Multi-objective Query Processing for Database Systems

Wolf-Tilo Balke
Computer Science Department
University of California
Berkeley, CA, USA
balke@eecs.berkeley.edu

Ulrich Güntzer
Institut für Informatik
University of Tübingen
Tübingen, Germany
guentzer@informatik.uni-tuebingen.de

Abstract

Query processing in database systems has developed beyond mere exact matching of attribute values. Scoring database objects and retrieving only the top k matches or Pareto-optimal result sets (skyline queries) are already common for a variety of applications. Specialized algorithms using either paradigm can avoid naïve linear database scans and thus improve scalability. However, these paradigms are only two extreme cases of exploring viable compromises for each user's objectives. To find the correct result set for arbitrary cases of multi-objective query processing in databases we will present a novel algorithm for computing sets of objects that are non-dominated with respect to a set of monotonic objective functions. Naturally containing top k and skyline retrieval paradigms as special cases, this algorithm maintains scalability also for all cases in between. Moreover, we will show the algorithm's correctness and instance-optimality in terms of necessary object accesses and how the response behavior can be improved by progressively producing result objects as quickly as possible, while the algorithm is still running.

1. Introduction

Optimizing parameters under multiple constraints and negotiating compromises between different objectives has a long history in economic problems. Though simplifying approaches often reduce business decisions to 'maximize profits', common problems often deal with non-monetary intangibles like product quality, public image, tradition, corporate identity or ethics like environmental concerns or safety features. But apart from mere business problems multi-objective optimization also plays a role in many areas of computer science:

- Multi-objective agents negotiate compromises on behalf of different users or interest groups
- Decision support systems try to integrate various interests to recommend strategic decisions
- Trade-offs in e-commerce environments e.g. between price, efficiency and quality of certain products have to be assessed
- Personal preferences of users requesting a Web service for a complex task have to be evaluated to select most appropriate services

Also in the field of databases and query optimization such optimization problems often occur like in [22] for the choice of query plans given different execution costs and latencies or in [19] for choosing data sources with optimized information quality. Let us mathematically formulate the problem of multi-objective optimization in database retrieval and then consider typical sample applications for information systems:

Multi-objective Retrieval: Given a database containing N objects $O := \{o_1,\ldots,o_N\}$, n characteristics $s_k(o)$ $(1 \leq k \leq n)$ to describe the objects (e.g. scoring functions for low-level features, aggregations of attribute values, etc.) and m monotonic functions f_i $(1 \leq i \leq m)$ aggregating subsets of the characteristics by objective functions, the problem is how to find the overall best database objects with respect to all m scoring functions.

For the scope of the paper we will assume that the characteristics s_k are scoring functions to evaluate certain characteristics of database objects assigning normalized scores in [0,1], i.e. we use a numerical domain for retrieval. Other approaches relying on more general characteristics like projections on the attributes themselves are more powerful in that they can also handle attribute-valued data with partial preference orders. However, all algorithms presented so far for this problem, are of quadratic complexity, whereas those assuming numerical scoring functions can use algorithms of essentially improved complexity, see e.g. [15]. To abstract from attribute-

Proceedings of the 30th VLDB Conference,
Toronto, Canada, 2004

valued domains, we thus might need to derive suitable metrics for each dimension like shown in [18] to always enable numerical scoring of database objects.

Let us now consider a common use case scenario in database applications. A database usually stores a large number of values characterizing certain real world objects. Database applications usually rely on these characteristics, but provide an added value by selecting and aggregating the data along either domain-specific or user-provided functions or algorithms. From a retrieval point of view the derived measures in case of very simple arithmetical functions (avg, max, etc.) can be directly integrated into SQL statements, but for more complex functions user defined functions (UDFs) or stored procedures become necessary. The naïve way of addressing the problem of getting best values is to calculate the UDF for *every* database object and then order the objects by the derived scores. Consider the following example:

Example 1: Real Estate Information

When considering to buy a house, a database of real estate information on available objects may be particularly helpful. Generally the database can provide a variety of basic data like a house's size, its price or its location. But in buying a convenient place to live users often consider other measures derived from these initial data. For instance a user can put a constraint on the maximum budget and minimum size, but then might be interested in bargains with a good price per square-meter, thus aggregating the basic price and size information. On the other hand a user might be interested to find a good location and thus use a function on the location information in terms of rating the neighborhood or ranking houses according to the relative distance from the user's workplace.

As we can see this example poses a multi-objective problem because ranking schemes in price and size simply will not do, but a complex function reordering the database objects is called for (Note that neither the top-ranked houses according to price nor the top-ranked houses according to size have to be top-ranked with respect to our price/size function). And the bargains offered of course do not affect their respective location, thus this gives the user an independent ranked list to choose from in a skylining fashion. The problem becomes even harder, if personal preferences come into the aggregation. Consider the following example:

Example 2: Web Information Services

Imagine a personalized route planning system like [3] where different characteristics of each route are mapped onto a numerical domain. Such characteristics may be the length of each route relative to the shortest one s_1, the probable delay by traffic jams s_2 (e.g. measured by aggregating the number, length and grade of congestion of current jams on a route) or the weather conditions s_3 aggregating visibility (rain, fog) or the danger of black ice. Retrieving the 'best' route for every driver now, however, poses a severe problem: whereas it is natural to aggregate some characteristics others might again be considered incomparable.

For instance users will generally be willing to drive a slightly longer route, if it is not congested. Hence the trade-off between relative length and congestion can usually be determined by taking the average pain/gain ratio. The length of a route and its estimated delay by traffic jams can thus be aggregated using a suitably weighted average as compensation function for economy (e.g. in terms of shortest expected travel times) and our two basic scores as input, i.e. f_{eco} (s_1, s_2). Unlike length and traffic density the weather conditions will generally not be aggregated as easily, because the relative gain through 'better' weather is quite subjective. A motorcyclist may insist on better weather routes whereas a car driver might care less and anyway the respective compensation function lacks intuition. What does it mean, if good weather is said to be e.g. 0.63 times more important than economy?

A good solution here is exploring the skyline recommending very economic routes with possibly bad weather conditions, less economic routes with fair conditions and some possible compromises in between. Depending on the personal preferences a user then might decide for one of these possibilities. Thus our multi-objective problem in route planning has three score functions (length, congestion and weather) that are mapped onto a two-dimensional objective space (economy, weather). The objective functions used are the compensation function for economy and the trivial projection for the weather scores, i.e. for each route x we consider $F(x) := (f_{eco}(s_1(x), s_2(x)), s_3(x))$. In the following we will revisit this sample problem and present a single optimal algorithm to solve it efficiently.

The rest of the paper is organized as follows: section 2 will investigate the nature of multi-objective retrieval and revisit basic approaches of top k retrieval and skylining as special cases in database retrieval. Section 3 will present our unifying multi-objective retrieval algorithm and prove its basic properties and instance-optimality. We show the possibility and optimality of progressive delivery of result objects in section 4 and focus on practical scenarios and their impact on the result complexity in section 5.

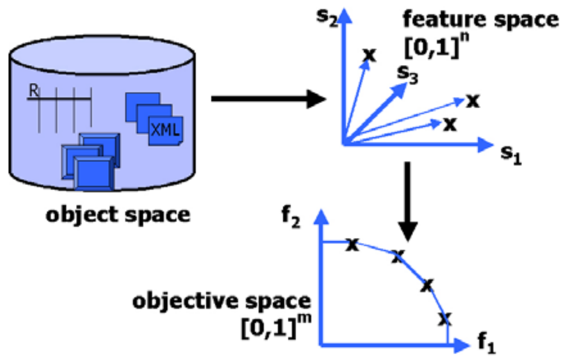

Fig. 1: Multi- objective retrieval

2. Towards Multi-objective Retrieval

In the problem definition and the use case scenario we have gained insight into the basic problem. Characteristics of database objects (like table data, images, XML documents, etc.) are numerically evaluated in the feature space and these scores are aggregated by means of user-specific objective functions (cf. fig. 1). To get a further understanding of the multi-objective optimization problem we now have to determine what 'overall best' means in this instance. For each database object o we get an m-dimensional vector containing the aggregated scores $(f_1(s_1(o),\ldots,s_n(o)),\ldots, f_m(s_1(o),\ldots, s_n(o)))$ for each objective f_i. The problem of picking best vectors among these is one of the central problems addressed in operations research, e.g. [9], [4]. Operations research literature basically distinguishes between returning all non-dominated solutions (also known as 'efficient frontier' or 'Pareto optimal' solutions) among all m-dimensional vectors, or using specific qualitative ordering techniques (usually complex functions, the so-called 'utilities') that allow to order vectors by further aggregating their score values and then returning only the best-ranked objects [12].

2.1. Utility-based Retrieval Models

In utility-based approaches doing worse in some features can be compensated by doing better in others. A utility function then aggregates each vector to a scalar overall utility. Often even the existence of a *single* global utility function is implicitly or explicitly assumed. This global aggregation of objective scores can be processed by **top k retrieval** algorithms. Often besides the scoring functions also some fixed a priori constraints on the values for each vector component are given, so-called bottleneck conditions. Before returning the result set to the user these constraints can discard solutions that violate essential constraints. However, top k retrieval cannot work with multiple objectives, i.e. incomparable utilities or goals.

Among the non-aggregated definitions of overall best objects, the notion of **Pareto optimality** (often referred to as skyline queries in database literature) is the broadest, because only objects are discarded from the final result set that are in all components of their describing vectors worse or equal (with a strict 'worse' in at least one component) than another object of the database, i.e. $o < w$, iff $(s_i(o) \leq s_i(w)\ (1 \leq i \leq n) \wedge \exists\ q \in [1,\ldots,n] : s_q(o) < s_q(w))$. Thus best objects with respect to any monotonic optimization function can be found among the Pareto set (and so can the optimal objects with respect to multiple objectives). This good behavior in recall is, however, generally affecting the precision and the size of Pareto sets has been shown to increase exponentially with growing dimensions of the vectors, i.e. numbers of data characteristics [4].

Besides Pareto optimality, there are other qualitative definitions of the term 'overall best' in the formulation of our problem. For example **lexicographic optimization**, where objective vectors are compared lexicographically, i.e. given a certain order of indexes from 1 to n: $x < y$ iff $s_k(x) < s_k(y)$, where k is the smallest index such that $s_i(x) = s_i(y)$ for all i $(1 \leq i < k \leq n)$. This can be done with respect to one (user-specified), or even all permutations of scoring functions. Operating on numerical domains, this behavior can be realized using top k retrieval, if the order of characteristics is enforced with high weightings in a compensation function such that more important characteristics are assigned weightings that cannot be compensated by less important characteristics, see e.g. [17].

For the use in personalization in database retrieval and engineering of user preferences recent research in [7] and [13] has shown that these three operators (score aggregation, Pareto accumulation and lexicographic ordering) are the most common and essential constructors to build complex user preferences into queries allowing arbitrary combinations. Besides, also the closure of preference construction with these operators is shown for general partial orders. However, up to now for query processing only the naïve algorithm accessing and pairwise comparing all database objects has been proposed.

2.2. Basic Object Access Model

Our work in multi-objective database retrieval aims at using a minimum number of object accesses before delivering the final result set. Unlike statistical approaches we will always guarantee a correct result set. There have been a number of approaches for top k or skyline algorithms (see sections 2.3 and 2.4) that show advantages in different architectures or for different data distributions. For the scope of this paper we will abstract from the underlying system architecture and just rely on some very basic access functions common in query processing literature. We will assume (several) ranked lists for each feature of the query in which database objects are ranked in descending order according to their score values with respect to a single feature. A *sorted access* iterates these lists and accesses objects rank by rank. A *random access* on the other hand can be posed to a data source retrieving the score values of a specific object with respect to a single feature.

Owing to this abstraction our approach can be applied in both middleware architectures and over central multidimensional indexes. Obviously the use of central indexes will essentially speed up the retrieval, but e.g. in Web information systems using various subsystems and assembling the information on the fly it is not always possible to operate on such a central index structure. The main difference is that in distributed systems the iteration over sorted lists for each characteristic (possibly provided by different subsystems) is stopped, if a certain condition becomes true and all unseen objects may be discarded, whereas using multidimensional indexes like R*-trees whole subtrees can instantaneously be pruned, if the upper/lower bounds for their maximum/minimum objects meet the necessary condition. Examples for this are e.g. the middleware top k retrieval in [11] and the central index top k retrieval in [8] or the distributed skylining in [1] and the central index skylining in [23]. However, in all these cases basic algorithms are similar. We will revisit their basic traits in brief within the following sections.

2.3. Top k Retrieval (m = 1)

Top k retrieval states the basic idea that a *single* monotonic compensation function can be used to aggregate a certain number of characteristics of database objects into a single score providing the final order of the all objects among which k best objects are to be singled out. For weighting the relative importance of each single characteristic and thus determining the degree of compensation, users are generally allowed to specify weights, e.g. in a weighted average as compensation function. This approach often occurs e.g. in information retrieval, where weighted averages for the keyword occurrences are calculated or in content-based retrieval over Multimedia databases. Please note, that in this case all database objects are ranked into a (non-strict) total ordering. Besides central index approaches like k-nearest-neighbor searches, the algorithm using a threshold condition has been proved to be most efficient for applications [20], [11], [21], [6], [2] and has even been shown to be instance-optimal [10]. To be self-contained we present the basic algorithm below.

Basic Top k Retrieval:

1. Get an object by sorted access on one of the lists
2. For new objects perform random accesses on the other lists and aggregate the object's total score using the compensation function
3. Aggregate a threshold using the current minimum scores in each list as input for the compensation function
4. If there are k objects having a higher or equal score than the current threshold, discard all unseen objects, otherwise return to step 1
5. Output the k objects with highest score as top k objects

Please note that top k algorithms generally will neither retrieve all, nor only strictly the optimal objects with respect to the compensation function. They will output some k best objects and stop, even if some of these k objects are already dominated or there are more than k objects having the optimal score. The choice of a suitable value of k for each application is therefore difficult. If all objects with the optimal score should be retrieved ('all top objects'), the condition in step 4 has to be altered to 'strictly higher (>)' instead of 'higher or equal (≥)'.

2.4. Skyline Queries and Pareto Optimality (m = n, $(\mathbf{f}_1,\ldots,\mathbf{f}_n) = (\mathbf{id}_{S1},\ldots,\mathbf{id}_{Sn})$)

A skyline query describes the case when *all* characteristics are considered to be incomparable and the objective functions are just the projections on the respective scores (i.e. $f_i(s_1,\ldots,s_n) = s_i$). Since no compensation between characteristics is possible, the query result (the so-called skyline) consists of all objects that are not dominated in all aspects by any other object in the sense of Pareto optimality. Efficient algorithms have been heavily researched in recent years; see e.g. [5], [24], [14], [23]. Also in this case the algorithms differ in a variety of heuristics and techniques used, but again we will present the basic optimal algorithm only using the abstract access methods over sorted lists for each aspect as given in [1].

Basic Skylining:

1. Get an object by sorted access on one of the lists
2. For new objects perform random accesses on the other lists
3. If an object has already occurred in all of the lists, retrieve all additional objects that have the same score as the current minimum in each list and discard all unseen objects, otherwise return to step 1
4. Compare objects pairwise and output all non-dominated objects as the skyline

3. An Multi-objective Retrieval Algorithm

Having revisited the special algorithms we are ready to present an efficient unifying algorithm that allows to handle *all* cases of multi-objective retrieval. Given sorted lists for each characteristic and using the two basic access techniques of sorted and random access all retrieval algorithms will start iterating over the lists (since wild guesses cannot be sensible). The naïve approach would just iterate completely over the lists, get the score values for each database object and then start the aggregation with the objective functions and compare the objects pairwise for domination. Obviously this algorithm accesses all database objects and is of quadratic complexity $O(N^2)$ in terms of pairwise comparisons. Thus it definitely cannot scale with large database sizes. To improve this behavior the important issue is to know when the iteration over the sorted lists can be stopped at the earliest time and the rest of database objects can be discarded. We will show the important result that our algorithm only uses an **instance-optimal** number of expensive object accesses.

But first we need some basic properties of the objective functions and then we are ready to present an algorithm introducing a virtual object. Throughout this paper we will assume that all objective scores are given by a set of monotonic functions. We will also assume the objective functions not to be constant on the set of all objects (constant functions could simply be omitted, because they do not influence the result). And we will denote the objective scores of an object o as $F(o) := (f_1(s_1(o),\ldots,s_n(o)),\ldots, f_m(s_1(o),\ldots,s_n(o)))$ and the domination in the sense of Pareto optimality as $F(x) > F(y)$.

Basic Multi-objective Retrieval:

0. Given *n* lists ranking *N* database objects, each sorted descending by score and *m* monotonic functions $f_1,\ldots,f_m$
1. Get an object *o* by sorted access from any list in a round robin fashion
2. For new objects perform random accesses on the other lists and calculate the object's objective scores F(*o*)

3. Create a virtual database object *p* characterized by the minimum score values that have occurred in each list, as its score values (i.e. $s_i(p)$ is the current minimum score in the *i*-th list) and calculate its objective scores F(*p*)
4. If some object *w* has already been seen for which holds F(*w*) > F(*p*), i.e. its objective scores are better or equal, but in at least one dimension strictly better than the virtual object's, discard all unseen objects, else return to step 1
5. Compare all seen objects pairwise and output all non-dominated objects as the result set of non-dominated objects.

Let us now consider our traffic information example (example 2) and see, how the algorithm works. We got three ranked lists of which the first two (distance and congestion) can be compensated say using a weighted average $f_{eco}(o) := ½\ (s_1(o) + s_2(o))$ and consider the third score (weather) under the notion of Pareto optimality. Please note that this is just a simple case of two objectives that could be realized by first running a top k algorithm on list s1 and s2 and then run a skyline algorithm on the materialized list f_{eco} and s3, however resulting in linear database scans accessing all database objects. Consider the following three lists s1, s2 and s3 containing database objects o_i and their scores in descending order. Table 1 shows the first few top ranked objects of each list.

	s_1 distance		s_2 jam free		s_3 weather	
rank	oid	score	Oid	score	oid	Score
1	o_1	0.98	o_2	0.92	o_3	0.9
2	o_4	0.94	o_5	0.84	o_6	0.8
3	o_7	0.9	o_8	0.82	o_9	0.8
4	o_{10}	0.87	o_{11}	0.81	o_{12}	0.8
…	…	…	…	…	…	…

Table 1: Sample values for three lists

Oid	score s_1	score s_2	Score s_3	Objective scores	scores of the virtual object at access time
o_1	**0.98**	0.62	0.3	(0.8, 0.3)	(0.99, 1.0)
o_2	0.88	**0.92**	0.1	(0.9, 0.1)	(0.95, 1.0)
o_3	0.5	0.5	**0.9**	(0.5, 0.9)	(0.95, 0.9)
o_4	**0.94**	0.82	0.8	(0.88, 0.8)	(0.93, 0.9)
o_5	0.56	**0.84**	0.2	(0.7, 0.2)	(0.89, 0.9)
o_6	0.6	0.4	**0.8**	(0.5, 0.8)	(0.89, 0.8)
o_7	**0.9**	0.5	0.7	(0.7, 0.7)	(0.87, 0.8)

Table 2: Object accesses using the new algorithm

We will start with sorted accesses in a round robin fashion on all three lists (shown bold) in table 2 (step 1). After each sorted access we do the necessary random accesses to calculate the object's objective scores (step 2). The first part of the objective score is then calculated using the function f_{eco}, whereas the second part here is simply the projection on s_3. Then we calculate the virtual object with scores using the minimum values in each list as input (step 3). After retrieving object o_7 however, we can according to step 4 of our algorithm already stop object accesses and discard all unseen objects, because object o_4 now dominates the virtual object. After comparing all seen object pairwise for domination (step 5) we will find that in our small example only o_2, o_3 and o_4 are non-dominated by other objects and can be output as optimal results (o_1, o_5, o_6 and o_7 are all dominated by o_4). Thus, even though in this case a implementation by top k/skylining algorithms would be possible, we are able to provide a definitive output already after the first few object accesses instead of needing a linear database scan.

Before we further refine the algorithm let us state its correctness and compare it to the basic algorithms in the special cases in 2.3 and 2.4.

Theorem 1: *Correctness of Multi-objective Retrieval* The basic algorithm for multi-objective retrieval always terminates and delivers all non-dominated and only the non-dominated objects with respect to the given set of monotonic objective functions F.

Proof:
If all objects are non-dominated, the algorithm obviously will terminate after one of the lists has been entirely processed and all objects have been seen and correctly output. However, generally not all objects will be non-dominated with respect to the objective functions F. Thus, if we have seen any two objects *x* and *y* with F(*x*) > F(*y*), i.e. *y* is dominated by *x*, termination is obviously guaranteed at the latest when *y* has already occurred in all lists, due to F(*x*)> F(*y*) ≥ F(*p*).

Now we have to prove that by discarding all unseen objects after F(*x*) > F(*p*) holds for any seen object *x* we will never discard a relevant, i.e. non-dominated, object. Let *U* be the set of still unseen (and thus in step 4 discarded) objects after termination and *Q* the set of all seen objects. Let's assume that until termination we have iterated all *i* lists down to values p_i $(1 \leq i \leq n)$ and have also retrieved in step 4 at least one object *q* dominating the virtual object. Since the objects in *U* have not been seen, we can conclude that their score is smaller or equal in all lists than the score of the virtual object *p* with $s_i(p) := p_i$. Due to the set of objective functions being monotonic we get: $\exists\, q \in Q\ \forall\, u \in U : \mathrm{F}(u) \leq \mathrm{F}(p) < \mathrm{F}(q)$.

Thus *q* dominates all the unseen objects and since in step 5 we check all seen objects pairwise for domination, we can neither leave out relevant objects nor return dominated objects. ■

Additionally, we will now show that our algorithm's I/O costs (i.e. total number of object accesses) are instance-optimal. The concept of instance optimality was defined by [10] over abstract classes of algorithms and database instances. If an algorithm's complexity over any possible instance of databases (i.e. any number of objects and any distribution of scores) is optimal among all algorithms in a certain class, it is said to be instance-optimal for this class of algorithms. Or more formally, consider our algorithm X as an element of the class $\boldsymbol{A}$ of all algorithms, which are capable of delivering correct multi-objective retrieval results for monotonic objective functions, and D as a specific instance of all possible database instances $\boldsymbol{D}$, which are sorted lists of database objects with score values assigned. Since we want to focus on I/O costs we have to consider the necessary number of object accesses, denoted as *accesses*(A). Then X is instance-optimal over $\boldsymbol{A}$ and $\boldsymbol{D}$, if for every algorithm $A \in \boldsymbol{A}$ and every database instance $D \in \boldsymbol{D}$ holds: *accesses*(X, D) = $\boldsymbol{O}$(*accesses*(A,D)), i.e. for any chosen algorithm A of class $\boldsymbol{A}$ we can state *accesses*(X,D) ≤ C *accesses* (A,D) + C' with some positive constants C ('optimality ratio') and C'. The following theorem shows the instance-optimality of our approach:

Theorem 2: *Instance-optimality of object accesses*
Let $\boldsymbol{D}$ be the class of all possible database instances in the form of *n* lists of database objects ranked by score, and $\boldsymbol{A}$ the class of all possible algorithms that use only sorted and random accesses and correctly retrieve all optimal objects from these lists for any set of *m* monotonic objective functions. The preceding multi-objective retrieval algorithm is instance-optimal over $\boldsymbol{A}$ and $\boldsymbol{D}$, i.e. accesses an optimal number of objects up to a constant factor.

Proof:
At our algorithm's termination (and thus the end of object accesses) the first object has been accessed, whose scores are better or equal than the virtual object's and strictly better in at least one dimension. We will show that for any algorithm that will not do sorted accesses up to that point there can be an object that is non-dominated with respect to the objective functions and thus a relevant object would have been missed by the algorithm. Hence the expansion of lists (and thus the number of objects accessed) for our multi-objective retrieval algorithm is instance-optimal.

Since random accesses can be performed only on objects that have been previously seen by sorted accesses and excluding 'wild guesses' on the database content, an algorithm can only access formerly unknown objects by sorted access. For the sake of contradiction let us now assume there would exist a correct algorithm A of class $\boldsymbol{A}$ that could stop object accesses before at least one seen object *o* fulfills F(*o*) > F(*p*), i.e. there is no object that has better or equal objective scores dominating the virtual object. We will show that now we can construct a database instance of $\boldsymbol{D}$ containing at least one object still unseen by A, but nevertheless optimal with respect to the objective functions F, and therefore we get a contradiction to algorithm A's correctness and hence its existence.

Let *p* be a (virtual) object, whose scores are given by the minimum seen by sorted access in each list. Assume that algorithm A has terminated over database instance $D \in \boldsymbol{D}$ having performed sorted accesses up to scores $p_1,\ldots,p_n$ in the *n* lists, however without having accessed at least one object with larger objective scores than the virtual object's (otherwise our algorithm would also already have terminated). We will now construct a database instance $D' \in \boldsymbol{D}$ that is exactly like instance D up to the object *p*'s scores $p_1,\ldots,p_n$, but immediately behind the last object that A has accessed by sorted access in each list on D, in D' we will insert a new object *w* having also score values $p_1,\ldots,p_n$. Obviously D' is a valid instance of class $\boldsymbol{D}$. Due to construction algorithm A will terminate over D' exactly at the point it terminated over D and since no object *o* with F(*o*) > F(*p*) in D has been accessed, also no object with F(*o*) > F(*p*) will be accessed in D'.

Let us now take a closer look at object *w*. Since it has not been accessed by algorithm A before termination, it cannot have been output in the result set of non-dominated objects. Thus either it must be dominated or algorithm A is not correct. We will now show that *w* cannot be dominated by any object in D' and therefore is optimal with respect to the objective functions. According to theorem 1 *w* cannot be dominated by any unseen object, because every unseen object has score values lesser or at most equal $p_1,\ldots,p_n$, i.e. due to the objective functions being monotonic, an unseen object cannot have a strictly better score in any dimension, which however is necessary for domination. Thus *w* would have to be dominated with respect to the objective functions by an object *o* already seen by algorithm A. But by dominating *w* the object *o* would also dominate the virtual object *p* and we would have an object that fulfills F(*o*) > F(*w*) = F(*p*) in contradiction to our assumption. Though having proved that the termination condition is necessary, there still could be an algorithm using a more sophisticated strategy to choose lists for the next access than round robin (cf. section 5.2). However, let *a* be this number of accesses, then a round robin strategy will at the latest stop after *n*a* accesses. Thus the instance optimality still holds with an optimality ratio C = *n*. Please note, that we did not make any assumptions on the distribution of scores or the nature of the objective functions (except being monotonic). Thus our basic algorithm is instance-optimal over all multi-objective retrieval algorithms for all possible database instances. ■

We will now state that both basic algorithms for top k retrieval and skylining are just special cases of our multi-objective retrieval algorithm. Please note that the respective algorithms were proven to be optimal in [10] and [1].

Observation 1: *Relationship of multi-objective retrieval to top k queries and skylining*
Top k retrieval and skylining are special cases of multi-objective retrieval and in either case the basic multi-objective retrieval algorithm will behave like the given basic algorithms for top k retrieval and skylining.

Proof:

The basic behavior in iterating the sorted lists and doing random accesses to get all score values is the same in all three algorithms. In the special case of top k retrieval (m = 1) the termination condition for multi-objective retrieval is $F(o) := f_1(s_1(o),\ldots,s_n(o)) > f_1(p_1,\ldots,p_n)$. Thus the complete number of objects having the top score is retrieved, i.e. all non-dominated objects being strictly better than the threshold. Generally we will get more than one object, but we will really get *all* top objects (the progressive algorithm in section 4 outputs objects with the respective top score one by one at the earliest possible point in time like progressive top k algorithms). To derive the top k objects, we can return any arbitrary subset of cardinality k out of the top objects or return all top objects and then run the algorithm again with these objects removed, if k is larger than the number of maximal elements.

In the case of skylining we have $n = m$ and $(f_1,\ldots,f_n) := id$. Thus our termination condition becomes:

$$\begin{aligned}F(o) &:= (id)(s_1(o),\ldots,s_n(o)) = (s_1(o),\ldots,s_n(o)) > (p_1,\ldots,p_n) = \\ &= (id)(p_1,\ldots,p_n) =: F(p)\end{aligned}$$

which means that $\forall\ 1 \le i \le n : s_i(o) \ge p_i$ with at least one dimension in which it is strictly better. Thus also in this case we are not allowed to terminate before $F(o) > F(p)$ holds, because otherwise we could miss optimal objects. If an object o has occurred in all lists and we retrieve all objects with minimum scores like in the skylining algorithm (step 3), obviously this object also dominates the virtual object and thus our algorithm would terminate at the same time and would deliver exactly the same correct result set. On the other hand, if $F(o) > F(p)$ holds and $f_i(s_i(o)) := s_i(o)$, then we must have accessed all lists down to $s_i(o) \ge s_i(p_i)$ and thus seen o in all lists. ■

4. Progressive Output of Result Sets

We have seen our algorithm in both special cases to behave like the known optimal algorithms and to handle any arbitrary number and all instances of monotonic objective functions with instance-optimal complexity. Now we enable our algorithm to output result objects not only in a single batch after termination, but *on the fly* as soon as they are found and have been proved to be Pareto-optimal. This successive output of objects essentially reduces the psychological response time. To allow for outputting a result object at the earliest possible point in time, we now investigate which objects could possibly dominate it.

Lemma 1: *Finding dominated objects*

Let f be any monotonic objective function defined over a subset S_f of the n ranked score lists. At any point in time an object o that is accessed in one of the lists of S_f can only be dominated with respect to f by an object w having a strictly better score than o in at least one list in S_f.

Proof:

To be dominated with respect to a single objective function means that $f(w) = f(s_h(w),\ldots,s_l(w)) > f(s_h(o),\ldots,s_l(o)) = f(o)$ with $s_h,\ldots,s_l$ being the score lists of S_f. Since f is monotonic, it follows directly that there has to be a list s_i such that $s_i(w) > s_i(o)$. ■

Lemma 2: *Distinguishing objects by objectives*

Assume that all objects seen by sorted access, are divided into some m sets $K_1,\ldots,K_m$ according to the lists in which they were seen and the objective function that uses these lists as input, i.e. if an object o is accessed by sorted access in list s_i ($1 \le i \le n$) and objective function f_k uses this set as (one of its) input(s), it is added to set K_k. Now let f_k be defined over the score lists $S_{fk} := \{s_h,\ldots,s_r\}$ and let $p_h,\ldots,p_r$ be the minimum scores accessed by sorted access in the lists of S_{fk}. Any object o, for which $f_k(s_h(o),\ldots,s_r(o)) > f_k(p_h,\ldots,p_r)$ holds, can only be dominated by any database object w that is already in the same set K_k.

Proof:

Let o be an object assigned to set K_k and let o be dominated by some object w. Let us further assume that $f_k(s_h(o),\ldots,s_r(o)) > f_k(p_h,\ldots,p_r)$ holds with s_i and p_i defined above, i.e. o dominates object p with score values $p_h,\ldots,p_r$ wrt. objective function f_k. If w would not be in set K_k, it cannot have been accessed by sorted access in any of the lists in S_{fk} yet. Since the score lists in S_{fk} have been accessed down to scores $p_h,\ldots,p_r$, it follows that $s_h(w) \le p_h,\ldots,s_r(w) \le p_r$. Due to lemma 1 w therefore cannot dominate the virtual object p. However, we know w to dominate object o, which in turn dominates the virtual object p, leading to a contradiction. Hence w must be part of K_k. ■

Theorem 3: *Correctness of the progressive output*

Let $f_1,\ldots,f_m$ be m monotonic objective functions, $K_1,\ldots, K_m$ be defined like in lemma 2 and add all objects that have been accessed by sorted access in a certain score list, to each set K_k of all objective functions f_k that are defined over this score list. Then:

a) If in any set K_k there is an object o for which holds $f_k(s_h(o),\ldots,s_r(o)) > f_k(p_h,\ldots,p_r)$ (with $s_h,\ldots,s_r$ and $p_h,\ldots,p_r$ as in lemma 2) and this object o is not dominated by any other member of K_k having a higher or equal score with respect to f_k, we can immediately output o as correct result with respect to our multi-objective retrieval problem.

b) If there is no object w for which holds $F(w) > F(p)$, all objects, also all seen objects z with $F(z) = F(p)$ can immediately be output.

Proof:

a) Accessing an object by sorted access means that due to the sorting of the score lists, we have already seen all objects having better scores in the list. Since each score list only contributes to objective scores if the respective objective function is also defined over that list, after each sorted access we can focus on the respective subset of sets K_k. This way we keep each set K_k as small as possible. Lemma 2 shows that for each single set K_k the condition $f_k(s_h(o),\ldots,s_r(o)) > f_k(p_h,\ldots,p_r)$ is sufficient for object o to be dominated with respect to f_k only by objects in the same set K_k. We thus only have to show that if o is not dominated in any *single* set K_k, it also cannot be dominated with respect to *all* objective functions $F := (f_1,\ldots,f_m)$ and therefore can be correctly output.

For the sake of contradiction let us assume that we have found an object *o* in any set K_k with the above characteristics and let us further assume that though o is not dominated by any object in K_k, it is dominated by another object $x \notin K_k$ with respect to F. Being dominated with respect to F means $F(x) > F(o)$, i.e. $f_i(s_1(x),\ldots,s_n(x)) \geq f_i(s_1(o),\ldots,s_n(o))$ for all $1 \leq i \leq m$ (with a strictly better for at least one i). Thus we also have $f_k(s_1(x),\ldots,s_n(x)) \geq f_k(s_1(o),\ldots,s_n(o))$. Since $f_k(s_h(o),\ldots,s_r(o)) > f_k(p_h,\ldots,p_r)$ and f_k is only defined over score lists $s_h,\ldots, s_r$, we can conclude: $f_k(s_h(x),\ldots,s_r(x)) \geq f_k(s_h(o),\ldots,s_r(o)) > f_k(p_h,\ldots,p_r)$

Therefore using lemma 1 we know that there must be at least one score $s_i(x) > p_i$ among all score lists over which f_k is defined. Since they have been accessed down to $p_h,\ldots,p_r$ we must already have accessed *x* by sorted access in one of the lists and thus *x* would be an element of K_k in contradicting our assumption. Hence *o* is optimal with respect to F, if *o* it is not dominated in any K_k.

b) The second statement is obvious, since if any object *z* with $F(z) = F(p)$ would be dominated by some seen object *w*, this object *w* would also satisfy $F(w) > F(z) = F(p)$. On the other hand *z* cannot be dominated by any unseen object *x*, since $F(z) = F(p) \geq F(x)$. ■

Observation 2: *Earliest possible progressive output* The conditions a) $f_k(s_h(o),\ldots,s_r(o)) > f_k(p_h,\ldots,p_r)$ in any set $K_{k,}$ and b) $F(o) = F(p)$, while there is no *w* with $F(w) > F(p)$, for successive output of results given in theorem 3 a) and b) lead to the earliest possible output of result objects with guaranteed correctness. That means for arbitrary database instances, all monotonic objective functions and any result object *o* our algorithm needs no more accesses (up to an universal multiplicative constant) to correctly return *o* as any other algorithm.

Proof:

For brevity we will just sketch the proof of observation 2; it works along the lines of the proof for theorem 2: If some algorithm A would deliver object *o* before either condition a) holds in some K_k, or condition b) holds, we can construct an object *q* hitherto *unseen* by algorithm A that has exactly the minimum scores $p_1,\ldots,p_n$ (if A had seen *q* it would have accessed objects at least in one score list with sorted accesses down to the respective score of *p* and then it would *not* be better than the round robin strategy of our algorithm up to a multiplicative constant). Since neither a) holds for any k, we get $f_k(s_h(o),\ldots,s_r(o)) \leq f_k(p_h,\ldots,p_r) = f_k(s_h(q),\ldots,s_r(q))$ for all k, nor b) holds, we get that there is a k with $f_k(s_h(o),\ldots,s_r(o)) < f_k(p_h,\ldots,p_r) = f_k(s_h(q),\ldots,s_r(q))$. Thus *q* dominates *o* and *o* would have been output incorrectly by A. Hence either A is incorrect or needs the same number of object accesses to output o than our algorithm (up to a multiplicative constant). ■

Let us now consider how the progressive output scheme works together with our basic multi-objective retrieval algorithm. The next theorem shows that when our termination condition becomes true, we have *already* progressively output *all* correct result objects and thus can immediately discard all other objects.

Theorem 4: *Completeness of output result objects* At the first point in time that any seen object *o* dominates the minimum scores in each list with respect to all objective functions, i.e. $F(o) > F(p)$, and objects have been output successively like stated in theorem 3, the *entire* correct multi-objective result set has already been output

Proof:

Theorem 3 shows that all objects *w* are output, immediately after the condition $f_k(s_1(w),\ldots,s_n(w)) > f_k(p_1,\ldots,p_n)$ has become true in some set K_k. So we have to show that no object, which has not yet been output, can be optimal with respect to F after F(o) > F(p) has become true for some object o. Let p_i be the minimum scores seen by sorted access in each score list. Since we have output all non-dominated objects *w* with $f_i(s_1(w),\ldots,s_n(w)) > f_i(p_1,\ldots,p_n)$ for all i, $1 \leq i \leq m$, we have to check that no object *q* with $f_i(s_1(q),\ldots,s_n(q)) \leq f_i(p_1,\ldots,p_n)$ for *all* i can be optimal with respect to F. However, since we have seen an object *o* dominating the virtual object p, for all $1 \leq i \leq m$ we know $f_i(s_1(q),\ldots,s_n(q)) \leq f_i(p_1,\ldots,p_n) \leq f_i(s_1(o),\ldots,s_n(o))$ and due to the definition of domination there is an index j for which $f_i(s_1(q),\ldots,s_n(q)) \leq f_i(p_1,\ldots,p_n) < f_i(s_1(o),\ldots,s_n(o))$. Thus object *q* is always dominated by object *o* and can not be in the non-dominated result set. ■

Now we are ready to formulate the improved algorithm and demonstrate its output behavior with a short example. We will further adopt the sets K_i to be sorted lists, where every new object is inserted in the right place according to its score with respect to f_i. This does not alter the algorithm's complexity in terms of accesses, but makes the necessary object comparisons easier. We will also use two variables a_i and b_i with each list K_i, such that all objects in K_i having larger scores than a_i with respect to f_i have already been output or discarded as dominated and b_i is the current value of f_i using the lowest scores seen so far as input. That means in every K_i we are done with objects having higher or equal values of f_i than a_i, and we have to consider objects for output having values between a_i and b_i and cannot yet output all objects with values up to b_i (for a detailed discussion of a_i and b_i see [1]). Furthermore we will use two sets R and X containing the output optimal result objects and the discarded dominated objects respectively. Thus we will never scrutinize already output objects or those known to be dominated.

Progressive Multi-objective Retrieval:

0. Given *n* lists of database objects sorted descending by score, *m* monotonic functions $f_1,\ldots,f_m$, two empty lists R (returned objects) and X (dominated objects) and *m* empty sorted lists $K_1,\ldots,K_m$ with variables $a_1, b_1,\ldots, a_m, b_m := 1$.

1. Get an object *o* by sorted access from any list s_i in a round robin fashion

2. For new objects perform random accesses on the other lists and calculate the object's objective scores F(*o*)

3. Create a virtual object *p* assuming the minimum score values that have occurred in each list to be its score values (i.e. $s_i(p)$ is the current minimum score in the *i*-th list) and calculate its objective scores F(*p*)

```
4. For all functions $f_k$ that are defined over $s_i$ do
    4.1. Insert $o$ into $K_k$, if o is not in set X
    4.2. $a_k := b_k$, $b_k := f_k(s_1(p),\dots,s_n(p))$
    4.3. Compare all objects $q \in K_k$ that are not in set R
         and for which $b_k < f_k(s_1(q),\dots,s_n(q)) \leq a_k$ holds, pairwise for domination.
         If any object is dominated, delete it from the list $K_k$ and add it to set X.
    4.4. Compare the remaining objects $q$ that are not in set R for domination
         with all objects $y \in K_k$ with $a_k < f_k(s_1(y),\dots,s_n(y))$.
         If $q$ is dominated by any object $y$ delete it from $K_k$ and add it to set X,
         else output $q$ and add it to set R.
5. If an object $w$ has already been seen for which holds $F(w) > F(p)$, i.e. its objective
   scores are better or equal, but in at least one dimension strictly better than the
   virtual object's, discard all objects that have not yet been output, and terminate the algorithm.
6. Consider all objects $z$ in any of the sets $K_k$ ($1 \leq k \leq m$) for which
   $f_k(s_1(z),\dots,s_n(z)) = b_k$ holds for all k. Output those objects $z$ and add them to set R.
7. Proceed with step 1
```

Before we will give an example that the output behavior of our progressive algorithm effectively enhances the basic version, we will state a short observation on the new algorithm's correctness and instance-optimality.

Observation 3: *Correctness and optimality for the progressive multi-objective retrieval algorithm*
The multi-objective retrieval algorithm with progressive output of result objects is correct and instance-optimal with respect to the object accesses needed and the response time till the delivery of the first result object.

Proof:
Theorem 3 a) + b) and 4 show that the progressive multi-objective retrieval algorithm only delivers correct result objects and does not miss any relevant objects. Since we have not altered the termination condition in step 5, also the instance-optimality of the number of necessary object accesses like shown in theorem 2 still holds. Thus our progressive algorithm is correct and instance-optimal. The last statement follows directly from observation 2. ■

Let us now focus on our example to show how the algorithm works. Again we will use the simple three list traffic information scenario given by example 2 (table 1). Like before we will aggregate the first two lists with an average and just project the third list into objective space. The algorithm runs as before, but now assigns objects accessed in lists s_1 and s_2 to set K_1 and objects accessed in list s_3 to set K_2. Table 2 shows our object accesses (sorted accesses shown bold) whereas table 3 shows our two sets K_1 and K_2. First we make a sorted access on s_1 and add the object o_1 to set K_1, the same happens for o_2 in list s_2. Object o_3 is added to K_2, because s_3 only contributes to f_2, whereas s_1 and s_2 only contribute to f_1 (=f_{eco}). Please note that processing more complex objectives may involve several or even all sets K_k for each object, but our algorithm keeps the K_k as small as possible to reduce necessary comparisons within the K_k for progressive output.

After we have accessed o_5 by sorted access in s_2, we can safely output o_2, because the minimum scores of s_1 and s_2 lead to the threshold $b_1 := f_1(0.94,0.84) = 0.89$ for the first time being strictly smaller than o_2's score of 0.9 with respect to f_1 and o_2 is not dominated by any other object in K_1 (all objects in K_1 have strictly lower scores with respect to f_1). For the same reasons we can safely output o_3 in K_2 after we have accessed o_6. And after accessing o_7 we can output o_4 from K_1 (because it could only be dominated by o_2 in K_1. However o_4's score with respect to f_2 is strictly higher than o_2's; thus it is not dominated by o_2). Like before, o_4 is also the first element dominating the virtual object and thus the algorithm terminates having successively output all optimal objects with respect to our multi-objective query. Table 3 shows the final state with counters $b_1 = 0.87$ and $b_2 = 0.8$. All objects with higher scores have been output.

K_1		
oid	score	
o_2	0.9	
o_4	0.88	←b_1
o_1	0.8	
o_5	0.7	
o_7	0.7	

K_2		
Oid	score	
o_3	0.9	
o_6	0.8	← b_2

prog. Output	
oid	Score
o_2	(0.9, 0.1)
o_3	(0.5, 0.9)
o_4	(0.88, 0.8)

Table 3: Lists K_1 and K_2 and progressive output

5. Experiences in Practical Applications

As we have discussed in section 2 the only ways to handle multiple objectives today are given by complex preference frameworks like given in [7] or [13], since the top k (compensating *all* score lists) and skyline (compensating *no* score lists) approaches only accommodate the extreme cases. However, the actual algorithms given for evaluating complex preference queries up to now always create views containing the non dominated objects by accessing all database objects and compare pairwise for domination. Our algorithm can do essentially better. Consider for instance our running example: using basic constructors, we would have three preferences P_1 (shortest route), P_2 (least congestion) and P_3 (best weather) given by our basic score lists s_1, s_2, s_3. Then we would have to evaluate a complex expression averaging the first two preferences and merging this new list with the third list s_3 in a Pareto optimal fashion. Since our algorithm has transparent access to score lists, it does not have to calculate a complex view over the base relations accessing all the database objects. It focuses on the *necessary* accesses only, using $F(o) = ((s_1(o)+s_2(o)/2, s_3(o))$ and with an instance optimal number of object accesses nevertheless always delivers the correct result set. In the following we will focus on the complexity of the result sets and the connection with the objective functions used.

5.1. Bounding of Multi-objective Result Complexity

When using the multi-objective algorithm for practical applications in information systems, queries can quickly involve a rather high number of objectives, i.e. dimensions. However, Pareto optimal sets tend to grow exponentially with increasing numbers of dimensions [4] (often referred to as 'curse of dimensionality'). Thus, we will have to investigate this relevant problem also for multi-objective retrieval and see how multiple objectives affect the size of the subsequently retrieved result set.

Let us first assume that all objective functions are defined over a *disjoint* set of score lists, e.g. f_1 is defined over $s_1,\ldots,s_k$, f_2 is defined over $s_{k+1},\ldots,s_r$, and so on. Then the complexity of the result set is always reduced from growing with the number of score lists *n* to only growing with the smaller number of objective functions *m*. The importance of this can be seen in our practical experiments in figure 2. Assuming statistically independent uniform score distributions, figure 2 shows the actual size of average result sets in our tests for different numbers of dimensions (3, 5, 10) and different database sizes. As we can see the result sets have manageable sizes only in case of low dimensionality, but in cases of only ten dimensions, we e.g. already have to return up to 50% of all database objects in the result set for 10,000 objects. This behavior is due to Pareto optimality being a rather weak concept and producing lots of incomparable result objects with growing numbers of dimensions.

A dimensionality reduction with suitable objective functions is therefore needed. Moreover, the strong assumption that all objective functions work on a disjunctive set of score lists generally does not hold leading to a certain degree of correlation in the results of the objective functions, which in practical experiments has been shown to further reduce results sizes, see e.g. [5]. So if the number of objective functions *m* is smaller than the number of score lists *n*, we can always assume the result size to be reasonably small. But what if a user adds objectives over a certain number of score lists? Assuming a sensible characteristic of the set of objective functions, in the following observations we will show that in this case not only does the result set does not grow exponentially with the number of objectives, but is always limited by the size of the basic Pareto set over the score lists. Let us therefore state the following definition:

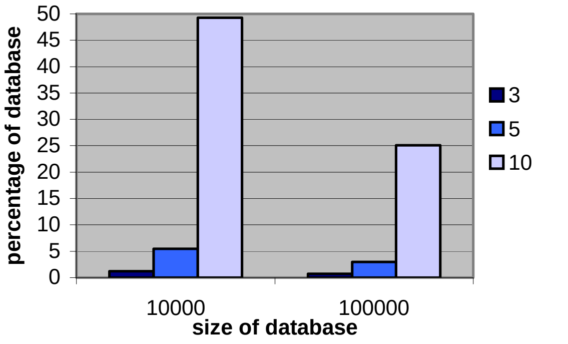

Fig. 2: Result sizes for different numbers of lists

Definition 1: *Strict monotonic set of functions*
As set of *n* objective functions $F := (f_1,\ldots,f_n)$ is called strict monotonic, if $(o < w) \Rightarrow (\forall\ 1 \leq i \leq n : f_i(o) \leq f_i(w) \wedge \exists\ 1 \leq k \leq n : f_k(o) < f_k(w))$.

Examples for basic strict monotonic functions include e.g. the large group of weighted sums like the average, geometrical means, etc. To obtain a set of strict monotonic functions it is sufficient that only a subset of the functions is strict monotonic. However, the subset then already has to be defined over all of the score lists s_i. Using that, let us now state a helpful bounding property.

Observation 4: *Bounding of multi-objective results*
The size of the multi-objective result set *R* for a strictly monotonic set F of *m* objective functions $f_1,\ldots,f_m$ over *n* ranked score lists $s_1,\ldots,s_n$ is always upper bounded by the size of the set of Pareto-optimal objects *P* over the ranked score lists $s_1,\ldots,s_n$.

Proof:
We will show the set inclusion $R \subseteq P$ and therefore the size of *P* is always an upper bound for the size of *R*. Assume for the sake of contradiction there would exist an object *o* in *R*, which is not in *P*. Since $o \notin P$, we have to conclude that there is some object $w \in P$ with $s_i(o) \leq s_i(w)$ $(1 \leq i \leq n)$ and there is a $1 \leq k \leq n$ for which $s_k(o) < s_k(w)$. Since the objective functions $f_1,\ldots,f_m$ are all monotonic we get $f_i(s_1(o),\ldots,s_n(o)) \leq f_i(s_1(w),\ldots,s_n(w))$ $(1 \leq i \leq m)$ and thus due to the set F being strictly monotonic *w* would also dominate *o* with respect to $f_1,\ldots f_m$. ■

Theorem 5: *Result set size and object accesses*
The cardinality of result sets for multi-objective retrieval over $F = (f_1,\ldots,f_m)$ grows only with increasing $\min(n, m)$. Moreover, computing the multi-objective result set never needs more object accesses than computing the respective skyline (i.e. the Pareto-optimal set).

Proof:
For the growth of the result sets cardinality, we have to distinguish two cases. If $m < n$ and all score lists are used by at least one objective function, there has to be at least one f_i that is defined over more than one score list, i.e. aggregating some score lists into a single one. Thus the Pareto-optimal set's cardinality over $f_1,\ldots,f_m$ can only be influenced by *m* score lists and thus grows with *m*. If $m \geq n$ observation 4 shows that the set of optimal result objects always is a subset of the basic set of all Pareto-optimal objects over *n* score lists, growing only with *n*.

For the optimality of object accesses we know from observation 1 that computing skylines is a special case of multi-objective retrieval and thus at the latest if the skyline computation terminates by accessing some object *o* with $(s_1(o),\ldots,s_n(o)) > (p_1,\ldots,p_n)$, due to F being strict monotonic also F(o) > F(p) holds and also our multi-objective retrieval algorithm would have terminated. ■

Though expensive Pareto set computations can be avoided in most practical cases, still high dimensional problems without suitable aggregation functions will exist. Also in these cases our algorithm returns the correct result set with an instance-optimal number of accesses.

5.2. Heuristics to Speed Up the Average Runtime

Though we have shown our algorithm to be instance-optimal in terms of object accesses especially in cases of skewed data distributions the average runtime can be improved up to a factor of n (cf. theorem 2) by an advanced control flow deciding which score list to access next by sorted access (which can easily be integrated into step 1 of the algorithm). Moreover, by ascertaining that once in a while *every* list will be accessed (e.g. after at most x accesses) we can even make sure that a greedy control flow will still guarantee an instance-optimal algorithm (however with a somewhat higher optimality ratio, cf. [10]). Adopting a better control flow than round robin strategy relies on making the necessary condition $F(o) > F(p)$ hold more quickly for some object o and virtual object p. Since the virtual object's score values are given by the minimum in each list, a virtual object with lower objective scores improves the chances that one of the already accessed objects is able to dominate it. Thus we will focus on fostering a quick decrease in the virtual object's objective scores. Assuming that all objective functions are partially differentiable, we will have to assess the influence that a sorted access on any list will have on the virtual object's objective scores. Let us assume that a sorted access on the i-th list will decrease the minimum score seen in that list by d_i. Then the objective scores of p will in each objective function f_k decrease by $(\partial f_k / \partial s_i)\, d_i$ $(1 \le k \le m)$. Since we want to decrease p's objective scores most quickly, we choose the list for the next sorted access that maximizes the sum of decreases in *all* f_k. There may be lists and objective functions discriminating well between objects and show rapidly declining score distributions, whereas other characteristics may not discriminate too well and the scores in the respective lists may change only slowly or even remain constant. For rapid decreases in the virtual object's scores and thus quick termination, we are interested in accessing highly discriminating lists first.

Since we cannot know how much scores in the *next* access on each list will decrease before we make an actual access on this list, we need another heuristic to estimate d_i. A similar approach in [11] assumes that the recent development in each list gives a sufficient estimation what will happen next. If we use the difference between the current minimum and a score that has been accessed before h accesses in that list, we will usually get a good estimation how scores in the list will decline. Let q be an (virtual) object that is given by the scores occurring h accesses before the current minimum in each list. We will estimate d_i by $d_i := 1/h\ (s_i(q) - s_i(p))$. With these simple heuristics we will define an indicator Δ_i for each list that represents the expected gain by accessing this list:

$$\Delta_i := 1/h\ \Sigma(\partial f_k / \partial s_i)\ (s_i(q) - s_i(p)) \qquad (1 \le k \le m)$$

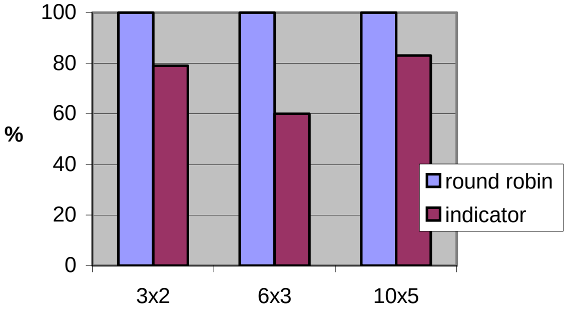

Fig. 3: Improved number of object accesses

We have run practical experiments on statistical averages of run times for different score lists with skewed data distributions and the average of arbitrary subsets of these score lists as objective functions. We basically have compared the round robin strategy in our basic algorithm against the use of our indicator technique in step 1 for choosing lists. We then measured the average numbers of object accesses (in %) in a database with 10000 objects for three different scenarios over a database with 10% skew in the score distributions. The different scenarios focus on 3 lists with two objective functions (3x2), 6 lists with 3 objective functions (6x3) and 10 lists with 5 objective functions (10x5). Figure 3 shows the respective improvement factors for these scenarios for the basic and improved algorithm. Depending on the scenario improvement factors even for these simple examples range between 20% and 40% and grow with more skew, less objective functions and more score lists to combine.

6. Summary and Outlook

This paper addresses the problem of multi-objective retrieval in database query processing. We have in deep discussed applications for this retrieval paradigm and the meaning of 'overall best' objects from a database perspective. Multi-objective retrieval is especially useful for personalization problems, where multiple user preferences have to be taken into account, and one has to compromise between certain desired characteristics of database objects to deliver high quality results. However, up to now only for two extreme cases of such retrieval, namely top k retrieval and skyline queries, efficient algorithms have been investigated. Handling cases involving several distinct objectives, still needs to access and compare all database objects. We have presented a novel multi-objective retrieval algorithm and proved that it always retrieves a correct result set and uses only an instance-optimal number of object accesses. Moreover, it contains the respective optimal algorithms for top k retrieval and skylining as special cases. We have subsequently enhanced it by allowing for a successive output of result objects at the earliest possible time while the algorithm is still running.

Finally we have addressed preliminary practical experiences with applications of our algorithm. Our algorithm can be easily integrated into practical personalization frameworks or relational query processing. Concerning the manageability of query results, we have also shown that the cardinality of the multi-objective result set is bounded by the size of Pareto-optimal sets over the minimum of the number of score lists and objective function limiting down the set's cardinality in most practical cases. Implementing an advanced control flow we then addressed how to save additional object accesses in the case of skewed data distribution by focusing on the most prominent objects at an early time. Based on the practical experiences gained with our algorithm our future work will focus on the problem of high dimensional multi-objective sets, where the quality even between optimal objects has to be assessed. Sophisticated sampling strategies (cf. e.g. [1]) that give users an overview of the expected result set, and subsequent refinement of the query may be techniques employed to tackle this problem.

7. References

[1] W.-T. Balke, U. Güntzer, J. Zheng. Efficient Distributed Skylining for Web Information Systems. In *Proc of the Int. Conf. on Extending Database Technology (EDBT'04)*, Heraklion, Crete, Greece, 2004.

[2] W.-T. Balke, U. Güntzer, W. Kießling. On Real-time Top k Querying for Mobile Services, In *Proc. of the Int. Conf. on Cooperative Information Systems (CoopIS'02)*, Irvine, USA, 2002.

[3] W.-T. Balke, W. Kießling, C. Unbehend. Personalized Services for Mobile Route Planning: A Demonstration. In *Proceedings of the Int. Conf. on Data Engineering (ICDE 2003)*, Bangalore, India, 2003.

[4] J. Bentley, H. Kung, M. Schkolnick, C. Thompson. On the Average Number of Maxima in a Set of Vectors and Applications. In *Journal of the ACM (JACM)*, vol. 25(4) ACM, 1978.

[5] S. Börzsönyi, D. Kossmann, K. Stocker. The Skyline Operator. In *Proc. of the Int. Conf. on Data Engineering (ICDE'01)*, Heidelberg, Germany, 2001.

[6] N. Bruno, L. Gravano, A. Marian. Evaluating Top-k Queries over Web-Accessible Databases. In *Proc. of the Int. Conf. on Data Engineering (ICDE'02)*, San Jose, USA, 2002.

[7] J. Chomicki. Querying with intrinsic preferences. In *Proc. of the Int. Conf. on Extending Database Technology (EDBT'02)*, Prague, Czech Republic, 2002.

[8] P. Ciaccia, M. Patella. The M2-tree: Processing Complex Multi-Feature Queries with Just One Index. In *Proc. of the Int. DELOS Workshop on Querying Digital Libraries*, Zurich Switzerland, 2000

[9] K. Deb. Multi-Objective Optimization Using Evolutionary Algorithms. J. Wiley & Sons, London, 2001

[10] R. Fagin, A. Lotem, M. Naor. Optimal Aggregation Algorithms for Middleware. *ACM Symp. on Principles of Database Systems (PODS'01)*, Santa Barbara, USA, 2001.

[11] U. Güntzer, W.-T. Balke, W. Kießling. Optimizing Multi-Feature Queries for Image Databases. In *Proc. of the Int. Conf. on Very Large Databases (VLDB'00)*, Cairo, Egypt, 2000

[12] R. Keeney, H. Raiffa. Decisions with Multiple Objectives: Preferences and Value Tradeoffs. Wiley, 1976

[13] W. Kießling. Foundations of Preferences in Database Systems. In *Proc. of the Int. Conf. on Very Large Databases (VLDB'02)*, Hong Kong, China, 2002

[14] D. Kossmann, F. Ramsak, S. Rost. Shooting Stars in the Sky: An Online Algorithm for Skyline Queries. In *Proc. of Conf. on Very Large Data Bases (VLDB'02)*, Hong Kong, China, 2002

[15] H. Kung, F. Luccio, F. Preparata. On Finding the Maxima of a Set of Vectors. *Journal of the ACM*, vol. 22(4), ACM, 1975

[16] M. Lacroix, P. Lavency. Preferences: Putting more Knowledge into Queries. In *Proc. of the Int. Conf. on Very Large Databases (VLDB'87)*, Brighton, UK, 1987

[17] A. Leubner, W. Kießling. Personalized Keyword Search with Partial-Order Preferences. In *Proc. of Brazilian Symp. on Databases (SBBD'02)*, Gramado, Brazil, 2002.

[18] A. Motro. VAGUE: A User Interface to Relational Databases that Permits Vague Queries. In *ACM Transactions on Office Information Systems (TOIS)* 6(3), 1988

[19] F. Naumann, U. Leser, J. Freytag. Quality-driven Integration of Heterogenous Information Systems. In *Proc. of Conf. on Very Large Data Bases (VLDB'99)*, Edinburgh, UK, 1999

[20] S. Nepal and M. Ramakrishna. Query processing issues in image (multimedia) databases. In *Proc. of Int. Conf. on Data Engineering (ICDE'99)*, Sydney, Australia, 1999

[21] M. Ortega, Y. Rui, K. Chakrabarti, et al. Supporting ranked boolean similarity queries in MARS. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, Vol. 10 (6), 1998

[22] C. Papadimitriou, M. Yannakakis. Multiobjective Query Optimization. In *Proc. of the ACM Symp. on Principles of Database Systems (PODS'01)*, Santa Barbera, USA, 2001

[23] D. Papadias, Y. Tao, G. Fu, B. Seeger. *An Optimal and Progressive Algorithm for Skyline Queries.* In *Proc. of the Int. ACM SIGMOD Conf. (SIGMOD'03),* San Diego, USA, 2003.

[24] K. Tan, P. Eng, B. Ooi. Efficient Progressive Skyline Computation. In Proc. *of Conf. on Very Large Data Bases (VLDB'01)*, Rome, Italy, 2001.