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.

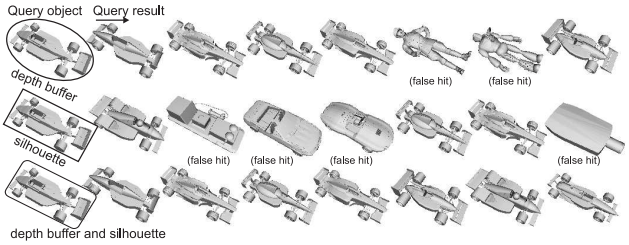

Figure 1: Improving effectiveness of 3D similarity search by combining two 3D feature vectors.

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* ($rout(O_i)$, $O_i \in \mathbb{S}$).

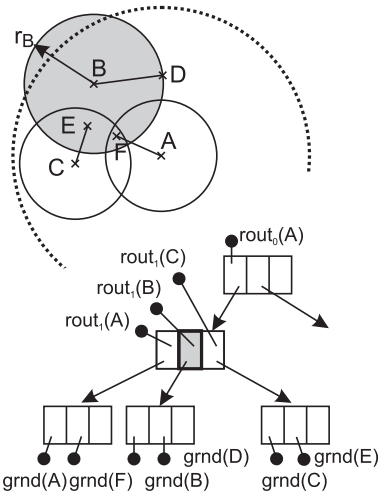

Figure 2: Example of an M-tree.

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

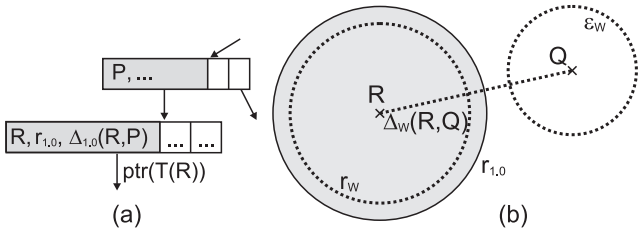

Figure 3: (a) Non-leaf node entries in M-tree. (b) Basic filtering in M-tree.

"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. ■

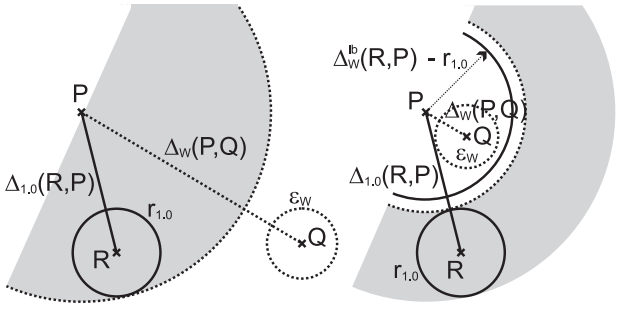

Figure 4: (a) Outer parent filtering in M-tree. (b) Inner parent filtering.

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 $r_{1.0}$. □

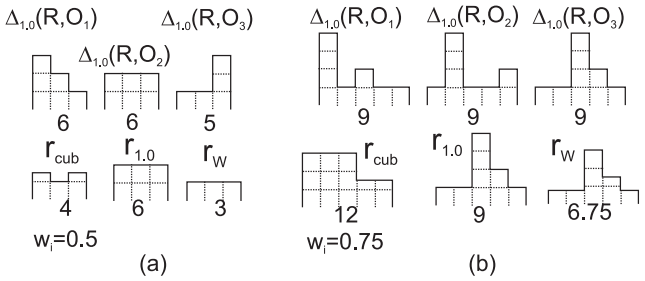

Figure 5: **(a)** $r_{\mathbb{W}} < r_{cub} < r_{1.0}$ $\qquad$ **(b)** $r_{\mathbb{W}} < r_{1.0} < r_{cub}$.

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.

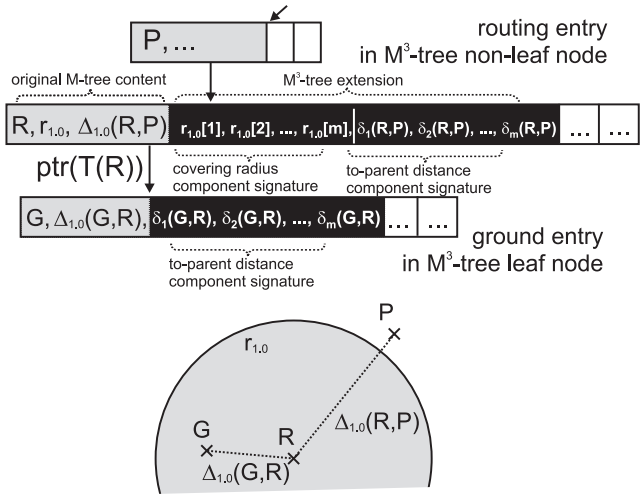

Figure 6: Structure of M³-tree nodes.

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 M³-tree Construction

The M³-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 M³-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 M³-tree

The M³-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 M³-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 M³-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/`

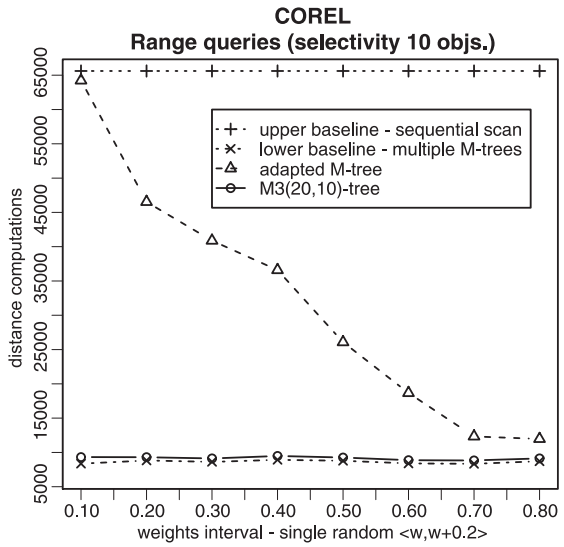

Figure 7: Corel image features: Range queries varying weights interval.

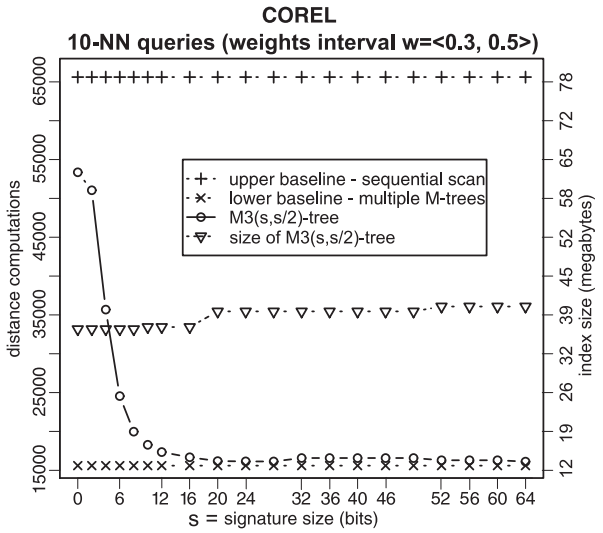

Figure 8: Corel image features: 10-NN queries varying signature size.

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

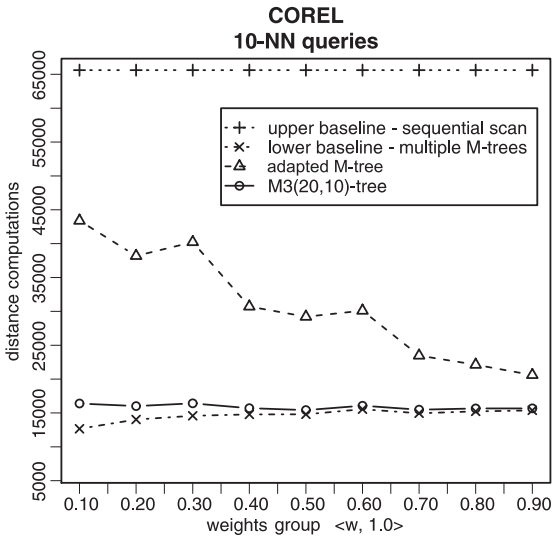

Figure 9: Corel image features: 10-NN queries varying weights group.

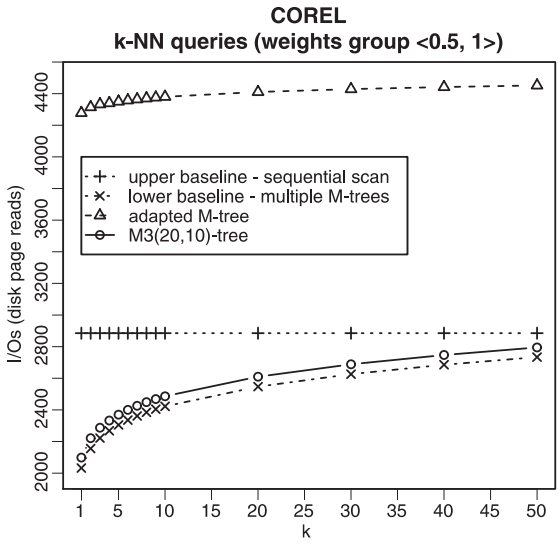

Figure 10: Corel image features: k-NN queries with fixed weights group.

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].

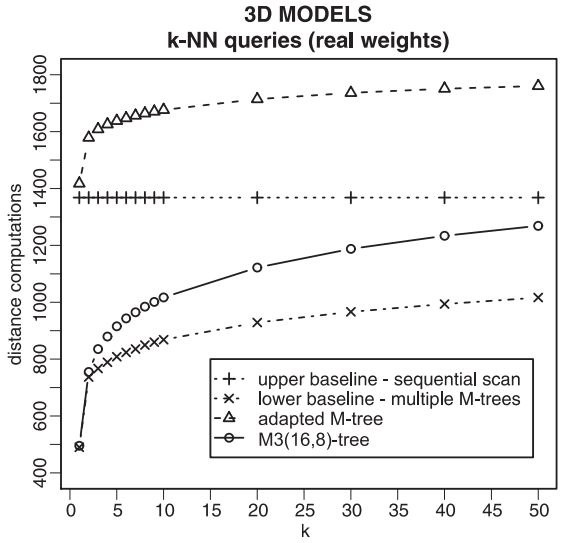

Figure 11: 3D models: k-NN queries.

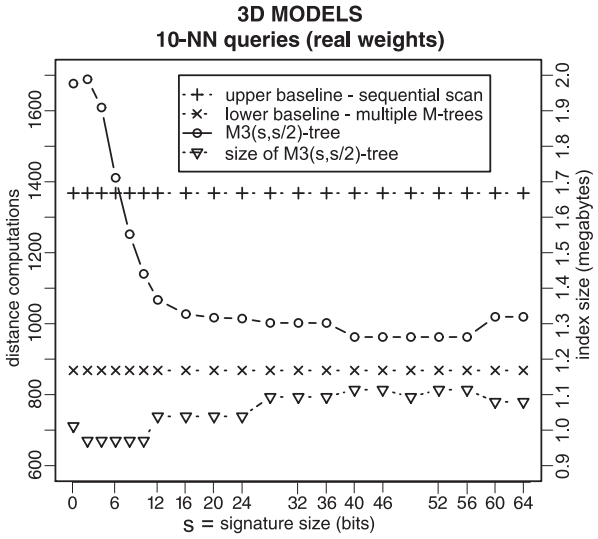

Figure 12: 3D models: 10-NN varying signature size.

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.