

# Efficient Skyline Querying with Variable User Preferences on Nominal Attributes

Raymond Chi-Wing Wong<sup>1</sup>, Ada Wai-Chee Fu<sup>2</sup>, Jian Pei<sup>3</sup>,  
Yip Sing Ho<sup>2</sup>, Tai Wong<sup>2</sup>, Yubao Liu<sup>4</sup>

<sup>1</sup> Computer Science and Engineering  
The Hong Kong University of Science and Technology  
raywong@cse.ust.hk

<sup>3</sup> School of Computer Science  
Simon Fraser University  
jpei@cs.sfu.ca

<sup>2</sup> Computer Science and Engineering  
The Chinese University of Hong Kong  
adafu@cse.cuhk.edu.hk

<sup>4</sup> Department of Computer Science  
Sun Yat-Sen University  
liuyubao@mail.sysu.edu.cn

## ABSTRACT

Current skyline evaluation techniques assume a fixed ordering on the attributes. However, dynamic preferences on nominal attributes are more realistic in known applications. In order to generate online response for any such preference issued by a user, one obvious solution is to enumerate all possible preferences and materialize all results of these preferences. However, the pre-processing and storage requirements of a full materialization are typically prohibitive. Instead, we propose a semi-materialization method called the IPO-tree Search which stores partial useful results only. With these partial results, the result of each possible preference can be returned efficiently. We have also conducted experiments to show the efficiency of our proposed algorithm.

## 1. INTRODUCTION

The skyline operator has emerged as an important summarization technique for multi-dimensional datasets. Given a set of  $m$ -dimensional data points, the *skyline*  $S$  is the set of all points  $p$  such that there is no other point  $q$  which *dominates*  $p$ .  $q$  is said to dominate  $p$  if  $q$  is *better* than  $p$  in at least one dimension and not *worse* than  $p$  in all other dimensions. Skyline queries have been studied since 1960s in the theory field where skyline points are known as *Pareto sets* and *admissible points* [13] or *maximal vectors* [12]. The problem of skyline queries is introduced in the database context in [1], and earlier algorithms such as [12, 11] are found to be unscalable for large databases.

Most of the existing studies consider numeric attributes or dimensions. In this paper, we use the terms “*attribute*” and “*dimension*” interchangeably. Some representative methods for finding skylines include the block nested loop (BNL) algorithm [1].

Permission to make digital or hard copies of portions of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyright for components of this work owned by others than VLDB Endowment must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers or to redistribute to lists requires prior specific permission and/or a fee. Request permission to republish from: Publications Dept., ACM, Inc. Fax +1 (212) 869-0481 or permissions@acm.org.

algorithm [1], the sort first skyline (SFS) algorithm [7], the bitmap method [24], the nearest neighbor (NN) algorithm in [9] and the branch and bound skylines (BBS) method [19, 20]. It is easy to see that a naive method requires  $n^2$  pairwise comparisons of the data, where  $n$  is the number of data points. Most of the known algorithms without indexing are shown to have a worst-case complexity of  $O(kn^2)$ , where  $k$  is the number of dimensions, and an average-case complexity at least linear in  $n$  [16]. It is shown in [10] that the skyline problem requires at least  $\lceil \log n! \rceil$  comparisons. Recently, an efficient algorithm over datasets with low-cardinality domains was proposed by [18]. Besides, sub-space skyline querying are also studied [27, 22, 26, 23, 21].

Suppose we look for a vacation package as shown in Table 1. We know that attribute Price and attribute Hotel-class are numeric attributes where a cheaper package is more preferable and a package with higher hotel-class is more preferable. However, attribute Hotel-group as shown in Table 1 is a categorical attribute. There can be partial ordering on categorical attributes. [2, 3, 4, 6, 5, 15, 14, 25] consider partially-ordered categorical attributes. In [2, 3], each partially-ordered attribute is transformed into two-integer attributes so that conventional skyline algorithms can be applied. [4] studies the cost estimation of the skyline operator involving the partially ordered attributes.

Known existing works on categorical attributes assume that *each attribute has only one order: either a total or a partial order*. In real life, it is not often that categorical attributes have a fixed predefined order. For example, different customers may prefer different realty locations, different car models, or different airlines. We call such a categorical attribute which does not come with a predefined order a *nominal attribute*. It is easy to name important applications with nominal attributes, such as realties (where type of realty, regions and style are examples of nominal attributes) and flight booking (where airline and transition airport are examples of nominal attributes). In this paper, we consider the scenarios where different users may have different preferences on nominal attributes. That is, more than one order need to be considered in nominal attributes.

Furthermore, typically, for a nominal attribute, there may be many different values, and a user would not specify an or-

Package	Price	Hotel-class	Hotel-group
<i>a</i>	1600	4	T (Tulips)
<i>b</i>	2400	1	T (Tulips)
<i>c</i>	3000	5	H (Horizon)
<i>d</i>	3600	4	H (Horizon)
<i>e</i>	2400	2	M (Mozilla)
<i>f</i>	3000	3	M (Mozilla)

**Table 1: Vacation packages**

der on all the values, but would only list a few of the most favorite choices. Table 2 shows different customer preferences on Hotel-group. The preference of Alice is “ $T \prec M \prec *$ ” which means that she prefers Tulips to Mozilla and prefers these two to other hotel groups (i.e., Horizon). We call such preferences *implicit preferences*. Note that different preferences yield different skylines. As shown in Table 2, the skyline is  $\{a, c\}$  for Alice’s preference but  $\{a, c, e, f\}$  for Fred’s preference. The numerous skylines make the problem highly challenging. Note that *e* and *f* are in the skyline with respect to Bob’s preference because both *e* and *f* are not dominated by any other packages.

[6, 5] study the problem of preference changes, whereupon the query results can be incrementally refined. In [15], a user or a customer can specify some values in nominal attributes as an equivalence class to denote the same “importance” for those values. [14] is an extension of [15]. In [14], whenever a user finds that there are a lot of irrelevant results for a query, s/he can modify the query by adding more conditions so that the result set is smaller to suit her/his need. These works focus either on the effects of the query changes on the result size, or the reuse of skyline results when a query is refined in a progressive manner, and not on efficient query evaluation. Here, we consider that different users may have different preferences and so the preferences are not undergoing refinement but can be different or conflicting from one query to another. Also, we focus on the issue of efficient query answering.

Nominal attributes are first considered in [25] but the study is about finding a set of partial orders with respect to which a given point is in the skyline. The problem studied in [25] is a reverse problem of this paper. Here, we study how to find a set of skyline points given an implicit preference. In [20], dynamic skyline is considered which is only for numeric data (implying a natural ordering in each dimension), and the “dynamic function” considered is based on distance from a user location. Here, we consider nominal attributes, and the “dynamic function” is any mapping between the nominal values and the *rankings* where each nominal value is assigned with a ranking value. Hence the BBS method does not work in our case.

To support online skyline queries with nominal attributes, a straightforward solution is to materialize the skylines of all possible implicit preferences. Let  $m'$  be the number of nominal attributes and  $c$  be the maximum cardinality of a nominal attribute. The number of all possible implicit preferences is given by

$$\sum_{i=0}^{c-1} P_i(c)^{m'} = O((c \cdot c!)^{m'})$$

Customer	Preference	Skyline
Alice	$T \prec M \prec *$	$\{a, c\}$
Bob	No special preference	$\{a, c, e, f\}$
Chris	$H \prec M \prec *$	$\{a, c, e\}$
David	$H \prec M \prec T$	$\{a, c, e\}$
Emily	$H \prec T \prec *$	$\{a, c\}$
Fred	$M \prec *$	$\{a, c, e, f\}$

**Table 2: Customer preferences**

where  $P_i(c)$  is the number of permutations of ordering  $i$  elements from  $c$  elements. In Table 1, suppose there are three nominal attributes each of which has 40 possible values. The total number of implicit preferences is equal to  $4.1 \times 10^9$ . With such a large number of possibilities, it is typically infeasible to materialize all results with all preferences.

Another method is to try to adopt an existing algorithm on skyline querying. We show how the SFS algorithm [7] can be extended to an *adaptive SFS* algorithm which computes the results given an implicit preference *on-the-fly*. However, the performance is still not scalable.

To adequately solve this problem, we propose a semi-materialization method which stores some useful partial results corresponding to certain implicit preferences. Given any implicit preference, we can derive the answer based on the stored information. In order to make the derivation efficiently, we propose a tree structure called *IPO-Tree* (implicit preference order tree) to store the partial results. We will show that the total number of nodes stored in the IPO-tree is

$$\sum_{i=0}^{m'} (c+1)^i = O(c^{m'}).$$

In the above example, it means 70,644 nodes, which is significantly smaller than  $4.1 \times 10^9$ . With the IPO-Tree, query evaluation requires  $O(x^{m'})$  set operations where  $m'$  is the number of nominal dimensions on which some preferences are specified by the user,  $x$  is the maximum number of values a user specifies for a dimension. Note that both  $x$  and  $m'$  are often small numbers [9, 20]. For example, if the user specifies two preferences for each of the three nominal attributes that s/he is interested in,  $x^{m'}$  becomes  $2^3 = 8$  only.

The rest of the paper is organized as follows. We formulate the problem in Section 2. In Section 3, we extend the SFS algorithm to handle any implicit preference on-the-fly. In Section 4, a semi-materialization approach is proposed. An empirical study is reported in Section 5, and the paper is concluded in Section 6.

## 2. PROBLEM DEFINITION

A skyline analysis involves multiple attributes. A user’s preference on the values in an attribute can be modeled by a partial order on the attribute. A *partial order*  $\preceq$  is a reflexive, asymmetric and transitive relation. A partial order is also a total order if, for any two values  $u$  and  $v$  in the domain, either  $u \preceq v$  or  $v \preceq u$ . We write  $u \prec v$  if  $u \preceq v$  and  $u \neq v$ . A partial order also can be written as  $R = \{(u, v) | u \preceq v\}$ .  $u \preceq v$  also can be written as  $(u, v) \in R$ . We call this model the *partial order model*.

By default, we consider points in an  $m$ -dimensional space  $\mathbb{S} = D_1 \times \dots \times D_m$ . For each dimension  $D_i$ , we assume that there is a partial or total order  $R_i$  on the values in  $D_i$ . For a point  $p$ ,  $p.D_i$  is the projection on dimension  $D_i$ . If  $(p.D_i, q.D_i) \in R_i$ , we also write  $p.D_i \preceq q.D_i$ .

For points  $p$  and  $q$ ,  $p$  dominates  $q$ , denoted by  $p \prec q$ , if, for any dimension  $D_i \in \mathbb{S}$ ,  $p \preceq_{D_i} q$ , and there exists a dimension  $D_{i_0} \in \mathbb{S}$  such that  $p \prec_{D_{i_0}} q$ . If  $p$  dominates  $q$ , then  $p$  is more preferable than  $q$  according to the preference orders. The dominance relation  $R$  can be viewed as the integration of the preference partial orders on all dimensions. Thus, we can write  $R = (R_1, \dots, R_m)$ . It is easy to see that the dominance relation is a strict partial order.

Given a data set  $\mathcal{D}$  containing data points in space  $\mathbb{S}$ , a point  $p \in \mathcal{D}$  is in the skyline of  $\mathcal{D}$  (i.e., a skyline point in  $\mathcal{D}$ ) if  $p$  is not dominated by any points in  $\mathcal{D}$ . Given a preference  $R$ , the skyline of  $\mathcal{D}$ , denoted by  $SKY(R)$ , is the set of skyline points in  $\mathcal{D}$ .

In many applications, there often exist some orders on some of the dimensions that hold for all users. In our example in Table 1, a lower price and a higher hotel-class are always more preferred by customers. Even for nominal attributes, there may exist some universal partial orders. For example, in a realty market, detached houses are often more preferred than semi-detached houses. Hence, we assume that we are given a template, which contains a partial order for every dimension. The partial orders in the template are applicable to all users. Each user can then express his/her specific preference by refining the template. The containment relation of orders captures the refinement.

For partial orders  $R$  and  $R'$ ,  $R'$  is a refinement of  $R$ , denoted by  $R \subseteq R'$ , if for any  $(u, v) \in R$ ,  $(u, v) \in R'$ . Moreover, if  $R \subseteq R'$  and  $R \neq R'$ ,  $R'$  is said to be stronger than  $R$ . Let  $R = \{(T, M)\}$  and  $R' = \{(T, M), (H, M)\}$ . Then,  $R \subseteq R'$ . That is,  $R'$  is a refinement of  $R$  by adding a preference  $H \prec M$ . As  $R \neq R'$ ,  $R'$  is stronger than  $R$ .

**PROPERTY 1.** For orders  $R = (R_1, \dots, R_m)$  and  $R' = (R'_1, \dots, R'_m)$ ,  $R \subseteq R'$  if and only if  $R_i \subseteq R'_i$  for  $1 \leq i \leq m$ . ■

**THEOREM 1 (MONOTONICITY).** ([25]) Given a data set  $\mathcal{D}$  and a template  $R$ , if  $p$  is not in the skyline with respect to  $R$ , then  $p$  is not in the skyline with respect to any refinement  $R'$  of  $R$ . ■

Theorem 1 indicates that, when the orders on the dimensions are strengthened, some skyline points may be disqualified. However, a non-skyline point never gains the skyline membership due to a stronger order. This monotonic property greatly helps in analyzing skylines with respect to various orders. For instance, suppose  $R = \emptyset$  and  $R' = \{(T, M)\}$ . Since  $b$  is not a skyline point with respect to  $R$  (because  $a$  dominates  $b$ ),  $b$  is also not a skyline point with respect to a refinement  $R'$  of  $R$ .

**DEFINITION 1 (CONFLICT-FREE).** ([25]) Let  $R$  and  $R'$  be two partial orders.  $R$  and  $R'$  are conflict-free if there exist no values  $u$  and  $v$  such that  $u \neq v$ ,  $(u, v) \in R$ , and  $(v, u) \in R'$ .

In a skyline query, for a nominal attribute, users typically would not explicitly order all values, but may specify a few of their favorite choices and also give them an ordering. For

example, a user may specify that the first choice is  $v$  and the second choice is  $v'$ . The implicit meaning is that  $v$  and  $v'$  are better than all the other choices, say  $v_1, v_2, \dots, v_k$ . This can be described by the partial order model, by including  $v \prec v'$ ,  $v \prec v_1, v \prec v_2, \dots, v \prec v_k$  and  $v' \prec v_1, v' \prec v_2, \dots, v' \prec v_k$ . This preference is denoted by

$$"v \prec v' \prec *"$$

where  $*$  means all choices other than  $v$  and  $v'$  (in this case,  $*$  corresponds to  $\{v_1, v_2, \dots, v_k\}$ ). We call this special kind of partial order an *implicit preference* and assume that it is represented in such a form. For example, the implicit preference " $H \prec M \prec *$ " corresponds to a set of binary orders  $\{(H, M), (H, T), (M, T)\}$  in the partial order model.

**DEFINITION 2 (IMPLICIT PREFERENCES).** Let  $v_1, v_2, \dots, v_k$  be all values in a nominal attribute  $D_i$ . An implicit preference  $\tilde{R}_i$  on  $D_i$  is given by

$$v_1 \prec v_2 \prec \dots \prec v_x \prec *.$$

where  $x \leq k$ . It is equivalent to the partial order given by

$$\{(v_i, v_j) | i < j \wedge i \in [1, x] \wedge j \in [1, k]\}.$$

In the above definition,  $\tilde{R}_i$  is said to be an  $x$ -th order implicit preference. Also, the order of  $\tilde{R}_i$ , denoted by  $order(\tilde{R}_i)$ , is defined to be  $x$  and the order of  $\tilde{R}$  is defined to be  $\max_i \{order(\tilde{R}_i)\}$ . A value  $v_j$  is said to be in  $\tilde{R}_i$  if  $v_j \in \{v_1, v_2, \dots, v_x\}$ . Also,  $v_j$  is said to be the  $j$ -th entry in  $\tilde{R}_i$ .  $\mathcal{P}(\tilde{R}_i)$  is defined to be  $\{(v_i, v_j) | i < j \text{ and } i \in [1, x] \text{ and } j \in [1, k]\}$ . Let  $\tilde{R} = (\tilde{R}_1, \tilde{R}_2, \dots, \tilde{R}_m)$ .  $\mathcal{P}(\tilde{R})$  is defined to be  $\bigcup_{i=1}^m \mathcal{P}(\tilde{R}_i)$ .

In this paper, we adopt the convention that  $\tilde{R}$  denotes an implicit preference and  $R$  denotes a partial order (which may or may not be an implicit preference). Also we denote  $SKY(\mathcal{P}(\tilde{R}))$  by  $SKY(\tilde{R})$ .

**DEFINITION 3 (PROBLEM).** Given a dataset  $\mathcal{D}$  and an implicit preference  $\tilde{R}$ , find the skyline  $SKY(\tilde{R})$  in  $\mathcal{D}$ . ■

We also say that we want to find a set of skyline points with respect to  $\tilde{R}$  in  $\mathcal{D}$ . In many applications, online response is required. We aim to find an efficient solution.

The problem of dynamic implicit preferences have some similar flavor as subspace skylines since materialization of the possible skylines seems to be a solution. However, as noted in [20], most applications involve up to five attributes, the dimensionality  $m'$  of a typical skyline problem is not high, and the number of subspaces which is equal to

$$\sum_{i=1}^{m'} C_i(m') = 2^{m'} - 1$$

(where  $C_i(m')$  is the number of ways of choosing  $i$  elements from  $m'$  elements) is limited, materialization of the skylines is quite feasible and has been investigated in recent works such as [27, 26, 23, 21]. For dynamic implicit preferences, the number of combinations is given by

$$\left( \sum_{i=0}^{c-1} P_i(c) \right)^{m'} = O((c \cdot c!)^{m'})$$

(where  $P_i(c)$  is the number of permutations of ordering  $i$  elements from  $c$  elements), which is exponential not only

Package	Price	Hotel-class	Hotel group	Airline
<i>a</i>	1600	4	T (Tulips)	G (Gonna)
<i>b</i>	2400	1	T (Tulips)	G (Gonna)
<i>c</i>	3000	5	H (Horizon)	G (Gonna)
<i>d</i>	3600	4	H (Horizon)	R (Redish)
<i>e</i>	2400	2	M (Mozilla)	R (Redish)
<i>f</i>	3000	3	M (Mozilla)	W (Wings)

**Table 3: A table with two nominal attributes.**

in the dimensionality but also in the cardinalities  $c$  of the attributes. Materialization of all subspace skylines becomes infeasible in most cases.

### 3. ADAPTIVE SFS

First we may consider to extend an existing algorithm which handles the skyline query with respect to fixed preference orders to handle any implicit preference *on-the-fly*. We choose the SFS algorithm [7] for such an extension because adapting other methods such as BBS, bitmap and NN methods is costly, and they need index rebuilt for different implicit preferences. The resulting algorithm is called Adaptive SFS and does not require a complete reprocessing of the data for each different user preference.

#### 3.1 Overview of SFS

First, we will briefly describe the method of Sort-First Skyline (SFS), which is for totally-ordered numerical attributes. With SFS, the data points are sorted according to their scores obtained by a preference function  $f$ , which can be the sum of all the numeric values in different dimensions of a data point. That is, the score of a point  $p$  is

$$f(p) = \sum_{i=1}^m p.D_i.$$

The criterion for the function is that if  $p \prec q$ , then  $f(p) < f(q)$ . The data points are then examined in ascending order of their scores. A *skyline list*  $L$  is initially empty. If a point is not dominated by any point in  $L$ , then it is inserted into  $L$ . The sorting takes  $O(n \log n)$  time while the scanning of the sorted list to generate the skyline points takes  $O(N \cdot n)$  time, where  $n$  is the number of data points in the data set and  $N$  is the size of the skyline.

#### 3.2 Adaptive SFS for Implicit Preferences

Next, we develop an adaptive SFS method for query processing in the data set with implicit preferences on nominal attributes, given the skyline set  $SKY(\tilde{R})$  for a template order  $\tilde{R}$  which is implicit. Let  $\tilde{R}'$  be an implicit refinement over  $\tilde{R}$ . From Theorem 1, any skyline point  $p$  for  $\tilde{R}'$  will also be a skyline point for  $\tilde{R}$ . Hence, in order to look for the skyline for  $\tilde{R}'$ , we only need to search  $SKY(\tilde{R})$ .

The idea is the following. We adopt the basic presorting step on  $SKY(\tilde{R})$  resulting in a sorted list  $L(\tilde{R})$ . When a query with a refinement  $\tilde{R}'$  arrives, we first try to re-sort the list  $L(\tilde{R})$  and obtain a new sorted list  $L(\tilde{R}')$ . The skyline generation step is then applied on  $L(\tilde{R}')$ . The key to the efficiency is that the resorting step complexity is  $O(l \log N)$ , where  $l$  is the number of data points affected by the refine-

ment  $\tilde{R}'$  and is typically much smaller than  $N$ . Next, we give more detailed description of the algorithm.

Each value  $v$  in a dimension  $D_i$  is associated with a rank denoted by  $r(v)$ . In a totally-ordered attribute  $D_i$ , we define

$$r(v) = v$$

for each  $v$  in  $D_i$ . Without loss of generality, we assume that a smaller value in a dimension  $D_i$  is more preferable than a larger value in the same dimension. For each value  $v$  in a nominal attribute  $D_i$ , we assign  $r(v)$  as follows. Let  $c_i$  be the cardinality of nominal dimension  $D_i$ . By default, for each value  $v$  for dimension  $D_i$ ,  $r(v) = c_i$ . For example, if there are 10 different values in dimension  $D_i$ , then by default  $r(v) = 10$  for each  $v$  in  $D_i$ . Given an implicit partial order  $\tilde{R}'_i$ , we can determine a ranking for the values that appear in  $\tilde{R}'_i$  so that  $r(v) < r(v')$  if and only if  $v \prec v'$  can be derived from  $\tilde{R}'_i$ . If  $\tilde{R}'_i$  is " $v_1 \prec v_2 \prec \dots \prec v_x \prec *$ ", then we set  $r(v_1) = 1, r(v_2) = 2, \dots, r(v_x) = x$ . We define

$$f(p) = \sum_{i=1}^m r(p.D_i).$$

Let  $l$  be the number of data points that contain some values in  $\tilde{R}'$ . The processing time of the sorting list is  $O(l \log N)$ . Algorithms 1 and 2 show the steps for preprocessing the data points and query processing, respectively.

---

#### Algorithm 1 Preprocessing

---

- 1: Compute the skyline set  $SKY(\tilde{R})$  for the given template  $\tilde{R}$
  - 2: Determine the ranking  $r$  based on  $SKY(\tilde{R})$  and  $f$
  - 3: Apply the presorting step of SFS based on  $r$  on  $SKY(\tilde{R})$
- 

---

#### Algorithm 2 Query Processing

---

**Input:** skyline query, with implicit preference  $\tilde{R}'$

- 1: Determine the ranking for the values in  $\tilde{R}'$
  - 2: Find the data points in  $SKY(\tilde{R})$  that contain values in  $\tilde{R}'$ . Alter the rankings for such data points if necessary
  - 3: Delete the points with altered rankings from the sorted list
  - 4: Re-insert the points just deleted using the new ranking
  - 5: Apply the skyline extraction step of SFS on the resulting sorted list
- 

**EXAMPLE 1 (ADAPTIVE SFS).** Consider Table 1. To illustrate, we transform Table 1 to Table 4 by subtracting the hotel-class value from 5 such that for each numeric attribute, a smaller value is more preferable. Suppose  $\tilde{R} = \emptyset$  and the cardinality of attribute Hotel-group is 3.

For pre-processing, firstly, we compute the skyline set  $SKY(\tilde{R}) = \{a, c, e, f\}$ . This step can be completed by some existing skyline algorithms. Then, for each package  $\in SKY(\tilde{R})$ , we compute its score as shown in Table 5. For example, attribute Price and attribute Reverse Hotel-class of package  $a$  are 1600 and 1, respectively. Consider attribute Hotel-group. Since  $\tilde{R} = \emptyset$ ,  $r(T) = r(H) = r(M) = 3$ . Thus, the score of package  $a$  is equal to  $1600 + 1 + 3 = 1604$ . After that, we sort them in ascending order of the score values. The ordering becomes  $\langle a, e, c, f \rangle$ .

Package	Price	Reverse Hotel-class	Hotel-group
<i>a</i>	1600	1	T (Tulips)
<i>b</i>	2400	4	T (Tulips)
<i>c</i>	3000	0	H (Horizon)
<i>d</i>	3600	1	H (Horizon)
<i>e</i>	2400	3	M (Mozilla)
<i>f</i>	3000	2	M (Mozilla)

Table 4: Vacation packages

After we pre-process the data, when there is a skyline query, we will perform the following steps. Suppose the query is “ $H \prec T \prec *$ ”. Then, we update the scores of all packages with Hotel-group equal to  $H$  or  $T$  where  $r(H) = 1$  and  $r(T) = 2$ . In other words, we just update packages  $a$  and  $c$  as shown in Table 6. We remove  $a$  and  $c$  from  $\langle a, e, c, f \rangle$  and re-insert them according to their updated score values. We obtain the same ordering  $\langle a, e, c, f \rangle$ . Now, we apply the skyline extraction step of SFS on this ordering. Initially, a variable  $L$  is set to  $\emptyset$ . We process packages according to the ordering  $\langle a, e, c, f \rangle$ . Since  $L = \emptyset$ , we can insert  $a$  into  $L$  directly. The next processed package is  $e$ . We check whether  $e$  is dominated by any point in  $L$ . Since  $e$  is dominated by  $a$ , we proceed to process the next package  $c$ . We find that no points in  $L$  dominate  $c$  and thus insert  $c$  into  $L$ . Finally, we process  $f$  which is found to be dominated by  $a$ . Thus, the skyline for this query is  $L = \{a, c\}$ . ■

In Step 2 in Algorithm 2, in order to find data points in  $SKY(\tilde{R})$  that contain values in  $\tilde{R}'$ , one possible way is to have an index for each nominal dimension. The index can be a simple sorted list or a more sophisticated tree index. An index lookup can quickly return the points that contain a particular value in  $\tilde{R}'$ . Such data points are collected in a set. Then, for each point  $p$  in the set, the value of  $f(p)$  based on  $\tilde{R}$  allows us to quickly locate the point in the sorted list. The point is deleted from the list and re-inserted with a new value for  $f(p)$  based on the refinement  $\tilde{R}'$ .

For the last step of the query processing, there is no need to follow the SFS from scratch. Instead, we reinsert the points in the ascending order of the new  $f(p)$  values. When a point  $p$  is re-inserted, we need only check if it may be dominated by the  $\tilde{R}'$  skyline points sorted before it. If so,  $p$  is not added; otherwise, we then check if it may dominate any  $SKY(\tilde{R})$  skyline point that are sorted after it. The points that it dominates will be removed. Let  $N' = |SKY(\tilde{R}')|$ ,  $N = |SKY(\tilde{R})|$ , and  $l$  be the number of points in  $SKY(\tilde{R})$  containing values in  $\tilde{R}'$ . The time complexity of this step will become  $O(l \log l + N' \cdot l + \min(N', l) \cdot N)$ . Since the resorting step takes  $O(l \log N)$  time, the total time is  $O(l \log N + \min(N', l) \cdot N)$ .

LEMMA 1. *In the adaptive SFS algorithm, given the sorted list  $L(\tilde{R})$  from the preprocessing step, and a skyline query with an implicit preference  $\tilde{R}'$ , the query processing time complexity is  $O(l \log N + \min(N', l) \cdot N)$ .*

Though the query processing time is improved from the original SFS complexity,  $O(n \log n + N \cdot n)$ , to  $O(l \log N + \min(N', l) \cdot N)$  it may still be unscalable when the skyline sets are large.

Package	Score
<i>a</i>	1604
<i>c</i>	3003
<i>e</i>	2406
<i>f</i>	3005

Table 5: Score of each package during pre-processing

Package	Score
<i>a</i>	1603
<i>c</i>	3001
<i>e</i>	2406
<i>f</i>	3005

Table 6: Score of each package when the query is “ $H \prec T \prec *$ ”

## 4. PARTIAL MATERIALIZATION: IPO-TREE SEARCH

In order to support online response, a naive approach is to materialize the skylines for all possible preferences. However, as noted in Section 2, this approach is very costly in storage and preprocessing. Our idea is therefore to materialize some useful partial results so that these partial results can be combined efficiently to form the query results. In particular, we propose to materialize the results with respect to the first-order implicit preference on each nominal attribute only. Since results for the second or higher order preferences are not stored, the number of combinations is significantly reduced. It is not obvious that this is feasible. For the skyline problem with fixed orders for all dimensions, it is well-known that the skyline of a set  $D$  of dimensions cannot be computed from the skylines of the subsets of  $D$ , say  $X$  and  $Y$ . This is because the union of the skylines of dimension subsets  $X$  and  $Y$  may not contain the skyline of  $X \cup Y$ . In the following, we describe an important property called the *merging property* which allows us to derive results of all possible implicit preferences of *any* order by simple operations on top of the *first-order* information maintained.

THEOREM 2 (MERGING PROPERTY). *Let two implicit preferences  $\tilde{R}'$  and  $\tilde{R}''$  differ only at the  $i$ -th dimension, i.e.,  $\tilde{R}'_j = \tilde{R}''_j$  for all  $j \neq i$ . Furthermore,  $\tilde{R}'_i = “v_1 \prec \dots \prec v_{x-1} \prec *”$  and  $\tilde{R}''_i = “v_x \prec *”$ . Let  $PSKY(\tilde{R}')$  be the set of points in  $SKY(\tilde{R}')$  with  $D_i$  values in  $\{v_1, \dots, v_{x-1}\}$ . Let  $\tilde{R}'''$  be an implicit preference which differs from  $\tilde{R}'$  and  $\tilde{R}''$  only at the  $i$ -th dimension where  $\tilde{R}'''_i = “v_1 \prec \dots \prec v_{x-1} \prec v_x \prec *”$ . The skyline with respect to  $\tilde{R}'''$  is*

$$(SKY(\tilde{R}') \cap SKY(\tilde{R}'')) \cup PSKY(\tilde{R}').$$

**Proof:** We need to show that a point  $p$  is in  $SKY(\tilde{R}''')$  if and only if it is in  $(SKY(\tilde{R}') \cap SKY(\tilde{R}'')) \cup PSKY(\tilde{R}')$ . For each direction, we prove by contradiction.

[A] Firstly, assume  $p$  is in  $SKY(\tilde{R}''')$ , and suppose that  $p$  is not in  $(SKY(\tilde{R}') \cap SKY(\tilde{R}'')) \cup PSKY(\tilde{R}')$ . Then, by Theorem 1, since  $p \in SKY(\tilde{R}''')$  and  $\tilde{R}'''$  is a refinement of  $\tilde{R}'$ , we deduce that

$$p \in SKY(\tilde{R}').$$

Thus,  $p$  must satisfy the following:

- Condition 1:  $p.D_i \notin \{v_1, \dots, v_{x-1}\}$  and
- Condition 2:  $p \notin SKY(\tilde{R}'')$ .

Consider Condition 2. Since  $p \notin SKY(\tilde{R}'')$ , there exists a data point  $q$  dominating  $p$  w.r.t  $\tilde{R}''$ . In other words, with respect to  $\tilde{R}''$ ,  $q.D_k \preceq p.D_k$  for all  $k$  and in at least one dimension  $D_j$ ,  $q.D_j \prec p.D_j$ . Let  $\mathcal{J}$  be the set of dimensions  $D_j$  where  $q.D_j \prec p.D_j$  w.r.t  $\tilde{R}''$ . Besides, for all dimensions  $D_k$  other than  $D_i$ , the partial orders of  $\tilde{R}''$  and  $\tilde{R}'''$  are the same. Hence, w.r.t.  $\tilde{R}'''$ ,  $q.D_k \preceq p.D_k$  for all  $k(\neq i)$ . There are two subcases: *Case (i)*:  $D_i \notin \mathcal{J}$  and *Case (ii)*:  $D_i \in \mathcal{J}$ .

*Case (i)*:  $D_i \notin \mathcal{J}$ . For all  $D_j \in \mathcal{J}$ , since  $q.D_j \prec p.D_j$  w.r.t  $\tilde{R}''$  and the partial orders in  $\tilde{R}_j''$  are those in  $\tilde{R}_j'''$ , we have

$$q.D_j \prec p.D_j$$

w.r.t.  $\tilde{R}'''$ . Also, w.r.t.  $\tilde{R}'''$ ,  $q.D_k \preceq p.D_k$  for all  $k \neq i$ . Hence, since  $i \notin \mathcal{J}$ , for dimension  $D_i$ , it must be the case that  $p.D_i \prec q.D_i$  w.r.t  $\tilde{R}'''$ . Otherwise,  $p$  is dominated by  $q$  w.r.t  $\tilde{R}'''$ , and  $p$  cannot be in  $SKY(\tilde{R}''')$ . Since  $p.D_i \prec q.D_i$  w.r.t  $\tilde{R}'''$ , we have

$$p.D_i \neq q.D_i.$$

Since  $q.D_k \preceq p.D_k$  w.r.t.  $\tilde{R}''$  for all  $k$ , and  $p.D_i \neq q.D_i$ , we have  $q.D_i \prec p.D_i$  w.r.t  $\tilde{R}''$ . Since the implicit preference in  $\tilde{R}''$  is " $v_x \prec *$ ", we conclude that  $p.D_i$  cannot be  $v_x$ . Since  $\tilde{R}'''$  is " $v_1 \prec \dots \prec v_x \prec *$ " and  $p.D_i \prec q.D_i$  w.r.t  $\tilde{R}'''$ ,  $p.D_i$  must be in  $\{v_1, \dots, v_{x-1}\}$ . However, this violates Condition 1 discussed above. Hence, we arrive at a contradiction.

*Case (ii)*:  $D_i \in \mathcal{J}$ . We obtain  $q.D_i \prec p.D_i$  w.r.t.  $\tilde{R}''$ . Besides, since the implicit preference in  $\tilde{R}''$  is " $v_x \prec *$ ",  $q.D_i$  must be equal to  $v_x$  and  $p.D_i$  cannot be equal to  $v_x$ . Since  $p \in SKY(\tilde{R}''')$ , there is no other point including  $q$  dominating  $p$  w.r.t.  $\tilde{R}'''$ . Note that, w.r.t.  $\tilde{R}'''$ ,  $q.D_k \preceq p.D_k$  for all  $k(\neq i)$ . We obtain

$$p.D_i \preceq q.D_i$$

w.r.t.  $\tilde{R}'''$ . (Otherwise,  $q.D_i \prec p.D_i$  w.r.t  $\tilde{R}'''$  and  $p$  is dominated by  $q$  w.r.t.  $\tilde{R}'''$ , which leads to a contradiction.) Besides, since  $q.D_i = v_x$ ,  $p.D_i \neq v_x$  and  $\tilde{R}'''$  is " $v_1 \prec \dots \prec v_x \prec *$ ",  $p.D_i$  must be in  $\{v_1, \dots, v_{x-1}\}$ . However, this violates Condition 1. Hence, we arrive at a contradiction.

[B] Conversely, consider a point  $p$  in  $(SKY(\tilde{R}') \cap SKY(\tilde{R}'')) \cup PSKY(\tilde{R}')$ . Suppose that  $p$  is not in  $SKY(\tilde{R}''')$ . Thus,  $p$  is dominated by some point  $q$  w.r.t.  $\tilde{R}'''$ . That is, w.r.t  $\tilde{R}'''$ ,  $q.D_k \preceq p.D_k$  for all  $k$  and  $q.D_j \prec p.D_j$  for at least one dimension  $D_j$ .

Since  $p \in (SKY(\tilde{R}') \cap SKY(\tilde{R}'')) \cup PSKY(\tilde{R}')$ , we know that at least one of the following two conditions holds.

- Condition 3:  $p.D_i \in \{v_1, \dots, v_{x-1}\}$  and  $p \in SKY(\tilde{R}')$ , or
- Condition 4:  $p \in SKY(\tilde{R}')$  and  $p \in SKY(\tilde{R}'')$ .

Consider Condition 3. Since  $p \in SKY(\tilde{R}')$  and  $p \notin SKY(\tilde{R}''')$  where  $\tilde{R}_i'''$  is a refinement of  $\tilde{R}_i'$ , and  $\tilde{R}_k''' = \tilde{R}_k'$  for all  $k \neq i$ , we deduce that  $q.D_i \prec p.D_i$  exists in partial orders of  $\tilde{R}'''$  but not in partial orders of  $\tilde{R}'$ . Since  $q.D_i \prec p.D_i$  w.r.t.  $\tilde{R}'''$ ,  $p.D_i \in \{v_1, \dots, v_{x-1}\}$  and  $\tilde{R}'''$  is

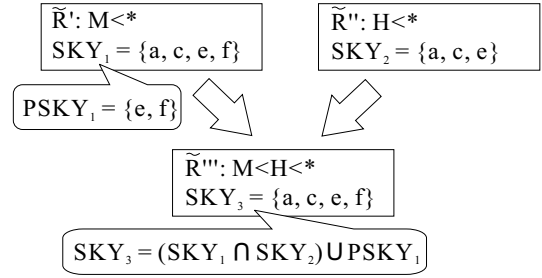


Figure 1: Illustration of the merging property

" $v_1 \prec \dots \prec v_x \prec *$ ", we deduce

$$q.D_i \in \{v_1, \dots, v_{x-2}\}.$$

For each possible binary order  $q.D_i \prec p.D_i$  w.r.t.  $\tilde{R}'''$  where  $p.D_i \in \{v_1, \dots, v_{x-1}\}$  and  $q.D_i \in \{v_1, \dots, v_{x-2}\}$ , we also conclude that  $q.D_i \prec p.D_i$  exists in the partial orders of  $\tilde{R}'$ , which leads to a contradiction.

Consider Condition 4. Since  $\tilde{R}'$ ,  $\tilde{R}''$  and  $\tilde{R}'''$  differ only at dimension  $D_i$ , we only need to check their implicit preferences to see that, whenever  $q.D_i \preceq p.D_i$  (or  $q.D_i \prec p.D_i$ ) w.r.t.  $\tilde{R}'''$ , it is also true w.r.t.  $\tilde{R}'$  or  $\tilde{R}''$ . Therefore,  $q$  also dominates  $p$  w.r.t.  $\tilde{R}'$  or  $\tilde{R}''$ . That is,  $p \notin SKY(\tilde{R}')$  or  $p \notin SKY(\tilde{R}'')$ , which leads to a contradiction. ■

For example, in Figure 1, let  $\tilde{R}'$  be " $M \prec *$ " and  $\tilde{R}''$  be " $H \prec *$ ". From Table 1, the skyline with respect to  $\tilde{R}'$  is  $SKY_1 = \{a, c, e, f\}$  and the skyline with respect to  $\tilde{R}''$  is  $SKY_2 = \{a, c, e\}$ .  $PSKY_1 = \{e, f\}$  is the set of skyline points with values in  $\{M\}$ . Let  $\tilde{R}'''$  be " $M \prec H \prec *$ ". By Theorem 2, the skyline  $SKY_3$  with respect to  $\tilde{R}'''$  is obtained as follows.

$$\begin{aligned} SKY_3 &= (SKY_1 \cap SKY_2) \cup PSKY_1 \\ &= (\{a, c, e, f\} \cap \{a, c, e\}) \cup \{e, f\} \\ &= \{a, c, e\} \cup \{e, f\} \\ &= \{a, c, e, f\} \end{aligned}$$

The derivation can be explained as follows.  $\mathcal{P}(\tilde{R}')$  and  $\mathcal{P}(\tilde{R}'')$  are not conflict-free because their union contains both  $(M, H)$  and  $(H, M)$ . Or, the only difference between  $\mathcal{P}(\tilde{R}') \cup \mathcal{P}(\tilde{R}'')$  and  $\mathcal{P}(\tilde{R}''')$  is that  $\mathcal{P}(\tilde{R}') \cup \mathcal{P}(\tilde{R}'')$  contains one more binary entry, namely  $(H, M)$ , which may disqualify some data points (in this example,  $f$ ). In order to remove the disqualifying effect, we augment the intersection  $SKY_1 \cap SKY_2$  by a union with  $PSKY_1$  where  $PSKY_1$  contains the points disqualified by  $(H, M)$  in  $SKY_1$ .

From Theorem 2, we can derive a powerful tool for the computation of the skyline with respect to any implicit preference of any order by building increasingly higher order refinement ( $\tilde{R}'''$  in the theorem) skyline from lower order ( $\tilde{R}'$  and  $\tilde{R}''$ ) ones, starting with the first-order. In the following two subsections, we introduce the IPO-tree for storing the first-order preference skylines and the query evaluation based on the IPO-tree.

## 4.1 Tree Construction

An *IPO-tree* (*implicit preference order tree*) stores results for combinations of first-order preferences. In this tree, each

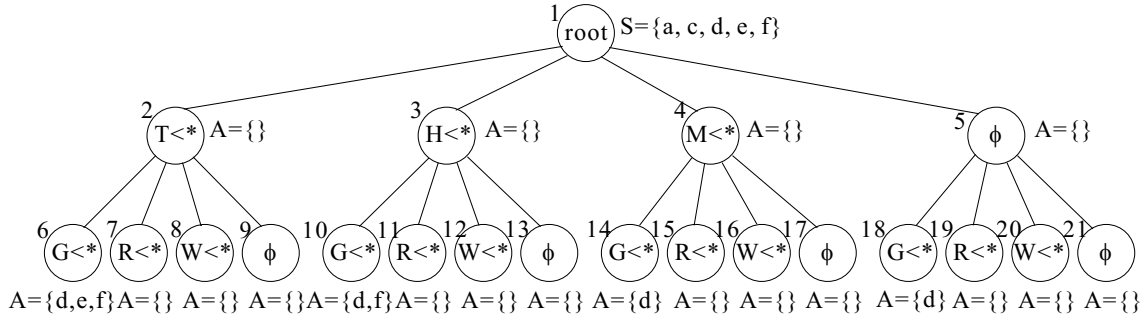


Figure 2: Illustration of an implicit preference order tree (IPO-tree)

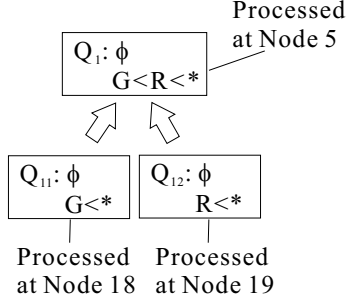


Figure 3: Query evaluation with an IPO-tree where query  $Q = "G < R < *"$

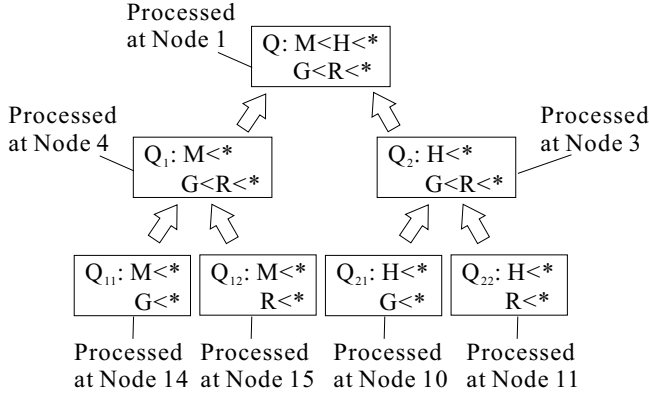


Figure 4: Query evaluation with an IPO-tree where query  $Q = "M < H < *, G < R < *"$

node is labeled with a first-order implicit preference, namely " $v < *$ ", where  $v \in D_i$  and  $D_i$  is a nominal dimension. The tree is of depth  $m' + 1$ , where  $m'$  is the number of nominal attributes. The root node stores the skyline  $SKY(R)$  with respect to template  $R$  in  $\mathcal{D}$ . The second level contains all nodes corresponding to first-order implicit preferences on nominal attribute  $D_1$ . In general, the children of an  $i$ -th level node correspond to all the first-order implicit preferences on nominal attribute  $D_i$ . A special child node is labeled  $\phi$  corresponding to no preference. Each non-root node has a label associated with a first-order implicit preference on a single nominal attribute, and maintains *results* that corresponds to the labels along the path to the root node. Figure 2 shows an IPO-tree from the data in Table 3, where the template  $R$  is set to  $\emptyset$ . Node 6 corresponds to

implicit preferences " $T < *, G < *$ ".

Furthermore, a root node is associated with a set  $S = SKY(R)$ . But, each non-root node is associated with a set  $\mathcal{A}$  of points where  $S - \mathcal{A}$  is the skyline for the corresponding implicit preference. Therefore,  $\mathcal{A}$  contains the points in  $SKY(R)$  that are *disqualified* from the skyline at the node because of the preference refinement. For example, since, in the IPO-tree shown in Figure 2, Node 6 corresponds to an implicit preference " $T < *, G < *$ ", which disqualifies points  $d, e, f$  in  $S$  as skyline points,  $\mathcal{A}$  of node 6 is equal to  $\{d, e, f\}$ . The purpose of  $\mathcal{A}$  is to allow us to find the skyline for the node given the skylines of the ancestors. It is also possible to store the exact skyline at each node instead.

#### 4.1.1 Tree Size

Let  $m'$  be the number of nominal attributes and  $c$  be the maximum cardinality of a nominal attribute. The height of the IPO-tree is  $m' + 1$ . It is easy to verify the following.

LEMMA 2. *The size of the tree in the number of nodes is given by*

$$\sum_{i=0}^{m'} (c+1)^i = O(c^{m'}).$$

As claimed in [9] and quoted in [20], most applications involve up to five attributes, and hence  $m'$  is typically very small. Note that the IPO-tree size is significantly smaller than the number of all possible implicit preferences which is given by

$$\left( \sum_{i=0}^{c-1} P_i(c) \right)^{m'} = O((c \cdot c!)^{m'})$$

where  $P_i(c)$  is the number of permutations of ordering  $i$  elements from  $c$  elements. For example, when  $m' = 3$  and  $c = 40$ , the size of the tree in the number of nodes is 70,644 only but the number of all possible implicit preferences is  $4.1 \times 10^9$ , which is 58,089.64 times larger than the tree size.

The tree size can be further controlled if we know the query pattern (e.g., from a history of user queries). Typically, there are popular and unpopular values. For values which are seldom or never chosen in implicit preferences, the corresponding tree nodes in the IPO-tree are not needed. It is possible to restrict the IPO-tree to say the 10 most popular values for each nominal attribute. For instance, suppose  $m' = 3$  and  $c = 40$ , the tree size is then reduced from 70,644 to 1,464. If a query containing unpopular values arrives, the adaptive SFS algorithm in Section 3 can be used instead.

### 4.1.2 Implementation

In order to find the set  $\mathcal{A}$  for each non-root node  $N$ , one can apply a skyline algorithm (e.g., adaptive SFS in Section 3). However, in our implementation, we make use of the *minimal disqualifying conditions* introduced in [25]. For a skyline point  $p$  and a template order  $R$ , a partial order  $R'$  is called a *minimal disqualifying condition* (or MDC for short) if

1.  $R' \cap R = \emptyset$ ,
2.  $R'$  and  $R$  are conflict-free,
3.  $p$  is not a skyline point with respect to  $R \cup R'$ , and
4. there exists no  $R''$  such that  $R'' \subset R'$  and  $p$  is not a skyline point with respect to  $R \cup R''$ .

The set of minimal disqualifying conditions for  $p$  is denoted by  $MDC(p)$ . The first step here is to find all MDCs of each skyline point in  $SKY(R)$ . One of the algorithms in [25] can be used for this step. Then, given the implicit preference  $\tilde{R}'$  corresponding to a node  $N$ , we check each point in  $SKY(R)$ , if any of the MDCs is a subset of  $\mathcal{P}(\tilde{R}')$ , then the point is disqualified and is inserted into  $\mathcal{A}$ .

## 4.2 Query Evaluation

IPO-tree has a simple structure and a well-controlled tree size. It can efficiently facilitate implicit preference querying based on the merging property (Theorem 2). Algorithm 3 shows the evaluation of a query with an implicit preference  $\tilde{R}'$ .

It is noted that the IPO-tree is built according to a certain (arbitrary) ordering of the nominal attributes. For example, the IPO-tree in Figure 2 is built according to attribute Hotel-group first and then attribute Airline. Thus, the children of the first level node correspond to attribute Hotel-group and the children of the second level node correspond to attribute Airline. It is trivial that the IPO-tree returns the skylines efficiently when the user issues the implicit preference only on the first attribute, Hotel-group. However, similar efficiency is guaranteed when the user issues the implicit preference only on the second attribute, Airline. At each node, there is a special child labeled  $\phi$  representing no preference on a certain attribute. For the query described above, there is no need to traverse all nodes corresponding to attribute Hotel-group. Instead, the node labeled with  $\phi$  (i.e., node 5) is traversed to reach the descendant nodes corresponding to attribute Airline. Thus the ordering of the attributes in the IPO-tree has no impact on the query evaluation complexity.

**EXAMPLE 2 (QUERY EVALUATION).** We use the IPO-tree in Figure 2 for the illustration of the detailed steps in implicit preference query evaluation. Let us consider six different queries for illustration, namely  $Q_A : "M \prec *"$ ,  $Q_B : "M \prec *, G \prec *"$ ,  $Q_C : "G \prec *"$ ,  $Q_D : "M \prec H \prec *, G \prec *"$ ,  $Q_E : "M \prec H \prec *, G \prec R \prec *"$  and  $Q_F : "G \prec R \prec *"$ .

Consider  $Q_A$  containing a first-order implicit preference on attribute Hotel-group. We first visit Node 1 and  $X$  is set to be  $S$  of Node 1 (i.e.,  $\{a, c, d, e, f\}$ ). Node 4 is then visited where  $\mathcal{A}$  is  $\emptyset$ , and  $X$  is still  $\{a, c, d, e, f\}$ , which is the skyline for  $Q_A$ .

$Q_B$  is a query containing a first-order implicit preference on attribute Hotel-group and a first-order implicit preference on attribute Airline. After visiting Node 1,  $X =$

---

### Algorithm 3 query( $d, \tilde{R}', N, S$ )

---

**Input:** dimension  $d$ , implicit preference  $\tilde{R}'$ , tree node  $N$ , set of potential skyline points  $S$   
 Local variable:  $\mathcal{Q}$  - a queue containing sets of points

- 1:  $X \leftarrow S$
- 2: **if**  $d \neq m'$  **then**
- 3:   **if**  $R'_d$  contains no preferences **then**
- 4:      $N_c \leftarrow$  the child node of  $N$  labeled  $\phi$
- 5:      $X \leftarrow$  **query**( $d + 1, \tilde{R}', N_c, S$ )
- 6:   **else**
- 7:      $\mathcal{Q} \leftarrow \emptyset$
- 8:     **for**  $i := 1$  to  $order(\tilde{R}'_d)$  **do**
- 9:        $v \leftarrow$  the  $i$ -th entry in  $\tilde{R}'_d$
- 10:        $N_c \leftarrow$  child node of  $N$  labeled " $v \prec *$ "
- 11:        $\mathcal{A} \leftarrow$  the disqualifying set of  $N_c$
- 12:        $Y \leftarrow$  **query**( $d + 1, \tilde{R}', N_c, S - \mathcal{A}$ )
- 13:       enqueue  $Y$  to  $\mathcal{Q}$
- 14:     **end for**
- 15:      $X \leftarrow$  **merge**( $d + 1, \mathcal{Q}, \tilde{R}'$ ) (See Algorithm 4)
- 16:   **end if**
- 17: **end if**
- 18: **return**  $X$

---



---

### Algorithm 4 merge( $d, \mathcal{Q}, \tilde{R}'$ )

---

**Input:** dimension  $d$ ,  $\mathcal{Q}$  storing sets of points, preference  $\tilde{R}'$

- 1: dequeue  $\mathcal{Q}$  and obtain the dequeued element  $Y$
- 2:  $X \leftarrow Y$
- 3: **for**  $i := 2$  to  $order(\tilde{R}'_d)$  **do**
- 4: dequeue  $\mathcal{Q}$  and obtain the dequeued element  $Y$
- 5: let  $\mathcal{R}$  be the set of the first to the  $(i - 1)$ -th entries in  $\tilde{R}'_d$
- 6:    $Z \leftarrow$  a set of points  $p$  in  $X$  with  $p.D_d \in \mathcal{R}$
- 7:    $X \leftarrow (X \cap Y) \cup Z$
- 8: **end for**

---

$\{a, c, d, e, f\}$ . Next, Node 4 and Node 14 are visited. The skyline is

$$\begin{aligned} X &= \{a, c, d, e, f\} - \{d\} \\ &= \{a, c, e, f\} \end{aligned}$$

Different from  $Q_A$ ,  $Q_C$  contains a first-order implicit preference on attribute Airline, instead of attribute Hotel-group. In this case, we visit Node 5 labeled  $\phi$  representing no preference on attribute Hotel-group. Then, from Node 5, we visit Node 18. Thus, the skyline is

$$\begin{aligned} X &= \{a, c, d, e, f\} - \{d\} \\ &= \{a, c, e, f\} \end{aligned}$$

For  $Q_D$  with a second-order implicit preference on attribute Hotel-group and a first-order implicit preference on attribute Airline, we split the query into subqueries " $M \prec *, G \prec *$ " and " $H \prec *, G \prec *$ ", with respective skylines of  $\{a, c, e, f\}$  and  $\{a, c, e\}$ . The subset  $PSKY_1$  of  $SKY_1$  with Hotel-group value  $M$  is  $\{e, f\}$ . By Theorem 2, the resulting skyline is

$$\begin{aligned} X &= (\{a, c, e, f\} \cap \{a, c, e\}) \cup \{e, f\} \\ &= \{a, c, e, f\} \end{aligned}$$

Consider  $Q_E$  containing a second-order implicit preference



on attribute Hotel-group and a second-order implicit preference on attribute Airline. As illustrated in Figure 4, we follow the breakdown and obtain the skyline with respect to  $Q_E$  equal to  $\{a, c, e, f\}$ .

Similar to  $Q_C$ , since  $Q_F$  contains a second-order implicit preference on attribute Airline, instead of attribute Hotel-group, we visit Node 5 labeled  $\phi$  representing no preference on attribute Hotel-group. Then, from Node 5, we follow the breakdown as shown in Figure 3 and obtain the skyline with respect to  $Q_F = \{a, c, e, f\}$ . ■

**THEOREM 3.** *With Algorithm 3,  $query(1, \tilde{R}', Root, SKY(R))$  returns  $SKY(\tilde{R}')$ , given a template  $R$  for a dataset  $\mathcal{D}$  and the corresponding IPO-tree with a root node of  $Root$ .* ■

The number of leaf nodes in a query evaluation tree diagram as the one shown in Figure 4 gives a bound on the number of set operations. Furthermore, if  $m''$  is the number of dimensions on which the user has specified any refinement, then the number of set operations is further bounded by the number of leaf nodes of a subtree containing only nodes  $\phi$  or a preference in one of the  $m''$  dimensions specified.

**LEMMA 3.** *Given a user query with  $x$ -th order implicit preferences on  $m''$  nominal attributes. The number of set operations required for an  $x$ -th order implicit preference is  $O(x^{m''})$ .*

Since  $x$  and  $m''$  depends on the query size and are expected to be very small, this number is also small. Note that the complexity is independent on the ordering of the dimensions in the tree.

**Implementation:** We have implemented the algorithm by accumulating the set of disqualified points. By Theorem 2, if  $A(\tilde{R}')$  and  $A(\tilde{R}'')$  are the sets of disqualified points for  $\tilde{R}'$  and  $\tilde{R}''$ , respectively, let  $\mathcal{B}$  be the set of points in  $A(\tilde{R}'')$  with  $D_i$  values in  $\{v_1, \dots, v_{x-1}\}$ , the accumulated set of disqualified points for  $\tilde{R}'''$  is given by

$$A(\tilde{R}') \cup (A(\tilde{R}'') - \mathcal{B}).$$

## 5. EMPIRICAL STUDY

Extensive experiments have been conducted on a Pentium IV 3.2GHz PC with 2GB memory, on a Linux platform. The algorithms were implemented in C/C++. In the experiments, we adopted the data set generator released by the authors of [25], which contains both numeric attributes and nominal attributes, where the nominal attributes are generated according to a Zipfian distribution. The default values of the experimental parameters are shown in Table 7. In the experiment, if the order of the implicit preference  $\tilde{R}'$  is set to  $x$ , it means that the order of  $\tilde{R}'_i$  for each nominal attribute  $D_i$  is  $x$ . Note that the total number of dimensions is equal to the number of numeric dimensions plus the number of nominal dimensions. By default, we adopted a template where the most frequent value in a nominal dimension has a higher preference than all other values. This corresponds to a more difficult setting as the skyline tends to be bigger. In the following, we use the default settings unless specified otherwise.

We denote our proposed partial materialization methods (IPO Tree Search) by *IPO Tree* and *IPO Tree-10* where

Parameter	Default value
No. of tuples	500K
No. of numeric dimensions	3
No. of nominal dimensions	2
No. of values in a nominal dimension	20
Zipfian parameter $\theta$	1
order of implicit preference	3

**Table 7: Default values**

*IPO Tree* is constructed based on all possible nominal values and *IPO Tree-10* is constructed based on only the 10 most frequent values for each nominal attribute. We denote the Adaptive SFS algorithm by *SFS-A*. We also compare our proposed methods with a baseline algorithm called *SFS-D*, which is the original SFS algorithm [7] returning  $SKY(\tilde{R}')$  with respect to implicit preference  $\tilde{R}'$  for dataset  $\mathcal{D}$ .

We evaluate the performance of the algorithms in terms of

- (1) pre-processing time,
- (2) the query time of an implicit preference and
- (3) memory requirement.

We also report

- (4) the proportion of the skyline points with respect to the template  $\tilde{R}$  (i.e.,  $|SKY(R)|/|\mathcal{D}|$ ),
- (5) the proportion of skyline points affected in  $SKY(\tilde{R})$  with respect to  $\tilde{R}'$  (i.e.,  $|AFFECT(R)|/|SKY(R)|$ ), where  $AFFECT(R)$  is the set of skyline points in  $SKY(\tilde{R})$  with values in  $\tilde{R}'$ , and
- (6) the proportion of skyline points with respect to  $\tilde{R}'$  in  $SKY(\tilde{R})$  (i.e.,  $|SKY(R')|/|SKY(R)|$ ).

For pre-processing, both *IPO Tree* and *IPO Tree-10* compute  $SKY(\tilde{R})$  and build the correspondence IPO trees, and *SFS-A* computes  $SKY(\tilde{R})$  and pre-sort the data according to the preference function  $f$ . Note that *SFS-D* does not require any preprocessing. The storage of *IPO Tree* or *IPO Tree-10* corresponds to the IPO tree stored. *SFS-A* stores the sorted data in  $SKY(\tilde{R})$ , and *SFS-D* does not use extra storage but reads the data directly from the dataset.

For measurements (1) and (3), each experiment was conducted 100 times and the average of the results was reported. For measurements (2), (4), (5) and (6), in each experiment, we randomly generated 100 implicit preferences, and the average query time is reported. We will study the effects of varying (1) database size, (2) dimensionality, (3) cardinality of nominal attribute and (4) order of implicit preference.

### 5.1 Synthetic Data Set

Three types of data sets are generated as described in [1]: (1) *independent data sets*, (2) *correlated data sets* and (3) *anti-correlated data sets*. The detailed description of these data sets can be found in [1]. For interest of space, we only show the experimental results for the anti-correlated data sets. The results for the independent data sets and the correlated data sets are similar in the trend but their execution times are much shorter.

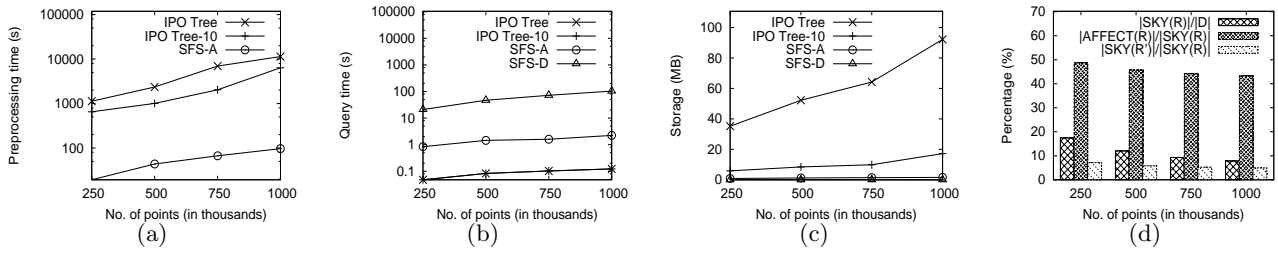


Figure 5: Scalability with respect to database size

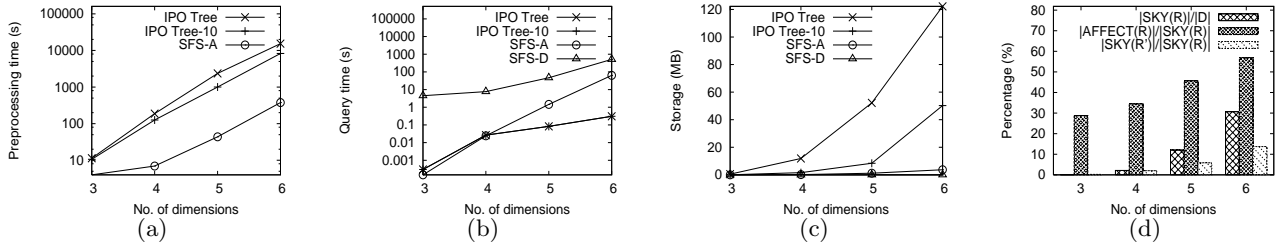


Figure 6: Scalability with respect to dimensionality where no. of nominal attributes is fixed to 2

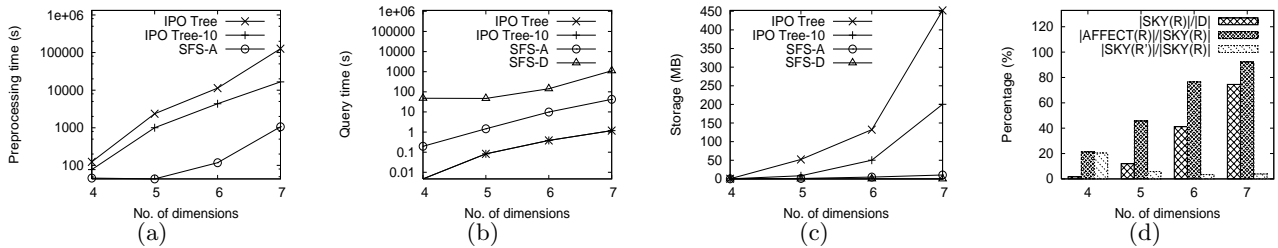


Figure 7: Scalability with respect to dimensionality where no. of numeric attributes is fixed to 3

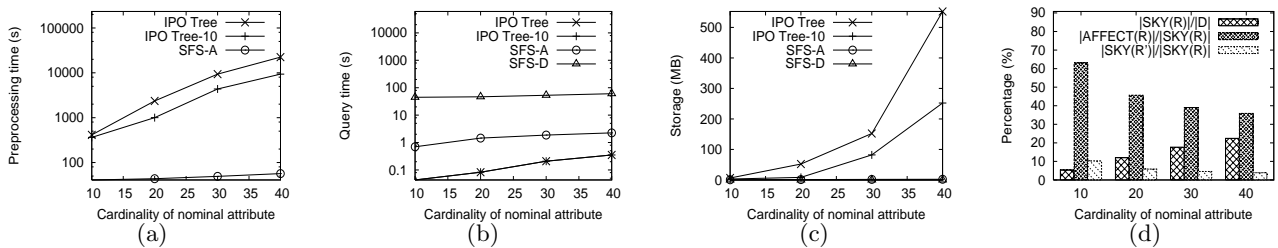


Figure 8: Scalability with respect to cardinality of nominal attribute

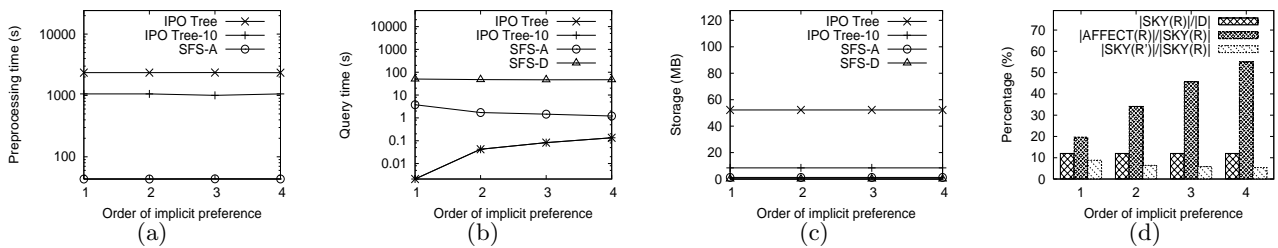


Figure 9: Effect of order of implicit preference

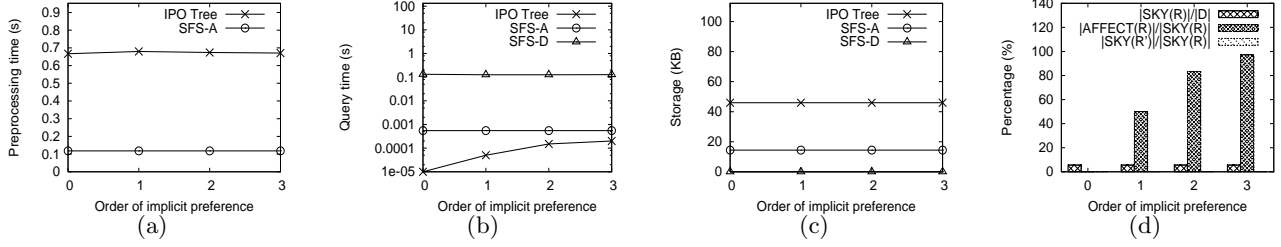


Figure 10: Effect of order of implicit preference (real data set)

### 5.1.1 Effect of Database Size

In Figure 5(d), we note that  $|SKY(R)|/|D|$  decreases slightly when the data size increases. This is because, when there are more data points, there is a higher chance that a data point is dominated by other data points. Nevertheless,  $|SKY(R)|$  increases with database size, and therefore we see an upward trend in run time and in storage. For the IPO tree methods, the skyline information size will increase with data size. For *SFS-A*, the preprocessing time is  $O(n \log n + Nn)$  and the query time is  $O(l \log N + \min(N', l) \cdot N)$ , where  $n$  is the data size,  $l = |AFFECT(R)|$ ,  $N = |SKY(\tilde{R})|$  and  $N' = |SKY(\tilde{R}')|$ . For *SFS-D* the query time is  $O(N \log N + Nn)$ . We can see that the results from graphs match with the complexity expectation.

### 5.1.2 Effect of Dimensionality

We study the effect of the number of nominal attribute  $m'$  where the number of numeric attributes is fixed to 3, with the results as shown in Figure 7. In Figure 7(d),  $|SKY(R)|/|D|$  increases. With more nominal attributes, it is less likely that the data points are dominated by others and thus  $|SKY(R)|$  increases.  $|AFFECT(R)|/|SKY(R)|$  also increases with  $m'$  because it is more likely that a data point is affected when the implicit preference contains preferences on more nominal attributes. The number of nodes in a full *IPO tree* is given by  $O(c^{m'})$  where  $c$  is the cardinality of a nominal attribute. Because of these factors, the preprocessing time and the query time of all algorithms increase with  $m'$ . For the same reason, the storage for *IPO Tree* and the storage of *SFS-A* also increase slightly. We have also studied the effect of the number of numeric attributes where the number of nominal attributes is fixed to 2. The results are similar to Figure 7.

### 5.1.3 Effect of Cardinality of Nominal Attribute

Figure 8(d) shows that  $|SKY(R)|/|D|$  increases with cardinality. This is because, when the cardinality increases, there is a higher chance that a data point is not dominated by other data points. Also, the number of nodes in a full *IPO tree* is given by  $O(c^{m'})$  where  $c$  is the cardinality of a nominal attribute and  $m'$  is the number of nominal attributes. Thus, the preprocessing time, query time and storage of our proposed algorithms increases with the cardinality. From Figure 8(b), the increase is dampened for *SFS-A* because the query time of *SFS-A* depends on  $|AFFECT(R)|$  and there is a decrease in  $|AFFECT(R)|/|SKY(R)|$ , which is caused by fewer data points with frequent nominal values when there are more values in a nominal attribute.

### 5.1.4 Effect of Order of Implicit Preference

For IPO tree, the number of set operations is given by  $O(x^{m'})$  where  $x$  is the order of implicit preference. Hence, in Figure 9(b), the query time for *IPO Tree* increases. The query times for *SFS-A* and *SFS-D* are slightly dropping because the skyline size decreases when the order of implicit preference increases. It is obvious that neither the preprocessing nor storage will be affected. Figure 9(d) shows that the size of affected skyline points increases. This is because more nominal values involved in the preference affect more data points.

## 5.2 Real Data Set

To demonstrate the usefulness of our methods, we ran our algorithms on a real data set, Nursery data set, which is publicly available from the UCIRvine Machine Learning Repository<sup>1</sup>. In this data set, there are 12,960 instances and 8 attributes. The experimental setup is same as [25]. There are six totally-order attributes and two nominal attributes, namely form of the family and the number of children. (Note that although the number of children is a numeric attribute, it is not clear whether a family with one child is “better” than a family with two children.) The cardinalities of both nominal attributes are equal to 4. The results in the performance are similar to those for the synthetic data sets. Figure 10 shows the results on the real data set with the effect of the order of implicit preference. It is noted that the storage of *IPO Tree* is smaller than 50KB. At the same time, the query time of *IPO Tree* is 0.0001s when the order of implicit preference is 2. On average, the query time of *IPO Tree* is more than 20 times shorter than that of *SFS-A*. In conclusion, in practice, *IPO Tree* is feasible to store partial results and is efficient for answering skyline queries.

It is noted that, in this real dataset, some points dominate most points and thus  $|SKY(R')|$  is small. So,  $|SKY(R')|/|SKR(R)|$  nearly equals 0 in Figure 10(d). The other figures (i.e., Figures 5, 6, 7, 8 and 9(d)) are based on synthetic anti-correlated data sets in which it is less likely that some points dominate most points. Since  $|SKY(R')|$  is not small,  $|SKY(R')|/|SKR(R)|$  is not close to 0 in the other figures.

## 5.3 Main Observations

The major findings from the experiments are the followings. The *SFS-D* algorithm cannot meet real-time requirements, since the query time is at least in terms of tens of seconds and, in some cases, exceeds 1000 seconds. In general, *IPO Tree* is the fastest. Besides, *SFS-A* returns the result for about 20 seconds in some cases and is orders of magnitude faster than *SFS-D*. The results with *IPO Tree-10*

<sup>1</sup><http://kdd.ics.uci.edu/>

show that, by handling a smaller set of nominal values, one can control both the pre-processing and storage costs.

## 6. CONCLUSION

Most previous works on the skyline problem consider data sets with attributes following a fixed ordering. However, nominal attributes with dynamic orderings according to different users exist in almost all conceivable real-life applications. In this work, we study the problem of online responses for such dynamic preferences, and a semi-materialization method is proposed. Our experiments show how our proposed algorithm computes the skyline results efficiently.

There are a lot of promising future directions with the consideration of nominal attributes. One of possible future directions is to investigate how the IPO-tree is updated when data is changed. Another possible direction is to study how the existing variations of traditional skyline models (which do not consider nominal attributes) can be applied in the data with nominal attributes. One example is to study how subspace skyline queries [27, 22, 26, 23, 21] can be applied in our problem setting with the consideration of nominal attributes. Another example is to find skyline efficiently with the consideration of nominal attributes over data streams [17]. Besides, it is also interesting to investigate how the concept of *reverse skyline* published recently in [8] can be applied in our problem setting.

## ACKNOWLEDGEMENTS

The research of R. C-W Wong and A. W-C Fu was supported in part by the RGC Earmarked Research Grant of HKSAR CUHK 4120/05E and 4118/06E. J. Pei was supported in part by an NSERC Discovery Grant. Y. Liu was supported in part by the National Natural Science Foundation of China (60703111). All opinions, findings, conclusions and recommendations in this paper are those of the authors and do not necessarily reflect the views of the funding agencies.

## 7. REFERENCES

- [1] S. Borzsonyi, D. Kossmann, and K. Stocker. The skyline operator. In *ICDE*, 2001.
- [2] C.-Y. Chan, P.-K. Eng, and K.-L. Tan. Stratified computation of skylines with partially-ordered domains. In *SIGMOD*, 2005.
- [3] C.Y. Chan, P.-K. Eng, and K.-L. Tan. Efficient processing of skyline queries with partially-ordered domains. In *ICDE*, 2005.
- [4] S. Chaudhuri, N. Dalvi, and R. Kaushik. Robust cardinality and cost estimation for skyline operator. In *ICDE*, 2006.
- [5] J. Chomicki. Database querying under changing preferences. In *Annals of Mathematics and Artificial Intelligence*, 2006.
- [6] J. Chomicki. Iterative modification and incremental evaluation of preference queries. In *FoIKS*, 2006.
- [7] J. Chomicki, P. Godfrey, J. Gryz, and D. Liang. Skyline with presorting. In *ICDE*, 2003.
- [8] E. Dellis and B. Seeger. Efficient computation of reverse skyline queries. In *International Conference on Very Large Data Bases (VLDB)*, 2007.
- [9] D. Kossmann et al. Shooting stars in the sky: An online algorithm for skyline queries. In *International Conference on Very Large Data Bases (VLDB)*, 2002.
- [10] H.T. Kung et al. On finding the maxima of a set of vectors. In *Journal of ACM*, 22(4), 1975.
- [11] J. L. Bentley et al. Fast linear expected-time algorithms for computing maxima and convex hulls. In *SODA '90*.
- [12] J. L. Bentley et al. On the average number of maxima in a set of vectors and applications. In *Journal of ACM*, 25(4), 1978.
- [13] O. Barndorff-Nielsen et al. On the distribution of the number of admissible points in a vector random sample. In *Theory of Probability and its Application*, 11(2), 1966.
- [14] W.-T. Balke et al. Eliciting matters - controlling skyline sizes by incremental integration of user preferences. In *DASFAA '07*.
- [15] W.-T. Balke et al. Exploiting indifference for customization of partial order skylines. In *the 10th International Database Engineering and Applications Symposium*, 2006.
- [16] P. Godfrey, R. Shipley, and J. Gryz. Maximal vector computation in large data sets. In *International Conference on Very Large Data Bases (VLDB)*, 2005.
- [17] X. Lin, Y. Yuan, W. Wang, and H. Lu. Stabbing the sky: Efficient skyline computation over sliding windows. In *ICDE*, 2005.
- [18] M. Morse, J. M. Patel, and H.V.Jagadish. Efficient skyline computation over low-cardinality domains. In *International Conference on Very Large Data Bases (VLDB)*, 2007.
- [19] D. Papadias, Y. Tao, G. Fu, and B. Seeger. An optimal and progressive algorithm for skyline queries. In *SIGMOD*, 2003.
- [20] D. Papadias, Y. Tao, G. Fu, and B. Seeger. Progressive skyline computation in database systems. In *TODS*, Vol. 30, No. 1, 2005.
- [21] J. Pei, A. W.-C. Fu, X. Lin, and H. Wang. Computing compressed multidimensional skyline cubes efficiently. In *ICDE*, 2007.
- [22] J. Pei, W. Jin, M. Ester, and Y. Tao. Catching the best views of skyline: A semantic approach based on decisive subspaces. In *International Conference on Very Large Data Bases (VLDB)*, 2005.
- [23] J. Pei, Y. Yuan, and X. Lin et al. Towards multidimensional subspace skyline analysis. In *ACM TODS*, 2006.
- [24] K.-L. Tan, P.K. Eng, and B.C. Ooi. Efficient progressive skyline computation. In *International Conference on Very Large Data Bases (VLDB)*, 2001.
- [25] R.C.W. Wong, J. Pei, A. Fu, and K. Wang. Mining favorable facets. In *SIGKDD*, 2007.
- [26] T. Xia and D. Zhang. Refreshing the sky: The compressed skycube with efficient support for frequent updates. In *SIGMOD*, 2006.
- [27] Y. Yuan, X. Lin, Q. Liu, W. Wang, J. X. Yu, and Q. Zhang. Efficient computation of the skyline cube. In *International Conference on Very Large Data Bases (VLDB)*, 2005.