Processing Complex Similarity Queries with Distance-based Access Methods*

Paolo Ciaccia[1], Marco Patella[1], and Pavel Zezula[2]

[1] DEIS - CSITE-CNR, University of Bologna - Italy, {pciaccia,mpatella}@deis.unibo.it
[2] IEI-CNR Pisa - Italy, zezula@iei.pi.cnr.it

Abstract. Efficient evaluation of similarity queries is one of the basic requirements for advanced multimedia applications. In this paper, we consider the relevant case where *complex* similarity queries are defined through a generic language $\mathcal{L}$ and whose predicates refer to a single feature F. Contrary to the language level which deals only with similarity scores, the proposed evaluation process is based on distances between feature values - known spatial or metric indexes use distances to evaluate predicates. The proposed solution suggests that the index should process complex queries *as a whole*, thus evaluating multiple similarity predicates at a time. The flexibility of our approach is demonstrated by considering three different similarity languages, and showing how the M-tree access method has been extended to this purpose. Experimental results clearly show that performance of the extended M-tree is consistently better than that of state-of-the-art search algorithms.

1 Introduction

Similarity queries are a primary concern in multimedia database systems, where users are interested in retrieving objects which best match query conditions. Efficient resolution of similarity queries is usually based on a process which includes the extraction of relevant features from the objects (e.g., color histograms from still images), and the indexing of such feature values, typically through either *spatial* access methods, such as the R-tree [Gut84], or *metric* trees, such as the M-tree [CPZ97]. In this context, low *distance* between feature values implies high similarity, and vice versa. In general, index support is experimentally shown to be valuable under many circumstances, even if the cost of evaluating a similarity query can sometimes be still very high – comparable to that of a sequential scan.

Efficient processing of *complex* similarity queries - queries with more than one similarity predicate - has some peculiarities with respect to traditional (Boolean) query processing which have been highlighted by recent works [CG96, Fag96, FW97]. The basic lesson is that, since the "similarity score" (*grade*) an object gets for the whole query depends on how the scores it gets for the single predicates are combined, predicates cannot be independently evaluated.

Example 1. Consider an image database where objects can be retrieved by means of predicates on `shape` and `color` features, and assume that the two sets of feature values are separately indexed. In order to retrieve the best match for the query `(shape = 'circular') and (color = 'red')` it is *not* correct to retrieve *only* the best match

* This work has been partially supported by the ESPRIT LTR project no. 9141, HERMES, and by Italian CNR project MIDA. The work of Pavel Zezula has also been supported by Grants GACR No. 102/96/0986 and KONTAKT No. PM96 S028.

for `color` (using an index on `color`) and the best match for `shape` (using an index on `shape`), since the best match for the *overall* query needs not to be the best match for the single conjuncts. □

The state-of-the-art solution to the above kind of queries is Fagin's $\mathcal{A}_0$ algorithm [Fag96], which returns the k best matches (nearest neighbors) for a complex query, on the assumption that evaluation of the single predicates in the query is carried out by independent *subsystems*, and that access to one subsystem is "synchronized" with that to others (a special case of the $\mathcal{A}_0$ algorithm is described in Section 6).

In this paper we concentrate on a relevant class of complex similarity queries, which arises when all the similarity predicates refer to a single feature.

Example 2. Consider an image database whose objects can be retrieved using a *Query-by-Sketch* modality. When a user draws a shape on the screen, the system searches the DB, and returns those, say, 10 images which contain a shape best matching (according to given similarity criterion for shapes) the user's input. The user can then "refine" the search by selecting those objects which are similar to what he/she had in mind and which are actually not. Suppose two "positive" and one "negative" samples are specified. Now, the system has to search the DB for those 10 objects which are most similar to *both* positive samples, and, at the same time, *not* similar to the negative one. This interactive process can be iterated several times, until the user gets satisfied with system's output. □

An interactive retrieval process, such as the one sketched above, typically occurs when querying multimedia repositories [Jai96], where the user has no clear idea on how to express what he/she is looking for, and relies on previous results to improve the effectiveness of subsequent requests.

Complex single-feature queries could be casted in the more general framework of multi-feature queries, however their nature suggests that a more efficient evaluation strategy could indeed be possible to devise. The one we propose starts from the idea to extend access methods in such a way that they can process complex queries *as a whole*, thus evaluating *multiple* predicates at a time. Since access methods typically evaluate the similarity of two objects by means of their distance in some feature space, we ground our proposed extension on a sound formal basis, which specifies how multiple distance measures (corresponding to multiple similarity predicates) should be combined to perform a both efficient and correct pruning of the search space. The three basic ingredients of the framework we introduce are: (a) a distance function, d, which compares feature values, (b) a functional mapping, h, from distances to similarity scores, and (c) a similarity language, $\mathcal{L}$, which, given scores for the single predicates, tells us how to combine them to yield a global score. Our solution is parametric on d, h, and $\mathcal{L}$, thus it provides a solution to the following general problem:
Given a similarity language $\mathcal{L}$, a distance function d, and a function h which converts distance values into similarity scores, **determine** how to efficiently evaluate a query Q expressed in the language $\mathcal{L}$, when Q consists of predicates on a single feature F, and values of F are indexed by a distance-based access method.

We show the feasibility of our approach by considering three different languages and by providing an extension of the M-tree access method [CPZ97]. Since the only assumption we need about the nature of the access method is that it uses distances to prune the search space, our results apply to *any* spatial (multi-dimensional) or metric access method. Experimental results show that our solution consistently outperforms state-of-the-art algorithms.

The rest of the paper is organized as follows. In Section 2 we provide the necessary definitions, review how access methods are used to solve simple similarity queries, and introduce three specific query languages for illustrative purpose. In Section 3 we establish the basis for evaluating similarity queries through distance measures, and Section 4 presents the basic result which allows a distance-based access method to process complex queries as a whole. In Section 5 we describe how the M-tree access method has been extended to this purpose. Section 6 shows some experimental results, and Section 7 considers related works and concludes.

2 Preliminaries

Consider a collection (class, relation, etc.) $\mathcal{C}$ of objects which are indexed on a feature F, whose values are drawn from a domain $\mathcal{D}$, $\mathcal{D} = dom(F)$. For simplicity, we assume that values of F univocally identify objects, thus no two objects have the same feature value.

A generic *similarity predicate* p on F has the form $F \sim v$, where $v \in \mathcal{D}$ is a constant (*query value*) and $\sim$ is a similarity operator. Evaluating p on an object O returns a score (grade), $s(p, O.F) \in [0, 1]$, which says how similar is object O to the query value v. The evaluation of predicate p on all the objects in $\mathcal{C}$ then yields a "graded set" $\{(O, s(p, O.F)) | O \in \mathcal{C}\}$. For instance, evaluating the predicate `color` $\sim$ `'red'` means to assign to each image in the collection a score assessing its "redness".

We consider two basic forms of similarity queries: *range* and *nearest neighbors* (best matches) queries. Given a predicate $p : F \sim v$, a *simple* range query returns all the objects whose similarity with respect to v is at least α (e.g. images which are "red enough"), whereas a *simple* nearest neighbors query would return the k ($k \geq 1$ being user-specified) objects having the highest similarity scores with respect to v (e.g. the 10 "most red" images), with ties arbitrarily broken.

Since our objective is to deal with *complex* (single-feature) similarity queries, we need a language $\mathcal{L}$ which allows multiple predicates to be combined into a *similarity formula*, f. The nature of the specific language is uninfluential to our arguments. We only require that, if $f = f(p_1, \ldots, p_n)$ is a formula of $\mathcal{L}$, then the similarity of an object O with respect to f, denoted $s(f, O.F)$, is computed through a corresponding *scoring function* [Fag96], s_f, which takes as input the scores of O with respect to the predicates of formula f, that is:

$$s(f(p_1, \ldots, p_n), O.F) = s_f(s(p_1, O.F), \ldots, s(p_n, O.F)) \tag{1}$$

Shortly, we will introduce three specific sample languages to construct similarity formulas. For the moment, we provide the following definitions which exactly specify the kinds of queries we are going to deal with:

Complex range query Given a similarity formula $f \in \mathcal{L}$ and a minimum similarity threshold α, the query $\texttt{range}(f, \alpha, \mathcal{C})$ selects all the objects in $\mathcal{C}$ (with their scores) such that $s(f, O.F) \geq \alpha$.

Complex nearest neighbors (k-NN) query Given a similarity formula f and an integer $k \geq 1$, the k-NN query $\texttt{NN}(f, k, \mathcal{C})$ selects the k objects in $\mathcal{C}$ having the highest similarity scores with respect to f. In case of ties, they are arbitrarily broken.

2.1 Distance-based Access Methods

Evaluating the similarity of an object with respect to a query value can be done in several ways, depending on the specific feature. A common approach, which is the basis for efficiently processing similarity queries through indexing, is to have an *indirect* evaluation of similarity scores. In this case, what is actually measured is the *distance* between feature values, being understood that high scores correspond to low distances and low scores to high distances.

In general, distance evaluation is carried out by using a *distance function*, d, which, for any pair of feature values, yields a non-negative real value, $d : \mathcal{D}^2 \to \Re_0^+$. Although arbitrary distance functions can in principle be used, it is both reasonable and useful to limit the analysis to "well-behaved" cases. In particular, we assume that d is a *metric*, that is, for each triple of values $v_x, v_y, v_z \in \mathcal{D}$ the following axioms hold:

1. $d(v_x, v_y) = d(v_y, v_x)$ (*symmetry*)
2. $d(v_x, v_y) > 0$ ($v_x \neq v_y$) and $d(v_x, v_x) = 0$ (*non negativity*)

3. $d(v_x, v_y) \leq d(v_x, v_z) + d(v_z, v_y)$ (*triangle inequality*)

Relevant examples of metrics include, among others, the Minkowski (L_p) metrics over n-dimensional points, which are defined as ($p \geq 1$) $L_p(v_x, v_y) = (\sum_{j=1}^{n} \mid v_x[j] - v_y[j] \mid^p)^{1/p}$, and include the Euclidean (L_2) and the Manhattan, or "city-block", (L_1) metrics, and the Levenshtein (*edit*) distance over strings, which counts the minimal number of changes (insertions, deletions, substitutions) needed to transform a string into another one.

Most tree-like access methods able to index complex features share a substantial similar structure, which can be summarized as follows. Each *node* N (usually mapped to a disk page) in the tree corresponds to a *data region*, $Reg(N) \subseteq \mathcal{D}$. Node N stores a set of entries, each entry pointing to a child node N_c and including the specification of $Reg(N_c)$. All indexed keys (feature values, i.e. points of $\mathcal{D}$) are stored in the leaf nodes, and those keys in the sub-tree rooted at N are guaranteed to stay in $Reg(N)$.

Example 3. The R-tree [Gut84] organizes n-dimensional point objects by enclosing them into Minimum Bounding Rectangles (MBR). This principle is applied at all levels of the R-tree, so that the region of each node in the R-tree is a (hyper-)rectangle, defined as the MBR of its child regions. □

Example 4. The M-tree [CPZ97] can index objects over generic metric spaces (see also Section 5). Given a distance function d, the region of a node N is implicitly defined by the predicate $d(v_r, v) \leq r(v_r)$, where v_r is a so-called *routing object* (or routing key value), and $r(v_r)$ is the *covering radius* of v_r. The intuition is that all objects v reachable from node N are within distance $r(v_r)$ from v_r. Note that the actual "shape" of M-tree regions depends on the specific metric space $(\mathcal{D}, d)$. □

The usual strategy adopted by access methods to process *simple* range similarity queries consists of two basic steps:

1. Take as input a (single) query value, v_q, and a *maximum distance threshold*, $r(v_q)$. This defines a *query region*, centered at v_q. In general, $r(v_q)$ is inversely related to the minimum similarity one wants to accept in the result.
2. Search the index, and access all and only the nodes N such that $Reg(N)$ and the query region intersect. This is practically done by computing $d_{min}(v_q, Reg(N))$, that is, the minimum distance an object in $Reg(N)$ can have from the query value v_q. If $d_{min}(v_q, Reg(N)) \leq r(v_q)$ then N has to be accessed, otherwise it can be safely pruned from the search space.

In the case of k-NN queries, the basic difference with respect to above strategy is that the distance threshold $r(v_q)$ is a *dynamic* one, since it is given by the distance of the k-th *current* nearest neighbor from the query value v_q. Because of $r(v_q)$ dynamicity, algorithms for processing nearest neighbors queries also implement a policy to decide the order to visit nodes which have not been pruned yet. Details can be found in [RKV95] (for R-tree) and [CPZ97] (for M-tree).

If we try to naively generalize the above approach to complex queries, some difficulties arise, which are summarized by the following questions:

1. Can a "distance threshold" be defined for arbitrarily complex queries?
2. Can we always decide if a query region and a data region intersect?

It turns out that both questions have a *negative* answer, as we will show in Section 4, where we also describe how our approach can nonetheless overcome these limitations.

2.2 Similarity Languages

A similarity language $\mathcal{L}$ comes with a syntax, specifying which are valid (well-formed) formulas, and a semantics, telling us how to evaluate the similarity of an object with respect to a complex query. Although the results we present are language-independent, it also helps intuition to consider specific examples.

The first two languages we consider, $\mathcal{FS}$ (fuzzy standard) and $\mathcal{FA}$ (fuzzy algebraic), share the same syntax and stay in the framework of *fuzzy logic* [Zad65, KY95]. Their formulas are defined by the following grammar rule:

$$f ::= p|f \wedge f|f \vee f|\neg f|(f)$$

where p is a similarity predicate. The semantics of a formula f is given by the following set of recursive rules:

	$\mathcal{FS}$	$\mathcal{FA}$
$s(f_1 \wedge f_2, v)$	$\min\{s(f_1, v), s(f_2, v)\}$	$s(f_1, v) \cdot s(f_2, v)$
$s(f_1 \vee f_2, v)$	$\max\{s(f_1, v), s(f_2, v)\}$	$s(f_1, v) + s(f_2, v) - s(f_1, v) \cdot s(f_2, v)$
$s(\neg f, v)$	$1 - s(f, v)$	$1 - s(f, v)$

The last language we consider, $\mathcal{WS}$ (weighted sum), does not use logical connectives at all, but allows *weights* to be attached to the predicates, in order to reflect the importance the user wants to assign to each of them. A formula f has the form:

$$f ::= \{(p_1, \theta_1), (p_2, \theta_2), \ldots, (p_n, \theta_n)\}$$

where each p_i is a similarity predicate, the θ_i's are positive weights, and $\sum_i \theta_i = 1$. The semantics of a $\mathcal{WS}$ formula f is simply $s(f, v) = \sum_{i=1}^{n} \theta_i \cdot s(p_i, v)$

Although the subject of deciding on which is the "best" language is not in the scope of the paper, it is important to realize that any specific language has some advantages and drawbacks, thus making the choice a difficult problem. We also remark that above languages are only a selected sample of the many one can conceive to formulate complex queries, and that our approach is not limited only to them. In particular, our results also apply to fuzzy languages, such as $\mathcal{FS}$ and $\mathcal{FA}$, when they are extended with weights, as shown in [FW97].

Example 5. Assume that we want to retrieve objects which are similar to both query values v_1 and v_2. With the $\mathcal{FS}$ and $\mathcal{FA}$ languages we can use the formula $f_1 = p_1 \wedge p_2$, where $p_i : F \sim v_i$. With $\mathcal{WS}$, assuming that both predicates have the same relevance to us, the formula $f_2 = \{(p_1, 0.5), (p_2, 0.5)\}$ is appropriate. Given objects' scores for the two predicates p_1 and p_2, Table 1 shows the final scores, together with the relative rank of each object. It is evident that objects' ranking highly depends on the specific language (see object O_1) – this can affect the result of nearest neighbors queries – and that the score of an object can be very different under different languages – this can influence the choice of an appropriate threshold for range queries. □

			$\mathcal{FS}$		$\mathcal{FA}$		$\mathcal{WS}$	
Object	$s(p_1, O_j.F)$	$s(p_2, O_j.F)$	$s(f_1, O_j.F)$	rank	$s(f_1, O_j.F)$	rank	$s(f_2, O_j.F)$	rank
O_1	0.9	0.4	0.4	4	0.36	3	0.65	1
O_2	0.6	0.65	0.6	1	0.39	2	0.625	3
O_3	0.7	0.5	0.5	3	0.35	4	0.6	4
O_4	0.72	0.55	0.55	2	0.396	1	0.635	2

Table 1. Similarity scores for complex queries.

Above example also clearly shows that determining the best match (the object with rank 1 in Table 1) for a complex query cannot be trivially solved by considering only the best matches for the single predicates. For instance, the best match for formula f_1 under $\mathcal{FA}$ semantics is object O_4, which is neither the best match for p_1 nor for p_2.

Similar considerations can be done for complex range queries too. Refer again to Table 1, and consider the query $\texttt{range}(f_2, 0.63, \mathcal{C})$, to be evaluated under $\mathcal{WS}$ semantics. Given the threshold value 0.63, which leads to select objects O_1 and O_4, which are (if any) appropriate thresholds for the single predicates such that the correct answer could still be derived?

3 Evaluating Similarity Through Distance

The first step towards an efficient evaluation of complex queries concerns how to compute similarity scores when only distances between feature values can be measured, which is the usual case. For this we need the definition of a *correspondence function*.

Definition 1 : Correspondence function.
We say that $h : \Re_0^+ \to [0,1]$ is a (distance to similarity) correspondence function iff it has the two following properties:

$$h(0) = 1 \tag{2}$$

$$x_1 \leq x_2 \Rightarrow h(x_1) \geq h(x_2) \qquad \forall x_1, x_2 \in \Re_0^+ \tag{3}$$

In other terms, a correspondence function assigns maximum similarity in case of 0 distance (exact-match), and makes similarity inversely related to distance. Usually, correspondence functions are not considered "first-class" elements in the context of similarity processing. The reason is that either only simple queries are considered or no reference is done to the actual work performed by indexes [CG96]. In the enlarged context we consider, where both complex queries and access methods are present, correspondence functions have a primary role. Indeed, they represent the "missing link" of the chain leading from feature values to similarity scores. The essence of this chain is captured by the following definition.

Definition 2 : Distance-based similarity environment.
A distance-based similarity environment is a quadruple $\mathcal{DS} = (\mathcal{D}, d, h, \mathcal{L})$, where $\mathcal{D}$ is a domain of feature values, d is a metric distance over $\mathcal{D}$, h is a correspondence function, and $\mathcal{L}$ is a similarity language. □

Given any similarity environment $\mathcal{DS}$, we are now ready to perform sequential evaluation of arbitrarily complex similarity queries. The algorithm for range queries is described below; the one for nearest neighbors queries is based on the same principles, but it is not shown here for brevity.

Algorithm `Range-Seq` (sequential processing of range queries)

Input: similarity environment $\mathcal{DS} = (dom(F), d, h, \mathcal{L})$, collection $\mathcal{C}$, formula $f = f(p_1, \ldots, p_n) \in \mathcal{L}$, minimum threshold α;
Output: $\{(O, s(f, O.F)) | O \in \mathcal{C}, s(f, O.F) \geq \alpha\}$;
1. For each $O \in \mathcal{C}$ do:
2. For each predicate $p_i : F \sim v_i$ compute $d_i = d(v_i, O.F)$;
3. Let $s_i = h(d_i)$, $i = 1, \ldots, n$;
4. If $s(f, O.F) \triangleq s_f(s_1, \ldots, s_n) \geq \alpha$ then add $(O, s(f, O.F))$ to the result;

Example 6. Consider the environment $\mathcal{DS} = (\Re^2, L_1, 1 - 0.1 \cdot x, \mathcal{FS})$, and the query Q : `range`$(p_1 \wedge p_2, 0.8, \mathcal{C})$, with $p_i : F \sim v_i (i = 1, 2)$. Refer to Figure 1, where $v_1 = (3, 2)$ and $v_2 = (5, 3)$, and consider the point (feature vector) $v = (3.5, 1)$. To evaluate its score, we first compute $d_1 = L_1(v_1, v) = |3 - 3.5| + |2 - 1| = 1.5$ and $d_2 = L_2(v_2, v) = |5 - 3.5| + |3 - 1| = 3.5$ (line 2 of the `Range-Seq` algorithm). In step 3, we apply the correspondence function $h(x) = 1 - 0.1 \cdot x$ to the two distance values, thus obtaining $s_1 = h(d_1) = 0.85$ and $s_2 = h(d_2) = 0.65$. Finally, since we are using the $\mathcal{FS}$ semantics, we have to take the minimum of the two scores to compute the overall score, i.e. $s(p_1 \wedge p_2, v) = \min\{s_1, s_2\} = 0.65$. Since this is less than the threshold value 0.8, point v does not satisfy the range query Q. □

It is important to observe that the choice of a specific correspondence function can affect the result of similarity queries. This is easily shown by referring to the above example, and redefining h as $h(x) = 1 - 0.05 \cdot x$. After simple calculations, it is derived that $s(p_1 \wedge p_2, v) = 0.825$, thus point v will be part of the result.

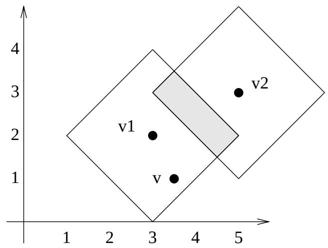

Fig. 1. The region of the query $\mathtt{range}((F \sim v_1) \wedge (F \sim v_2), 0.8, \mathcal{C})$ is shaded

4 Extending Distance-based Access Methods

The `Range-Seq` algorithm is correct but clearly inefficient for large objects' collections. Exploitation of an index built over a feature F is possible, in principle, with two different modalities. The "traditional" one independently evaluates the predicates, and then combines the partial results *outside* of the index itself. This approach, besides being inefficient because it leads to access parts of the index more than once, cannot be applied at all for generic complex queries. The case of nearest neighbors queries has been analyzed in [Fag96]. Here, the problem is that the best match for a complex query cannot be determined by looking only at the best matches of the single predicates (see also Example 5). In the case of complex range queries, independent evaluation is possible only under the strict assumption that a distance constraint for each single predicate in the query can be derived from the overall minimum similarity threshold, which is not always the case.

Example 7. Consider Example 6. In order to process the query $\mathtt{range}(p_1 \wedge p_2, 0.8, \mathcal{C})$, by independently evaluating predicates p_1 and p_2, we can proceed as follows. Since we are using the $\mathcal{FS}$ semantics and the correspondence function h, it has to be $\min\{1 - 0.1 \cdot d(v_1, v), 1 - 0.1 \cdot d(v_2, v)\} \geq 0.8$. This can also be expressed as $\max\{d(v_1, v), d(v_2, v)\} \leq 2$. It can be easily verified that if v satisfies above inequality, then v lies in the shaded query region of Figure 1. Since above constraint can be also written as $(d(v_1, v) \leq 2) \wedge (d(v_2, v) \leq 2)$, the complex query can be evaluated by independently performing two simple range queries, taking the intersection of objects' results, and then computing the final scores. □

Example 8. Assume now that the correspondence function has the form $h(x) = \exp(-x)$, and consider the query $\mathtt{range}(\{(p_1, 0.4), (p_2, 0.6)\}, 0.5, \mathcal{C})$ in the $\mathcal{WS}$ language. The similarity constraint is:

$$0.4 \cdot e^{-d(v_1,v)} + 0.6 \cdot e^{-d(v_2,v)} \geq 0.5$$

which cannot be decomposed into two *bounded* simple range queries. Indeed, if v is in the result, then v necessarily satisfies the two constraints (each using a single query value) $0.4 \cdot e^{-d(v_1,v)} + 0.6 \geq 0.5$ and $0.4 + 0.6 \cdot e^{-d(v_2,v)} \geq 0.5$, which are obtained by setting, respectively, $d(v_2, v) = 0$ and $d(v_1, v) = 0$. However, since the first constraint is satisfied by *any* $d(v_1, v)$ value, the corresponding simple range query is $d(v_1, v) \leq \infty$, which amounts to access the whole data collection. □

The second possibility of evaluating complex queries is the one we propose, and suggests that the index should process complex queries *as a whole*. We first show how queries in the two above examples would be managed by the new approach, then we generalize to generic similarity environments.

Example 9. Consider Example 7, and assume, without loss of generality, that feature values are indexed by an M-tree. We can prune a node N with routing object v_r and covering radius $r(v_r)$ if its region, $Reg(N)$, only contains points v such that

$$\min\{1 - 0.1 \cdot d(v_1, v), 1 - 0.1 \cdot d(v_2, v)\} < 0.8 \qquad (4)$$

Because of the triangular inequality and non-negativity properties of d, the following lower bound on $d(v_i, v)$ can be derived:

$$d(v_i, v) \geq d_{min}(v_i, Reg(N)) \stackrel{\text{def}}{=} \max\{d(v_i, v_r) - r(v_r), 0\} \qquad i = 1, 2$$

If we substitute such lower bounds into (4), we obtain

$$\min\{1 - 0.1 \cdot d_{min}(v_1, Reg(N)), 1 - 0.1 \cdot d_{min}(v_2, Reg(N))\} < 0.8$$

From this, we can immediately decide whether node N has to be accessed or not. $\square$

Example 10. Consider now Example 8. We can adopt the same approach as in Example 9. Node N can be pruned if each point v in its region satisfies $0.4 \cdot e^{-d(v_1,v)} + 0.6 \cdot e^{-d(v_2,v)} < 0.5$. By using the $d_{min}(v_i, Reg(N))$ lower bounds, it is obtained $0.4 \cdot e^{-d_{min}(v_1,Reg(N))} + 0.6 \cdot e^{-d_{min}(v_2,Reg(N))} < 0.5$. Once again, checking above constraint is all that is needed to decide if node N has to be accessed. $\square$

In order to generalize our approach to generic similarity environments and arbitrarily complex queries, we need a preliminary definition.

Definition 3 : Monotonicity.
We say that a scoring function $s_f(s(p_1, v), \ldots, s(p_n, v))$ is monotonic increasing (respectively decreasing) in the variable $s(p_i, v)$ if given any two n-tuples of scores' values $(s_1, \ldots, s_i, \ldots, s_n)$ and $(s_1, \ldots, s'_i, \ldots, s_n)$ with $s_i \leq s'_i$, it is $s_f(s_1, \ldots, s_i, \ldots, s_n) \leq s_f(s_1, \ldots, s'_i,$
$\ldots, s_n)$ (resp. $s_f(s_1, \ldots, s_i, \ldots, s_n) \geq s_f(s_1, \ldots, s'_i, \ldots, s_n)$). If a scoring function s_f is monotonic increasing (resp. decreasing) in all its variables, we simply say that s_f is monotonic increasing (resp. decreasing). $\square$

Monotonicity is a property which allows us to somewhat predict the behavior of a scoring function in a certain data region, which is a basic requirement for deciding whether or not the corresponding node in the index should be accessed. Note that both queries in Examples 9 and 10 are monotonic increasing.

Before presenting our major result, it has to be observed that *a scoring function can be "monotonic in all its arguments" without being neither monotonic increasing nor monotonic decreasing* (on the other hand, the converse is true). For instance, $s(p_1, v) \cdot (1 - s(p_2, v))$, which is the scoring function of $p_1 \wedge \neg p_2$ in the $\mathcal{FA}$ language, is monotonic increasing in $s(p_1, v)$ and monotonic decreasing in $s(p_2, v)$.

In case a certain predicate occurs more than once in a formula, we need to distinguish its occurrences. For instance, the formula $f : p_1 \wedge \neg p_1$ with $p_1 : F \sim v_1$, has to be rewritten as, say, $p_1 \wedge \neg p_2$, with $p_2 : F \sim v_2$, and $v_1 \equiv v_2$. Under the $\mathcal{FA}$ semantics, say, the scoring function of f, that is $s(p_1, v) \cdot (1 - s(p_2, v))$, is now monotonic increasing in $s(p_1, v)$ and monotonic decreasing in $s(p_2, v)$. By distinguishing single occurrences of predicates, it can be seen that the $\mathcal{WS}$ language can only generate formulas having monotonic increasing scoring functions, whereas all scoring functions of formulas of languages $\mathcal{FS}$ and $\mathcal{FA}$ are guaranteed to be monotonic in all their arguments.

We are now ready to state our major result.

Theorem 4.
Let $\mathcal{DS} = (dom(F), d, h, \mathcal{L})$ be a similarity environment, $f = f(p_1, \ldots, p_n) \in \mathcal{L}$ ($p_i : F \sim v_i, i = 1 \ldots, n$) a similarity formula such that each predicate occurs exactly once, and $\mathcal{C}$ a collection of objects indexed by a distance-based tree $\mathcal{T}$ on the values of feature F. Let $s_f(s(p_1, v), \ldots, s(p_n, v))$ ($v \in \mathcal{D}$) be the scoring function of f. If s_f is monotonic in all its variables, then a node N of $\mathcal{T}$ can be pruned if

$$s_{max}(f, Reg(N)) \stackrel{\text{def}}{=} s_f(h(d_B(v_1, Reg(N))), \ldots, h(d_B(v_n, Reg(N)))) < \alpha \qquad (5)$$

where

$$d_B(v_i, Reg(N)) = \begin{cases} d_{min}(v_i, Reg(N)) \text{ if } s_f \text{ is monotonic increasing in } s(p_i, v) \\ d_{max}(v_i, Reg(N)) \text{ if } s_f \text{ is monotonic decreasing in } s(p_i, v) \end{cases} \quad (6)$$

with $d_{min}(v_i, Reg(N))$ ($d_{max}(v_i, Reg(N))$) *being a lower (upper) bound on the minimum (maximum) distance from* v_i *of any value* $v \in Reg(N)$*, and where* α *is*

– *the user-supplied minimum similarity threshold, if the query is* `range`$(f, \alpha, \mathcal{C})$*;*
– *the k-th highest similarity score encountered so far, if the query is* `NN`$(f, k, \mathcal{C})$*. If less than k objects have been evaluated, then* $\alpha = 0$*.* □

Proof: Assume that $Reg(N)$ contains a point v^* such that $s_f(h(d(v_1, v^*)), \ldots, h(d(v_n, v^*))) \geq \alpha$. By construction, it is $d(v_i, v^*) \geq d_B(v_i, Reg(N))$, if s_f is monotonic increasing in $s(p_i, v)$, and $d(v_i, v^*) \leq d_B(v_i, Reg(N))$, if s_f is monotonic decreasing in $s(p_i, v)$. Since h is a monotonic decreasing function, it is also

$$h(d(v_i, v^*)) \leq (\geq) h(d_B(v_i, Reg(N))) \quad \text{if} \quad d(v_i, v^*) \geq (\leq) d_B(v_i, Reg(N))$$

Since s_f is monotonic in all its arguments, it is impossible to have

$$s_f(h(d(v_1, v^*)), \ldots, h(d(v_n, v^*))) > s_f(h(d_B(v_1, Reg(N))), \ldots, h(d_B(v_n, Reg(N))))$$

which proves the result. □

Theorem 4 generalizes to complex queries the basic technique used to process simple range and nearest neighbors queries. It does so by considering how a (single occurrence of a) predicate can affect the overall score, and by using appropriate bounds on the distances from the query values. These are then used to derive an upper bound, $s_{max}(f, Reg(N))$, on the maximum similarity score an object in the region of node N can get with respect to f, that is $s_{max}(f, Reg(N)) \geq s(f, v)$, $\forall v \in Reg(N)$.

Example 11. A query which has been proved in [Fag96] to be a "difficult one" is `NN`$(p_1 \wedge \neg p_1, 1, \mathcal{C})$, with $p_1 : F \sim v_1$. This can be processed as follows. First, rewrite the formula as $p_1 \wedge \neg p_2$ ($p_2 : F \sim v_2, v_2 \equiv v_1$). Without loss of generality, assume the standard fuzzy semantics $\mathcal{FS}$, and the correspondence function $h(x) = 1 - 0.1 \cdot x$. The scoring function can therefore be written as $\min\{s(p_1, v), 1 - s(p_2, v)\}$. By substituting bounds on distances and applying the correspondence function we finally get:

$$s_{max}(p_1 \wedge \neg p_1, Reg(N)) = \min\{1 - 0.1 \cdot d_{min}(v_1, Reg(N)), 0.1 \cdot d_{max}(v_1, Reg(N))\}$$

where we have turned back to the original v_1 notation. □

Theorem 4 provides a general way to handle complex queries in generic similarity environments, using *any* distance-based index. The only part specific to the index at hand is the computation of the $d_{min}(v_i, Reg(N))$ and $d_{max}(v_i, Reg(N))$ bounds, since they depend on the kind of data regions managed by the index. For instance, in M-tree above bounds are computed as $\max\{d(v_i, v_r) - r(v_r), 0\}$ and $d(v_i, v_r) + r(v_r)$, respectively [CPZ97]. Simple calculations are similarly required for other metric trees [Chi94, Bri95, BO97], as well as for spatial access methods, such as R-tree (see [RKV95]).

4.1 False Drops at the Index Level

The absence of any specific assumption about the similarity environment and the access method in Theorem 4 makes it impossible to guarantee the absence of "false drops" at the level of index nodes. More precisely, if inequality (5) is satisfied, it is guaranteed that node N cannot lead to qualifying objects, and can therefore be safely pruned. This is also to say that $Reg(N)$ and the query region do not intersect. On the other hand, if (5) is *not* satisfied (i.e. $s_{max}(v, Reg(N)) \geq \alpha$) it can still be the case that $Reg(N)$ and the query region do not intersect.

Example 12. Consider the environment $\mathcal{DS} = (\Re^2, L_2, \exp(-x), \mathcal{FS})$, and the query $\mathtt{range}(p_1 \wedge p_2, 0.5, \mathcal{C})$, with $p_i = F \sim v_i$, $v_1 = (1,2)$ and $v_2 = (2,2)$. Consider the M-tree data region: $Reg(N) = \{v | d(v_r = (1.5,1), v) \leq r(v_r) = 0.43\}$. As Figure 2 shows, $Reg(N)$ does not intersect the query region, represented by the intersection of the two circles of radius $\ln(1/0.5)$ centered in v_1 and v_2. However, the maximum possible similarity for $Reg(N)$ is estimated as $\min\{e^{-(d(v_1,v_r)-r(v_r))}, e^{-(d(v_2,v_r)-r(v_r))}\}$ $\approx 0.502 > 0.5$. Therefore, node N cannot be pruned. □

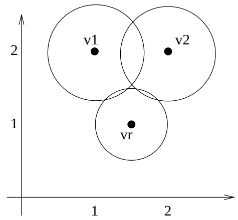

Fig. 2. The false drops phenomenon.

Although the phenomenon of false drops can lead to explore unrelevant parts of the tree, thus affecting the efficiency of the retrieval, it does not alter at all the correctness of the results, since, at the leaf level, we evaluate the actual similarities of the objects, for which no bounds are involved and actual distances are measured.

Resolving the false drop problem for generic similarity environments appears to be a difficult task. In order to derive a tighter $s_{max}(v, Reg(N))$ bound, the $n \cdot (n-1)/2$ relative distances between the n query values could be taken into account. However, without specific assumptions on the similarity environment, additional hypotheses on the scoring functions (such as differentiability) seem to be needed to obtain major improvements. We leave this problem as a future research activity.

In the case of spatial access methods, which only manage similarity environments of type $\mathcal{DS}_{sam} = (\Re^n, L_p, h, \mathcal{L})$, that is, vector spaces with L_p (Euclidean, Manhattan, etc.) metrics,[3] the similarity bound established by (5) could be improved by trying to exploit the geometry of the Cartesian space. However, for arbitrarily complex queries this still remains a difficult task [SK97, HM95].

5 The Extended M-tree

In order to verify actual performance obtainable by processing complex queries with a distance-based access method, we have extended the M-tree. The M-tree stores the indexed objects into fixed-size nodes, which correspond to regions of the metric space. Each entry in a leaf node has the format $[v_j, \mathrm{oid}(v_j)]$, where v_j are the feature values of the object whose identifier is $\mathrm{oid}(v_j)$. The format of an entry in an internal node is $[v_r, r(v_r), \mathrm{ptr}(N)]$, where v_r is a feature value (*routing object*), $r(v_r) > 0$ is its *covering radius*, and $\mathrm{ptr}(N)$ is the pointer to a child node N. We remind that the semantics of the covering radius is that each object v in the sub-tree rooted at node N satisfies the constraint $d(v_r, v) \leq r(v_r)$.[4] Thus, the M-tree organizes the metric space into a set of (possibly overlapping) regions, to which the same principle is recursively applied.

Our M-tree implementation is based on the Generalized Search Tree (GiST) C++ package, a structure extendible both in the data types and in the queries it can support [HNP95]. In this framework, a specific access method is obtained by providing the code for a limited set of required methods. Among them, only the `Consistent` method is used

[3] Indeed, they can also manage quadratic distance functions, as shown in [SK97], but this does not make a substantial difference.

[4] For performance reasons, each entry also stores the distance between the feature value of that entry and the parent routing object. For more specific details see [CPZ97].

during the search phase, and had therefore to be modified to support complex queries. The `Consistent` method returns *false* iff all objects in the sub-tree of the considered node are guaranteed to be outside of the query region. The search algorithm uses this information to descend all paths in the tree whose entries are consistent with the query.

The new version of the `Consistent` method we have developed is based on the results obtained in the previous Section, and is fully parametric in the similarity environment $\mathcal{DS}$. The overall architecture of the method is shown in Figure 3.

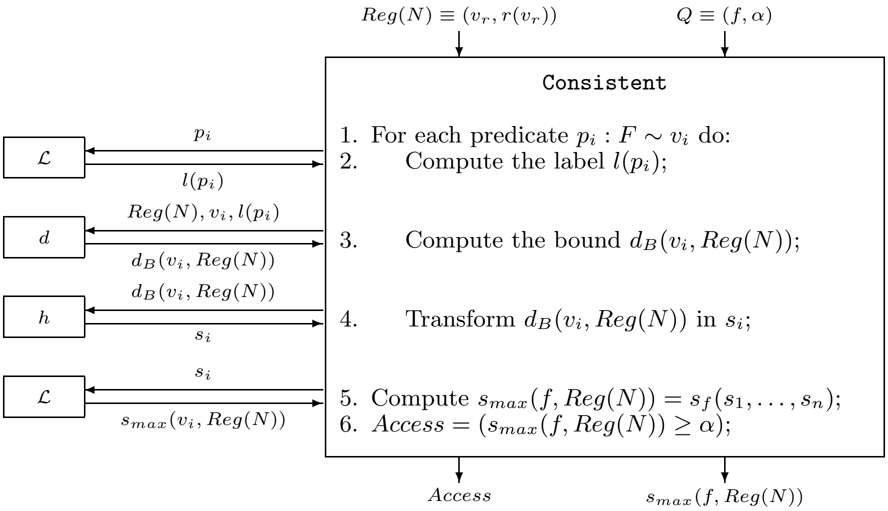

Fig. 3. The `Consistent` method.

The input of the `Consistent` method is an entry $Reg(N) \equiv (v_r, r(v_r))$, representing the region of node N, and a query $Q \equiv (f, \alpha)$, where $f = f(p_1, \ldots, p_n)$ is (a suitable encoding of) a formula of language $\mathcal{L}$, and α is a minimum similarity threshold. In order to optimize nearest neighbors queries, the new version of `Consistent` also returns an upper bound, $s_{max}(f, Reg(N))$, on the similarity between the objects in the region $Reg(N)$ and the formula f. At each step of the k-NN algorithm, the node with the highest bound is selected and fetched from disk.

The architecture of `Consistent` is independent of the similarity environment $\mathcal{DS}$, since all specific computations are performed by external modules implementing each component of the environment (these are shown on the left side in Figure 3). In particular, the two modules denoted with $\mathcal{L}$ are those which encode language-specific information. The execution flow closely follows the logic of Theorem 4 (refer to Figure 3): at step 2, we compute a Boolean *label* for each predicate p_i, to determine if the scoring function s_f is monotonic increasing or decreasing in the variable $s(p_i, v)$. Then, distance bounds are computed, depending on the value of $l(p_i)$ and transformed, by using the correspondence function h, into similarity bounds s_i. The overall upper bound $s_{max}(f, Reg(N))$ is then computed, by applying the scoring function s_f to the bounds obtained for all the predicates. Finally, if $s_{max}(f, Reg(N))$ is lower than α ($Access =$ *false*) N can be pruned from the search.

6 Experimental Results

In this Section we compare the performance of the extended M-tree with that of other search techniques. For the sake of definiteness, we consider the specific similarity environment $\mathcal{DS} = ([0,1]^5, L_\infty, 1-x, \mathcal{FS})$, where $L_\infty(v_x, v_y) = \max_j\{|\, v_x[j] - v_y[j]\,|\}$. The results we present refer to 10-NN conjunctive queries $f : p_1 \wedge p_2 \wedge \ldots$, evaluated over a collection of 10^4 objects. These are obtained by using the procedure described in [JD88], which generates normally-distributed clusters. The intra-cluster variance is $\sigma^2 = 0.1$, and clusters' centers are uniformly distributed in the 5-dimensional unit hypercube. Unless otherwise stated, the number of clusters is 10. The M-tree implementation uses a node size of 4 KBytes.

The alternative search algorithms against which we compare M-tree are a simple linear scan of the objects – the worst technique for CPU costs, but not necessarily for I/O costs – and the $\mathcal{A}_0'$ algorithm [Fag96], a variant of the general $\mathcal{A}_0$ algorithm suitable for conjunctive queries under $\mathcal{FS}$ semantics, which is briefly described below.[5]

Algorithm $\mathcal{A}_0'$ (Fagin's algorithm for $\mathcal{FS}$ conjuctive queries)

Input: conjunctive $\mathcal{FS}$ formula $f : p_1 \wedge \ldots \wedge p_n$, collection $\mathcal{C}$, integer $k \geq 1$;
Output: the k best matches in $\mathcal{C}$ with respect to f;

1. For each p_i, open a *sorted access* index scan and insert objects in the set X^i; stop when there are at least k objects in the intersection $L = \cap_i X^i$.
2. Compute $sim(f, v)$ for each $v \in L$. Let v_0 be the object in L having the least score, and p_{i_0} the predicate such that $sim(f, v_0) = sim(p_{i_0}, v_0)$.
3. Compute $sim(f, v_c)$ $\forall$ *candidate* $v_c \in X^{i_0}$, such that $sim(p_{i_0}, v_c) \geq sim(p_{i_0}, v_0)$
4. Return the k candidates with the highest scores.

In step 1, *sorted access* means that the index scan will return, one by one, objects in decreasing score order with respect to p_i, stopping when the intersection L of values returned by each scan contains at least k objects. The scan i_0 is the one which returned the object v_0 with the least score in L (step 2). For all the *candidates* [Fag96], that is objects v_c returned by such a scan *before* v_0 ($sim(p_{i_0}, v_c) \geq sim(p_{i_0}, v_0)$), step 3 computes the overall score. Since the M-tree does not implement yet a sorted access scan, to evaluate the costs of algorithm $\mathcal{A}_0'$ we simulated its behavior by repeatedly performing k_i nearest neighbors query for each predicate, with increasing values of k_i, until 10 common objects were found in the intersection. The sorted access costs are then evaluated as the costs for the last n queries (one for each predicate). Note that this is an optimistic cost estimate, since it is likely that a "real" sorted access scan would cost more than a single k_i nearest neighbors query which "magically" knows the right value of k_i. Step 3 is charged only with CPU costs, since we assume that all the *candidates* are kept in main memory.

Figures 4 and 5 compare CPU and I/O costs, respectively, for 10-NN queries consisting of the conjunction of two predicates, $f : p_1 \wedge p_2$, as a function of the distance between the two query objects. CPU costs are simply evaluated as the number of distance computations, and I/O costs are the number of page reads. As the graphs show, $\mathcal{A}_0'$ performs considerably better when the query objects are relatively "close", because in this case it is likely that the two query objects will have almost the same neighbors, thus leading to cheaper costs for the sorted access phase. On the other hand, the M-tree approach is substantially unaffected by the distance between the query objects. When query objects are "close" (i.e. the user asks for a conjunction of similar objects) the CPU costs for both approaches are very similar, while I/O costs for $\mathcal{A}_0'$ tend to be twice those of our approach. Comparison with linear scan show that our approach is highly effective in reducing CPU costs, which can be very time-consuming for complex distance functions, and also lowers I/O costs. $\mathcal{A}_0'$, on the other hand, is much worse than linear scan for "distant" query objects.

Figures 6 and 7 show the effect of varying data distribution, by generating 10^3 clusters. Now the distribution of objects' relative distances has a lower variance with respect to the previous case, thus making more difficult the index task. Major benefits are still obtained from reduction of CPU costs, whereas I/O costs of M-tree are

[5] We have slightly changed notation and terminology to better fit our scenario. In particular, we access data through index scans, whereas Fagin considers generic independent *subsystems*.

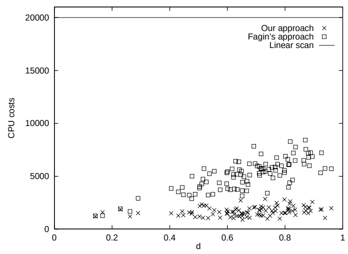

Fig. 4. CPU costs. $f : p_1 \wedge p_2$. 10 clusters

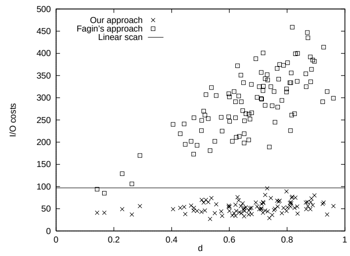

Fig. 5. I/O costs. $f : p_1 \wedge p_2$. 10 clusters

comparable to that of a linear scan only for not too far query objects.

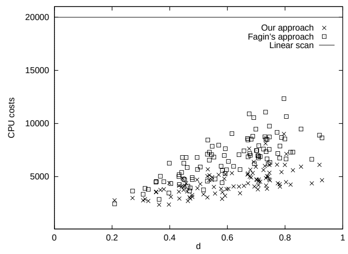

Fig. 6. CPU costs. $f : p_1 \wedge p_2$. 10^3 clusters

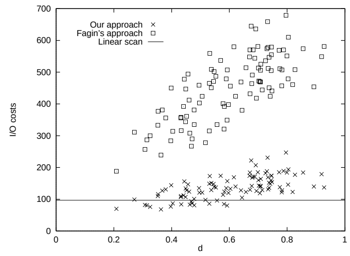

Fig. 7. I/O costs. $f : p_1 \wedge p_2$. 10^3 clusters

We now consider the case where one of the two predicates is negated i.e. $f : p_1 \wedge \neg p_2$. In this case, as Figures 8 and 9 show, the trend of index-based algorithms is somewhat inverted with respect to the previous cases, thus favoring, as to I/O costs, the linear scan when query objects are close. The performance degradation of M-tree in this case is because, due to negation, best matches are far away from both the query objects, thus leading to access a major part of the tree.

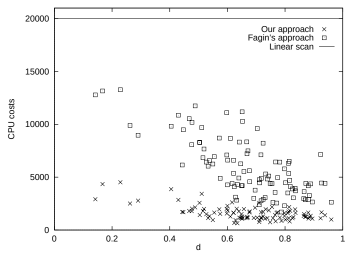

Fig. 8. CPU costs. $f : p_1 \wedge \neg p_2$. 10 clusters

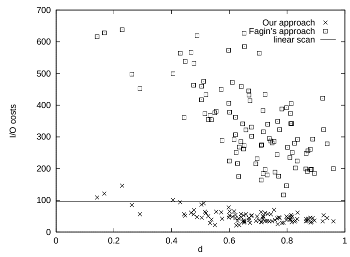

Fig. 9. I/O costs. $f : p_1 \wedge \neg p_2$. 10 clusters

Finally, we compared our approach with $\mathcal{A}_0'$ in the case of n positive conjuncts, that is $f : p_1 \wedge \ldots \wedge p_n$, for n in the range $[2, 5]$. Results, not shown here, report that I/O savings are never less than 90%, while CPU savings have a decreasing trend, starting from 85% down to 45%. This is explained by observing that, at each call of the `Consistent` method, we evaluate distances with respect to all the predicates. On the other hand, $\mathcal{A}_0'$ algorithm does this only for the *candidate* objects, as above explained.

The overall conclusions we can draw from above results, as well as others not shown here for brevity, can be so summarized:

1. Processing complex queries as a whole is *always* better than performing multiple sorted access scans (as $\mathcal{A}_0'$ does), both as to I/O and CPU costs.
2. For a givenformula, our approach is almost insensitive to the specific choice of the query objects, thus leading to stable performance. This is not the case for the $\mathcal{A}_0'$ algorithm.

3. In some cases linear scan can be preferable as to I/O costs, but our approach leads in any case to a drastic reduction of CPU costs, which can become the dominant factor for CPU-intensive distance functions typical of multimedia environments.

7 Related Work and Conclusions

In this paper we have considered the problem of efficiently evaluating *complex* similarity queries over generic feature domains. We have introduced a general formal framework to deal with complex queries and have shown how query evaluation can be carried out by using *any* access method which assesses similarity through distance measures, which is the usual case. Finally, we have demonstrated the effectiveness of our approach by extending the M-tree metric access method, and showing how it leads to considerable performance improvements over other query processing strategies.

The paper by Fagin [Fag96] on processing complex nearest neighbors queries has been fundamental to the development of our work. However, while Fagin is concerned with general queries over multiple features (and multiple systems), we have considered the specific case of single-feature queries and have consequently exploited this fact to derive a more efficient algorithm. The work by Chaudhuri and Gravano [CG96] addresses issues similar to [Fag96]. The strategy the authors propose to transform complex (multi-feature) nearest neighbors queries into a conjunction of simple range queries could in principle be also applied to our framework. However, this requires some knowledge of data and distance distributions, which might not be always available. Furthermore, and more important, it only applies if distance constraints can be derived for the single range queries, which is not always the case (see Example 8).

Our work shares some similarities with relevance feedback techniques used by text retrieval systems [Har92]. Such techniques derive a new query from a previous one and from user's relevance judgments on previous results. In our terminology, this amounts to a change of a (single) query value, trying to "move" it closer towards relevant documents, whereas our approach queries the database with *multiple* query values, without collapsing them into a single value. Although we have not worked out the implications of our approach in the context of document retrieval, it is clear that we are more flexible since known relevance feedback techniques might be obtained as special cases, by setting up specific similarity environments. In some sense, our approach can be viewed as a means to apply relevance feedback over *any* feature domain.

The only assumptions we have done throughout the paper concern the distance functions, which are to be *metrics*, and the *scoring functions* used to evaluate the overall objects' scores, which have to be monotonic in all their arguments (but not necessarily monotonic increasing or decreasing). Both these assumptions are general enough to capture almost any scenario of interest in multimedia environments.

We need the distance function to be a metric only for practical reasons, since no access method we are aware of is able to index over non-metric spaces. However, it is easy to see that relaxing this assumption would not affect our main formal result (Theorem 4). Thus, an hypothetical access method for non-metric spaces could exploit the results in this paper without modifications. A practical non-metric distance for which this would be helpful is described in [FS96].

We need scoring functions to be monotonic in all their arguments in order to derive similarity bounds within a data region, which is a basic requirement for Theorem 4 to apply. Although monotonicity is clearly a reasonable property to demand to a scoring function [Fag96], there are indeed cases for which it does not hold. For instance, assume one wants to retrieve all the objects v which are similar to v_1 as they are to v_2 (up to a certain tolerance ϵ), that is, $\mid s(F \sim v_1, v) - s(F \sim v_2, v) \mid \leq \epsilon$. This "equi-similarity" (or "equi-distance") query does not fit our framework, however it appears that access methods *can* indeed be further extended to process it too. We plan to investigate this and similar problems in our future work.

Acknowledgements

We thank Fausto Rabitti, Pasquale Savino, and Roger Weber for helpful discussions about the subject of the paper. We also acknowledge the comfortable environment of the HERMES project within which this work has originated.

References

[BO97] T. Bozkaya and M. Ozsoyoglu. Distance-based indexing for high-dimensional metric spaces. In *Proceedings of the 1997 ACM SIGMOD International Conference on Management of Data*, pages 357–368, Tucson, AZ, May 1997.

[Bri95] S. Brin. Near neighbor search in large metric spaces. In *Proceedings of the 21st VLDB International Conference*, pages 574–584, Zurich, Switzerland, September 1995.

[CG96] S. Chaudhuri and L. Gravano. Optimizing queries over multimedia repositories. In *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data*, pages 91–102, Quebec, Canada, June 1996.

[Chi94] T. Chiueh. Content-based image indexing. In *Proceedings of the 20th VLDB International Conference*, pages 582–593, Santiago, Chile, September 1994.

[CPZ97] P. Ciaccia, M. Patella, and P. Zezula. M-tree: An efficient access method for similarity search in metric spaces. In *Proceedings of the 23rd VLDB International Conference*, pages 426–435, Athens, Greece, August 1997.

[Fag96] R. Fagin. Combining fuzzy information from multiple systems. In *Proceedings of the 15th ACM Symposium on Principles of Database Systems*, pages 216–226, Montreal, Canada, June 1996.

[FS96] R. Fagin and L. Stockmeyer. Relaxing the triangle inequality in pattern matching. Research Report RJ 10031, IBM, June 1996.

[FW97] R. Fagin and E.L. Wimmers. Incorporating user preferences in multimedia queries. In *Proceedings of the 6th ICDT International Conference*, pages 247–261, Delphi, Greece, January 1997.

[Gut84] A. Guttman. R-trees: A dynamic index structure for spatial searching. In *Proceedings of the 1984 ACM SIGMOD International Conference on Management of Data*, pages 47–57, Boston, MA, June 1984.

[Har92] D. Harman. Relevance feedback and other query modification techniques. In W.B. Frakes and R. Baeza-Yates, editors, *Information Retrieval: Data Structures and Algorithms*, chapter 11, pages 241–263. Prentice Hall PTR, 1992.

[HM95] A. Henrich and J. Möller. Extending a spatial access structure to support additional standard attributes. In *Proceedings of the 4th International Symposium on Advances in Spatial Databases (SSD'95)*, volume 951 of *LNCS*, pages 132–151, Zurich, Switzerland, August 1995. Springer-Verlag.

[HNP95] J.M. Hellerstein, J.F. Naughton, and A. Pfeffer. Generalized search trees for database systems. In *Proceedings of the 21st VLDB International Conference*, pages 562–573, Zurich, Switzerland, September 1995.

[Jai96] R. Jain. Infoscopes: Multimedia information systems. In B. Furht, editor, *Multimedia Systems and Techniques*, chapter 7, pages 217–253. Kluwer Academic Publishers, 1996.

[JD88] A.K. Jain and R.C. Dubes. *Algorithms for Clustering Data.* Prentice-Hall, 1988.

[KY95] G.J. Klir and B. Yuan. *Fuzzy Sets and Fuzzy Logic.* Prentice Hall PTR, 1995.

[RKV95] N. Roussopoulos, S. Kelley, and F. Vincent. Nearest neighbor queries. In *Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data*, pages 71–79, San Jose, CA, May 1995.

[SK97] T. Seidl and H.-P. Kriegel. Efficient user-adaptable similarity search in large multimedia databases. In *Proceedings of the 23rd VLDB International Conference*, pages 506–515, Athens, Greece, August 1997.

[Zad65] L.A. Zadeh. Fuzzy sets. *Information and Control*, 8:338–353, 1965.