# NM-tree: Flexible Approximate Similarity Search in Metric and Non-metric Spaces

Tomáš Skopal and Jakub Lokoč

Charles University in Prague, FMP, Department of Software Engineering,
Malostranské nám. 25, 118 00 Prague, Czech Republic
{skopal,lokoc}@ksi.mff.cuni.cz

**Abstract.** So far, an efficient similarity search in multimedia databases has been carried out by metric access methods (MAMs), where the utilized similarity measure had to satisfy the metric properties (reflexivity, non-negativity, symmetry, triangle inequality). Recently, the introduction of TriGen algorithm (turning any nonmetric into metric) enabled MAMs to perform also nonmetric similarity search. Moreover, it simultaneously enabled faster approximate search (either metric or nonmetric). However, a simple application of TriGen as the first step before MAMs' indexing assumes a fixed "approximation level", that is, a user-defined tolerance of retrieval precision is preset for the whole index lifetime. In this paper, we push the similarity search forward; we propose the NM-tree (nonmetric tree) – a modification of M-tree which natively aggregates the TriGen algorithm to support *flexible approximate* nonmetric or metric search. Specifically, at query time the NM-tree provides a user-defined level of retrieval efficiency/precision trade-off. We show the NM-tree could be used for general (non)metric search, while the desired retrieval precision can be flexibly tuned on-demand.

## 1 Introduction

As the digital devices for capturing multimedia data become massively available, the similarity search in multimedia databases steadily becomes more important. The metadata-based searching (using text/keywords/URL attached to multimedia documents, e.g., as at `images.google.com`) provides either limited search capabilities or even is not applicable (for raw data). On the other hand, the *content-based similarity retrieval* provides a native solution. The multimedia objects are retrieved based on their similarity to a query object (i.e., we suppose the *query-by-example* modality). The similarity measure is domain-specific – we could measure similarity of two images based on, for example, color histogram, texture layout, shape, or any combination. In most applications the similarity measure is regarded as computationally expensive.

In order to search a multimedia database efficiently enough, the database has to be indexed so that the volume of explicitly computed similarity scores to answer a query is minimized. That is, we try to avoid sequential scan over all the objects in the database, and their comparing against the query object.

S.S. Bhowmick, J. Küng, and R. Wagner (Eds.): DEXA 2008, LNCS 5181, pp. 312–325, 2008.

### 1.1 Metric Search

A few decades ago, the database-oriented research established a metric-based class of access methods for similarity search – the *metric access methods* (MAMs). The similarity measure $\delta$ (dissimilarity or distance, actually) is modeled by a metric distance function, which satisfies the properties of reflexivity, non-negativity, symmetry and triangle inequality. Based on these properties, the MAMs partition (or index) the metric data space into classes, so that only some of the classes have to be searched when querying; this results in a more efficient retrieval. To date, many MAMs were developed, addressing various aspects – main-memory/database-friendly methods, static/dynamic indexing, exact/approximate search, centralized/distributed indexing, etc. (see [21,15,4]). Although efficient in query processing, MAMs force their users to employ just the metric similarity measures, which is becoming a serious limitation nowadays.

### 1.2 Nonmetric Similarity

As the quantity/complexity of multimedia data grows, there is a need for more complex similarity measuring. Here the metric model exhibits its drawbacks, since the domain experts (being not computer scientists) are forced to "implant" metric properties into their nonmetric measures, which is often impossible.

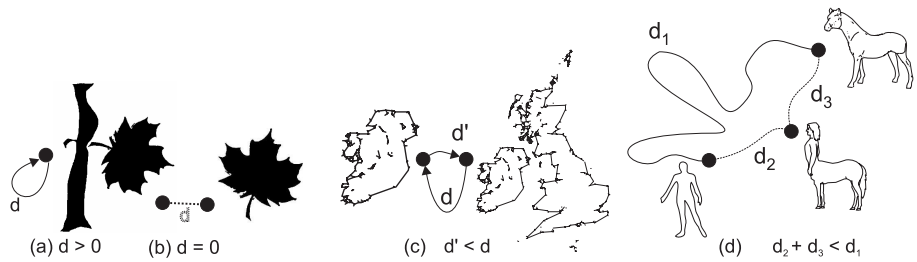


**Fig. 1.** Objections against metric properties in similarity measuring: (a) reflexivity (b) non-negativity (c) symmetry (d) triangle inequality

However, a nonmetric similarity has also a qualitative justification. In particular, the reflexivity and non-negativity have been refuted by claiming that different objects could be differently self-similar [11,19]. For example, in Figure 1a the leaf on a trunk can be viewed as positively self-dissimilar if we consider the less similar parts of the objects (here the trunk and the leaf). Or, alternatively, in Figure 1b the leaf-on-trunk and leaf could be treated as identical if we consider the most similar parts of the objects (the leaves). The symmetry was questioned by showing that a prototypical object can be less similar to an indistinct one than vice versa [13,14]. In Figure 1c, the more prototypical "Great Britain and Ireland" is more distant to the "Ireland alone" than vice versa. The triangle inequality is the most attacked property. Some theories point out the similarity has not to be transitive [1,20]. 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 1d).

### 1.3 Related Work

When compared to the rich research output in the area of metric access methods, there exist only few approaches to efficient nonmetric search. They include mapping methods [2,12,7], where the nonmetric space is turned into a vector space (mostly Euclidean). The distances in the target space are preserved more or less approximately, while the approximation error is fixed and/or not known. Similar approximate results achieve classification techniques [8,10]. Recently, an exact search over a nonmetric database was introduced, assuming the query distribution is restricted to the database distribution [5], while the indexing is very expensive (all pairwise distances on database objects must be computed).

**Universal Solution.** In our previous work we have introduced the TriGen algorithm [17,16] – a universal approach to searching in (non)metric spaces – where we addressed exact metric search, approximate metric search, (almost) exact nonmetric search and approximate nonmetric search. All these retrieval types can be achieved by turning the input (non)metric $\delta$ into a distance which satisfies the triangle inequality to some degree (including the full metric case). Such a modified measure can be used by any MAM for indexing/querying. However, as the proposed solution separates the measure conversion from the subsequent MAM-related issues, querying on an index built using the modified measure can be used to retrieval that is unchangeable in retrieval precision, that is, a retrieval always exact or always approximate to some fixed extent. If the user wants to adjust the desired precision, the entire database has to be reindexed.

**Paper Contribution.** In this paper we propose the NM-tree, a nonmetric (and also metric) access method extending the M-tree. The NM-tree natively utilizes the TriGen algorithm and provides retrieval precision adjustable at query time.

## 2 TriGen

The metric access methods are efficient because they use metric properties to index the database, especially the triangle inequality. However, in nonmetric spaces a dissimilarity measure $\delta$ is not constrained by any properties, so we have no clue for indexing. A way how to enable efficient nonmetric search is a transformation to the (nearly) metric case by so-called T-bases. The reflexivity, non-negativity and symmetry can be easily added to any nonmetric $\delta$ used for similarity search (see [17]). We also assume the values produced by $\delta$ are scaled into $\langle 0, 1\rangle$, which is achieved for free when fixing the reflexivity and non-negativity.

The hard task is enforcing the triangle inequality. The TriGen algorithm [17,16] can put more or less of the triangle inequality into any *semimetric* $\delta$ (i.e., into any reflexive, non-negative, symmetric distance), thus any semimetric distance can be turned into an equivalent full metric[1], or to a semimetric which

[1] In fact, the metric preserves the original query orderings (which is sufficient [17]).

satisfies the triangle inequality to some user-defined extent. Conversely, TriGen can also turn any full metric into a semimetric which preserves the triangle inequality only partially; this is useful for faster but only approximate search. For its functionality the TriGen needs a (small) sample of the database objects.

### 2.1 T-bases

The principle behind TriGen is a usage of triangle triplets and T-bases. A triplet of numbers $(a, b, c)$ is *triangle triplet* if $a + b \geq c, b + c \geq a, a + c \geq b$. The triangle triplets can be viewed as witnesses of triangle inequality of a distance $\delta$ – if all triplets $(\delta(O_i, O_j), \delta(O_j, O_k), \delta(O_i, O_k))$ on all possible objects $O_i, O_j, O_k$ are triangle triplets, then $\delta$ satisfies the triangle inequality. Using triangle triplets we measure the *T-error* – a degree of triangle inequality violation, computed as the proportion of non-triangle triplets in all examined distance triplets.

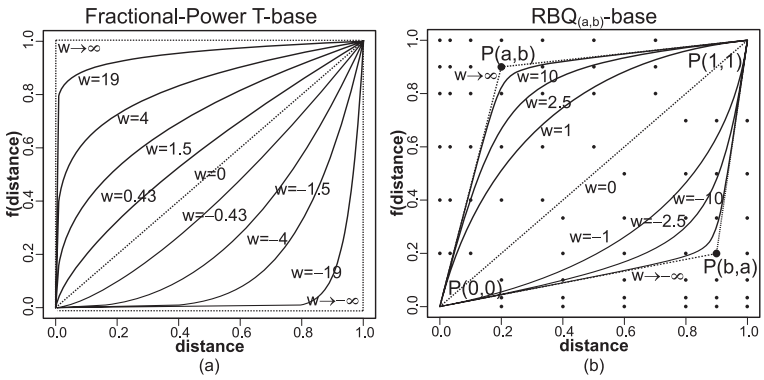


**Fig. 2.** T-bases: (a) FP-base (b) RBQ(a,b)-base; for their formulas see [17]

A *T-base* $f(x, w)$ is an increasing function (where $f(0, w) = 0$ & $f(1, w) = 1$) which turns a value $x \in \langle 0, 1 \rangle$ of an input (semi)metric $\delta$ into a value of a target (semi) metric $\delta^f$, i.e., $\delta^f(\cdot, \cdot) = f(\delta(\cdot, \cdot), w)$. Besides the input distance value $x$, the T-base is parameterized also by a fixed weight $w \in \langle -\infty, \infty \rangle$ which determines how concave or convex $f$ should be. The higher $w > 0$, the more concave $f$, which means also the lower T-error of any $\delta^f$. Conversely, the lower $w < 0$, the more convex $f$ and the higher T-error of any $\delta^f$ ($w = 0$ means $f$ is identity). For example, in Figure 2 see two T-bases, the *fractional power T-base* (FP-base) and one of the *rational Bézier quadratic T-bases* (RBQ-bases).

### 2.2 Intrinsic Dimensionality

When choosing very high $w$ (i.e., very concave $f$), we could turn virtually any semimetric $\delta$ into a full metric $\delta^f$. However, such a modification is not very useful. The more concave $f$, the higher *intrinsic dimensionality* [4,3] of the data space – a characteristic related to the mean and variance computed on the set of

pairwise distances within the data space. Simply, a high intrinsic dimensionality of the data leads to poor partitioning/indexing by any MAM (resulting in slower searching), and vice versa. On the other hand, the more convex $f$, the lower intrinsic dimensionality of the data space but also the higher the T-error – this results in fast but only approximate searching, because now the MAMs' assumption on fully preserved triangle inequality is incorrect. Hence, we have to make a trade-off choice – whether to search quickly but only approximately using a dissimilarity measure with higher T-error, or to search slowly but more precisely.

### 2.3 The TriGen Algorithm

Given a user-defined T-error tolerance $\theta$, a sample $\mathcal{S}$ of the database, and an input (semi)metric $\delta$, the TriGen's job is to find a modifier $f$ so that the T-error of $\delta^f$ is kept below $\theta$ and the intrinsic dimensionality of $(\mathcal{S}, \delta^f)$ is minimized[2]. For each of the predefined T-bases the minimal $w$ is found (by halving the weight interval), so that the weight $w$ cannot be further decreased without T-error exceeding $\theta$. Among all the processed T-bases and their final weights, the one is chosen which exhibits the lowest intrinsic dimensionality, and returned by TriGen as the *winning T-modifier* (for details of TriGen see [17]).

The winning T-modifier could be subsequently employed by any MAM to index and query the database using $\delta^f$. However, at this moment a MAM's index built using $\delta^f$ is not usable if we want to change the approximation level (the T-error of $\delta^f$), that is, to use another $f$. This is because MAMs accept the distance $\delta^f$ as a black box; they do not know it is a composition of $\delta$ and $f$. In such case we have to throw the index away and reindex the database by a $\delta^{f_2}$.

In this paper we propose the NM-tree, a MAM based on M-tree natively utilizing TriGen. In NM-tree, any of the precomputed T-modifiers $f_i$ can be flexibly chosen at query time, allowing the user to trade performance for precision.

## 3 M-tree

The *M-tree* [6] is a dynamic metric access method that provides good performance in database environments. The M-tree index is a hierarchical structure, where some of the data objects are selected as centers (references or local *pivots*) of ball-shaped regions, and the remaining objects are partitioned among the regions in order to build up a balanced and compact hierarchy, see Figure 3a. Each region (subtree) is indexed recursively in a B-tree-like (bottom-up) way of construction. The inner nodes of M-tree store *routing entries*

$$rout_l(O_i) = [O_i, r_{O_i}, \delta(O_i, Par(O_i)), ptr(T(O_i))]$$

where $O_i$ is a data object representing the center of the respective ball region, $r_{O_i}$ is a *covering radius* of the ball, $\delta(O_i, Par(O_i))$ is so-called *to-parent* distance

[2] As shown in [17,16], the real retrieval error of a MAM using $\delta^f$ is well estimated by the T-error of $\delta^f$, hence, $\theta$ can be directly used as a retrieval precision threshold.

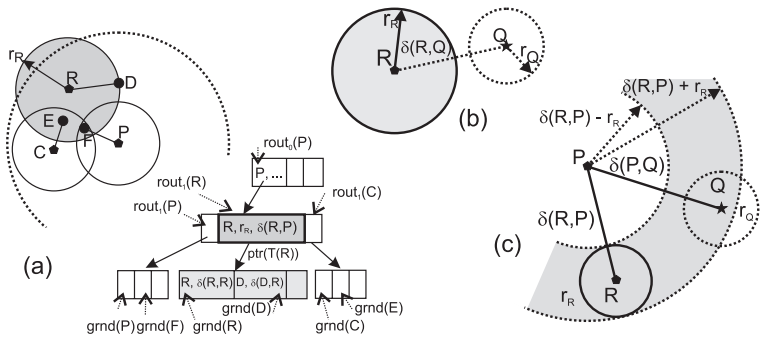


**Fig. 3.** (a) M-tree (b) Basic filtering (c) Parent filtering

(the distance from $O_i$ to the object of the parent routing entry), and finally $ptr(T(O_i))$ is a pointer to the entry's subtree. The data is stored in the leaves of M-tree. Each leaf contains *ground entries*

$$grnd(O_i) = [O_i, id(O_i), \delta(O_i, Par(O_i))]$$

where $O_i$ is the data object itself (externally identified by $id(O_i)$), and $\delta(O_i, Par(O_i))$ is, again, the to-parent distance. See an example of entries in Figure 3a.

### 3.1 Query Processing

The range and $k$ nearest neighbors (kNN) queries are implemented by traversing the tree, starting from the root[3]. Those nodes are accessed whose parent regions $(R, r_R)$ described by the routing entry are overlapped by the query ball $(Q, r_Q)$.

In case of a kNN query (we search for $k$ closest objects to $Q$), the query radius (or range) $r_Q$ is not known in advance, so we have to additionally employ a heuristic to dynamically decrease the radius during the search. The radius is initially set to the maximum distance in the metric space, that is, to 1.0 since we have assumed a dissimilarity measure scaled to $\langle 0, 1 \rangle$, see Section 2.

**Basic filtering.** The check for region-and-query overlap requires an explicit distance computation $\delta(R, Q)$, see Figure 3b. In particular, if $\delta(R, Q) \leq r_Q + r_R$, the data ball $R$ overlaps the query ball, thus the child node has to be accessed. If not, the respective subtree is filtered from further processing.

**Parent filtering.** As each node in the tree contains the distances from the routing/ground entries to the center of its parent node, some of the non-relevant M-tree branches can be filtered out without the need of a distance computation, thus avoiding the "more expensive" basic overlap check (see Figure 3c). In particular, if $|\delta(P, Q) - \delta(P, R)| > r_Q + r_R$, the data ball $R$ cannot overlap the query ball, thus the child node has not to be re-checked by basic filtering. Note $\delta(P, Q)$ was computed in the previous (unsuccessful) parent's basic filtering.

[3] We outline just the principles, for details see the original M-tree algorithms [6,18].

# 4 NM-tree

The NM-tree is an extension of M-tree in terms of algorithms, while the data structure itself is unchanged. The difference in indexing relies in encapsulating the M-tree insertion algorithm by the more general NM-tree insertion (see Section 4.1). The query algorithms have to be slightly redesigned (see Section 4.2).

## 4.1 Indexing

The distance values (to-parent distances and covering radii) stored in NM-tree are all metric, that is, we construct a regular M-tree using a full metric. Since the NM-tree's input distance $\delta$ is generally a semimetric, the TriGen algorithm must be applied before indexing, in order to turn $\delta$ into a metric $\delta^{f_M}$ (i.e., searching for a T-modifier $f_M$ under $\theta = 0$). However, because at the beginning of indexing the NM-tree is empty, there is no database sample available for TriGen. Therefore, we distinguish two phases of indexing and querying on NM-tree.

**Algorithm 1** *(dynamic insertion into NM-tree)*

```
method InsertObject(O_new) {
 if database size < smallDBlimit then
   store O_new into sequential file
 else
   insert O_new into NM-tree (using original M-tree insertion algorithm under δ^{f_M})
 endif
 if database size = smallDBlimit then
   run TriGen algorithm on the database, having θ_M = 0, θ_1, θ_2, ..., θ_k > 0 ⇒
     obtaining T-bases f_M, f_{e_1}, f_{e_2}, ..., f_{e_k} with weights w_M, w_{e_1}, w_{e_2}, ..., w_{e_k}
   for each object O_i in the sequential file
     insert O_i into NM-tree (using original M-tree insertion algorithm under δ^{f_M})
   empty the sequential file
 end if }
```

For the whole picture of indexing in NM-tree, see Algorithm 1. The first phase just gathers database objects until we get a sufficiently large set of database objects. In this phase a possible query is solved sequentially, but this is not a problem because the indexed database is still small. When the database size reaches a certain volume (say, $\approx 10^4$ objects, for example), the TriGen algorithm is used to compute $f_M$ using the database obtained so far. At this moment we run the TriGen also for other, user-defined $\theta_i$ values, so that alternative T-modifiers will be available for future usage (for approximate querying). Finally, the first phase is terminated by indexing the gathered database using a series of the original M-tree insertions under the metric $\delta^{f_M}$ (instead of $\delta$). In the second phase the NM-tree simply forwards the insertions to the underlying M-tree.

**Notice:** In contrast to the original TriGen [17], in NM-tree we require the T-bases $f_i$ to be additionally *inversely symmetric*, that is, $f_i(f_i(x, w), -w) = x$. In other words, when knowing a T-base $f_i$ with some weight $w$, we know also the inverse $f_i^{-1}(\cdot, w)$, which is determined by the same T-base and $-w$. The FP-base and all RBQ-bases (see Section 2.1) are inversely symmetric.

### 4.2 Query Processing

When querying, we distinguish two cases – exact search and approximate search.

**Exact search.** The exact case is simple, when the user issues a query with zero desired retrieval error, the NM-tree is searched by the original M-tree algorithms, because of employing $\delta^{f_M}$ for searching, which is the full metric used also for indexing. The original user-specified radius $r_Q$ of a range query $(Q, r_Q)$ must be modified to $f_M(r_Q)$ before searching. After the query result is obtained, the distances of the query object $Q$ to the query result objects $O_i$ must be modified inversely, that is, to $f_M^{-1}(\delta^{f_M}(Q, O_i))$ (regardless of range or kNN query).

**Approximate search.** The approximate case is more difficult, while here the main qualitative contribution of NM-tree takes its place. Consider user issues a query which has to be processed with a retrieval error $e_i \in \langle 0, 1 \rangle$, where for $e_i = 0$ the answer has to be precise (with respect to the sequential search) and for $0 > e_i \geq 1$ the answer may be more or less approximate. The $e_i$ value must be one of the T-error tolerances $\theta_i$ predefined before indexing (we suppose the T-error models the actual retrieval error, i.e., $e_i = \theta_i$).

An intuitive solution for approximate search would be a modification of the required $\delta^{f_M}$-based distances/radii stored in NM-tree into $\delta^{f_{e_i}}$-based distances/radii. In such case we would actually get an M-tree indexed by $\delta^{f_{e_i}}$, as used in [17], however, a dynamic one – a single NM-tree index would be interpreted as multiple M-trees indexed by various $\delta^{f_{e_i}}$ distances. Unfortunately, this "online interpretation" is not possible, because NM-tree (M-tree, actually) stores not only direct distances between two objects (the to-parent distances) but also radii, which consist of aggregations. In other words, except for the two deepest levels (leaf and pre-leaf level), the radii stored in routing entries are composed from two or more direct distances (a consequence of node splitting). To correctly re-modify a radius into the correct one, we would need to know all the components in the radius, but these are not available in the routing entry.

Instead of "emulating" multiple semimetric M-trees as mentioned above, we propose a technique performing the exact metric search at higher levels and approximate search just at the leaf and pre-leaf level. In Figure 4 see all the distances/radii which are modified to semimetric ones during the search, while the modification is provided by T-bases associated with their user-defined retrieval errors. Besides the to-parent distances, we also consider the query radius and covering radii at the pre-leaf level, because these radii actually represent real distances to a furthest object in the respective query/data region. The query radius and entry-to-query distances (computed as $\delta(\cdot, \cdot)$) are not stored in NM-tree, so these are modified simply by $f_{e_i}$ (where $f_{e_i}$ is a T-base modifier obtained for retrieval error $e_i$). The remaining distances stored in NM-tree ($\delta^{f_M}(\cdot, \cdot)$-based to-parent distances and covering radii), have to be modified back to the original ones and then re-modified using $f_{e_i}$, that is, $f_{e_i}(f_M^{-1}(\delta^{f_M}(\cdot, \cdot)))$.

**Algorithm 2** *(NM-tree range query)*

```
RangeQuery(Node N, RQuery (Q, r_Q), retrieval error e_k) {
 let O_p be the parent routing object of N // if N is root then δ(O_i, O_p)=δ(O_p, Q)=0

 if N is not a leaf then {
   if N is at pre-leaf level then {                                          // pre-leaf level
     for each rout(O_i) in N do {
       if |f_{e_k}(δ(O_p, Q)) − f_{e_k}(f_M^{-1}(δ^{f_M}(O_i, O_p)))| ≤ f_{e_k}(r_Q) + f_{e_k}(f_M^{-1}(r_{O_i}^{f_M})) then {   // (parent filt.)
         compute δ(O_i, Q)
         if f_{e_k}(δ(O_i, Q)) ≤ f_{e_k}(r_Q) + f_{e_k}(f_M^{-1}(r_{O_i}^{f_M})) then   // (basic filtering)
           RangeQuery(ptr(T(O_i)), (Q, r_Q), e_k)
       }
     } // for each ...
   } else {                                                                  // higher levels
     for each rout(O_i) in N do {
       if |f_M(δ(O_p, Q)) − δ^{f_M}(O_i, O_p)| ≤ f_M(r_Q) + r_{O_i}^{f_M} then {   // (parent filtering)
         compute δ(O_i, Q)
         if f_M(δ(O_i, Q)) ≤ f_M(r_Q) + r_{O_i}^{f_M} then   // (basic filtering)
           RangeQuery(ptr(T(O_i)), (Q, r_Q), e_k)
       }
     } // for each ...
   }
 } else {                                                                    // leaf level
   for each grnd(O_i) in N do {
     if |f_{e_k}(δ(O_p, Q)) − f_{e_k}(f_M^{-1}(δ^{f_M}(O_i, O_p)))| ≤ f_{e_k}(r_Q) then {   // (parent filtering)
       compute δ(O_i, Q)
       if δ(O_i, Q) ≤ r_Q then
         add O_i to the query result
     }
   } // for each ...
 }
```

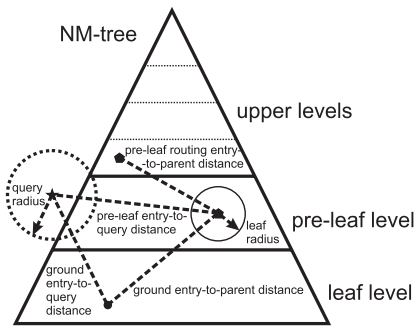


**Fig. 4.** Dynamically modified distances when searching approximately

In Algorithm 2 see the modified range query algorithm[4]. In the pseudocode the "metrized" distances/radii stored in the index are denoted as $\delta^{f_M}(\cdot,\cdot)$, $r_{O_i}^{f_M}$, while an "online" distance/radius modification is denoted as $f_{e_k}(\cdot)$, $f_M^{-1}(\cdot)$. If removed $f_M$, $f_M^{-1}$, $f_{e_k}$ from the pseudocode, we would obtain the original M-tree range query, consisting of parent and basic filtering steps (see Section 3.1).

[4] For the lack of space we omit the kNN algorithm, however, the modification is similar.

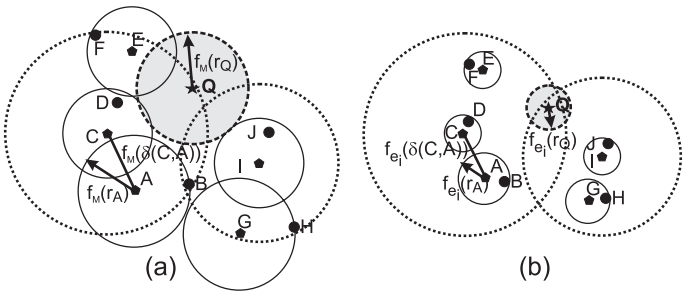


**Fig. 5.** (a) Exact (metric) search (b) Approximate search

In Figure 5 see a visualization of exact and approximate search in NM-tree. In the exact case, the data space is inflated into a (nearly) metric space, so the regions tend to be huge and overlap each other. On the other hand, for approximate search the leaf regions (and the query region) become much tighter, the overlaps are less frequent, so the query processing becomes more efficient.

## 5 Experimental Results

To examine the NM-tree capabilities, we performed experiments with respect to the efficiency and retrieval error, when compared with multiple M-trees (each using a fixed modification of $\delta$, related to a user-defined T-error tolerance). We have focused just on the querying, since the NM-tree's efficiency of indexing is the same as that of M-tree. The query costs were measured as the number of $\delta$ computations needed to answer a query. Each query was issued 200 times for different query objects and the results were averaged. The precision of approximate search was measured as the real retrieval error (instead of just T-error); the normed overlap distance $E_{NO}$ between the query result $QR_{NMT}$ returned by the NM-tree (or M-tree) and the correct query result $QR_{SEQ}$ obtained by sequential search of the database, i.e. $E_{NO} = 1 - \frac{|QR_{NMT} \cap QR_{SEQ}|}{max(|QR_{NMT}|,|QR_{SEQ}|)}$.

### 5.1 The TestBed

We have examined 4 dissimilarity measures on two databases (images, polygons), while the measures $\delta$ were considered as black-box semimetrics. All the measures were normed to $\langle 0, 1 \rangle$. The database of images consisted of 68,040 32-dimensional *Corel features* [9] (the color histograms were used). We have tested one semimetric and one metric on the images: the $L_{\frac{3}{4}}$ distance [17] (denoted **L0.75**), and the Euclidean distance (**L2**). As the second, we created a synthetic dataset of 250,000 2D polygons, each consisting of 5 to 15 vertices. We have tested one semimetric and one metric on the polygons: the dynamic time warping distance with the $L_2$ inner distance on vertices [17] (denoted **DTW**) and the Hausdorff distance, again with the $L_2$ inner distance on vertices [17] (denoted **Hausdorff**).

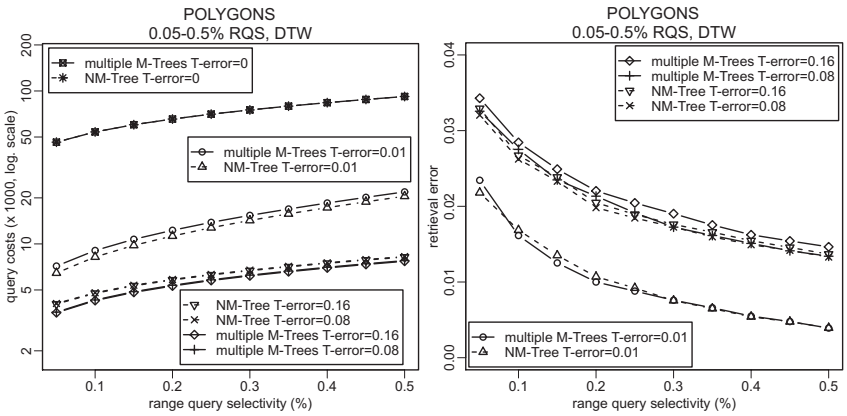


**Fig. 6.** Range queries on Polygons under DTW

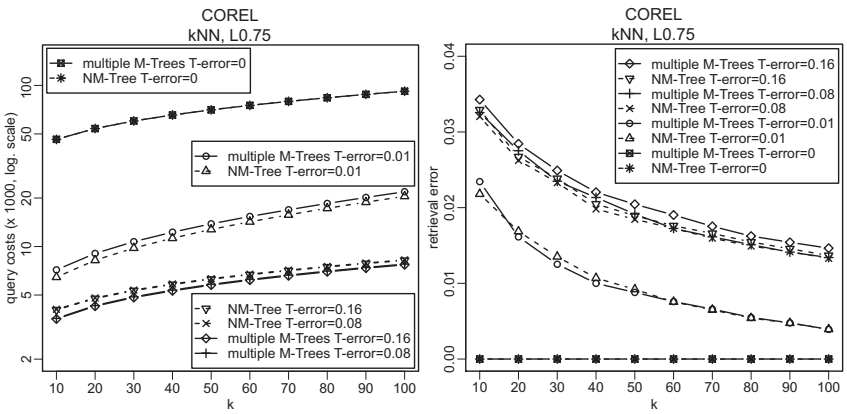


**Fig. 7.** kNN queries on Corel under $L_{\frac{3}{4}}$

The TriGen inside NM-tree was configured as follows: the T-base pool consisting of the FP-base and 115 RBQ-bases (as in [17]), sample size 5% of Corel, 1% of Polygons. The NM-tree construction included creation of $4 \cdot 10 = 40$ T-modifiers by TriGen (concerning all the dissimilarity measures used), defined by T-error tolerances [0, 0.0025, 0.005, 0.01, 0.015, 0.02, 0.04, 0.08, 0.16, 0.32] used by querying. The node capacity of (N)M-tree was set to 30 entries per node (32 per leaf); the construction method was set to SingleWay [18]. The (N)M-trees had 4 levels (leaf + pre-leaf + 2 higher) on both Corel and Polygons databases. The leaf/inner nodes were filled up to 65%/69% (on average).

### 5.2 Querying

In the first experiment we have examined query costs and retrieval error of range queries on Polygons under DTW, where the range query selectivity (RQS) ranged

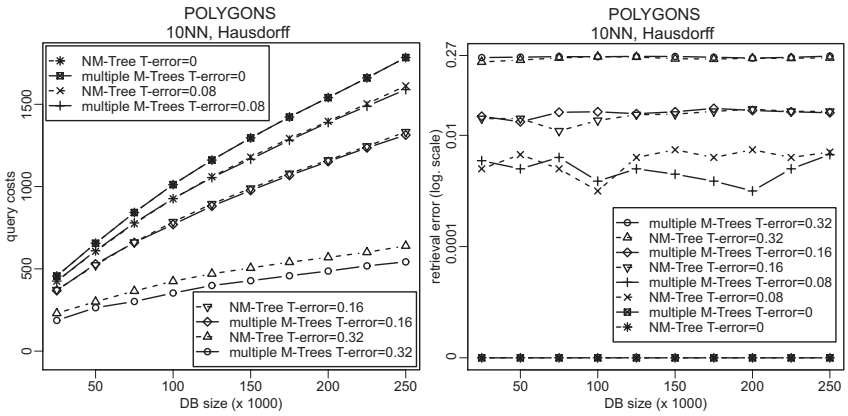


**Fig. 8.** 10NN queries on varying size of Polygons under Hausdorff

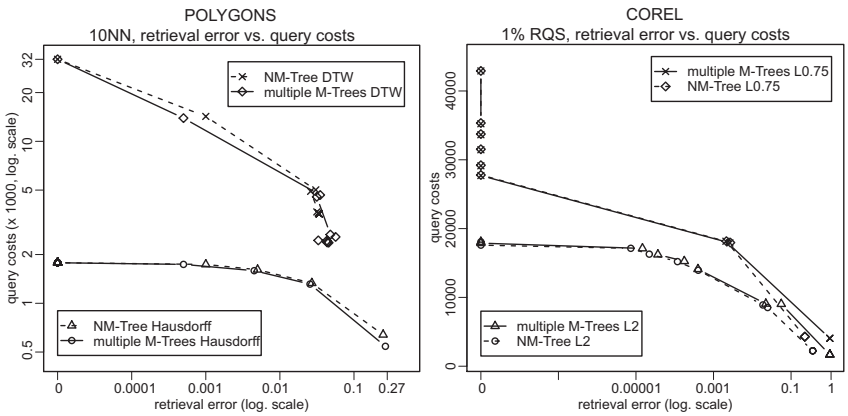


**Fig. 9.** Aggregated performance of 10NN queries for Polygons and Corel

from 0.05% to 0.5% of the database size (i.e., 125–1250 objects), see Figure 6. We can see that a single NM-tree index can perform as good as multiple M-tree indexes (each M-tree specifically created for a user-defined retrieval error). In most cases the NM-tree is even slightly better in both observed measurements – query costs and retrieval error.

Note that here the NM-tree is an order of magnitude faster than sequential file when performing exact (nonmetric!) search, and even two orders of magnitude faster in case of approximate search (while keeping the retrieval error below 1%). The Figure 6 also shows that if the user allows a retrieval error as little as 0.5–1%, the NM-tree can search the Polygons an order of magnitude faster, when compared to the exact (N)M-tree search.

In the second experiment we have observed the query costs and retrieval error for kNN queries on the Corel database under nonmetric $L_{\frac{3}{4}}$ (see Figure 7). The

results are very similar to the previous experiment. We can also notice (as in the first experiment) that with increasing query result the retrieval error decreases.

Third, we have observed 10NN queries on Polygons under Hausdorff, with respect to the growing database size, see Figure 8. The query costs growth is slightly sub-linear for all indexes, while the retrieval errors remain stable. Note the T-error tolerance levels (attached to the labels in legends) specified as an estimation of the maximal acceptable retrieval error are apparently a very good model for the retrieval error.

In the last experiment (see Figure 9) we have examined the aggregated performance of 10NN queries for both Polygons and Corel and all the dissimilarity measures. These summarizing results show the trade-off between query costs and retrieval error achievable by an NM-tree (and the respective M-trees).

## 6 Conclusions

We have introduced the NM-tree, an M-tree-based access methods for exact and approximate search in metric and nonmetric spaces, which incorporates the TriGen algorithm to provide nonmetric and/or approximate search. The main feature on NM-tree is its flexibility in approximate search, where the user can specify the approximation level (acceptable retrieval error) at query time. The experiments have shown that a single NM-tree index can search as fast as if used multiple M-tree indexes (each built for a certain approximation level). From the general point of view, the NM-tree, as the only access method for flexible exact/approximate nonmetric similarity search can achieve up to two orders of magnitude faster performance, when compared to the sequential search.

## Acknowledgments

This research has been partially supported by Czech grants: "Information Society program" number 1ET100300419 and GAUK 18208.

## References

1. Ashby, F., Perrin, N.: Toward a unified theory of similarity and recognition. Psychological Review 95(1), 124–150 (1988)
2. Athitsos, V., Hadjieleftheriou, M., Kollios, G., Sclaroff, S.: Query-sensitive embeddings. In: SIGMOD 2005: Proceedings of the 2005 ACM SIGMOD international conference on Management of data, pp. 706–717. ACM Press, New York (2005)
3. Chávez, E., Navarro, G.: A Probabilistic Spell for the Curse of Dimensionality. In: Buchsbaum, A.L., Snoeyink, J. (eds.) ALENEX 2001. LNCS, vol. 2153, pp. 147–160. Springer, Heidelberg (2001)
4. Chávez, E., Navarro, G., Baeza-Yates, R., Marroquín, J.L.: Searching in metric spaces. ACM Computing Surveys 33(3), 273–321 (2001)
5. Chen, L., Lian, X.: Efficient similarity search in nonmetric spaces with local constant embedding. IEEE Transactions on Knowledge and Data Engineering 20(3), 321–336 (2008)

6. Ciaccia, P., Patella, M., Zezula, P.: M-tree: An Efficient Access Method for Similarity Search in Metric Spaces. In: VLDB 1997, pp. 426–435 (1997)
7. Farago, A., Linder, T., Lugosi, G.: Fast nearest-neighbor search in dissimilarity spaces. IEEE Transactions on Pattern Analysis and Machine Intelligence 15(9), 957–962 (1993)
8. Goh, K.-S., Li, B., Chang, E.: DynDex: a dynamic and non-metric space indexer. In: ACM Multimedia (2002)
9. Hettich, S., Bay, S.: The UCI KDD archive (1999), `http://kdd.ics.uci.edu`
10. Jacobs, D., Weinshall, D., Gdalyahu, Y.: Classification with nonmetric distances: Image retrieval and class representation. IEEE Pattern Analysis and Machine Intelligence 22(6), 583–600 (2000)
11. Krumhansl, C.L.: Concerning the applicability of geometric models to similar data: The interrelationship between similarity and spatial density. Psychological Review 85(5), 445–463 (1978)
12. Kruskal, J.B.: Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis. Psychometrika 29(1), 1–27 (1964)
13. Rosch, E.: Cognitive reference points. Cognitive Psychology 7, 532–547 (1975)
14. Rothkopf, E.: A measure of stimulus similarity and errors in some paired-associate learning tasks. J. of Experimental Psychology 53(2), 94–101 (1957)
15. Samet, H.: Foundations of Multidimensional and Metric Data Structures. Morgan Kaufmann, San Francisco (2006)
16. Skopal, T.: On fast non-metric similarity search by metric access methods. In: Ioannidis, Y., Scholl, M.H., Schmidt, J.W., Matthes, F., Hatzopoulos, M., Böhm, K., Kemper, A., Grust, T., Böhm, C. (eds.) EDBT 2006. LNCS, vol. 3896, pp. 718–736. Springer, Heidelberg (2006)
17. Skopal, T.: Unified framework for fast exact and approximate search in dissimilarity spaces. ACM Transactions on Database Systems 32(4), 1–46 (2007)
18. Skopal, T., Pokorný, J., Krátký, M., Snášel, V.: Revisiting M-tree Building Principles. In: Kalinichenko, L.A., Manthey, R., Thalheim, B., Wloka, U. (eds.) ADBIS 2003. LNCS, vol. 2798, pp. 148–162. Springer, Heidelberg (2003)
19. Tversky, A.: Features of similarity. Psychological review 84(4), 327–352 (1977)
20. Tversky, A., Gati, I.: Similarity, separability, and the triangle inequality. Psychological Review 89(2), 123–154 (1982)
21. Zezula, P., Amato, G., Dohnal, V., Batko, M.: Similarity Search: The Metric Space Approach (Advances in Database Systems). Springer, Secaucus (2005)