Efficient Skyline Querying with Variable User Preferences on Nominal Attributes

Raymond Chi-Wing Wong[1], Ada Wai-chee Fu[1], Jian Pei[2], Yip Sing Ho[1], Tai Wong[1], Yubao Liu[3]
[1] The Chinese University of Hong Kong [2] Simon Fraser University [3] Sun Yat-Sen University
cwwong,adafu@cse.cuhk.edu.hk jpei@cs.sfu.ca 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, we propose two methods of different characteristics. The first one is a semi-materialization method and the second is an adaptive SFS method. Finally, we conduct experiments to show the efficiency of our proposed algorithms.

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. Consider a customer looking for a vacation package to Cancun using three criteria: price, hotel-class and number of stops. We know that lower price, higher hotel class and less stops are more preferable. Thus, if p is in the skyline, then there is no other package q which has lower price, higher hotel class and less stops compared with p.

Skyline queries have been studied since 1960s in the theory field where skyline points are known as *Pareto sets* and *admissible points* [10] or *maximal vectors* [9]. However, earlier algorithms such as [9, 8] are inefficient when there are many data points in a high dimensional space. The problem of skyline queries was introduced in the database context in [1].

Most of the existing studies handle only numeric attributes. Consider an example as shown in Table 1 showing a set of vacation packages with three attributes or dimensions[1], Price, Hotel-class and Hotel-group. Most existing works consider the first two attributes which are numeric, where lower price and higher hotel-class are more preferable. Many efficient methods have been proposed for so-called full-space skyline queries which return a set of skyline points in a specific space (a set of dimensions such as price and hotel-class). Some representative methods include a block nested loop (BNL) algorithm [1], a sort first skyline (SFS) algorithm [7], a bitmap method [19], a nearest neighbor (NN) algorithm [13] and a branch and bound skylines (BBS) method [14, 15]. Recently, skyline computation has been extended to consider subspace skyline queries which return the skylines in subspaces [23, 17, 22, 18, 16].

Hotel-group as shown in Table 1 is a categorical attribute. There can be partial ordering on categorical attributes. Some recent studies [3, 2, 4, 6, 5, 12, 11, 20] consider partially-ordered categorical attributes. In [3, 2], each partially-ordered attribute is transformed into two-integer attributes such that the conventional skyline algorithms can be applied. [4] studies the cost estimation of the skyline operator involving the partially ordered attributes.

Nevertheless, known existing work on categorical attributes assumes 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 order 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

[1]In this paper, we use the terms "*attribute*" and "*dimension*" interchangeably.

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

Table 1. Vacation packages

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

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.

Some latest works [6, 5] study the problem of preference changes, whereupon the query results can be incrementally refined. In [12], a user or a customer can specify some values in nominal attributes as an equivalence class to denote the same "importance" for those values. [11] is an extension of [12]. In [11], 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. However, these works only 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, but not on finding efficient algorithms. Here, we consider that different users may have different preferences and so the preferences are not undergoing refinement but they 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 [20] but there the study is about finding a set of partial orders with respect to which a given point is in the skyline.

In [15], dynamic skyline is considered but it is only for numeric data, 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. The BBS method does not work in our case.

Our contributions include the following. (1) To the best of our knowledge, this is the first work to study the problem of efficient skyline querying with respect to dynamic implicit preference on nominal attributes. (2) We propose two efficient algorithms of different flavors, namely IPO-Tree Search and Adaptive SFS. IPO-Tree is a partial materialization of the skylines for all possible implicit preferences. It facilitates the efficient computation of the skyline for any implicit preference. Adaptive SFS is a little slower but it does not require materialization and has the nice properties of being progressive and allows for incremental maintenance. (3) We have conducted extensive experiments to show the the efficiency of our proposed algorithms.

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 as the *partial order model*.

By default, we consider points in an m-dimensional space $\mathbb{S} = D_1 \times \cdots \times D_m$. 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 $R = (R_1, \ldots, R_m)$. 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. 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 $R = (R_1, \ldots, R_m)$ and $R' = (R'_1, \ldots, R'_m)$, $R \subseteq R'$ if and only if $R_i \subseteq R'_i$ for $1 \leq i \leq m$.* ∎

Theorem 1 (Monotonicity) *([20]) 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.

Definition 1 (Conflict-free) *([20]) 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 $(v, u) \in R'$.*

Although the model of partial order refinements can model diverse individual preferences, it does not fit tightly the real world scenarios. 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, the second choice is v'. The implicit meaning is that v and v' are better than all the other choices, say $v_1, v_2, ..., v_k$. We can model this by the partial order model, by including $v \prec v'$, $v \prec v_1$, $v \prec v_2$, ..., $v \prec v_k$ and $v' \prec v_1$, $v' \prec v_2$, ..., $v' \prec v_k$. We denote this preference by "$v \prec v' \prec *$" where $*$ means all choices other than v and v' (in this case, $*$ corresponds to $\{v_1, v_2, ..., v_k\}$). 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 $v_1, v_2, ...v_k$ be all the values in a nominal attribute D_i. An implicit preference $\widetilde{R}_i$ on D_i is given by $v_1 \prec v_2 \prec ...v_x \prec *$. 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, $\widetilde{R}_i$ is said to be an *x-th order implicit preference*. Also, the *order* of $\widetilde{R}_i$, denoted by $order(\widetilde{R}_i)$, is defined to be x and the *order* of $\widetilde{R}$ is defined to be $\max_i\{order(\widetilde{R}_i)\}$. A value v_j is said to be in $\widetilde{R}_i$ if $v_j \in \{v_1, v_2, ..., v_x\}$. Also, v_j is said to be the *j-th entry* in $\widetilde{R}_i$. $\mathcal{P}(\widetilde{R}'_i)$ is defined to be $\{(v_i, v_j) | i < j$ and $i \in [1, x]$ and $j \in [1, k]\}$. Let $\widetilde{R}' = (\widetilde{R}'_1, \widetilde{R}'_2, ..., \widetilde{R}'_m)$. $\mathcal{P}(\widetilde{R}')$ is defined to be $\bigcup_{i=1}^{m} \mathcal{P}(\widetilde{R}'_i)$.

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

Table 3. A table with two nominal attributes.

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

Definition 3 (Problem) *Given a dataset $\mathcal{D}$ and an implicit preference $\widetilde{R}'$, find the skyline $SKY(\widetilde{R}')$ in $\mathcal{D}$.* ∎

The problem defined above is our objective in this paper. We also say that we want to find a set of skyline points with respect to $\widetilde{R}'$ in $\mathcal{D}$. In many applications, online response is required. The extensive study in [15] reports that all the existing algorithms have some serious shortcomings and a new algorithm BBS is proposed which is much more efficient than previous methods. However, the data partitioning in BBS is based on fixed orderings on the dimensions and the same partitioning cannot be used for dynamic or variable preferences on nominal attributes. Therefore, new mechanisms need to be explored.

The problem of dynamic implicit preferences have some similar flavor to subspace skylines since materialization of the possible skylines seems to be a solution. However, as noted in [15], most applications involve up to five attributes, the dimensionality of a typical skyline problem is not high, and therefore materialization of the skylines is quite feasible and has been investigated in recent works such as [23, 22, 18, 16]. For dynamic implicit preferences, the number of combinations is exponential not only in the dimensionality but also in the cardinalities of the attributes, which makes the problem much more challenging.

3 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 the above, this approach is very costly in storage and preprocessing. Our study in [21] shows that, even with an index and with compression by removing redundancies in shared skylines, the cost is still prohibitive.

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

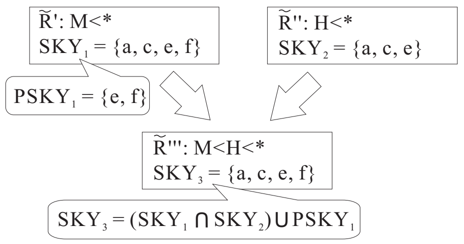

Figure 1. Illustration of the merging property

stored, the number of combinations is significantly reduced. 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 $\widetilde{R}'$ and $\widetilde{R}''$ differ only at the i-th dimension, i.e., $\widetilde{R}'_j = \widetilde{R}''_j$ for all $j \neq i$. Furthermore, $\widetilde{R}'_i$ ="$v_1 \prec ... \prec v_{x-1} \prec *$" and $\widetilde{R}''_i$ ="$v_x \prec *$". Let $PSKY(\widetilde{R}')$ be the set of points in $SKY(\widetilde{R}')$ with D_i values in $\{v_1, ...v_{x-1}\}$. Let $\widetilde{R}'''_i$ ="$v_1 \prec ... \prec v_{x-1} \prec v_x \prec *$". The skyline with respect to $\widetilde{R}'''$ is $(SKY(\widetilde{R}') \cap SKY(\widetilde{R}'')) \cup PSKY(\widetilde{R}')$.*

Proof: A proof is given in the Appendix. ■

For example, in Figure 1, let $\widetilde{R}'$ be "$M \prec *$" and $\widetilde{R}''$ be "$H \prec *$". From Table 1, the skyline with respect to $\widetilde{R}'$ is $SKY_1 = \{a, c, e, f\}$ and the skyline with respect to $\widetilde{R}''$ is $SKY_2 = \{a, c, e\}$. $PSKY_1 = \{e, f\}$ is the set of skyline points with values in $\{M\}$. Let $\widetilde{R}'''$ be "$M \prec H \prec *$". By Theorem 2, the skyline SKY_3 with respect to $\widetilde{R}'''$ is obtained as follows. $SKY_3 = (SKY_1 \cap SKY_2) \cup PSKY_1$ = $(\{a, c, e, f\} \cap \{a, c, e\}) \cup \{e, f\} = \{a, c, e\} \cup \{e, f\} = \{a, c, e, f\}$. The derivation can be explained as follows. $\mathcal{P}(\widetilde{R}')$ and $\mathcal{P}(\widetilde{R}'')$ are not conflict-free because their union contains both (M, H) and (H, M). Or, the only difference between $\mathcal{P}(\widetilde{R}') \cup \mathcal{P}(\widetilde{R}'')$ and $\mathcal{P}(\widetilde{R}''')$ is that $\mathcal{P}(\widetilde{R}') \cup \mathcal{P}(\widetilde{R}'')$ contains one more binary entry, namely (H, M), which may disqualify some data points (in this example, it disqualifies 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 ($\widetilde{R}'''$ in the theorem) skyline from lower order ($\widetilde{R}'$ and $\widetilde{R}''$) 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.

3.1 Tree Construction

An *IPO-tree* (*implicit preference order tree*) stores results for combinations of first-order preferences. In this tree, each node is labeled with a first-order implicit preference, namely "$v \prec *$", 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 ϕ 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 "$T \prec *, G \prec *$".

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 "$T \prec *, G \prec *$", 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.

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 4). However, in our implementation, we make use of the *minimal disqualifying conditions* introduced in [20]. 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 [20] can be used for this step. Then, given the implicit preference $\widetilde{R}'$ corresponding to a node N, we check each point in $SKY(R)$, if any of the MDCs is a subset of $\mathcal{P}(\widetilde{R}')$, then the point is disqualified and is inserted into $\mathcal{A}$.

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 $m' + 1$. The size of the tree in number of nodes is given by $O(c^{m'})$. As claimed in [13]

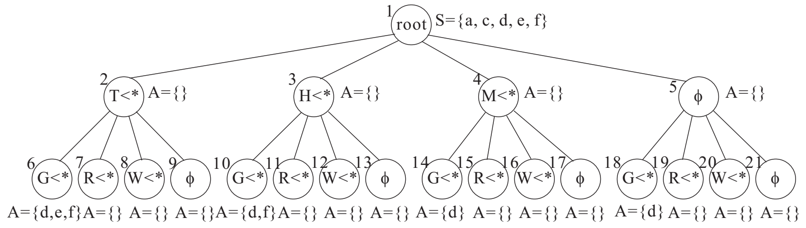

Figure 2. Illustration of an implicit preference order tree (IPO-tree)

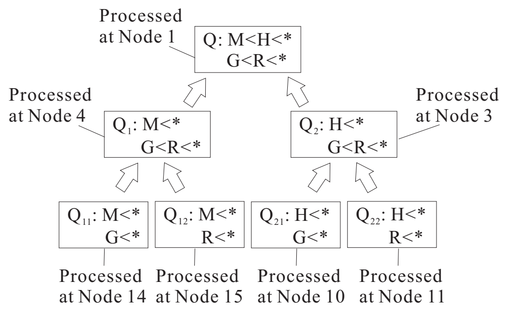

Figure 3. Query evaluation with an IPO-tree

and quoted in [15], most applications involve up to five attributes, and hence m' is very small. Note that the IPO-tree size is significantly smaller than the number of possible implicit preferences which is given by $O((c \cdot c!)^{m'})$.

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. If a query containing unpopular values arrives, the adaptive SFS algorithm in Section 4 can be used instead.

3.2 Query Evaluation

IPO-tree has a nice structure with a well-controlled tree size and can efficiently facilitate implicit preference querying based on the merging property (Theorem 2). Algorithm 1 shows the evaluation of a query with an implicit preference $\widetilde{R}'$.

Algorithm 1 query(d, $\widetilde{R}'$, N, S)

```
Input: dimension $d$, implicit preference $\widetilde{R}'$, 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($d+1$, $\widetilde{R}'$, $N_c$, $S$)
6:    else
7:      $\mathcal{Q} \leftarrow \emptyset$
8:      for $i := 1$ to $order(\widetilde{R}'_d)$ do
9:        $v \leftarrow$the $i$-th entry in $\widetilde{R}'_d$
10:       $N_c \leftarrow$ child node of $N$ labeled with "$v \prec *$"
11:       $\mathcal{A} \leftarrow$ the disqualifying set of $N_c$
12:       $Y \leftarrow$query($d+1$, $\widetilde{R}'$, $N_c$, $S - \mathcal{A}$)
13:       enqueue $Y$ to $\mathcal{Q}$
14:     $X \leftarrow$ merge($d+1$, $\mathcal{Q}$, $\widetilde{R}'$) (See Algorithm 2)
15: return $X$
```

Algorithm 2 merge(d, $\mathcal{Q}$, $\widetilde{R}'$)

```
Input: dimension $d$, $\mathcal{Q}$ storing sets of points, preference $\widetilde{R}'$
1: dequeue $\mathcal{Q}$ and obtain the dequeued element $Y$
2: $X \leftarrow Y$
3: for $i := 2$ to $order(\widetilde{R}'_d)$ 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 $\widetilde{R}'_d$
6:   $Z \leftarrow$ a set of points $p$ in $X$ with $p.D_d \in \mathcal{R}$
7:   $X \leftarrow (X \cap Y) \cup Z$
```

Example 1 (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 four different queries for illustration, namely Q_A : "$M \prec *$", Q_B : "$M \prec *, G \prec *$", Q_C : "$M \prec H \prec *, G \prec *$" and Q_D : "$M \prec H \prec *, G \prec R \prec *$".

Consider Q_A. 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 $\emptyset$, X is still $\{a, c, d, e, f\}$, which is the skyline for Q_A.

Consider Q_B. After visiting Node 1, $X = \{a, c, d, e, f\}$. Next, Node 4 and Node 14 are visited. The skyline is $X = \{a, c, d, e, f\} - \{d\} = \{a, c, e, f\}$.

Consider Q_C. 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 $(\{a, c, e, f\} \cap \{a, c, e\}) \cup \{e, f\} = \{a, c, e, f\}$.

Consider Q_D. As illustrated in Figure 3, we follow the breakdown and obtain the skyline with respect to Q_D equal to $\{a, c, e, f\}$. ■

Theorem 3 *With Algorithm 1,* ***query****(1,* $\widetilde{R}'$*,* $Root$*, SKY(R)) returns* $SKY(\widetilde{R}')$*, 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 3 gives a bound on the number of set operations. The number of set operations required for an x-th order implicit preference is $O(x^{m'})$. Since x and m' are very small, this number is also small.

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

Another efficient implementation is to store the skyline for each node in the IPO-tree by means of a bitmap (replacing $\mathcal{A}$) and to create an inverted list for each nominal attribute for an easy lookup to determine a bitmap for $PSKY(\widetilde{R}')$ (see Theorem 2). Efficient bitwise operations can then be used for the set operations.

4 Progressive Algorithm: Adaptive SFS

The IPO-tree method requires much preprocessing cost and storage. It is also more appropriate for more static datasets since changes in the datasets require rebuilding the entries in the tree. It is of interest to find an efficient algorithm which does not involve major overheads, and in addition allows incremental maintenance to accommodate dynamic updating of the datasets. Here, we propose such a method for real-time querying which is based on the Sort-First Skyline Algorithm (SFS) [7]. The algorithm is called Adaptive SFS and is efficient since it does not require a complete resorting of the data for each different user preference. It also allows skyline points to be returned in a progressive manner.

4.1 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.

4.2 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 $SKY(\widetilde{R})$ for a template order $\widetilde{R}$ which is implicit. Let $\widetilde{R}'$ be an implicit refinement over $\widetilde{R}$. From Theorem 1, any skyline point p for $\widetilde{R}'$ will also be a skyline point for $\widetilde{R}$. Hence, in order to look for the skyline for $\widetilde{R}'$, we only need to search $SKY(\widetilde{R})$.

Algorithm 3 Preprocessing

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

Algorithm 4 Query Processing

```
Input: skyline query, with implicit preference $\widetilde{R}'$
1: Determine the ranking for the values in $\widetilde{R}'$
2: Find the data points in $SKY(\widetilde{R})$ that contain values in $\widetilde{R}'$.
   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
```

Our idea is the following. We adopt the basic presorting step on $SKY(\widetilde{R})$ resulting in a sorted list $L(\widetilde{R})$. When a query with a refinement $\widetilde{R}'$ arrives, we first try to re-sort the list $L(\widetilde{R})$ and obtain a new sorted list $L(\widetilde{R}')$. The skyline generation step is then applied on $L(\widetilde{R}')$. 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 refinement $\widetilde{R}'$ 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 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 $\widetilde{R}'_i$, we can determine a ranking for the values that appear in $\widetilde{R}'_i$ so that $r(v) < r(v')$ if and only if $v \prec v'$ can be derived from $\widetilde{R}'_i$. If $\widetilde{R}'_i$ is "$v_1 \prec v_2 \prec ... \prec v_x \prec *$", then we set $r(v_1) = 1$, $r(v_2) = 2$, ..., $r(v_x) = x$. 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 $\widetilde{R}'$. The processing time of the sorting list is $O(l \log n)$. Algorithms 3 and 4 show the steps for preprocessing the data points and query processing, respectively.

In Step 2 in Algorithm 4, in order to find data points in $SKY(\widetilde{R})$ that contain values in $\widetilde{R}'$, 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 $\widetilde{R}'$. Such data points are collected in a set. Then, for each point p in the set, the value of $f(p)$ based on $\widetilde{R}$ 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 $\widetilde{R}'$.

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 a is re-inserted, we need only check if it may be dominated by the $\widetilde{R}'$ skyline points sorted before it. If so, a is not added; otherwise, we then check if it may dominate any $SKY(\widetilde{R})$ skyline point that are sorted after it. The points that it dominates will be removed. Let $c = |SKY(\widetilde{R}')|$, $n = |SKY(\widetilde{R})|$, and l be the number of points in $SKY(\widetilde{R})$ containing values in $\widetilde{R}'$. The time complexity of this step will become $O(l \log l + c \cdot l + \min(c, l) \cdot n)$. Since the resorting step takes $O(l \log n)$ time, the total time is $O(l \log n + \min(c, l) \cdot n)$.

4.3 Properties of Adaptive SFS

The presorting ensures that a point p dominating another point q must be visited before q. This leads to a *progressive* behavior, meaning that any point inserted into the skyline list L must be in the skyline set, and it can be reported immediately. The presorting also enhances the pruning since it is more likely that candidate points with lower scores dominate more other points. Another desirable property of adaptive SFS is that it allows *incremental maintenance*. Assume that the algorithm which finds $SKY(\widetilde{R})$ is incremental. After data is updated, the set $SKY(\widetilde{R})$ is modified. The sorted list in the method is altered by simple insertions or deletions. The time complexity is $O(\log n)$ for each such update.

5 Empirical Study

We have conducted extensive experiments on a Pentium IV 3.2GHz PC with 2GB memory, on a Linux platform. The algorithms were implemented in C/C++. In our experiments, we adopted the data set generator released by the authors of [20], 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 4. In the experiment, if the order of the implicit preference $\widetilde{R}'$ is set to x, it means that the order of $\widetilde{R}'_i$ 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 *IPO Tree* is constructed based on all possible nominal values

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 θ	1
order of implicit preference	3

Table 4. Default 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 $SKY(\widetilde{R}')$ with respect to implicit preference $\widetilde{R}'$ 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 $\widetilde{R}$ (i.e., $|SKY(R)|/|\mathcal{D}|$), (5) the proportion of skyline points affected in $SKY(\widetilde{R})$ with respect to $\widetilde{R}'$ (i.e., $|AFFECT(R)|/|SKY(R)|$), where $AFFECT(R)$ is the set of skyline points in $SKY(\widetilde{R})$ with values in $\widetilde{R}'$, and (6) the proportion of skyline points with respect to $\widetilde{R}'$ in $SKY(\widetilde{R})$ (i.e., $|SKY(R')|/|SKY(R)|$). For pre-processing, both *IPO Tree* and *IPO Tree-10* compute $SKY(\widetilde{R})$ and build the correspondence IPO trees, and *SFS-A* compute $SKY(\widetilde{R})$ 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 $SKY(\widetilde{R})$, 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.

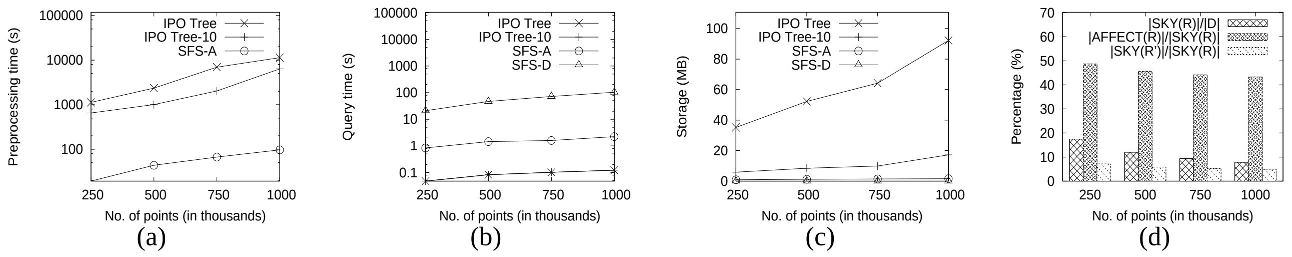

Figure 4. Scalability with respect to database size

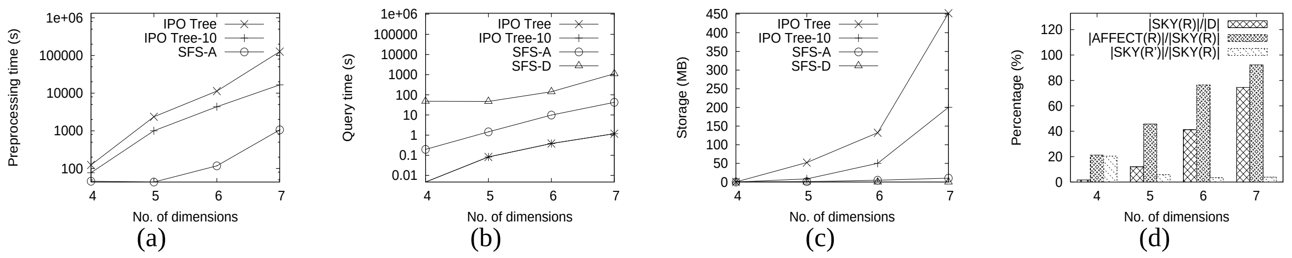

Figure 5. Scalability with respect to dimensionality where no. of numeric attributes is fixed to 3

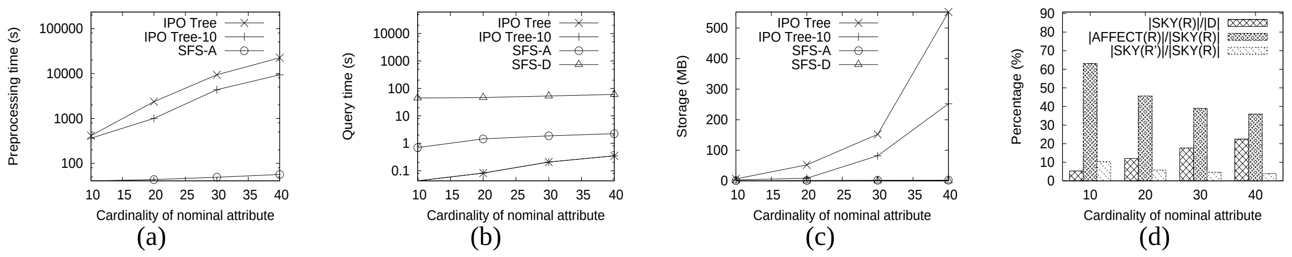

Figure 6. Scalability with respect to cardinality of nominal attribute

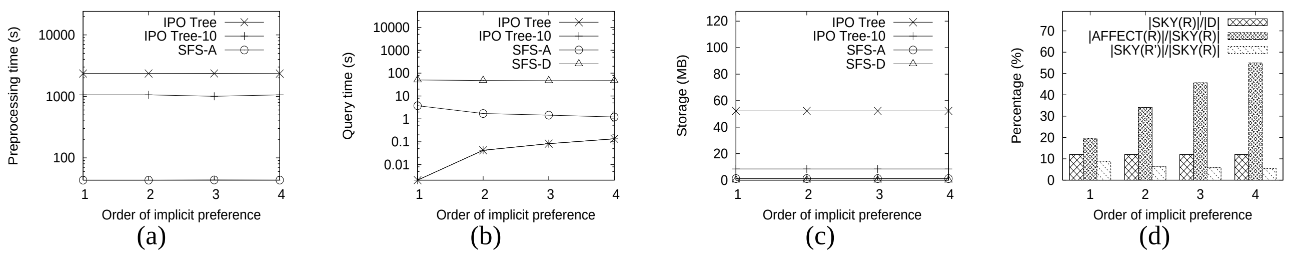

Figure 7. Effect of order of implicit preference

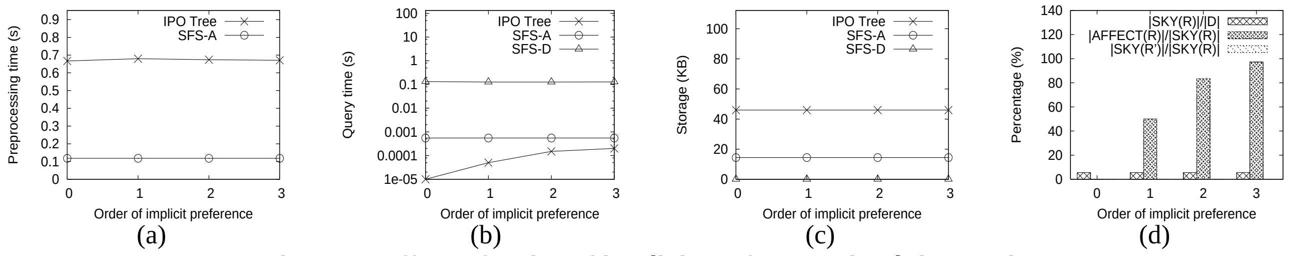

Figure 8. Effect of order of implicit preference (real data set)

Effect of the database size: In Figure 4(d), we note that $|SKY(R)|/|\mathcal{D}|$ 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 $O(NlogN + Nn)$ and the query time is $O(l \log n + \min(c, l) \cdot n)$, where N is the data size, $l = |AFFECT(R)|$, $c = |SKY(R')|$ and $n = |SKY(R)|$. For *SFS-D* the query time is $O(NlogN + Nn)$. We can see that the results from graphs match with the complexity expectation.

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 5. In Figure 5(d), $|SKY(R)|/|\mathcal{D}|$ 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.

Effect of Cardinality of Nominal Attribute: Figure 6(d) shows that $|SKY(R)|$ 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 6(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.

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 7(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 pre-processing or storage will be affected. Figure 7(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[2]. In this data set, there are 12,960 instances and 8 attributes. The experimental setup is same as [20]. 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 cardinality of both nominal attributes are equal to 4. The results in the performance are similar to those for the synthetic data sets. Figure 8 shows the results on the real data set with the effect of the order of implicit preference.

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 but *SFS-A* can also return the result within a second in most cases and under 20 seconds in the worst case, and is orders of magnitude faster than *SFS-D*. The results with *IPO Tree-10* show that, by handling a smaller set of nominal values, one can control both the preprocessing and storage costs. A hybrid approach adopting *IPO Tree* for popular values and *SFS-A* for handling queries involving the remaining values is a sound solution.

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 response for such dynamic preferences, two methods are proposed with different flavors: a semi-materialization method and an adaptive SFS method. Our experiments show how our proposed algorithms are useful in different problem settings.

References

[1] S. Borzsonyi, D. Kossmann, and K. Stocker. The skyline operator. In *ICDE*, 2001.

[2] C. Chan, P.-K. Eng, and K.-L. Tan. Efficient processing of skyline queries with partially-ordered domains. In *ICDE*, 2005.

[2]`http://kdd.ics.uci.edu/`

[3] C.-Y. Chan, P.-K. Eng, and K.-L. Tan. Stratified computation of skylines with partially-ordered domains. In *SIGMOD*, 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] J. L. B. et al. Fast linear expected-time algorithms for computing maxima and convex hulls. In *SODA'90*.

[9] J. L. B. et al. On the average number of maxima in a set of vectors and applications. In *Journal of ACM, 25(4)*, 1978.

[10] O. B.-N. et al. On the distribution of the number of admissable points in a vector random sample. In *Theory of Probability and its Application, 11(2)*, 1966.

[11] W.-T. B. et al. Eliciting matters - controlling skyline sizes by incremental integration of user preferences. In *DASFAA'07*.

[12] W.-T. B. et al. Exploiting indifference for customization of partial order skylines. In *the 10th International Database Engineering and Applications Symposium*, 2006.

[13] D. Kossmann, F. Ramsak, and S. Rost. Shooting stars in the sky: An online algorithm for skyline queries. In *VLDB*, 2002.

[14] D. Papadias, Y. Tao, G. Fu, and B. Seeger. An optimal and progressive algorithm for skyline queries. In *SIGMOD*, 2003.

[15] D. Papadias, Y. Tao, G. Fu, and B. Seeger. Progressive skyline computation in database systems. In *ACM Transactions on Database Systems, Vol. 30, No. 1*, 2005.

[16] J. Pei, A. W.-C. Fu, X. Lin, and H. Wang. Computing compressed multidimensional skyline cubes efficiently. In *ICDE'07*.

[17] J. Pei, W. Jin, M. Ester, and Y. Tao. Catching the best views of skyline: A semantic approach based on decisive subspaces. In *VLDB*, 2005.

[18] J. Pei, Y. Yuan, and X. L. et al. Towards multidimensional subspace skyline analysis. In *ACM TODS*, 2006.

[19] K.-L. Tan, P. Eng, and B. Ooi. Efficient progressive skyline computation. In *VLDB*, 2001.

[20] R. Wong, J. Pei, A. Fu, and K. Wang. Mining favorable facets. In *SIGKDD*, 2007.

[21] R. Wong, J. Pei, A. Fu, and K. Wang. Online skyline analysis with dynamic preferences on nominal attributes. In *Technical Report, http://www.cse.cuhk.edu.hk/∼kdd*, 2007.

[22] T. Xia and D. Zhang. Refreshing the sky: The compressed skycube with efficient support for frequent updates. In *SIGMOD*, 2006.

[23] Y. Yuan, X. Lin, Q. Liu, W. Wang, J. X. Yu, and Q. Zhang. Efficient computation of the skyline cube. In *VLDB*, 2005.

7 Appendix: Proof of Theorem 2

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

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

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

Consider Condition 2. Since $p \notin SKY(\widetilde{R}'')$, there exists a data point q dominating p w.r.t $\widetilde{R}''$. In other words, with respect to $\widetilde{R}''$, $q.D_k \preceq p.D_k$ 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 $\widetilde{R}''$. Besides, for all dimensions D_k other than D_i, the partial orders of $\widetilde{R}''$ and $\widetilde{R}'''$ are the same. Hence, w.r.t. $\widetilde{R}'''$, $q.D_k \preceq p.D_k$ 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 $\widetilde{R}''$ and the partial orders in $\widetilde{R}''_j$ are those in $\widetilde{R}'''_j$, we have $q.D_j \prec p.D_j$ w.r.t. $\widetilde{R}'''$. Also, w.r.t. $\widetilde{R}'''$, $q.D_k \preceq p.D_k$ 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 $\widetilde{R}'''$. Otherwise, p is dominated by q w.r.t $\widetilde{R}'''$, and p cannot be in $SKY(\widetilde{R}''')$. Since $p.D_i \prec q.D_i$ w.r.t $\widetilde{R}'''$, we have $p.D_i \neq q.D_i$. Since $q.D_k \preceq p.D_k$ w.r.t. $\widetilde{R}''$ for all k, and $p.D_i \neq q.D_i$, we have $q.D_i \prec p.D_i$ w.r.t $\widetilde{R}''$. Since the implicit preference in $\widetilde{R}''$ is "$v_x \prec *$", we conclude that $p.D_i$ cannot be v_x. Since $\widetilde{R}'''$ is "$v_1 \prec ... \prec v_x \prec *$" and $p.D_i \prec q.D_i$ w.r.t $\widetilde{R}'''$, $p.D_i$ must be in $\{v_1, ...v_{x-1}\}$. 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. $\widetilde{R}''$. Besides, since the implicit preference in $\widetilde{R}''$ is "$v_x \prec *$", $q.D_i$ must be equal to v_x and $p.D_i$ cannot be equal to v_x. Since $p \in SKY(\widetilde{R}''')$, there is no other point including q dominating p w.r.t. $\widetilde{R}'''$. Note that, w.r.t. $\widetilde{R}'''$, $q.D_k \preceq p.D_k$ for all $k(\neq i)$. We obtain $p.D_i \preceq q.D_i$ w.r.t. $\widetilde{R}'''$. (Otherwise, $q.D_i \prec p.D_i$ w.r.t $\widetilde{R}'''$ and p is dominated by q w.r.t. $\widetilde{R}'''$, which leads to a contradiction.) Besides, since $q.D_i = v_x$, $p.D_i \neq v_x$ and $\widetilde{R}'''$ is "$v_1 \prec ... \prec v_x \prec *$", $p.D_i$ must be in $\{v_1, ...v_{x-1}\}$. However, this violates Condition 1. Hence, we arrive at a contradiction.

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

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

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

Consider Condition 3. Since $p \in SKY(\widetilde{R}')$ and $p \notin SKY(\widetilde{R}''')$ where $\widetilde{R}'''_i$ is a refinement of $\widetilde{R}'_i$, and $\widetilde{R}'''_k = \widetilde{R}'_k$ for all $k \neq i$, we deduce that $q.D_i \prec p.D_i$ exists in partial orders of $\widetilde{R}'''$ but not in partial orders of $\widetilde{R}'$. Since $q.D_i \prec p.D_i$ w.r.t. $\widetilde{R}'''$, $p.D_i \in \{v_1, ..., v_{x-1}\}$ and $\widetilde{R}'''$ is "$v_1 \prec ... \prec v_x \prec *$", we deduce $q.D_i \in \{v_1, ...v_{x-2}\}$. For each possible binary order $q.D_i \prec p.D_i$ w.r.t. $\widetilde{R}'''$ where $p.D_i \in \{v_1, ..., v_{x-1}\}$ and $q.D_i \in \{v_1, ...v_{x-2}\}$, we also conclude that $q.D_i \prec p.D_i$ exists in the partial orders of $\widetilde{R}'$, which leads to a contradiction.

Consider Condition 4. Since $\widetilde{R}'$, $\widetilde{R}''$ and $\widetilde{R}'''$ 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. $\widetilde{R}'''$, it is also true w.r.t. $\widetilde{R}'$ or $\widetilde{R}''$. Therefore, q also dominates p w.r.t. $\widetilde{R}'$ or $\widetilde{R}''$. That is, $p \notin SKY(\widetilde{R}')$ or $p \notin SKY(\widetilde{R}'')$, which leads to a contradiction. ∎