Optimal Incremental Evaluation of Preference Queries Based on Ranked Sub-queries[⋆] (Extended Abstract)

Ilaria Bartolini and Paolo Ciaccia

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

Abstract. We consider the problem of efficiently answer *preference queries* when access to the database is through a set of *ranked lists* and preferences define a *strict partial order* over the database objects. The iMPO-1 algorithm we introduce minimizes the number of database accesses and can also *incrementally* return objects as soon as they are guaranteed to be part of the result. As such it generalizes known techniques developed for specific cases, such as Skyline queries.

1 Introduction

Modern information systems are increasingly faced with the problem of efficiently answering *preference queries*. Unlike ordinary queries, which specify a set of constraints that database objects have to satisfy in order to qualify, a preference query retrieves those objects that "best match" the query specification, thus automatically relaxing the query conditions depending on the actual database contents. Although a multitude of works have addressed the problem of how to efficiently evaluate preference queries when the goodness of an object is measured through a (numerical) *scoring* (or *utility*) function, recent works have highlighted the intrinsic limits of such approach [5] and proposed more general preference models. Remarkable extensions are so-called *Skyline* queries [4], dealing with Pareto preferences, and Best-Matches-Only (BMO) queries [5, 9, 12], dealing with preferences defining an arbitrary strict partial order over database objects. Efficient query evaluation techniques for these extended models can be found, among others, in [1, 2, 4, 10, 11], both for centralized environments, in which access to objects is through a single multi-dimensional index, as well as for the distributed case, in which objects' features are to be accessed through separate index structures and/or they are remotely distributed over multiple information sources. While for Skyline queries both cases have been addressed and optimal solutions are nowadays available [11, 2], this is definitely not the case for the more general BMO queries, for which only sequential algorithms are known [6, 12]. In this paper we partially fill this gap by providing an optimal algorithm, called iMPO-1, for answering the "interesting" fragment of BMO queries under the

⋆ This work is partially supported by the WISDOM MIUR Project

distributed access model. More precisely, iMPO-1 can correctly compute the result of any BMO query that respects a natural assumption of monotonicity with respect to the results returned by the underlying sources.

2 Query and Access Model

Our query and access model is deliberately simple and abstract, so as to avoid making unnecessary hypotheses on the underlying architecture. In particular, we do not make any assumption about the specific data model and query language.

Staying in the stream opened by Fagin's pioneering work [7], and then adopted by many others (see, e.g., [8, 2, 1]), we consider a collection C of objects, $C = \{o_1, \ldots, o_n\}$, where all o_i's share (at least) a set of m common features (attributes) $F = \{F_1, \ldots, F_m\}$. A *preference* query is a pair $\mathcal{Q} = (SQ, \succ)$, where $SQ = (Q_1, Q_2, \ldots, Q_m)$ consists of m (preference) sub-queries (each Q_q refers to a distinct objects' feature[1]) and $\succ$ is to be explained below. For each sub-query Q_q we assume that objects in C are assigned a *partial score*, $s_{i,q} \in [0, 1]$, which synthesizes "how well" object o_i matches Q_q, with higher values being clearly better. The specific modalities adopted for the evaluation of sub-queries are uninfluential to our arguments. We just require that the evaluation of Q_q yields a corresponding *ranked list* L_q of pairs $(o_i, s_{i,q})$, ordered by descending score values. Relevant information can then be retrieved through one of two distinct access modalities: A *sorted access* retrieves from a list L_q the next unseen object on that list, say o_i, together with its partial score, $s_{i,q}$; a *random access*, on the other hand, given an object o_i seen via sorted access on some list L_q, gets the partial score $s_{i,q'}$ for sub-query $Q_{q'}$ ($q' \neq q$).

Sub-queries in SQ map each object o_i into a point $\mathbf{p_i} = (s_{i,1}, \ldots, s_{i,m})$ of the m-dimensional *answer space* $A = [0, 1]^m$. The image of C in the answer space is denoted P.[2] The result of evaluating $\mathcal{Q}$ over C, $\mathcal{Q}(C)$, is then taken to be the set of objects corresponding to the "overall best" points in the answer space. Clearly, what "overall best" actually means heavily depends on user preferences. For this we first remind the notion of a *preference relation*.

Definition 1 (Preference Relation) *A* preference relation *over a domain X is a binary relation $\succ \subseteq X \times X$. 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* incomparable *(or* indifferent*), written $x_1 \sim x_2$.*

In order to define $\mathcal{Q}(C)$ we adopt the semantics of Best-Matches-Only (BMO) queries as formalized by the Best operator [12]:[3]

$$\beta_{\succ}(P) = \{\mathbf{p} \in P \mid \nexists \mathbf{p}' \in P, \mathbf{p}' \succ \mathbf{p}\} \tag{1}$$

[1] Generalization to the case where Q_q refers to a subset of features is immediate.

[2] Note that this abstraction allows us to take an uniform view of the effects of sub-queries evaluation, thus disregarding problems related, say, to normalization of scores, which are orthogonal to the problem we are dealing with.

[3] This is called *winnow* in [5, 6] and *preference selection* in [9].

Thus, $\beta_{\succ}(P)$ returns all the undominated points in P and $\mathcal{Q}(C)$ is the set of corresponding objects, $\mathcal{Q}(C) = \{o_i \in C \mid \mathbf{p_i} \in \beta_{\succ}(P)\}$. Slightly abusing notation, we also write $\beta_{\succ}(C)$ in place of $\beta_{\succ}(P)$ and $o_i \succ o_j$ whenever $\mathbf{p_i} \succ \mathbf{p_j}$ holds.

Preference relations expressible through a *scoring* function (e.g., min, max, avg, etc.), $S : A \to [0,1]$, which assigns to each point $\mathbf{p_i}$ an overall score $s_i = S(\mathbf{p_i})$, are obviously covered by Definition 1, since for any S one can define $\succ_S$ as $\mathbf{p_i} \succ_S \mathbf{p_j} \Leftrightarrow S(\mathbf{p_i}) > S(\mathbf{p_j})$. However scoring functions can only represent *weak orders*, i.e., linear orders with ties,[4] thus they are far too restrictive if one aims to support more powerful and flexible preference models.

Recent works [9, 5, 6, 12] have considered the more general case in which $\succ$ is a *strict partial order* (PO for short), thus an irreflexive ($x \not\succ x$) and transitive ($x_1 \succ x_2, x_2 \succ x_3 \Rightarrow x_1 \succ x_3$) relation. Given our distributed access model, is it possible to devise some efficient algorithm to compute $\beta_{\succ}(C)$ when $\succ$ is a generic PO? Unfortunately the answer is negative. The intuitive reason is that, if there is no correlation at all between the partial scores and the "overall goodness" of an object (as established by $\succ$), then it could be well the case that the overall best matches of $\mathcal{Q}$ are the ones at the end of the ranked lists. Note that this would be rather counter-intuitive, since it is reasonable to demand that doing better on sub-queries should not worsen the overall goodness of objects. This is formalized by the following definition.

Definition 2 (Monotonicity of preference relations) *A preference relation $\succ$ over the answer space $A = [0,1]^m$ is* monotone *if $s_{j,q} \leq s_{i,q}$ $\forall q$ implies $\mathbf{p_j} \not\succ \mathbf{p_i}$, and* strictly monotone *if $s_{j,q} < s_{i,q}$ $\forall q$ implies $\mathbf{p_i} \succ \mathbf{p_j}$.*

Note that a non-strict monotone preference relation has the pitfall of being potentially insensitive to changes in partial scores, that is, being o_i better than o_j on *all* sub-queries would not guarantee $\mathbf{p_i} \succ \mathbf{p_j}$. Again, we find this very counter-intuitive, the reason of why in the sequel we will only consider strictly monotone PO's, a first example of which are the well-known *Skyline preferences* [4].

Definition 3 (Skyline preferences) *The Skyline preference relation $\succ_{SL}$ over $A = [0,1]^m$ is defined as: $\mathbf{p_i} \succ_{SL} \mathbf{p_j} \Leftrightarrow (\forall q : s_{j,q} \leq s_{i,q}) \wedge (\exists q : s_{j,q} < s_{i,q})$.*

Thus, object 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 o_i performs better than o_j. The set of objects of C for which there is no object that dominates them according to $\succ_{SL}$ is called the *Skyline* of C. Skyline preferences coincide with the notion of *Pareto optimality* from decision theory, where the Skyline is known as the Pareto set. Their importance stems from the observation that each Pareto-optimal object o_i maximizes some monotone scoring function S_i over the objects in C. This is also to say that the Skyline provides us with an "overall view" of the potential best alternatives, without the need to tune the parameters of a scoring function.

Without distracting the reader with too many details on the engineering of complex preferences (see, e.g., [9]), in the following we concentrate on a kind of preferences based on the concept of *priority* among regions of the answer space.

[4] Equivalently, a weak order is a partial order whose indifference relation is transitive.

Consider the case where o_{bad} has scores, say, $\mathbf{p_{bad}} = (0.8, 0.01, \ldots, 0.01)$, and assume 0.8 is the best score for sub-query Q_1, with no other object obtaining such score. Regardless of the poor partial scores $s_{bad,q} = 0.01$ $(q = 2, \ldots, m)$, this alone is sufficient to guarantee that o_{bad} will be part of the Skyline. This could be questionable to the user, especially if the database contains (possibly many) other objects with somewhat "more balanced" score values. On the other hand, the user could be interested in o_{bad} if no such alternative solutions are currently available. Imposing hard constraint on partial scores clearly does not work, since the problem is to return to the user the "best" results compatible with the *actual* contents of the database. Region prioritization easily solves the dilemma. Let $Y = \{A_1, \ldots, A_P\}$ be a partition of the answer space and let $\succ_Y$ be a strictly monotone PO preference relation over Y.

Definition 4 (RS preferences) *Let $Reg : A \rightarrow Y$ 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:*

$$\mathbf{p_i} \succ_{RS} \mathbf{p_j} \Leftrightarrow (Reg(\mathbf{p_i}) \succ_Y Reg(\mathbf{p_j})) \vee ((Reg(\mathbf{p_i}) = Reg(\mathbf{p_j})) \wedge (\mathbf{p_i} \succ_{SL} \mathbf{p_j}))$$

Thus, within a same region the Skyline logic applies, whereas priority among regions prevails if two points belong to different regions.

Example 1. Consider the following Hotels relation, and a query aiming to find cheap and good hotels, thus $m = 2$. Regardless of how partial scores are obtained, those for Q_1 will be negatively correlated with Price and those for Q_2 will exhibit a positive correlation with hotel rating.

HotelID	Price (F_1)	Rating (F_2)
H1	\$ 30	medium
H2	\$ 35	low
H3	\$ 60	high
H4	\$ 60	very high

The Skyline will consist of hotels H1 and H4. On the other hand, with RS preferences one could define, say, a "soft threshold" on the Price attribute, so as to avoid getting too costly alternatives. Let Price=50 be the chosen soft threshold, which leads to have only H1 in the result, since H4 is too costly. If H1 is deleted, the best alternative becomes H2, which now dominates H4. Finally, if also H2 is cancelled, then H4 comes back since no hotel with Price$\leq$ 50 is left. This behavior cannot be obtained by setting Price=50 as a hard constraint, which would simply lead to discard H4. □

In order to guarantee the strict monotonicity of $\succ_{RS}$, priority among regions should not contrast with Skyline preferences. For this it is sufficient to have that, whenever $Reg(\mathbf{p_i}) \neq Reg(\mathbf{p_j})$ and $s_{j,q} < s_{i,q}$ $\forall q$, it is also $Reg(\mathbf{p_i}) \succ_Y Reg(\mathbf{p_j})$.

Although we have combined region prioritization with Skyline preferences, one could easily generalize Definition 4 and use within each region *any* strictly monotone PO preference relation to compare objects. For instance, one could define $Y = \{A_1, A_2, A_3, A_4\}$, and within each region use a, possibly different (!), preference relation, say $\succ_{SL}$ in A_1, $\succ_{\min}$ in A_2, and so on.

3 The iMPO-1 Algorithm

Our iMPO-1 algorithm works for any strictly monotone PO preference relation and has an *incremental* (online) behavior. The logic of iMPO-1 can be explained as follows (see Figure 1). At each step iMPO-1 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 o_i via random access (step 5). The so-obtained representative point $\mathbf{p_i}$ is then compared with the current objects in the *Result* set (steps 7 and 8). If no object o_j dominates o_i, o_i is inserted in *Result* (possibly also removing objects dominated by o_i itself), otherwise o_i is discarded. At each point iMPO-1 maintains a *threshold point* $\underline{\mathbf{p}}$, whose q-th component, $\underline{s_q}$, is the lowest partial score seen so far under sorted access on list L_q (step 11). iMPO-1 progressively delivers all objects that are *not* dominated by $\underline{\mathbf{p}}$ (step 12) and stops as soon as an object o_i is found such that $\mathbf{p_i}$ dominates the threshold point $\underline{\mathbf{p}}$ (step 2).

Algorithm iMPO-1

```
(1)  Set $Result = \emptyset$; Set $Output = \emptyset$; Set $\underline{\mathbf{p}} = (1, \ldots, 1)$; /* $\underline{\mathbf{p}}$ is the threshold point */
(2)  While ($\nexists (o_i, \mathbf{p_i}) \in Result$ such that $\mathbf{p_i} \succ \underline{\mathbf{p}}$):
(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 $\mathbf{p_i}$; /* random accesses */
(6)      Set $Dominated$ = false;
(7)      While (not($Dominated$) $\wedge\ \exists\ (o_j, \mathbf{p_j}) \in Result$ unmatched with $\mathbf{p_i}$):
(8)        Compare $\mathbf{p_i}$ with $\mathbf{p_j}$: $\begin{cases} \mathbf{p_i} \succ \mathbf{p_j} & \text{remove } (o_j, \mathbf{p_j}) \text{ from } Result, \\ \mathbf{p_i} \sim \mathbf{p_j} & \text{do nothing,} \\ \mathbf{p_j} \succ \mathbf{p_i} & \text{set } Dominated = \text{true;} \end{cases}$
(9)      End While;
(10)     If not($Dominated$) insert $(o_i, \mathbf{p_i})$ in $Result$;
(11)     Let $\underline{s_q}$ be the lowest score seen by sorted access on list $L_q$; Set $\underline{\mathbf{p}} = (\underline{s_1}, \ldots, \underline{s_m})$;
(12)     Set $Output = Output \cup \{(o_i, \mathbf{p_i}) \in Result \mid \underline{\mathbf{p}} \not\succ \mathbf{p_i}\}$; /* send to output */
(13)   End For;
(14) End While.
```

Fig. 1. The iMPO-1 algorithm

In order to prove that iMPO-1 is both correct and optimal (in a sense to be made precise below), we assume that $\succ$ is *Pareto-consistent*, that is, $(x_1 \succ x_2) \wedge (x_2 \succ_{SL} x_3) \Rightarrow (x_1 \succ x_3)$ and $(x_1 \succ_{SL} x_2) \wedge (x_2 \succ x_3) \Rightarrow (x_1 \succ x_3)$. This hypothesis has a negligible impact on the general applicability of iMPO-1. Indeed, any reasonable strictly monotone PO preference relation is also Pareto-consistent.[5] A non-Pareto-consistent preference relation would indeed exhibit a rather strange behavior: A point $\mathbf{p_i}$ is preferred to point $\mathbf{p_k}$ but not to another point $\mathbf{p_j}$ whose partial scores are all less than or equal to those of $\mathbf{p_k}$![6]

[5] For instance, this is the case if $\succ$ represents a strictly monotone scoring function S. Skyline and Region-prioritized Skyline preferences are also Pareto-consistent.

[6] However, extending algorithm iMPO-1 so as to process also non Pareto-consistent preferences is easy: After $\mathbf{p_i} \succ \underline{\mathbf{p}}$ has been verified, "freeze" $\underline{\mathbf{p}}$ and execute some further sorted accesses so as to see on all lists partial scores less than those of $\underline{\mathbf{p}}$.

Theorem 1 (Correctness) *The iMPO-1 algorithm correctly computes $\beta_{\succ}(C)$ for any strictly monotone PO preference relation $\succ$ that is Pareto-consistent.*

Proof. ($\beta_{\succ}(C) \subseteq \mathit{Result}$). If an object $o_j \in \beta_{\succ}(C)$ has been retrieved via sorted access, then step 8 guarantees $o_j \in \mathit{Result}$ when the algorithm stops. Thus, assume by contradiction $o_j \in \beta_{\succ}(C)$, yet o_j has not been seen by the algorithm. Let o_i be the object that is found at step 2 to dominate the threshold point. Unless $\mathbf{p_j}$ is coincident with the threshold point, in which case we are done, at least one partial score of o_j is strictly less than the corresponding threshold value. It follows that $\underline{\mathbf{p}} \succ_{SL} \mathbf{p_j}$. Since $\succ$ is Pareto-consistent it is $\mathbf{p_i} \succ \mathbf{p_j}$ (since $\mathbf{p_i} \succ \underline{\mathbf{p}}$), thus $o_j \notin \beta_{\succ}(C)$.
($\mathit{Result} \subseteq \beta_{\succ}(C)$). We prove that $o_j \notin \beta_{\succ}(C)$ implies $o_j \notin \mathit{Result}$. If o_j has not been seen then it cannot be part of the final result. Thus, assume o_j has been seen. Since $o_j \notin \beta_{\succ}(C)$ and $\succ$ is a PO there is at least one object $o_i \in \beta_{\succ}(C)$ such that $o_i \succ o_j$. Since we have already proved that $\beta_{\succ}(C) \subseteq \mathit{Result}$, such o_i has been seen. Thus, either o_i and o_j have been compared or $\exists\ o_k$ such that $o_i \succ o_k$, $o_k \succ o_j$, and o_k and o_j have been compared. Therefore, $o_j \notin \mathit{Result}$. □

Theorem 2 (Correctness of delivery condition) *The iMPO-1 algorithm correctly delivers all the objects in $\beta_{\succ}(C)$.*

Sketch of proof. Having proved that $\mathit{Result} = \beta_{\succ}(C)$ it remains to show that $\mathit{Result} = \mathit{Output}$. $\mathit{Result} \subseteq \mathit{Output}$, which says that when iMPO-1 stops all objects in the final *Result* have already been sent to *Output*, follows since all objects in *Result* are pairwise indifferent and $\mathbf{p_i} \succ \underline{\mathbf{p}}$ implies $\underline{\mathbf{p}} \not\succ \mathbf{p_j}\ \forall \mathbf{p_j} \in \mathit{Result}$. $\mathit{Output} \subseteq \mathit{Result}$, which asserts that the delivery condition $\underline{\mathbf{p}} \not\succ \mathbf{p_j}$ is correct, follows from Pareto-consistency. □

Besides being correct, iMPO-1 is also *instance-optimal* [8]. Given a class $\mathbf{A}$ of algorithms and a class $\mathbf{DB}$ of (database) instances (inputs), an algorithm $A \in \mathbf{A}$ is instance-optimal over $\mathbf{A}$ and $\mathbf{DB}$ iff $\forall B \in \mathbf{A}$ and $\forall DB \in \mathbf{DB}$ it is $\mathit{Cost}(A, DB) = O(\mathit{Cost}(B, DB))$, where *Cost* is a suitable cost measure. Thus, instance-optimality means optimality up to a constant factor (thus independent of the instance size) over every possible instance, which is a much stronger property than, say, worst-case optimality. Given our access model, the obvious choice is to take as cost measure the total number of (sorted + random) accesses needed to compute the query result.

Theorem 3 (Optimality) *Let* $\mathbf{DB}$ *be the class of all instances ranking objects into m lists, and* $\mathbf{A}$ *be the class of all algorithms accessing such lists using sorted and random accesses and not making wild guesses that correctly compute the result when $\succ$ is as in Theorem 1. The iMPO-1 algorithm is instance-optimal over* $\mathbf{DB}$ *and* $\mathbf{A}$.

Sketch of proof. First observe that an algorithm makes a "wild guess" if it performs a (blind) random access for an object that has not been seen via sorted access. Such algorithms have only a theoretical interest and would not be considered for implementation. The proof essentially amounts to show that any correct algorithm $A \in \mathbf{A}$ can stop *only if* it finds o_i such that $\mathbf{p_i} \succ \underline{\mathbf{p}}$, and that, no matter how smart is the scheduling of accesses with respect to the round-robin strategy of iMPO-1, this requires at least a number of accesses that can improve over those performed by iMPO-1 by at most a constant factor m. □

Theorem 4 (Optimality of delivery times) *Let* **DB** *and* **A** *as in Theorem 3. No algorithm in* **A** *can output* o_j *by performing less accesses than iMPO-1 does (up to a constant factor).*

Proof. Omitted.□

4 Experimental Results

In this section we provide some experimental evidence of the actual performance of iMPO-1, that we implemented in C++ on top of Windsurf [3]. Windsurf is a region-based image retrieval system that, using wavelet transform and k-means clustering, segments each image into a set of regions and then represents each region through a 37-dimensional feature vector. When an image query $\mathcal{Q}$ is submitted the same procedure is adopted and each of the resulting regions becomes a distinct sub-query. Partial scores for a given query region are obtained by using the *Bhattacharyya metric*. The results we present are obtained using a real-world image collection of about 10,000 color images and by averaging performance over a sample of 100 randomly-chosen query images. For integrating sub-queries results we considered two scoring functions (min and avg), and the Skyline (SL) and Region-prioritized Skyline (RS) preferences. For RS preferences we defined on each of the m coordinates of the answer space a “soft threshold” θ_q $(0 < \theta_q < 1)$, then assigning 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 a partition Y of 2^m regions, each represented by an m-bit code. Given regions A_i and A_j, we define $A_i \succ_Y A_j \Leftrightarrow \mathit{code}(A_i) \wedge \mathit{code}(A_j) = \mathit{code}(A_j)$, where bitwise AND is used and $\mathit{code}(A_i)$ is the binary code of region A_i. For instance, when $m = 4$, region 1011 dominates region 1000, whereas it is indifferent to region 0100. Results we show are obtained with $\theta_q = 0.4 \ \forall q$.

Figure 2 clearly demonstrates the advantage of an early delivery of objects, which reduces the user’s waiting time by orders of magnitude. For instance, the 1st result just requires 8 sorted accesses (2 on each list), rather than 628 if the incremental delivery condition would be dropped and all objects returned at the end of execution. Figure 3 reports the total number of accesses needed to deliver k objects, depending on the specific preferences. It is evident that SL is the faster alternative, saving on the average about 66% and 75% accesses against avg and min, respectively. Efficiency of RS is only slightly poorer, however reaching a performance level that is always better than that of both avg and min (30% and 50% speed-up, respectively). The reason why the performance of iMPO-1, although always optimal, can vary with the preference relation depends on the

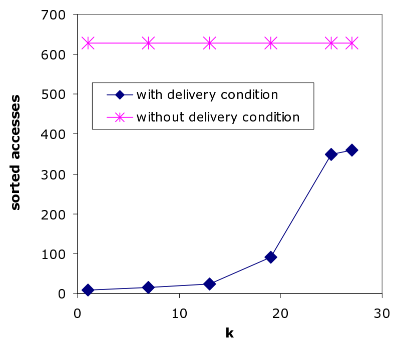

Fig. 2. Skyline preferences: Sorted accesses vs no. of delivered objects (k)

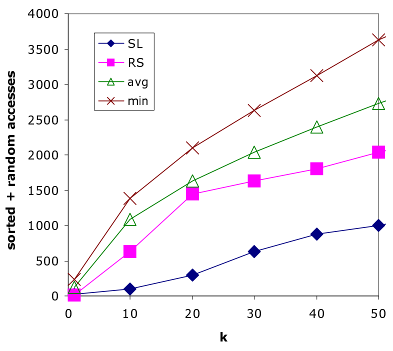

Fig. 3. Sorted + random accesses vs no. of delivered objects (k)

specific delivery condition. For instance, SL performs faster than RS since it can be shown that $(\underline{\mathbf{p}} \not\succ_{RS} \mathbf{p_i}) \Rightarrow (\underline{\mathbf{p}} \not\succ_{SL} \mathbf{p_i})$, thus the delivery condition of RS is always more restrictive than that of SL. In particular, this is also true for *any other* preference relation, thus Skyline preferences will always return faster their k-th result object.

References

1. W.-T. Balke and U. Güntzer. Multi-Objective Query Processing for Database Systems. In *Proceedings of VLDB'04*, pages 936–947, 2004.
2. W.-T. Balke, U. Güntzer, and J. X. Zheng. Efficient Distributed Skylining for Web Information Systems. In *Proceedings of EDBT'04*, pages 256–273, 2004.
3. I. Bartolini, P. Ciaccia, and M. Patella. A Sound Algorithm for Region-Based Image Retrieval Using an Index. In *Proceedings of QPMIDS'00*, pages 930–934, 2000.
4. S. Börzsönyi, D. Kossmann, and K. Stocker. The Skyline Operator. In *Proceedings of ICDE'01*, pages 421–430, 2001.
5. J. Chomicki. Querying with Intrinsic Preferences. In *Proceedings of EDBT'02*, pages 34–51, 2002.
6. J. Chomicki. Preference Formulas in Relational Queries. *ACM Transactions on Database Systems (TODS)*, 28(4):427–466, 2003.
7. R. Fagin. Combining Fuzzy Information from Multiple Systems. In *Proceedings of PODS'96*, pages 216–226, 1996.
8. R. Fagin, A. Lotem, and M. Naor. Optimal Aggregation Algorithms for Middleware. In *Proceedings of PODS'01*, pages 216–226, 2001.
9. W. Kießling. Foundations of Preferences in Database Systems. In *Proceedings of VLDB'02*, pages 311–322, 2002.
10. D. Kossmann, F. Ramsak, and S. Rost. Shooting Stars in the Sky: An Online Algorithm for Skyline Queries. In *Proceedings of VLDB'02*, pages 275–286, 2002.
11. D. Papadias, Y. Tao, G. Fu, and B. Seeger. An Optimal and Progressive Algorithm for Skyline Queries. In *Proceedings of ACM SIGMOD 2003*, pages 467–478, 2003.
12. R. Torlone and P. Ciaccia. Which Are My Preferred Items? In *Proceedings of RPeC'02*, pages 1–9, 2002.