The M^2-tree: Processing Complex Multi-Feature Queries with Just One Index

Paolo Ciaccia, Marco Patella

DEIS - CSITE-CNR, University of Bologna, Italy

{pciaccia,mpatella}@deis.unibo.it

Abstract

Motivated by the needs for efficient similarity retrieval in multimedia digital libraries, we present basic principles of a new paged and balanced index structure, the M^2-tree. The M^2-tree can be applied whenever "complex" range and/or best matches queries over different descriptions (*features*) of objects need to be solved. The proposed approach combines within a single index structure information from multiple metric spaces, thus being able to efficiently support queries on arbitrary combinations of indexed features. Efficiency of the structure is presented through preliminary experimental results over a real-world data-set.

1 Introduction

Similarity queries are a primary concern for supporting content-based retrieval in multimedia digital libraries (MM-DLs). A common approach views similarity search as a search for close points in some (high-dimensional) feature space, with closeness measured through some domain- (and possibly user-) specific distance function. For instance, retrieving images with similar colors can be approached by representing the color content of an image through an histogram (i.e. a vector) and then by measuring the Euclidean distance between histograms. For the efficient processing of "simple" similarity queries, i.e., where only one matching criterion/predicate is specified, several index structures have been proposed, including multi-dimensional (spatial) trees, such as the X-tree [1] and the SR-tree [7], metric trees (M-tree [3], Slim-tree [9], etc.), and signature-based approaches, such as the VA-file [10]. These solutions, however, are not suitable for the more general case where *more than one* similarity criterion is specified, each referring to a specific object's feature, for instance: *"Find video news where Bill Clinton is talking about the Kosovo War"*. Exploration of MM-DLs using such complex (multimodal) queries is the rule rather than the exception. For instance, a major objectives of the Informedia-II project at CMU is to: *"... allow multidimensional queries that may combine image elements, video clips, text and speech"*.[1]

The problem we address in this paper is how to efficiently support complex similarity queries over large MM-DLs. Since we do not want to limit ourselves to a particular domain, the solution we seek should also satisfy the two following requirements:

1. **(Generality)** Due to the complex nature of MM objects, it should work not only on vector spaces, but it should also be able to deal with the more general case of metric spaces, i.e. when feature values can only be compared using a distance function.

2. **(Flexibility)** We would also retain the possibility to issue queries over only a subset of objects' features, as well as to vary at query time the relevance of each feature in determining the final result.

Complex similarity queries are composed of simple predicates, each involving a single feature. Such predicates are then combined so as to yield, for each object, an overall evaluation score (see Section 2.1). Current approaches for processing complex queries can be summarized as follows.

Synchronized evaluation of predicates. This approach, well represented by Fagin's $\mathcal{A}_0$ [5] and Quick-Combine [6] algorithms, deals with *best-matches* (also called k-nearest neighbors or k-NN) queries, where the k objects with the best overall scores are sought. It is assumed that each predicate can be efficiently evaluated by an index, and that indices provide a *sorted access* scan modality, with a `GetNext()` method that returns the best match for that predicate among the not-seen-yet objects. At each step, the $\mathcal{A}_0$ algorithm checks whether the best solution computed so far can be improved by using objects not yet retrieved

---

[1]The CMU Informedia-II Website: `http://www.informedia.cs.cmu.edu/dli2/`.

by the scans. If not, the sorted access can be stopped; otherwise, at least another step is required. To compute the final result, a second *random access* phase is required to evaluate, by taking into account scores for simple predicates, the overall score for each object retrieved during the previous sorted access phase. Unfortunately, the performance of this approach rapidly deteriorates with the number of predicates.

Statistics-based evaluation. In [2], the authors propose to transform a k-NN query into a multi-dimensional range query by exploiting statistics on data distributions. However, this approach works only for (low-dimensional) vector spaces, and can only support a limited class of scoring functions.

Signature-based approach. The VA-file [10], which is a sequential structure that stores binary approximations of high-D feature values, can be straightforwardly used to process complex queries by building a VA-file over each feature in the query. However, this solution only deals with vector spaces and its complexity scales linearly with the data-set size.

"Collapsing" approach. Finally, one could use a single index for the combination of all features. Since we are looking for a "metric solution" (1st requirement), this would require to predetermine an overall distance function according to which objects could be organized, which is in conflict with our 2nd requirement.

In conclusion, no known processing technique can efficiently support complex queries and, at the same time, satisfies both our requirements.

2   The M^2-tree

Consider a collection $\mathcal{C}$ of database objects than can be described by way of a set $\mathcal{F} = \{F_1, \ldots, F_m\}$ of features (in this work, we do not consider the problem of choosing the number and the type of features which are more suitable, from the efficiency and/or effectiveness point of view, for the domain at hand). For each feature F_i, whose values are drawn from a domain $\mathcal{D}_i = dom(F_i)$, (dis-)similarity between feature values is assessed by way of a *distance function* $d_i : \mathcal{D}_i^2 \rightarrow \Re_0^+$, which, for any pair of feature values from $\mathcal{D}_i$, yields a non-negative real value, being understood that low distances correspond to similar values and high distances to dissimilar values.

In the following, we assume that d_i is a metric, i.e. a non-negative and symmetric function which also satisfies the triangle inequality. Each couple of feature domain and distance function, thus, forms a metric space $(\mathcal{D}_i, d_i)$.

2.1   Query Model

We consider generic (simple) predicates p having the form $F_i \sim q$, where $q \in \mathcal{D}_i$ is a constant, also called *query value*, and $\sim$ is a (dis-)similarity operator. Evaluating p on an object $O \in \mathcal{C}$ equals to compute $d_i(q, O.F_i)$, where with $O.F_i$ we denote the value of feature F_i extracted from object O. In the following, for ease of representation, we will write $\delta(p, O) = d_i(q, O.F_i)$, meaning that assessing the distance between a simple predicate $p : F_i \sim q$ and an object O equals to compute the d_i function between the query value q and the feature value $O.F_i$. Metric trees, like the M-tree [3], are able to solve range and k-NN queries based on simple predicates.

Since our objective is to generalize to *complex* queries, we need a way to combine multiple predicates, possibly referencing different features, into a formula f in order to obtain a single distance value, to compare with the user-provided threshold, for a complex range query, or on which to order objects, for a complex k-NN query. We only require that, if $f = f(p_1, \ldots, p_n)$ is a formula composed of predicates $p_1, \ldots p_n$, then the overall distance value of an object O with respect to f, denoted as $\delta(f, O)$ is computed by way of a corresponding *scoring function* [5], d_f, taking as input the distances of O with respect to each predicate p_j of f, that is:

$$\delta(f(p_1, \ldots, p_n), O) = d_f(\delta(p_1, O), \ldots, \delta(p_n, O)) \tag{1}$$

Although, in line of principle, any kind of scoring function would do the job, in this work we only consider *monotonic* scoring functions, in the sense of the following definition.

Definition 1 (Monotonicity) *We say that a scoring function $d_f(\delta(p_1, O), \ldots, \delta(p_n, O))$ is monotonic increasing (respectively decreasing) in the variable δ_j if, given any two n-tuples of distance values $(\delta_1, \ldots, \delta_j, \ldots, \delta_n)$ and $(\delta_1, \ldots, \delta_j', \ldots, \delta_n)$ with $\delta_j \leq \delta_j'$, it is $d_f(\delta_1, \ldots, \delta_j, \ldots, \delta_n) \leq d_f(\delta_1, \ldots, \delta_j', \ldots, \delta_n)$ (respectively $d_f(\delta_1, \ldots, \delta_j, \ldots, \delta_n) \geq d_f(\delta_1, \ldots, \delta_j', \ldots, \delta_n)$). If a scoring function d_f is monotonic increasing (resp. decreasing) in all its variables, we simply say that d_f is monotonic increasing (resp. decreasing).* □

Such property is surely reasonable from a user's point of view. It is also the case that commonly used scoring functions are monotonic in all their arguments: For example, the `min` and `max` functions are monotonic increasing. As another example, weighted sums, with $d_f(\delta_1, \ldots, \delta_n) = \sum_{j=1}^{n} \theta_j \delta_j$, where θ_j's are positive weights, and $\sum_{j=1}^{n} \theta_j = 1$, are monotonic increasing [4].

Given a formula f whose scoring function d_f is monotonic in all its arguments, we consider *range queries* $\text{range}_{f,r}(\mathcal{C})$, selecting all the objects whose overall distance (computed by way of the scoring function d_f) to f is not higher than a specified threshold r ($\delta(f, O) \leq r$), and *k-NN queries* $\text{NN}_{f,k}(\mathcal{C})$, selecting the k objects having the lowest $\delta(f, O)$ distance to f.

It should be noted that metric trees are already able to efficiently process complex similarity queries when all the predicates refer to a *single* feature ($F_{i_1} = F_{i_2} = \ldots = F_{i_n}$) [4]. In this work we deal with the more general case of multi-feature queries.

2.2 Principles

Our approach reconciles the two requirements of Section 1 by providing a solution which can be seen as a "multi-dimensional" extension of the M-tree [3], much like as spatial access methods can be viewed, at some extent, as generalizations of the B$^+$-tree to D-dimensional vector spaces. The following table summarizes this point.

		no. of "coordinates"	
		1	many
Space type	Vector	B$^+$-tree	R-tree, X-tree, ...
	Metric	M-tree, ...	**M^2-tree**

For metric indices, like the M-tree, we say that they use 1 "coordinate" since they can organize objects by using only 1 distance function. Given a query value q_j, a metric space can be viewed as an half-line departing from q_j, the so-called *distance space*. All points of $\mathcal{D}_j$ are mapped to this line with a distance from the origin which is equal to their distance from q_j. A simple range query is equivalent to an interval query over such space (see Figure 1 (a)). If we consider a complex formula $f(p_1, \ldots, p_n)$, with $p_j : F_{i_j} \sim q_j$, each q_j induces an independent distance space on $\mathcal{D}_{i_j}$, and we obtain what can be conventionally called a *multiple distance space*.[2] Thus, a complex range query is equivalent to a region query over such space (see Figure 1 (b)). Therefore, just as spatial access methods are generalizations of the B$^+$-tree, since they are able to organize multi-dimensional feature (scalar) spaces, the M^2-tree generalizes the M-tree in the sense that it can support multiple distance spaces.

Figure 1: A simple range query in a distance space (a) and a complex range query in a multiple distance space (b).

From the above table it can be seen that the M^2-tree: (1) can deal with the more general case of metric spaces, and (2) can index objects by using *many* distance functions at the time. These are indeed the key ingredients to satisfy both requirements expressed in Section 1.

Just as the M-tree, the M^2-tree is a balanced and dynamic tree whose fixed-size nodes are mapped to disk pages and where indexed feature values are stored in the leaf nodes. To highlight the main difference between M-tree and M^2-tree, it is useful to show how regions associated with each node of these structures are defined:

- Each node N of the M-tree correspond to a *region* of the indexed metric space $(\mathcal{D}, d)$. The region of node N is $Reg(N) = \{v \in \mathcal{D} | d(v, v^{[N]}) \leq r^{[N]}\}$, where $v^{[N]}$ is the *routing value* of node N and $r^{[N]}$ is its *covering radius*. All the objects in the sub-tree rooted at N are then guaranteed to belong to $Reg(N)$, thus their distance from $v^{[N]}$ does not exceed $r^{[N]}$ (the region is equivalent to the interval $[0, r^{[N]}]$ of the distance space induced by $v^{[N]}$).

---

[2]It should be noted that each coordinate in such space corresponds to a single (possibly multi-dimensional) feature space.

- For an M^2-tree built over a set of m ($m \geq 1$) features $\mathcal{F} = \{F_1, \ldots, F_m\}$, with corresponding metric spaces $(\mathcal{D}_i, d_i)$ ($i = 1, \ldots, m$), the region associated to node N is now defined as:

$$Reg(N) = \{(v_1, \ldots, v_m) \in \mathcal{D}_1 \times \ldots \mathcal{D}_m | d_i(v_i, v_i^{[N]}) \leq r_i^{[N]}, i = 1, \ldots, m\}$$

where $v_i^{[N]}$ is the routing value for the i-th feature and $r_i^{[N]}$ is the corresponding covering radius. Thus, $Reg(N)$ consists of the hyper-rectangle $[0, r_1^{[N]}] \times \ldots \times [0, r_m^{[N]}]$ in the m-dimensional distance space induced by the routing values.

2.3   Format of M^2-tree Entries

Each entry e in an M^2-tree node consists of a set of m values $v_i \in \mathcal{D}_i$ for each considered feature F_i. For entries in a leaf node, such values correspond to the features extracted from each indexed object, whereas, for internal nodes, such values are obtained by way of a specific *promotion* algorithm (for space reason, we do not give details here of M^2-tree maintenance algorithms). Entries of internal nodes also include a set of m covering radii $r_i^{[N]}$ and a pointer $\mathtt{ptr}(T(e))$, referencing the root N of a sub-tree $T(e)$, whereas entries in leaf nodes include an identifier $\mathtt{oid}(e)$, which is used to provide access to the whole object, which may reside in a separate data file. The semantics of the covering radii is similar to that of M-tree: All the objects stored in the sub-tree pointed by $\mathtt{ptr}(T(e))$ are within distance $r_i^{[N]}$, considering the metric d_i, from $v_i^{[N]}$, i.e. $\forall O \in T(e), \forall i = 1, \ldots, m, d_i(v_i^{[N]}, O.F_i) \leq r_i^{[N]}$. As for M-tree, the distances between entry feature values and parent routing values $d_i(v_i, v_i^{[N]})$ are stored within each entry in order to prune sub-trees during the search phase.

Example 1 *We have to index a collection of images, which have been manually annotated by different human operators, who assigned a set of keywords to each of them. We also would like to search the collection for images having similar color distributions. To this end, a color descriptor is extracted from each image by using the technique described in [8], resulting in a 9-dimensional vector (see Figure 2). Distance between images for the description feature is assessed as the number of common keywords divided by the total number of keywords for the images, i.e. $d_1(v_{i_1}, v_{j_1}) = 1 - \frac{\|v_{i_1} \cap v_{j_1}\|}{\|v_{i_1} \cup v_{j_1}\|}$, whereas distance between color descriptors is computed as the Euclidean distance between associated vectors, i.e. $d_2(v_{i_2}, v_{j_2}) = \sqrt{\sum_{h=1}^{9} |v_{i_2}[h] - v_{j_2}[h]|^2}$*

image	name	keywords	color distribution vector
	tiger.bmp	nature, animals, mammals, feline, tiger	$(1.73, 0.381, 0.483, 1.97, 0.407, 0.518, 0.957, 0.133, 0.0903)$
	lion.jpg	nature, animals, mammals, feline, lion	$(1.32, 0.380, 0.729, 1.60, 0.480, 0.776, -2.27, -0.686, -1.34)$
	tiger_cat.jpg	animals, domestic, feline, cat	$(0.754, 0.325, 0.730, 0.922, 0.409, 0.757, 1.58, 0.645, 1.41)$
	tiger_shrimp.jpg	nature, animals, crustacean, tiger, shrimp	$(3.23, 0.104, 0.853, 3.42, 0.189, 0.884, -1.24, 0.222, -0.282)$

Figure 2: Four images with their description and the corresponding color distribution vectors.

Suppose that the images of Figure 2 are inserted in an M^2-tree node: The two chosen routing values could be, for example, $v_1^{[N]} = $ (nature, animals, feline, tiger) and $v_2^{[N]} = (1.76, 0.297, 0.699, 1.98, 0.371, 0.734, -0.243, 0.0786, -0.0293)$. The corresponding structure of leaf node N is depicted in Figure 3. The representative of node N in the parent node $P(N)$ is shown in Figure 4, along with a graphical representation of $Reg(N)$ in the multiple distance space induced by $v^{[N]}$. □

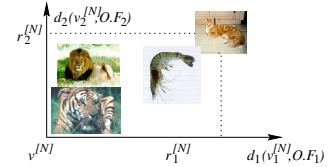## 2.4   Solving complex queries with M^2-tree

In this Section we show how complex queries can be resolved by the M^2-tree access structure. Suppose that we want to solve the query $\mathtt{range}_{f,r}(\mathcal{C})$: In order to see whether a node N of the M^2-tree has to be accessed,

	oid(e)	$O.F_1$	$O.F_2$	$d_1(O.F_1, v_1^{[N]})$	$d_2(O.F_2, v_2^{[N]})$
e_1	tiger.bmp	nature, animals, mammals, feline, tiger	$(1.73, 0.381, 0.483, 1.97, 0.407, 0.518, 0.957, 0.133, 0.0903)$	0.2	1.25
e_2	lion.jpg	nature, animals, mammals, feline, lion	$(1.32, 0.380, 0.729, 1.60, 0.480, 0.776, -2.27, -0.686, -1.34)$	0.2	2.60
e_3	tiger_cat.jpg	animals, domestic, feline, cat	$(0.754, 0.325, 0.73, 0.922, 0.409, 0.757, 1.58, 0.645, 1.41)$	0.667	2.80
e_4	tiger_shrimp.jpg	animals, crustacean, tiger, shrimp	$(3.23, 0.104, 0.853, 3.42, 0.189, 0.884, -1.24, 0.221, -0.282)$	0.5	2.33

Figure 3: The structure of an M^2-tree node N.

	ptr$T(e)$	$O.F_1$	$O.F_2$	$r_1^{[N]}$	$r_2^{[N]}$
...	...	...	...	...	...
e_i	ptr$(T(e_i))$	nature, animals, feline, tiger	$(1.76, 0.297, 0.699, 1.98, 0.371,$ $0.734, -0.243, 0.0786, -0.0293)$	0.667	2.80
...	...	...	...	...	...

(a)   (b)

Figure 4: The routing object for node N (a) and a graphical representation of $Reg(N)$ (b).

we compute a lower bound on the distance between any object reachable from N and the complex query formula f, that is, between f and the region associated to N, $\delta_{\min}(f, Reg(N))$, just as we would do for a simple query with an M-tree. To compute such bound we compute a lower bound for each predicate $p_j : F_{i_j} \sim q_j$ in f, and combine such bounds by way of the scoring function d_f. Therefore, $\delta_{\min}(f, Reg(N))$ is computed as $d_f(\delta_{\min}(p_1, Reg(N)), \dots, \delta_{\min}(p_n, Reg(N)))$. Since the scoring function d_f is monotonic increasing in all its arguments, no object reachable from N could lead to a value of δ lower than $\delta_{\min}(f, Reg(N))$.

Bounds on individual predicates can be easily computed by taking into account information about N stored in its parent, i.e. the routing values $v_i^{[N]}$ and the covering radii $r_i^{[N]}$.[3]

$$\delta_{\min}(p_j, Reg(N)) = \min\{\delta(p_j, v_{i_j}^{[N]}) - r_{i_j}^{[N]}, 0\} = \min\{d_{i_j}(q_j, v_{i_j}^{[N]}) - r_{i_j}^{[N]}, 0\}$$

Only nodes N for which $\delta_{\min}(f, Reg(N)) \leq r$ are accessed during the search. When the leaf level is reached, we can easily compute the overall distance $\delta_{\min}(f, O)$ for each object O in the leaf node using Equation 1. It should be noted that above consideration can be also used for complex k-NN queries, by substituting the threshold value r with the k-th lowest overall distance encountered so far.

3 Experimental Results

Existing approaches, like those proposed in [5, 2, 6], solve complex range queries by separately indexing each feature, e.g. with an M-tree, and by independently accessing the indices to solve a corresponding simple range query; finally, results of all queries are combined. In the case of Figure 5 (a), the M_1 index would retrieve objects O_2, O_3, O_5, and O_6, whereas index M_2 would retrieve objects O_1, O_2, and O_5; then, objects O_2 and O_5 are correctly returned as results. The M^2-tree approach, on the other hand, combines information from all the metric spaces, such that, in order to solve a complex range query, only those nodes whose region overlaps the query region are accessed during the search (see Figure 5 (b)) and no work is wasted to access objects that do not satisfy the query (like objects O_1, O_3, and O_6 in Figure 5 (a)).

To show the efficiency of the M^2-tree, we compared its performance for complex k-NN queries over a real data-set composed of over 15,000 images with respect to those of the $\mathcal{A}_0$ algorithm [5] and of a simple sequential scan. Features used to assess object similarity were (1) a string representing the image name (compared using the L_{edit} distance, i.e. the minimum number of characters to be inserted, deleted or substituted to transform a string into another) and a 32-bins histogram representing color information for the image (compared using the Euclidean distance). We built an M^2-tree over the 2 features and 2 M-trees over each single feature, and executed several k-NN conjunctive queries ($d_f = \max\{\delta_1, \delta_2\}$) by varying the value of k. Figures 6 (a) and (b) show the average computed distances and the average page reads for each query as a function of k. From the graphs, we see that the M^2-tree is indeed more efficient than the $\mathcal{A}_0$ algorithm, for both CPU and I/O costs. Moreover, we also see that, for high values of k, performance of the $\mathcal{A}_0$ algorithm rapidly decreases beyond that of the sequential scan, whereas M^2-tree costs are always the lowest.

---

[3]If the d_f function is monotonic decreasing in its j-th argument, we compute a upper bound $\delta_{\max}(p_j, Reg(N)) = d_{i_j}(q_j, v_{i_j}^{[N]}) + r_{i_j}^{[N]}$ and use it in place of $\delta_{\min}(p_j, Reg(N))$.
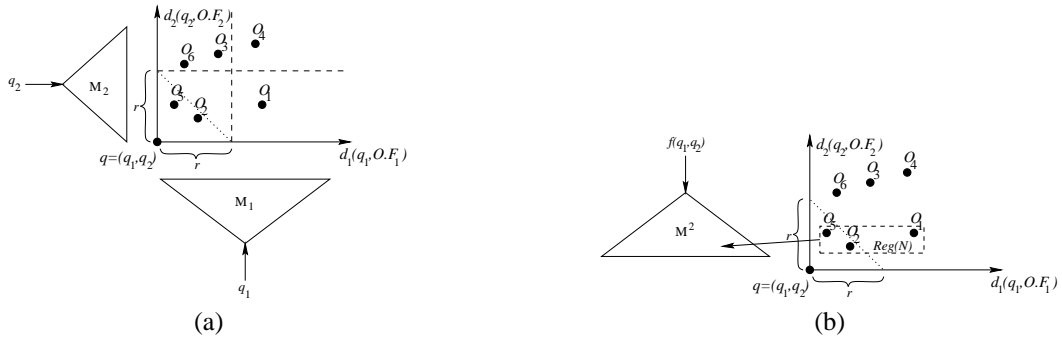
Figure 5: Solving a complex range query ($d_f = \delta_1 + \delta_2$) with 2 M-trees (a) and with an M^2-tree (b).
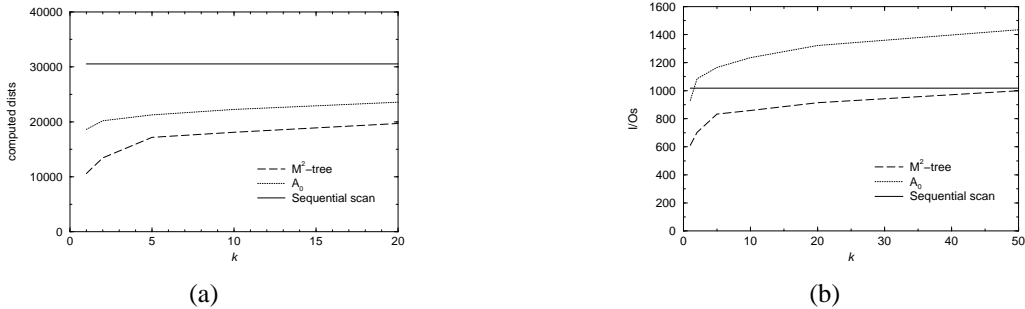

Figure 6: Average CPU (a) and I/O (b) costs for solving a complex k-NN conjunctive query.

4 Conclusions

In this paper we have presented basic principles of M^2-tree, a balanced search tree designed for indexing multiple metric spaces. We have shown how the M^2-tree can be used to perform complex similarity search over multimedia objects represented by multiple features. Structure of M^2-tree nodes and the sketch of searching algorithms have been illustrated. Finally, we have demonstrated the efficiency of the proposed structure over other state-of-the-art approaches through some preliminary experiments.

Current and future work includes the complete specification of index maintenance algorithms (choosing the node in which a new object should be inserted, splitting a node, choosing the routing values, etc.), as well as a wider query model, to include other queries that can be efficiently solved by the M^2-tree (e.g. intersection and composition of complex queries). Thorough experimentation with different real data-sets is also planned.

References

[1] S. Berchtold, D. A. Keim, and H.-P. Kriegel. The X-tree: An index structure for high-dimensional data. In *VLDB'96*, pp. 28–39, Mumbai (Bombay), India, Sept. 1996.

[2] S. Chaudhuri and L. Gravano. Optimizing queries over multimedia repositories. In *SIGMOD 1996*, pp. 91–102, Montreal, Canada, June 1996.

[3] P. Ciaccia, M. Patella, and P. Zezula. M-tree: An efficient access method for similarity search in metric spaces. In *VLDB'97*, pp. 426–435, Athens, Greece, Aug. 1997.

[4] P. Ciaccia, M. Patella, and P. Zezula. Processing complex similarity queries with distance-based access methods. In *EDBT'98*, pp. 9–23, Valencia, Spain, Mar. 1998.

[5] R. Fagin. Combining fuzzy information from multiple systems. In *PODS'96*, pp. 216–226, Montreal, Canada, June 1996.

[6] U. Güntzer, W.-T. Balke, and W. Kießling. Optimizing multi-feature queries for image databases. In *VLDB 2000*, pp. 419–428, Cairo, Egypt, Sept. 2000.

[7] N. Katayama and S. Satoh. The SR-tree: An index structure for high-dimensional nearest neighbor queries. In *SIGMOD 1997*, pp. 369–380, New York, NY, May 1997.

[8] M. Stricker and M. Orengo. Similarity of color images. In *Storage and Retrieval for Image and Video Databases SPIE*, volume 2420, pp. 381–392, San Jose, CA, Feb. 1995.

[9] C. Traina Jr., A. J. M. Traina, B. Seeger, and C. Faloutsos. Slim-trees: High performance metric trees minimizing overlap between nodes. In *EDBT 2000*, pp. 51–65, Konstanz, Germany, Mar. 2000.

[10] R. Weber, H.-J. Schek, and S. Blott. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In *VLDB'98*, pp. 194–205, New York, NY, Aug. 1998.