Getting Prime Cuts from Skylines over Partially Ordered Domains

Wolf-Tilo Balke[1], Ulrich Güntzer[2], Wolf Siberski[1]

[1]Forschungszentrum L3S
Universität Hannover
Appelstr. 4
30167 Hannover
balke@l3s.de
siberski@l3s.de

[2]Institut für Informatik
Universität Tübingen
Sand 13
72076 Tübingen
ulrich.guentzer@informatik.uni-tuebingen.de

Abstract: Skyline queries have recently received a lot of attention due to their intuitive query formulation: users can state preferences with respect to several attributes. Unlike numerical preferences, preferences over discrete value domains do not show an inherent total order, but have to rely on partial orders as stated by the user. In such orders typically many object values are incomparable, increasing the size of skyline sets significantly, and making their computation expensive. In this paper we explore how to enable interactive tasks like query refinement or relevance feedback by providing 'prime cuts'. Prime cuts are interesting subsets of the full Pareto skyline, which give users a good overview over the skyline. They have to be small, efficient to compute, suitable for higher numbers of query predicates, and representative. The key to improved performance and reduced result set sizes is the relaxation of Pareto semantics to the concept of weak Pareto dominance. We argue that this relaxation yields intuitive results and show how it opens up the use of efficient and scalable query processing algorithms. Assessing the practical impact, our experiments show that our approach leads to lean result set sizes and outperforms Pareto skyline computations by up to two orders of magnitude.

1. Introduction

Due to the ever growing volume of database content and the personalization needs in information searches, human preferences already play an essential part in today's information systems. This is because mere SQLstyle queries only too often produce empty or too numerous results. First approaches at cooperative databases as those by [LL87, Mo88], handled user queries that retrieved empty results with respect to a database instance by automatic relaxation of query predicates. Using score values to express the utility of database objects with respect to a query, cooperative queries come in various flavors:

- **Top-k queries** (see e.g. [GBK00, FLN01]) have shifted retrieval models from exact matching of attribute values to the notion of best matching database objects. Top-k models rely on basic scorings of objects for each query predicate and a utility function to aggregate the objects' total scores.

- **Skyline queries** extend this principle to cases where still score-based preferences exist for each query predicate, but no utility function is a-priori known to compromise between predicates (see e.g. [BKS01,TEO01,PTF03,BGZ04]). Skyline approaches adopt the principle of Pareto optimality, i.e. only those objects are returned, where no object exists in the database having better or equal predicate values.
- **Multi-objective retrieval** [BG04] finally allows for the interleaved evaluation of arbitrary compositions of skyline and top-k queries with proven instance-optimal complexity.

Especially the skyline paradigm has proven its usefulness in a variety of applications (e.g., digital item adaptation [KB06] or location-based services [HJ04]), since users generally cannot be expected to provide sensible weightings for a utility function. But while score-based approaches generally allow for efficient query evaluation, their expressiveness in terms of human user preferences remains rather limited, cf. [Fi99]. With the use of preferences modelled as strict partial orders with intuitive "I like A better than B" semantics ([Ch02, Ki02]), this lack of expressiveness was remedied at the price of more expensive query evaluation. A first evaluation algorithm of such partial order preference queries was given only recently by [CET05]. Also here the Pareto principle was used for evaluating queries involving several partial order preferences:
- In [Ki02] and [CET05] a strong Pareto dominance principle called **Pareto accumulation** is used: an object has to be better or identical in all attribute values for the query predicates, and strictly better in at least one to dominate another object.
- In contrast [Ch03] and [BG05] propose a weak Pareto dominance principle called **Pareto composition**, where an object's attribute values has to be better, identical or *incomparable* in all predicates, and strictly better in at least one to dominate another object.

The Pareto principle extends querying capabilities and the result set contains *all* possible best database objects with respect to arbitrary utility functions. On the other hand Pareto sets grow exponentially in size with increasing numbers of preferences [Be78]. Thus, typical tasks during the query process (like query refinement) rather need a good (and efficiently computed) overview over skylines.

For instance, [KRR02] presents an online algorithm where users can influence the order in which skyline objects are produced. Eventually the entire skyline is calculated, but at every stage of the computation users can provide a direction where most relevant objects might be expected. The work in [BZG05] also relies on user interaction, by presenting the user with a representative sample of the expected skyline set, and then exploiting user feedback to elicit an appropriate utility function for the final result ranking. [KP05] proposes to cover the skyline set with ε-spheres where each center of a sphere is a representative for all skyline objects within a distance of at most ε. This set of representatives is subsequently returned to the user. However, the computation of an ε-sphere cover was shown to be NP-hard for more than 2 independent predicates. Moreover, the calculation of such approximations always needs expensive computations of the entire skyline. These approaches only focus on total-order preferences: all objects can be compared in each predicate, which makes combinations of different predicates simple. Due to the indifference property in partial order preferences the Pareto combination leads to even bigger result sets: if an object is incomparable to other objects with respect to just a single preference, it still is Pareto-optimal and thus part of the skyline, even if it is the least preferred object with respect to all other preferences. In practical applications such incomparability often occurs: users can be indifferent between items and very rarely model preference relations between all possible attribute values for a query predicate anyway.

Recent research in [Ki05] has started to combat such indifference in partial order preferences by means of 'substitute values'. The substitute values (SV) semantics assigns equal usefulness to some incomparable values. Still, this semantics only remedies a small number of cases and is comparable in size and evaluation time to the complete skyline.

Our goal is somewhat more ambitious. We want to efficiently provide 'prime cuts' of the skyline that can be used in an interactive query process. These prime cuts have to be both manageable in size and representative of the Pareto skyline. In this paper we present an innovative algorithm for the efficient computation of such prime cuts which relies on weak Pareto dominance, as defined in [BG05]. Weak Pareto dominance changes the preference semantics to an even higher degree than SV semantics. The resulting 'restricted skyline', i.e., all objects not weakly Pareto dominated, contains intuitively appealing objects, can be derived surprisingly efficient, and thus will deliver our prime cuts. The contribution of our approach is therefore twofold:

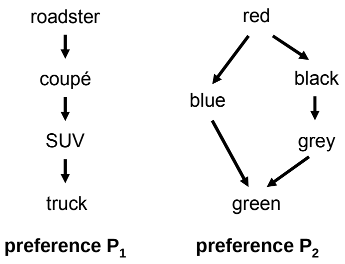
- Restricted skylines derive manageable subsets of the partial order skylines (useful e.g. as a preview, or for query refinement) by taming the effects of incomparability. Our evaluation shows that sizes of restricted skylines are usually lean.
- Our approach allows to efficiently approximate these restricted skylines without having to compute the entire Pareto set first. Query processing relies on progressive iteration of ranked result lists for each predicate and allows for pruning.

In the following we will give a motivating scenario for partial order skylines and explain the semantics of the restricted skyline set. We will present the efficient evaluation algorithm for restricted skylines and perform extensive experiments to prove the practical applicability.

2. Weak Pareto Dominance and Restricted Skylines

The following example will illustrate Pareto skylines and lead to the basic notion of weak Pareto dominance.

Example 1: Given preferences P1 on car types and P2 on colors in Figure 1 and the following database instance: a green roadster, a black coupé, a blue SUV, a yellow truck and a pink limousine. None of them are dominated. The green roadster is maximal in P_1. It does not dominate the black coupé, because black color is preferred over green. Furthermore, the user is indifferent between black and blue cars, thus the blue SUV is



color	type
black	coupé
blue	SUV
green	roadster
pink	limousine
yellow	truck

database instance

Figure 1. Partial order preference example

not dominated by the black coupe, nor dominated by the green roadster because of P_2. Though the yellow truck has the worst car type, the user has not given any judgment on its color, thus making it incomparable. Finally, the pink limousine is completely incomparable to all other objects. Thus, the Pareto skyline contains all five elements.

Using the normal definition of Pareto sets, in Example 1 the entire database would have to be retrieved and returned to the user. Since a user usually is interested in refining queries according to the most promising result objects, retrieving a sophisticated selection from the skyline is a far more cooperative behavior. Our restricted skyline is such a selection. But on what grounds can we select 'better' objects from the full Pareto skyline?

Generally speaking, skyline queries are only sensible if no ordering or weightings between individual predicates are provided. Otherwise utility-based ranking schemes such as top-*k* queries would be far more efficient to use. Pareto sets are designed to consist of all optimal objects with respect to all possible utility functions. Therefore, selecting a subset of the skyline will always ignore objects that are nevertheless optimal for some utility function. In other words, any selection will consider some utility functions as being more probable than others. Such an assessment has to be based on heuristics. We rely on the heuristic that all user preferences should be relaxed evenly and as little as possible, i.e. the relaxation scheme should be fair. In any case, a selection doesn't have to be the final result set. If it can be computed reasonably fast and yields manageable result sets, it can also be used as a good starting point for focused searches such as the online algorithm in [KRR02] or the feedback algorithm in [BZG05]. Since our selection relies on weak Pareto dominance, we will formalize its semantics in the following definition (cf. Pareto composition in [Ch03]):

Definition 1: (weak Pareto dominance)
Let O be a set of database objects and $x, y \in O$. An object x is said to *weakly dominate* object y with respect to partial order preferences $P_1, \ldots, P_n$, if and only if there is an index i $(1 \leq i \leq n)$ such that x dominates y with respect to P_i and there is no index j $(1 \leq j \leq n)$ such that y dominates x with respect to P_j. That means, with $>_P$ denoting the domination with respect to partial order P:

$$x \text{ weakly dominates } y \iff \exists\, i\ (1 \leq i \leq n)\text{: } x >_{P_i} y \wedge \neg\exists\, j\ (1 \leq j \leq n)\text{: } y >_{P_j} x$$

We call the set of all non-weakly-dominated objects the 'restricted' skyline. Please note that for total order preferences, weak and strong Pareto dominance coincide, because

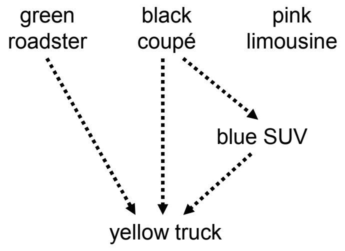


Figure 2. Sample weak dominance graph

there are no incomparable objects. Let us reconsider our example and see what changes, if we restrict the skyline set using weak Pareto dominance.

Example 1 (cont.): Consider the objects from above under the notion of weak Pareto dominance (Figure 2). There is still no weak dominance relation between the green roadster, and the black coupé, because black color is preferred to green, but a roadster is deemed better than a coupé. However, both of them now weakly dominate the yellow truck and it can be removed in the restricted skyline. Removing the yellow truck seems indeed a very intuitive thing to do, because P_1 tells us that everything is better than a truck and the user, although voicing explicit color preferences, did not express his/her opinions on yellow cars. Moreover, we have to take a closer look at the relation between the black coupé and the blue SUV. The user is indifferent between both colors. But the black coupé fits his/her car type wishes to a higher degree, hence is probably more desirable. The weak dominance relation reflects this semantics: the blue SUV is weakly dominated by the black coupé and can be removed. Please note that the pink limousine with incomparable predicate values only is still not dominated by anything and will thus also be part of the restricted skyline. This reflects the notion that an item may be desirable, even if a user was not aware of it when formulating the query.

In the end, the result size in our small example is almost halved and only less intuitive candidates have been pruned. Our work in [BG05] shows that restricted skylines are a proper subset of the normal skyline, i.e. the strong Pareto set. The same applies to the substitute values skyline, as shown in [Ki05]. Finally, it can be shown that the restricted skyline is always a subset of the SV-skyline.

3. Efficiently Computing Restricted Skylines

Unlike numerical skylines, any partial order algorithm needs to handle object incomparability. This makes algorithms on total orders (such as NN [KRR02] and BBS [PTF03]) unsuitable. In contrast we rely on a scheme using topologically ordered lists: for each query predicate a list of all database objects sorted according to the respective user preference is created. Incomparability can be resolved by exploiting the level order of the preference. The algorithm's main challenge is to determine, if all relevant objects have already been seen. In each query evaluation our algorithm therefore first computes possible value combinations (so-called *l*-cuts), which guarantee safe pruning: if a set of

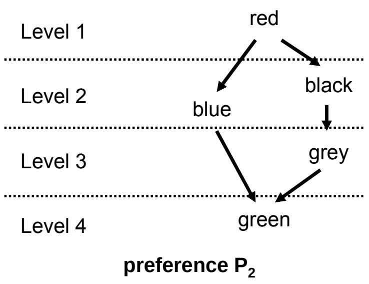


Figure 3. Level order example

objects instantiate any *l*-cut no relevant object can exist in the tails (higher than level *l*) of the sorted lists. The creation of the sorted lists and the calculation of the pruning thresholds are only dependent on preference size, not on database size, and therefore fast to compute.

3.1 Level Order for Partial Order Preferences

For pruning, we have to arrange for sorted access to objects for each query predicate: possibly relevant objects should be returned earlier than rather irrelevant objects. To create a proper sorting from the given partial preference orders, we use a simple breadth first topological ordering defining 'levels':

Definition 2: (level order)
Let *P* be a partial order preference. A value *v* is said to *belong to level l* or *level*(*v*) = *l* with respect to *P*, if and only if the longest path from any maximum attribute value in P to *v* consists of (*l* - 1) edges. Values not explicitly expressed in P belong to level 1. We denote the set of all values in level *l* as $level_l := \{v \mid level(v) = l\}$.

Analogously, a database object *x* is said to be in level *l* with respect to *P*, iff its attribute value is in level *l*.

This notion of levels imposes an intuitive sorting: all maximum (i.e. non-dominated) objects of *P* are on level 1, all objects that are only dominated in *P* by maximum objects are on level 2, and so on. We call this order *level order*. In the special case of numerical or total order preferences the level corresponds to each object's rank, if objects with identical scores/attribute values are considered to have equal rank. But for partial orders this level order has another nice property:

Lemma 1: (level order domination)
Let *O* be a set of database objects and $x, y \in O$. Then object *x* can only dominate object *y* with respect to a partial order preference *P*, if $level(x) < level(y)$ with respect to *P*.

Proof: If *x* dominates *y* there is a path of length $q > 0$ from *x* to *y* in *P*. Thus it directly follows from the definition of levels by longest paths in Definition 2, that:

$$level(x) < level(x) + q \leq level(y). \quad \blacksquare$$

Though objects can only be dominated by objects in smaller levels, due to the partial order semantics they do not *have to* be dominated by all objects in these levels, but can also be incomparable. For example, blue cars are in a smaller level than grey cars for our preference P_2, although both are incomparable (see Figure 3). In the following we will assume all database objects to be accessed in level order for each preference.

Note that it is not necessary to compute a complete object index based on the level order for each incoming query. Instead, the database maintains object sets clustered by value, i.e., the sets of objects sharing the same value for a predicate. Then, creating a list in level order just means to sort references to these sets, not to sort all database objects. Since user preferences are typically rather small, producing level orders is fast even for large databases.

3.2 Identifying the Pruning Thresholds

In the last section we have defined a sorted list of objects for each predicate. For pruning we introduce the concept of *l*-cuts. While iterating over the lists, we have to check whether all relevant (i.e. not weakly dominated) objects have been accessed already.

Definition 3: (*l*-cut of preference orders)
For a partial order preference P and natural number l, a subset of values $C \subseteq P$ is called *l-cut,* if

(a) $\forall\, v \in C : level(v) \leq l$

(b) $\forall\, (w \in P \backslash C)\ \exists\, v \in C : v >_P w$

A set of database objects D forms an instance of an *l*-cut C if for each $v \in C\ \exists\, o \in D$: o has attribute value v. An *l*-cut C is *minimal*, if no subset $C' \subset C$ is an *l*-cut.

The intuitive meaning of *l*-cuts is to form sets of attribute values that if instantiated by database objects, dominate all object values beyond the *l*-th level. Every completely instantiated level of values forms a trivial *l*-cut. But generally *l*-cuts will be much smaller, and in the following we only need to consider minimum *l*-cuts.

Example 1 (cont.): Every single red car is instance of a 1-cut with respect to P_2. A 2-cut is instantiated by any pair of a blue and a black car. Regarding preference P_1, every roadster is instance of the 1-cut, every coupé instantiates a 2-cut, and so on.

For efficient pruning in our skyline evaluation we have to allow for quick tests whether a set of objects instantiating an *l*-cut has already been accessed. Hence, our first step in query evaluation is to compute all minimal *l*-cuts for each preference dimension. If we later find some object set instantiating any such cut we have found a pruning threshold. We now present a simple way to calculate minimal *l*-cuts. We first split the preference graph into levels, according to Definition 2:

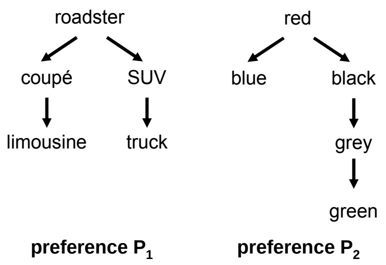
Algorithm 1 (calculating attribute value levels)

```
0.  Select $level_1$ as the set of all maximum attribute values in a preference graph P, i.e.
    all attribute values that are not dominated by any other attribute value.
    $l := 1$
1.  $level_{l+1} := \emptyset$
2.  While there are attribute values in $level_l$ do
    2.1. Consider the next attribute value $x$ in $level_l$
    2.2. For each attribute value $y$ directly dominated by attribute value $x$ with respect
         to P do
         2.2.1. If $y \notin level_0 \cup \ldots \cup level_{l+1}$,
                    then $level_{l+1} := level_{l+1} \cup \{y\}$
         2.2.2. If $y \in level_j$ for some $j \leq l$,
                    then remove $y$ from $level_j$ and set
                    $level_{l+1} := level_{l+1} \cup \{y\}$
3.  If $level_{l+1}$ is not empty, set $l := l+1$ and proceed with step 1.
```

From these level sets, we can now determine minimal *l*-cuts. Obviously, each complete set $level_l$ is a cut candidate, because all objects having attribute values in $level_j$ with $l < j$ are dominated by some object having an attribute value from set $level_l$. Moreover, if we replace some cut element by any object dominating that cut element, the resulting set still forms a cut. Thus, to find all possible cut candidate value sets, we have to systematically enumerate all possible replacements. For this purpose, we first build a cut candidate value set from each complete $level_l$ and then exhaustively replace attribute values by dominating values. Finally we remove redundant values to identify minimal cuts.

Algorithm 2 (calculating minimal cut value sets)

```
0.  Given $n$ sets of attribute values $level_1, \ldots, level_n$ as output by algorithm 1 and
    initialize $candidates_1, \ldots, candidates_n := \emptyset$, $replace_1, \ldots, replace_n := \emptyset$ and
    $minimalcuts_1, \ldots, minimalcuts_n := \emptyset$.
1.  For $l := 1$ to $n$ do
    1.1. If $level_l \notin candidates_l$
             then $candidates_l := candidates_l \cup \{level_l\}$
    1.2. For $j := 1$ to $|level_l|$ do
         1.2.1. Consider the $j$-th attribute value $a_j$ in an enumeration of $level_l$ and
                initialize
                $replace_j := a_i$
         1.2.2. For each $y$ with $a_j <_P y$ and $y \notin replace_j$ do $replace_j := replace_j \cup \{y\}$
    1.3. Generate all possible combinations $\{x_1, \ldots, x_{|level\ l|}\}$ with $x_l \in replace_l$ and in
         each combination remove redundant attribute values, i.e. duplicates and values
         dominated by another value in the set.
         $candidates_l := candidates_l \cup \{x_1, \ldots, x_{|level\ l|}\}$
    1.4. Consider all candidate sets $cand$ in $candidates_l$ and if no subset of $cand$ is in
         $candidates_l$, $mincuts_l := mincuts_l \cup \{cand\}$
```



color	type
blue	coupé
blue	SUV
green	SUV
grey	limousine

database instance

Figure 4. False positives due to pruning

Algorithm 2 is exponential in the size of the partial order preference. However, this size is typically rather small, and *l*-cut computation is always independent of the actual database size. Therefore, query processing efficiency is dominated by the actual skyline computation described in the next subsection. Please not that we nevertheless include the cost of the minimal *l*-cut computations in the query processing times in all our experiments.

3.3 Correctly Pruning Database Objects

In each preference we identified all minimal *l*-cuts. Now we are ready to present a way for pruning irrelevant parts of the database without missing elements of the restricted skyline. This is the major component needed to build an efficient evaluation algorithm for partial order preference queries under the weak Pareto dominance paradigm. The following theorem will show a sufficient condition to correctly prune database objects:

Theorem 1: (absence of false negatives)
Let *O* be a set of database objects and $S_1,\ldots, S_n$ be level-ordered lists of *O* with respect to partial order preferences $P_1,\ldots,P_n$. Let $o_1,\ldots,o_k \in O$ and assume that $o_1,\ldots, o_k$ have already been accessed in all level ordered lists and $\{o_1,\ldots, o_k\}$ form an *l*-cut with respect P_i for some numbers *i* and *l*. Then no object that for all $1 \leq j \leq n$ occurs on a higher level than *l* in S_j can be part of the restricted skyline.

Proof: Let $\{o_1,\ldots, o_k\}$ be as defined above and $u \in O$ be an object that has not yet been accessed in any P_j $(1 \leq j \leq n)$. For the sake of contradiction we will assume that object *u* belongs to the restricted skyline set, i.e. it is not weakly dominated by any other object. Since $\{o_1,\ldots, o_k\}$ form an *l*-cut in P_i, *u* has to be dominated by some object o_m $(1 \leq m \leq k)$ with respect to P_i. Because we have assumed *u* to be not weakly dominated by any object, there has to be at least one preference where *u* dominates o_m Since o_m has already been accessed in all P_j and u has not yet been accessed with respect to any P_j, its level $level_j(u) \geq level_j(o_m)$. Now according to Lemma 1 *u* cannot dominate o_m in any preference and thus must be weakly dominated. This contradicts the assumption of *u* being part of the restricted skyline. ■

Now we know that unseen objects can never be part of the restricted skyline and can be correctly pruned after we have a completely known set of objects that instantiates an *l*-cut. Unfortunately, due to the intransitivity of weak Pareto dominance, in some rare cases an unseen object could still weakly dominate a member of the restricted skyline candidate set, thus resulting in a false positive.

Example 3: Given preferences P_1 on car types and P_2 on colors in Figure 4 and the following database instance: a blue coupé, a blue SUV, a green SUV and a grey limousine. Let us assume that we have iterated over the sorted lists up to level 2 in each preference, i.e. we have seen all roadsters, coupés and SUVs and all red, blue and black cars. Given the database instance, we have accessed the blue coupé, blue SUV and green SUV. Moreover the first two items have been accessed in both preferences and form a 2-cut with respect to P_1. Following theorem 1 we can now prune all remaining objects, i.e. the grey limousine. This pruning is indeed correct, since the grey limousine is weakly dominated by the blue coupé. But whereas neither the blue coupé, nor the blue SUV dominate the green SUV, it is dominated by the pruned grey limousine and thus a false positive in the restricted skyline set.

As we can see from Example 2, preference graphs where such false positives can occur have to consist of long isolated branches and the database instance should be rather sparse on top objects. In fact, finding these conditions in all preferences is very unlikely (cf. Section 4.7).

3.4 Efficiently Approximating the Restricted Skyline

For computing the correct restricted skyline we have to

- derive the Pareto skyline,
- test all elements against all other database objects for weak Pareto dominance, and
- finally remove all weakly dominated objects.

However, this is very inefficient since for Pareto skyline computation with partial order preferences usually all database objects have to be accessed (for example on a database with only 500,000 tuples and 5 partial order preferences calculating the Pareto skyline takes about 22 minutes). Exploiting sorted lists and the pruning condition defined in section 3, in the following algorithm we will take a few false positives into account. However, in return we may prune large parts of the database and thus get an efficient query processing, while still always correctly deriving all objects of the restricted skyline (for example calculating the approximate restricted skyline in the same scenario and setting as above takes only 35 seconds).

Algorithm 3: (approx. restricted skyline computation)

```
0.  Given a set of database objects $O$ and a query containing $n$ partial order preferences
    $P_1,\ldots, P_n$; given a set of $n$ sorted lists $S_1,\ldots, S_n$ of $O$ with respect to $P_1,\ldots,P_n$ in level
    order. Initialize a set for all accessed objects $accessed := \varnothing$, sets for all objects
    accessed in the $n$ lists $accessed_1,\ldots, accessed_n := \varnothing$, a set for all objects already
    accessed with respect to all preferences $complete := \varnothing$, and a set for all objects
    currently under consideration $current := \varnothing$. Compute all sets of minimal cuts
    $mincuts_{i,l}$ for the $1 \le i \le n$ preferences and $1 \le l \le maxlevel_i$ levels, using Algorithms
    1 and 2. Initialize a counter for the levels $l := 1$, for the current preference $i := 1$.
1.  If none of the preferences $P_1,\ldots,P_n$ has an $l$-th level, then return $\varnothing$ as the restricted
    skyline and terminate. If preference $P_i$ has no $l$-th level, proceed with step 4.
2.  Get all attribute values of level $l$ for the $i$-th preference $P_i$ and iterate over list $S_i$
    retrieving all objects having any of these attribute values into the set $current$.
3.  If $current \neq \varnothing$ then
    3.1.  $accessed_i := accessed_i \cup current$ and $accessed := accessed \cup current$
    3.2.  $complete := accessed_1 \cap \ldots \cap accessed_n$
    3.3.  If there exists some set of objects $C \subseteq complete$ such that the respective set of
          i-th attribute values of the objects in C is equal to some element of $mincuts_{i,l}$,
          i.e. the objects in $C$ instantiate an $l$-cut with respect to $P_i$ and have already been
          accessed in all lists $S_1,\ldots, S_n$, do
          3.3.1. For $j := i+1$ to $n$ do get all attribute values of level $l$ for the $j$-th preference
                 $P_j$ (if level $l$ exists in $P_j$) and iterate over list $S_j$. Union all objects having
                 any of these attribute values with the sets $accessed_j$ and $accessed$ like in
                 step 3.1.
          3.3.2. $complete := accessed_1 \cap \ldots \cap accessed_n$
          3.3.3. Compare all objects from set $accessed$ pairwise for weak domination and
                 subsequently remove all weakly dominated objects from set $accessed$.
          3.3.4. Return the set $accessed$ as the restricted skyline and terminate.
4.  If $i < n$, then set $i := i+1$, else set $i := 1$ and $l := l+1$. Set $current := \varnothing$ and proceed
    with step 1.
```

Basically the algorithm iterates over the preference information in a round robin fashion. It considers all objects that form a level in a preference. Of course instead of using sorted lists, all objects with a certain attribute value could also be retrieved using a database index (step 2). The algorithm then checks if an l-cut has been instantiated by completely known objects on the current level, and – if not – proceeds to process the next preference. Whenever a round is complete, it proceeds to the next level. If an l-cut has been instantiated by completely known objects, the current level is completed in all preferences and all higher levels are pruned (which is correct according to Theorem 1). The algorithm then checks for weak dominations and removes all dominated objects.

Parameter	Value
Database size	25000
Data distribution	Uniform
Number of preferences	5
Preference size	15
Isolated incomparable values	0
Preference depth	5
Edge ratio	1.2

Table 1: Default evaluation settings

4. Evaluation

To evaluate the performance of our algorithm and compare skyline sizes, we conducted extensive experiments with various parameter settings. To avoid bias, both data and preferences are synthesized randomly, and we show averages over multiple runs in our evaluation. The database content is generated according to several different distributions. Preferences are generated based on several parameters:

- *Preference size*. The number of attribute values in a preference.
- *Preference depth*. The number of levels in the preference graph (cf. Definition 2).
- *Edge ratio*. The ratio between nodes and edges in the preference graph, i.e. the average node degree.

We evaluated different scenarios to study the influence of these parameters. In all scenarios, we measured the time required to compute the skylines (runtimes) and the skyline sizes for the restricted skyline, substitute values (SV) skyline and Pareto skyline. For restricted skyline computation, we use the algorithm described in Section 3. For all other skylines, we need to do a pair-wise object comparison for all object pairs[1].

Table 1 shows our default configuration used as baseline setting. In all experiments, parameters not explicitly mentioned are set to these default values. We ran all experiments on a 2.4 GHz AMD Opteron64 Dual-processor Linux machine, equipped with 20GB main memory. The algorithm is not (yet) parallelized, therefore only one processor was actually used. Memory consumption was not regularly captured. We only measured it for the largest database size (1 million objects), where the computation of restricted skylines required 811MB.

In the next sections, we describe each experiment and its outcome in detail. Please note that we always use a logarithmic scale for both time and size to suit the large differences between skyline types.

[1] The BBS+ or SDC+ algorithms described in [CET05] may yield better runtime results for the Pareto skyline case. However, the experiments in [CET05] show results only for queries including 1 or 2 partially ordered preferences, and due to the underlying R-Tree indexing structure performance is bound to suffer for higher-dimensional skyline queries.

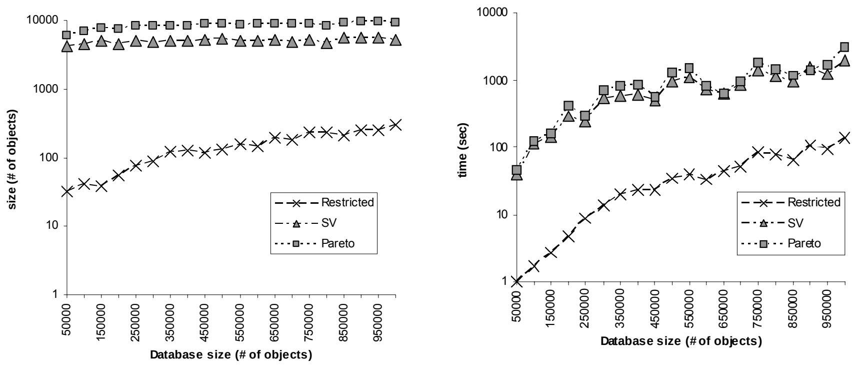


Figure 5. Database size effect on a) runtime and b) skyline size

4.1 Influence of database size

To determine the influence of the database size in terms of our algorithm's scalability, we varied the number of database objects between 50.000 and 1.000.000. Restricted skylines are in all cases computed by about two orders of magnitude faster than SV and Pareto skylines (see Figure 5a). In absolute figures, runtimes for Pareto skylines of more than 15 minutes on a powerful server can hardly be considered practical. Our algorithm can compute the skyline about two orders of magnitude faster. The dominant operations are pair-wise object comparisons which proceed for each object until a) a dominating object is found or b) the object has been compared to all others. Due to the weakened domination definition case a) occurs much more frequently in our approach. Additionally, on average far fewer comparisons are required until a dominating object is found. Figure 5b shows that the restricted skyline size starts very small (32 for 50.000 objects) and stays manageable even for large databases (297 for 1 million objects). In contrast, SV and Pareto skylines always comprise several thousand objects, already an unacceptable size for practical usage, e.g., in query refinement or relevance feedback. In summary, our proposed skyline algorithm scales well in terms of computation and skyline size.

4.2 Influence of query dimensionality

The goal in this scenario was to see how the number of preferences specified in a query affects skylines. After a small decrease in skyline sizes for 2-3 dimensions (where domination relationships are not yet outweighed by incomparability between the growing number of possible pairs of dimensions), the SV and Pareto skylines are touched by the curse of dimensionality. Like comparable work shows: their sizes quickly increase significantly up to nearly the whole database. On the other hand, the restricted skyline size only increases slightly (see Figure 6b). But what is more, even for large numbers of preferences it is still computed about an order of magnitude faster than SV and Pareto skylines, as shown in Figure 6a. We can state that the restricted skyline approach makes interactive refinement or feedback in high-dimensional skyline querying practical.

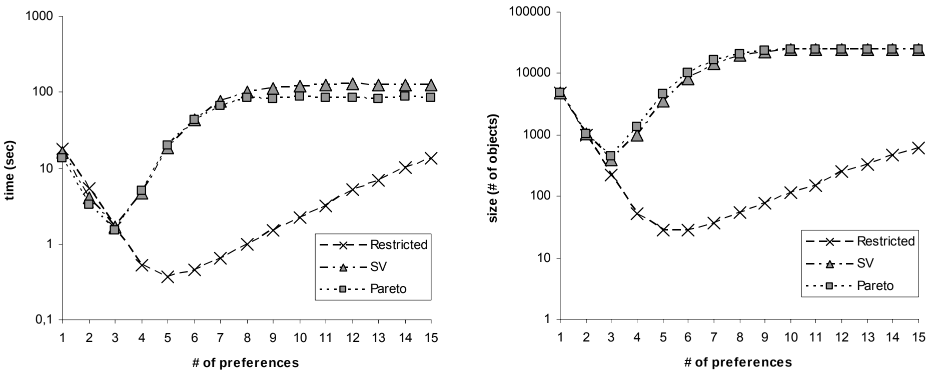


Figure 6. Preference dimensionality effect on a) runtime and b) skyline size

4.3 Influence of preference size and shape

In this experiment, we varied the preference size between 5 and 30 attribute values. Figure 7a shows that between 5 to 20 attribute values, SV and Pareto skyline sizes grow up to 20%, rsp. 25% of the database. Further increase of preference sizes doesn't show a significant impact on skyline sizes. In contrast, the restricted skyline shrinks to a minimum at preference sizes of 20, and then stays fairly constant. For preference depth, we see a different picture (Figure 7b). Pareto and SV skyline shrink notably when increasing depth from 2 to 15. This happens due to the reduction of incomparable attribute values. For depth 15, we already get a linear dominance order, without any incomparable value pairs left. For this case, Pareto, SV, and restricted skylines becomes identical, since weak and strong dominance coincide.

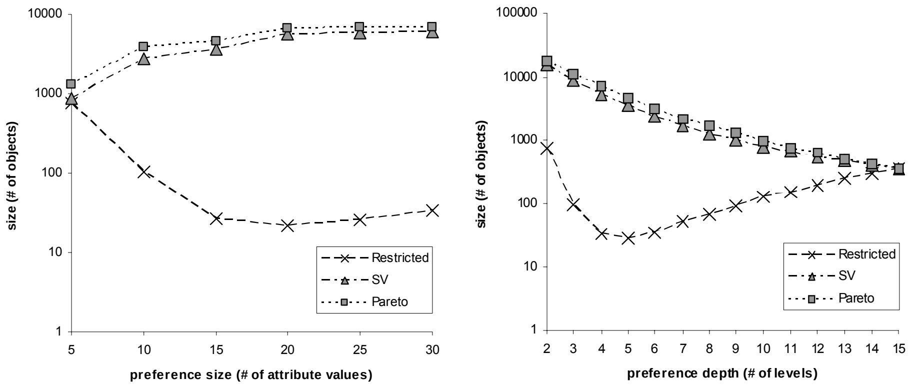


Figure 7. Influence of a) preference size and b) preference depth on skyline size

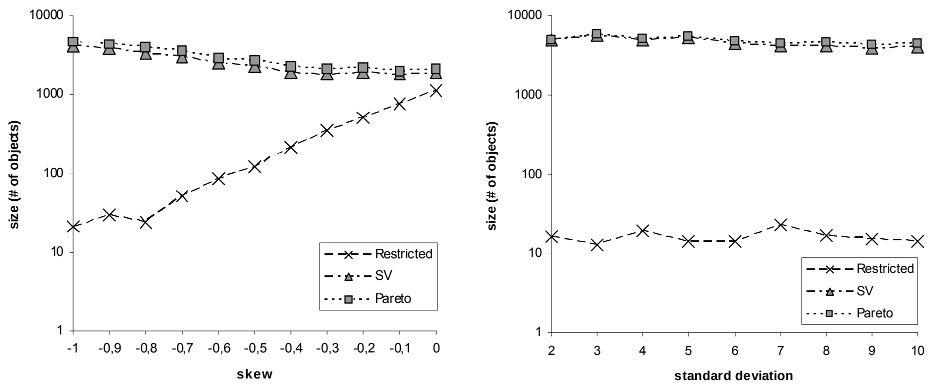


Figure 8. a) Zipf skew and b) Gaussian distribution effect on skyline size

4.4 Influence of Skewed data distribution

For our next set of experiments, we changed the distribution of our data collection. Using a Zipf distribution, we varied skews from uniform (skew parameter -1.0) to highly skewed (0.0). In the latter case, the most preferred attribute values in each preference is already assumed by 14% of all database objects. As we can see in Figure 8a, with growing skew the different skyline types coincide more and more. With so many objects having top attribute values, the chance for incomparability gets lower, and a set of rather similar top objects is bound to dominate the whole rest of the database. Similar effects can be observed when shifting the head of the Zipf distribution to the least preferred objects, thus creating a multitude of overall bad objects. In both cases, the restricted of skyline is computed an order of magnitude faster than the Pareto skyline.

4.5 Influence of Gaussian data distribution

Finally, we investigated the influence of Gaussian data distribution on skylines. Varying the standard deviation, we measured sizes and runtimes. This distribution encourages the creation of objects with medium preferred attribute values in all preferences. As Figure 8b shows, the restricted skyline size constantly stays about two orders of magnitude lower than in the Pareto and SV semantics case, independently of the standard deviation. Also here, the restricted skyline is computed about a magnitude faster than Pareto and SV skylines.

4.6 Coverage of Pareto by Restricted Skyline

To investigate how good the Pareto skyline is covered by the restricted skyline, i.e., how representative our selection from the original skyline set is, we performed a separate evaluation. We compared the coverage of the restricted skyline over the full Pareto skyline with the coverage of a random sample of the Pareto skyline. As measure for

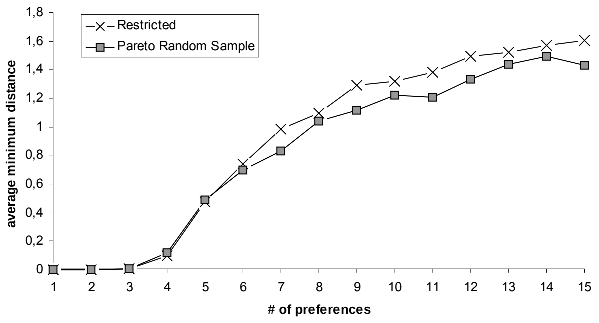


Figure 9. Average minimum distance

coverage, we use the average minimum distance of all Pareto skyline points to the objects in the restricted skyline. Small average minimum distances show a good approximation of the original set. To calculate this measure, we select for each object in the Pareto skyline the nearest object of the restricted skyline, and compute their Euclidean distance. As we have no natural numeric distances, we use again the level order to translate preference differences to numeric difference: for each preference P, the object value *v* is replaced with the numeric value *v' = level*$_P$(*v*) / *maxlevel*$_P$. The same measure is used to compute the coverage of an equally large random sample of the Pareto skyline. Such a random sample can bee seen as optimal regarding representativeness, with respect to its size. As shown in Figure 9, the restricted skyline exhibits nearly the same coverage as the random sample. This shows that the restricted skyline does not bias toward a specific area of the Pareto skyline.

4.7 Occurrence of False Positives

In all described settings, besides computing the restricted skyline according to Algorithm 3, we also computed it by exhaustive comparison of all database objects, to identify false positives. Even with our small edge ratio of 1.2, we did not encounter a single false positive. A closer look shows that it is indeed highly improbable to create preference graphs with long isolated branches, while at the same time having a database instance where the values at dominating positions are not occupied by some object.

5. Summary and Conclusions

Although skylines on partial order domains gain importance in practical applications due to their intuitive query capabilities, their evaluation times and especially their large result set sizes are still hampering their usefulness in typical interactive tasks, such as query refinement or for providing relevance feedback. Therefore the concept of weak Pareto dominance has recently been introduced, allowing to derive the restricted skyline. Restricted skylines generally allow to retrieve only the best matching objects with respect to the user's preferences. Moreover, the relaxed semantics of restricted skylines usually lead to intuitive results.

For fast query processing, we designed an efficient evaluation algorithm to approximate restricted skylines. It iterates over object lists for each preference, topologically sorted according to the level order of the respective preference. Hence, our algorithm allows for the pruning of possibly large irrelevant chunks of the database with proven correctness. While the complete restricted skyline is retrieved, some false positives can theoretically occur in the approximation. However, our evaluation indicates that these cases are very rare, and the amount of false positives is negligible in practice.

To quantify the practical impact of our approach, we performed extensive experiments. Varying preference characteristics and data distributions, our experiments show that restricted skylines are efficient to compute, as well as lean in size. Restricted skylines can be computed generally up to two orders of magnitude less expensive than Pareto skylines and stay lean even in the face of growing database sizes. They are also significantly less prone to the curse of dimensionality in face of larger numbers of user-provided preferences. Moreover, compared to similar-sized, representative random samples of the original Pareto skyline, restricted skylines do not exhibit a significant bias.

In summary, restricted skylines together with the proposed evaluation algorithm do indeed provide useful 'prime cuts' of the original Pareto skyline to the user: efficient to compute, suitable for higher dimensions, and representative.

Our future work will focus on reconciling skyline computations with utility-based ranking schemes, at least up to a certain point. In that respect, our level-ordering and sorted object lists can be seen as a first step towards mappings from purely qualitative rankings to approximate utilities for characteristic attribute combinations.

References

[BG04] W.-T. Balke, U. Güntzer. Multi-objective Query Processing for Database Systems. In *Proc. of the Int. Conf. on Very Large Databases (VLDB)*, Toronto, Canada, 2004.

[BG05] W.-T. Balke, U. Güntzer. Efficient Skyline Queries under Weak Pareto Dominance. In *Proc. of the IJCAI-05 Multidisciplinary Workshop on Advances in Preference Handling (PREFERENCE)*, Edinburgh, UK, 2005.

[BGZ04] 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)*, LNCS 2992, Heraklion, Crete, Greece, 2004.

[BZG05] W.-T. Balke, J. Zheng, U. Güntzer. Approaching the Efficient Frontier: Cooperative Database Retrieval Using High-Dimensional Skylines. In *Proc. of the Int. Conf. on Database Systems for Advanced Applications (DASFAA)*, Beijing, China, 2005.

[Be78] 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.

[BKS01] S. Börzsönyi, D. Kossmann, K. Stocker. The Skyline Operator. In *Proc. of the Int. Conf. on Data Engineering (ICDE)*, Heidelberg, Germany, 2001.

[CET05] C. Chan P. Eng, K. Tan. Stratified Computation of Skylines with Partially Ordered Domains. In *Proc. of the Int. Conf. on Management of Data (SIGMOD)*, Baltimore, MD, USA, 2005.

[Ch02] J. Chomicki. Querying with Intrinsic Preferences. In *Proc. of the Int. Conf. on Extending Database Technology (EDBT)*, LNCS 2287, Prague, Czech Republic, 2002.

[Ch03] J. Chomicki. Preference Formulas in Relational Queries. *In ACM Transactions on Database Systems (TODS)*, Vol. 28(4), 2003.

[FLN01] R. Fagin, A. Lotem, M. Naor. Optimal Aggregation Algorithms for Middleware. In *ACM Symp. on Principles of Database Systems (PODS)*, Santa Barbara, USA, 2001.

[Fi99] P. Fishburn. Preference Structures and their Numerical Representations. *Theoretical Computer Science*, vol. 217, 1999.

[GBK00] U. Güntzer, W.-T. Balke, W. Kießling. Optimizing Multi-Feature Queries for Image Data-bases. In *Proc. of the Int. Conf. on Very Large Databases (VLDB)*, Cairo, Egypt, 2000.

[HJ04] X. Huang, C. Jensen. In-Route Skyline Querying for Location-Based Services. In *Proc. of the Int. Workshop on Web and Wireless Geographical Information Systems (W2GIS)*, Goyang, Korea, 2004.

[Ki02] W. Kießling. Foundations of Preferences in Database Systems. In *Proc. of the Int. Conf. on Very Large Databases (VLDB)*, Hong Kong, China, 2002.

[Ki05] W. Kießling. Preference Queries with SV-Semantics. *In Proc. of the Int. Conf. on Management of Data (COMAD)*, Goa, India, 2005.

[KB06] B. Köhncke and W.-T. Balke. Personalized Digital Item Adaptation in Service-Oriented Environments. In *Proc. of the Int. Workshop on Semantic Media Adaptation and Personalization (SMAP 2006)*, Athens, Greece, 2006.

[KP05] V. Koltun, C. Papadimitriou. Approximately Dominating Representatives. In *Proc. of the Int. Conf. on Database Theory (ICDT)*, Edinburgh, UK, 2005.

[KRR02] D. Kossmann, F. Ramsak, S. Rost. Shooting Stars in the Sky: An Online Algorithm for Skyline Queries. In *Proc. of the Int. Conf. on Very Large Data Bases (VLDB)*, Hong Kong, China, 2002.

[LL87] M. Lacroix, P. Lavency. Preferences: Putting more Knowledge into Queries. In *Proc. of the Int. Conf. on Very Large Databases (VLDB)*, Brighton, UK, 1987.

[Mo88] A. Motro. VAGUE: A User Interface to Relational Databases that Permits Vague Queries. In *ACM Transactions on Office Information Systems (TOIS)*, vol. 6(3), 1988.

[PTF03] D. Papadias, Y. Tao, G. Fu, et.al. An Optimal and Progressive Algorithm for Skyline Queries. *In Proc. of the Int. ACM SIGMOD Conf. (SIGMOD'03)*, San Diego, USA, 2003.

[TEO01] K.-L. Tan, P.-K. Eng, B. C. Ooi. Efficient Progressive Skyline Computation. In *Proc. of Conf. on Very Large Data Bases (VLDB'01)*, Rome, Italy, 2001