Efficient Computation of Trade-Off Skylines

Christoph Lofi
Institute for Information Systems
Mühlenpfordtstr. 23
38104 Braunschweig, Germany
lofi@ifis.cs.tu-bs.de

Ulrich Güntzer
Institute of Computer Science
University of Tübingen
72076 Tübingen, Germany
ulrich.guentzer@informatik.uni-tuebingen.de

Wolf-Tilo Balke
Institute for Information Systems
Mühlenpfordtstr. 23
38104 Braunschweig, Germany
balke@ifis.cs.tu-bs.de

ABSTRACT
When selecting alternatives from large amounts of data, trade-offs play a vital role in everyday decision making. In databases this is primarily reflected by the top-k retrieval paradigm. But recently it has been convincingly argued that it is almost impossible for users to provide meaningful scoring functions for top-k retrieval, subsequently leading to the adoption of the skyline paradigm. Here users just specify the relevant attributes in a query and all suboptimal alternatives are filtered following the Pareto semantics. Up to now the intuitive concept of compensation, however, cannot be used in skyline queries, which also contributes to the often unmanageably large result set sizes. In this paper we discuss an innovative and efficient method for computing skylines allowing the use of qualitative trade-offs. Such trade-offs compare examples from the database on a focused subset of attributes. Thus, users can provide information on how much they are willing to sacrifice to gain an improvement in some other attribute(s). Our contribution is the design of the first skyline algorithm allowing for qualitative compensation across attributes. Moreover, we also provide an novel trade-off representation structure to speed up retrieval. Indeed our experiments show efficient performance allowing for focused skyline sets in practical applications. Moreover, we show that the necessary amount of object comparisons can be sped up by an order of magnitude using our indexing techniques.

Categories and Subject Descriptors
H.3.3 [**Information Systems**]: Information Search and Retrieval.

General Terms
Algorithms, Human Factors

Keywords
Skyline Queries, Preferences, Trade-off Management.

1. INTRODUCTION
Recently skyline queries have successfully implemented the concept of Pareto optimality for filtering suboptimal database items. Any query result item can be safely ignored, if there exists some item that simply shows better values with respect to *all* query attributes: we say the ignored item is '*dominated'*. This is indeed a very intuitive concept. If for example two car dealers in the neighborhood offer exactly the same model (with same warranties, etc.) at different prices, why should one want to consider the more expensive car? Furthermore, user preferences for skyline queries only have to be specified within each attribute (cheaper price, stronger engine, favorite color), which makes the paradigm intuitive to use. And while in most works on skyline queries only numerical domains and preferences are considered [3][11][15], skylining can even be extended to respect categorical preference (e.g. on colors) modeled as partial orders [12][4].

The focus on individual attribute domains and the complete fairness of the Pareto paradigm are major advantages of skyline queries. However, this also induces some shortcomings. Skylines completely lack the ability to relate attribute domains to each other and thus disallow compensation, weighting or ranking *between* domains. This often results in large amounts of incomparable objects and generally causes skylines results to be rather large, especially in the case of anti-correlated attributes. It has been shown that already for only 5 to 10 attributes, skylines can easily contain 30% or more of the entire database instance [2][3][7][8] and thus tend to be unmanageable.

In order to decrease skyline sizes, there have recently been several approaches to rank skyline items according to some structural characteristics like counting how many other items does each skyline item dominate [4][5], or in how many subspace skylines each skyline item is present [17], etc. For the latter approaches, a large number of subspace skylines have to be computed, often employing 'sky-cube'-like algorithms [17][16]. However, it is not quite clear whether those structural properties are really related to the usefulness of items and will be helpful for the user. In contrast to merely structural approaches, a more user-centered focusing of the skyline set can be archived by incorporating feedback information provided by the user, thus introducing an additional layer of personalization. Most prominent among these approaches are techniques allowing users to interactively modify and extend their preferences in a cooperative fashion like [6] or [13]. In particular, users might consider some attributes to be more important than others. For example, [13] allows users to provide a total order for attributes; the evaluation then focuses on prioritized subspaces.

The alternative approach to query personalization is *Top-K Retrieval* [10] paying particular attention to weightings of attributes. For each item high values in some attribute(s) may compensate for shortcomings with respect to other attributes following some scoring function (also called a *utility function*). Unfortunately, this compensation feature is not very intuitive to use. For instance, the semantics of an expression like "0.4*price + 0.8*color + 0.2* performance" is not intuitively understandable by most users.

However, none of the mentioned approaches are able to support simple and natural *trade-offs* or *compromises*, which real world sales persons use every day. Consider, for example, three cars: let

EDBT 2010, March 22-26, 2010, Lausanne, Switzerland.

item A be a 'blue metallic' car for \$18000 and item B be a 'blue' car for \$17000, accompanied by a preference favoring cheaper cars and metallic colors. Looking at the ranking on attribute level, both cars are incomparable with respect to the *Pareto order*: one car is cheaper, the other has the preferred color. In this scenario, a natural question of a car dealer would be, whether the customer is willing to compromise on those attributes, i.e. if he/she is willing to pay an additional \$1000 for the metallic paint job (we call such compromises *trade-offs*). If the answer is yes, then item A is the better choice and would dominate item B with respect to a *trade-off enhanced Pareto order*. Still, if some item C like a 'blue' car for \$15000 exists, A and C would remain incomparable as the premium for the metallic color on that car C is larger than the said \$1000 the user is willing to pay. From the viewpoint of *order theory*, the logical implications and problems of trade-off enhanced Pareto orders were already discussed in detail in our previous work [1]. Moreover, in [14] we presented a simple criterion for checking whether a set of trade-offs and attribute preferences are consistent and non-conflicting.

However, actually computing a trade-off enhanced skyline efficiently from this order has still been impossible. This is because in contrast to Pareto orders, the trade-off enhanced order loses the characteristic of *seperability* [9] regarding the individual attributes. However, exactly this characteristic allows for building efficient skyline algorithms. In particular, when checking any two objects A and B for domination a skyline algorithms does not have to materialize the separable Pareto order. But any two items can be simply checked for domination *componentwise*: If attribute values of item A are better or equal regarding each attribute than item B's (with 'strictly better' in at least one), then A dominates B and B can be pruned from the skyline. Moreover, if two items only differ with respect to a single attribute the test for domination is even simpler due to the *ceteris-paribus* ('all-else-being-equal') semantics, see e.g., [9].

In contrast to preferences, trade-offs always *span over several attributes*. For instance our example of a metallic paint job for \$1000 tightly links the price attribute to the color. Now a componentwise comparison can still sort out all Pareto-dominated items, but as soon as the price of two items differs by at most \$1000, a domination regarding the color attribute might be compensated for. This has to be checked individually (and thus inefficiently) for all trade-off pair. While for a single trade-off this test is still relatively inexpensive, introducing multiple trade-offs ('Would you pay more for a stronger engine?') will inevitably lead to a complex order where attributes are intrinsically linked. But simply materializing the trade-off enhanced Pareto order is not possible for efficient domination checks. Therefore, until now no algorithm for computing skylines enhanced by trade-offs is known.

In this paper we present the *first skyline algorithm able to integrate the intuitive concept of trade-offs* into database retrieval. Our algorithm always correctly computes the complete trade-off skyline and thus offers effective means for personalization. The contribution of this paper is that these characteristics can be obtained by materializing only a simple and non-redundant auxiliary trade-off representation structure. This contribution is twofold: on one hand, more information about the user can be consistently integrated into the retrieval process and thus be used to personalize the result set. On the other hand, the result sets are reduced without any arbitrary assumptions like 'an item is better, if it dominates more other items'[4][5] or 'good items appear in more subspace skylines' [17].

Our paper is structured as follows: we introduce basic theoretical concepts in section 2, starting with a formal definition of trade-offs and corresponding trade-off skylines. We identify the object domination check as the crucial operation when adding trade-offs to the skyline paradigm. In section 3, we introduce our novel algorithm for efficiently computing the trade-off enhanced skyline. The main innovation is our data structure for computing the object domination check. The skyline algorithm and the corresponding representation structure is extensively evaluated in section 4. We show the effectiveness of the skyline reduction gained by trade-offs on real world data sets, and our algorithm's efficient run times suitable for real-life applications. We conclude our work in section 4.3 with a short summary and outlook.

2. BASIC CONCEPTS

A complete order theoretical framework for preference structures is given in [1]. To be self-contained we will briefly reiterate the basic definitions and notation as they will be needed later.

2.1 Pareto Skylines

Assume a database relation $R \subseteq D_1 \times ... \times D_n$ on n attributes.

- A preference P_i on an attribute A_i with domain D_i is a *strict partial order* over D_i. If some attribute value $a \in D_i$ is preferred over another value $b \in D_i$, then $(a, b) \in P_i$. This is written as $a >_i b$ (reads as "a dominates b wrt. to P_i").
- Analogously, an equivalence Q_i is an equivalence relation on D_i compatible with P_i. If two attribute values $a, b \in D_i$ are equivalent, i.e. $(a, b) \in Q_i$, we write $a \approx_i b$.
- Finally, if an attribute value $a \in D_i$ is either preferred over or equivalent to another value $b \in D_i$, we write $a \gtrsim_i b$.

As an example, assume some transitive preferences for buying a car considering the attributes price, color, horsepower, and air conditioning as illustrated in Figure 1:

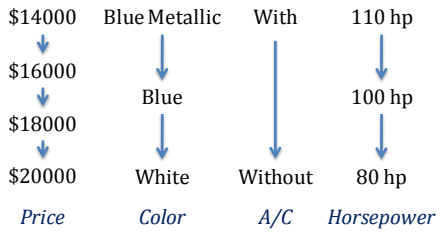

Figure 1. Some preferences within the domain of cars

To compute the skyline, the attribute preferences can be aggregated with respect to the Pareto semantics (so called Pareto aggregation) resulting in the full product order P. This order can then be used to test whether an object o_1 from the database dominates any other object o_2 (which is the case if o_1 shows better or at least equal performance regarding all attributes than o_2):

Definition 1: full product order P

For $P \subseteq (D_1 \times ... \times D_n) \times (D_1 \times ... \times D_n)$ and for any $o_1, o_2 \in (D_1 \times ... \times D_n)$, $(o_1, o_2) \in P$ denoted by $o_1 >_P o_2$ is defined by $\forall\, i \in \{1, ..., n\}$: $o_{1,i} \gtrsim_i o_{2,i} \;\wedge\; \exists\, i \in \{1, ...n\}$: $o_{1,i} >_i o_{2,i}$. □

The skyline of R is then defined as all non-dominated objects of the database R with respect to the full product order:

Definition 2: Pareto skyline

$Sky := \{o_2 \in R \mid \neg\exists\, o_1 : o_1 >_P o_2\}$ (see e.g., [11]). □

Note that product orders quickly grow very large with increasing number of attributes and domain cardinalities. However, for Pareto skyline computation it is not really necessary to *materialize* the full product order. For testing whether $o_1 >_P o_2$ holds, it is only necessary to compare the objects *componentwise* using the individual attribute preferences. The ability to perform the dominance check with respect to the product order by separately testing each component is called *separability* and is heavily exploited by all existing skyline algorithms, see e.g. [2][3][11][15].

2.2 Trade-Off Skylines

Trade-offs can be considered as a user decision between two sample objects focusing on a subset of the available attributes. For example, considering the domain of cars, a user could focus on the attributes color and price. A trade-off then describes in a *qualitative* fashion how much a user is willing to sacrifice in some dimension(s) to gain better performance in some other dimension(s) on the basis of a practical example. A possible trade-off t_1 could be: "I would prefer a car for \$18000 with a metallic blue paint job over a car for \$16000 with a plain blue paint job." We write $t_1 := \big((\$18000, blue\ metallic) \rhd (\$16000, blue)\big)$.

More formally, a trade-off t is always defined over a set of attributes given by respective indexes denoted as $\mu \subseteq \{1, \dots, n\}$ (see Figure 2 for the example trade-off t defined over attributes A_2, A_3, and A_4). To further simplify notation, let $D_\mu := \bigtimes_{i \in \mu} D_i$ and $\bar{\mu} := \{1, \dots, n\} \setminus \mu$. Then, a trade-off is a relationship between two tuples $x, y \in D_\mu$ denoted as $t := (x \rhd y)$. Obviously the two components of a trade-off x and y have to be in incomparable, i.e. no domination relationship $x \geq_P y$, nor $x \geq_P y$ exists between them. Each trade-off is defined on an individual attribute set μ, i.e. some t_1 may be defined on μ_1 and t_2 on μ_2.

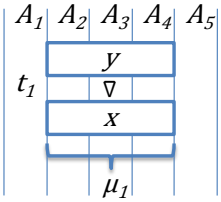

Figure 2. Trade-off t on $\mu := \{2, 3, 4\}$

Trade-offs usually induce new domination relationships between database items augmenting the full product order and thus also the final skyline. In particular, given the above sample trade-off all cars for \$18000 with a blue metallic paint job (or even better features price- and color-wise) will dominate all cars for \$16000 with a plain blue paint job (or even less preferred features) under the ceteris paribus semantics. For example, after introducing t_1, the car $(\$17000,\ blue\ metallic, 100\ hp, a/c)$ would now dominate a (previously incomparable) car with the characteristics $(\$16000, white, 100\ hp,\ no\ a/c)$.

Assuming a set of trade-offs T, the resulting skyline analogously to Pareto skylines consists of all non-dominated objects of R with respect to the new domination relationships induced by all the given trade-offs. It is denoted with $>_T$ in the following (reads as "*dominates with respect to the trade-offs in T*", see the following subsection for a formalization):

Definition 3: trade-off skyline

$Sky := \{o_2 \in R \mid \neg\exists\, o_1 : o_1 >_T o_2\}$ □

This new criterion respects *all trade-offs* in T, but of course has to also respect *all original Pareto preferences*, too. Additionally, it has to also mind all *interactions between trade-offs and Pareto preferences*, which are given by the transitive closure of all preferences and trade-offs. However, this new order and the resulting check criterion are not separable anymore, i.e. the information provided by the trade-offs and their interactions cannot be mapped back to the attribute preferences without losing information (and thus, simple, attribute based comparisons are not sufficient anymore).

2.2.1 Trade-Off Dominance Relationships

Analogously to $>_P$, we can define $>_T$ using the notion of domination relationships by providing an adequate *comparison criterion*.

Definition 4: trade-off domination relationships

The strict pre-order $>_\mathbf{T}$ is defined as the *smallest preorder* containing the full product order induced by the *Pareto semantics*, enhanced by all additional domination relationships induced by *individual trade-offs* or *arbitrary combinations of trade-offs* in $\boldsymbol{T}$ (completed using the *ceteris paribus semantics*). □

In the following we will show inductively how to construct $>_\mathbf{T}$.

For a set $T := \{t_1 := (x_1 \rhd y_2)\}$ containing only a single trade-off the construction is straightforward: given two objects o_1 and o_2, o_1 may either dominate o_2 according to the original Pareto semantics or by actually applying the trade-off using both ceteris paribus and Pareto semantics. The trade-off can only be applied, if for all attributes the trade-off is defined on: o_1 dominates the attribute values of the first trade-off component x_1 and the attribute values of the second component y_1 dominate o_2. Moreover, it is easy to see that a single trade-off can only be applied once [14]. Formally:

Proposition 1: constructing $o_1 >_\mathbf{T} o_2$ with $|T| = 1$

For any $o_1, o_2 \in D_{\{1,\dots,n\}}$ and a single trade-off $x_1 \rhd y_2$: $o_1 >_T o_2$ holds, if

$$(o_1 >_P o_2) \vee \begin{pmatrix} \forall i \in \mu : (o_{1,i} \gtrsim_i x_{1,i}) \wedge (y_{1,i} \gtrsim_i o_{2,i}) \\ \wedge\, \forall i \in \bar{\mu} : o_{1,i} \gtrsim_i o_{2,i} \end{pmatrix}$$ □

Figure 3 visualizes this concept (assuming 4-attribute objects and a 2-attribute trade-off with $\mu = \{1, 2\}$). Please note that all comparisons needed are *component-wise* analogous to the case of Pareto skylines and thus are efficient to perform.

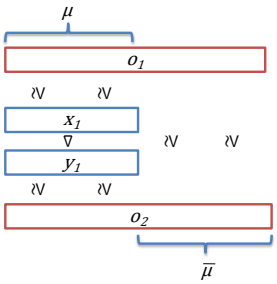

Figure 3. $o_1 >_T o_2$ with a single trade-off

For multiple trade-offs, $>_T$ is more difficult to construct. Let the set T contain two trade-offs $T := \{t_1 := (x_1 \rhd y_1), t_2 := (x_2 \rhd y_2)\}$. Now, o_1 dominates o_2 in either of the following cases:

1. By applying simple Pareto semantics, i.e. $o_1 \succ_P o_2$
2. By applying trade-off t_1 using ceteris paribus semantics, i.e. for attributes in μ_1: $(o_1 \succsim_P x_1) \wedge (y_1 \succsim_P o_2)$ and for all other attributes $(o_1 \succsim_P o_2)$ (cf. proposition 1)
3. By applying trade-off t_2 using ceteris paribus semantics, i.e. for attributes in μ_2: $(o_1 \succsim_P x_2) \wedge (y_2 \succsim_P o_2)$ and for all other attributes $(o_1 \succsim_P o_2)$ (cf. proposition 1)
4. By applying both trade-offs t_1 and t_2. In this case, both trade-offs need to be applied *consecutively* in order to establish a domination relationship between o_1 and o_2, i.e. they form a *trade-off sequence* (consecutive trade-off application will be denoted with $\circ$ in the following). In this case, several options may be possible: $t_1 \circ t_2$, and $t_2 \circ t_1$, but also $(t_1 \circ t_2) \circ t_1$ and $(t_2 \circ t_1) \circ t_1$ (Please note that no longer sequences exist for two *consistent* trade-offs, see [14] for more details).

In general, a trade-off sequence $t_1 \circ t_2$ is possible, if for all attributes appearing in both trade-offs, the second tuple of the first trade-off y_1 component-wise dominates or equals the first tuple of the second trade-off x_2 with respect to the attribute preferences, i.e. $\forall i \in (\mu_1 \cap \mu_2) : (y_{1,i} \succsim_i x_{2,i})$. Formally, the concept of domination via trade-off sequences can be expressed as follows:

Proposition 2: constructing $o_1 \succ_T o_2$ with $|T| = 2$:

Given $t_1 = (x_1 \rhd y_1)$ with μ_1 and $t_2 = (x_2 \rhd y_2)$ with μ_2. To show how the domination relationships are constructed, we will exemplify the forth case from above; the remaining cases are constructed analogously. Assume t_1 and t_2 can be applied in sequence $t_1 \circ t_2$, i.e. $\forall i \in (\mu_1 \cap \mu_2): (y_{1,i} \succsim_i x_{2,i})$. Then, for any $o_1, o_2 \in D_{\{1,\dots,n\}}$ a domination relationship $o_1 \succ_T o_2$ via a sequence $t_1 \circ t_2$ holds, if

$$\begin{aligned}
&\big(\forall i \in \mu_1 : (o_{1,i} \succsim_i x_{1,i}) \\
&\wedge \forall i \in (\mu_1 \cap \mu_2): (y_{1,i} \succsim_i x_{2,i}) \\
&\wedge \forall i \in (\mu_2 \setminus \mu_1): (o_{1,i} \succsim_i x_{2,i}) \\
&\wedge \forall i \in \mu_2 : (y_{2,i} \succsim_i o_{2,i}) \\
&\wedge \forall i \in (\mu_1 \setminus \mu_2): (y_{1,i} \succsim_i o_{2,i}) \\
&\wedge \forall i \in \overline{(\mu_1 \cup \mu_2)}: (o_{1,i} \succsim_i o_{2,i})\big)
\end{aligned}$$

Of course as stated above, also all other possible sequences of trade-offs can be used to establish a domination relationship between o_1 and o_2.

Refer to Figure 4 for an illustration of the trade-off sequence $t_1 \circ t_2$.

Thus, to fully capture the semantics of $o_1 \succ_T o_2$ for two trade-offs, all four previously mentioned cases need to be considered, i.e. testing for Pareto dominance of o_1 and o_2, dominance by considering any single one trade-off, and finally considering all sequences possible with both two trade-offs.

Obviously, the complexity for expressing $o_1 \succ_T o_2$ thus can exponentially increase with the number of trade-offs. For m trade-offs, $o_1 \succ_T o_2$ holds in any of the following cases

- If o_1 dominates o_2 by plain Pareto semantics
- If o_1 dominates o_2 by any single trade-off
- If o_1 dominates o_2 by any possible trade-off sequence which can be formed using two trade-offs
- …
- If o_1 dominates o_2 by any possible trade-off sequence which can be formed using m trade-offs

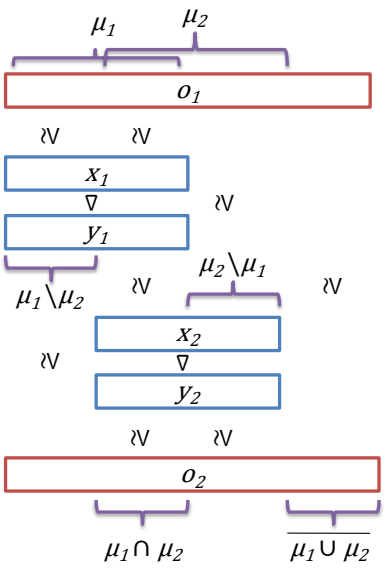

Figure 4. $o_1 \succ_T o_2$ with two trade-offs $t_1 \circ t_2$

2.2.2 *Merging Trade-Off Tuples*

For building an efficient algorithm and ease notation for the theorem and proof in 2.2.3, we now introduce the concept of merging tuples (denoted by $\curvearrowleft$). The idea is to merge two tuples with *differing μ-attribute sets* and subsequently work on the resulting trade-off. The merging operation is not commutative as a 'dominant' tuple is merged with values of a 'weaker' tuple. The merged tuple will have all attribute values where the dominant tuple is defined and all those attributes not defined by the dominant tuple will be completed with the weaker tuple's values. See Figure 5 for illustration.

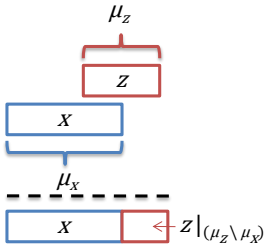

Figure 5. Merged Tuple $(\boldsymbol{x} \curvearrowleft \boldsymbol{z})$

Definition 4: The Merge Operator $\curvearrowleft$:

Let $\mu_x, \mu_z \subseteq \{1, \dots, n\}$. For $x \in D_{\mu_x}$ and $z \in D_{\mu_z}$, the tuple $(x \curvearrowleft z)$ is defined component-wise as:

$$(x \curvearrowleft z) := u \; with \; u_i = \begin{cases} x_i : \; i \in \mu_x \\ z_i : \; i \in (\mu_z \setminus \mu_x) \end{cases}$$ □

The merge operator is particularly helpful for simplifying the attribute-based notation of object domination criterions, because it allows performing the case-differentiation implicitly appearing in the criterions (when two or more trade-offs are involved).

Using the new merge notation, now the condition for object domination via one single trade-off (Proposition 1) may be simplified by merging the non-affected parts of the objects under scrutiny directly into the trade-off conditions (and the separate consideration of attributes in μ and $\bar{\mu}$ is not necessary anymore):

Proposition 1 (cont.): constructing $o_1 \succ_T o_2$ with the merge operator for $|T| = 1$

$$o_1 \succ_T o_2 \Leftrightarrow (o_1 \succ_P o_2) \vee \left((o_1 \succsim_P x_1 \curvearrowleft o_1) \wedge (y_1 \curvearrowleft o_1 \succsim_P o_2)\right) \quad \square$$

An example using our trade-off t_1 introduced above: $t_1 = \left((\$18000, blue\ metallic) \rhd (\$16000, blue)\right)$ is given in Figure 6. Assume that o_1 dominates x_1 with respect to all attributes in μ_1. The trade-off components can now be merged on attributes in $\overline{\mu_1}$ with the values of o_1. Now the (artificially) created intermediate objects containing the trade-off values are in a proper domination relationship ($x_1 \curvearrowleft o_1 \succ_T y_1 \curvearrowleft o_1$) according to the trade-off plus the ceteris paribus semantics (i.e. values in the non-affected parts are equal). If now o_2 is dominated by $y_1 \curvearrowleft o_1$, it is also dominated by o_1 via the trade-off t_1. Note, that only simple and *separable* domination checks -as already used in traditional skyline algorithms- are needed.

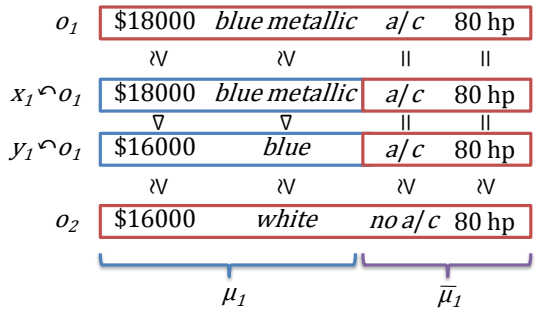

Figure 6: $o_1 \succ_T o_2$ with trade-off $\left((\$18000, blue\ metallic) \rhd (\$16000, blue)\right)$

Of course also the previous condition for $t_1 \circ t_2$ in Proposition 2 may be rewritten using tuple merging. Here we can see the benefit of the merge operator: the need for case differentiation on different subsets combination of μ_1 and μ_2 is completely removed.

Proposition 2 (cont.): constructing $o_1 \succ_T o_2$ with the merge-operator for $|T| = 2$

$$o_1 \succ_T o_2 \Leftarrow (o_1 \succsim_P x_1 \curvearrowleft o_1) \wedge (y_1 \curvearrowleft o_1 \succsim_P x_2 \curvearrowleft y_1 \curvearrowleft o_1) \wedge (y_2 \curvearrowleft y_1 \curvearrowleft o_1 \succsim_P o_2) \quad \square$$

To also understanding this concept, consider again our example for two trade-offs t_1 and t_2 as follows:

$t_1 = \left((\$18000, blue\ metallic) \rhd (\$16000, blue)\right)$
$t_2 = \left((blue,\ a/c) \rhd (\ blue\ metallic,\ no\ a/c)\right)$

These trade-offs can be applied in sequence as either as $t_1 \circ t_2$ or $t_2 \circ t_1$. The common attribute is $color$, and in case of $t_1 \circ t_2$ the value *blue metallic* allows for establishing the sequence. After applying both trade-offs in sequence, as illustrated in Figure 7 the car $(\$18000, blue\ metallic, 80\ hp, a/c)$ dominates the second car $(\$16000, blue\ metallic, 80\ hp, no\ a/c)$ via $t_1 \circ t_2$.

Of course we still have to show that the merging does not in any form alter our dominance checks as originally defined. For brevity we give the proof only for the more complex Proposition 2:

Proof for Proposition 2 (cont.): the component-wise dominance check is equivalent to a dominance check using merging

Transform each of the three clauses using the merge operator $\curvearrowleft$ given in Proposition 2 (cont.) into component-wise comparisons in order to create the comparison terms used in the original Proposition 2. All light-gray printed terms are trivially true and can be ignored. The six resulting component-based comparisons terms are exactly those given by the original Proposition 2.

1) $(o_1 \succsim_P x_1 \curvearrowleft o_1)$
$\equiv \left(\forall i \in \mu_1 : (o_{1,i} \succsim_i x_{1,i}) \wedge \forall i \in \overline{\mu_1} : (o_{1,i} \succsim_i o_{1,i})\right)$
$\equiv \forall i \in \mu_1 : (o_{1,i} \succsim_i x_{1,i})$

2) $(y_1 \curvearrowleft o_1 \succsim_P x_2 \curvearrowleft y_1 \curvearrowleft o_1)$
$$\equiv \begin{pmatrix} \forall i \in (\mu_1 \setminus \mu_2) : (y_{1,i} \succsim_i y_{1,i}) \\ \wedge\, \forall i \in (\mu_1 \cap \mu_2) : (y_{1,i} \succsim_i x_{2,i}) \\ \wedge\, \forall i \in (\mu_2 \setminus \mu_1) : (o_{1,i} \succsim_i x_{2,i}) \\ \wedge\, \forall i \in (\overline{\mu_1 \cup \mu_2}) : (o_{1,i} \succsim_i o_{1,i}) \end{pmatrix}$$
$$\equiv \begin{pmatrix} \forall i \in (\mu_1 \cap \mu_2) : (y_{1,i} \succsim_i x_{2,i}) \\ \wedge\, \forall i \in (\mu_2 \setminus \mu_1) : (o_{1,i} \succsim_i x_{2,i}) \end{pmatrix}$$

3) $(y_2 \curvearrowleft y_1 \curvearrowleft o_1 \succsim_P o_2)$
$$\equiv \begin{pmatrix} \forall i \in (\mu_1 \setminus \mu_2) : (y_{1,i} \succsim_i o_{2,i}) \\ \wedge\, \forall i \in \mu_2 : (y_{2,i} \succsim_i o_{2,i}) \\ \wedge\, \forall i \in (\overline{\mu_1 \cup \mu_2}) : (o_{1,i} \succsim_i o_{2,i}) \end{pmatrix}$$ ■

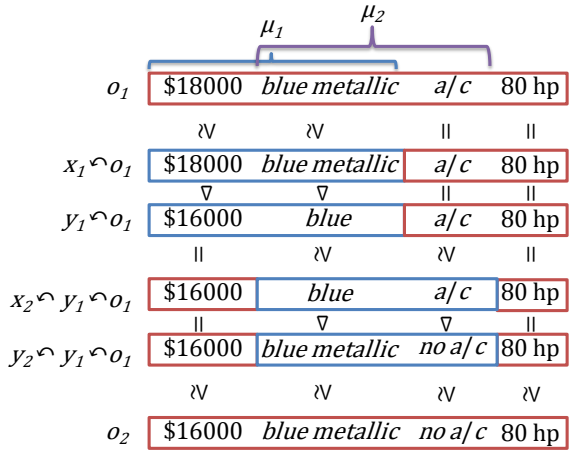

Figure 7. $o_1 \succ_T o_2$ with two trade-offs $t_1 \circ t_2$

2.2.3 *Integrating Trade-Off Chains into Single Trade-offs*

When examining the previous example more closely, it can be observed that the comparisons between components of t_1 and t_2 are independent of o_1 and o_2, i.e. the fact whether two trade-offs can be applied in sequence or not and which effects they have on object comparisons can be pre-computed independently of the database content. More generally, any valid trade-off sequence can be understood and handled as *a single* trade-off by checking for applicability and then combining the respective components. For example, trade-off sequence $t_1 \circ t_2$ acts like a single trade-off $t_{1\circ 2} \coloneqq ((\$18000,\ blue\ metallic, a/c) \rhd (\$16000,\ blue\ metallic,\ no\ a/c))$, i.e. instead of testing $o_1 \succ_T o_2$ with respect to the sequence $t_1 \circ t_2$, also $o_1 \succ_T o_2$ with respect to the *integrated trade-off* $t_{1\circ 2}$ could be tested.

This new trade-off $t_{1\circ 2}$ is defined as following:

Definition 5: trade-off integration

Assume that $t_1 \coloneqq (x_1 \rhd y_1)$ on μ_1, and $t_2 \coloneqq (x_2 \rhd y_2)$ on μ_2, and t_1 and t_2 can be applied in sequence, i.e. . $\forall i \in (\mu_1 \cap \mu_2)\colon (y_{1,i} \gtrsim_i x_{2,i})$. Then a new trade-off $t_{1\circ 2}$ can be defined which is equivalent with respect to object domination to the trade-off sequence $t_1 \circ t_2$. This *integrated trade-off* is given by:

$$\text{If } \forall i \in (\mu_1 \cap \mu_2) \text{ holds } (y_{1,i} \gtrsim_i x_{2,i}) \text{ then}$$

$$t_{1\circ 2} \coloneqq ((x_1 \curvearrowleft x_2) \rhd (y_2 \curvearrowleft y_1)) \qquad \square$$

Theorem 1: Integrating trade-off sequences into a single trade-off does not affect subsequent skyline computation.

Proof (sketch): First consider the integration of two trade-offs: assume $t_{1\circ 2}$ as a single trade-off t_3 and use the domination criterion for a single trade-off as given by Proposition 1. This can be transformed to be equivalent to the component-wise domination criterion for two trade-offs as given by Proposition 2:

$o_1 >_T o_2$ with respect to $t_3 \coloneqq t_{1\circ 2}$ is given by Proposition 1 (cont.) as: $o_1 >_T o_2 \equiv ((o_1 \gtrsim_P x_3 \curvearrowleft o_1) \wedge (y_3 \curvearrowleft o_1 \gtrsim_P o_2))$

Replace x_3 and y_3 with their definition as given by Definition 5

$$\equiv ((o_1 \gtrsim_P (x_1 \curvearrowleft x_2) \curvearrowleft o_1) \wedge ((y_2 \curvearrowleft y_1) \curvearrowleft o_1 \gtrsim_P o_2))$$

Transform both clauses to component-wise comparisons:

$$(o_1 \gtrsim_P (x_1 \curvearrowleft x_2) \curvearrowleft o_1) \equiv \begin{pmatrix} \forall i \in \mu_1\colon (o_{1,i} \gtrsim_i x_{1,i}) \\ \wedge \forall i \in (\mu_2 \setminus \mu_1)\colon (o_{1,i} \gtrsim_i x_{2,i}) \\ \wedge \forall i \in \overline{\mu_1}\colon (o_{1,i} \gtrsim_i o_{1,i}) \end{pmatrix}$$

$$((y_2 \curvearrowleft y_1) \curvearrowleft o_1 \gtrsim_P o_2) \equiv \begin{pmatrix} \forall i \in (\mu_1 \setminus \mu_2)\colon (y_{1,i} \gtrsim_i o_{2,i}) \\ \wedge \forall i \in \mu_2\colon (y_{2,i} \gtrsim_i o_{2,i}) \\ \wedge \forall i \in (\overline{\mu_1 \cup \mu_2})\colon (o_{1,i} \gtrsim_i o_{2,i}) \end{pmatrix}$$

Additionally, $\forall i \in (\mu_1 \cap \mu_2)\colon (y_{1,i} \gtrsim_i x_{2,i})$ holds, since we assumed the existence of the trade-off chain to form $t_{1\circ 2}$. Thus, all six terms (and no additional ones) of the original Proposition 2 have been derived. The theorem then follows by induction over the numbers of trade-offs in the sequence. ■

3. THE TRADE-OFF SKYLINE

In this section, we present the baseline algorithm for computing a trade-off skyline and introduce a basic tree-shaped representation structure for materializing trade-off chains.

3.1 A Basic Trade-Off Skyline Algorithm

Obviously a trade-off skyline is a *subset* of the original Pareto skyline (as the definition of $o_1 >_T o_2$ includes $o_1 >_P o_2$). Thus, a basic algorithm may first compute the Pareto skyline (by any one of the many popular algorithms) and then filter all additional objects which are dominated by another object via any trade-off sequence. Please note that this second domination check is much more expensive than the simple check for Pareto domination, thus interleaving Pareto and trade-off dominance checks generally leads to inferior performance. Checking the domination between objects with respect to trade-off sequences can be summarized as:

- Generate all integrated trade-offs representing *all possible* trade-off sequences
- $o_1 >_T o_2$ holds, *if applying any of the integrated trade-offs* leads to an domination relationship

When using the nested loop approach (as introduced by [11]), this results in the following basic algorithm:

Algorithm 1: Basic Trade-Off Skyline Computation:

Parameters:

- Database relation R on n attributes $A_1, \dots, A_n$
- Attribute preferences $P_1, \dots, P_n$
- Set of trade-offs T

Algorithm:

1. Compute Pareto skyline sky with given database R and preferences $P_1, \dots, P_n$
2. Generate the set off all possible integrated trade-offs T'
3. For all $o_1 \in sky$:
   3.1. For all $o_2 \in sky \wedge o_2 \neq o_1$:
      3.1.1. If $o_1 >_T o_2$ (tested using Algorithm 2)
         3.1.1.1. Remove o_2 from sky

Algorithm 2: Basic Object Domination Test $>_T$

Parameters:

- Database objects o_1 and o_2
- All possible integrated trade-offs T'

Returns: $true$ if $o_1 >_T o_2$, $false$ otherwise

Algorithm:

1. For all trade-offs $t_i \coloneqq (x \rhd y) \in T'$
   1.1. If $((o_1 \gtrsim_P x \curvearrowleft o_1) \wedge (y \curvearrowleft o_1 \gtrsim_P o_2))$ return $true$
   (i.e.: o_1 dominates o_2 via t_i, see Proposition 1 (cont.))
2. Return $false$

The requirement for the set T' is that each integrated trade-off (representing some trade-off sequence) for a given set of basic trade-offs T is contained. While those sequences may only consist of $|T|$ different trade-offs, they may have arbitrary length (i.e. the same trade-off may appear even multiple times in a sequence) as long as each directly adjacent trade-off pair t_j and t_{j+1} in that sequence fulfills the sequencing condition $\forall i \in (\mu_j \cap \mu_{j+1})\colon (y_{j,i} \gtrsim_i x_{j+1,i})$. Please note that sequences of unlimited length can only occur for *inconsistent trade-off sets* which can be efficiently tested and excluded before skyline computation (see [14] for theoretical results on possible sequence patterns, sequence finiteness, and the consistency test), thus the length of each trade-off sequence and the number of sequences are both finite.

Obviously, simply generating and testing all these integrated trade-offs without further optimizations is not very efficient. Thus, in the following we introduce a compact representation for performing these tests with the minimal possible efforts.

3.2 Representing Trade-Off Sequences

The idea is to incrementally create a tree structure (called trading tree ($TTree$) in the following) enumerating all currently possible trade-off sequences (represented by integrated trade-offs). In its basic version, the tree structure does not provide significant advantages over a simple list of all integrated trade-offs. However, the tree structure will enable our optimizations in section 3.3.

Each node of the tree denotes a trade-off, which is either a user provided trade-off or some integrated trade-off representing a trade-off sequence (as described in Definition 5). Initially, the tree starts with a single, empty root trade-off $t_\emptyset \coloneqq (( ) \rhd ( ))$ with empty $\mu_\emptyset \coloneqq \emptyset$, i.e. nothing is traded. Then, trade-offs are inserted one by one from the set of all user trade-offs T. To complete the

tree, for each existing node we test whether the trade-off sequence it represents can be extended with the newly inserted user trade-off (the root can always be extended, i.e. $t_\emptyset \circ t_i = t_i$). Also, every newly created node needs to be extended with all previously inserted trade-offs if possible:

Algorithm 3: Basic Algorithm for Trading Tree creation:

Parameters:

- Attribute preferences $P_1, \dots, P_n$
- Set of trade-offs T

Returns: A tree containing all integrated trade-offs possible with trade-off set T and attribute preferences $P_1, \dots, P_n$

Algorithm:

1. $TTree := \{t_\emptyset\}$
2. $includedT := \emptyset$
3. For each user trade-off $t_i := (x_1 \rhd y_1) \in T$
   - 3.1. $includedT := includedT \cup t_i$; $newnodes = \emptyset$;
   - 3.2. For each existing tree node $t_o \in TTree$
     - 3.2.1. Test whether $t_o \circ t_i$ is possible, i.e.: $\forall i \in (\mu_1 \cap \mu_2) : (y_{1,i} \gtrsim_i x_{2,i})$
       - 3.2.1.1. Create integrated trade-off $t_{o \circ i}$ and add to $TTree$ by attaching to t_o
       - 3.2.1.2. Add $t_{o \circ i}$ to $newnodes$
   - 3.3. For each newly created node $t_n \in newnodes$
     - 3.3.1. For each user trade-off $t_u \in includedT$
       - 3.3.1.1. If $t_n \circ t_u$ is possible, i.e.: $\forall i \in (\mu_n \cap \mu_u) : (y_{n,i} \gtrsim_i x_{u,i})$
         - 3.3.1.1.1. Create integrated trade-off $t_{n \circ u}$ and add to $TTree$ by attaching to t_n
         - 3.3.1.1.2. Add $t_{n \circ u}$ to $newnodes$

As an example, consider the trade-offs t_1 and t_2 on {price, color} and {color, air conditioning} from above:

$t_1 = (\$18000, blue\ metallic) \rhd (\$16000, blue))$
$t_2 = ((blue,\ a/c) \rhd (\ blue\ metallic,\ no\ a/c))$

Let us insert t_1 first and directly attach it to the root (see Figure 8). The integrated trade-off $t_{1 \circ 1}$ is obviously not possible (but would directly result in a cyclic inconsistency).

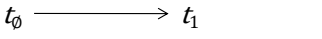

Figure 8. Basic TTree for $T := \{t_1\}$

Next, t_2 is inserted and also attached to the root. Now, the node t_1 has to be tested whether it can be extended, i.e. if a trade-off sequence $t_1 \circ t_2$ is possible (c.f. Proposition 2). In our case, it is possible and thus the node $t_{1 \circ 2}$ is created as an integrated trade-off (see Definition 5). Since the node $t_{1 \circ 2}$ is new, we still have to test whether this sequence can be extended to either form $t_1 \circ t_2 \circ t_1$ or $t_1 \circ t_2 \circ t_2$, but evaluating the conditions only the first case is possible. The same procedure is also applied to the newly appended leaf node t_2. The resulting TTree is shown in figure 9.

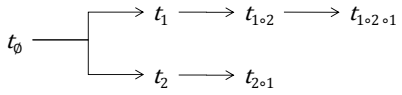

Figure 9. Basic TTree for $T := \{t_1, t_2\}$

Now, consider a third trade-off:

$t_3 := ((\$15000, 100\ HP) \rhd (\$14000, 80\ HP))$.

This trade-off can, for example, be applied in sequence with t_2, allowing the sequence $t_2 \circ t_3$. It cannot, however, be integrated with t_1 (thus neither $t_1 \circ t_3$ nor $t_2 \circ t_1 \circ t_3$ is possible). Considering all possibilities, this results in the following tree (Figure 10):

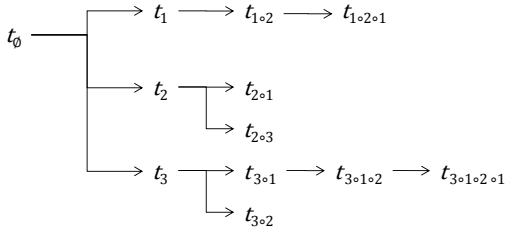

Figure 10. Basic TTree for $T = \{t_1, t_2, t_3\}$

It can also be observed that each node within a branch represents an integrated trade-off sequence extending the sequences of all its ancestors. Still, each integrated trade-off has two distinct components: an upper part (called y in Figure 2) and a lower part (called x in Figure 2). When comparing the individual components of a node to the components of any of its ancestors, it can be observed that the complete values of any ancestor's upper part are by construction also contained in the upper part component of all its descendents (this directly follows from the definition of integrated trade-offs (Definition 5) and the merge-operation (Definition 4)). Thus, for each trade-off integrated into some existing trade-off, the ancestor's upper part component is only extended by values for previously undefined attributes. As soon as all attributes have been defined, the upper part component of all trade-offs in a branch becomes stable; integrating additional trade-offs into it, has no effects anymore. Thus, it is possible to store just the lower part component of trade-offs in the $TTree$ in order to save memory, and just link the first component from the respective parent.

To integrate the trading tree into the basic skyline algorithm, Algorithm 3 has to be used in Algorithm 1 for computing all trade-off sequences, and the nodes of the tree are treated as the set of possible trade-offs.

3.3 Removing Redundancy from the Tree

Although the previously described $TTree$ allows for systematically generating all possible trade-off relationships, it still tends to grow rather large. But taking a closer look, it still contains a lot of redundant information severely hampering the performance of domination check operations. In the following, we exploit some structural properties of the $TTree$ to remove all redundancies with respect to object domination.

For example, considering the sample tree in the previous section, it contains both integrated trade-offs $t_{2 \circ 3}$ and $t_{3 \circ 2}$. However, both of these trade-offs are *identical*, i.e.: $t_{2 \circ 3} = t_{3 \circ 2} := (\$15000,\ 100\ HP, blue,\ a/c) \rhd (\$14000,\ 80\ HP, blue\ met.,\ no\ a/c)$.[1] Obviously, with respect to testing whether $o_1 \succ_T o_2$ either one of the two integrated trade-offs would do, whereas the other one is redundant. Also, any future extension which can be applied to

[1] Please note, that due to the not commutative merging operation also the integration of trade-offs is generally not commutative, but here the trade-offs are identical because $\mu_2 \cap \mu_3 = \emptyset$.

either trade-off, could also be applied to the other one. Thus, it is valid to completely remove either node without any loss of the $TTree$'s functionality.

Actually, the case of two trade-offs being identical is just a special case of the more general concept of trade-offs subsuming other trade-offs. A trade-off t_1 is said to be *subsumed* by another trade-off t_2, written as $t_1 \subseteq t_2$, if whenever a domination relationship $o_1 >_T o_2$ established by using any trade-off sequence containing t_1, holds, there is also a domination relationship using a trade-off sequence containing t_2. This can be interpreted as t_2 being 'more general' or 'broader' than t_1. In particular, t_2 can act as t_1's proxy without any loss of information with respect to the domination between objects. Thus, t_1 can completely be removed from the tree, as long as t_2 is still present. Formally, the criterion for subsumption is given by the following:

Definition 6: subsumption criterion for $t_1 \subseteq t_2$:

For two trade-offs $t_1 := (x_1 \rhd y_1)$ and $t_2 := (x_2 \rhd y_2)$,

$t_1 \subseteq t_2$, iff $\big(x_1 \geq_P (x_2 \curvearrowleft x_1)\big) \wedge \big((y_2 \curvearrowleft y_1) \geq_P y_1\big) \wedge \mu_2 \subseteq \mu_1$

For visualization, consider Figure 11:

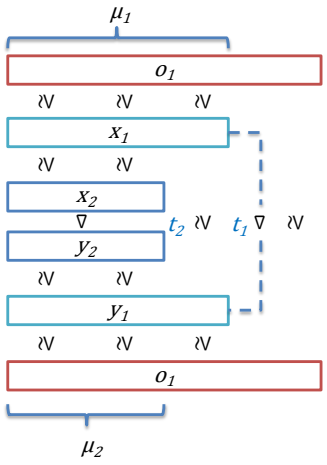

Figure 11. Trade-Off $(x_1 \rhd y_1)$ subsuming $(x_2 \rhd y_2)$

The correctness of the subsumption criterion is easy to see considering the above visualization for $t_1 \subseteq t_2$: Every time $o_1 >_T o_2$ via t_1 holds, then domination relationship also holds via the less restrictive t_2, i.e. $\forall i \in \mu_1 : ((o_1 \geq_i x_1) \wedge (y_1 \geq_i o_2)) \Rightarrow \forall i \in \mu_2 : ((o_1 \geq_i x_2) \wedge (y_2 \geq_i o_2))$ is true, iff $t_1 \subseteq t_2$.

As an example, consider following trade-offs:

$t_1 = \big((\$18000, blue\ metallic) \rhd (\$16000, blue)\big)$

$t_4 = \big((\$19000, blue\ metallic) \rhd (\$15000, blue)\big)$

Then, $t_1 \subseteq t_4$ because $\$18000 \geq_{price} \$19000$ and $\$15000 \geq_{price} \$19000$ holds (again, don't be confused here: the cheaper a car, the better it is, hence $cheap \geq_{price} expensive$) and the color attributes is identical within both trade-offs. This means that t_4 is less restrictive than t_1, or in other words: any object dominations that can be established via t_1 could also be established via t_4. Unlike in our previous example where two trade-offs are identical (i.e. subsuming each other), this is not the case here. For example, consider a blue-metallic car o_1 for \$18500 and a blue car o_2 for \$15500 with o_1 being more desirable than o_2 with respect to horsepower and air-condition dimensions. Then, $o_1 >_T o_2$ via t_4, but not via t_1. (see Figure 12).

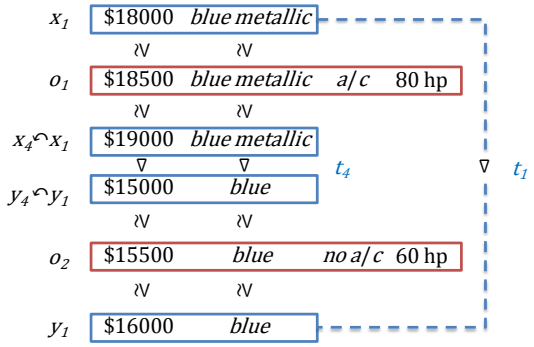

Figure 12. Subsumption $t_1 \subseteq t_4$

Furthermore, this concept also extends to integrated trade-off sequences: any sequence possible with the subsumed trade-off t_1 is also possible with t_4. To better understand this fact, consider an trade-off t_5 such that $t_{1 \circ 5}$ is possible, i.e. $\forall i \in (\mu_1 \cap \mu_5) : (y_{1,i} \gtrsim_i x_{5,i})$. Then also $t_{4 \circ 5}$ can be created, as it requires $\forall i \in (\mu_4 \cap \mu_5) : (y_{4,i} \gtrsim_i x_{5,i})$ and this is always true as $\mu_4 \subseteq \mu_1$ and $\forall i \in \mu_4 : (y_{4,i} \gtrsim_i y_{1,i})$ per Definition 6.

To account also for subsumption, Algorithm 3 needs to be extended in the following way: for new each node t_n which is created check, whether it is subsumed by or does subsume any node already present in the tree. If it is subsumed by an existing node, discard t_n as it does not provide any additional information. If t_n does subsume an already existing node t_o, discard t_o and all its children (because t_n will express all domination relationships previously handled by t_o). Picking up the example started in the previous section in Figure 10, the trade-off $t_{3 \circ 2}$ would have never been created as it is subsumed by $t_{2 \circ 3}$ (Figure 13).

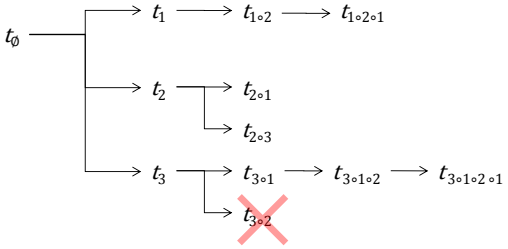

Figure 13. TTree for $T = \{t_1, t_2, t_3\}$ with subsumption

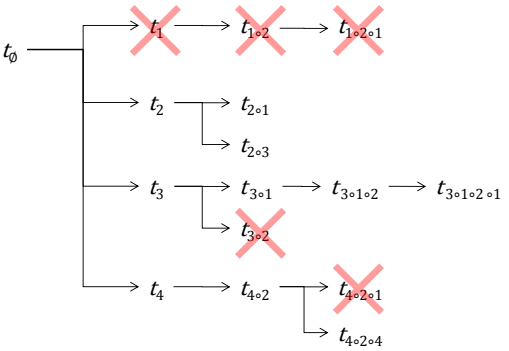

Figure 14. TTree for $T = \{t_1, t_2, t_3, t_4\}$ with subsumption

When adding $\boldsymbol{t_4}$, the algorithm deduces that it subsumes $\boldsymbol{t_1}$ and thus $\boldsymbol{t_1}$'s complete subtree is removed, resulting in the tree illustrated in Figure 14 (please note that for the sake of clarity, the tree was illustrated after $\boldsymbol{t_4}$ was attached to the root and expanded but before it was attached to either $\boldsymbol{t_2}$ or $\boldsymbol{t_3}$).

The supsumption extension of Algorithm 3 can be achieved by replacing the code block in step 3.3 with the following:

Algorithm 4: Extension for Reducing Tree Redundancy

3.3. For each newly created node $t_n \in newnodes$
  3.3.1. For each existing tree node $t_o \in TTree$
    3.3.1.1. If $t_n \subseteq t_o$
      3.3.1.1.1. Remove t_n from $newnodes$ and from $TTree$
      3.3.1.1.2. Continue with 3.3
    3.3.1.2. If $t_o \subseteq t_n$
      3.3.1.2.1. Remove t_o and all its descendants from TT
  3.3.2. For each user trade-off $t_u \in includedT$
    3.3.2.1. If $t_n \circ t_u$ is possible, i.e.: $\forall i \in (\mu_n \cap \mu_u) : (y_{n,i} \gtrsim_i x_{u,i})$
      3.3.2.1.1. Create integrated trade-off $t_{n \circ u}$ and add to TT by attaching to t_n
      3.3.2.1.2. Add $t_{n \circ u}$ to $newnodes$

3.4 Indexing for Improved Match-Making

When testing whether $o_1 >_T o_2$ as described previously in Algorithm 2, all possible trade-off sequences are tested one by one to check, if any of them establishes a domination relationship between o_1 and o_2. In the worst case, this means that *all* sequences are tested, e.g. in the quite usual case that there is no trade-off domination between the two objects and both remain in the trade-off skyline. Obviously this is not very efficient. An approach to remedy this problem is to check only those trade-off sequences which *potentially* could establish a domination relationship, while ignoring those which definitely cannot do so. A simple heuristic for implementing this idea is to first determine all trade-offs in T which potentially may dominate o_2 (independently of o_1), i.e. $potentialEnd := \{t := (x \rhd y) \in T \mid \forall i \in \mu: y_i \gtrsim_i o_{2,i}\}$. Then, only those integrated trade-offs (i.e. sequences) in the trading tree can be selected for subsequent domination checks ending with one of these potentially applicable trade-offs. Due to construction of the integrated trade-offs, their respective lower part components always contain the attribute values of the lower component of the *last* trade-off in the respective sequence, see Definition 5. Therefore, any integrated trade-off which does not fulfill the $potentialEnd$-condition ($\forall i \in \mu : y_i \gtrsim_i o_{2,i}$), thus has no chance to dominate o_2 (cf. Proposition 1). For selecting all integrated trade-offs ending with a potentially applicable trade-off identified in $potentialEnd$, a simple index (e.g., hash table) can be used. Then, only the selected trade-off sequences will be tested for establishing a domination relationship between o_1 and o_2. We will refer to this method as *1-way-indexing* of the trading tree.

Moreover, the same argumentation can also be applied for the *first* trade-off in each trade-off sequence: identify all those objects which may potentially dominate any (simple or integrated) trade-off, that means $potentialStart := \{t := (x \rhd y) \in T \mid \forall i \in \mu_i : o_{1,i} >_i x_i\}$. Thus, if out of all sequences ending with any $t_i \in potentialEnd$, only those starting with any $t_j \in potentialStart$ are selected for dominance testing, no domination information is lost. This selection can actually be performed quite efficiently, if two indexes are folded into each other (a 2-dimensional index). Then, for establishing $o_1 >_T o_2$ only the selected sequences have to be tested. We will refer to this approach as *2-way-indexing*.

Assuming that all sequences in the trading tree have been indexed by their respective last and first trade-off, Algorithm 5 may be used for determining $o_1 >_T o_2$ instead of the basic Algorithm 2.

Algorithm 5: Object Domination $>_T$ with 2-way-indexing:

Parameters:

- Database objects o_1 and o_2
- All possible integrated trade-offs IT, i.e. the set of nodes of $TTree$
- Index I on all trade-offs in IT by sequence end trade-off and sequence start trade-off, i.e. $I(t_{end}, t_{start})$

Returns: $true$ if $o_1 >_T o_2$, $false$ otherwise

Algorithm:

1. $potentialStart := \{t := (x \rhd y) \in T \mid \forall i \in \mu: o_{1,i} >_i x_i\}$
2. $potentialEnd := \{t := (x \rhd y) \in T \mid \forall i \in \mu: y_i >_i o_{2,i}\}$
3. $potentialT := \emptyset$
4. For all $t_e \in potentialEnd$
  4.1. For all $t_s \in potentialStart$
    4.1.1. $potentialT = potentialT \cup I(t_e, t_s)$
5. For all $t := (x \rhd y) \in potentialT$
  5.1. If $((o_1 \gtrsim_P x \curvearrowleft o_1) \wedge (y \curvearrowleft o_1 \gtrsim_P o_2))$ return $true$ (i.e.: o_1 dominates o_2 via t, see Proposition)
6. Return $false$

3.5 Correctness of the Pruned Tree

In this section, we will briefly discuss the correctness of the pruned trading tree for improved skyline computation efficiency by looking at its completeness and minimality.

Algorithm 3 will incrementally insert user-provided trade-offs into the $TTree$ structure and test for each individual trade-off whether it may prolong any already existing sequence (which will result in creating a new node). For any newly created node, it then also has to be checked whether any of the already known user trade-offs can prolong the sequence. This might create a new node which, in turn, is also checked, if it can be prolonged by any user trade-off, and so on. As previously discussed, endless sequences are not possible, and also the number of sequences is finite. Thus, each newly created node is recursively prolonged as far as the current set of trade-offs allows and a new trade-off will also expand recursively any existing node whenever possible (which then also are prolonged). Up to this point, the tree is complete (and basically enumerating all possibilities). However, when adding subsumption constraints, for each newly created node Algorithm 4 checks whether there is any subsumption relationship between the new node and any other existing node. But as we have argued in section 3.3, any object domination relationship which was possible with the subsumed (less general) trade-off is also possible with the subsuming (more general) trade-off. Furthermore, this is also true for integrated trade-off sequences: as the subsuming trade-off is more general, it will form at least all those sequences which could also be formed with the subsumed trade-off. Thus, the pruned tree still remains *complete* with respect to the object domination, if new trade-offs are added.

Following up on the example that was given in Figure 12, it can be shown that the tree resulting after subsumption is indeed *redundancy free*, i.e. the removal of any additional integrated trade-off will inevitably result in an incomplete set of domination crite-

ria (thus, some object domination relationships cannot be discovered anymore). The formal proof by induction over the tree size is relatively straightforward to perform, and is omitted for brevity reasons.

4. EXPERIMENTAL EVALUATIONS

In this section, we first perform a quantitative evaluation of the performance of our algorithmic features using synthetic performance tests. In particular we show the manageability of our $TTree$ data structure and the considerable performance gain by our indexing scheme. To investigate the real life use of our trade-off skylines, we then perform a qualitative analysis on trade-off interactions with real-world datasets and show how using only a small number of trade-offs can significantly reduce skyline sizes. Finally, we show that also the run-times of our trade-off skyline algorithm scale well for large datasets.

All our experiments were executed on a simple notebook computer featuring an Intel Core 2 Duo 2.33 GHz CPU with 4.0 GB RAM and Sun Java 6.

4.1 Trading Tree Size

For each evaluation on synthetic datasets, we randomly generated sets of consistent trade-offs to be integrated. The underlying database relation has 6 attributes with domains of about 20 distinct values each (based on values often occurring in e-commerce settings). Trade-offs are chosen randomly on two to four of those attributes. As default, up to 10 trade-offs are used per set (which is already quite a lot considering that trade-offs have to be elicited individually from the user).

In our first evaluation, 10,000 consistent trade-off sets are generated and the respective trading trees are constructed without subsumption and with subsumption. We measured the tree sizes (i.e. number of tree nodes) and analyzed them according to different percentage quantiles (e.g. "2%-quantile = 139" means 2% of all trees are smaller than 139 nodes). The measured results can be found in Table 1 and Figure 16. Looking at the first few quantiles, the advantage of subsumption are not too pronounced. However, subsumption really takes off for large-sized trees (which usually involves a lot of redundancy). Thus, the average size for trading trees with subsumption is just about half of the average size of trees without subsumption (46%). This size difference will later manifest itself in a significant performance advantage, when the $TTree$ structure is used for actual skyline computation.

Table 1: Percentage quantiles of trading tree sizes with (w.s.) and without (wo.s.) subsumption.

T-Tree	2%	25%	Median	75%	98%	Mean
w/o.s.	139	445	1006	2696	49812	6364
w.s.	126	388	830	2029	27163	2929

4.2 Comparison Speed-Up Using Indexes

In this experiment, we also used the trade-off sets from the previous evaluation and tested the pure performance of object domination tests: "$o_i \succ_T o_j$?". For each trade-off set 10,000 pairs of skyline objects (i.e. pairwise Pareto incomparable) were generated and checked for dominance with respect to the trade-off set. For each object pair, we used the basic algorithm (Algorithm 2), an algorithm with 1-way-indexing, and the algorithm for 2-way-indexing (Algorithm 5), each with trading trees with and without subsumption respectively. The measured values, as shown in Figure 16 and Table 2, have been normalized to reflect comparisons per second, thus larger numbers indicate more efficient algorithms. It is clearly visible that for all algorithms using trading trees with subsumption are roughly performing 2.5 times as fast as those without. Also, the effect of sequence indexing within the tree is very strong: Using 1-way-indexing in contrast to the basic algorithm results into roughly 8 times increased performance, while using 2-way-indexing grants an additional increase by more than 6 times, i.e. on average 50 times faster than the baseline.

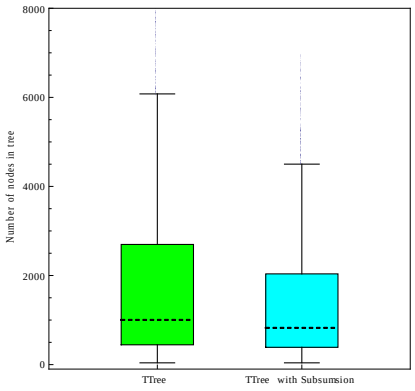

Figure 15. TTree size without and with subsumption

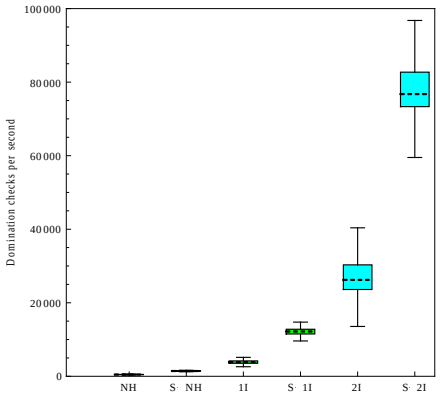

Figure16. Domination checks per second for all algorithms: No Hashing (NH), 1-Way-Indexing (1I), 2-Way-Indexing (2I) (all algorithms are shown without and with subsumption (S-))

Table 2: Object comparison per second

	Basic		1-Way-Index		2-Way-Index	
T-Tree	wo.s.	w.s.	wo.s.	w.s.	wo.s.	w.s.
2%	280	1,190	2,090	8,859	14,063	60,335
25%	439	1,393	3,563	11,521	23,602	73,346
Median	479	1,448	3,840	12,177	26,217	76,740
75%	524	1,495	4,207	12,806	30,315	82,719
98%	2,717	3,755	25,566	46,554	137,874	166,196
mean	590	1,573	4,987	13,760	32,441	82,189

4.3 Computing Trade-Off Skylines

To showcase the effect of trade-offs on the skyline size reduction, as well as the practical run-times, we actually computed some trade-off skylines using two typical real world datasets. Our real

world sets are genuine E-commerce data crawled in August 2009 and contain 998 notebook offers (http://www.shopping.com) and 1,350 real estate offers from the German city of Hamburg (http://www.immobilienscout24.de), respectively.

The following evaluations can only be of a *qualitative* nature as trade-offs need to be semantically meaningful within their domain in order to be effective. In contrast arbitrarily chosen trade-offs (like used for the quantitative measurements) will rarely have useful effects on the actual skyline size. As the problem of suggesting and eliciting suitable trade-offs (see e.g. recent work on example critiquing) is beyond the scope of this work, we simply show that by even using very basic and straightforward heuristics, already significant reductions in skyline size are possible.

The base idea of most trade-off suggestion algorithms is to reduce incomparability induced by a high degree of anti-correlation between two attributes and their respective correlated attributes. Consider our notebook dataset which features the following six numerical attributes: CPU speed, ram capacity, hard drive capacity, screen size, weight, and price. Based on the data, trade-offs can be straightforwardly suggested as follows: First, the two attributes with the strongest degree of anti-correlation are determined and their respective domain values are clustered. The attribute which shows clusters with a higher degree of separation and inner cohesion is selected as main attribute. Then the centroid values of the larger clusters form the base of one half of the suggested trade-offs, e.g. for notebooks the main attribute is screen size with 15", 14.1", 13.3", etc as clusters. These trade-offs' components are then expanded by the cluster's average values of all attributes which are correlated or anti-correlated with the main attribute, e.g. correlation coefficient $\rho \geq 0.5$ or $\rho \leq -0.5$. For notebooks, these attributes are main memory, cpu speed, and weight, resulting in trade-off parts representing average 15", 14.1", and 13.3" with respect to the selected attributes, e.g. (15", 3203 MB, 2082 MHz, 2.7 kg). Depending on the current scenario, two of these trade-off fragments are composed to a trade-off.

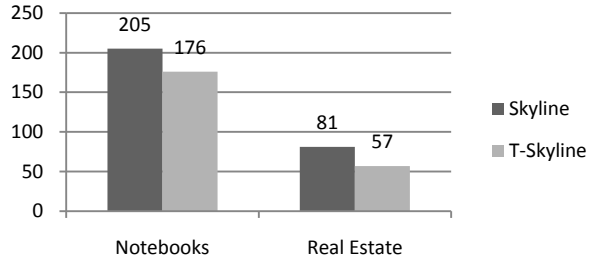

Figure 17. Decrease of skyline size after integrating only two trade-offs

For our *notebook dataset* of 998 items, the Pareto skyline still contains 205 items. Obviously still too many to process manually, and –as is the nature of skylines– also far too diverse. Thus, assume for example that a user intends to buy a *mobile office laptop* for writing papers on the road. Then a 15" screen size might be optimal for a given user while larger or smaller screens are less desirable. This information can be used to generate several trade-offs, e.g. ((15", *cluster avg. values*) ⊳ (13.3", *cluster avg. values*)), ((15", *cluster avg. values*) ⊳ (17", *cluster avg. values*)), etc. In order to express a further emphasis on the preferred cluster, the average values of the first part of the trade-off are slightly shifted in the less desired direction by ¾ of their standard deviation while the second part is improved by ¾ of the respective deviation. This result for example in the following trade-off typical for current mobile office laptops: (15", 2302 MB, 1980 MHz, 2.9 kg) ⊳ (13.3", 3909 MB, 2297 MHz, 1.71 kg). Incorporating two such trade-offs decreased the skyline size on average already by 15% to 176 items (see Figure 17). The resulting trade-off skylines also show a higher degree of focus, e.g. if emphasis is given on 15" notebooks, sub-par smaller or larger notebooks are removed if there is a corresponding good 15" notebook. However, in contrast to strict filtering, the Pareto characteristics are still maintained, e.g. sub-par 13.3" notebooks are only removed if a corresponding good 15" notebook exists; any outstanding 13.3" notebook will still remain in the skyline.

We then did the same for the crawled *real estate dataset*. Again starting with the attribute pair showing the highest degree of anti-correlation, a trade-off was elicited. In this case the trade-off focused on space vs. price and we focused on rather spacey apartments and thus the trade-off can be relied on to remove average small-sized, but inexpensive apartments. For the real estate dataset originally consisting of 1350 items, the initial skyline set had still 81 items. When only integrating two trade-offs with respect to space and price this on average was already decreased by 30% (see Figure 17).

On this practical data also the *computation performance* is very high. For the notebook dataset the computation of the initial skyline took 4 ms using an optimized block-nested loop implementation, while the trade-off skyline consumed additional 42 ms on average. For the real estate data set, the respective values are 3 ms and 12 ms. Of course we also wanted to investigate how the runtime performance of our algorithms scale for larger collections. Thus, we used a database of 50.000 items (which is already rather extensive for e-shopping portals). For the performance measurement we integrated up to 10 (again randomly chosen) trade-offs in a skyline query and calculated the respective results. The runtime results are presented in figure 18. All three algorithms (naïve baseline, 1-way indexing, and 2-way indexing) have been used with subsumption check. We can see that due to the non-indexed baseline is entirely out of scope starting at 3 seconds for only 2 trade-off, and quickly deteriorating (more than 3 minutes for 10 trade-offs). In contrast, the indexed version perform much better and even for 10 trade-off stay in the range of only a few seconds (including the time to build the index). Again we can see that the 2-way indexing clearly outperforms the 1-way indexing. Overall, these experiments show the practical applicability of trade-off skylines in real world scenarios like e.g., e-shopping.

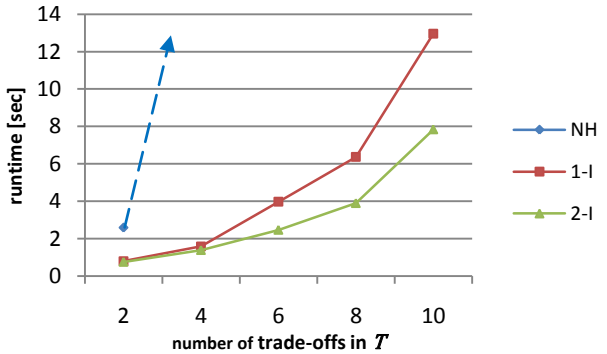

Figure 18. Runtime for integrating random trade-off into an independent dataset with $d = 6$ and $n = 50,000$

5. SUMMARY & OUTLOOK

In this work we extended the well-established skyline paradigm with the concept of *trade-offs*, which allow for compensation between individual attribute dimensions in a *qualitative* fashion. Compensating (or trading) between different choices is indeed a very natural concept, frequently encountered in every days' decision processes. Trade-offs help to increase the focus and manageability of skylines without any arbitrary assumptions beyond the control of the user. However, up until now it was not possible to augment the strict Pareto semantics of traditional skylines with the compensating semantics of trade-offs. This is mainly caused by the fact that trade-offs break the convenient property of seperability when performing object domination tests, an integral component within any existing skyline algorithm.

In this paper, we introduced the formal notion of trade-off skylines and designed the first trade-off skyline algorithm. We developed the object domination test criterion for skyline computation with respect to trade-offs. After establishing the formal properties, we derived a baseline trade-off algorithm which refines a given Pareto skyline with additional trade-off semantics. But in contrast to this baseline, our final algorithm does not rely on the exponential materialization of the full product order. To improve the performance, we showcased a tree data structure $TTree$ containing all necessary trade-off sequences to decide whether two objects dominate each other or not. This data structures especially paid attention to avoiding the storage of redundant information, thus saving storage space and computation time. In fact, it increases performance roughly by the factor of one order of magnitude over the baseline. Furthermore, we introduced an efficient two indexing scheme for the $TTree$ structure providing an additional performance boost by roughly the factor of 50 over the impractical baseline. Finally, a qualitative investigation of the skyline size reduction, and runtime measurements confirmed the benefits and practical applicability of our approach in real world scenarios like e.g., e-shopping.

After establishing the theoretical foundations for trade-off skylines in this work, our future focus will be twofold: on one hand, the psychological aspects of the user interaction with trade-off skylines need to be examined, e.g., how should effective trade-offs be elicited? Can trade-offs be suggested to the user in a semantically meaningful way? Which kinds of interfaces are suitable, which are not? How strongly are skylines focused by considering trade-offs, how do they affect the manageability of the result sets and thus helping the user to satisfy his/her information requirements? On the other hand, also technical performance improvements of the algorithm remain an issue for future exploration. These will cover further tuning of the algorithms, spanning from more efficient generation schemes or compressed storage of the sequences, to more efficient indexing allowing for faster object domination tests.

6. ACKNOWLEDGMENTS

Part of this work was supported by a grant of the German Research Foundation (DFG) within the Emmy Noether Program of Excellence.

7. REFERENCES

[1] Balke, W.-T., Lofi, C., Güntzer, U. Incremental Trade-Off Management for Preference Based Queries, *Int. Jour. of Computer Science & Applications (IJCSA)*, 4 (Jun. 2007).

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

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

[4] Chan, C.-Y., Jagadish, H.V., Tan, K.-L., Tung, A. K. H., Zhang, Z. Finding k-dominant skylines in high dimensional space, *Int. Conf. on Management of Data (SIGMOD)*, Chicago, IL, USA, 2006.

[5] Chan, C.-Y., Jagadish, H.V., Tan, K.-L., Tung, A. K. H., Zhang, Z. On High Dimensional Skylines, *Advances in Database Technology (EDBT)*, Munich, Germany, 2006.

[6] Chomicki, J. Iterative Modification and Incremental Evaluation of Preference Queries. *Int. Symp. on Found. of Inf. and Knowledge Systems (FoIKS)*, Budapest, Hungary, 2006.

[7] Godfrey, P. Skyline cardinality for relational processing. How many vectors are maximal? *Int. Symp. on Foundations of Information and Knowledge Systems (FoIKS)*. Vienna, Austria, 2004.

[8] Godfrey, P.; Shipley, R.; Gryz. J. Maximal Vector Computation in Large Data. *Int. Conf. on Very Large Databases (VLDB)*, Trondheim, Norway, 2005.

[9] Hansson, S.O. Preference Logic, *Handbook of Philosophical Logic*, Second Edition, 4 (2002)

[10] Ihab F. Ilyas and George Beskales and Mohamed A. Soliman. A Survey of Top-K Query Processing Techniques in Relational Database Systems, ACM Computing Surveys, 40-4 (2008)

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

[12] Lacroix, M., Lavency., P., Preferences: Putting more Knowledge into Queries. *In.t Conf. Very Large Databases (VLDB)*, Brighton, UK, 1987.

[13] Lee, J., You, G., Hwang, S. Telescope: Zooming to Interesting Skylines. *Int. Conf. on Database Systems for Advanced Applications (DASFAA)*, Bangkok, Thailand, 2007.

[14] Lofi, C., Balke, W.-T., Güntzer, U. Efficiently Performing Consistency Checks for Multi-Dimensional Preference Trade-Offs, *Int. Conf. on Research Challenges in Computer Science (RCIS)*, Marrakech, Morocco, 2008.

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

[16] Pei, J., Jin, W., Ester, M., Tao, Y. Catching the best views of skyline: a semantic approach based on decisive subspaces. *Int. Conf. on Very Large Data Bases*, Trondheim, Norway, 2005

[17] Yuan, Y., Lin, X., Liu, Q., Wang, W.,Yu, J.Y., Zhang, Q., Efficient computation of the skyline cube, *Int. Conf. on Very Large Data Bases*, Trondheim, Norway, 2005.