# User Interaction Support for Incremental Refinement of Preference-Based Queries

Wolf-Tilo Balke, Ulrich Güntzer, Christoph Lofi

*Abstract*—**Preference-based queries (or skylines) play an important role in cooperative query processing. However, their prohibitive result sizes pose a severe challenge to the paradigm's practical applicability. Since skyline sizes can only be predicted under strong assumptions, we present a sophisticated user interface to interactively refine queries in case the respective skyline set proves to be too large. Our approach allows users to incrementally augment preferences given by their Hasse diagrams. Moreover, we prove the correctness of the incremented skyline and in our experiments show the approach's superior efficiency.**

*Index Terms* — **Information Systems, User Interfaces**

## I. Introduction

Preference-based queries, usually called skyline queries in database research [8], [1], [15], have become a prime paradigm for cooperative information systems. Their major appeal is the intuitiveness of use in contrast to other query paradigms like e.g. rigid set-based SQL queries, which only too often return an empty result set, or efficient, but hard to use top-k queries, where the success of a query depends on choosing the right scoring or utility functions.

Skyline queries do not need the a-priori specification of utility functions or weightings for individual query predicates; they return the so-called Pareto-optimal set, i.e. all database objects optimal with respect to any utility function. However, the intuitiveness of querying comes at a price. Skyline sets are known to grow exponentially in size [7], [11] with the number of query predicates and may reach unreasonable result sets (of about half of the original database size, cf. [5],) already for as little as six independent query predicates. Moreover, skyline queries are expensive to compute. Evaluation times in the range of several minutes or even hours over large databases are not unheard of.

Hence, making this valuable paradigm applicable is still a great challenge in information systems research. One possible solution is based on the idea of refining skyline queries incrementally using user interaction. This approach is promising, since it benefits skyline sizes as well as evaluation times. Recently, several approaches have been proposed for user-centered refinement:

- either using an interactive, exploratory process steering the progressive computation of skyline objects [14]
- or by exploiting feedback on a representative sample of the original skyline result [6], [13]
- or by projecting the complete skyline on subsets of predicates using pre-computed skycubes [16], [18].

The benefit of offering intuitive querying and a cooperative system behavior to the user in all three approaches can be obtained with a minimum of user interaction to guide the further refinement of the skyline. However, when dealing with a massive amount of result tuples, the first approach needs a certain user expertise for steering the progressive computation effectively. The second approach faces the problem of deriving representative samples efficiently, i.e. avoiding a complete skyline computation for each sample. In the third approach the necessary pre-computations are expensive in the face of updates of the database instance.

In this paper we will follow a different approach and allow the user to interactively engineer their base preferences. We have already proposed a basic algorithm [2] to allow users to incrementally add domination and equivalence relationships between individual predicate values. Now we extend the approach deploying a sophisticated user interface: users are enabled to work on their base preferences for different predicates as given by their respective Hasse diagrams. We will model the preferences utilizing the OWL framework [17] to allow for easy handling and interoperability. Moreover, a graphical user interface where both additional preferences and trade-offs between different attribute values can be graphically expressed is designed. Our main contribution is to show that the incremental computation of the skyline can be done in a efficient manner only considering the local changes in the individual preference graphs. Our extensive evaluation confirms that our scheme is well suited for incorporating user feedback into skyline computations.

The rest of the paper is organized as follows: section 2 introduces the notion of preference-based skylines and will also discuss a suitable modeling of preferences using the OWL standard. Section 3 gives the foundation for our interface and proves that manipulating Hasse diagrams indeed leads to the

Manuscript received October 22, 2006. Part of this work was supported by a grant of the German Research Foundation (DFG) within the Emmy Noether Program of Excellence.

Wolf-Tilo Balke and Christoph Lofi are with L3S Research Center, Leibniz University Hannover, Appelstr. 9a 30167 Hannover, Germany, {balke, lofi}@l3s.de.

Ulrich Güntzer is with Institute of Computer Science, University of Tübingen, Sand 13, 72076 Tübingen, Germany, ulrich.güntzer@informatik. uni-tübingen.de.

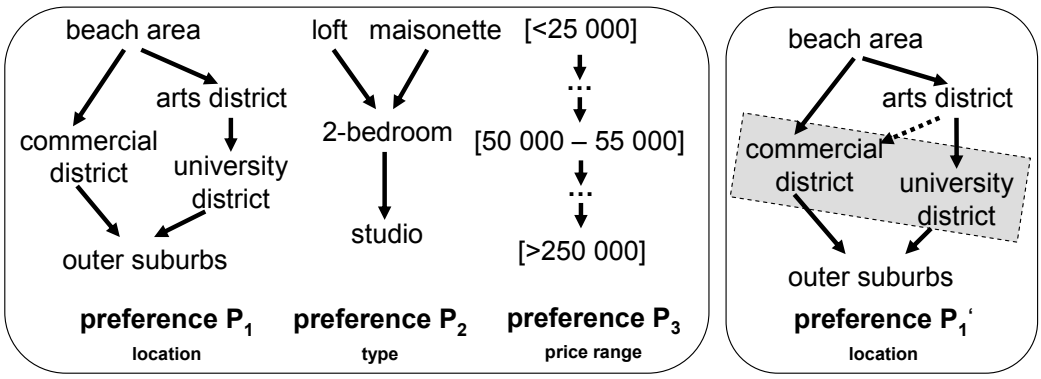


Fig. 1. Three typical user preferences (left) and an enhanced preference (right)

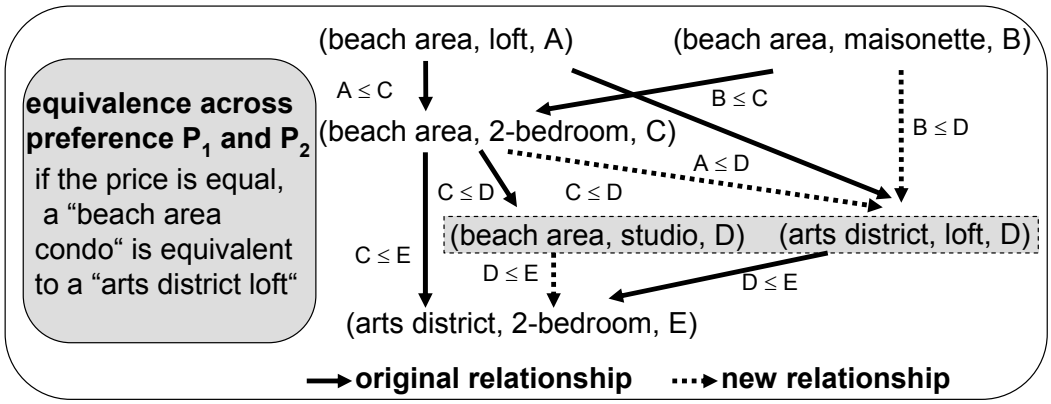


Fig. 2. Original and induced preference relationships for trade offs

desired incremented skylines and is even more efficient then working on the larger preference relations. Section 4 contains our thorough experimental section and will introduce our interface. We close with a short summary and outlook.

## II. A Skyline Query Use-Case and Modeling

This section will briefly describe our incremental skyline approach with a practical use-case. In subsections B and C, we will introduce and discuss our novel approach for modeling the example introduced in subsection A with OWL, the current standard language for ontology modeling [17].

### *A. Basic Concepts of Partial Order Skyline Processing*

Before discussing the basic concepts of skyline processing, let us first take a closer look at a motivating scenario:

**Example**: Anna is currently looking for a new apartment. Naturally, she has some preferences how and where she wants to live. Figure 1 shows Hasse diagrams of Anna’s preferences modeled as a *strict partial order* (cf. [12], [10]) on three predicates: location, apartment type and price. These preferences might either be stated explicitly by Anna together with the query and might be automatically complemented with preferences derived from Anna’s user profile and/or activity history [9]. Some of these preferences may even be common domain knowledge (c.f. [3]) like for instance that in case of two equally desirable objects, the less costly one is generally preferred. Based on such preferences, Anna may now retrieve the skyline over a real-estate database. The result is the *Pareto-optimal set* [1] of available apartments consisting of all apartments which are not dominated by others, e.g. a cheap beach area loft immediately dominates all more expensive 2-bedrooms or studios, but can, for instance, not dominate any maisonette. After the first retrieval Anna has to manually review a probably large skyline.

In the first few retrieval steps skylines usually contain a large portion of all database objects due to the incomparability between many objects. But the size of the Pareto-optimal set may be reduced incrementally by providing suitable additional information on top of the stated preferences, which will then result in new domination relationships on the level of the database objects and thus remove less preferred objects from the skyline. Naturally, existing preferences might be extended by adding some new preference relations. But also explicit equivalences may be stated between certain attributes express-

ing actual indifference and thus resulting in new domination relationships, too.

**Example (cont):** Let's assume the skyline provided to Anna still contains too many apartments. Thus, Anna interactively refines her originally stated preferences. For example, she might state that she actually *prefers* the arts district over the university district and the latter over the commercial district which would turn the preference $P_1$ into a totally ordered relation. This would for instance allow apartments located in the arts and university district to dominate those located in the commercial district with respect to $P_1$, resulting in a decrease of the size of the Pareto-optimal set of skyline objects. Alternatively, Anna might state that she actually does not care whether her flat is located in the university district or the commercial district – that these two attributes are *equally desirable* for her. This is illustrated in the right hand side part of figure 1 as preference $P_1$'. In this case, it is reasonable that all arts district apartments will dominate commercial district apartments with respect to the location preference.

Preference relationships over predicates as previously described lead to domination relationships on object level, when the skyline operator is applied to a given database. These resulting domination relationships are illustrated by the solid arrows in figure 2. However, users might also weigh some predicates as more important than others and hence might want to model trade-offs they are willing to consider. Our preference modeling approach introduced in [2] allows expressing such trade-offs by providing new preference relations or equivalence relations between attributes of different predicates. This means 'amalgamating' some predicates in the skyline query and subsequently reduces the dimensionality of the query.

**Example (cont):** While refining her preferences, Anna realizes that she actually would consider the area in which her new apartment is located as more important than the actual apartment type – in other words: for her, a relaxation in the apartment type is less severe than a relaxation in the area predicate. Thus she states that she would consider a beach area studio (the least desired apartment type in the best area) still as equally desirable to a loft in the art district (the best apartment type in a less preferred area). This statement induces new domination relations on database object level (illustrated as the dotted arrows in figure 2), allowing for example any cheaper beach area 2-bedroom to dominate all equally priced or more expensive arts district lofts. In this way, the result set size of the skyline query can be decreased. For instance, if there is no beach area loft or only an expensive one, then some arts district loft were part of the original skyline. But after the trade-off they are rightfully eliminated.

### *B. Storing and managing preferences using OWL*

Preferences as described in the previous section can be considered as a special case of more general taxonomies or ontologies. With ontologies and Semantic Web technologies gaining more and more importance and acceptance in information systems recently (see e.g., [3] for a discussion), we decided to take advantage of the already established and elaborated OWL standard (Web Ontology Language) for storing and managing preferences in our framework. Using OWL (OWL-DL [17] to be exact) for representing preferences provides several advantages:

- *Standard Acceptance*: OWL is proposed by W3C as the current standard for ontology engineering and semantic Web applications [17]. It is supported by many practitioners, is well known and widely spread. As a result, many commercial applications and tools support OWL, allowing for an easier integration and interoperability of our framework with existing technologies and products.
- *Extensibility and Flexibility*: OWL provides simple to use constructs for describing relations between object classes and instances. In our framework, we describe preference and equivalence relations over predicate attributes. But due to the language's flexibility, this concept can be easily extended with additional relations. Also, additional properties of relations can be specified like e.g. conditional information describing in which cases the given preference can/can't be applied.
- *Validation*: OWL was designed with automated reasoning in mind. This feature can be exploited for performing validation of preferences by utilizing external reasoners with additional rules formalized in a rule language. In our framework any preference is valid only if no inconsistent relationships have been stated. The complex rules to check consistency of an instantiated preference system [2] form the rule-base of a reasoner. Moreover, a variety of simpler semantic constraints on preferences (like compatible types, etc.) can be expressed by using so-called restrictions, preventing the authoring of invalid preference models even without additional rules or use of reasoners.

In the following, we briefly explain our approach for modeling the example preference introduced in figure 1 and figure 2 in OWL. Our OWL model contains classes for the database objects (the apartments in our example), the predicates and possible trade-off predicates (modeled as pairs of predicates) together with the respective domains and ranges. This leads to a class structure as illustrated in figure 4.

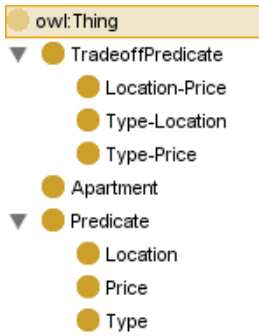


Fig. 4. OWL-Protégé Class Hierarchy

The class `Predicate` is used for the different predicates (type, location, price) of `Apartment` which itself is used for representing the database objects. `Apartment` relates to exactly one of each of `Predicate`'s subclasses - each apartment has exactly one type, one price and one area, in which it is located. This can be ensured by restrictions as illustrated in figure 5.


Fig. 5. OWL-Protégé Restrictions for Class `Apartment`

When modeling a preference each possible predicate value (like loft, maisonette, etc) is represented by an instance of the respective predicate's sub-classes. The class `Predicate` itself is treated as an abstract class and may not have instances that are not also instances of one of its subclasses. Between instances of a `Predicate` class, two special relations are possible `isPreferredTo` and `isEquivalentTo` represent preference or equivalence relationships, respectively. For ensuring that basic preference and equivalence modeling is only performed between instances of the same predicate subclass, specific restrictions ensuring this property have to be introduced for each class. Example restrictions for `Location` are illustrated in figure 6.


Fig. 6. OWL-Protégé Restrictions for Class `Location`

Unfortunately, it is not possible to ensure the path integrity of the preferences by just using OWL itself. For preventing invalid instances such as preference cycles (A is preferred to B is preferred to C is preferred to A), an external reasoner has to be deployed. An example OWL extract for the predicate `Location` is given by code 1.

```
<!-- Class Location -->
<owl:Class rdf:about="#Location">
    <!-- Restrict the isPreferredOver relation to
this Type only. -->
    <rdfs:subClassOf>
      <owl:Class rdf:ID="Predicate"/>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:allValuesFrom
            rdf:resource="#Location"/>
        <owl:onProperty>
          <owl:ObjectProperty
              rdf:ID="isPreferredOver"/>
        </owl:onProperty>
      </owl:Restriction>
    </rdfs:subClassOf>

    ...

  </owl:Class>
```

Code 1. OWL extract of `Location`

For modeling trade-offs between two different predicates, we need to relate a 2-tuple of predicate values to another 2-tuple of predicate values with matching types (like "beach area studios are considered to be equivalent to arts district lofts"). These sets are expressed by suitable subclasses of `TradeoffPredicate`. These subclasses represent all meaningful predicate sets that might be involved in a trade-off. Each actual trade-off can then be modeled by relating an instance of a subclass of the class `TradeoffPredicate` to another instance of the same subclass using again one of the relationships `isPreferredTo` or `isEquivalentTo`. As in the case of normal predicates, the base class is treated as abstract and invalid relations between instances of its different subclasses are prevented by using restrictions. This is illustrated in figure 7 and results in the OWL extract given by code 2 (both for `Type-Location`, the amalgamation of the predicates `Type` and `Location`).

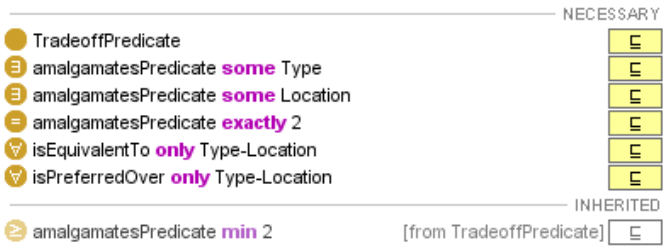


Fig. 7. OWL-Protégé Restrictions for Class `Type-Location`

As a result of this approach, only those relations illustrated in figure 8 are possible. Invalid relations between predicates or amalgamated predicates of different types are prevented by the introduced restrictions.

```
<!-- Class Type-Location -->
<owl:Class rdf:ID="Type-Location">

    <!-- Restrict tradeoff preference to only
other Type-Location-Instances. -->
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty>
          <owl:TransitiveProperty
        rdf:about="#isPreferredOver"/>
        </owl:onProperty>
        <owl:allValuesFrom
             rdf:resource="#Type-Location"/>
      </owl:Restriction>
    </rdfs:subClassOf>

    ...

    <!-- Restrict that at least one Location
predicate has to be amalgamated. -->
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:someValuesFrom
                    rdf:resource="#Location"/>
        <owl:onProperty>
          <owl:ObjectProperty
           rdf:about="#TradeoffPredicate"/>
        </owl:onProperty>
      </owl:Restriction>
    </rdfs:subClassOf>

    ...

    <!-- Restrict to exactly 2 amalgamations.
Together with the restrictions above, this en-
sures validity of the preference and equivalence
relations. -->
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty>
          <owl:ObjectProperty
           rdf:about="#TradeoffPredicate"/>
        </owl:onProperty>
        <owl:cardinality rdf:datatype="…#int">
          2
        </owl:cardinality>
      </owl:Restriction>
    </rdfs:subClassOf>

  </owl:Class>
```

Code 2. OWL extract of Class `Type-Location`

### C. *Preference Modeling Limitations of OWL*

While the modeling approach which we presented in the last subsection is functional and covers the aspects of our incremental preference approach, it still contains some problems inherent in the limited expressiveness of OWL.

Though for the practical application in our framework using simple OWL restrictions was useful, it is still a limited approach for ensuring the validity of the created preference models. For example, restrictions on complex attributes of some types for individuals participating in relationships are not possible. This especially posed a problem when modeling trade-offs that have to be expressed by a relation over two amalgamated predicates (i.e. a 2-tuple of predicate values).

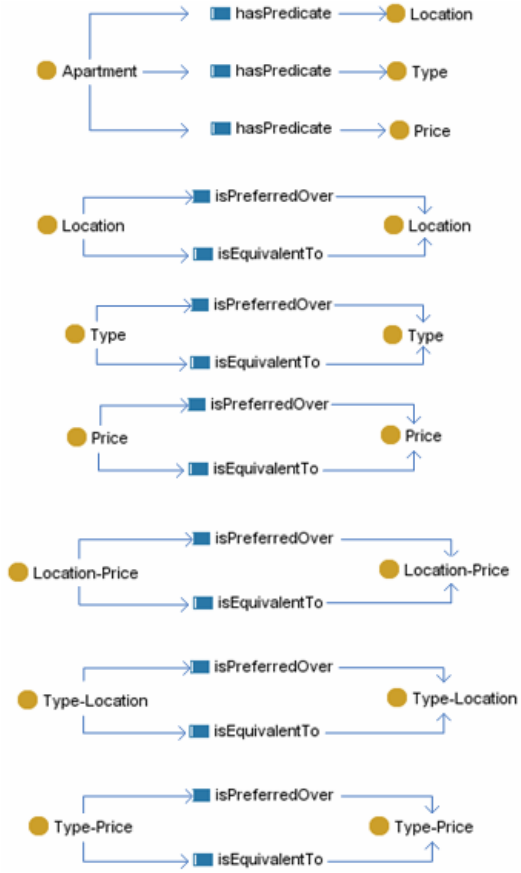


Fig. 8. Relations and their Domains and Ranges

Each set may only contain individuals of *different* types. Moreover, preferences or equivalences may only be modeled between amalgamated predicates which contain *compatible* predicate instances (this means that if the source amalgamated predicate of a trade-off relation contains e.g., one individual of type `Location` and of type `Type`, the target may only contain two individuals with these respective types, too. However, these kinds of restrictions are not expressible using OWL and lead to our (in terms of conceptual modeling somewhat clumsy) approach of having to introduce subclasses of `TradeoffPredicate` for each meaningful predicate type combination. When all predicate combinations are considered as being meaningful in the case of n different predicates, this will lead to $2^n$-(n+2) subclasses of `TradeoffPredicate`.

## III. Formalization of Incremental User Interaction

In this section we formalize the semantics of adding incremental preference or equivalence information. Building on the work in [2] we show that it suffices to calculate incremental changes on the Hasse diagram and that local changes in the preference graph only result in locally restricted recomputations for the incremented skyline.

### A. Incremental Preference Elicitation

Throughout our work we understand preferences in their most general form as strict partial orders following an intuitive 'I like A better than B' semantics [12], and enrich them by adding symmetric equivalence information stating that attribute values are considered equally desirable. As presented in section two, an easy way for users to specify preference information is graphically, using the Hasse-diagram. Basically the Hasse diagram is a simple graphical representation of attribute values and preference edges.

**Definition 1:** Let $P$ be a preference in the form of a finite strict partial order. The *Hasse diagram H(P)* for preference $P$ denotes a (not necessarily minimal) preference such that the transitive closure $H(P)^+ = P$.

Please note that there may be several preference graphs provided (or incrementally completed) by the user to express the same preference information (given by the transitive closure of the graph), although there is only one minimal Hasse diagram for each preference. But in the user interface we only need to prevent inconsistent (i.e. conflicting) information, not redundant information. Thus, we use not necessarily the minimal Hasse diagram, but the diagram as expressed by the user. In general this diagram will still be a lot smaller than its transitive closure. In this section we will take a closer look at what actually has to be considered for calculating the incremented skyline. To be self-contained we will now revisit the formal definition of the incremented preference as given in [2]:

**Definition 2:** Let $O$ be a set of database objects, $P \subseteq O^2$ be a strict preference relation, $P^{conv} \subseteq O^2$ be the set of converse preferences with respect to $P$, and $Q \subseteq O^2$ be an equivalence relation that is compatible with $P$, i.e. $P \circ Q \subseteq P$ and $Q \circ P \subseteq P$. Let further $S \subseteq O^2$ be a set of object pairs (new preference relationships) such that

$$\forall\, x, y \in O\text{: } (x,y) \in S \Rightarrow (y,x) \notin S$$
$$\text{and} \quad S \cap (P \cup P^{conv} \cup Q) = \varnothing$$

and let $E \subseteq O^2$ be a set of object pairs (new equivalence relationships) such that

$$\forall\, (x,y) \in O\text{: } (x,y) \in E \Rightarrow (y,x) \in E$$
$$\text{and} \quad E \cap (P \cup P^{conv} \cup Q \cup S) = \varnothing.$$

Then we will define $T$ as the transitive closure $T := (P \cup Q \cup S \cup E)^+$ and the incremented preference relation $P^*$ and the incremented equivalence relation $Q^*$ as

$$P^* := \{ (x,y) \in T \mid (y,x) \notin T\}$$
$$\text{and} \quad Q^* := \{ (x,y) \in T \mid (y,x) \in T\}$$

The basic intuition is that $S$ and $E$ contain the new preference and equivalence relationships that have been elicited from the user additionally to those given in $P$ and $Q$ (for a discussion of preference elicitation methods see e.g. [9]). The only constraints on $S$ and $E$ are that neither can they directly contradict each other, nor are they allowed contradicting already known information. The sets $P^*$ and $Q^*$ then are the new, i.e. incremented, preference/equivalence sets that incorporate all the information from $S$ and $E$ and that will be used to calculate the incremented (and generally smaller) skyline set. If the increments E and S are consistent with P and Q, i.e. $T \cap P^{conv} = \varnothing$, evaluating the skyline based on the incremented preference $P^*$ from definition 2 does indeed result in the desired incremented skyline set as we showed in [2].

But the incremented preference $P^*$ explicitly contains all transitive information and thus is difficult to handle in both the user interface and the actual computation of the incremental skyline. We want to avoid handling P* and rather only incorporate increments of new preference information as well as new equivalences into the Hasse diagram instead. The following two theorems show how to do this:

**Theorem 1:** Let $O$ be a set of database objects and $P$, $P^{conv}$, and $Q$ as in definition 2 and $E := \{(x, y), (y, x)\}$ new equivalence information such that $(x, y), (y, x) \notin (P \cup P^{conv} \cup Q)$. Then $P^*$ can be calculated as

$$P^* = (P \cup (P \circ E \circ P) \cup (Q \circ E \circ P) \cup (P \circ E \circ Q)).$$

**Proof:** Assume $(a, b) \in T$ as defined in definition 2. The edge can be represented by a chain $(a_0, a_1) \circ \ldots \circ (a_{n-1}, a_n)$, where each edge $(a_{i-1}, a_i) \in (P \cup Q \cup E)$ and $a_0 := a$, $a_n := b$. This chain can even be transcribed into a representation with edges from $(P \cup Q \cup E)$, where at most one single edge is from $E$. This is because, if there would be two (or more) edges from $E$ namely $(a_{i-1}, a_i)$ and $(a_{j-1}, a_j)$ (with $i < j$) then there are four possibilities:

a) both edges are $(x, y)$ or both edges are $(y, x)$, in both of which cases the sequence $(a_i, a_{i+1}) \circ \ldots \circ (a_{j-1}, a_j)$ forms a cycle and can be omitted

b) the first edge is $(x, y)$ and the second edge is $(y, x)$, or vice versa, in both of which cases $(a_{i-1}, a_i) \circ \ldots \circ (a_{j-1}. a_j)$ forms a cycle and can be omitted, leaving no edge from $E$ at all.

Since we have defined $Q$ as compatible with $P$ in definition 2, we know that $(P \cup Q)^+ = (P \cup Q)$ and since elements of $T$ can be represented with at most one edge from $E$, we get $T = P \cup Q \cup ((P \cup Q) \circ E \circ (P \cup Q))$.

In this case both edges in E are consistent with the already known information, because there are no cyclic paths in $T$ containing edges of $P$ ( c.f. condition 1.4.c) in [2]): This is because if there would be a cycle with edges in $(P \cup Q \cup E)$ and at least one egde from $P$ (i.e. the new equivalence information would create an inconsistency in $P^*$), the cycle could be represented as $(a_0, a_1) \in P$ and $(a_1, a_2) \circ \ldots \circ (a_{n-1}, a_0)$ would at most contain one edge from $E$ and thus the cycle is either of the form $P \circ (P \cup Q)$, or of the form $P \circ (P \cup Q) \circ E \circ (P \cup Q)$. In the first case there can be no cycle, because otherwise $P$ and $Q$ would already have been inconsistent, and if there would be cycle in the second case, there would exist objects $a, b \in O$ such that $(a, x) \in P$, $(x, y) \in E$ and $(y, b) \in (P$

$\cup Q$) and $(y, x) = (y, a) \circ (a, x) \in (P \cup Q) \circ P \subseteq P$ contradicting $(x, y) \notin P^{conv}$.

Because of $T = (P \cup Q \cup (P \circ E \circ P) \cup (Q \circ E \circ P) \cup (P \circ E \circ Q) \cup (Q \circ E \circ Q))$ and $P^* = T \setminus Q^*$ and since $(P \cup (P \circ E \circ P) \cup (Q \circ E \circ P) \cup (P \circ E \circ Q)) \cap Q^* = \emptyset$ (if the intersection would not be empty then due to $Q^*$ being symmetric there would be a cycle in $P^*$ with edges from $(P \cup Q \cup E)$ and at least one edge from $P$ contradicting the condition 1.4 above), we finally get $P^* = (P \cup (P \circ E \circ P) \cup (Q \circ E \circ P) \cup (P \circ E \circ Q))$. ■

We have now found a way to derive $P^*$ in the case of a new incremental equivalence relationship, but still $P^*$ is a large relation containing all transitive information. We will now show that we can also get $P^*$ by just manipulating a respective Hasse diagram in a very local fashion. Locality here results to only having to deal with edges that are directly adjacent in the Hasse diagram to the additional edges in $E$. Let us define an abbreviated form of writing such edges:

**Definition 3:** Let $R$ be a binary relation over a set of database objects $O$ and let $x \in O$. We write:

$(\_\ R\ x) := \{ y \in O \mid (y, x) \in R\}$ and

$(x\ R\ \_) := \{ y \in O \mid (x, y) \in R\}$

and if R is an equivalence relation we write the objects in the equivalence class of x in R as:

$R[x] := \{ y \in O \mid (x, y), (y, x) \in R\}$

With these abbreviations we will show what objects sets have to be considered for actually calculating $P^*$ via a given Hasse diagram:

**Theorem 2:** Let $O$ be a set of database objects and $P$, $P^{conv}$, and $Q$ as in definition 2 and $E := \{(x, y), (y, x)\}$ new equivalence information such that $(x, y), (y, x) \notin (P \cup P^{conv} \cup Q)$. If $H(P) \subseteq P$ is some Hasse diagram of $P$, and with $H(P)^* := (H(P) \cup (H(P) \circ E \circ Q) \cup (Q \circ E \circ H(P)))$, holds:

$$(H(P)^*)^+ = P^*$$

i.e. $H(P)^*$ is a Hasse diagram for $P^*$, which can be calculated as:

$$H(P)^* = H(P) \cup (\_\ H(P)\ x \times Q[y]) \cup (\_\ H(P)\ y \times Q[x]) \cup (Q[x] \times y\ H(P)\_) \cup (Q[y] \times x\ H(P)\_).$$

**Proof:** We know from theorem 1 that $P^* = (P \cup (P \circ E \circ P) \cup (Q \circ E \circ P) \cup (P \circ E \circ Q))$ and for Hasse diagrams $H(P)$ of $P$ holds:

a) $P = \mathrm{H(P)}^+ \subseteq (H(P)^*)^+$

b) $(P \circ E \circ P) = (P \circ E) \circ P \subseteq (P \circ E \circ Q) \circ P = (H(P)^+ \circ E \circ Q) \circ H(P)^+ \subseteq (H(P)^*)^+$, because $(H(P)^+ \circ E \circ Q) \subseteq (H(P)^*)^+$ and $H(P)^+ \subseteq (H(P)^*)^+$.

c) Furthermore $(P \circ E \circ Q) = H(P)^+ \circ E \circ Q \subseteq (H(P) \circ E \circ Q)^+ \subseteq (H(P)^*)^+$

d) And similarly $(Q \circ E \circ P) = Q \circ E \circ H(P)^+ \subseteq (Q \circ E \circ H(P))^+ \subseteq (H(P)^*)^+$

Using a) – d) we get $P^* \subseteq (H(P)^*)^+$ and since $H(P)^* \subseteq P^*$, we get $(H(P)^*)^+ \subseteq (P^*)^+ = P^*$ and thus $(H(P)^*)^+ = P^*$.

To calculate $H(P)^*$ we have to consider the terms in $H(P) \cup (H(P) \circ E \circ Q) \cup (Q \circ E \circ H(P))$: The first term is just the old Hasse diagram. Since the second and third terms both contain a single edge from $E$ (i.e. either $(x, y)$ or $(y, x)$), the terms can be written as $(H(P) \circ E \circ Q) = (\_\ H(P)\ x \times Q[y]) \cup (\_\ H(P)\ y \times Q[x])$ and $(Q \circ E \circ H(P)) = (Q[x] \times y\ H(P)\_) \cup (Q[y] \times x\ H(P)\_)$ ■

In general these sets will be rather small, because first they are only derived from the Hasse diagram which is usually considerably smaller than preference $P$ and second in these small sets usually there will be only few edges originating or ending in $x$ or $y$. Furthermore, these sets can be computed easily using an index on the first and second entry of the binary relation $H(P)$ and $Q$. Getting a set like e.g., $\_\ H(P)\ x$ then is just an inexpensive index lookup of the type 'select all first entries from $H(P)$ where second entry is $x$'.

Therefore we can calculate the incremented preference $P^*$ by simple manipulations on $H(P)$ and the computation of a transitive closure like shown in the commutating diagram in figure 9.

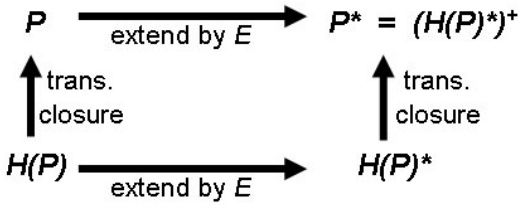


Fig. 9. Diagram for deriving incremented skylines

Having completed incremental changes introduced by new equivalence information, we will now consider incremental changes by new preference information.

**Theorem 3:** Let $O$ be a set of database objects and $P$, $P^{conv}$, and $Q$ as in definition 2 and $S := \{(x, y)\}$ new preference information such that $(x, y) \notin (P \cup P^{conv} \cup Q)$. Then $P^*$ can be calculated as

$$P^* = (P \cup (P \circ S \circ P) \cup (P \circ S \circ Q) \cup (Q \circ S \circ P) \cup (Q \circ S \circ Q)).$$

**Proof:** The proof is similar to the proof of theorem 1. Assume $(a, b) \in T$ as defined in definition 2. The edge can be represented by a chain $(a_0, a_1) \circ \ldots \circ (a_{n-1}, a_n)$, where each edge $(a_{i-1}, a_i) \in (P \cup Q \cup S)$ and $a_0 := a$, $a_n := b$. This chain can even be transcribed into a representation with edges from $(P \cup Q \cup S)$, where edge $(x, y)$ occurs at most once. This is because, if $(x, y)$ occurs twice the two edges would enclose a cycle that can be removed.

Since we have assumed $Q$ to be compatible with $P$ in definition 2, we know $T = P \cup Q \cup ((P \cup Q) \circ S \circ (P \cup Q))$ and (like in theorem 1) edge $(x, y)$ is consistent with the already known information, because there are no cyclic paths in $T$ containing edges of $P$ ( c.f. condition 1.4.c) in [2]): This is because if there would be a cycle with edges in $(P \cup Q \cup S)$ and at least one egde from $P$ (i.e. the new preference information would create an inconsistency in $P^*$), the cycle could be represented as $(a_0, a_1) \in P$ and $(a_1, a_2) \circ \ldots \circ (a_{n-1}, a_0)$ would at most contain one edge from $S$ and thus the cycle is either of the form $P$, or of the form $P \circ (P \cup Q) \circ S \circ (P \cup Q)$. In the first case there can be no cycle, because otherwise $P$ would already have been inconsistent, and if there would be cycle in the second case, there would exist objects $a, b \in O$ such that $(a, x) \in P$ and $(y, b) \in (P \cup Q)$ and $(y, x) = (y, a) \circ (a, x) \in (P \cup Q) \circ P \subseteq P$ contradicting $(x, y) \notin P^{conv}$.

Similarly, there is no cycle with edges in $(Q \cup S)$ and at least one egde from $S$, either: if there would be such a cycle, it could be transformed into the form $S \circ Q$, i.e. $(x, y) \circ (a, b)$ would be a cycle with $(a, b) \in Q$, forcing $(a, b) = (y, x) \in Q$ and thus due to $Q$'s symmetry a contradiction to $(x, y) \notin Q$.

Because of $T = (P \cup Q \cup (P \circ S \circ P) \cup (Q \circ S \circ P) \cup (P \circ S \circ Q) \cup (Q \circ S \circ Q))$ and $P^* = T \setminus Q^*$ and since $(P \cup (P \circ S \circ P) \cup (Q \circ S \circ P) \cup (P \circ S \circ Q) \cup (Q \circ S \circ Q)) \cap Q^* = \emptyset$ (if the intersection would not be empty then due to $Q^*$ being symmetric there would be a cycle in $P^*$ with edges from $(P \cup Q \cup S)$ and at least one edge from $P$ contradicting the condition 1.4 above), we finally get $P^* = (P \cup (P \circ S \circ P) \cup (Q \circ S \circ P) \cup (P \circ S \circ Q) \cup (Q \circ S \circ Q))$. ■

Analogously to theorem 1 and 2 in the case of a new incremental preference relationship, we can also derive $P^*$ very efficiently by just working on the small Hasse diagram instead on the large preference relation $P$.

**Theorem 4:** Let $O$ be a set of database objects and $P$, $P^{conv}$, and $Q$ as in definition 2 and $S := \{(x, y)\}$ new preference information such that $(x, y) \notin ( P \cup P^{conv} \cup Q)$. If $H(P) \subseteq P$ is some Hasse diagram of $P$, and with $H(P)^* := (H(P) \cup (Q \circ S \circ Q))$, holds: $(H(P)^*)^+ = P^*$
i.e. $H(P)^*$ is a Hasse diagram for $P^*$, which can be calculated as:
$$H(P)^* = H(P) \cup (Q[x] \times Q[y]) \quad \text{with } H(P) \cap (Q[x] \times Q[y]) = \emptyset.$$

**Proof:** We know from theorem 3 that $P^* = (P \cup (P \circ S \circ P) \cup (P \circ S \circ Q) \cup (Q \circ S \circ P) \cup (Q \circ S \circ Q))$ and for Hasse diagrams $H(P)$ of $P$ holds:

a) $P = H(P)^+ \subseteq (H(P)^*)^+$

b) since $S \subseteq Q \circ S \circ Q \subseteq H(P)^*$, it follows $(P \circ S \circ P) \subseteq (H(P)^*)^+ \circ H(P)^* \circ (H(P)^*)^+ \subseteq (H(P)^*)^+$

c) since $Q \circ S \subseteq (Q \circ S) \circ Q \subseteq H(P)^*$, it follows $((Q \circ S) \circ P) \subseteq H(P)^* \circ (H(P)^*)^+ \subseteq (H(P)^*)^+$

d) analogously $(P \circ (S \circ Q)) \subseteq (H(P)^*)^+ \circ H(P)^* \subseteq (H(P)^*)^+$

e) finally by definition $(Q \circ S \circ Q) \subseteq H(P)^* \subseteq (H(P)^*)^+$

Using a) – e) we get $P^* \subseteq (H(P)^*)^+$ and since $H(P)^* \subseteq P^*$, we get $(H(P)^*)^+ \subseteq (P^*)^+ = P^*$ and thus $(H(P)^*)^+ = P^*$.

To calculate $H(P)^*$ analogously to theorem 2 we have to consider the terms in $H(P) \cup (Q \circ S \circ Q)$: The first term again is just the old Hasse diagram. Since the second term contains $(x, y)$, it can be written as $(Q \circ S \circ Q) = (Q[x] \times Q[y])$. Moreover, if there would exist $(a, b) \in H(P) \cap (Q[x] \times Q[y])$ then $(a, b) \in H(P)$ and there would also exist $(a, x), (y, b) \in Q$. But then $(x, y) = (x, a) \circ (a, b) \circ (b, y) \in P$, because $Q$ is compatible with $P$, which is a contradiction. ■

Thus, we also can calculate the incremented preference $P^*$ by simple manipulations on $H(P)$ in the case of incremental preference information. Again the necessary set can efficiently be indexed for fast retrieval.

In summary, we have shown that the incremental refinement of skylines is possible efficiently by manipulating only the Hasse diagrams. We will now present our simple to use interface for interactively stating such refinements and also quantify the gain in the run times to calculate incremented skylines by refining the Hasse diagrams instead of relation $P$.

## IV. Experimental Section

In this section, we present our graphical user interface for incremental preference refinement. To show the superior efficiency of our approach, we also present results of runtime and object access measurements for large synthetic datasets and in a realistic scenario. To enable simple user interaction and to confirm our theoretical results in a practical setting, we created a simulation and test application for our incremental skyline approach. The system consists of two main components:
a) an automated experimentation framework used for conducting experiments on synthetic data as used in the next subsection
b) a graphical user interface for incremental preference manipulations.

### A. An user interface for incremental preference elicitation

Our user interface application uses a convenient graph-based approach for modeling a user's preferences. A screenshot of our prototypical implementation is shown in figure 10. A user can pick any desired predicate, add it to the query and model the respective preferences in a personalized fashion. For simplifying the visual representation, only a Hasse diagram of the (generally much larger) preference relation is displayed in the interface. Hence, the approach is especially suitable for partial-order preference modeling and supports the user by providing direct visual feedback and a good overview.

For conveniently modeling preferences, predicate values can be added as nodes to the diagram and placed freely. Preferences and equivalences can simply be drawn as edges between value nodes. Trade-Offs are expressed using special nodes, which amalgamate two predicates as described in sec-

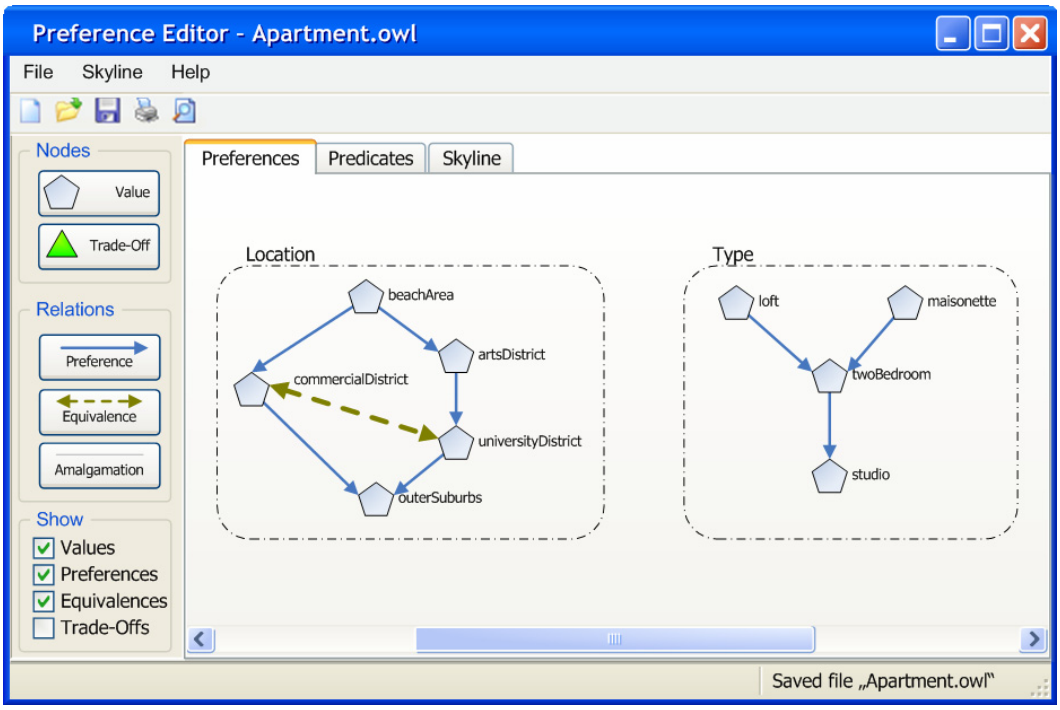


Fig. 10. Preference Editor User-Interface

tion 2. Like normal value nodes, these trade-off nodes can be set into relation to each other using preference or equivalence relations. After the user finished modeling his/her preferences, the respective skyline set is computed and the results displayed. If the result is still unsatisfying, the preferences can be refined and a new incremented skyline is computed using the incremental skyline algorithm described in the previous section.

An especially advanced feature is that the application already can import or export preferences from or to OWL files modeled as described in section 2. This feature is helpful for several tasks, like storing long-term preferences, exchanging preferences between different users or communities, and extending our skyline approach to a more general context where other, more extensive ontologies (e.g. modeling common domain knowledge) may be needed.

### B. *Efficiency measurements for simulated scenarios*

In this subsection we will evaluate the runtime and numbers of object accesses needed for incremental skyline computations for large synthetic datasets. While smaller examples with few predicate dimensions and smaller datasets can be computed quite fast (the example used in the last subsection computes in less than a second for 1000 database objects on a standard PC), the runtime significantly increases when more predicate dimensions are used or the dataset size is increased. Please note that the incrementally computed skylines used during preference refinement can be computed considerably faster than a full skyline, because the size of the data set necessary for a correct computation monotonically decreases, i.e. objects dominated at any point do not have to be reconsidered.

The parameters used in all our evaluations can be found in table 1 and table 2. These parameters reflect a realistic large-scale scenario for skyline applications [4].

TABLE I
BASE PARAMETERS FOR EVALUATIONS

| Quantity | Value |
|---|---|
| Database Size | 100,000 |
| Value Distribution | uniform |
| Number of Query Predicates | 6 |
| Predicates' Domain Size (# distinct attribute values) | 30 |
| Preference Depth (longest path within graph) | 8 |
| Edge Degree (ratio between number of graph nodes and number of edges) | 1.2 |
| Unconnected Degree (ratio between isolated and connected nodes in graph) | 0.05 |

TABLE II
TECHNICAL SETUP FOR EVALUATIONS

| Quantity | Value |
|---|---|
| Processor | AMD Opteron 2400+ |
| Memory | 40 GB RAM |
| Programming Language | Sun Java 5.0.9 |

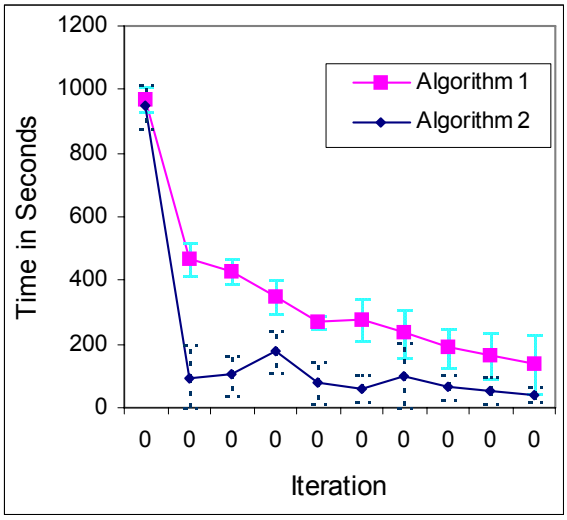


Fig. 10. Time needed to calculate a Skyline increment (seconds) and the corresponding standard deviations.

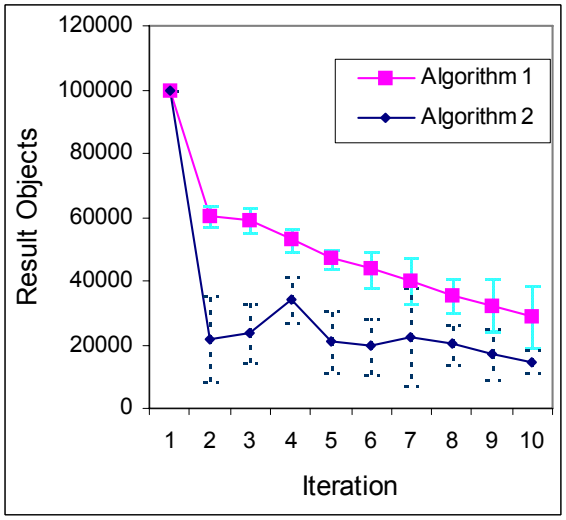


Fig. 11. Database objects candidates considered for Skyline calculation and corresponding standard deviations.

For performing the evaluation, we calculate incremental skylines comparing our algorithm against a baseline and measure the size of the considered candidate set and the resulting runtime. In each run 25 skyline calculations are performed and their results averaged. For each computation, a fresh uniformly distributed dataset and a random preference relation is generated and the skyline is computed. Using the initial preference relation and skyline, the preference relation is refined and the skyline incrementally recomputed. Each refinement step consists of adding randomly either a new preference, an equivalence, or a trade-off edge. For incremental skyline computation, we distinguish two algorithms. While the actual method of skyline computation is identical, the essential difference is in the selection of the database objects which are considered as candidates for the incremented skyline. Basically the incremented skyline will always be a subset of the previous skyline with objects newly dominated by the refined preference removed. As a baseline the first algorithm considers all database objects which were part of the previous skyline as candidates.

The second algorithm considers only those objects involving predicate values which are part of the previous skyline and are also part of $P^* \setminus P$ as described in section 3. Thus algorithm 2 outperforms algorithm 1, because it has to consider less candidates for its skyline computation.

Both algorithms use a very basic approach for actually computing the skyline, in which simply all possible candidate pairs are checked for dominance (complexity of $O(n^2)$). Of course our approach of incrementally computing skylines and restricting the candidate set based on $P^* \setminus P$ can also be integrated into more sophisticated and more efficient algorithms with a better average case complexity like those described in [5] or [15].

The result of our evaluation can be found in figure 10 showing averages and the respective variances of the absolute runtime needed to perform the skyline computation and figure 11 illustrating the size of the candidate set considered. The first displayed data-point in each diagram represents the initial skyline while the skylines recomputed after incremental preference refinement (one refinement between recomputations) are shown along the x-axis.

The initial skyline is computed in 967 sec on average. As stated before, for each increment a smaller candidate set can be considered based on the previous skyline and the newly added preference information. We can see that the size of the candidate set closely correlates with the algorithms' runtimes and that – as could be expected – the runtime for the increments is indeed significantly smaller than the initial skyline's runtime.

As anticipated, the candidate set in our approach (algorithm 2) is much smaller than the candidate set for algorithm 1 as it takes advantage of the locality of preference refinement (see section 3). As a result, algorithm 2 impressively outperforms algorithm 1 throughout our experiments (94 seconds compared to 467 seconds for the first increment). Please note that algorithm 1 performs in a monotonic way, the candidate set and the runtime of the next incremental skyline is smaller or equal to those of the previous increment. In contrast to that, algorithm 2 is not monotonic as the size of its candidate set depends on the size of $P^* \setminus P$ and the distribution of predicate values among database objects.

Although the absolute runtimes of our implementation are still too high to be used in an interactive OLAP (On Line Analytical Processing) fashion, the incremental skyline computation shows promising capabilities for usage in such a scenario, if only the general computation process is accelerated. One possible solution is the deployment of indexes and data statistics as mentioned before.

## V. Conclusion

In this paper we introduced an efficient algorithm to compute incremented skylines driven by user interaction. Our approach allows users to interactively model their preferences and explore the resulting skyline sets by incrementally providing additional information like new preferences, equivalence relations, or acceptable trade-offs. We also proposed an extensible OWL model for storing and exchanging such preferences and equivalences. Moreover, we investigated the efficient evaluation of incremented skylines by considering only those relations that were directly affected by a user's changes in preference information. Finally we implemented and evaluated a graphical user interface based on our approach. Our interface deploys Hasse diagrams for modeling preferences, thus allowing for simple and intuitive preference modifications. Skylines sets are then incrementally computed during the modeling process. This computation takes advantage of the local nature of incremental changes in preference information leading to far superior performance over the baseline algorithms.

Although this work is an advance for the application of the skyline paradigm in real world applications, still several challenges remain largely unresolved. The time necessary for computing initial skylines is still too high hampering the paradigm's applicability in large scale scenarios. Here, introducing suitable index structures, heuristics, and statistics might prove beneficial. Also, though our use of modeling techniques and tools from the area of Semantic Web paves the way towards integrating more complex ontologies and semantics into skyline processing, the modeling framework of OWL is still rather limited and our basic model needs to be improved.

## Acknowledgements

We would like to thank Holger Wache of VU Amsterdam for insightful discussions about chances and limitations in OWL.

## References

[1] W.-T. Balke, U. Güntzer. Multi-objective Query Processing for Database Systems. *Int. Conf. on Very Large Data Bases (VLDB),* Toronto, Canada, 2004.

[2] W.-T. Balke, U. Güntzer, C. Lofi. Eliciting Matters – Controlling Skyline Sizes by Incremental Integration of User Preferences. *Int. Conf. on Database Systems for Advanced Applications (DASFAA), Bangkok, Thailand, 2007.*

[3] W.-T. Balke, M. Wagner. Through Different Eyes - Assessing Multiple Conceptual Views for Querying Web Services. *Int. World Wide Web Conference (WWW 2004)*, New York, USA, 2004.

[4] W.-T. Balke, U. Güntzer, W. Siberski. Exploiting Indifference for Customization of Partial Order Skylines. *Int. Database Engineering and Applications Symp. (IDEAS)*, Delhi, India, 2006.

[5] W.-T. Balke, J. Zheng, U. Güntzer. Efficient Distributed Skylining for Web Information Systems. *Int. Conf. on Extending Database Technology (EDBT)*, Heraklion, Greece, 2004.

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

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

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

[9] L. Chen, P. Pu. Survey of Preference Elicitation Methods. *EPFL Technical Report IC/2004/67*, Lausanne, Swiss, 2004.

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

[11] P. Godfrey. Skyline Cardinality for Relational Processing. *Int Symp. on Foundations of Information and Knowledge Systems (FoIKS)*, Wilhelminenburg Castle, Austria, 2004.

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

[13] V. Koltun, C. Papadimitriou. Approximately Dominating Representatives. *Int. Conf. on Database Theory (ICDT)*, Edinburgh, UK, 2005.

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

[15] D. Papadias, Y. Tao, G. Fu, B. Seeger. An Optimal and Progressive Algorithm for Skyline Queries. *Int. Conf. on Management of Data (SIGMOD)*, San Diego, USA, 2003.

[16] J. Pei, W. Jin, M. Ester, Y. Tao. Catching the Best Views of Skyline: A Semantic Approach Based on Decisive Subspaces. *Int. Conf. on Very Large Databases (VLDB)*, Trondheim, Norway, 2005.

[17] World Wide Web Consortium (W3C). OWL: Web Ontology Language Reference. *W3C Recommendation*, 2004. (http://www.w3.org/TR/owl-ref/)

[18] Y. Yuan, X. Lin, Q. Liu, W. Wang, J. Yu, Q. Zhang. Efficient Computation of the Skyline Cube. *Int. Conf. on Very Large Databases (VLDB)*, Trondheim, Norway, 2005.