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
Dept. of Computer Science
NJ Inst. of Technology
Newark, NJ, USA
oria@cis.njit.edu

M. Tamer Özsu
School of Computer Science
University of Waterloo
Waterloo, ON, Canada
tozsu@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 is applied to return only the* top k *highest scored objects.*

In this paper we consider a more general approach, based on 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 gives a larger flexibility in shaping what the user is looking for. For the purpose of efficient evaluation, we propose two integration algorithms, both able to work with* any *(monotone) partial order: 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. Our analysis demonstrates that using qualitative preferences pays off. In particular, using Skyline and the new* Region-prioritized Skyline *preferences for queries on a real image database, we show that iMPO yields results whose quality is comparable to that obtainable from algorithms using scoring functions. However, iMPO performs faster, saving up to about 70% database accesses.*

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), 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 [9], an integration of partial results is needed. Relevant examples of "middleware" algorithms that address this problem are $\mathcal{A}_0$ [9], TA [11], and MEDRANK [10]. Their common rationale 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 (possibly weighted) *scoring function*, like $\min$, $\max$, avg and median, 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. The choice of the scoring function and of the weights can be both critical factors for the determination of the final result, that 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 due to their 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]. A third limit of scoring

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

functions is that they provide a poor "coverage" of the best available alternatives. 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 [14] 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*, able to overcome the above drawbacks. Qualitative preferences, which have been recently used in the context of relational databases [5], 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 ($o_i \not\succ o_j$). This approach includes scoring functions as a special case, since, given a scoring function that associates the overall score s_i to the object o_i, 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 $o_i \not\succ o_i$ (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 *Winnow* [5] 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 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 3) 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 and give the user additional control over the query results. We demonstrate, through experiments on a real-world image database, that the quality of the results returned by iMPO and TA [11], 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.

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.

2 The Integration Problem

Consider a collection C of objects and a *complex* MM query of the form $\mathcal{Q} = (Q_1, Q_2, \ldots, Q_m)$ 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 ($q = 1, \ldots, m$) 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.

Given m lists L_q resulting from the evaluation of the m sub-queries of the complex query Q, the integration problem is to return as a result the "best" (in some sense to be defined) objects from the m lists. We model access to the database in the usual way [9, 11], assuming that objects can be retrieved through one of the two distinct modalities: The *sorted* access modality in which one retrieves from a list L_q the next unseen object on that list, say o_i, together

with its partial score, $s_{i,q}$; the *random* access modality in which, 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 $\mathbf{s_i} = (s_{i,1}, \ldots, s_{i,m})$, whose coordinate values are its m partial scores for the m sub-queries.

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 recent 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 [7] 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 $\mathcal{A}_0$ [9] and TA [11], to name a few, just rely on the sorted and random access modalities and on the hypothesis of *monotonicity* of the scoring function.

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

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 $\mathbf{1} = (1, \ldots, 1)$, 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. We observe that 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*: 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.

It is clear that any top k query based on a monotone scoring function can only return points that are potential j best matches, $j \leq k$. 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 [14]. 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.

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 2 *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* ($x \not\succ x$) 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 $x_2 \not\succ x_1$, 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 3 (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 $\mathbf{s_j} \not\succ \mathbf{s_i}$. 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 $\mathbf{s_j} \not\succ \mathbf{s_i}$ and $\mathbf{s_i} \not\succ \mathbf{s_j}$, 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 4 (Skyline preferences) *The Skyline preference relation $\succ_{SL}$ over $A = [0,1]^m$ is defined as follows:*

$$o_i \succ_{SL} o_j \Leftrightarrow (\forall q : s_{j,q} \leq s_{i,q}) \wedge (\exists q : s_{j,q} < s_{i,q}) \quad (1)$$

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.*

Figure 1 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 $\mathcal{Q}$ and the potential best matches (i.e., Skyline) of $\mathcal{Q}$. Note that most of them belong to the same semantic class, "Birds", of $\mathcal{Q}$ (more details on this point are given in Section 5) and that they are quite spread over the answer space. Results of this kind are incomparable to those obtainable from scoring functions, like $\min$ and avg.

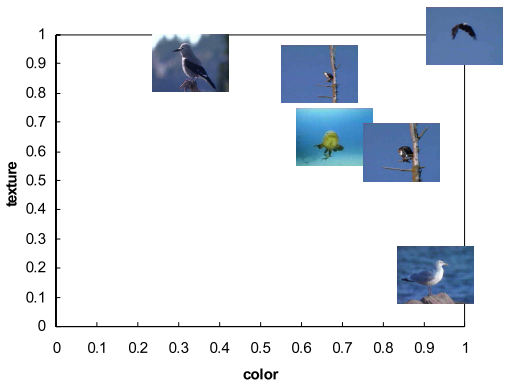


Figure 1. The Skyline of the "eagle" image

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 $A_1, \ldots, A_P$, and that a preference relation is also expressed over such regions.

Definition 5 (RS preferences) *Let $Y = \{A_1, \ldots, A_P\}$ 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:*

$$o_i \succ_{RS} o_j \quad \Leftrightarrow \quad (Reg(\mathbf{s_i}) \succ_{Reg} Reg(\mathbf{s_j})) \vee ((Reg(\mathbf{s_i}) = Reg(\mathbf{s_j})) \wedge (\mathbf{s_i} \succ_{SL} \mathbf{s_j})) \quad (2)$$

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.

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 1 *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.
(only if) Immediate from Definitions 3 and 5.
(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. □

It should be clear that if in Definition 5 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 [5] operators. The Best operator is defined as:

$$\beta_{\succ}(C) = \{o \in C \mid \nexists o' \in C, o' \succ o\} \qquad (3)$$

In Figure 2 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 1 *The MPO algorithm correctly computes* $\beta_{\succ}(C)$.

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

(1) Set $Result = \emptyset$; Set $\underline{\mathbf{s}} = (1, \ldots, 1)$; /* $\underline{\mathbf{s}}$ is the threshold point */
(2) While ($\nexists (o_i, \mathbf{s_i}) \in Result$ such that $\mathbf{s_i} \succ \underline{\mathbf{s}}$):
(3) For each sub-query Q_q ($q = 1, \ldots, m$) 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 (not($Dominated$) $\wedge\ \exists\ (o_j, \mathbf{s_j}) \in Result$ unmatched with $\mathbf{s_i}$):
(8) Compare $\mathbf{s_i}$ with $\mathbf{s_j}$: $\begin{cases} \mathbf{s_i} \succ \mathbf{s_j} & \text{remove } (o_j, \mathbf{s_j}) \text{ from } Result, \\ \mathbf{s_i} \sim \mathbf{s_j} & \text{do nothing,} \\ \mathbf{s_j} \succ \mathbf{s_i} & \text{set } Dominated \text{ = true;} \end{cases}$
(9) End While;
(10) If 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 $\underline{\mathbf{s}} = (\underline{s_1}, \ldots, \underline{s_m})$;
(12) End For;
(13) End While;
(14) If ($\mathbf{s_i} \succ \underline{\mathbf{s}} \wedge \underline{\mathbf{s}} \succ_{SL} \mathbf{s_j}$) 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 2. The MPO algorithm

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. □

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.[1] Indeed, the following result shows that "natural" preference relations do not require the execution of the final cycle.

[1] For lack of space we omit here the description of one such preference relation.

Lemma 2 *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.*

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. □

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]:[2]

$$\beta^1_\succ(C) = \beta_\succ(C) \quad (4)$$

$$\beta^{\ell+1}_\succ(C) = \beta_\succ(C - \cup_{i=1}^{\ell}\beta^i_\succ(C)) \quad (5)$$

Thus, $\beta^\ell_\succ(C)$ retrieves the ℓ-th "layer" of (the partial order induced by $\succ$ on) C. We are now ready to define the BesTop operator.

Definition 6 (BesTop operator) *Let $\ell(k) \geq 1$ be the smallest integer that satisfies the inequality $\sum_{i=1}^{\ell(k)} \mid \beta^i_\succ(C) \mid \geq k$.*

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

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

A naïve approach to compute $\beta^{[k]}_\succ(C)$ 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.[3] 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 $\beta^1_\succ(C)$. 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.

[2] An analogous extension has been proposed for the Winnow operator [5].

[3] Note 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 $\underline{\mathbf{s}} = (1, \ldots, 1)$;
(2) While ($NoOfResults < k$):
(3) While ($\nexists (o_i, \mathbf{s_i}) \in ThisLayer$ such that $\mathbf{s_i} \succ \underline{\mathbf{s}} \wedge NoOfResults < k$):
(4) For each sub-query Q_q ($q = 1, \ldots, m$) 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 (not($Dominated$) $\wedge \exists (o_j, \mathbf{s_j}) \in ThisLayer$ unmatched with $\mathbf{s_i}$):
(9) Compare $\mathbf{s_i}$ with $\mathbf{s_j}$:
- $\mathbf{s_i} \succ \mathbf{s_j}$: move $(o_j, \mathbf{s_j})$ from $ThisLayer$ to $NextLayer$,
- $\mathbf{s_i} \sim \mathbf{s_j}$: do nothing,
- $\mathbf{s_j} \succ \mathbf{s_i}$: set $Dominated$ = true and insert $(o_i, \mathbf{s_i})$ in $NextLayer$;

(10) End While;
(11) If 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 $\underline{\mathbf{s}} = (\underline{s_1}, \ldots, \underline{s_m})$;
(13) Output all objects $(o_i, \mathbf{s_i}) \in ThisLayer$ s.t. $\underline{\mathbf{s}} \not\succ \mathbf{s_i}$ 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. $\underline{\mathbf{s}} \not\succ \mathbf{s_i}$ and update $NoOfResults$;
(20) End If;
(21) End While.

Figure 3. The incremental MPO algorithm

Algorithm iMPO (incremental MPO), summarized in Figure 3, elegantly solves the above problem. To understand the logic of iMPO the following observation is useful.

Observation 1 *Let $\succ$ be a strictly monotone PO preference relation. If $\underline{\mathbf{s}} \not\succ \mathbf{s_i}$ and $\forall q : s_{j,q} < \underline{s_q}$ (thus, $\underline{\mathbf{s}} \succ \mathbf{s_j}$) then $\mathbf{s_j} \not\succ \mathbf{s_i}$.*

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

iMPO exploits the above observation as follows. Each time the threshold point $\underline{\mathbf{s}}$ changes, iMPO checks if some object o_i that has already been retrieved is *not* dominated by $\underline{\mathbf{s}}$, i.e, $\underline{\mathbf{s}} \not\succ \mathbf{s_i}$ (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 $\underline{\mathbf{s}} \not\succ \mathbf{s_i}$ 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 3 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).

5 Experimental Analysis

For reference purpose, we contrast MPO and iMPO algorithms to the TA algorithm [11], a well-known algorithm that works for any monotone scoring function S. 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.[4] 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.

We evaluated the performance of the above algorithms using a real-world image collection consisting of about 10,000 color images. 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. 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).

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.[5] 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 which would consequently come into play we opted for clean, easy to understand, metrics.

Effectiveness. As possible measures of how good the results of an algorithm (when equipped with a specific preference relation or scoring function) are, we consider the classical *precision* (P) metric (i.e., the percentage of relevant images found by a query) and 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.

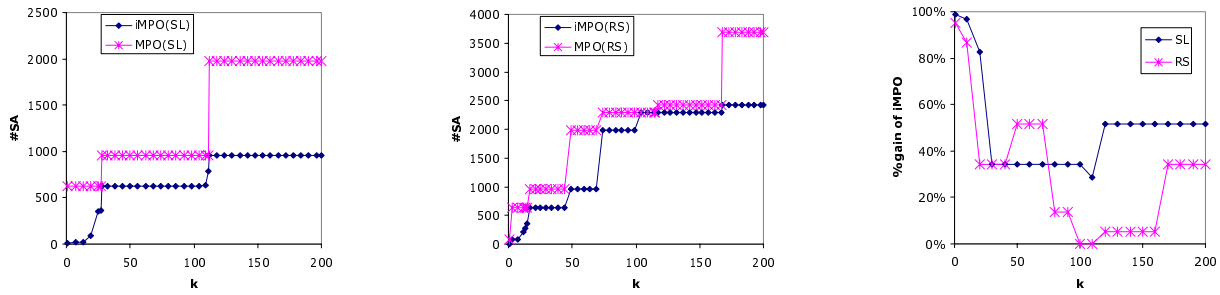
For MPO and iMPO algorithms we consider both Skyline (SL) and Region-prioritized Skyline (RS) preferences, whereas for the TA algorithm we consider $\min$ and avg scoring functions. We use the notation MPO(SL) to mean algorithm MPO using SL preferences, and so on. In order to generate RS preferences we proceed as follows (see also Definition 5 in Section 3). 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

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

[5] In 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.



(a) (b) (c)

Figure 4. 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

$\succ_{Reg}$ defines a Boolean lattice over regions (with region $11\ldots1$ being the best region and $00\ldots0$ the worst one) it is easy to show that Lemma 1 is satisfied, thus $\succ_{RS}$ is a strictly monotone PO.

Although we experimented with several combinations of soft threshold values, for lack of space 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 4 we show efficiency results for a specific query (using either SL or RS preferences), results for other queries being similar. For lack of space 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 4 (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 4 (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 goal is to compare iMPO and TA in terms of quality of results.

Figure 5 (a) shows precision values versus k_{rel}, the number of relevant objects retrieved. 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.

Clearly, similar precision values do not imply similar results. Indeed, we experimentally found that, on the average and for any value of k, iMPO and TA share less than 50% of the relevant objects retrieved. Motivated by this observation, we were led to investigate how to precisely characterize such difference of results. To this end, we considered the distribution of relevant objects over the answer space. By comparing the distributions of the relevant objects returned by iMPO (considering both SL and RS) and TA (using avg and min), it is possible to establish which one better fits the distribution of all relevant images in the query class (thus, which one better represents the actual class contents).

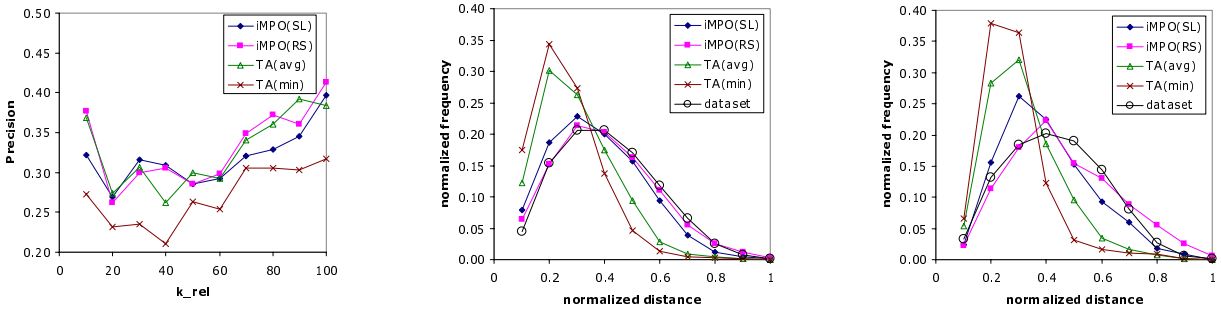
To properly compare the distributions of relevant results, we use an information-theoretic measure, related to the cross-entropy of two distributions, known as the *Kullback-Leibler (KL) divergence* [8]. Given a reference distribution f and a test one g, the KL divergence of g with respect to f is defined 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 5 (b) shows the (averaged over all query images) actual distribution of the whole data set (label dataset in



(a) (b) (c)

Figure 5. Effectiveness comparison between iMPO and TA: (a) Precision vs no. of relevant retrieved objects; global (b) and relative to the "TreeLeaves" class (c) distance distributions of relevant objects

the figure) and those of TA and iMPO. Figure 5 (c) does the same but just for queries of a specific class ("TreeLeaves"). Table 1 synthesizes everything using KL divergence values.

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

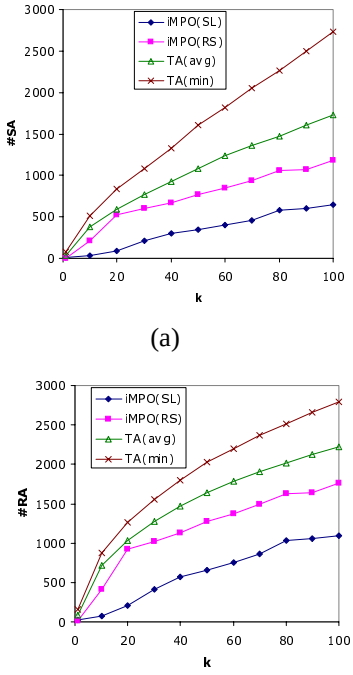
Table 1. Kullback-Leibler (KL) divergence values for the distributions in Figure 5 (b) and (c)

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

Even if this is outside the scope of the paper, we remark that an important advantage derived from having a small value of KL is related to the implementation of effective relevance feedback mechanisms [14]. Common to all these methods is the idea to exploit the user feedback (given on the query outcome) in order to refine the initial query. Thus, giving to the user a more accurate "overall view" of the content of the query class, it is possible to cut down the number of user-system interactions needed to lead to acceptable results.

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

Figure 6 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, saving up to about 70% and 80% database accesses against TA(avg) and TA(min), respectively. Efficiency of iMPO(RS) is slightly poorer, however reaching a performance level that is always better than that of both TA(avg) and TA(min) (35% and 60% speed-up, respectively).



(a)

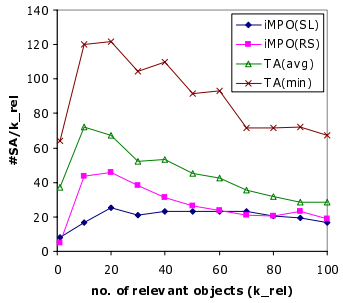
(b)

Figure 6. Sorted accesses (a) and random accesses (b) vs no. of retrieved objects (k)

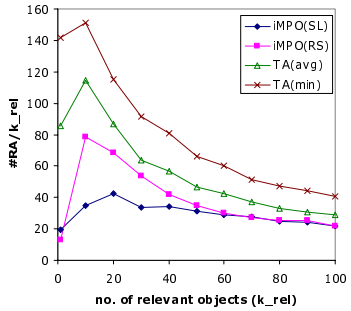
The reason iMPO(SL) outperforms iMPO(RS) is in the two different delivery conditions used by the two methods. Indeed, considering how RS preferences are defined, it can be shown that $(\underline{\mathbf{s}} \not\succ_{RS} \mathbf{s_i}) \implies (\underline{\mathbf{s}} \not\succ_{SL} \mathbf{s_i})$, 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 $\underline{\mathbf{s}}$ belongs to a region that dominates the region of $\mathbf{s_i}$, which is sufficient to have $\underline{\mathbf{s}} \succ_{RS} \mathbf{s_i}$.

Finally we present graphs where efficiency and quality of results can be observed together. Figure 7 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).



(a)



(b)

Figure 7. Sorted accesses (a) and random accesses (b) vs no. of relevant retrieved objects (k_{rel})

6 Conclusions

In this paper, we have analyzed the impact of qualitative preferences on multimedia query processing. 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.

References

[1] I. Bartolini, P. Ciaccia, and F. Waas. FeedbackBypass: A New Approach to Interactive Similarity Query Processing. *VLDB 2001*, pages 201–210, 2001.

[2] M. Basseville. Distance Measures for Signal Processing and Pattern Recognition. *European Journal of Signal Processing*, 18(4):349–369, Dec. 1989.

[3] K. Böhm, M. Mlivoncic, H.-J. Schek, and R. Weber. Fast Evaluation Techniques for Complex Similarity Queries. *VLDB 2001*, pages 211–220, 2001.

[4] S. Borzsonyi, D. Kossmann, and K. Stocker. The Skyline Operator. *ICDE 2001*, pages 421–430, 2001.

[5] J. Chomicki. Querying with Intrinsic Preferences. *EDBT 2002*, pages 34–51, 2002.

[6] J. Chomicki. Preference Formulas in Relational Queries. *ACM TODS*, 28(4):1–39, 2003.

[7] P. Ciaccia, M. Patella, and P. Zezula. Processing Complex Similarity Queries with Distance-based Access Methods. *EDBT 1998*, pages 9–23, 1998.

[8] T. M. Cover and J. A. Thomas. *Elements of Information Theory*. Wiley, 1991.

[9] R. Fagin. Combining Fuzzy Information from Multiple Systems. *PODS 1996*, pages 216–226, 1996.

[10] R. Fagin, R. Kumar, and D. Sivakumar. Efficient Similarity Search and Classification via Rank Aggregation. *SIGMOD 2003*, pages 301–312, 2003.

[11] R. Fagin, A. Lotem, and M. Naor. Optimal Aggregation Algorithms for Middleware. *PODS 2001*, pages 216–226, 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 TKDE*, 10(6):905–925, 1998.

[14] S. M. Kaushik Chakrabarti, Michael Ortega and K. Porkaew. Evaluating Refined Queries in Top-k Retrieval Systems. *IEEE TKDE*, 16(2):256–270, 2004.

[15] R. Torlone and P. Ciaccia. Which Are My Preferred Items? In *AH2002 Workshop on Recommendation and Personalization in eCommerce (RPeC 2002)*, pages 1–9, 2002.