Classic Vector Model

In the classic vector model, each document D_j in a collection C ($0 \leq j \leq m$, $m = |C|$) is characterized by a single vector d_j, where each coordinate of d_j is associated with a term t_i from the set of all unique terms in C ($0 \leq i \leq n$, where n is the number of terms). The value of a vector coordinate is a real number $w_{ij} \geq 0$ representing the *weight* of the i-th term in the j-th document. Hence, a collection of documents can be represented by an $n \times m$ *term-by-document* matrix A. There are many ways how to compute the term weights w_{ij} stored in A. A popular weight construction is computed as $tf * idf$ (see e.g. [4]).

[1] For a comparison over various TR models we refer to [20, 11].

Queries. The most important problem about the vector model is the querying mechanism that searches matrix A with respect to a query, and returns only the relevant document vectors (appropriate documents respectively). The query is represented by a vector q the same way as a document is represented. The goal is to return the most similar (relevant) documents to the query. For this purpose, a similarity function must be defined, assessing a similarity value to each pair of query and document vectors (q, d_j). In the context of TR, the *cosine measure* $\mathrm{SIM}_{cos}(q, d_j) = \frac{\sum_{k=1}^{n} q_k \cdot w_{kj}}{\sqrt{\sum_{k=1}^{n} {q_k}^2 \cdot \sum_{k=1}^{n} {w_{kj}}^2}}$ is widely used. During a query processing, the columns of A (the document vectors) are compared against the query vector using the cosine measure, while the sufficiently similar documents are returned as a result. According to the query extent, we distinguish *range queries* and *k-nearest neighbors (k-NN) queries*. A range query returns documents similar to the query more than a given similarity threshold. A k-NN query returns the k most similar documents.

Generally, there are two ways how to specify a query. First, a *few-term query* is specified by the user using a few terms, while an appropriate vector for such a query is very sparse. Second, a *many-term query* is specified using a text document, thus the appropriate query vector is usually more dense. In this paper we focus just on the many-term queries, since they better satisfy the similarity search paradigm which the vector model should follow.

1.2 LSI Vector Model (simplified)

Simply said, the LSI (latent semantic indexing) model [11, 4] is an algebraical extension of the classic vector model. First, the term-by-document matrix A is decomposed by singular value decomposition (SVD) as $A = U\Sigma V^T$. The matrix U contains *concept vectors*, where each concept vector is a linear combination of the original terms. The concepts are *meta-terms* (groups of terms) appearing in the original documents. While the term-by-document matrix A stores document vectors, the *concept-by-document* matrix ΣV^T stores *pseudo-document* vectors. Each coordinate of a pseudo-document vector represents a weight of an appropriate concept in a document.

Latent Semantics. The concept vectors are ordered with respect to their significance (appropriate singular values in Σ). Consequently, only a small number of concepts is really significant – these concepts represent (statistically) the main themes present in the collection – let us denote this number as k. The remaining concepts are unimportant (noisy concepts) and can be omitted, thus the dimensionality is reduced from n to k. Finally, we obtain an approximation (rank-k SVD) $A \approx U_k \Sigma_k V_k^T$, where for sufficiently high k the approximation error will be negligible. Moreover, for a low k the effectiveness can be subjectively even higher (according to the precision/recall values) than for a higher k [3]. When searching in a real-world collection, the optimal k is usually ranged from several tens to several hundreds. Unlike the term-by-document matrix A, the concept-by-document matrix $\Sigma_k V_k^T$ as well as the concept base matrix U are dense.

Queries. Searching for documents in the LSI model is performed the same way as in the classic vector model, the difference is that matrix $\Sigma_k V_k^T$ is searched instead of A. Moreover, the query vector q must be projected into the concept base, i.e. $U_k^T q$ is the *pseudo-query vector* used by LSI. Since the concept vectors of U are dense, a pseudo-query vector is dense as well.

1.3 Vector Query Processing

In this paper we focus on efficiency of vector query processing. More specifically, we can say that a query is processed efficiently in case that only a small proportion of the matrix storage volume is needed to load and process. In this section we outline several existing approaches to the vector query processing.

Document Vector Scanning. The simplest method how to process a query is the sequential scanning of all the document vectors (i.e. the columns of A, $\Sigma_k V_k^T$ respectively). Each document vector is compared against the query vector using the similarity function, while sufficiently similar documents are returned to the user. It is obvious that for any query the whole matrix must be processed. However, sequential processing of the whole matrix is sometimes more efficient (from the disk management point of view) than a random access to a smaller part of the matrix used by some other methods.

Term Vector Filtering. For sparse query vectors (few-term queries respectively), there exists a more efficient scanning method. Instead of the document vectors, the term vectors (i.e. the rows of the matrix) are processed. The cosine measure is computed simultaneously for all the document vectors, "orthogonally" involved in the term vectors. Due to the simultaneous cosine measure evaluation a set of m accumulators (storing the evolving similarities between each document and the query) must be maintained in memory. The advantage of term filtering is that only those term vectors must be scanned, for which the appropriate term weights in the query vector are nonzero. The term vector filtering can be easily provided using an inverted file – as a part of the boolean model implementation [15].

The simple method of term filtering has been improved by an approximate approach [19] reducing the time as well as space costs. Generally, the improvement is based on early termination of query processing, exploiting a restructured inverted file where the term entries are sorted according to the decreasing occurrences of a term in document. Thus, the most relevant documents in each term entry are processed first. As soon as the first document is found in which the number of term occurrences is less than a given addition threshold, the processing of term entry can stop, because all the remaining documents have the same or less importance as the first rejected document. Since some of the documents are never reached during a query processing, the number of used accumulators can be smaller than m, which saves also the space costs. Another improvement

of the inverted file exploiting *quantized weights* was proposed recently [2], even more reducing the search costs.

Despite the above mentioned improvements, the term vector filtering is generally not so much efficient for many-term queries, because the number of filtered term vectors is decreased. Moreover, the term vector filtering is completely useless for the LSI model, since each pseudo-query vector is dense, and none of the term vectors can be skipped.

Signature Methods. Signature files are a popular filtering method in the boolean model [13], however, there were only few attempts made to use them in the vector model. In that case, the usage of signature files is not so straightforward due to the term weights. Weight-partitioned signature files (WPSF) [14] try to solve the problem by recording the term weights in so-called TF-groups. A sequential file organization was chosen for the WPSF which caused excessive search of the signature file. An improvement was proposed recently [16] using the S-trees [12] to speedup the signature file search. Another signature-like approach is the VA-file [6]. In general, usage of the signature methods is still complicated for the vector model, and the results achieved so far are rather poor.

2 Metric Indexing

Since in the vector model the documents are represented as points within an n-dimensional vector space, in our approach we create an index for the term-by-document matrix (for the concept-by-document matrix in case of LSI) based on metric access methods (MAMs) [8]. A property common to all MAMs is that they exploit only a metric function for the indexing. The metric function stands for a similarity function, thus metric access methods provide a natural way for similarity search. Among many of MAMs, we have chosen the M-tree.

2.1 M-tree

The M-tree [9, 18, 21] is a dynamic data structure designed to index objects of metric datasets. Let us have a metric space $\mathcal{M} = (\mathbb{U}, d)$ where $\mathbb{U}$ is an object universe (usually a vector space), and d is a function measuring distance between two objects in $\mathbb{U}$. The function d must be a metric, i.e. it must satisfy the axioms of reflexivity, positivity, symmetry and triangular inequality. Let $\mathbb{S} \subseteq \mathbb{U}$ be a dataset to be indexed. In case of the vector model in TR, an object $O_i \in \mathbb{S}$ is represented by a (pseudo-)document vector of a document D_i. The particular metric d, replacing the cosine measure, will be introduced in Section 2.2.

Like the other indexing trees based on B^+-tree, the M-tree structure is a balanced hierarchy of nodes. In M-tree the objects are distributed in a hierarchy of *metric regions* (each node represents a single metric region) which can be, in turn, interpreted as a hierarchy of object clusters. The nodes have a fixed capacity and a minimum utilization threshold. The leaf nodes contain *ground entries grnd(O_i)* of the indexed objects themselves, while in the inner nodes the

routing entries rout(O_j) are stored, representing the metric regions and routing to their covering subtrees. Each routing entry determines a metric region in space $\mathcal{M}$ where the object O_j is a center of that region and r_{O_j} is a radius bounding the region. For the hierarchy of metric regions (routing entries $rout(O_j)$ respectively) in the M-tree, the following requirement must be satisfied:

All the objects of ground entries stored in the leaves of the covering subtree of $rout(O_j)$ must be spatially located inside the region defined by $rout(O_j)$.

The most important consequence of the above requirement is that many regions on the same M-tree level may overlap. An example in Figure 1 shows several objects partitioned among metric regions and the appropriate M-tree. We can see that the regions defined by $rout_1(O_1)$, $rout_1(O_2)$, $rout_1(O_4)$ overlap. Moreover, object O_5 is located inside the regions of $rout_1(O_1)$ and $rout_1(O_4)$ but it is stored just in the subtree of $rout_1(O_4)$. Similarly, the object O_3 is located even in three regions but it is stored just in the subtree of $rout_1(O_2)$.

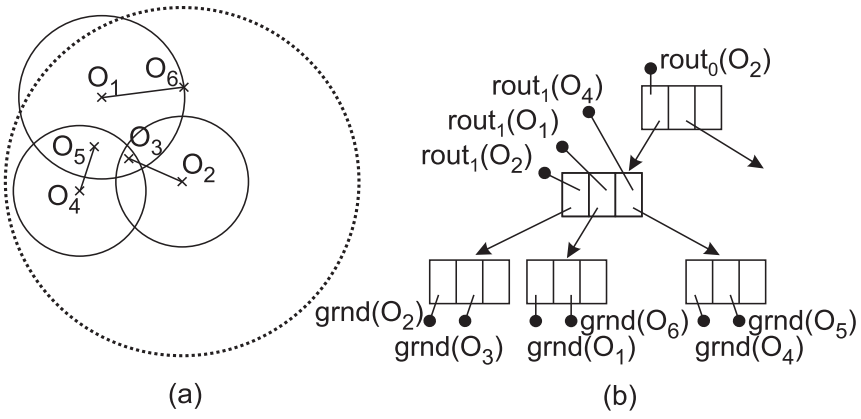


Fig. 1. Hierarchy of metric regions (a) and the appropriate M-tree (b)

Similarity Queries in the M-tree. The structure of M-tree natively supports similarity queries. The similarity function is represented by the metric function d where the close objects are interpreted as similar.

A range query `RangeQuery(`Q`,`r_Q`)` is specified as a *query region* given by a query object Q and a query radius r_Q. The purpose of a range query is to retrieve all such objects O_i satisfying $d(Q, O_i) \leq r_Q$. A k-nearest neighbours query (k-NN query) `kNNQuery(`Q`,`k`)` is specified by a query object Q and a number k. A k-NN query retrieves the first k nearest objects to Q.

During the range query processing (k-NN query processing respectively), the M-tree hierarchy is being traversed down. Only if a routing entry $rout(O_j)$ (its metric region respectively) overlaps the query region, the covering subtree of $rout(O_j)$ is relevant to the query and thus further processed.

2.2 Application of M-tree in the Vector Model

In the vector model the objects O_i are represented by (pseudo-)document vectors d_i, i.e. by columns of term-by-document or concept-by-document matrix, respectively. We cannot use the cosine measure $\mathrm{SIM}_{cos}(d_i, d_j)$ as a metric function directly, since it does not satisfy the metric axioms. As an appropriate metric, we define the *deviation metric* $d_{dev}(d_i, d_j)$ as a vector deviation

$$d_{dev}(d_i, d_j) = arccos(\mathrm{SIM}_{cos}(d_i, d_j))$$

The similarity queries supported by M-tree (utilizing d_{dev}) are exactly those required for the vector model (utilizing SIM_{cos}). Specifically, the range query will return all the documents that are similar to a query more than some given threshold (transformed to the query radius) while the k-NN query will return the first k most similar (closest respectively) documents to the query.

In the M-tree hierarchy similar documents are clustered among metric regions. Since the triangular inequality for d_{dev} is satisfied, many irrelevant document clusters can be safely pruned during a query processing, thus the search efficiency is improved.

3 Semi-Metric Search

In this section we propose the concept of semi-metric search – an approximate extension of metric search applied to M-tree. The semi-metric search provides even more efficient retrieval, considerably resistant to the curse of dimensionality.

3.1 Curse of Dimensionality

The metric indexing itself (as is experimentally verified in Section 4) is beneficial for searching in the LSI model. However, searching in a collection of high-dimensional document vectors of the classic vector model is negatively affected by a phenomenon called *curse of dimensionality* [7, 8]. In the M-tree hierarchy (even the most optimal hierarchy) the curse of dimensionality causes that clusters of high-dimensional vectors are not distinct, which is reflected by huge overlaps among metric regions.

Intrinsic Dimensionality. In the context of metric indexing, the curse of dimensionality can be generalized for general metric spaces. The major condition determining the success of metric access methods is the *intrinsic dimensionality* of the indexed dataset. The intrinsic dimensionality of a metric dataset (one of the interpretations [8]) is defined as

$$\rho = \frac{\mu^2}{2\sigma^2}$$

where μ and σ^2 are the mean and the variance of the dataset's *distance distribution histogram*. In other words, if all pairs of the indexed objects are almost

equally distant, then the intrinsic dimensionality is maximal (i.e. the mean is high and/or the variance is low), which means the dataset is poorly intrinsically structured. So far, for datasets of high intrinsic dimensionality there still does not exist an efficient MAM for exact metric search. In case of M-tree, a high intrinsic dimensionality causes that almost all the metric regions overlap each other, and searching in such an M-tree deteriorates to sequential search.

In case of vector datasets, the intrinsic dimensionality negatively depends on the correlations among coordinates of the dataset vectors. The intrinsic dimensionality can reach up to the value of the classic (embedding) dimensionality. For example, for uniformly distributed (i.e. not correlated) n-dimensional vectors the intrinsic dimensionality tends to be maximal, i.e. $\rho \approx n$.

In the following section we propose a concept of semi-metric modifications that decrease the intrinsic dimensionality and, as a consequence, provide a way to efficient approximate similarity search.

3.2 Modification of the Metric

An increase of the variance of distance distribution histogram is a straightforward way how to decrease the intrinsic dimensionality. This can be achieved by a suitable modification of the original metric, preserving the similarity ordering among objects in the query result.

Definition 1. Let us call the *increasing modification* d^f_{dev} of a metric d_{dev} a function

$$d^f_{dev}(O_i, O_j) = f(d_{dev}(O_i, O_j))$$

where $f : \langle 0, \pi\rangle \rightarrow R^+_0$ is an increasing function and $f(0) = 0$. For simplicity, let $f(\pi) = 1$.

Definition 2. Let $s : \mathbb{U} \times \mathbb{U} \rightarrow R^+_0$ be a similarity function (or a distance function) and $SimOrder_s : \mathbb{U} \rightarrow \mathcal{P}(\mathbb{S} \times \mathbb{S})$ be a function defined as

$$\langle O_i, O_j\rangle \in SimOrder_s(Q) \Leftrightarrow s(O_i, Q) < s(O_j, Q)$$

$\forall O_i, O_j \in \mathbb{S}, \forall Q \in \mathbb{U}$. In other words, the function $SimOrder_s$ orders the objects of dataset $\mathbb{S}$ according to the distances to the query object Q.

Proposition. For the metric d_{dev} and every increasing modification d^f_{dev} the following equality holds:

$$SimOrder_{d_{dev}}(Q) = SimOrder_{d^f_{dev}}(Q), \forall Q \in \mathbb{U}$$

Proof:
"$\subset$": The function f is increasing. If for each $O_i, O_j, O_k, O_l \in \mathbb{U}$, $d_{dev}(O_i, O_j) > d_{dev}(O_k, O_l)$ holds, then $f(d_{dev}(O_i, O_j)) > f(d_{dev}(O_k, O_l))$ must also hold.
"$\supset$": The second part of proof is similar. □

As a consequence of the proposition, if we process a query sequentially over the entire dataset $\mathbb{S}$, then it does not matter if we use either d_{dev} or d^f_{dev}, since both of the ways will return the same query result.

If the function f is additionally *subadditive*, i.e. $f(a) + f(b) \geq f(a+b)$, then f is *metric-preserving* [10], i.e. $f(d(O_i, O_j))$ is still metric. More specifically, concave functions are metric-preserving (see Figure 2a), while convex (even partially convex) functions are not – let us call them *metric-violating* functions (see Figure 2b). A metric modified by a metric-violating function f is a *semi-metric*, i.e. a function satisfying all the metric axioms except the triangular inequality.

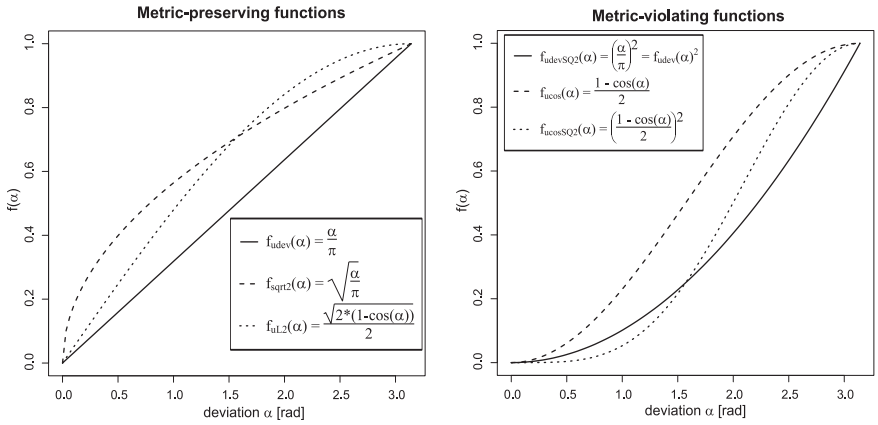


Fig. 2. (a) Metric-preserving functions (b) Metric-violating functions

Clustering Properties. Let us analyze the clustering properties of modifications d^f_{dev} (see also Figure 2). For concave f, two objects close to each other according to d_{dev} are more distant according to d^f_{dev}. Conversely, for convex f, the close objects according to d_{dev} are even closer according to d^f_{dev}. As a consequence, the concave modifications d^f_{dev} have a negative influence on clustering, since the object clusters become indistinct. On the other side, the convex modifications d^f_{dev} even more tighten the object clusters, making the cluster structure of the dataset more evident. Simply, the convex modifications increase the distance histogram variance, thereby decreasing the intrinsic dimensionality.

3.3 Semi-Metric Indexing and Search

The increasing modifications d^f_{dev} can be utilized in the M-tree instead of the deviation metric d_{dev}. In case of a semi-metric modification d^f_{dev}, the query processing is more efficient because of smaller overlaps among metric regions in the M-tree. Usage of metric modifications is not beneficial, since their clustering properties are worsen, and the overlaps among metric regions are larger.

Semi-Metric Search. A semi-metric modification d^f_{dev} can be used for all operations on the M-tree, i.e. for M-tree building as well as for M-tree searching. With respect to M-tree construction principles (we refer to [21]) and the proposition in Section 3.2, the M-tree hierarchies built either by d or d^f_{dev} are the

same. For that reason, an M-tree built using a metric d can be queried using any modification d^f_{dev}. Such *semi-metric queries* must be extended by the function f, which stands for an additional parameter. For a range query the query radius r_Q must be modified to $f(r_Q)$. During a semi-metric query processing, the function f is applied to each value computed using d as well as it is applied to the metric region radii stored in the routing entries.

Error of the Semi-Metric Search. Since the semi-metric d^f_{dev} does not satisfy the triangular inequality property, a semi-metric query will return more or less approximate results. Obviously, the error is dependent on the convexity of a modifying function f. As an output error, we define a *normed overlap error*

$$E_{NO} = 1 - \frac{|result_{Mtree} \cap result_{scan}|}{max(|result_{Mtree}|, |result_{scan}|)}$$

where $result_{Mtree}$ is a query result returned by the M-tree (using a semi-metric query), and $result_{scan}$ is a result of the same query returned by sequential search over the entire dataset. The error E_{NO} can be interpreted as a *relative precision* of the M-tree query result with respect to the result of full sequential scan.

Semi-Metric Search in Text Retrieval. In the context of TR, the searching is naturally approximate, since precision/recall values do never reach up to 100%. From this point of view, the approximate character of semi-metric search is not a serious limitation – acceptable results can be achieved by choosing such a modifying function f, for which the error E_{NO} will not exceed some small value, e.g. 0.1. On the other side, semi-metric search significantly improves the search efficiency, as it is experimentally verified in the following section.

4 Experimental Results

For the experiments we have chosen the Los Angeles Times collection (a part of TREC 5) consisting of 131,780 newspaper articles. The entire collection contained 240,703 unique terms. As "rich" many-term queries, we have used articles consisting of at least 1000 unique terms. The experiments were focused on disk access costs (DAC) spent during k-NN queries processing. Each k-NN query was repeated for 100 different query documents and the results were averaged. The access to disk was aligned to 512B blocks, considering both access to the M-tree index as well as to the respective matrix. The overall query DAC are presented in megabytes. The entries of M-tree nodes have contained just the document vector identifiers (i.e. pointers to the matrix columns), thus the M-tree storage volume was minimized. In Table 1 the M-tree configuration used for experiments is presented (for a more detailed description see [21]).

The labels of form `Dev`xxx in the figures below stand for modifying functions f used by semi-metric search. Several functions of form `DevSQ`$p(\alpha) = \left(\frac{\alpha}{\pi}\right)^p$ were chosen. The queries labeled as `Dev` represent the original metric queries presented in Section 2.2.

Table 1. The M-tree configuration

Page size: 512 B; Capacity (leaves: 42, nodes: 21)
Construction: `MinMax` + `SingleWay` + `SlimDown`
Tree height: 4; Avg. util. (leaves: 56%, nodes: 52%)

4.1 Classic Vector Model

First, we performed tests for the classic vector model. The storage of the term-by-document matrix (in CCS format [4]) took 220 MB. The storage of M-tree index was about 4MB (i.e. 1.8% of the matrix storage volume (MSV)).

In Figure 3a the comparison of document vector scanning, term vector filtering as well as metric and semi-metric search is presented. It is obvious that using document vector scanning the whole matrix (i.e. 220 MB DAC) was loaded and processed. Since the query vectors contained many zero weights, the term vector filtering worked more efficiently (76 MB DAC, i.e. 34% of MSV).

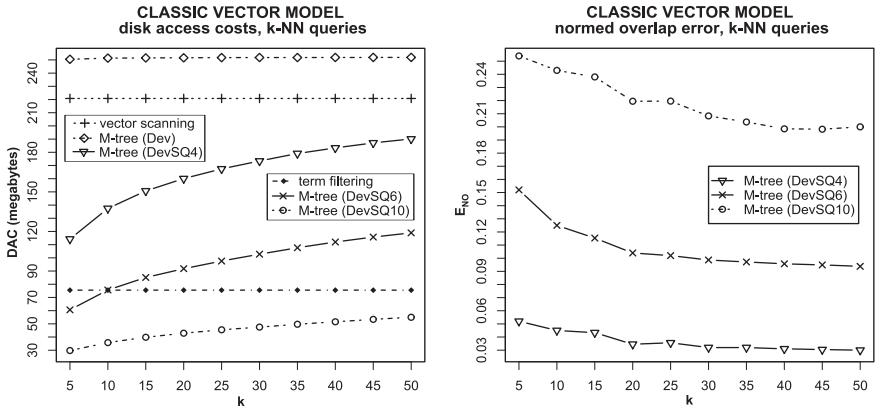


Fig. 3. Classic vector model: (a) Disk access costs (b) E_{NO} error

The metric search `Dev` did not performed well – the curse of dimensionality (n = 240,703) forced almost 100% of the matrix to be processed. The extra 30 MB DAC overhead (beyond the 220 MB of MSV) was caused by the non-sequential access to the matrix columns. On the other side, the semi-metric search performed better. The `DevSQ10` queries for $k = 5$ consumed only 30 MB DAC (i.e. 13.6% of MSV). Figure 3b shows the normed overlap error E_{NO} of the semi-metric search. For `DevSQ4` queries the error was negligible. The error for `DevSQ6` remained below 0.1 for $k > 35$. The `DevSQ10` queries were affected by a relatively high error from 0.25 to 0.2 (with increasing k).

4.2 LSI Model

The second set of tests was made for the LSI model. The target (reduced) dimensionality was chosen to be 200. The storage of the concept-by-document matrix took 105 MB, while the size of M-tree index was about 3 MB (i.e. 2.9 % of MSV).

Because the size of term-by-document matrix was very large, the direct calculation of SVD was impossible. Therefore, we have used a two-step method [17], which in first step calculates a *random projection* [1, 5] of document vectors into a smaller dimensionality of *pseudo-concepts*. This is done by multiplication of a zero-mean unit-variance random matrix and the term-by-document matrix. Second, a rank-$2k$ SVD is calculated on the resulting *pseudoconcept-by-document* matrix, giving us a very good approximation of the classic rank-k SVD.

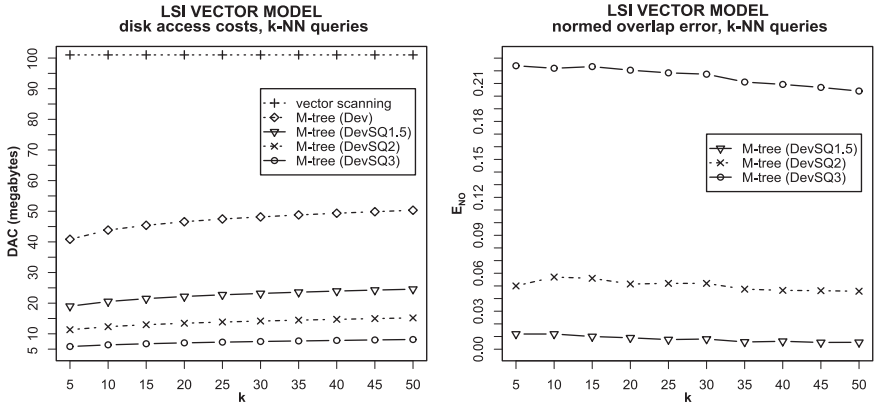


Fig. 4. LSI model: (a) Disk access costs (b) E_{NO} error

The Figure 4a shows that metric search `Dev` itself was more than twice as efficient as the document vector scanning. Even better results were achieved by the semi-metric search. The `DevSQ3` queries for $k = 5$ consumed only 5.8 MB DAC (i.e. 5.5% of MSV). Figure 4b shows the error E_{NO}. For `DevSQ1.5` queries the error was negligible, for `DevSQ2` it remained below 0.06. The `DevSQ3` queries were affected by a relatively high error.

5 Conclusion

In this paper we have proposed a metric indexing method for an efficient search of documents in the vector model. The experiments have shown that metric indexing itself is suitable for an efficient search in the LSI model. Furthermore, the approximate semi-metric search allows us to provide quite efficient similarity search in the classic vector model, and a remarkably efficient search in the LSI model. The output error of semi-metric search can be effectively tuned by choosing such modifying functions, that preserve an expected accuracy sufficiently.

In the future we would like to compare the semi-metric search with some other methods, in particular with the VA-file (in case of LSI model). We also plan to develop an analytical error model for the semi-metric search in M-tree, allowing to predict and control the output error E_{NO}.

This research has been partially supported by GAČR grant No. 201/00/1031.

References

1. D. Achlioptas. Database-friendly random projections. In *Symposium on Principles of Database Systems*, 2001.
2. V. N. Anh, O. de Kretser, and A. Moffat. Vector-space ranking with effective early termination. In *Proceedings of the 24th annual international ACM SIGIR*, pages 35–42. ACM Press, 2001.
3. R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval*. Addison Wesley, New York, 1999.
4. M. Berry and M. Browne. *Understanding Search Engines, Mathematical Modeling and Text Retrieval*. Siam, 1999.
5. E. Bingham and H. Mannila. Random projection in dimensionality reduction: applications to image and text data. In *Knowledge Discovery and Data Mining*, pages 245–250, 2001.
6. S. Blott and R. Weber. An Approximation-Based Data Structure for Similarity Search. Technical report, ESPRIT, 1999.
7. C. Böhm, S. Berchtold, and D. Keim. Searching in High-Dimensional Spaces – Index Structures for Improving the Performance of Multimedia Databases. *ACM Computing Surveys*, 33(3):322–373, 2001.
8. E. Chávez and G. Navarro. A probabilistic spell for the curse of dimensionality. In *Proc. 3rd Workshop on Algorithm Engineering and Experiments (ALENEX'01), LNCS 2153*. Springer-Verlag, 2001.
9. P. Ciaccia, M. Patella, and P. Zezula. M-tree: An Efficient Access Method for Similarity Search in Metric Spaces. In *Proceedings of the 23rd Athens Intern. Conf. on VLDB*, pages 426–435. Morgan Kaufmann, 1997.
10. P. Corazza. Introduction to metric-preserving functions. *Amer. Math Monthly*, 104(4):309–23, 1999.
11. S. C. Deerwester, S. T. Dumais, T. K. Landauer, G. W. Furnas, and R. A. Harshman. Indexing by latent semantic analysis. *Journal of the American Society of Information Science*, 41(6):391–407, 1990.
12. U. Deppisch. S-tree: A Dynamic Balanced Signature Index for Office Retrieval. In *Proceedings of ACM SIGIR*, 1986.
13. C. Faloutsos. Signature-based text retrieval methods, a survey. *IEEE Computer society Technical Committee on Data Engineering*, 13(1):25–32, 1990.
14. D. L. Lee and L. Ren. Document Ranking on Weight-Partitioned Signature Files. In *ACM TOIS 14*, pages 109–137, 1996.
15. A. Moffat and J. Zobel. Fast ranking in limited space. In *Proceedings of ICDE 94*, pages 428–437. IEEE Computer Society, 1994.
16. P. Moravec, J. Pokorný, and V. Snášel. Vector Query with Signature Filtering. In *Proc. of the 6th Bussiness Information Systems Conference, USA*, 2003.
17. C. H. Papadimitriou, H. Tamaki, P. Raghavan, and S. Vempala. Latent semantic indexing: A probabilistic analysis. In *Proocedings of the ACM Conference on Principles of Database Systems (PODS), Seattle*, pages 159–168, 1998.
18. M. Patella. *Similarity Search in Multimedia Databases*. Dipartmento di Elettronica Informatica e Sistemistica, Bologna, 1999.
19. M. Persin. Document filtering for fast ranking. In *Proceedings of the 17th annual international ACM SIGIR*, pages 339–348. Springer-Verlag New York, Inc., 1994.
20. G. Salton and M. McGill. *Introduction to Modern Information Retrieval*. McGraw Hill Publications, 1st edition, 1983.
21. T. Skopal, J. Pokorný, M. Krátký, and V. Snášel. Revisiting M-tree Building Principles. In *ADBIS 2003, LNCS 2798, Springer, Dresden, Germany*, 2003.

Chapter 8

Modified LSI Model for Efficient Search by Metric Access Methods

Tomáš Skopal
Pavel Moravec

Modified LSI Model for Efficient Search by Metric Access Methods [44]

Regular paper at the 27th European Conference on IR Research (ECIR 2005), Santiago de Compostela, Spain, March 2005

Published in the *Lecture Notes in Computer Science* (LNCS), vol. 3408, pages 245–259, *Springer-Verlag*, ISSN 0302-9743, ISBN 978-3-540-25295-5

Modified LSI Model for Efficient Search by Metric Access Methods

Tomáš Skopal[1] and Pavel Moravec[2]

[1]Charles University in Prague, FMP, Department of Software Engineering
Malostranské nám. 25, 118 00 Prague, Czech Republic
tomas@skopal.net
[2]Technical University of Ostrava, FEECS, Department of Computer Science
17. listopadu 15, 708 33 Ostrava, Czech Republic
pavel.moravec@vsb.cz

Abstract. Text collections represented in LSI model are hard to search efficiently (i.e. quickly), since there exists no indexing method for the LSI matrices. The inverted file, often used in both boolean and classic vector model, cannot be effectively utilized, because query vectors in LSI model are dense. A possible way for efficient search in LSI matrices could be the usage of metric access methods (MAMs). Instead of cosine measure, the MAMs can utilize the deviation metric for query processing as an equivalent dissimilarity measure. However, the intrinsic dimensionality of collections represented by LSI matrices is often large, which decreases MAMs' performance in searching. In this paper we introduce σ-LSI, a modification of LSI in which we artificially decrease the intrinsic dimensionality of LSI matrices. This is achieved by an adjustment of singular values produced by SVD. We show that suitable adjustments could dramatically improve the efficiency when searching by MAMs, while the precision/recall values remain preserved or get only slightly worse.

1 Introduction

Text collections represented in the classic vector model (CVM) can be efficiently (i.e. quickly) searched using the inverted file. More precisely, the inverted file provides a way for very efficient processing of queries, the vectors of which are sparse (such a query contains only several terms). However, in case of LSI model the query vectors are dense, and the usage of inverted file becomes useless, since processing of any query deteriorates to sequential search over the entire concept-by-document matrix.

In this paper we utilize a method of searching in LSI collections by metric access methods (MAMs). The metric access methods are, however, sensitive to the curse of dimensionality, i.e. they become inefficient for high dimensionalities. Therefore, in this paper we propose σ-LSI, a modified LSI model in which we artificially reduce the intrinsic dimensionality of the indexed collection. This is achieved by an adjustment of singular values produced by SVD. We show that suitable adjustments could dramatically improve the efficiency when searching

by MAMs, while the precision/recall values remain preserved or get only slightly worse.

The paper is organized as follows: In the rest of this section we briefly overview CVM, the LSI model, and formulate the problem of searching in LSI model. In Section 3 we show how the classic similarity search in CVM (LSI model respectively) can be turned into metric search. We also mention the principles of metric access methods and the problem of high intrinsic dimensionality. In Section 4 we propose σ-LSI model allowing a more efficient search by MAMs. The *effectiveness* (the quality) and *efficiency* (the response time) of retrieval in the σ-LSI model are evaluated in Section 5.

1.1 Classic Vector Model

In CVM, a given text collection (containing n documents consisting of m unique terms) is represented by an $m \times n$ *term-by-document matrix* A, where each column vector d_j in A represents a single document D_j. Thus, the documents are represented as points in m-dimensional vector space (the *document-space*). Each dimension of the document-space is associated with a single term, while each coordinate in a *document vector* d_j represents a weight of the respective term in the document. There are many ways how to compute the term weights A_{ij} – a popular weight construction is computed as $tf \cdot idf$ (see e.g. [3]).

term \ doc.	D_1	D_2	D_3	D_4	D_5
database	0	0.48	0.05	0	0.70
vector	0.23	0	0.23	0	0
index	0.43	0	0	0	0
image	0	0	0.10	0	0.54
compression	0	0	0	0	0.21
multimedia	0.12	0.52	0.62	0	0

Fig. 1. Term-by-document matrix A.

The most important part of CVM is the query semantics for searching the matrix A with respect to a query Q, and returning only the relevant document vectors (appropriate documents respectively). The query Q is represented by a vector q in the document space the same way as a document D_j is represented by d_j. The goal is to return the most similar documents to the query. For this purpose a similarity measure must be defined, assessing a similarity score for each pair of query and document vectors (q, d_j). In many cases, the *cosine measure*

$$\mathrm{SIM}_{cos}(q, d_j) = \frac{\sum_{i=1}^{m} q_i d_{ji}}{\sqrt{\sum_{i=1}^{m} {q_i}^2 \cdot \sum_{i=1}^{m} {d_{ji}}^2}}$$

is widely used. Besides the simple ranking to q (used for *ranked lists*), we also distinguish bounded queries, in particular *range queries* and *k-nearest neighbors*

(k-NN) queries. A range query returns documents with similarity to the query higher than a given similarity threshold t. A k-NN query returns the k most similar documents[1].

2 Latent Semantic Indexing

Latent semantic indexing (*LSI*) [3, 4] is an algebraic extension of CVM. Its benefits rely on discovering *latent semantics* hidden in the term-by-document matrix A. Informally, LSI discovers significant groups of terms (called *concepts*) and represents the documents as linear combinations of the concepts. Moreover, the concepts are ordered according to their significance in the collection, which allows us to consider only the first k concepts important (the remaining ones are interpreted as "noise" and discarded). To name the advantages, LSI helps solve problems with synonymy and homonymy. Furthermore, LSI is often referred to as more successful in recall when compared to CVM [4], which was proved for pure (only one topic per document) and style-free collections [17].

Formally, we decompose the term-by-document matrix A by *singular value decomposition* (*SVD*), calculating singular values and singular vectors of A. SVD is especially suitable in its variant for sparse matrices (Lanczos [13]). Several approximate methods for faster SVD calculation were offered recently, such as using random projection of document vectors into suitable subspace before LSI calculation [17] or application of Monte-Carlo method [11].

There are several other methods for latent semantic indexing, such as ULV-decomposition [5], random indexing [16] (and some other approaches achieving similar goals, e.g. language modeling [19]), which we do not discuss in this paper.

Theorem 1 (Singular value decomposition [4]). *Let A is an $m \times n$ rank-r matrix. Be values $\sigma_1, \ldots, \sigma_r$ calculated from eigenvalues of matrix AA^T as $\sigma_i = \sqrt{\lambda_i}$. Then there exist column-orthonormal matrices $U = (u_1, \ldots, u_r)$ and $V = (v_1, \ldots, v_r)$, where $U^T U = I_m$ a $V^T V = I_n$, and a diagonal matrix $\Sigma = diag(\sigma_1, \ldots, \sigma_r)$, where $\sigma_i > 0, \sigma_i \geq \sigma_{i+1}$. The decomposition*

$$A = U \Sigma V^T$$

is called singular decomposition *of matrix A and the numbers $\sigma_1, \ldots, \sigma_r$ are* singular values *of the matrix A. Columns of U (or V) are called* left (or right) singular vectors *of matrix A.*

Now we have a decomposition of the original term-by-document matrix A. The left and right singular vectors (i.e. U and V matrices) are not sparse. We get r nonzero singular numbers, where r is the rank of the original matrix A. Because the singular values usually fall quickly, we can take only k greatest singular values with the corresponding singular vector coordinates and create a *k-reduced singular decomposition* of A.

[1] In the next section we independently use k for another parameter (rank-k SVD), but in either case the respective meaning of k is obvious from the actual context.

Definition 1. *Let us have* k $(0 < k < r)$ *and singular value decomposition of* A

$$A = U\Sigma V^T \approx A_k = (U_k U_0) \begin{pmatrix} \Sigma_k & 0 \\ 0 & \Sigma_0 \end{pmatrix} \begin{pmatrix} V_k^T \\ V_0^T \end{pmatrix}$$

We call $A_k = U_k \Sigma_k V_k^T$ *a* k*-reduced singular value decomposition (rank-*k *SVD).*

Instead of the A_k matrix, a *concept-by-document matrix* $D_k = \Sigma_k V_k^T$ is used in LSI as the representation of document collection. The document vectors (columns in D_k) are now represented as points in k-dimensional space (the *pseudodocument-space*). For an illustration of rank-k SVD see Figure 2.

The value of k was experimentally determined as several tens or hundreds (e.g. 50–250), however, the optimal[2] value of k is hard to choose; it is dependent on the number of topics in collection. Rank-k SVD is the best rank-k approximation of the original matrix A, regarding to Frobenius norm (see e.g. [12]). This means, that any other decomposition will increase the sum of squares of matrix $A - A_k$. However, this does not tell us that we could not obtain better precision and recall values with a different approximation.

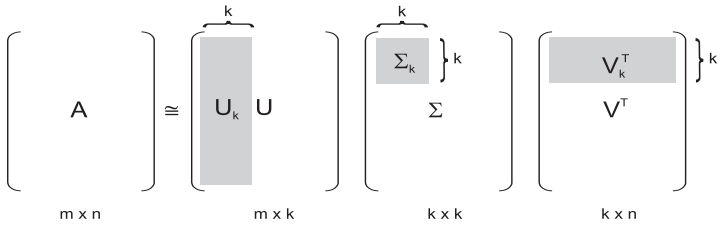


Fig. 2. k-reduced singular value decomposition

To execute a query Q in the pseudodocument-space, we create a reduced query vector $q_k = U_k^T q$ (another approach is to simply use a matrix $D'_k = V_k^T$ instead of D_k, and $q'_k = \Sigma_k^{-1} U_k^T q$). Instead of A against q, the matrix D_k against q_k (or q'_k) is evaluated using the cosine measure. The crucial property is that, due to the projection by dense matrix U_k^T, q_k is dense as well (even if q is sparse).

2.1 LSI model and inverted files

In CVM, searching the term-by-document matrix A according to a query Q can be provided using *inverted file* [15, 18, 1], which can be viewed as the matrix A stored by rows. For a given matrix A the inverted file consists of m lists, each list is associated with a single term. Each list stores entries, which are pairs consisting of a document id and weight of the term in corresponding document

[2] optimal in sense of best achieved precision/recall values

(obviously, entries with zero weights are not stored). When a query is processed, only the lists representing terms from the query are sequentially searched.

The inverted file is very efficient for processing of sparse query vectors (few-term queries respectively), because only several lists have to be processed. Unfortunately, in case of LSI the pseudo-query vector is dense and usage of inverted file for indexing D_k would deteriorate to sequential search over the entire file and thus, over the entire matrix D_k.

3 Metric Indexing

Recently, there has been introduced an approach to searching in LSI model, based on *metric indexing* [20]. Instead of inverted file, the *M-tree* [9] was used for indexing the matrix D_k. Before we discuss benefits of the metric approach, we must turn the cosine measure (similarity) into metric (distance).

3.1 Turning Vector Model into Metric Model

The cosine measure $\mathrm{SIM}_{cos}(d_i, d_j)$ itself is not a metric, since it does not satisfy three metric properties (reflexivity, positivity and triangular inequality). Even $1-\mathrm{SIM}_{cos}(d_i, d_j)$ is not a metric, since it does not satisfy the triangular inequality. As an appropriate metric, we use the *deviation metric* (or angular distance) $d_{dev}(d_i, d_j)$, defined as

$$d_{dev}(d_i, d_j) = arccos(\mathrm{SIM}_{cos}(d_i, d_j))$$

Instead of cosine, the deviation metric measures directly the angle between two vectors[3]. Since *arccos* is strictly decreasing on $\langle -1, 1\rangle$, the deviation metric preserves the semantic meaning of cosine measure. There is only a difference in terminology – cosine measure is *similarity function* (similar documents have a high score), while the deviation metric is *dissimilarity function* (similar documents have a lower score, i.e. they are close). Hence, the k-dimensional pseudodocument-space $\mathbb{R}^k$ together with the deviation metric d_{dev} can be regarded as a metric space $\mathcal{M} = (\mathbb{R}^k, d_{dev})$.

The queries in metric model are evaluated in similar way as in CVM; the difference is that range queries select objects within a *query radius* r_Q (which equals to *arccos* of the desired similarity threshold t), while k-NN queries select the k closest objects.

3.2 Metric Access Methods

The *metric access methods* [8] organize (or index) a given metric dataset $\mathbb{S} \subset \mathcal{M}$ in a way that metric queries (e.g. range or k-NN queries) can be processed efficiently – without a need of processing the entire dataset $\mathbb{S}$. The main principle

[3] Actually, we can view the deviation metric d_{dev} as a kind of Euclidean (L_2) distance, defined just on the surface of unitary hyper-sphere.

behind all MAMs is the triangular inequality property satisfied by every metric. Due to the triangular inequality, MAMs can organize the objects in equivalence classes (the classes are some regions in the metric space). When a query is processed, many irrelevant equivalence classes are filtered (those with metric regions not overlapping the query region), and so the searching becomes more efficient. Another advantage is that MAMs use solely the metric function for indexing, no information about the indexed objects representation is necessary. This feature allows to index/search non-vectorial datasets, too.

There has been developed a plenty of MAMs, varying in applicability to different problems. Besides others, we name *M-tree* [9], *vp-tree* [22], *LAESA* [14], *D-index* [10], etc.

3.3 Intrinsic Dimensionality

The metric indexing itself (as was presented in [20]) could be quite beneficial for searching in the LSI model. However, searching in a collection of high-dimensional document vectors is negatively affected by a phenomenon called the *curse of dimensionality* [6, 7]. For MAMs the curse of dimensionality causes almost all equivalence classes to be overlapped by nearly every "reasonable" query region, so that searching deteriorates to sequential scan over all the classes.

In the context of metric indexing, the curse of dimensionality can be generalized for general metric spaces. The major condition determining the efficiency limits of any metric access method is the *intrinsic dimensionality* of the indexed dataset, defined as (proposed in [7]):

$$\rho(\mathbb{S}, d) = \frac{\mu^2}{2\sigma^2}$$

where μ and σ^2 are the mean and the variance of the dataset's *distance distribution* (according to a metric d). In other words, the intrinsic dimensionality is low if there exist tight clusters of objects. Conversely, if all pairs of the indexed objects are almost equally distant, the intrinsic dimensionality is high (i.e. the mean is high and/or the variance is low), which means the dataset is poorly intrinsically structured. In Figure 3 see an example of distance distribution histograms (DDHs) indicating lower ($\rho \approx 2$) and higher ($\rho \approx 30$) intrinsic dimensionalities.

In case of vector datasets, the intrinsic dimensionality can reach up to (or even beyond) the value of the classic (embedding) dimensionality. For example, for uniformly distributed n-dimensional vectors (i.e. not clustered) $\rho \approx n$.

So far, for datasets of high intrinsic dimensionality there still does not exist an efficient MAM for exact[4] metric search.

[4] Nevertheless, efficient searching in high-dimensional datasets can be realized by approximate or probabilistic MAMs, but such methods often suffer from lower precision/recall values [23, 7].

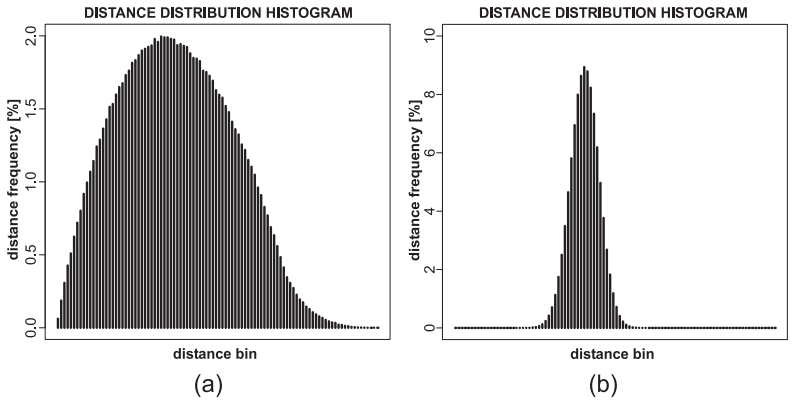


Fig. 3. DDHs indicating (a) low (b) high intrinsic dimensionality

4 The σ-LSI Model

In case of LSI, we are concerned by intrinsic dimensionality of the pseudodocument vectors (columns in D_k), with respect to the deviation metric d_{dev}. The smaller ρ, the greater search efficiency can be achieved for the MAMs.

In this section we propose the σ-LSI model, a modification of LSI in which we are able to artificially decrease the intrinsic dimensionality of D_k.

4.1 Motivation

In order to understand the intrinsic dimensionality of D_k, we first consider the simpler approach of LSI, where the pseudodocument matrix is just $D'_k = V_k^T$ (instead of $D_k = \Sigma_k V_k^T$). This is equivalent to $D'_k = \Sigma_k^0 V_k^T$, where Σ_k^0 is unitary matrix (the singular values σ_i are powered by 0). To illustrate the situation on an example, we use a term-by-document matrix A (closely described in Section 5) decomposed using rank-k SVD, $k = 100$.

In Figure 4a see the DDH for columns in D'_k with respect to d_{dev}. The intrinsic dimensionality is $\rho = 98.1$, so we can claim that in this case $k \approx \rho$. This interesting observation arises from the fact that rows in V_k^T are orthonormal and columns in V_k^T (the pseudodocument vectors) are (almost) uniformly distributed.

Second, we consider the pseudodocument matrix $D_k = \Sigma_k V_k^T$ (the classic LSI). In Figure 4b see the DDH for columns in D_k with respect to d_{dev}, the intrinsic dimensionality is now $\rho = 52.6$. Obviously, the difference between $\rho(D'_k, d_{dev})$ and $\rho(D_k, d_{dev})$ is in the multiplication of V_k^T by Σ_k. Since the singular values σ_i fall with increasing i, the uniformly distributed columns of V_k^T (i.e. D'_k) turn into non-uniformly distributed columns of $\Sigma_k V_k^T$ (i.e. D_k). Furthermore, multiplication with greater σ_i makes the i-th dimension (i-th concept resp.) more significant and vice versa. In consequence, only the most significant

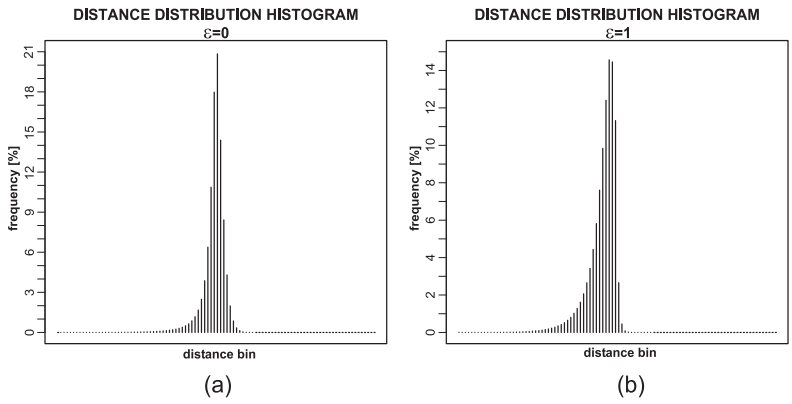


Fig. 4. (a) DDH for D'_k (b) DDH for D_k

dimensions can affect the spatial distribution of pseudodocument vectors; the small values in insignificant dimensions can "shift" the vectors only fractionally. Hence, the quicker falling of σ_i, the smaller number of significant dimensions and, in turn, the smaller intrinsic dimensionality of D_k.

4.2 Singular Values Modification

To decrease the intrinsic dimensionality of D_k, we can adjust the singular values σ_i such that they fall more quickly (with increasing i). This can be achieved by a suitable modifying function f.

$$\Sigma_k = diag(\sigma_1, \ldots, \sigma_k) \qquad \Longrightarrow \qquad \Sigma_k^f = diag(f(\sigma_1), \ldots, f(\sigma_k))$$

The function f must be increasing in order to preserve the ordering of singular values (they are ordered by values). Moreover, f must be convex, because we need to make the falling of σ_i faster (concave functions do the opposite).

Finally, we apply the modified values in Σ_k^f instead of the original Σ_k, i.e. we use $D_k^f = \Sigma_k^f V_k^T$ instead of D_k and $q_k^f = \Sigma_k^f \Sigma_k^{-1} U_k^T q$ instead of q_k.

In the following we have chosen functions $f(x) = x^\varepsilon$ ($\varepsilon \geq 1$), so we will denote Σ_k^f as Σ_k^ε, D_k^f as D_k^ε, and q_k^f as $q_k^\varepsilon = \Sigma_k^{\varepsilon-1} U_k^T q$. Note the notation is consistent with the simple LSI (i.e. usage of Σ_k^0). In Figure 5 see a normed visualization of the singular values modified by several functions $f(x) = x^\varepsilon$. The greater ε, the more quick falling of σ_i^ε.

From the semantic point of view, a convex modification of singular values means that we even more emphasize the significant concepts and even more inhibit the less significant ones. It seems that we perform a kind of an additional dimensionality reduction.

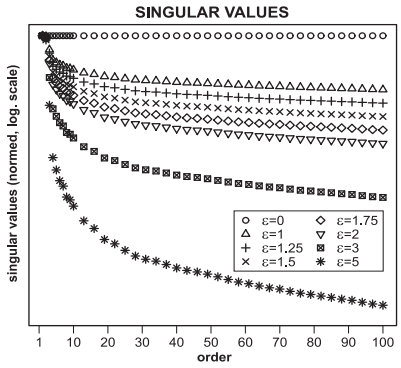


Fig. 5. Visualization of modified singular numbers σ_i^ε (for different ε)

On the other side, any modification of singular values surely must increase the approximation error mentioned in Section 2. However, this kind of error is algebraical; the human-dependent effectiveness measures (e.g. the precision and the recall) are something else. We present an experimental evaluation of the σ-LSI model effectiveness in Section 5.1.

4.3 Intrinsic Dimensionality Reduction

In Figure 6 see distance distribution histograms for D_k^ε, $\varepsilon = 1.5$ and $\varepsilon = 3$. The intrinsic dimensionality for $D_k^{1.5}$ (or D_k^3) is $\rho = 21.22$ ($\rho = 1.72$ respectively).

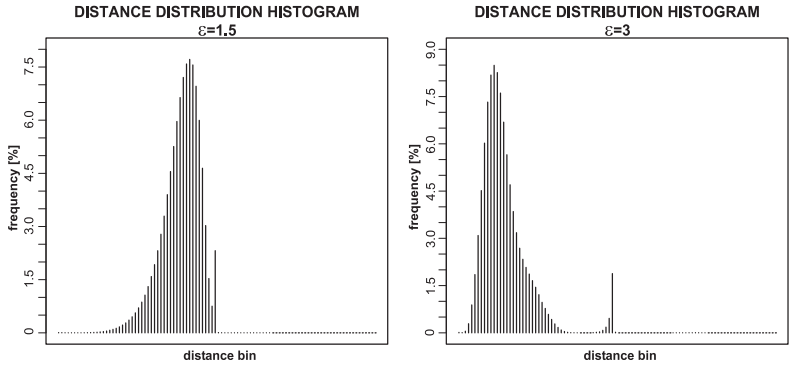


Fig. 6. DDHs for $D_k^{1.5}$ and D_k^3

In Figure 7 the intrinsic dimensionality ρ of D_k^ε is presented in dependence on ε. As we have assumed, ρ is decreasing with growing ε, which should be

reflected by a more efficient searching by MAMs. The search efficiency achieved by the M-tree is presented in Section 5.2.

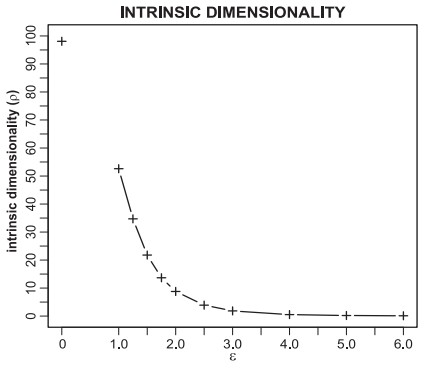


Fig. 7. Dependence of $\rho(D_k^{\varepsilon}, d_{dev})$ on ε

5 Experimental Query Evaluation

For testing of our approach, we used a subset of TREC collection [21], consisting of 30,000 Los Angeles Times articles (years 1989 and 1990), from which 16,889 articles were assessed in TREC-8 ad-hoc queries (see below). The remaining articles were added chronologically (from January to April 1989) and should provide finer LSI concepts. We indexed this collection, removing well-known stop-words and terms appearing in more than 25% of documents, thus obtaining 49,689 terms. Rank-100 SVD of the term-by-document matrix A was then calculated.

5.1 Effectiveness

For the evaluation of σ-LSI model, we need some qualitative measures for evaluating query results. We used *precision* (P) and *recall* (R), which are calculated from set *Rel* of objects relevant to the query (usually determined by manual annotation of the collection, giving us subjective human assessment of documents' relevance) and a set *Ret* of retrieved objects. Based on these sets, we define precision and recall as:

$$P = \frac{|Rel \cap Ret|}{|Ret|}, \quad R = \frac{|Rel \cap Ret|}{|Rel|}$$

For the overall comparison of precision and recall across different methods, we can use rank lists and evaluate precision on 11 standard recall levels (0.0, 0.1, 0.2, ..., 0.9, 1.0). Since the queries may have different number of relevant documents, we can use interpolated values for each query. For complete description of this method, see e.g. [2].

Unfortunately, it was observed that with the increase of recall, the precision usually decreases. This means that when it is necessary to retrieve more relevant objects, a higher percentage of irrelevant will be probably retrieved, too. To obtain a single ratio for evaluation of the retrieval performance, we can employ a measure called F-score – harmonic mean of recall and precision. Determination of the maximum value for F can be interpreted as an attempt to find the best possible compromise between recall and precision.

The universal version of F-score employs a coefficient β, by which can be the precision-recall ratio tuned. We will use the basic form of F score with $\beta = 1$:

$$F_\beta = \frac{(1+\beta^2) \cdot P \cdot R}{\beta^2 P + R}, \quad F = F_1 = \frac{2 \cdot P \cdot R}{P + R}$$

To measure the effectiveness of σ-LSI, we must know the values of precision and recall for both the original method (LSI) and the modification (σ-LSI). Since we use a subset of TREC collection, we have a baseline for the effectiveness measurement via a set of predefined topics and assessed documents, called TREC Queries. TREC topics (written in SGML) contain at least the following tags:

```
<top>
   <num> Number: 401
   <title> foreign minorities, Germany
   <desc> Description:
      What language and cultural differences impede the
      integration of foreign minorities in Germany?
   <narr> Narrative:
      A relevant document will focus on ...
</top>
```

For every topic, there is a set of relevance assessments for selected documents, which indicates, whether the particular assessed document was relevant or irrelevant. The remaining unassessed documents were *assumed* irrelevant.

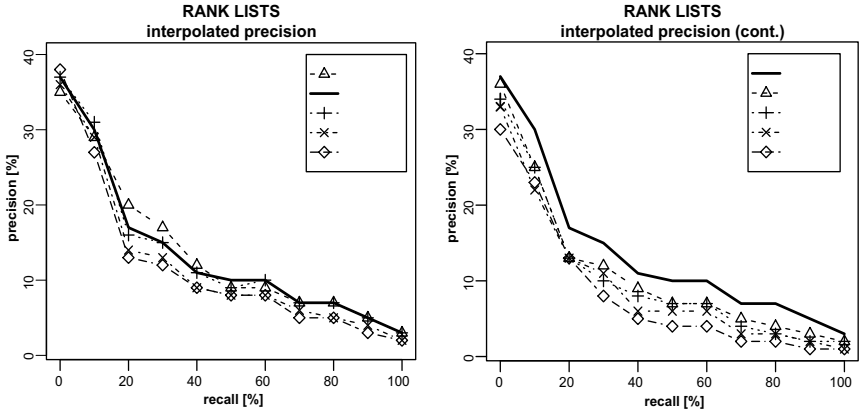


Fig. 8. Precision for 11 standard recall levels calculated from rank lists

We used TREC-8 Ad-hoc topics 401-450 with their relevance assessments for Los Angeles Times subcollection for our task. Term weights in query vectors were calculated from term frequency (tf) component, the query vectors were then projected to pseudodocument space for given ε. The values of ε have been chosen from $\{0\} \cup < 1, 9 >^5$. The cosine measure SIM_{cos} (deviation metric d_{dev} respectively) values were calculated for both k-NN queries and rank lists for each TREC Query in the pseudodocument spaces.

Firstly, we used rank lists and measured interpolated average precision of the above mentioned TREC Queries for 11 standard recall levels. The comparison for different values of ε and original LSI ($\varepsilon = 1$) is addressed in Figure 8. The precision-recall curves for reasonably small values of ε are very similar to classic LSI, thus the method yields similar results even with much smaller intrinsic dimensionality, which is suitable for MAMs.

Additionally, we calculated the mean average precision for all relevant documents in rank lists. The results for σ-LSI are shown in Figure 9a together with the mean average precision of corresponding CVM representation.

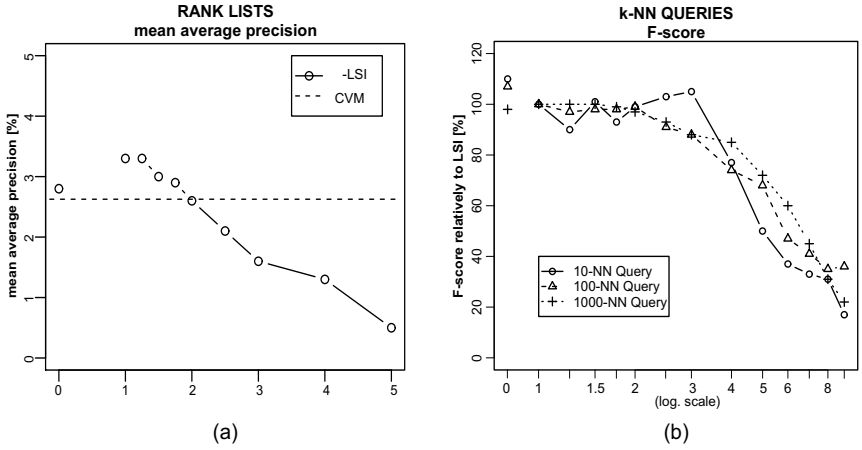


Fig. 9. (a) Mean average precision of σ-LSI for all relevant documents for different values of ε with CVM baseline (b) F-score of k-NN queries for different values of ε

Secondly, we executed TREC Queries as k-NN queries for several values of k, ranging from 10 to 1000 and compared the F-score for different values of ε. Some of the results are shown in Figure 9b. We can observe, for the values of $\varepsilon < 3$ the precision and F-score seem to be well-preserved.

[5] For $\varepsilon = 1$, we obtain classic LSI model with $D_k = \Sigma_k V_k^T$, which we used as a baseline; for $\varepsilon = 0$ we get simple LSI with $D'_k = V_k^T$.

5.2 Efficiency

The motivation and main reason for introduction of the σ-LSI model is an improvement of query evaluation efficiency, when using MAMs. Among the many metric access methods, we have chosen the M-tree [9] as a "database-friendly" MAM (M-tree is a balanced, paged and dynamic structure), which we employed to index several D_k^ε matrices. The matrices were stored externally (the M-tree index contained just pointers to the respective vectors in D_k^ε) and size of each matrix was about 12 MB. The size of each M-tree index was quite small, about 600 kB.

As search costs of k-NN queries, we measured the I/O costs (disk accesses) and also the realtimes. Each k-NN query was executed 1000 times, every time for a (new) randomly selected vector from D_k^ε (i.e. as query vectors we have reused the pseudodocument vectors). The results were averaged. To have an efficiency baseline, we also present results for searching by simple sequential scanning of the entire matrix D_k^ε.

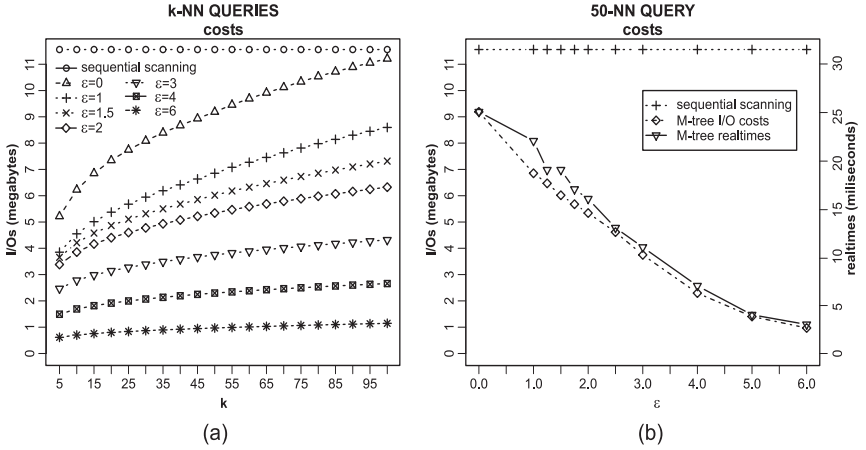


Fig. 10. (a) k-NN queries costs (b) 50-NN query costs, depending on ε

In Figure 10a see the costs of k-NN queries evaluation for several values of ε. With growing ε the query evaluation is more efficient, up to 8 times for $\varepsilon = 6$ and $k = 100$, when related to $\varepsilon = 1$ (the classic LSI). Even in case when $\varepsilon = 3$ (for which the F-score is still well-preserved) the efficiency is improved more than twice, when compared to $\varepsilon = 1$.

The dependence of efficiency on ε is presented in Figure 10b. For 50-NN queries, both I/O costs and realtimes decrease with growing ε. However, had we compared Figures 10b and 7, the intrinsic dimensionality drops much faster than

the costs needed for processing a 50-NN query by the M-tree. This observation indicates that an "ideal" MAM should perform even better than the M-tree.

6 Conclusions

In this paper we have proposed σ-LSI – a novel modification of LSI model for efficient searching in document collections by metric access methods. To battle high intrinsic dimensionality, a convex modification of singular values σ_i by calculating $\sigma_i^{\varepsilon},\ \varepsilon \geq 1$ was proposed. We have shown that for reasonable values of ε the intrinsic dimensionality drops quickly, while the similarity of documents is still well-preserved. In fact, we have observed that our collection seemed to yield almost the same results for $\varepsilon \leq 2.5$, while the search efficiency was doubled.

In future, we would like to apply other convex functions on singular values, testing whether they yield better global results for precision, recall and intrinsic dimensionality than the currently proposed approach. We would like test the approach on a greater collection, too, using some probabilistic methods of LSI calculation, if needed.

Because rank-k SVD is also often used on other types of data, especially images, it would be interesting to evaluate the impact of our method on other metrics (e.g. L_2), query results and intrinsic dimensionality in these collections, too.

Additionally, with the techniques of local dimension reduction, approximate LSI, and σ-LSI modification for better metric indexing, we may be able to build a really viable LSI index.

Acknowledgement

This research has been partially supported by Czech Science Foundation (GAČR) grants Nr. 201/05/P036 and Nr. 201/03/1318.

References

1. V. N. Anh, O. de Kretser, and A. Moffat. Vector-space ranking with effective early termination. In *Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 35–42. ACM Press, 2001.
2. R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval*. Addison Wesley, New York, 1999.
3. M. Berry and M. Browne. *Understanding Search Engines, Mathematical Modeling and Text Retrieval*. Siam, 1999.
4. M. Berry, S. Dumais, and T. Letsche. Computation Methods for Intelligent Information Access. In *Proceedings of the 1995 ACM/IEEE Supercomputing Conference*, 1995.
5. M. W. Berry and R. D. Fierro. Low-Rank Orthogonal Decomposition for Information Retrieval Applications. *Numerical Algebra with Applications*, 1(1):1–27, 1996.

6. C. Böhm, S. Berchtold, and D. Keim. Searching in High-Dimensional Spaces – Index Structures for Improving the Performance of Multimedia Databases. *ACM Computing Surveys*, 33(3):322–373, 2001.
7. E. Chávez and G. Navarro. A probabilistic spell for the curse of dimensionality. In *Proc. 3rd Workshop on Algorithm Engineering and Experiments (ALENEX'01), LNCS 2153*. Springer-Verlag, 2001.
8. E. Chávez, G. Navarro, R. Baeza-Yates, and J. L. Marroquín. Searching in metric spaces. *ACM Compututing Surveys*, 33(3):273–321, 2001.
9. P. Ciaccia, M. Patella, and P. Zezula. M-tree: An Efficient Access Method for Similarity Search in Metric Spaces. In *Proceedings of the 23rd Athens Intern. Conf. on VLDB*, pages 426–435. Morgan Kaufmann, 1997.
10. V. Dohnal, C. Gennaro, P. Savino, and P. Zezula. D-index: Distance searching index for metric data sets. *Multimedia Tools Applications*, 21(1):9–33, 2003.
11. A. Frieze, R. Kannan, and S. Vempala. Fast Monte-Carlo Algorithms for Finding Low Rank Approximations. In *Proceedings of 1998 FOCS*, pages 370–378, 1998.
12. G. H. Golub and C. F. V. Loan. *Matrix computations (3rd ed.)*. Johns Hopkins University Press, 1996.
13. R. M. Larsen. Lanczos bidiagonalization with partial reorthogonalization. Technical report, University of Aarhus, 1998.
14. M. L. Micó, J. Oncina, and E. Vidal. An algorithm for finding nearest neighbour in constant average time with a linear space complexity. In *International Conference on Pattern Recognition*, pages 557–560, 1992.
15. A. Moffat and J. Zobel. Fast ranking in limited space. In *Proceedings of the Tenth International Conference on Data Engineering*, pages 428–437. IEEE Computer Society, 1994.
16. J. K. P. Kanerva and A. Holst. Random Indexing of Text Samples for Latent Semantic Analysis. In *Proceedings of the 22nd Annual Conference of the Cognitive Science Society*, page 1036, 2000.
17. C. H. Papadimitriou, H. Tamaki, P. Raghavan, and S. Vempala. Latent semantic indexing: A probabilistic analysis. In *Proocedings of the ACM Conference on Principles of Database Systems (PODS)*, pages 159–168, 1998.
18. M. Persin. Document filtering for fast ranking. In *Proceedings of the 17th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 339–348. Springer-Verlag New York, Inc., 1994.
19. J. Ponte and W. Croft. A language modelling approach to IR. In *Proceedings of the 21 st ACM SIGIR Conference*, pages 275–281, 1998.
20. T. Skopal, P. Moravec, J. Pokorný, and V. Snášel. Metric Indexing for the Vector Model in Text Retrieval. In *Proceedings of the 11th Symposium on String Processing and Information Retrieval (SPIRE), Padova, Italy, LNCS 3246, Springer-Verlag*, pages 183–195, 2004.
21. E. M. Voorhees and D. Harman. Overview of the sixth text REtrieval conference (TREC-6). *Information Processing and Management*, 36(1):3–35, 2000.
22. P. N. Yanilos. Data Structures and Algorithms for Nearest Neighbor Search in General Metric Spaces. In *Proceedings of Fourth Annual ACM/SIGACT-SIAM Symposium on Discrete Algorithms - SODA*, pages 311–321, 1993.
23. P. Zezula, P. Savino, G. Amato, and F. Rabitti. Approximate Similarity Retrieval with M-Trees. *VLDB Journal*, 7(4):275–293, 1998.

Chapter 9

The Geometric Framework for Exact and Similarity Querying XML Data

Michal Krátký
Jaroslav Pokorný
Tomáš Skopal
Václav Snášel

The Geometric Framework for Exact and Similarity Querying XML Data [31]

Regular paper at the EurAsia-ICT 2002: Information and Communication Technology, Shiraz, Iran, October 2002

Published in the *Lecture Notes in Computer Science* (LNCS), vol. 2510, pages 35–46, *Springer-Verlag*, ISSN 0302-9743, ISBN 3-540-00028-3

The Geometric Framework for Exact and Similarity Querying XML Data

Michal Krátký[1], Jaroslav Pokorný[2], Tomáš Skopal[1], and Václav Snášel[1]

[1] Department of Computer Science, VŠB-Technical University of Ostrava, Czech Republic
[2] Department of Software Engineering, Charles University, Prague, Czech Republic

michal.kratky@vsb.cz, jaroslav.pokorny@ksi.ms.mff.cuni.cz, tomas.skopal@vsb.cz, vaclav.snasel@vsb.cz

Abstract. Using the terminology usual in databases, it is possible to view XML as a language for data modeling. To retrieve XML data from XML databases, several query languages have been proposed. The common feature of such languages is the use of regular path expressions. They enable the user to navigate through arbitrary long paths in XML data. If we considered a path content as a vector of path elements, we would be able to model XML paths as points within a multidimensional vector space. This paper introduces a geometric framework for indexing and querying XML data conceived in this way. In consequence, we can use certain data structures for indexing multidimensional points (objects). We use the UB-tree for indexing the vector spaces and the M-tree for indexing the metric spaces. The data structures for indexing the vector spaces lead rather to exact matching queries while the structures for indexing the metric spaces allow us to provide the similarity queries.

1 Introduction

Using the terminology usual in databases, it is possible to view XML as a language for data modelling. The notions like XML database and XML query language logically extend this idea [5, 14]. So called native XML databases are implemented in increasing extent. To reach a quality of conventional relational databases, appropriate tools for manipulating have been designed. Among many attempts to query languages over XML data, the language XQuery [15] seems to be the leading approach now. The common feature of such languages is the use of regular path expressions. They enable the user to navigate through arbitrary long paths in XML data. Obviously, in the next step to XML databases some appropriate index structures have to be constructed for their data. Particularly, paths can be objects of indexing. In [9], we consider a path content as a vector of path elements. Then we can model XML paths as points within a multidimensional vector space. To speed-up access to such vectors, either various multidimensional trees (such as the R*-tree [3], X-tree [4] or UB-tree [1]), or metric trees can be used for their indexing (e.g., the M-tree [8] and the mvp-tree

[6]). Only few these data structures have been used for indexing XML data. In [9], we used UB-trees for indexing path contents for more efficient exact querying XML data. In this work we pursue a different, in some sense complementary, direction that is based on M-trees. Metric trees only require the distance between points to be a metric, thus they can be used even when no vector representation exists. We show how M-trees can be used for indexing XML paths and how similarity querying XML data can be supported. Section 2 introduces to us the geometric framework used in this paper. We shortly describe necessary basics of vector and metric spaces. Section 3 contains the vector model for indexing and querying XML data. The approach is based on the notion of path content. The main contribution of the paper – a similarity indexing XML data with M-tree – is contained in Section 4. We introduce briefly M-trees and propose a cumulated metric based in the Hamming metric for indexing XML paths. The section is completed with experimental evaluation of M-tree index applied on a real XML data set. In conclusions we summarize the approach.

2 Geometric Framework

In our approach to indexing and querying XML data we exploit the properties of two geometric models. Both of these models treat the XML data as objects/points within a space. In the first case within a *vector space* and in the second case within a *metric space*. As we will see, each of the models is suitable for a different purpose. We can say that they are complementary to each other.

There are two initial problems. First, we need to find a technique of transformation (so-called feature transformation) of the XML data into objects within a vector or metric space. Second, we need to find the data structures for storage and effective querying XML data according to the given model.

2.1 Vector Spaces

Vector model treats the XML data as points within multidimensional vector space. This approach allows us to index values and even the structure of XML documents and provides an ability of *exact matching* range queries. High vector space dimension (greater than approx. 20) is unfortunately associated with *curse of dimensionality* which has a negative influence on the range queries efficiency (see [2]). A representative data structure for the vector model is the UB-tree (see [1]). We discuss the vector model for indexing and querying XML data in Section 3.

2.2 Metric Spaces

In a metric space there are generally neither the dimension nor the vectors. However, in this paper we share the same representation of objects for the metric spaces and for the vector spaces – i.e. multidimensional points. An important difference is that each metric space has defined a metric – i.e. function measuring

a distance (or similarity) between every two objects. This function d must satisfy following conditions:

$$d(o_i, o_i) = 0 \tag{1}$$
$$d(o_i, o_j) > 0 \qquad (o_i \neq o_j) \tag{2}$$
$$d(o_i, o_j) = d(o_j, o_i) \tag{3}$$
$$d(o_i, o_k) + d(o_k, o_j) \geq d(o_i, o_j) \tag{4}$$

The presence of the metric prompts that the metric model provides an ability of *similarity queries*. A representative data structure for the metric model is the M-tree, see section 4.

3 The vector model for indexing and querying XML data

In our approach to indexing XML documents we model the XML data as points within multidimensional vector space and thus we can use certain index structures for multidimensional indexing (for example UB-tree). This approach was introduced in [9]. The data structures for indexing the vector spaces lead rather to *exact queries*.

We distinguish between indexing XML data with and without "mixed content" in [9]. Here we show only the latter case. The example of DTD for documents without "mixed content" and an XML document valid w.r.t. the DTD are in Figure 1a) and 1b), respectively. We will not consider the attributes of elements in our approach.

Example 1 (Querying XML document).
The example of the DTD and the valid XML document is in Figure 1. The path `accounts/account/name` denotes a query for obtaining all account customer names from the document.

```
<!DOCTYPE accounts [
  <!ELEMENT accounts (account*)>
  <!ELEMENT account (id, name)>
  <!ELEMENT id (#PCDATA)>
  <!ELEMENT name (#PCDATA)>
]>
```

a)

```
<?xml version="1.0" ?>
<accounts>
 <account><id>1234-8952</id>
   <name>Thomas Newell</name></account>
 <account><id>1234-4123</id>
  <name>David Moore</name></account>
 <account><id>5842-5321</id>
  <name>David Moore</name></account>
</accounts>
```

b)

Fig. 1. a) An example of DTD for XML documents without "mixed contents". b) An example of valid XML document without "mixed contents".

3.1 Indexing path contents and XML structure

In our approach to indexing XML documents, we consider the *n-dimensional points representing path contents for XML structure indexing* of all paths from the root to all its leafs. The dimension n of the space is equal to the length of the maximal path in XML-tree, i.e. the number of edges from the root to its leaf element. To estimate the number n from DTD, we will consider only the "nonrecursive" DTDs in our approach.

Definition 1 (path content).
*Given a path $e = e_1/e_2/\ldots/e_k$, $e \in \mathcal{X}_P$, $\mathcal{X}_P$ is **set of paths**, the **path content** is defined as a sequence of string values $s = s_1/s_2/\ldots/s_k$, $s \in \mathcal{X}_{PC}$, $\mathcal{X}_{PC}$ is **set of path contents**. Each s_i, except s_k, can be empty (ϵ).*

Because string values can have a different length, it is necessary to use a procedure, which maps different strings into binary numbers of the same length. We use the signatures in our approach (e.g. [10]). The main idea of signatures is to reflect the data items into bit patterns and store them in a separate file which acts as a filter to eliminate the non-qualifying data items for an information request. We will denote the function generating signatures by $\mathrm{sig}(x)$, where x is a variable of string type.

The XML document is represented by m points within n-dimensional space, where m is the considered number of path contents. All these points are inserted into any index structures for multidimensional indexing. All complete paths contents are stored in other data structures. It is important to create binding between the elements of XML document having the same parent. We can create this binding using the elements unique numbers in the point representing path content for XML structure indexing. Of course, it is possible to index even paths (see Section 3.2).

Example 2 (Transformation of XML data to n-dimensional points).
We will show the transformation of the XML document from 1b) to the points of multidimensional space. We see the space has $n = 3$. We determine the length of the domains as 64b. This signature value is large enough for the signature s_i. But generally, there is not cause for domain cardinalities to be the same. The cardinality of domain for signatures of #PCDATA and for unique numbers of root elements can be different for example. The important role plays here the analysis of DTD.

If we are browsing through document in Figure 1b), then the following path contents are obtained: $\epsilon/\epsilon/$`1234-8952`, $\epsilon/\epsilon/$`Thomas Newell`, $\epsilon/\epsilon/$`1234-4123`, $\epsilon/\epsilon/$`David Moore`, $\epsilon/\epsilon/$`5842-5321` and $\epsilon/\epsilon/$`David Moore`. It is necessary to group these path contents according to the relationship to particular `accounts` and `account` elements. Therefore we nest the unique numbers of `accounts` and `account` elements into 1^{st} (2^{nd} respectively) coordinate of points representing path contents. The points representing path contents will be `(0,0,sig("1234-8952"))`, `(0,0,sig("Thomas Newell"))`, `(0, 1, sig("1234-4123"))`, `(0, 1, sig("David Moore"))`, `(0, 2, sig("5842-5321"))`, and `(0, 2, sig("David`

Moore")). The points in 3-dimensional space are depicted in Figure 2. These points are inserted into the indexing structure. If we index paths (see Section 3.2), then we will work with 4-dimensional space.

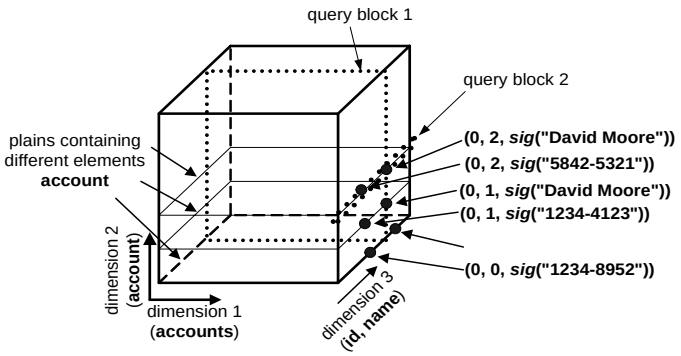


Fig. 2. The 3-dimensional space with indexed XML document without "mixed contents".

Example 3 (Querying XML document).
We show now how it is possible to query the XML document from Figure 1b) transformed to the points within multidimensional space by above mentioned technique. Let us take the query `accounts/account[name='David Moore']`, i.e. we want to get all `account` elements for `David Moore`'s account. First we need to transform this query to the *range query*. It means to find all the points from Figure 2, that are contained by *query block 1*. It is necessary to determine the coordinates of two points defining the query block. By means of range query we get the points from the 3-dimensional space which represent the unique numbers of parent elements of `name` element with content `David Moore`. We get the result set and if user will want to obtain the contents of child elements of any `account` element, for example, then the query block like *query block 2* from Figure 2 is effected for their retrieval. To distinguish the points representing the path contents for different paths it is necessary to index even the paths (see Section 3.2).

3.2 Indexing paths

The indexing XML data as it is proposed in Section 3.1 considers only a path content. If the XML document is transformed to points of a space in this way, the element tags are lost. If we consider the XML document from Figure 1 then we will be not able to distinguish the points representing the path contents for paths `accounts/account/id` or `accounts/account/name`.

We consider a binary relation PPC [9] between paths and their path contents. All points representing paths will be inserted to other index structure. Besides

the point coordinates and pointers to data structures containing the whole paths we insert even the path unique numbers in another dimension of the space which contains the path contents. In fact, the relation PPC is built by adding other dimension to the space which contains path contents, i.e. the dimension of space will be $n+1$. It is hereby possible to index even the documents valid to different DTD in one index structure in this way.

Example 4 (Indexing paths).
We get two different paths `accounts/account/id` and `accounts/account/name` from the XML document in Figure 1b). So we get two points (we get two paths) representing paths in 3-dimensional space (paths contain three elements). These points are inserted into other indexing structure. The point `(sig("accounts"), sig("account"),sig("id"))` representing path `accounts/account/id` is inserted with unique number 0 and point representing path `(sig("accounts"), sig("account"),sig("name"))` with unique number 1. The points representing paths are in Figure 3. The points representing the path contents have last coordinate equal to the unique number of the associated path.

We see the space to have $n = 4$ (one dimension will be for unique numbers of paths). The gained path contents are in Example 2. Let us take the path content ϵ/ϵ`/1234-8952` and point representing the path contents `(0,0,sig("1234-8952"))` for example. The path unique number of path `accounts/account/id` from index structures which contain points representing paths is append as fourth coordinate to the point. We get point `(0,0,sig("1234-8952"),0)` in this manner. The all six points gained by the same way are inserted into index structure containing path contents.

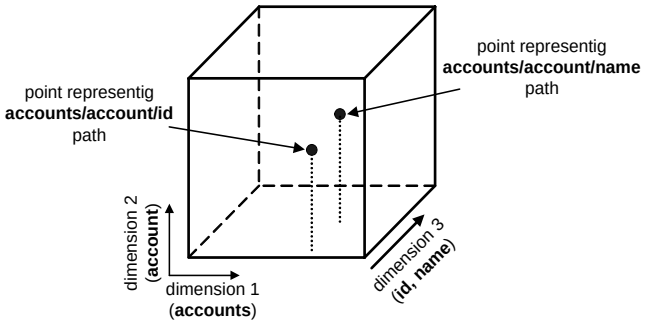


Fig. 3. The 3-dimensional space with points representing paths.

Example 5 (Querying XML document).
It is important to get by a point query the unique number of the desired path from index structure containing paths. After that we get desired points by a range query from the indexing structure containing path contents. It is necessary to

work with four dimensions in the case of defined coordinates of points determining the query block.

4 Similarity Queries

Another aspect of indexing XML data, in addition to the structural indexing, is the *similarity indexing*. In such an XML index we can query for XML objects that are similar to a query object.

Properties of metric spaces, where the metric represents the notion of similarity, are suitable formal basis for indexing similarities inside XML data. Following subsection describes a data structure M-tree which allows to index general objects of metric spaces.

4.1 M-tree

Data structure M-tree (introduced in [8] and closely discussed in [13]) was developed for indexing and querying objects within metric spaces. Its main characteristic is that M-tree allows to process similarity queries. It is, in fact, dynamic, persistent, paged and balanced tree like e.g. the B-tree. The difference is in the semantics of the nodes. Indexed objects themselves, i.e. *ground objects*, lie in the leaf nodes. The inner nodes contain *routing objects* that represent a hierarchy of specific metric regions.

- The record of a routing object O_r in inner node contains:
  1. a ground object O_r (its significant properties respectively). This ground object determines the center of the metric region.
  2. pointer $ptr(T(O_r))$ to its own subtree $T(O_r)$ – i.e. *covering tree*
  3. value $r(O_r)$ – *covering radius* of the metric region
  4. value $d(O_r, P(O_r))$ – distance to the parent routing object $P(O_r)$

  Notes:
  The ground object in the routing object (inner node) is one of the ground objects remaining in the child leaf nodes of $T(O_r)$. The distance function d is a metric of a metric space.
- The record of a ground object looks similarly, but it also contains $oid(O_j)$ – identifier of the whole object (stored outside of M-tree) – instead of covering tree and covering radius.

Hierarchy of M-tree is based on partition of the metric space onto metric subregions which do not have to be strictly disjunct. This regions are formed by the routing objects O_r where the child routing objects (their regions respectively) and the child ground objects of its covering tree $T(O_r)$ are within the distance $r(O_r)$ to the center of O_r. Formally,

$$\forall O_i \in T(O_r), \qquad d(O_r, O_i) \leq r(O_r)$$

The precalculated distance value $d(O_r, P(O_r))$ to the parent object along with the covering radius $r(O_r)$ allow to eliminate the *untouched regions* from the process of an operation on M-tree (i.e. searching, insertion, deletion). Structure of the M-tree and the routing object relations are depicted in figure 4.

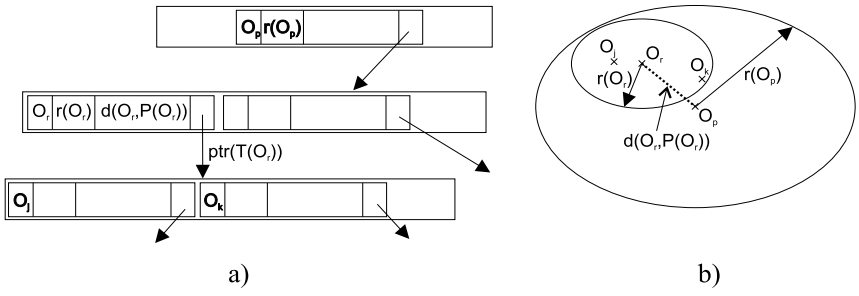


Fig. 4. (a) Nodes of M-tree contain object records. (b) Routing objects – metric regions.

Searching the M-tree We must take into account two factors of complexity when we make some operation on the M-tree. The first one is the number of *accesses to disk pages* (number of regions being searched respectively) and the second one (specifically to M-tree) is the number of *distance calculations*. The goal is to minimize both these factors.

We can meet two kinds of queries by metric trees. The *range queries* search for all the objects within certain distance to the query object. The *k-nearest neighbours queries* search for the first k nearest objects to the query object. In both cases we can see a tendency to order the metric space – relative to the query object.

Managing the regions The crucial factor of the M-tree's cost-effectiveness is a "good layout" of the metric regions stored within the M-tree. As we have said earlier, the regions can overlap another ones. This property arises from the M-tree's universality which is caused due to specifying only a metric of the metric space. High "overlap rate" leads, in the worst case, to sequential search – i.e. to linear complexity.

With the design of the M-tree there were also developed some techniques for minimizing this "overlap rate". The first technique is "embedded" into the phase of a tree node(page) splitting and consists of a choice of *split policy* and a mechanism of creating the best routing object – *promoting* phase. This is the dynamic technique. The second technique, more efficient, is the *bulk loading* algorithm. This algorithm takes at the beginning the whole collection of objects and loads all of them into the empty M-tree at once. The loading is based on preliminary clustering where prospective regions of objects are created at once. This is the static method.

Summary M-tree is balanced, highly parametrizable data structure making possible to index objects of a metric space. The M-tree operations are performed with approximately logarithmic time complexity (if well build) but the M-tree doesn't represent a complete linear order like other trees (B-tree, UB-tree, ...) do. On the other side, M-tree is more general than the *Spatial Access Methods* based on vector spaces.

4.2 Indexing XML data with M-tree

If we consider XML paths as simple objects, we can index such objects into a metric space or actually into the M-tree. For example, path `BOOK/AUTHOR/SURNAME` is object to store within M-tree. All paths in given XML document(s) can be transformed in this way into a collection of this simple XML objects. XML objects can also have assigned to every element tag its element content, which will increase the number of unique objects. For example, `BOOK{technical}/AUTHOR{writer}/SURNAME{Walsh}`, but furthermore, for simplicity, we will ignore the possibility of any content.

XML object o_i (path) can be represented as a variable vector of strings (element tags), $o_i = (o_i^1, o_i^2, \cdots, o_i^{l_i})$.

Choosing metric for paths Metric chosen for XML indexing must take as arguments two XML objects (paths) and calculate distance between them. We propose as an example *cumulated metric* which is defined as:

$$D(o_i, o_j) = \sum_{k=1}^{max(l_i, l_j)} d(o_i^k, o_j^k)$$

where $d(x, y)$ is an ordinary metric (e.g. Hamming metric) between two strings.

Hamming metric [7] adds up the mismatching pairs of characters where the first character of a pair is located on a position in the first string while the second character is on the same position in the second string. Formally,

$$d_H(x, y) = \sum_{i=1}^{min(|x|,|y|)} sgn(|x[i] - y[i]|) + ||x| - |y||$$

For example, $d_H($`AUTHORS`, `AUTOMATON`$) = 0 + 0 + 0 + 1 + 1 + 1 + 1 + 2 = 6$

Example 6.

Let d be the Hamming metric. Then

$D($`BOOK/AUTHOR/SURNAME`, `BOOK/AUTHOR/FIRSTNAME`$) = 0 + 0 + 8 = 8$

Let d be the discrete (yes/no) metric. Then

$D($`BOOK/PREFACE/TITLE`, `BOOK/BOOKINFO/TITLE`$) = 0 + 1 + 0 = 1$

Note: In this section, the paths used in examples are generated according to the DocBook DTD, see [12].

Processing queries We have defined objects of metric space (XML paths) as well as metric (cumulated metric) thus we have accomplished the requirements for indexing with the M-tree.

We can distinguish two types of queries:

1. *similarity queries.* An object o_i in query result is within some distance r (*query radius*) to the query object o_q, i.e. the M-tree is traversed with condition $D(o_q, o_i) \leq r$. This kind of query allows to obtain the similar XML paths.

Example 7 (cumulated Hamming metric).
Query object = `BOOK/PART/CHAPTER/PARA/ACRONYM`, $r = 6$
Query result = {`BOOK/PART/CHAPTER/PARA/ACRONYM` (distance 0)
`BOOK/PART/CHAPTER/PARA/SCREEN` (distance 4)
`BOOK/PART/CHAPTER/TITLE/ACRONYM` (distance 5)
`BOOK/PART/CHAPTER/PARA/FILENAME` (distance 6) }

2. *exact matching queries.* An object o_i in query result must exactly match the query object o_q, i.e. the M-tree is traversed with the condition $D(o_q, o_i) = 0$. This is the special case of similarity query with $r = 0$ – no differences are allowed.

Notes:

- The query object is not expressed by any query language, its structure is the same as the structure of any ground object.
- The syntax of query object can be extended with keyword "*", where using this keyword on the k-th coordinate of object vector brings evaluation of $d(o_q^k, o_i^k)$ always as 0 (match).
  Example: D(`BOOK/AUTHOR/*`, `BOOK/AUTHOR/FIRSTNAME`) $= 0 + 0 + 0 = 0$.
  This extension allows to treat the exact matching queries as range queries.
- The objects in query result give only the information about existence of such paths in XML tree but the objects cannot tell the exact location. This lack of "context" can be removed with additional property of XML object – unique identifier of the last path element pointing into an external data structure (e.g. the source XML tree or UB-tree index). This improvement makes possible the consequential navigation in the external XML tree.

4.3 Testing with M-tree

We have performed particular tests with M-tree – XML path indexing and XML similarity queries. XML data we have indexed was a XML file containing the documentation to DocBook. The size of this file was about 3MB.

In the first phase, we have transformed the whole file into collection of XML objects (unique paths) – 972 unique paths were extracted. Second, we have inserted all of these objects one-by-one into the M-tree. Page size of the M-tree was 1kB and cumulated Hamming metric was chosen. Each object (path vector) of the M-tree was aligned on size of 256 bytes.

After the indexing phase, the M-tree has acquired following statistics: pages(nodes) count: 1568, leafs count: 590. Table 1 shows for each level of the M-tree its pages count and average radius of all routing objects(regions) within the level.

Level	Pages count	Avg. radius
0 (root)	3	207.33
1	5	183.00
2	10	135.40
3	16	121.75
4	23	102.74
5	34	85.18
6	53	65.79
7	83	54.02
8	141	37.14
9	233	22.95
10	376	13.12
11 (leafs)	590	6.01

Table 1: M-tree statistics

Furthermore, disk access costs test was performed. A series of queries was produced by specifying the query object as:
(`"BOOK/PART/CHAPTER/SECT1/SECT2/PARA/ACRONYM"`) and by increasing the query radius from $r = 0$ (exact matching query) to $r = 32$. The results are shown in figure 5.

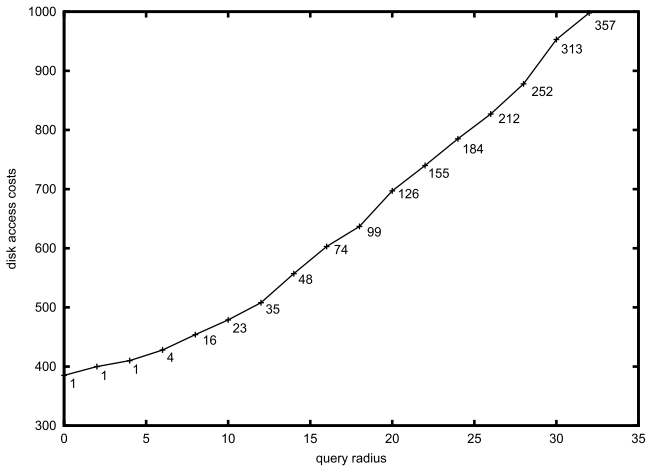


Fig. 5. Results of disk access costs test. The numbers below particular results are the total numbers of objects returned in particular query result (objects similar to the query object within the current radius).

5 Conclusions and Outlook

In this paper we have shown that XML data can be modelled in multidimensional vector spaces and in metric spaces. We use the UB-tree for indexing the vector spaces and the M-tree for indexing metric the metric spaces in our approach of indexing XML data. The data structures for indexing the vector spaces lead rather to the *exact queries* while structures for indexing of the metric spaces allow us to provide *similarity queries*. In the course of writing this paper some interesting questions appeared, e.g. new metric designs or different feature transformations. Their solution will be the topic of our future work. Furthermore, presented data structures are independent and our intention is to integrate them into a single hybrid data structure providing a possibility of XML data storage and also efficient exact and similarity querying.

Acknowledgments

This research was supported in part by GACR grant 201/00/1031.

References

1. Bayer R.: The Universal B-Tree for multidimensional indexing: General Concepts. In: *Proc. Of World-Wide Computing and its Applications 97 (WWCA 97)*. Tsukuba, Japan, 1997.
2. Böhm C., Berchtold S., Keim D.A.: Searching in High-dimensional Spaces – Index Structures for Improving the Performance of Multimedia Databases. *ACM*, 2002
3. Beckmann, N., Kriegel, H.-P., Schneider, R., Seeger, B.: The R*-tree: An efficient and robust access method for points and rectangles. In: *Sigmod'90*, Atlantic City, NY, 1990, pp. 322-331.
4. Brechtold, S., Keim, A., Kriegel, H.-P.: The X-tree: An index structure for high-dimensional data. In: *Proc. of 22nd Intern. Conference on VLDB'96*, Bombay, India, 1996, pp. 28-39.
5. Bourret, R.: *XML and Databases*. `http://www.rpbourret.com/xml/ XMLAndDatabases.htm`. 2001.
6. Bozkaya, T., Özsoyoglu, M.: Distance-based indexing for high-dimensional metric spaces. In: *Sigmod '97*, Tuscon, AZ, 1997, pp. 357-368.
7. Baeza-Yates, R., Ribeiro-Neto, B.: *Modern Information Retrieval*. Addison Wesley, New Yourk, 1999.
8. Ciaccia P, Pattela M., Zezula P.: M-tree: An Efficient Access Method for Similarity Search in Metric Spaces. *Proc. 23rd Athens Intern. Conf. on VLDB (1997)*, 426-435.
9. Krátký M., Pokorný, J., Snášel V.: Indexing XML data with UB-trees. *ADBIS 2002*, Bratislava, Slovakia, accepted.
10. Lee, D.L., Kim, Y.M., Patel, G.: Efficient Signature File Methods for Text Retrieval., *Knowledge and Data Engineering*, Vol. 7, No. 3, 1995, pp. 423-435.
11. Markl, V.: *Mistral: Processing Relational Queries using a Multidimensional Access Technique*, `http://mistral.in.tum.de/results/ publications/Mar99.pdf`, 1999
12. *The DocBook open standard*, Organization for the Advancement of Structured Information Standards (OASIS), 2002, `http://www.oasis-open.org/committees/docbook`
13. M. Patella: *Similarity Search in Multimedia Databases*. Dipartmento di Elettronica Informatica e Sistemistica, Bologna 1999 `http://www-db.deis.unibo.it/~patella/MMindex.html`
14. Pokorný, J.: XML: a challenge for databases?, Chap. 13 In: *Contemporary Trends in Systems Development* (Eds.: Maung K. Sein), Kluwer Academic Publishers, Boston, 2001, pp. 147-164.
15. *XQuery 1.0: An XML Query Language*. W3C Working Draft 20 December 2001, `http://www.w3.org/TR/2001/ WD-xquery-20011220/`

Chapter 10

Conclusions

We have presented selected results of author's research, carried out through years 2002–2006. The efficiency issues of similarity search were addressed by several extensions of the M-tree (an access method for searching in metric spaces). In particular, construction techniques for obtaining more compact M-tree hierarchies were presented in Chapter 2, resulting in better filtering at the expense of higher construction costs. The PM-tree, presented in Chapters 3 and 4, was proved to perform significantly better than the M-tree, considering both the original M-tree construction techniques as well as the modified ones producing more compact hierarchies.

An access method supporting multi-metric queries – the M^3-tree – was presented in Chapter 5. The M^3-tree was experimentally proved to achieve almost the same efficiency as having multiple M-tree indices (each for a particular query metric), while the space overhead needed to store the additional information in M^3-tree index is negligible. Unlike the M-tree adapted for multi-metric queries, the M^3-tree is not sensitive to the distribution of query weights.

In Chapter 6 a nonmetric-to-metric transformation was presented, allowing to employ the metric access methods for indexing of non-metric datasets. Although the idea of triangle-generating modifiers is simple (from the mathematical point of view), the experimental results have shown that finding a suitable modifier produces a metric that is powerful enough to effectively filter the dataset with respect to a query.

In Chapters 7 and 8 the approximate search in metric spaces was discussed. Both of the presented methods utilize triangle-violating modifiers (an opposite construction to the non-metric transformation). In the former case, the modifiers are applied directly on a given semimetric, while in the latter case the modifiers have been used for specific transformation of vectors in the LSI model of text retrieval.

In the last chapter we have presented a model of measuring similarity of XML documents (XML paths, actually) by cumulated string metrics.

10.1 Current Research

In our current research we continue improving the general access methods for searching in metric spaces. In particular, a paper on extending the M-tree nodes by nearest-neighbor graphs was submitted recently. We also plan to integrate the ideas used in M^3-tree into the PM-tree.

The ideas of triangle-generating/violating modifiers have been unified in a paper invited by the EDBT PC to ACM TODS (submitted recently). To mention the main contribution, we have developed a generalized version of the TriGen algorithm, in order to handle any dissimilarity measure (either a metric or semi-metric). In this unified similarity framework, we are able the search exactly or approximately by metric or non-metric dissimilarity measure.

10.2 Future Work

In the future we would like to move a bit more to applications, in order to acquire more evidence for improving various kinds of similarity search. In particular, the improvement of feature extraction from images is the key problem to achieve more *effective* image retrieval. Nowadays, the extraction techniques are mostly focused on producing a single feature vector which is then passed into a simple L_p distance. The multi-metric approach (together with M^3-tree) and the non-metric approach (together with TriGen) challenge us to design more complex data representations (even non-vectorial) and distance measures, aiming to perform a search that is more "semantic" (ideally to partially bridge the semantic gap in image retrieval).

Bibliography

[1] Extensible Markup Language (XML) 1.0, W3C Recommendation, http://www.w3.org/TR/1998/REC-xml-19980210, 1998.

[2] Giuseppe Amato, Fausto Rabitti, Pasquale Savino, and Pavel Zezula. Region proximity in metric spaces and its use for approximate similarity search. *ACM Transactions on Information Systems*, 21(2):192–227, 2003.

[3] F.G. Ashby and N.A. Perrin. Toward a unified theory of similarity and recognition. *Psychological Review*, 95(1):124–150, 1988.

[4] Sihem Amer-Yahia Att. Texquery: A full-text search extension to xquery.

[5] Ricardo A. Baeza-Yates and Berthier Ribeiro-Neto. *Modern Information Retrieval*. Addison-Wesley Longman Publishing, 1999.

[6] M.W. Berry and M. Browne. *Understanding Search Engines, Mathematical Modeling and Text Retrieval*. Siam, 1999.

[7] Christian Böhm, Stefan Berchtold, and D Keim. Searching in High-Dimensional Spaces – Index Structures for Improving the Performance of Multimedia Databases. *ACM Computing Surveys*, 33(3):322–373, 2001.

[8] Tolga Bozkaya and Meral Özsoyoglu. Indexing large metric spaces for similarity search queries. *ACM Transactions on Database Systems*, 24(3):361–404, 1999.

[9] C. Brambilla, A. Della Ventura, I. Gagliardi, and R. Schettini. Multiresolution wavelet transform and supervised learning for content-based image retrieval. *icmcs*, 01:9183, 1999.

[10] Sergey Brin. Near neighbor search in large metric spaces. In *Proceedings of the 21th International Conference on Very Large Data Bases*, pages 574–584. Morgan Kaufmann Publishers Inc., 1995.

[11] B. Bustos, D. Keim, D. Saupe, T. Schreck, and D. Vranić. Automatic selection and combination of descriptors for effective 3D similarity search. In

Proc. IEEE International Workshop on Multimedia Content-based Analysis and Retrieval (MCBAR'04), pages 514–521. IEEE Computer Society, 2004.

[12] B. Bustos, D. Keim, D. Saupe, T. Schreck, and D. Vranić. Using entropy impurity for improved 3D object similarity search. In *Proc. IEEE International Conference on Multimedia and Expo (ICME'04)*, pages 1303–1306. IEEE, 2004.

[13] B. Bustos, D. Keim, and T. Schreck. A pivot-based index structure for combination of feature vectors. In *Proc. 20th Annual ACM Symposium on Applied Computing, Multimedia and Visualization Track (SAC-MV'05)*, pages 1180–1184. ACM Press, 2005.

[14] Benjamin Bustos and Gonzalo Navarro. Probabilistic proximity search algorithms based on compact partitions. *Journal of Discrete Algorithms*, 2(1):115–134, 2004.

[15] Benjamin Bustos, Gonzalo Navarro, and Edgar Chávez. Pivot selection techniques for proximity searching in metric spaces. *Pattern Recognition Letters*, 24(14):2357–2366, 2003.

[16] Benjamin Bustos and Tomáš Skopal. Dynamic Similarity Search in Multi-Metric Spaces. In *Proceedings of ACM Multimedia, MIR workshop*, pages 137–146. ACM Press, 2006.

[17] Barbara Catania, Anna Maddalena, and Athena Vakali. Xml document indexes: A classification. *IEEE Internet Computing*, 9(5):64–71, 2005.

[18] Edgar Chávez and Gonzalo Navarro. A Probabilistic Spell for the Curse of Dimensionality. In *ALENEX'01, LNCS 2153*, pages 147–160. Springer, 2001.

[19] Paolo Ciaccia and Marco Patella. Bulk loading the M-tree. In *Proceedings of the 9th Australian Conference (ADC'98)*, pages 15–26, 1998.

[20] Paolo Ciaccia and Marco Patella. The M^2-tree: Processing Complex Multi-Feature Queries with Just One Index. In *DELOS Workshop: Information Seeking, Searching and Querying in Digital Libraries*, Zurich, Switzerland, June 2000.

[21] Paolo Ciaccia and Marco Patella. Searching in metric spaces with user-defined and approximate distances. *ACM Database Systems*, 27(4):398–437, 2002.

[22] Paolo Ciaccia, Marco Patella, and Pavel Zezula. M-tree: An Efficient Access Method for Similarity Search in Metric Spaces. In *VLDB'97*, pages 426–435, 1997.

[23] Brian F. Cooper, Neal Sample, Michael J. Franklin, Gísli R. Hjaltason, and Moshe Shadmon. A fast index for semistructured data. In *VLDB*, pages 341–350, 2001.

[24] Scott C. Deerwester, Susan T. Dumais, Thomas K. Landauer, George W. Furnas, and Richard A. Harshman. Indexing by latent semantic analysis. *Journal of the American Society of Information Science*, 41(6):391–407, 1990.

[25] Vlastislav Dohnal, Claudio Gennaro, Pasquale Savino, and Pavel Zezula. D-index: Distance searching index for metric data sets. *Multimedia Tools and Applications*, 21(1):9–33, 2003.

[26] C. Faloutsos and K. Lin. Fastmap: A Fast Algorithm for Indexing, Data-Mining and Visualization of Traditional and Multimedia Datasets. In *SIGMOD*, 1995.

[27] Roberto F. Santos Filho, Agma J. M. Traina, Caetano Traina, and Christos Faloutsos. Similarity search without tears: The OMNI family of all-purpose access methods. In *ICDE*, 2001.

[28] Michael Freeman. Evaluating dataflow and pipelined vector processing architectures for fpga co-processors. In *DSD '06: Proceedings of the 9th EUROMICRO Conference on Digital System Design*, pages 127–130, Washington, DC, USA, 2006. IEEE Computer Society.

[29] G. D. Guo, A. K. Jain, W. Y. Ma, and H. J. Zhang. Learning similarity measure for natural image retrieval with relevance feedback. *IEEE Neural Networks*, 13(4):811–820, 2002.

[30] G. R. Hjaltason and Hanan Samet. Properties of embedding methods for similarity searching in metric spaces. *IEEE Patt.Anal. and Mach.Intell.*, 25(5):530–549, 2003.

[31] Michal Krátký, Jaroslav Pokorný, Tomáš Skopal, and Václav Snášel. The Geometric Framework for Exact and Similarity Querying XML Data. In *Proceedings of First EurAsian Conferences, EurAsia-ICT 2002, Shiraz, Iran.* Springer-Verlag LNCS 2510, October 27-31, 2002.

[32] C. L. Krumhansl. Concerning the applicability of geometric models to similar data: The interrelationship between similarity and spatial density. *Psychological Review*, 85(5):445–463, 1978.

[33] Chen Li, Edward Chang, Hector Garcia-Molina, and Gio Wiederhold. Clustering for approximate similarity search in high-dimensional spaces. *IEEE Transactions on Knowledge and Data Engineering*, 14(4):792–808, 2002.

[34] Thomas Mandl. Learning similarity functions in information retrieval. In *EUFIT*, 1998.

[35] Maria Luisa Micó, José Oncina, and Enrique Vidal. An algorithm for finding nearest neighbour in constant average time with a linear space complexity. In *Int. Cnf. on Pattern Recog.*, 1992.

[36] Gonzalo Navarro. Searching in metric spaces by spatial approximation. *The VLDB Journal*, 11(1):28–46, 2002.

[37] Eleanor Rosch. Cognitive reference points. *Cognitive Psychology*, 7:532–47, 1975.

[38] E. Rothkopf. A measure of stimulus similarity and errors in some paired-associate learning tasks. *J. of Experimental Psychology*, 53(2):94–101, 1957.

[39] Yossi Rubner, Carlo Tomasi, and Leonidas J. Guibas. The earth mover's distance as a metric for image retrieval. *Int. J. Comput. Vision*, 40(2):99–121, 2000.

[40] Simone Santini and Ramesh Jain. Similarity measures. *IEEE Pattern Analysis and Machine Intelligence*, 21(9):871–883, 1999.

[41] Tomás Skopal. On fast non-metric similarity search by metric access methods. In *Proc. 10th International Conference on Extending Database Technology (EDBT'06)*, LNCS 3896, pages 718–736. Springer, 2006.

[42] Tomáš Skopal. *Metric Indexing in Information Retrieval*. PhD thesis, Technical University of Ostrava, `urtax.ms.mff.cuni.cz/skopal/phd/thesis.pdf`, 2004.

[43] Tomáš Skopal. Pivoting M-tree: A Metric Access Method for Efficient Similarity Search. In *Proceedings of the 4th annual workshop DATESO, Desná, Czech Republic, ISBN 80-248-0457-3, also available at CEUR, Volume 98, ISSN 1613-0073,* `http://www.ceur-ws.org/Vol-98`, pages 21–31, 2004.

[44] Tomáš Skopal and Pavel Moravec. Modified lsi model for efficient search by metric access methods. In *ECIR 2005*, pages 245–259. LNCS 3408, Springer-Verlag, 2005.

[45] Tomáš Skopal, Pavel Moravec, Jaroslav Pokorný, and Václav Snášel. Metric Indexing for the Vector Model in Text Retrieval. In *SPIRE, Padova, Italy*, pages 183–195. LNCS 3246, Springer, 2004.

[46] Tomáš Skopal, Jaroslav Pokorný, Michal Krátký, and Václav Snášel. Revisiting M-tree Building Principles. In *ADBIS, Dresden*, pages 148–162. LNCS 2798, Springer, 2003.

[47] Tomáš Skopal, Jaroslav Pokorný, and Václav Snášel. PM-tree: Pivoting metric tree for similarity search in multimedia databases. In *ADBIS '04, Budapest, Hungary*, pages 99–114, 2004.

[48] Tomáš Skopal, Jaroslav Pokorný, and Václav Snášel. Nearest Neighbours Search using the PM-tree. In *DASFAA '05, Beijing, China*, pages 803–815. LNCS 3453, Springer, 2005.

[49] Caetano Traina Jr., Agma Traina, Bernhard Seeger, and Christos Faloutsos. Slim-Trees: High performance metric trees minimizing overlap between nodes. *Lecture Notes in Computer Science*, 1777, 2000.

[50] Andrew Trotman and Borkur Sigurbjornsson. Narrowed extended xpath i (nexi). In *INEX*, 2005.

[51] Ertem Tuncel, Hakan Ferhatosmanoglu, and Kenneth Rose. Vq-index: an index structure for similarity searching in multimedia databases. In *MULTIMEDIA '02: Proceedings of the tenth ACM international conference on Multimedia*, pages 543–552, New York, NY, USA, 2002. ACM Press.

[52] A. Tversky and I. Gati. Similarity, separability, and the triangle inequality. *Psychological Review*, 89(2):123–154, 1982.

[53] Amos Tversky. Features of similarity. *Psychological review*, 84(4):327–352, 1977.

[54] Jeffrey K. Uhlmann. Satisfying general proximity/similarity queries with metric trees. *Information Processing Letters*, 40(4):175–179, 1991.

[55] Stephan Volmer. Buoy indexing of metric feature spaces for fast approximate image queries. In *Proceedings of the sixth Eurographics workshop on Multimedia 2001*, pages 131–140, New York, NY, USA, 2002. Springer-Verlag New York, Inc.

[56] Peter N. Yianilos. Data structures and algorithms for nearest neighbor search in general metric spaces. In *ACM SIAM SODA*, pages 311–321, 1993.

[57] Pavel Zezula, Giuseppe Amato, Vlastislav Dohnal, and Michal Batko. *Similarity Search: The Metric Space Approach (Advances in Database Systems)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2005.

[58] Pavel Zezula, Pasquale Savino, Giuseppe Amato, and Fausto Rabitti. Approximate Similarity Retrieval with M-Trees. *VLDB Journal*, 7(4):275–293, 1998.

[59] Xiangmin Zhou, Guoren Wang, Jeffrey Yu Xu, and Ge Yu. M^+-tree: A New Dynamical Multidimensional Index for Metric Spaces. In *Proceedings of the Fourteenth Australasian Database Conference - ADC'03, Adelaide, Australia*, 2003.