

Integrating the Results of Multimedia Sub-Queries Using Qualitative Preferences

Ilaria Bartolini*, Paolo Ciaccia
DEIS - IEIIT-BO/CNR
University of Bologna, Bologna, Italy
{ibartolini,pciaccia}@deis.unibo.it

Vincent Oria
Computer Science NJIT
University Heights, Newark, NJ, USA
oria@cis.njit.edu

M. Tamer Özsu
School of Computer Science
University of Waterloo, Waterloo, Canada
tozsu@db.uwaterloo.ca

Abstract

When similarity queries over multimedia databases are processed by splitting the overall query condition into a set of sub-queries, the problem of how to efficiently and effectively integrate the sub-queries' results arises. The common approach is to use a (monotone) *scoring function*, like min and average, to compute an *overall* similarity score by aggregating the *partial* scores an object obtains on the sub-queries. In order to minimize the number of database accesses, a “middleware” algorithm, like TA by Fagin, Lotem and Naor, is applied to return only the *top k* highest scored objects.

In this paper we consider the use of *qualitative preferences* for the integration of partial scores. With qualitative preferences one can define arbitrary *partial* (rather than only linear) orders on database objects, which adds a large flexibility and avoids some drawbacks of scoring functions when query refinement is involved. For the purpose of efficient evaluation, we propose two integration algorithms: MPO, which delivers objects one layer at a time, layers being defined by the specific partial order at hand, and iMPO, which is an *incremental* algorithm that delivers one object at a time, thus suitable for top *k* queries. We provide formal evidence that iMPO, when equipped with Skyline preferences, will perform faster than TA, and demonstrate this

*Part of this work was performed while this author was visiting NJIT.

through experiments on a real image database. Besides being faster, iMPO can deliver results whose quality, in terms of standard *precision* measure, is comparable to that obtainable with scoring functions. However, results of iMPO provide a much better (i.e., less unbiased) overall view of the objects relevant to a query, which highly simplifies query refinement tasks.

1 Introduction

Specification and evaluation of multimedia (MM) queries are both difficult problems to be addressed for the development of effective MM tools and applications. Indeed, the formulation of a query on a MM database has to take into account both the intrinsic complexity to properly characterize the semantic content of multimedia objects and the difficulty that a user experiences when trying to exactly formulate her needs. With a large MM database, in which each object is characterized by means of a set of relevant, automatically extracted, low-level *features* (e.g., color, texture, and shape in the case of still images) and where the most common query modality is QBE (Query By Example), the user provides the system with a “target” (query) object and expects as result the “most similar” database objects. For this retrieval model to effectively work, it is well recognized that the similarity function used to compare objects has to be properly adapted, possibly by means of some *relevance feedback* [14], to fit the subjective user preferences.

When dealing with *complex* MM queries involving multiple features, the scenario is further complicated. Indeed, since it is a common case that features are separately indexed [13] or even managed by independent specialized sub-systems [10], an integration of partial results is needed. The usual approach (see, e.g., [10, 13, 11, 5]) is to have an independent, yet synchronized, evaluation of sub-queries, one for each involved feature. Each object returned by a sub-query has an associated *partial score* for the corresponding feature, which are then aggregated by means of some *scoring function*, like min, max, and avg, into an *overall score*. Under this view one object is better than (i.e., preferred to/ranked higher than) another iff its overall score is higher.

Example 1.1 (Scoring functions) *Consider a user who, through the system graphical interface, draws the image query shown on top of Figure 1 (image o_0) and aims to retrieve from the database those images with a palm tree on the right side (1st sub-query) and a sun on the left side (2nd sub-query). Consider two images o_1 and o_2 and assume that the partial scores of o_1 are, respectively, $s_{1,1} = 0.6$ and $s_{1,2} = 0.4$, and those of o_2 are $s_{2,1} = 0.5$ and $s_{2,2} = 0.6$. If the scoring function is min we get $s_1 = 0.4$ and $s_2 = 0.5$, thus image o_2 has to be preferred to o_1 . On the other hand if the scoring function is $s_i = 0.7 \times s_{i,1} + 0.3 \times s_{i,2}$ then we get $s_1 = 0.54$ and $s_2 = 0.53$,*

thus o_1 is now better than o_2 .

As above example shows, the choice of the scoring function can be a critical factor for the determination of the final result, that, when using scoring functions, usually consists of the k highest scored (top k) objects. A further problem is that it is a hard, if not impossible, task to derive a scoring function that can suitably represent user preferences.

Example 1.2 (Images) Consider the same query used in Example 1.1. If an image like o_0 does not exist in the database, the user would prefer images with both objects on the right (image o_1) rather than to have the sun on the right and a palm on the left (image o_2). Further, images with only the sun on the left (image o_3) are preferred to images with only a palm on the right (image o_4). The overall preference specification is represented in the figure, where we draw an arrow from o_i to o_j if o_i is preferred to o_j . We claim that no scoring function, defined independently of the actual

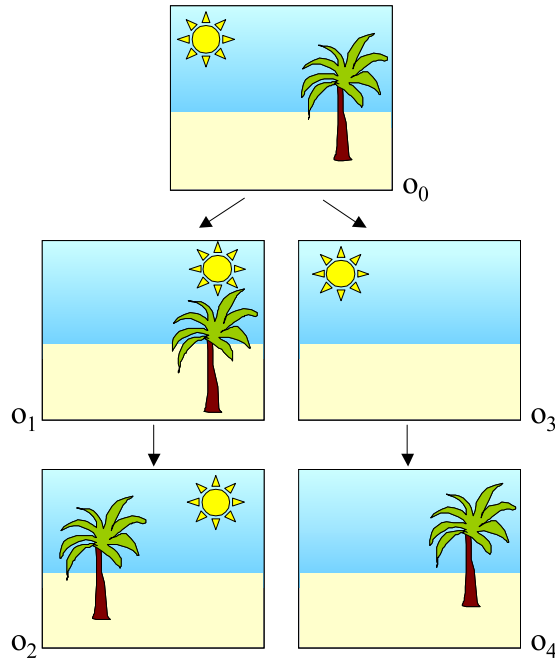


Figure 1: The target image o_0 and two user preferences

content of the database, can capture above preferences.¹ Indeed, assume that the database contains images $\{o_0, o_1, o_2, o_3, o_4\}$. Since no preference exists between o_1 and o_3 and between o_2 and o_4 , it is necessarily the case that $s_0 > s_1 = s_3 > s_2 = s_4$. Assume now that the only images in the database are $\{o_0, o_1, o_2, o_4\}$. In this case the “right” scoring function should

¹See, for a similar example, Example 7.4 in [7].

yield $s_0 > s_1 = s_4 > s_2$, since only o_0 is now to be preferred to o_4 . As a conclusion, in the first database instance we have $s_1 > s_4$, whereas in the second it is $s_1 = s_4$, which demonstrates the point.

What above example highlights is that scoring functions have indeed limited expressive power, since they can only capture a limited type of preferences. These, also called *quantitative preferences*, are exactly those that assign to each object a numerical value (its score, or “utility”) *regardless of the other objects in the database* [12]. As the example shows, user preferences are not always of this kind.

A third problem with scoring functions is that they provide a poor “coverage” of the best available alternatives. As shown in Example 1.1, the “best” objects depend on the adopted scoring function. This implies that, once a scoring function has been chosen, only a (very) limited portion of the potential best objects can be retrieved. Although relevance feedback mechanisms can alleviate this well-known problem, by allowing the user to progressively shift her focus towards interesting regions of the search space, they usually require several iterations before leading to acceptable results, thus generating a not negligible overhead on the system [1].

In this paper we propose a novel, more general, approach to sub-query integration based on *qualitative preferences* as an alternative to quantitative scoring functions. Qualitative preferences, which have been recently used in the context of relational databases [6], require only that, given a pair of objects o_i and o_j , one has some (binary) *preference relation* stating whether o_i is preferred to o_j ($o_i \succ o_j$) or not ($o_i \not\succeq o_j$). This approach includes scoring functions as a special case, since, given a scoring function, one can always define $o_i \succ o_j$ iff $s_i > s_j$. Note that with qualitative preferences it might be the case that neither $o_i \succ o_j$ nor $o_j \succ o_i$ hold, in which case o_i and o_j are *indifferent* (written $o_i \sim o_j$).

Although qualitative preferences enjoy much more flexibility than scoring functions, in this paper we focus on the well-defined and intuitive case, which we call *partial order (PO) integration* where the preference relation defines a (strict) partial order on database objects, thus $o_i \not\succeq o_i$ (irreflexivity) and $o_i \succ o_j, o_j \succ o_k \Rightarrow o_i \succ o_k$ (transitivity).

The rationale for using PO stems from the basic observation that *all* the problems that plague scoring functions are no longer a concern. Indeed, expressiveness is not a problem with PO, as argued above. PO is not forced to use a scoring function in order to rank objects, rather it can use other, more sophisticated and flexible, criteria that directly take into account all the partial scores. This has the consequence that there is no risk of choosing “bad parameter values” for the scoring function. Finally, with PO it is possible to get an “overall view” of the potential best objects for a given query, a fact that highly simplifies the task of focusing on the right part of the search space. To this end we will show that PO’s relevant results

cover much better than scoring functions the search space, as demonstrated through experiments on a real-world image database.

The model of queries we consider includes the standard one, where one is interested in obtaining the top k results, the major difference being, of course, the criterion according to which objects are ranked. To this end we rely on the well-defined (equivalent) semantics of the *Best* [15] and *Winnnow* [6] operators, recently proposed in the context of relational databases. The Best operator $\beta_{\succ}(C)$ returns all the objects o in a collection C such that there is no object in C better than o according to relation \succ . Ranking can be easily obtained by recursively applying the Best operator to the remaining objects (i.e., those not in $C - \beta_{\succ}(C)$, and so on). This leads to a *layered* view of the search space where all the objects in one layer are indifferent. Thus, besides top k queries, PO also naturally supports a “first ℓ layers” query model, which adds further flexibility to the retrieval phase.

For the purpose of efficient query evaluation, we propose two integration algorithms. Algorithm MPO applies to *any* preference relation that defines a strictly monotone PO (see Definition 6) and returns the first ℓ layers of the partial order. In order to efficiently support top k queries, we then modify MPO into an *online* algorithm, iMPO, which delivers its results to the user in an incremental way. We experiment with two specific preference relations: 1) so-called (layered) *Skyline* (SL) preferences [4], for which an object o belongs to $\beta_{\succ_{SL}}(C)$ iff there is no other object that dominates o on all sub-queries; and 2) *Region-prioritized Skyline* (RS) preferences, which avoid some drawback of Skyline preferences when used for multimedia queries.

Our contributions are summarized as follows:

- We provide a formal demonstration of the general inadequacy of scoring functions in supporting the retrieval of the potential best matches of a query (Theorem 1).
- We introduce qualitative preferences, in particular strictly monotone partial orders, as a new model for the integration of sub-queries’ results, and present the new Region-prioritized Skyline (RS) preference relation.
- We detail two integration algorithms, MPO and iMPO, for the efficient computation of the result of “first ℓ layers” and top k queries, respectively.
- Under rather broad assumptions, we prove (Theorem 3) that iMPO, equipped with Skyline preferences, will deliver results faster than TA [11], which is the algorithm using scoring functions we take for reference purpose.
- We demonstrate, through experiments on a real-world image database, that the quality of the results returned by iMPO and TA, as measured

in terms of classical *precision*, is comparable; however:

- iMPO requires less database accesses than TA to get the same number of relevant objects, and
- iMPO results, especially when using RS preferences, reflect much better than TA (using either min or avg) the actual distribution of relevant objects in the database, thus highly simplifying query refinement activities.

The paper is organized as follows. In Section 2 we provide the basic definitions concerning the query scenario and the integration problem; then we discuss scoring functions and their intrinsic limits. Section 3 introduces qualitative preferences. In Section 4 we present the MPO and iMPO algorithms, and in Section 5 we describe experimental results. Table 1 lists relevant symbols used in the paper.

Symbol	Description
C	collection of objects
o_i	an object of collection C
\mathcal{Q}	complex query with m sub-queries
Q_q	q -th sub-query ($q = 1, \dots, m$)
$s_{i,q}$	partial score of o_i for Q_q ($s_{i,q} \in [0, 1]$)
A	answer space ($A = [0, 1]^m$)
\mathbf{s}_i	representative point of o_i ($\mathbf{s}_i \in A$)
$\underline{\mathbf{s}}$	the “threshold point”
S	scoring function
s_i	overall score of object o_i ($s_i = S(\mathbf{s}_i)$)
\succ	preference relation
β_{\succ}	the Best operator

Table 1: Relevant symbols used in the paper

2 The Integration Problem

Consider a collection C of objects and a *complex* MM query of the form $\mathcal{Q} = (Q_1, Q_2, \dots, Q_m)$ where each Q_q is a sub-query. With respect to the condition expressed by Q_q , we assume that each object $o_i \in C$ can be evaluated and assigned a *partial score*, $s_{i,q} \in [0, 1]$, assessing “how well” o_i matches Q_q condition (higher values are clearly better). Thus, the underlying collection C can be regarded as being composed of m lists, one for each sub-query, where the list L_q ($q = 1, \dots, m$) contains pairs of the form $(o_i, s_{i,q})$ and is linearly ordered according to scores, with higher partial scores at the beginning of the list.

The integration problem is to return as a result of \mathcal{Q} the “best” (in some sense to be defined) objects based on the L_q lists. We model access to the database in the usual way [10, 11, 5], assuming that objects can be retrieved through one of two distinct modalities:

- In the *sorted* (i.e., sequential) access modality one retrieves from a list L_q the next unseen object on that list, say o_i , together with its partial score, $s_{i,q}$.
- In the *random* access modality, given an object o_i seen via sorted access on some list L_q , one can retrieve from the database the needed features and, consequently, evaluate the missing partial scores for o_i .

2.1 Scoring Functions

The standard approach to define the semantics of a complex MM query is based on so-called *scoring functions*. Consider the m -dimensional space $A = [0, 1]^m$, which we can conveniently call the *answer space*. For a given query \mathcal{Q} , each object o_i is univocally represented in A by a point $\mathbf{s}_i = (s_{i,1}, \dots, s_{i,m})$, whose coordinate values are its m partial scores for the m sub-queries.

Definition 1 (Scoring function) *A scoring function on A is any function $S : A \rightarrow [0, 1]$ that assigns to each point $\mathbf{s}_i \in A$ a value $s_i = S(\mathbf{s}_i)$, called the overall score of o_i .*

Intuitively, the scoring function S is the “rule” that one wants to adopt to give proper credit to the partial scores and, consequently, to the corresponding sub-queries.

In the last years a large variety of algorithms have been proposed to efficiently evaluate the result of *top k* complex queries, i.e., when one is interested in obtaining the k highest scored objects, with ties to be arbitrarily broken. Approaches that make specific hypotheses on how the sub-queries are evaluated, like [8] and [3], can yield superior performance, yet they have limited applicability and imply a modification of the access methods used for sub-query evaluation. On the other hand, “middleware” algorithms, like \mathcal{A}_0 [10], TA and NRA [11], and Upper [5], to name a few, just rely on the sorted and random access modalities and on the hypothesis of *monotonicity* of the scoring function.

Definition 2 (Monotonicity of scoring functions) *An m -ary scoring function S is monotone if $s_{j,q} \leq s_{i,q}$ for all q implies $s_j = S(s_{j,1}, \dots, s_{j,m}) \leq s_i = S(s_{i,1}, \dots, s_{i,m})$.*

In practice, most commonly used scoring functions, like min, max, avg, etc., as well as their “weighted” extensions, are not only monotone, but also *strictly* monotone, that is, $s_{j,q} < s_{i,q}$ for all q implies $s_j < s_i$.

Monotonicity and strict monotonicity can be given a simple yet useful geometric interpretation. For this consider the “target” point in A defined as $\mathbf{1} = (1, \dots, 1)$, which corresponds to the best possible evaluation for all the sub-queries, and let R_i be the hyper-rectangle having points \mathbf{s}_i and $\mathbf{1}$ as opposite vertices. We call R_i the hyper-rectangle of o_i .

Observation 1 *If S is (strictly) monotone and \mathbf{s}_j is a point of (resp., in the interior of) the hyper-rectangle R_i of o_i , then $S(\mathbf{s}_i) \leq S(\mathbf{s}_j)$ ($S(\mathbf{s}_i) < S(\mathbf{s}_j)$).*

As a simple corollary, if there are at least k points of C in the hyper-rectangle of o_i , then there is no strictly monotone scoring function that can make o_i one of the k highest scored objects in C . Further, since ties can be arbitrarily broken, o_i can also be safely neglected in case of monotone scoring functions. This is precisely captured by the concept of *potential (k) best match*:

Definition 3 (Potential k best match) *An object o_i for which there are $k - 1$ points of C in its hyper-rectangle is called a potential k best match. A potential 1 best match is simply called a potential best match.*

2.2 (In)completeness of Scoring Functions

It is clear that any top k query based on a monotone scoring function can only return points that are potential k best matches. In practice, *which* of them are more relevant to the user is difficult to assess *a priori*, and relevance feedback mechanisms are usually used to this end. Under this view, the search process is better understood as the search of a “good” scoring function within a *class of scoring functions with parameters* (usually called “weights”) [1]. As an example, one could use the class of weighted averages and change the weight to be assigned to each partial score (sub-query) depending on the correlation between the sub-query scores and objects’ relevance, as assessed by the user.

A natural requirement for a class of scoring functions is that of *completeness*.

Definition 4 (Completeness of scoring functions) *A class $\mathcal{C}(\mathcal{W}) = \{S_W\}$ of scoring functions with parameters \mathcal{W} is complete iff there is an assignment of values W to the \mathcal{W} parameters such that if \mathbf{s}_i is a potential best match, then $S_W(\mathbf{s}_i) \geq S_W(\mathbf{s}_j)$ holds for all \mathbf{s}_j .*

Thus, a complete class makes it possible to “explore” the space of potential best matches, whereas if $\mathcal{C}(\mathcal{W})$ is incomplete it will inevitably miss some potentially useful answer. To obviate this one could use larger values of k . However, for any fixed k value it is always possible to produce a collection of objects such that \mathbf{s}_i is a potential k best match, yet o_i cannot be retrieved by any scoring function in the given (incomplete) class.

Let $\min(\mathcal{W})$ be the “weighted min” class defined as follows:

$$\min_W(\mathbf{s}_i) = \min_q \{w_q \times s_{i,q}\} \quad (1)$$

where weights w_q have to be chosen in such a way to ensure $\min_W(\mathbf{s}_i) \in [0, 1]$. We have the following positive result.

Lemma 1 *The class $\min(\mathcal{W})$ is complete.*

Proof. Consider the case $k = 1$, the proof for $k \geq 1$ requiring similar arguments. Let \mathbf{s}_i be a potential best match and assume, without loss of generality, that $s_{i,1} = \max_q \{s_{i,q}\}$. Define $w_q = s_{i,1}/s_{i,q}$ $q = 1, \dots, m$ and denote with W_i the resulting weight vector. By substituting in Equation (1) it is immediately obtained that $\min_{W_i}(\mathbf{s}_i) = s_{i,1}$. Consider now another point \mathbf{s}_j . Since \mathbf{s}_i is a potential best match, \mathbf{s}_j is external to the R_i hyper-rectangle. In turn, this implies that there exists at least one coordinate (sub-query), say q^* , for which $s_{j,q^*} < s_{i,q^*}$. The overall score of object o_j is:

$$\min_{W_i}(\mathbf{s}_j) = \min_q \left\{ \frac{s_{i,1}}{s_{i,q}} \times s_{j,q} \right\} = s_{i,1} \times \min_q \left\{ \frac{s_{j,q}}{s_{i,q}} \right\} < s_{i,1}$$

thus $\min_{W_i}(\mathbf{s}_j) < \min_{W_i}(\mathbf{s}_i)$ as claimed. \square

Although $\min(\mathcal{W})$ is complete, it is hardly used in practice because it is not differentiable with respect to the \mathcal{W} parameters, a fact which precludes common (numerical) feedback mechanisms to be applied. Further, \min (thus $\min(\mathcal{W})$ as well) has often been criticized because of its lack of sensitivity to changes in non-minimal partial scores. To this end, for the purpose of sub-query integration, weighted averages are much more appropriate:

$$\text{avg}_W(\mathbf{s}_i) = \text{avg}_q \{w_q \times s_{i,q}\} \quad (2)$$

The following result however shows that even $\text{avg}(\mathcal{W})$ is not exempt of problems.

Lemma 2 *The class $\text{avg}(\mathcal{W})$ is incomplete.*

Sketch of proof. Consider the case $k = 1$ and 2 sub-queries, i.e., $m = 2$. Let $\mathbf{s}_0 = (s_{0,1}, s_{0,2})$, where both partial scores are strictly less than 1, be a potential best match. Take 2 points \mathbf{s}_1 and \mathbf{s}_2 with scores:

$$\begin{aligned} \mathbf{s}_1 &= (s_{0,1} + \delta, s_{0,2} - \epsilon) \\ \mathbf{s}_2 &= (s_{0,1} - \epsilon, s_{0,2} + \delta) \end{aligned}$$

with $\delta > \epsilon > 0$. Any choice of the weight values in Equation (2) either yields $\text{avg}_W(\mathbf{s}_0) < \text{avg}_W(\mathbf{s}_1)$ or $\text{avg}_W(\mathbf{s}_0) < \text{avg}_W(\mathbf{s}_2)$. Extension to the

case $m > 2$ requires m (non-collinear) points $\mathbf{s}_1, \dots, \mathbf{s}_m$ chosen in such a way that \mathbf{s}_0 lies “below” the hyper-plane spanned by such points. \square

The following major result proves the general inadequacy of scoring functions to cover the space of possible best matches.

Theorem 1 *A class of monotone scoring functions is complete iff it includes $\min(\mathcal{W})$.*

Sketch of proof. The “if” part is Lemma 1. To prove necessity, we rely on geometric arguments applied to the case $k = 1$, the extension to $k \geq 1$ being straightforward. Let \mathbf{s}_0 be a potential best-match, and consider a class $\mathcal{C}(\mathcal{W})$ that does not include $\min(\mathcal{W})$. In particular, $\mathcal{C}(\mathcal{W})$ does not include the scoring function \min_{W_0} , defined as in the proof of Lemma 1. Consider a scoring function $S_W \in \mathcal{C}(\mathcal{W})$ and the closed region $Reg_W(\mathbf{s}_0) \subset [0, 1]^m$ delimited by the space boundaries and bounded from below by the hyper-surface $B_W(\mathbf{s}_0)$ passing for \mathbf{s}_0 with equation $S_W(\mathbf{s}_j) = S_W(\mathbf{s}_0)$ ($B_W(\mathbf{s}_0)$ is just the locus of points with the same overall score of o_0). Since S_W is monotone and $S_W \neq \min_{W_0}$, for *any* choice of W the region $Reg_W(\mathbf{s}_0)$ strictly includes the hyper-rectangle R_0 of o_0 defined by \mathbf{s}_0 and $\mathbf{1}$. That is, the “delta region” $\Delta_W(\mathbf{s}_0) = Reg_W(\mathbf{s}_0) - R_0$ is not empty. It follows that it is always possible to have a point $\mathbf{s}_1 \in \Delta_W(\mathbf{s}_0)$. Then, since \mathbf{s}_1 is “above” $B_W(\mathbf{s}_0)$, it is $S_W(\mathbf{s}_0) < S_W(\mathbf{s}_1)$. \square

Above result highlights an intrinsic limit of very natural classes of scoring functions, such as $\text{avg}(\mathcal{W})$. This, together with the observation that the exploration of the space of potential best matches, through relevance feedback mechanisms, is inherently slow, suggests us to look for other, more general and efficient, alternatives to the integration problem.

3 Qualitative Preferences

We ground our approach on the basic notion of *qualitative preferences*. Unlike quantitative ones, qualitative preferences do not necessarily require the specification of a scoring function, rather they are directly represented by a *preference relation*.

Definition 5 *Let X be a domain of values. A preference relation over X is a binary relation \succ over $X \times X$. If $x_1, x_2 \in X$ and $(x_1, x_2) \in \succ$, we also write $x_1 \succ x_2$ and say that x_1 is preferable to x_2 or, equivalently, that x_1 dominates x_2 . If neither $x_1 \succ x_2$ nor $x_2 \succ x_1$ hold, we say that x_1 and x_2 are indifferent, written $x_1 \sim x_2$.*

In the context of sub-query integration, the X domain is the answer space $A = [0, 1]^m$, whose values are the representative points of the objects. Let o_i and o_j be two objects in the C collection and \mathbf{s}_i and \mathbf{s}_j be their corresponding

points in A . With a slight abuse of notation we write $o_i \succ o_j$, and say that o_i dominates (is preferable to) o_j , whenever $\mathbf{s}_i \succ \mathbf{s}_j$.

In this paper we focus our attention on a specific case of qualitative preferences, namely those for which the preference relation is a *partial order* (PO). We remind that \succ is a (strict) partial order if it is *irreflexive* ($x \not\succeq x$) and *transitive* ($x_1 \succ x_2$ and $x_2 \succ x_3$ imply $x_1 \succ x_3$). Note that *asymmetry*, that is $x_1 \succ x_2$ implies $x_2 \not\succeq x_1$, directly follows from the irreflexivity and transitivity properties. Dealing with PO preference relations is strictly more general than using scoring functions, since *any* scoring function S can be viewed as a PO preference relation \succ_S by defining $o_i \succ_S o_j \Leftrightarrow S(\mathbf{s}_i) > S(\mathbf{s}_j)$ [12].

What kind of PO preference relations are suitable for sub-query integration? A reasonable requirement is that \succ , although not necessarily based on the comparison of overall scores, still enjoys some kind of “monotonicity” property that establishes some positive correlation between partial scores and the dominance relation.

Definition 6 (Monotonicity of preference relations) *A preference relation \succ over the answer space $A = [0, 1]^m$ is monotone if $s_{j,q} \leq s_{i,q}$ for all q implies $\mathbf{s}_j \not\succeq \mathbf{s}_i$. If $s_{j,q} < s_{i,q}$ for all q implies $\mathbf{s}_i \succ \mathbf{s}_j$ we say that \succ is strictly monotone.*

Strict monotonicity, as in the case with scoring functions, excludes the presence of “indifference regions”. Indeed, if \succ is monotone, but not strictly monotone, and $s_{j,q} < s_{i,q}$ holds for all q , we might have $\mathbf{s}_j \not\succeq \mathbf{s}_i$ and $\mathbf{s}_i \not\succeq \mathbf{s}_j$, thus $\mathbf{s}_i \sim \mathbf{s}_j$.

In the following we will only consider strictly monotone PO preference relations. This does not appear to be a relevant restriction, and has the major advantage of allowing us to derive efficient integration algorithms (see next section). The first example of a strictly monotone PO preference relation are the so-called *Skyline preferences* [4].

Definition 7 (Skyline preferences) *The Skyline preference relation \succ_{SL} over $A = [0, 1]^m$ is defined as follows:*

$$o_i \succ_{SL} o_j \Leftrightarrow (\forall q : s_{j,q} \leq s_{i,q}) \wedge (\exists q : s_{j,q} < s_{i,q}) \quad (3)$$

where $s_{i,q}$ ($s_{j,q}$) is the partial score that object o_i (resp., o_j) obtains for sub-query Q_q . Thus, o_i is preferred to o_j iff it is at least as good as o_j on all sub-queries and there is at least one sub-query for which the partial score of o_i is strictly higher than that of o_j . The set of points (objects) of a collection C for which there is no object that dominates them according to \succ_{SL} is called the *Skyline of C* .

There is a simple connection between Skyline preferences and the potential best matches of C (see Definition 3).

Lemma 3 *An object o_i is a potential best match of C iff it belongs to the Skyline of C .*

Proof. Immediate from the definitions. \square

Figure 2 shows the results of a sample query over an image database when Skyline preferences are used to integrate the results of the sub-queries, over color and texture features, respectively. The figure shows the target “eagle” image Q and the potential best matches (i.e., Skyline) of Q . Note that most of them belong to the same semantic class, “Birds”, of Q (more details on this point are given in Section 5) and that they are quite spread over the answer space. Even for this specific example it can be shown that no scoring function will be able to return the set of all potential best matches.

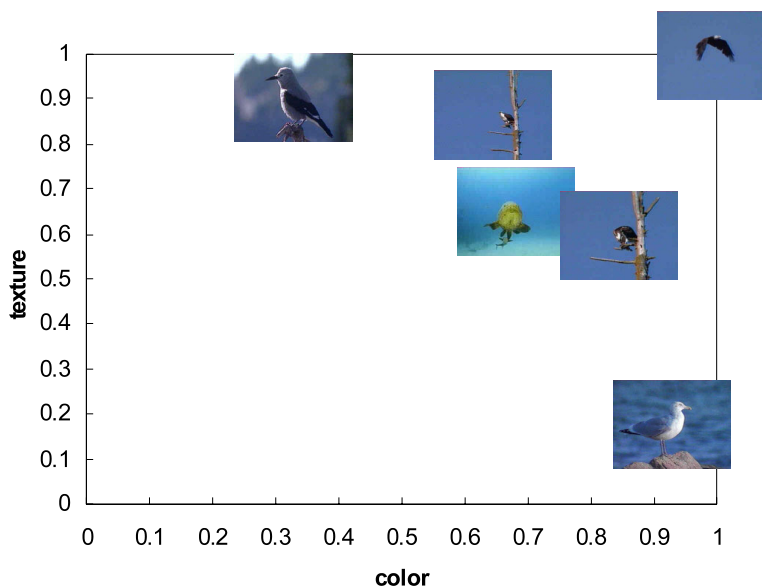


Figure 2: The Skyline of the “eagle” image

Skylines are not the whole story about strictly monotone POs. To give a flavor of the power of PO-based integration, we introduce a generalization of Skyline preferences that we call *Region-prioritized Skyline* (RS) preferences. For this we assume that the answer space A is partitioned into a set of regions A_1, \dots, A_P , and that a preference relation is also expressed over such regions.

Definition 8 (RS preferences) *Let $Y = \{A_1, \dots, A_P\}$ be any partition of $A = [0, 1]^m$, and let \succ_{Reg} be a PO preference relation over Y . Let $Reg()$ be a function that maps each point of A into its (unique) region of Y . The*

Region-prioritized Skyline (RS) preference relation \succ_{RS} over $A = [0, 1]^m$ is defined as follows:

$$o_i \succ_{RS} o_j \Leftrightarrow (Reg(\mathbf{s}_i) \succ_{Reg} Reg(\mathbf{s}_j)) \vee ((Reg(\mathbf{s}_i) = Reg(\mathbf{s}_j)) \wedge (\mathbf{s}_i \succ_{SL} \mathbf{s}_j)) \quad (4)$$

Thus, if two points belong to the same region the Skyline logic applies, whereas priority among regions prevails when two points belong to different regions. As a simple example, let $m = 2$ and $Y = \{A_1, A_2\}$, with $A_1 = [0, 1] \times [0.7, 1]$, $A_2 = [0, 1] \times [0, 0.7)$, and $A_1 \succ_{Reg} A_2$. Any point in the “upper rectangle” A_1 will dominate points in the “lower rectangle” A_2 . Intuitively, this will favor objects with a good partial score for sub-query Q_2 . Among such objects (if any), the best matches will be determined using Skyline preferences. If region A_1 is empty, then the best matches will be found in region A_2 .

Algorithm MPO (Input: query \mathcal{Q} , collection C , strictly monotone preference relation \succ)

- (1) Set $Result = \emptyset$; Set $\underline{s} = (1, \dots, 1)$; /* \underline{s} is the threshold point */
- (2) While ($\nexists (o_i, \mathbf{s}_i) \in Result$ such that $\mathbf{s}_i \succ \underline{s}$):
- (3) For each sub-query Q_q ($q = 1, \dots, m$) do:
- (4) Retrieve the next unseen object o_i from L_q ; /* sorted access */
- (5) Retrieve missing scores for the other sub-queries and obtain \mathbf{s}_i ; /* random accesses */
- (6) Set $Dominated = false$;
- (7) While ($\text{not}(Dominated) \wedge \exists (o_j, \mathbf{s}_j) \in Result$ unmatched with \mathbf{s}_i):
- (8) Compare \mathbf{s}_i with \mathbf{s}_j : $\begin{cases} \mathbf{s}_i \succ \mathbf{s}_j & \text{remove } (o_j, \mathbf{s}_j) \text{ from } Result, \\ \mathbf{s}_i \sim \mathbf{s}_j & \text{do nothing,} \\ \mathbf{s}_j \succ \mathbf{s}_i & \text{set } Dominated = true; \end{cases}$
- (9) End While;
- (10) If $\text{not}(Dominated)$ insert (o_i, \mathbf{s}_i) in $Result$;
- (11) Let \underline{s}_q be the lowest score seen by sorted access on list L_q ; Set $\underline{s} = (\underline{s}_1, \dots, \underline{s}_m)$;
- (12) End For;
- (13) End While;
- (14) If $(\mathbf{s}_i \succ \underline{s} \wedge \underline{s} \succ_{SL} \mathbf{s}_j)$ does not imply $\mathbf{s}_i \succ \mathbf{s}_j$ then /* execute the “final cycle” */
- (15) Repeat from step 3 to 10 until on each list L_q is found an object o_{j_q} with partial score $s_{j_q, q} < \underline{s}_q$;
- (16) Return $Result$.

Figure 3: The MPO algorithm

RS preferences can also be used to limit some pathological behavior of Skyline preferences. Indeed, let o_i be an object with $\mathbf{s}_i = (0.8, 0.1)$, and assume that 0.8 is the best score for sub-query Q_1 . If there is no other object with equal score for Q_1 , it is guaranteed that o_1 belongs to the Skyline, which a user could find questionable if she also sees (possibly many) other objects with somewhat “more balanced” score values. The classical threshold-based solution, besides suffering the problem of requiring detailed knowledge of the objects’ distribution in order to set a suitable threshold value, ignores

that the choice of whether or not o_i has to be discarded could depend on the presence of “more preferred” objects. Region prioritization easily solves the dilemma.

In order to guarantee strict monotonicity of \succ_{RS} we need to ensure that priority among regions does not contrast with Skyline preferences.

Lemma 4 *The preference relation \succ_{RS} is strictly monotone iff when $Reg(\mathbf{s}_i) \neq Reg(\mathbf{s}_j)$ and $s_{j,q} < s_{i,q}$ for all q then it is $Reg(\mathbf{s}_i) \succ_{Reg} Reg(\mathbf{s}_j)$.*

Sketch of proof. Proving that \succ_{RS} is indeed a PO is left as an exercise. (only if) Immediate from Definitions 6 and 8.

(if) We consider two cases: (a) if $Reg(\mathbf{s}_i) = Reg(\mathbf{s}_j)$ then the result follows from the strict monotonicity of \succ_{SL} ; (b) if $Reg(\mathbf{s}_i) \neq Reg(\mathbf{s}_j)$ and $s_{j,q} < s_{i,q}$ for all q , by hypothesis $Reg(\mathbf{s}_i) \succ_{Reg} Reg(\mathbf{s}_j)$, thus $o_i \succ_{RS} o_j$, as required. \square

It should be clear that if in Definition 8 we replace \succ_{SL} with *any* other strictly monotone PO preference relation we still get a valid strictly monotone PO region-prioritized preference relation. For instance, one could define four regions, $Y = \{A_1, A_2, A_3, A_4\}$, and within each of them use a, possibly different (!), preference relation, say \succ_{SL} in A_1 , \succ_{\min} in A_2 (where we make use of the min scoring function to compare objects), etc. Also, nothing would prevent to recursively apply region-prioritization, that is, by partitioning A_1 into sub-regions, and so on.

4 Query Evaluation

In this section we present two algorithms for efficiently performing PO-based integration of sub-query results. The first algorithm, called MPO, works for any strictly monotone PO preference relation. Since MPO returns *all* the potential best matches, it is unsuitable if one wants to explicitly control the cardinality of the result and the amount of resources needed to solve the query. To this end we introduce an *incremental* (i.e., online) algorithm, called iMPO, which returns an object o to the user as soon as it can be proven that, among the objects not yet returned, there is no object that dominates o .

4.1 The MPO Algorithm

The first algorithm we introduce is based on the (equivalent) semantics of the Best [15] and the Winnow [6] operators. The Best operator is defined as:

$$\beta_{\succ}(C) = \{o \in C \mid \nexists o' \in C, o' \succ o\} \quad (5)$$

In Figure 3 we present the MPO algorithm, whose logic is as follows. At each step MPO retrieves via sorted access (step 4) the best “unseen” object

o_i from one of the m sorted lists, and then obtains missing partial scores for such object via random access (step 5). The so-obtained representative point \mathbf{s}_i is then compared with the current objects in $\beta_{\succ}(C)$ (steps 7 and 8). If no objects o_j dominates o_i , o_i is inserted in $\beta_{\succ}(C)$ (possibly also removing objects dominated by o_i itself), otherwise o_i is discarded. At each point MPO maintains a “threshold point” $\underline{\mathbf{s}}$, whose q -th component, \underline{s}_q , is the lowest partial score seen so far under sorted access on list L_q . As soon as a point o_i is found such that \mathbf{s}_i dominates the threshold point $\underline{\mathbf{s}}$ the main loop of the algorithm ends (step 2). Before delivering the result (step 16), the condition at step 14 is checked. If implication is not guaranteed, then the main loop is executed again (but without changing anymore the threshold point $\underline{\mathbf{s}}$) so as to ensure that on each sorted list L_q there is no other object with a partial score equal to \underline{s}_q . We call this the “final cycle” of MPO.

Theorem 2 *The MPO algorithm correctly computes $\beta_{\succ}(C)$.*

Proof. Clearly $Result \subseteq \beta_{\succ}(C)$. To show containment in the other way, let o_j be an object of C that has not been seen under sorted access by the algorithm, and let o_i be the object that is found at step 2 to dominate the threshold point. Unless \mathbf{s}_j is coincident with the threshold point (i.e., $\mathbf{s}_j = \underline{\mathbf{s}}$), in which case we are obviously done, at least one partial score of o_j is strictly less than the corresponding threshold value. Thus, $\forall q : s_{j,q} \leq \underline{s}_q$ and $\exists q : s_{j,q} < \underline{s}_q$, which coincides with the definition of Skyline dominance. If, depending on the preference relation \succ at hand, $\mathbf{s}_i \succ \underline{\mathbf{s}}$ and $\underline{\mathbf{s}} \succ_{SL} \mathbf{s}_j$ imply $\mathbf{s}_i \succ \mathbf{s}_j$ (see step 14), we are guaranteed that $o_j \notin \beta_{\succ}(C)$. On the other hand, when the implication does not hold, the final cycle (step 15) guarantees that, for each sub-query Q_q , $s_{j,q} < \underline{s}_q$. Thus $\underline{\mathbf{s}} \succ \mathbf{s}_j$ follows since \succ is strictly monotone. Since \succ is a PO, $\mathbf{s}_i \succ \mathbf{s}_j$ follows by transitivity. \square

The need for the “final cycle” in the MPO algorithm may not be obvious. Indeed, MPO resembles middleware algorithms that were developed for the case when the integration is based on a scoring function and only the k highest scored objects are requested, *with ties arbitrarily broken*. This is not the case with MPO, which returns *all* the potential best matches, regardless of how many they are. It is easy to show that, for a generic \succ , omitting the final cycle could lead to missing some object in $\beta_{\succ}(C)$.

From a more pragmatical point of view, it has to be remarked that preference relations requiring the final cycle are more an exception than the rule.² Indeed, the following result shows that “natural” preference relations do not require the execution of the final cycle.

Lemma 5 *Let \succ be either \succ_{SL} , \succ_{RS} , or \succ_S , where S is any monotone scoring function. In all such cases MPO correctly computes $\beta_{\succ}(C)$ even if the final cycle is omitted.*

²We omit here the description of one such preference relation.

Sketch of proof. The proof amounts to showing that for each of the considered preference relations $\mathbf{s}_i \succ \underline{\mathbf{s}}$ and $\underline{\mathbf{s}} \succ_{SL} \mathbf{s}_j$ imply $\mathbf{s}_i \succ \mathbf{s}_j$. We omit the proof’s details. \square

4.2 The Incremental MPO Algorithm

If one wants to exert explicit control on the cardinality of the result, MPO is not the best alternative. Indeed, it is known that the size of the Skyline can become quite large, and grows fast with the number of dimensions when partial scores have a negative correlation [4]. On the other hand, when the size of $\beta_{\succ}(C)$ becomes too small it would be advisable to allow the user to retrieve also further “good” objects, even if they are not in $\beta_{\succ}(C)$.

To achieve both of the above goals we start by introducing a new operator, called BesTop, that combines the semantics of Best and Top- k operators. For its definition it is first useful to remind the “layered” version of the Best operator [15]:³

$$\beta_{\succ}^1(C) = \beta_{\succ}(C) \quad (6)$$

$$\beta_{\succ}^{\ell+1}(C) = \beta_{\succ}(C - \cup_{i=1}^{\ell} \beta_{\succ}^i(C)) \quad (7)$$

Thus, $\beta_{\succ}^{\ell}(C)$ retrieves the ℓ -th “layer” of (the partial order induced by \succ on) C . We are now ready to define the BesTop operator.

Definition 9 (BesTop operator) *Let $\ell(k) \geq 1$ be the smallest integer that satisfies the inequality $\sum_{i=1}^{\ell(k)} |\beta_{\succ}^i(C)| \geq k$.*

The BesTop operator, $\beta_{\succ}^{[k]}(C)$ ($k \geq 1$), retrieves k objects from C such that:

- $\beta_{\succ}^{[k]}(C)$ includes all the objects in the first $\ell(k) - 1$ layers of C (thus $\cup_{i=1}^{\ell(k)-1} \beta_{\succ}^i(C) \subseteq \beta_{\succ}^{[k]}(C)$);
- it further includes other $k - \sum_{i=1}^{\ell(k)-1} |\beta_{\succ}^i(C)|$ objects from the $\ell(k)$ -th layer of C .

A naïve approach to compute $\beta_{\succ}^{[k]}(C)$ would be to iterate algorithm MPO up to layer $\ell(k)$, and then to select all the objects in the first $\ell(k) - 1$ layers plus others from layer $\ell(k)$, so as to reach the desired result cardinality k .⁴ The major drawback of this approach is made evident through a simple example. Let $k = 1$, thus $\ell(1) = 1$ (the first layer is obviously enough). If we run MPO and wait until its completion we would miss the opportunity to stop as soon as we can conclude that *a single object* belongs to $\beta_{\succ}^1(C)$. In general, we see that no object of a layer ℓ can be returned by MPO before it is discovered that no further object belongs to layer ℓ , which might severely affect performance.

³An analogous extension has been proposed for the Winnow operator [6].

⁴Note that at layer $\ell(k)$ ties are arbitrarily broken, as it is customary for top k queries.

Algorithm iMPO (Input: query \mathcal{Q} , collection C , strictly monotone preference relation \succ , integer k)

```

(1) Set  $NoOfResults = 0$ ; Set  $ThisLayer = NextLayer = \emptyset$ ; Set  $\underline{s} = (1, \dots, 1)$ ;
(2) While ( $NoOfResults < k$ ):
(3)   While ( $\nexists (o_i, \mathbf{s}_i) \in ThisLayer$  such that  $\mathbf{s}_i \succ \underline{s} \wedge NoOfResults < k$ ):
(4)     For each sub-query  $Q_q$  ( $q = 1, \dots, m$ ) do:
(5)       Retrieve the next unseen object  $o_i$  from  $L_q$ ; /* sorted access */
(6)       Retrieve missing scores for the other sub-queries and obtain  $\mathbf{s}_i$ ; /* random accesses */
(7)       Set  $Dominated = false$ ;
(8)       While ( $\text{not}(Dominated) \wedge \exists (o_j, \mathbf{s}_j) \in ThisLayer$  unmatched with  $\mathbf{s}_i$ ):
(9)         Compare  $\mathbf{s}_i$  with  $\mathbf{s}_j$ :
          
$$\begin{cases} \mathbf{s}_i \succ \mathbf{s}_j & \text{move } (o_j, \mathbf{s}_j) \text{ from } ThisLayer \text{ to } NextLayer, \\ \mathbf{s}_i \sim \mathbf{s}_j & \text{do nothing,} \\ \mathbf{s}_j \succ \mathbf{s}_i & \text{set } Dominated = true \text{ and insert } (o_i, \mathbf{s}_i) \text{ in } NextLayer; \end{cases}$$

(10)        End While;
(11)        If  $\text{not}(Dominated)$  insert  $(o_i, \mathbf{s}_i)$  in  $ThisLayer$ ;
(12)        Let  $\underline{s}_q$  be the lowest score seen by sorted access on list  $L_q$ ; Set  $\underline{s} = (\underline{s}_1, \dots, \underline{s}_m)$ ;
(13)        Output all objects  $(o_i, \mathbf{s}_i) \in ThisLayer$  s.t.  $\underline{s} \not\succeq \mathbf{s}_i$  and update  $NoOfResults$ ;
(14)      End For;
(15)    End While;
(16)    If ( $NoOfResults < k$ ) then:
(17)      Set  $ThisLayer = NextLayer$ ; Set  $NextLayer = \emptyset$ ; /* starts to process the next layer */
(18)      For all  $o_i, o_j \in ThisLayer$  s.t.  $\mathbf{s}_i \succ \mathbf{s}_j$  move  $(o_j, \mathbf{s}_j)$  from  $ThisLayer$  to  $NextLayer$ ;
(19)      Output all objects  $(o_i, \mathbf{s}_i) \in ThisLayer$  s.t.  $\underline{s} \not\succeq \mathbf{s}_i$  and update  $NoOfResults$ ;
(20)    End If;
(21) End While.

```

Figure 4: The incremental MPO algorithm

Algorithm iMPO (incremental MPO), summarized in Figure 4, elegantly solves the above problem. To understand the logic of iMPO the following observation is useful.

Observation 2 *Let \succ be a strictly monotone PO preference relation. If $\underline{s} \not\succeq \mathbf{s}_i$ and $\forall q : s_{j,q} < \underline{s}_q$ (thus, $\underline{s} \succ \mathbf{s}_j$) then $\mathbf{s}_j \not\succeq \mathbf{s}_i$.*

Indeed, if $\mathbf{s}_j \succ \mathbf{s}_i$ then by transitivity it would follow that $\underline{s} \succ \mathbf{s}_i$, thus contradicting the hypothesis.

iMPO exploits the above observation as follows. Each time the threshold point \underline{s} changes, iMPO checks if some object o_i that has already been retrieved is *not* dominated by \underline{s} , i.e. $\underline{s} \not\succeq \mathbf{s}_i$ (step 13). If this is the case, as soon as we see on all the m lists L_q partial scores that are strictly lower than \underline{s}_q , we can conclude that no unseen object o_j can dominate o_i . This is sufficient to assert that o_i can be *immediately* delivered to the user. For this reason we call $\underline{s} \not\succeq \mathbf{s}_i$ the *delivery condition* of iMPO for object o_i .

Again, it has to be remarked that the need to wait for seeing partial scores strictly less than the threshold values is not necessary for relevant preference relations. In order to avoid unnecessary complications to the

algorithm description, in Figure 4 we omit to detail this “final cycle”.

The second major feature that distinguishes iMPO from MPO is the management of multiple layers of the partially ordered collection C . Rather than simply removing objects that are found to be dominated by some other object (as MPO does), iMPO keeps them in a *NextLayer* structure. Objects in such a structure are processed again upon completion of a layer, and before restarting to retrieve other objects via sorted access (steps 17 and 18).

4.3 Analysis of Algorithms

In this section we prove some basic facts that shed light on the behavior of MPO and iMPO algorithms.

The first simple, yet important, observation is that iMPO will always return an object o_i before MPO does so. This stems from the following basic result, in which we consider MPO extended so as to deal with multiple layers (rather than just the first one) of C . Having already introduced iMPO, this extension is straightforward, since it essentially amounts to dropping the incremental delivery condition from the logic of iMPO.

Lemma 6 *Let o_j be an object at layer ℓ of C , and let o_i be the object that allows MPO to terminate the elaboration of the ℓ -th layer (thus, $\mathbf{s}_i \succ \underline{\mathbf{s}}$). Let $\#SA(MPO)$ be the number of sorted accesses that MPO has executed up to this point, and $\#SA(iMPO)$ those executed by iMPO when it delivers object o_j . It is $\#SA(iMPO) \leq \#SA(MPO)$.*

Proof. We have $\mathbf{s}_i \succ \underline{\mathbf{s}}$ and, by hypothesis, $\mathbf{s}_i \sim \mathbf{s}_j$. From these we can conclude that $\underline{\mathbf{s}} \not\succeq \mathbf{s}_j$, which proves the result \square

For reference purpose, we now contrast our algorithms to the TA algorithm [11], a well-known algorithm that works for any monotone scoring function S . Here we do not consider variants of TA, such as TA-EP and Upper, aiming to reduce the number of random accesses [5].

The logic of TA is somewhat similar to that of MPO. Both algorithms, after performing a sorted access that retrieves an object o_i , get, through random access, the missing partial scores, so as to obtain \mathbf{s}_i . At this point TA computes $S(\mathbf{s}_i)$, the overall score of o_i , and the *threshold value*, $S(\underline{\mathbf{s}})$. If at least k points have been found such that $S(\mathbf{s}_i) \geq S(\underline{\mathbf{s}})$, TA stops and returns the k objects with the highest scores.

Although TA delivers objects only at its termination, it is not difficult to turn it into an incremental algorithm.⁵ For this it is sufficient to check, whenever the threshold value changes, if $S(\mathbf{s}_i) \geq S(\underline{\mathbf{s}})$ and then *immediately* return o_i to the user. We call this the delivery condition of TA.

⁵To the best of our knowledge this is new, in that no incremental version of TA and related algorithms has ever been proposed before.

A comparison between (the incremental version of) TA and iMPO without making any hypothesis on which scoring function and preference relation are respectively used is indeed hopeless, due to the wide range of possibilities that preclude any general conclusion to be drawn on the relative behavior of the two algorithms. Therefore, in the following we prove some basic facts that, although based on specific assumptions, indeed cover the most interesting cases.

We first show under which provable conditions iMPO, when equipped with Skyline preferences, delivers results faster than TA with *any* monotone scoring function.

Theorem 3 *Let $o_i \in C$ be an object at the ℓ -th layer ($\ell \geq 1$) of C when \succ_{SL} is used. Assume that TA, using the scoring function S , returns o_i after having executed $\#SA(TA)$ sorted accesses, and let $\#SA(iMPO)$ be the number of sorted accesses needed by iMPO to return the same object. If iMPO starts to process layer ℓ after a number of sorted accesses less than $\#SA(TA)$, then:*

1. *if TA returns o_i because $S(\mathbf{s}_i) > S(\underline{\mathbf{s}})$, then $\#SA(iMPO) \leq \#SA(TA)$;*
2. *if TA returns o_i because $S(\mathbf{s}_i) = S(\underline{\mathbf{s}})$, $\#SA(iMPO) > \#SA(TA)$ only if \mathbf{s}_i lies on the surface of the hyper-rectangle having the origin $\mathbf{0} = (0, \dots, 0)$ and $\underline{\mathbf{s}}$ as opposite vertices. In this case $\#SA(iMPO) - \#SA(TA)$ will linearly depend on the number of objects having the same score of o_i for the sub-query on which o_i has been retrieved via sorted access;*
3. *if the scoring function S is strictly monotone in each argument, that is, increasing one partial score and leaving the others unchanged leads to an increase in the overall score (this is the case, say, of avg) [11], it is guaranteed that $\#SA(iMPO) \leq \#SA(TA)$.*

Sketch of proof. The condition requiring that iMPO has already started to process layer ℓ before TA delivers o_i eliminates the possibility that o_i cannot be output since it does not belong to the current top layer.

- (1) Immediate, since $S(\mathbf{s}_i) > S(\underline{\mathbf{s}})$ implies $\underline{\mathbf{s}} \not\succeq_{SL} \mathbf{s}_i$;
- (2) Necessity follows from the definition of Skyline preferences. To prove the rest of the result, observe that if there are X objects with the same partial score of o_i in the sub-query where o_i has been retrieved via sorted access, $O(X)$ sorted accesses are enough to lower the threshold value on that list, after which $\underline{\mathbf{s}} \not\succeq_{SL} \mathbf{s}_i$;
- (3) In this case $S(\mathbf{s}_i) \geq S(\underline{\mathbf{s}})$ always implies $\underline{\mathbf{s}} \not\succeq_{SL} \mathbf{s}_i$. \square

Thus, when $S(\mathbf{s}_i) \geq S(\underline{\mathbf{s}})$ implies $\underline{\mathbf{s}} \not\succeq_{SL} \mathbf{s}_i$, and o_i is an object of the current top layer, we have that:

- if both TA and iMPO (equipped with Skyline preferences) are executed so as to deliver the same number of objects k , iMPO will run faster than TA;
- if TA and iMPO are allowed to execute the same number of steps, the result of TA will be included in that of iMPO.

Let us say that a collection C has the *uniqueness property* if on each sub-query Q_q no two objects get the same partial score, i.e., $\forall q, o_i, o_j \neq o_i : s_{i,q} \neq s_{j,q}$ [11]. When this is the case the delivery conditions of TA and iMPO can be relaxed, by using as threshold point \underline{s}_ϵ in place of \underline{s} , where $\forall q : \underline{s}_{\epsilon,q} = \underline{s}_q - \epsilon$ ($\epsilon > 0$).

Theorem 4 *Assume that the collection C satisfies the uniqueness property. Under the same hypotheses of Theorem 3, $\#SA(iMPO) \leq \#SA(TA)$.*

Proof. Omitted \square

5 Experimental Analysis

We evaluated the performance of MPO and iMPO algorithms using a real-world image collection consisting of about 10,000 color images.⁶ Although this data set is not particularly large, we chose it for two reasons:

- Since each image comes with a manually assigned semantic classification into one of 7 classes, this allows us to evaluate effectiveness (quality) of results, which would not be possible without an objective “ground truth”. To this end, given a query image, any image in the same class of the query is considered relevant, whereas all other images are considered not relevant, regardless of their actual low-level feature contents. Note that classes are just used for evaluation purposes and not during the retrieval phase (i.e., algorithms know nothing about the class of an image). This leads to hard-to-solve conceptual queries, since within a same class feature values may wildly vary. However, this is exactly the hard task that a MM retrieval system has to face!
- Further, since in this paper we are not dealing with issues related to the evaluation of sub-queries, the actual size of the data set is not particularly relevant in assessing performance. Indeed, although our system uses indexes to efficiently evaluate sub-queries, relative figures are not shown here (any method able to return ranked lists would serve the purpose).

⁶IMSI MasterPhotos 50,000: <http://www.imsisoft.com>.

Each image, using wavelet transform, was automatically segmented into a set of homogeneous *regions*, based on the proximity of wavelet coefficients, which convey information about color and texture features. Each region corresponds to a cluster of pixels and is represented through a 37-dimensional feature vector.⁷ On average, 4 regions were obtained from each image. The same procedure is adopted when an image query Q is submitted. If m is the number of regions extracted from Q , each of the m regions becomes a sub-query. Partial scores for a given query region Q_q are obtained by using a distance function based on the *Bhattacharyya metric* [2], which is commonly used to compare ellipsoids.

We implemented MPO (extended so as to answer “first ℓ layers” queries), iMPO, and TA algorithms in C++ and run all the experiments on a 1.6 GHz Pentium machine. All the results we present are averaged over a sample of 100 randomly-chosen query images. Our major interest is to understand how our algorithms (iMPO in particular) perform in terms of efficiency and effectiveness:

Efficiency. The metrics we use are the number of sorted accesses, $\#SA$, and the number of random accesses, $\#RA$, executed by the algorithms. This ensures a fair, system-independent, comparison. Note that actual execution times are indeed expected to vary in a significant way depending on the relative cost of sorted and random accesses, the nature of the underlying system(s) evaluating sub-queries (e.g., Web-based or not), the available access methods, etc. To avoid distracting the reader with too many parameters and variables we opted for clean, easy to understand, metrics.

Effectiveness. We consider the classical *precision* (P) metric, that is, the percentage of relevant images found by a query, as a measure of how good the results of an algorithm (when equipped with a specific preference relation or scoring function) are. In addition we also analyze the extent to which relevant images are representative of the query class, that is, how well they fit the actual distribution of all images in the query class. This allows a finer assessment of the quality of results that P alone cannot provide.

For MPO and iMPO algorithms we consider both Skyline (SL) and Region-prioritized Skyline (RS) preferences, whereas for the TA algorithm we consider min and avg scoring functions. We use the notation MPO(SL) to mean algorithm MPO using SL preferences, and so on. In order to generate RS preferences we proceed as follows (see also Definition 8 in Section 3).

⁷In detail: 12 dimensions are used for cluster’s centroids (3 color channels \times 4 frequency sub-bands), 24 coefficients store the 3×3 (symmetric) covariance matrices of the 4 sub-bands, and 1 coefficient represents the cluster size.

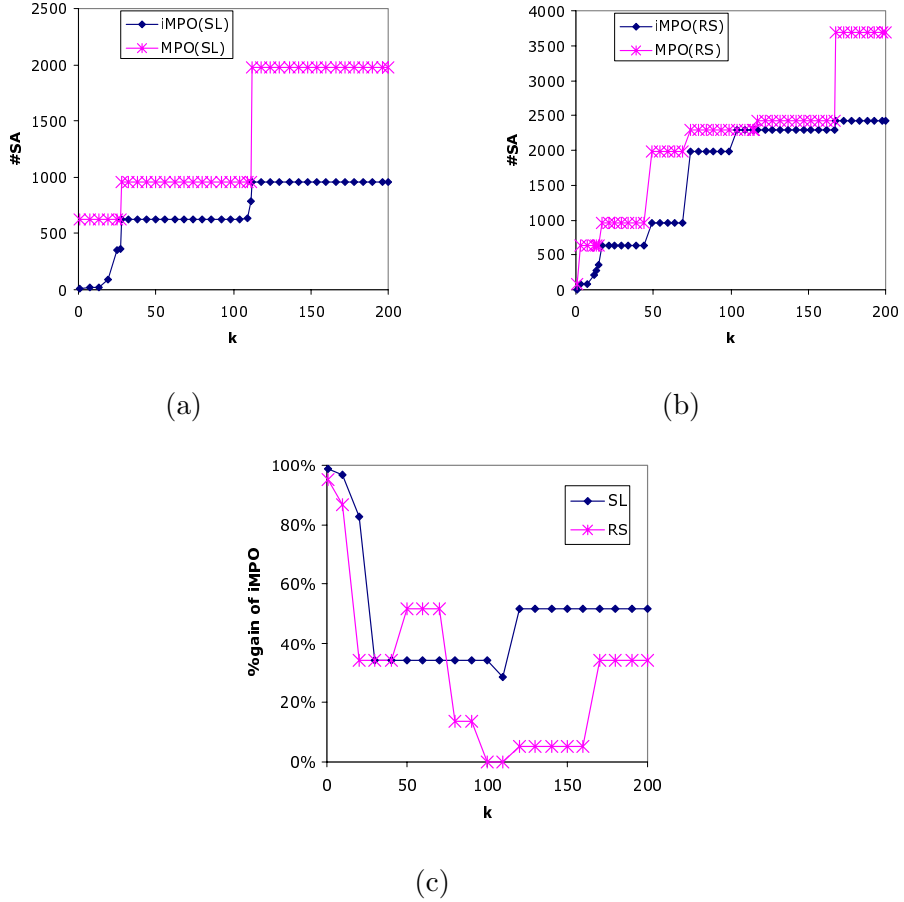


Figure 5: Sorted accesses of MPO and iMPO algorithms for a specific query: (a) Skyline (SL) preferences; (b) Region-prioritized Skyline (RS) preferences. In (c) the percentage gain of iMPO over MPO is shown. The abscissa reports the no. of retrieved objects k

On each of the m coordinates of the answer space A , we set a “soft threshold” θ_q ($0 < \theta_q < 1$) and assign a 0 bit to the “below-threshold” interval $[0, \theta_q)$ and a 1 bit to the “above-threshold” interval $[\theta_q, 1]$. This leads to 2^m regions, each univocally represented by an m -bit binary code. Given regions A_i and A_j , the preference relation for such regions is

$$A_i \succ_{Reg} A_j \Leftrightarrow code(A_i) \wedge code(A_j) = code(A_j)$$

where bitwise AND is used and $code(A_i)$ is the binary code of region A_i . For instance, when $m = 4$, this says that the region with code 1011 dominates the region with code 1000, whereas it is indifferent to region 0100. Since \succ_{Reg} defines a Boolean lattice over regions (with region 11...1 being the

best region and $00\dots 0$ the worst one) it is easy to show that Lemma 4 is satisfied, thus \succ_{RS} is a strictly monotone PO.

Although we experimented with several combinations of soft threshold values, here we just report results for the case $\theta_q = 0.4 \forall q$.

5.1 Experimental Results

Experiment 1: The aim of our first experiment is to measure the relative efficiency of iMPO versus MPO. Quality of results is not a concern here, since both MPO and iMPO return the same objects, although at different times.

Our results confirm that iMPO consistently outperforms MPO. In Figure 5 we show efficiency results for a specific query (using either SL or RS preferences), results for other queries being similar. Here we just show the number of sorted accesses. MPO, by its nature, delivers objects in bursts, each burst corresponding to the termination of one layer. For instance, in Figure 5 (a) MPO needs 628 sorted accesses to return all the 27 images in the 1st layer of the Skyline, 956 to complete the 2nd layer, and so on. A somewhat bursty behavior is also observed with iMPO, starting from the 2nd layer. To explain this, consider that even if *all* objects in the current layer have been output, iMPO still needs to wait that the test $\mathbf{s}_i \succ \underline{\mathbf{s}}$ succeeds before moving to the next layer. This “waiting time” leads to accumulate objects in the *NextLayer*, most of which are subsequently delivered as soon as the test succeeds. Nonetheless, Figure 5 (c) shows that the gain in efficiency of iMPO over MPO is remarkable; therefore in the sequel we do not consider MPO anymore.

Experiment 2: In this second series of experiments our objective is to compare iMPO and TA in terms of quality of results.

Figure 6 (a) shows precision values versus k . It can be seen that SL, and RS in particular, preferences attain precision levels comparable to that of avg, whereas min has a definitely poor behavior. Figure 6 (b) shows similar results, now plotting graphs with variable k_{rel} , the number of relevant objects retrieved.

Clearly, similar precision values do not imply similar results. Indeed, on the average and for any value of k , iMPO and TA share less than 50% of the relevant objects retrieved. The distribution of relevant objects over the answer space can play a fundamental role for the implementation of effective relevance feedback mechanisms, as well as for giving to the user an accurate view of the actual database contents. Therefore, we analyzed how much the different approaches lead to an “unbiased” result. For this we use an information-theoretic measure, related to the cross-entropy of two distributions, known as the *Kullback-Leibler (KL) divergence* [9]. Given a reference distribution f and a test one g , the KL divergence of g with respect

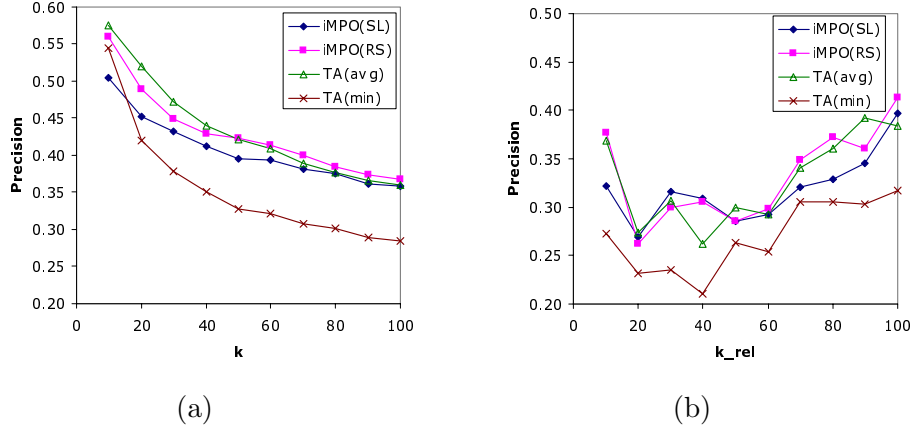


Figure 6: Precision vs. no. of retrieved objects (a) and no. of relevant retrieved objects (b)

to f is defined as

$$KL(g; f) = \int_x f(x) \ln \left(\frac{f(x)}{g(x)} \right) dx.$$

$KL(g; f) \geq 0$, with 0 attained only if $g = f$. Thus, $KL(g_1; f) < KL(g_2; f)$ denotes that g_1 fits f better than g_2 .

In our case, we take f to be the distance distribution of *all* the relevant objects for a query Q , and the g_i 's be the (approximate) distance distributions of the relevant objects returned in the first $k = 100$ results by TA and iMPO algorithms. All distances are measured over the answer space, by computing the Euclidean distance between the representative points of relevant images.

Figure 7 (a) shows the (averaged over all query images) actual distribution of the whole data set (label dataset in the figure) and those of TA and iMPO. Figure 7 (b) does the same but just for queries of a specific class (“TreeLeaves”). Table 2 synthesizes everything using KL divergence values.

	iMPO(SL)	iMPO(RS)	TA(avg)	TA(min)
Global	0.032	0.006	0.297	0.550
TreeLeaves	0.050	0.031	0.312	0.680

Table 2: Kullback-Leibler (KL) divergence values for the distributions in Figure 7

It is evident that iMPO with RS preferences, besides leading to precision values comparable to those of avg, has a remarkably better capability to

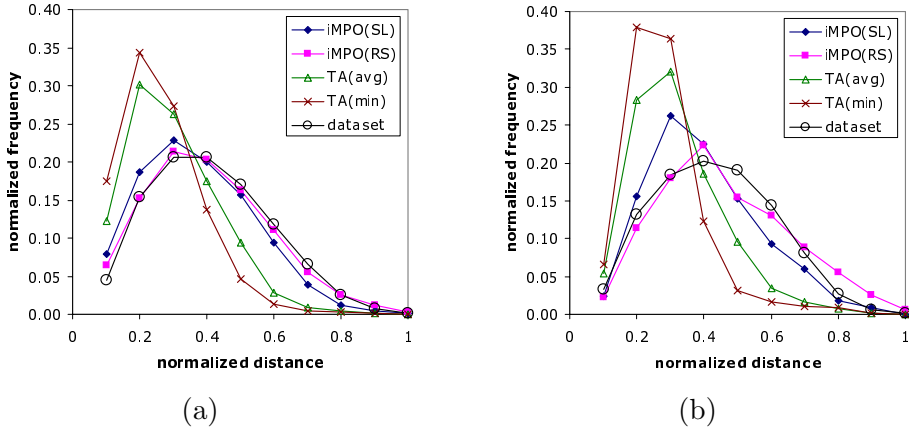


Figure 7: Global (a) and relative to the “TreeLeaves” class (b) distance distributions of relevant objects

reflect the actual distribution of relevant objects. Skyline preferences are, to this end, slightly worse, even if divergence values are still one order of magnitude better than those of avg and min.

Experiment 3: In the third experiment our objective is to analyze the efficiency of iMPO and TA in answering top k queries.

Figure 8 shows how many sorted accesses and random accesses are needed by the analyzed algorithms to deliver k objects. In this case iMPO(SL) is undoubtedly the winner, whereas iMPO(RS) is slower, however reaching a performance level that is always better than that of both TA(avg) and TA(min). Thus, not only qualitative preferences are better than scoring functions in delivering high-quality results, they are also faster.

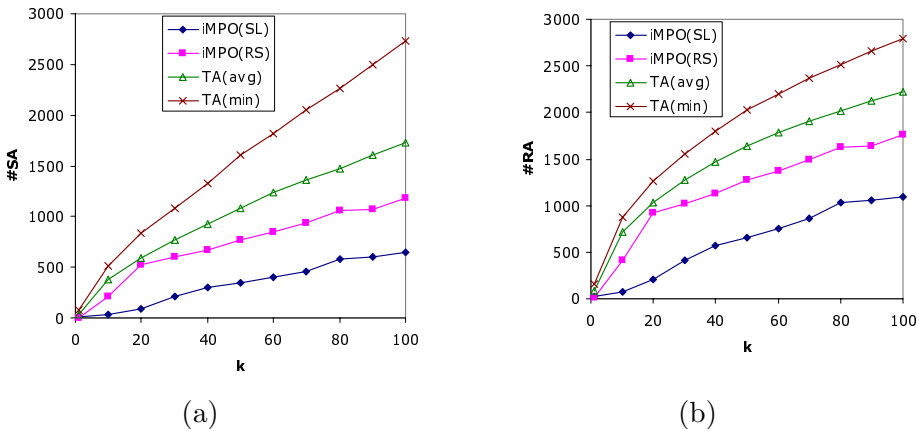


Figure 8: Sorted accesses (a) and random accesses (b) vs. no. of retrieved objects (k)

The reason why iMPO(SL) outperforms iMPO(RS) depends on the different delivery conditions used by the two methods. Indeed, considering how RS preferences are defined, it can be shown that $(\underline{s} \not\prec_{RS} \mathbf{s}_i) \implies (\underline{s} \not\prec_{SL} \mathbf{s}_i)$, thus the delivery condition of iMPO(RS) is always stronger than that of iMPO(SL). In particular, for iMPO(RS) it is likely the case that the threshold point \underline{s} belongs to a region that dominates the region of \mathbf{s}_i , which is sufficient to have $\underline{s} \succ_{RS} \mathbf{s}_i$.

Finally we present graphs where efficiency and quality of results can be observed together. Figure 9 shows how much we have to pay (in terms of sorted and random accesses, respectively) for each relevant object we retrieve. The graphs confirm previous results, in particular the superior performance of qualitative preferences, and also show that, starting with $k_{rel} \geq 50$, the reduced efficiency of iMPO(RS) with respect to iMPO(SL) is compensated by its superior effectiveness, which leads to a “per relevant object” cost of iMPO(RS) almost equal to that of iMPO(SL).

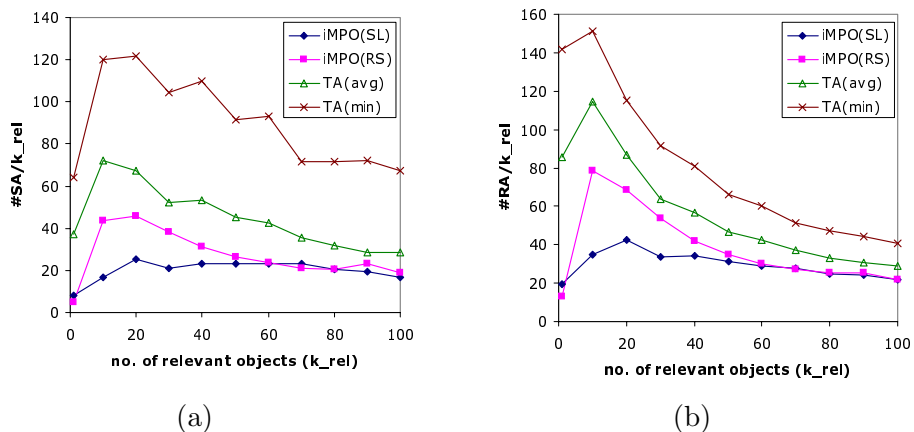


Figure 9: Sorted accesses (a) and random accesses (b) vs no. of relevant retrieved objects (k_{rel})

6 Conclusions

In this paper we have analyzed how qualitative preferences can impact multimedia query processing and quality of results. We have introduced two algorithms for efficiently performing integration of sub-queries and demonstrated their efficiency on a real-world image database. We have also introduced the class of Region-prioritized Skyline (RS) preferences, which provide the best quality of results and enjoy a cost per relevant object comparable to that of Skyline preferences.

Our work opens new interesting lines of research. First, it would be interesting to apply qualitative preferences to other challenging tasks, such

as classification of multimedia objects, for which scoring functions have been considered the only viable alternative. Second, qualitative preferences could also be profitably used for the evaluation of sub-queries, thus generalizing the common approach requiring a distance metrics to compare objects' features. The exploitation of qualitative preference for query refinement is a further interesting research direction. We also plan to extend our algorithms to Web-based scenarios in which random accesses might not be possible for some of the sub-queries and some underlying source might not provide partial scores for the returned objects.

References

- [1] I. Bartolini, P. Ciaccia, and F. Waas. FeedbackBypass: A New Approach to Interactive Similarity Query Processing. In *Proceedings of the 27th International Conference on Very Large Data Bases (VLDB 2001)*, pages 201–210, Rome, Italy, Sept. 2001.
- [2] M. Basseville. Distance Measure for Signal Processing and Pattern Recognition. *European Journal of Signal Processing*, 18(4):349–369, Dec. 1989.
- [3] K. Böhm, M. Mlivonic, H.-J. Schek, and R. Weber. Fast Evaluation Techniques for Complex Similarity Queries. In *Proceedings of the 27th International Conference on Very Large Data Bases (VLDB 2001)*, pages 211–220, Rome, Italy, Sept. 2001.
- [4] S. Borzsonyi, D. Kossmann, and K. Stocker. The Skyline Operator. In *Proceedings of the 17th International Conference on Data Engineering*, pages 421–430, Heidelberg, Germany, Apr. 2001.
- [5] N. Bruno, L. Gravano, and A. Marian. Evaluating Top- k Queries over Web-Accessible Databases. In *Proceedings of the 18th International Conference on Data Engineering (ICDE'02)*, pages 369–382, San Jose, California, USA, Feb. 2002.
- [6] J. Chomicki. Querying with Intrinsic Preferences. In *Proceedings of the 8th International Conference on Extending Database Technology (EDBT 2002)*, pages 34–51, Prague, Czech Republic, Mar. 2002.
- [7] J. Chomicki. Preference Formulas in Relational Queries. *ACM Transactions on Database Systems (TODS)*, 28(4):1–39, 2003.
- [8] P. Ciaccia, M. Patella, and P. Zezula. Processing Complex Similarity Queries with Distance-based Access Methods. In *Proceedings of the 6th International Conference on Extending Database Technology (EDBT'98)*, pages 9–23, Valencia, Spain, Mar. 1998.

- [9] T. M. Cover and J. A. Thomas. *Elements of Information Theory*. Wiley, 1991.
- [10] R. Fagin. Combining Fuzzy Information from Multiple Systems. In *Proceedings of the Fifteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 216–226, Montreal, Canada, June 1996.
- [11] R. Fagin, A. Lotem, and M. Naor. Optimal Aggregation Algorithms for Middleware. In *Proceedings of the Twentieth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 216–226, Santa Barbara, California, USA, May 2001.
- [12] P. C. Fishburn. Preference Structures and Their Numerical Representations. *Theoretical Computer Science*, 217(2):359–383, 1999.
- [13] M. Ortega, Y. Rui, K. Chakrabarti, K. Porkaew, S. Mehrotra, and T. S. Huang. Supporting Ranked Boolean Similarity Queries in MARS. *IEEE Transactions on Knowledge and Data Engineering*, 10(6):905–925, 1998.
- [14] Y. Rui, T. S. Huang, M. Ortega, and S. Mehrotra. Relevance Feedback: A Power Tool for Interactive Content-Based Image Retrieval. *IEEE Transaction on Circuits and Systems for Video Technology*, 8(5):644–655, Sept. 1998.
- [15] R. Torlone and P. Ciaccia. Which Are My Preferred Items? In *AH2002 Workshop on Recommendation and Personalization in eCommerce (RPeC02)*, pages 1–9, Malaga, Spain, May 2002.