Online Skyline Analysis with Dynamic Preferences on Nominal Attributes

Raymond Chi-Wing Wong, *Member*, *IEEE*, Jian Pei, *Senior Member*, *IEEE*, Ada Wai-Chee Fu, *Member*, *IEEE*, and Ke Wang, *Senior Member*, *IEEE*

Abstract—The importance of skyline analysis has been well recognized in multicriteria decision-making applications. All of the previous studies assume a fixed order on the attributes in question. However, in some applications, users may be interested in skylines with respect to various total or partial orders on nominal attributes. In this paper, we identify and tackle the problem of online skyline analysis with dynamic preferences on nominal attributes. We investigate how changes of orders in attributes lead to changes of skylines. We address two novel types of interesting queries: a *viewpoint query* returns with respect to which orders a point is (or is not) in the skylines, and a *refined skyline query* retrieves the skyline with respect to a specific order. We develop two methods systematically and report an extensive performance study using both synthetic and real data sets to verify the effectiveness and the efficiency of our methods.

Index Terms—Skyline, materialization, data warehouses, preferences.

1 Introduction

Dominance analysis is important in many multicriteria decision-making applications. As an example, consider a customer looking for a vacation package to Beijing using three criteria: price, hotel class, and number of stops. For two packages p and q, if p is better than q in one factor and is not worse than q in any other factors, then p is said to *dominate* q. For example, we have three packages in (price, hotel class, number of stops): p_1 (1,600, 4, 1), p_2 (3,000, 5, 2) and p_3 (2,000, 3, 2). We know that a lower price, a higher hotel class, and fewer stops are more preferable. Thus, p_1 dominates p_3. Package p_2 and p_3 do not dominate each other because p_3 has a lower price than p_2 and p_2 has a higher hotel class than p_3.

A point that is not dominated by any other point is said to be a *skyline point* or to be in the *skyline*. The packages in the skyline are the best possible trade-offs among the three factors in question. For example, in the above vacation package example, p_1 and p_2 are in the skyline, and p_3 is not.

In order to conduct a skyline analysis, an (either total or partial) order is assumed on each attribute to reflect the users' preference. For example, a lower price, a higher hotel class, and fewer stops are more preferable. Most previous studies [5], [11], [22], [16], [17], [18], [7], [6], [8] assume that *orders on attributes are predefined* for all users. Recently, some studies [9], [10], [2], [1] consider that orders on attributes are different with different users. However, we are the first to investigate how different orders may affect the skyline property of a point.

• *R.C.-W. Wong is with the Department of Computer Science and Engineering, Hong Kong University of Science and Technology, Clear Water Bay, Kowloon, Hong Kong. E-mail: raywong@cse.ust.hk.*
• *J. Pei and K. Wang are with the School of Computing Science, Simon Fraser University, 8888 University Drive, Burnaby, BC V5A 1S6, Canada. E-mail: {jpei, wangk}@cs.sfu.ca.*
• *A.W.-C. Fu is with the Department of Computer Science and Engineering, Chinese University of Hong Kong, Shatin, Hong Kong. E-mail: adafu@cse.cuhk.edu.hk.*

Manuscript received 6 June 2007; revised 26 Nov. 2007; accepted 2 June 2008; published online 10 June 2008.
For information on obtaining reprints of this article, please send e-mail to: tkde@computer.org, and reference IEEECS Log Number TKDE-2007-06-0258.
Digital Object Identifier no. 10.1109/TKDE.2008.115.

1.1 Motivating Example

Consider a skyline analysis on selecting vacation packages. Table 1 shows a synthesis data set as our running example. Different from attributes price and hotel class on which a total order exists for all customers, on attribute hotel group, different users may have different preferences. We refer to such an attribute that does not come with a fixed order for all users a **nominal attribute**.

Nominal attributes are common in data analysis. When vacation packages are analyzed, some commonly used nominal attributes include hotel group and airline. In addition to skyline analysis on vacation packages illustrated here, it is easy to name a few other important business applications such as choosing realities (where the type of realty, style, and regions are examples of nominal attributes) and selecting air flights (where airline and transition airport are examples of nominal attributes).

What challenges do nominal attributes bring to skyline analysis? The change of preference orders on nominal attributes may lead to changes in skylines. Due to changes of orders on nominal attributes, some skyline points may become dominated by other points, and some points that are not in the skyline may become so if they are preferred by the updated order.

For example, consider the packages in Table 1. When a customer prefers Mozilla to Horizon and Tulips, package f is in the skyline. However, for another customer preferring Tulips to Mozilla, f is not in the skyline anymore since it is dominated by a. Another interesting observation is that packages a and c are always in the skyline no matter which preference order is chosen (because a has the lowest price and c has the highest hotel class).

TABLE 1
A Set of Packages

Package ID	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)

1.2 Challenges

The changes of skylines due to dynamic preference orders on attributes lead to two important and interesting problems, which have not been investigated before.

Problem 1: Viewpoint queries. *A data point in a data set may or may not be in the skyline, depending on the preference orders on the nominal attributes. Suppose a sales manager is trying to campaign for a vacation package p. (S)he may want to know with respect to what preference orders on the hotel groups and airlines that p is (or is not) in the skyline. This information helps the manager to target the customers whose preferences are consistent with the orders.*

Generally, for a point p, a viewpoint query returns the summarization of the orders with respect to which p is (or is not) in the skyline. With a viewpoint query, we can understand not only whether p is in the skyline but also the conditions on dimensions that can affect the skyline membership of p.

Unfortunately, the existing skyline computation methods cannot answer viewpoint queries unless the skylines of *all possible* preference orders are computed and compared, which is often prohibitively expensive on large databases.

Problem 2: Refined skyline queries. *With the changes of preference orders on some nominal attributes, how does the skyline change? Given a set of preference orders on the nominal attributes, what is the skyline?*

Refined skyline queries are important for online skyline analysis, since different customers may bear different preferences and many customers may want their skyline queries to be answered *online.* An existing skyline computation method can be used to compute *on the fly* the skyline for every refined skyline query. However, it is too costly on large databases of high dimensionality to support online skyline analysis.

The above two problems are challenging when there are many data points in the data set and, particularly, when online query answering is required. In this paper, we will develop materialization methods to support online query answering for the above two types of queries.

1.3 Our Contributions

In this paper, we study the problem of online skyline analysis with dynamic preferences on nominal attributes. To the best of our knowledge, this is the first effort to tackle the problem. We make the following contributions.

First, we identify two new types of skyline queries: viewpoint queries and refined skyline queries, which can disclose novel information about skylines that cannot be addressed by the existing methods. Both types of queries have not been studied by others before.

Second, we develop two effective methods. Our Minimal Disqualifying Condition (MDC) method finds the minimal conditions on the preference orders that disqualify a point from the skyline. Using those MDCs, we can answer the new types of skyline queries effectively.

Our Compressed Ordered Skyline Tree (CST) method virtually computes the skylines for all possible preference orders. To reduce the cost in both space and computation time, the sharing in both computation and storage among skylines of various orders is systematically exploited.

Last, we present a comprehensive performance study using both real data sets and synthetic data sets to verify the effectiveness and the efficiency of our methods. The experimental results show that the skyline analysis with dynamic preferences on nominal attributes is interesting, and our proposed methods can online answer the new types of skyline queries effectively.

The rest of the paper is organized as follows: In Section 2, we investigate the changes of skylines due to changes of orders on nominal attributes. The MDC and CST methods are developed in Sections 3 and 4, respectively. A systematic performance study is reported in Section 5. We discuss related work in Section 6. The paper is concluded in Section 7.

2 Skylines and Orders on Nominal Attributes

A skyline analysis involves multiple attributes. A user's preferences 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 becomes 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 $(u, v) \in R$.

By default, we consider points in an m-dimensional[1] 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. 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 $p.D_i \preceq q.D_i$ or simply $p \preceq_{D_i} q$. We can omit D_i if it is clear from the context.

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 n points in space $\mathbb{S}$, a point $p \in \mathcal{D}$ is in the **skyline** of $\mathcal{D}$ (or a **skyline point** in $\mathcal{D}$) if p is not dominated by any points in $\mathcal{D}$. The skyline of $\mathcal{D}$, denoted by $SKY(\mathcal{D})$, is the set of skyline points in $\mathcal{D}$.

In an application, there often exist some orders on the dimensions that hold for all users. In our running example, as shown in Table 1, a lower price and a higher hotel class are always more preferred by a customer. Even, for nominal

1. In this paper, we use the terms "*attribute*" and "*dimension*" exchangeably.

attributes, there may exist some universal partial orders. For example, in the realty search, detached houses are often more preferred than semidetached 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 **weaker** than R', and R' is said to be **stronger** than R.

Example 1 (preference orders). In Table 1, the numeric attributes, price and hotel class, are totally ordered. The lower the price and the higher the hotel class, the more preferable a vacation package is.

Hotel group is a nominal attribute. Different users may have different preferences on that attribute. For example, 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 weaker than R', and 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* $1 \leq i \leq m$.

Due to dynamic preference orders on nominal attributes, we propose the refined skyline queries.

Definition 1 (refined skyline query). *Let R be a template. Given a refinement R' of R, a* ***refined skyline query*** *is to find the skyline points with respect to R'.*

Example 2 (refined skyline queries). In our running example, suppose the template order R on dimension hotel group is $\emptyset$. That is, the template does not prefer any hotel group to another. On attributes price and hotel class, the lower the price and the higher the hotel class, the more preferable a vacation package is.

Given a refinement R' of R such that $R' = \{(T, M), (H, M)\}$, a refined skyline query is to find all packages in the skyline with respect to R', which are a and c.

Naïvely, one can adopt any existing algorithms such as [7] and [6] to compute the skyline set of a refinement R' of R. However, when there are many data points in the data set, those algorithms are costly and cannot achieve online query answering. This motivates us to propose two materialization methods in Sections 3 and 4, which first precompute the skyline set of each possible order and compress them in a way such that after the preprocessing, the skyline set can be returned very quickly. The compression is based on some inherent properties in this problem. One of the properties is the monotonicity property.

Consider order R and points p and q such that $q \preceq p$ with respect to R. Then, for any attribute D, we have $q \preceq_D p$ with respect to R. Let R' be a refinement of R where $R \subseteq R'$. Then, $q \preceq_D p$ still holds in R'. That is, $q \preceq p$ still holds in R'. We observe the following interesting property.

Theorem 1 (monotonicity). *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.*

Proof. Since p is not in the skyline with respect to R, there exists another data point $q \in \mathcal{D}$ which dominates p with respect to R. That is, for any dimension $D_i \in \mathbb{S}$, with respect to R, $q.D_i \preceq q.D_i$, and there exists a dimension $D_{i_0} \in \mathbb{S}$ such that $q.D_{i_0} \prec p.D_{i_0}$. Since R' is a refinement of R and $R \subseteq R'$, q also dominates p with respect to R'. Therefore, p is not in the skyline with respect to R'. □

Theorem 1 indicates that when the orders on dimensions are strengthened, some skyline points may be disqualified. However, a nonskyline point never gains the skyline membership due to a stronger order. This monotonic property greatly helps in analyzing skylines with respect to various orders.

Generally, all the possible refinements of a given template order form a lattice, where the template order serves as the unit element. In our running example, let the template be as claimed in Example 2. All possible orders on attribute hotel group form a lattice, as shown in Fig. 1. Each node in the lattice is associated with a refinement of the template. The skylines with respect to the orders are also shown. We write the orders in the transitive closure. For example, order $\{(T, H), (H, M)\}$ is not shown in the lattice, since the closure of $\{(T, H), (H, M)\}$ is $\{(T, H), (H, M), (T, M)\}$.

The node at the top corresponds to the template order. Particularly, b is not a skyline point because it is dominated by a on attributes price and hotel class, and it has the same value with a on hotel group.

Interestingly, although e and f are worse than a on attributes price and hotel class, a does not dominate e nor f since the template does not prefer T to M. In fact, *both e and f are skyline points with respect to the template.*

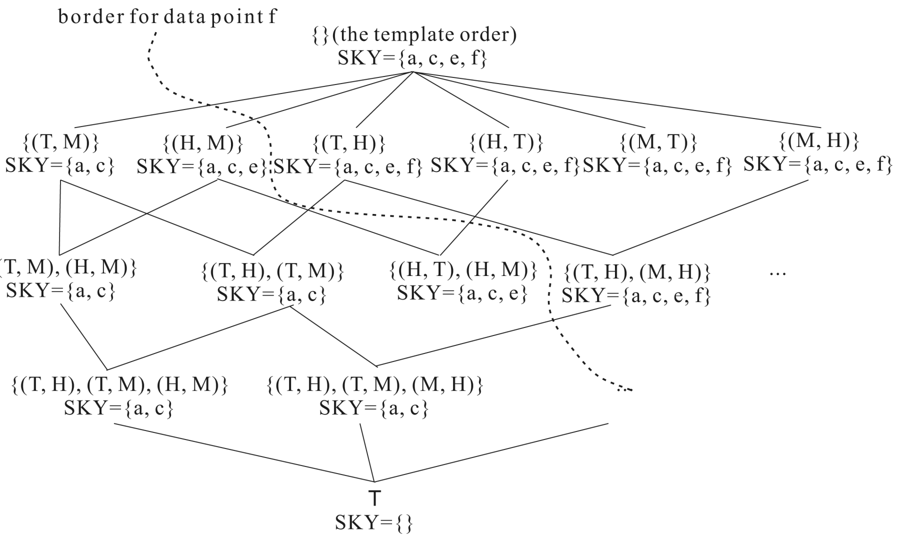
The node at the bottom is a special node that corresponds to a zero element $\top$. It makes the lattice complete. The skyline with respect to $\top$ is defined to be the empty set.

When there are multiple nominal attributes, each attribute has a corresponding lattice based on the template and refinements. The lattice for the multiple attributes is the product of the lattices for all attributes. The properties for the single nominal attribute carry forward to the product lattice.

Since a point may appear in the skylines with respect to some orders, it is interesting to query with respect to which orders a point is in the skylines.

Definition 2 (viewpoint query). *If p is a skyline point with respect to an order R_p, R_p is called a* ***skyline viewpoint*** *for p. Given a data point p and a template R, a* ***viewpoint query*** *is to find all R_p that are skyline viewpoints for p and are refinements of R (i.e., $R \subseteq R_p$).*

One naïve solution to answer a viewpoint query is to compute the skyline sets for all possible refinements R' of the template R, as shown in Fig. 1, and find all refinements with respect to which a given query point p is in the skylines. However, when there are many possible refinements, the naïve method has to compute skylines many times. In Table 1, as long as Tulip does not dominate Mozilla, e will be a skyline point. The satisfied refinements are $M \prec T \prec H$, $T \prec H$, $H \prec T$, $H \prec M$, $M \prec H \prec T$, $M \prec T$, $M \prec H$, or no preference on hotel. Instead of



border for data point f
{}(the template order) SKY={a, c, e, f}
{(T, M)} SKY={a, c}
{(H, M)} SKY={a, c, e}
{(T, H)} SKY={a, c, e, f}
{(H, T)} SKY={a, c, e, f}
{(M, T)} SKY={a, c, e, f}
{(M, H)} SKY={a, c, e, f}
{(T, M), (H, M)} SKY={a, c}
{(T, H), (T, M)} SKY={a, c}
{(H, T), (H, M)} SKY={a, c, e}
{(T, H), (M, H)} SKY={a, c, e, f}
...
{(T, H), (T, M), (H, M)} SKY={a, c}
{(T, H), (T, M), (M, H)} SKY={a, c}
T SKY={}

Fig. 1. The lattice of refinement orders.

returning all refinements, returning a succinct summarization of the refinements is more useful. In this paper, we will propose a concise and succinct representation of answers to viewpoint queries based on the following observation.

Example 3 (viewpoint queries). In Table 1, with respect to what refinements of the template R that f is in the skylines, let us answer the viewpoint query with the help of the refinement order lattice in Fig. 1.

Point f is in the skyline provided that none of the following two conditions hold: $T \prec M$ (otherwise, $a \prec f$) or $H \prec M$ (otherwise, $c \prec f$). Thus, (T, M) and (H, M) are called the *MDCs*.

In the refinement order lattice, if an order R_x contains neither $T \prec M$ nor $H \prec M$, any order weaker than R_x does not contain these conditions. On the other hand, if an order R_x contains one of these two conditions, so does every order stronger than R_x. In other words, the *border* between the orders that qualify f as a skyline point and those that disqualify f partitions the lattice into two parts: f is a skyline point in the upper part and is not a skyline point in the lower part, as shown in Fig. 1.

Based on the observations in Example 3, to answer a viewpoint query, we only need to compute the border. Particularly, we can use the set of MDCs.

Let R and R' be two partial orders. R and R' are **conflict free** if there exist no values u and v, where $u \neq v$, such that $(u, v) \in R$ and $(v, u) \in R'$. $R \cup R'$ is still a partial order if R and R' are conflict free. For example, if $R = \{(T, H)\}$ and $R' = \{(T, M)\}$, R and R' are conflict free.

Definition 3 (MDC). *For a skyline point p and a template order R, a partial order R' is called an MDC 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 MDCs for p is denoted by $MDC(p)$.

Example 4 (MDC). In our running example, $R' = \{(T, M)\}$ and $R'' = \{(H, M)\}$ are the minimal conditions that disable f as a skyline point. They are the MDCs of f. That is, $MDC(f) = \{\{(T, M)\}, \{(H, M)\}\}$.

Based on the monotonicity in Theorem 1, we can show that MDCs can be used to answer viewpoint queries effectively.

Theorem 2 (answering viewpoint queries). *For a point p and a template order R, p is in the skyline with respect to a refinement R' of R if and only if 1) p is in the skyline with respect to R and 2) $\forall R_x \in MDC(p)$, $R_x \not\subseteq R'$.*

Proof. First, we will show that if for a point p and a template order R, p is in the skyline with respect to a refinement R' of R, then 1) p is in the skyline with respect to R and 2) $\forall R_x \in MDC(p)$, $R_x \not\subseteq R'$. We prove by contradiction. Suppose p is not in the skyline with respect to R. By Theorem 1, p is not in the skyline with respect to a refinement R' of R. Suppose there exists $R_x \in MDC(p)$ such that $R_x \subseteq R'$. Since $R_x \in MDC(p)$, by Definition 3, p is not a skyline point with respect to $R \cup R_x$. That is, with respect to $R \cup R_x$, there exists another data point q that dominates p such that for any dimension $D_i \in \mathbb{S}$, $q.D_i \preceq p.D_i$, and there exists a dimension $D_{i_0} \in \mathbb{S}$, where $q.D_{i_0} \prec p.D_{i_0}$. Since $R_x \subseteq R'$, p is not a skyline point with respect to $R \cup R'$.

Second, we will show that, for a point p and a template order R, if 1) p is in the skyline with respect to R and 2) $\forall R_x \in MDC(p)$, $R_x \not\subseteq R'$, then p is in the skyline with respect to a refinement R' of R. We prove by contradiction. Suppose p is not in the skyline with respect to a refinement R' of R. There exists a set Y of data points that dominate p with respect to R'. That is, for each point $q \in Y$, with respect to R', for any dimension $D_i \in \mathbb{S}$, $q.D_i \preceq p.D_i$, and there exists a dimension $D_{i_0} \in \mathbb{S}$, where $q.D_{i_0} \prec p.D_{i_0}$. Let $R_q = \{(q.D_i, p.D_i) | (q.D_i, p.D_i) \notin R \text{ for all } i\}$. It is noted that $R_q \subseteq R'$. Thus, by Definition 3, there exists a data point $q \in Y$ such that $R_q \in MDC(p)$ and $R_q \subseteq R'$. This leads to a contradiction. □

3 The MDC Approach

The MDCs can be used to answer skyline viewpoint queries and skyline queries. According to Theorem 1, we only need to consider the skyline points with respect to the template order. Only those points can be in the skylines of the refinements.

If for each skyline point with respect to the template order, we can compute its MDCs, then we can use the conditions to answer viewpoint queries and refined skyline queries.

To answer a viewpoint query for point p, we can use the MDCs for p directly according to Theorem 2.

To answer a refined skyline query about the skyline with respect to an order R' that is a refinement of the template R, we can check R' against the MDCs of each skyline point p with respect to R. If R' is stronger than (i.e., contains) one of the MDCs of p, then p cannot be in the skyline with respect to R'. Otherwise, p is a skyline point with respect to R'. The query answering cost is $O(|SKY(R)| \cdot l)$, where l is the maximum number of MDCs of a skyline point.

Now, the problem becomes how to compute the MDCs for a skyline point.

For two skyline points p and q with respect to the template order $R = (R_1, \ldots, R_m)$, two cases may arise.

Case 1. There is an attribute D_{i_0} such that $p \prec_{D_{i_0}} q$. Then, q never dominates p in any refinement of R and thus cannot lead to an MDC for p. For example, in Table 1, c never dominates a because the price value of a is smaller than that of c.

Case 2. There does not exist any attribute D_{i_0} such that $p \prec_{D_{i_0}} q$. That is, p never dominates q on any dimension. Then, q may dominate p with respect to some refinements of R and thus may lead to an MDC of p. Formally, q **quasi-dominates** p if $(p.D_i, q.D_i) \notin R_i$ for $1 \leq i \leq m$. For example, in Table 1, a quasi-dominates f because a has a lower price and a higher hotel class than f. The only reason that a does not dominate e in the template is that T does not dominate M. As shown in Example 4, in a refinement $R' = \{(T, M)\}$ of R, a dominates f. Thus, R' is an MDC for f.

Given a template order $R = (R_1, \ldots, R_m)$, if q quasi-dominates p, the **minimum condition** that q dominates p is $R^{q \to p} = \{(q.D_i, p.D_i) | q.D_i \neq p.D_i \text{ and } (q.D_i,\ p.D_i) \notin R_i \text{ for } 1 \leq i \leq m\}$. It is easy to see that q dominates p in $(R \cup R^{q \to p})$. $R^{q \to p}$ strengthens the template in a minimal way such that q dominates p.

Based on the above idea, we have the simple MDC algorithm, as shown in Algorithm 1, to compute the MDCs as follows: Let R be the template order. First of all, we compute the skyline $SKY(R)$. Let $SKY(R) = \{p_1, \ldots, p_k\}$. Then, we initialize $MDC(p_i)$ to $\emptyset$ for $1 \leq i \leq k$. For each pair of p_i and p_j, where $1 \leq i, j \leq k$, and $i \neq j$, such that p_i quasi-dominates p_j, if $MDC(p_j)$ does not have a condition weaker than $R^{p_i \to p_j}$, then we insert $R^{p_i \to p_j}$ into $MDC(p_j)$ and remove any stronger conditions in $MDC(p_j)$.

TABLE 2
A Set of Hotels, Having Two Nominal Attributes

Package ID	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)

Algorithm 1. Algorithm **MDC**
Input: data set $\mathcal{D}$ and order template R
Output: the set of skyline points with respect to R and their MDCs

```
1: compute the skyline with respect to $R$, suppose the
   skyline is $\{p_1, \ldots, p_k\}$
2: set $MDC(p_i) = \emptyset$ for $(1 \leq i \leq k)$
3: for $i := 1$ to $(k-1)$ do
4:   for $j := (i+1)$ to $k$ do
5:     if $p_i$ quasi-dominates $p_j$ then
6:       if $MDC(p_j)$ does not have a condition weaker than
         $R^{p_i \to p_j}$ then
7:         add $R^{p_i \to p_j}$ to $MDC(p_j)$ and remove any stronger
           conditions in $MDC(p_j)$
8:       end if
9:     end if
10:    if $p_j$ quasi-dominates $p_i$ then
11:      if $MDC(p_i)$ does not have a condition weaker than
         $R^{p_j \to p_i}$ then
12:        add $R^{p_j \to p_i}$ to $MDC(p_i)$ and remove any stronger
           conditions in $MDC(p_i)$
13:      end if
14:    end if
15:  end for
16: end for
```

Example 5 (the MDC algorithm). Let us illustrate Algorithm 1 using Table 2, which is an extension of Table 1 by adding one more nominal attribute, airline. Suppose the template R on hotel group and airline is set to $\emptyset$.

In the first step, we compute the skyline with respect to R, which is $\{a, c, d, e, f\}$. We can remove point b because b is not in the skyline with respect to R.

Then, we check whether any skyline point quasi-dominates another. We compare a and c first, which do not quasi-dominate each other. Then, we compare a and d. a quasi-dominates d. The minimum condition that a dominates d $R^{a \to d} = \{(T, H), (G, R)\}$. Similarly, we compare a and e and obtain $R^{a \to e} = \{(T, M), (G, R)\}$. We compare a and f and obtain $R^{a \to f} = \{(T, M), (G, W)\}$. We obtain the MDC of each data point, as shown in Table 3.

Next, we compare c with other points. First, we compare c and d. c quasi-dominates d. The minimum condition that c dominates d $R^{c \to d} = \{(G, R)\}$. As shown in Table 3, the MDC of $d = \{(T, H), (G, R)\}$, which is stronger than $\{(G, R)\}$. Thus, we insert $\{(G, R)\}$ and remove $\{(T, H), (G, R)\}$. Similarly, we compare c with e and f. We obtain the MDCs, as shown in Table 4, after

TABLE 3
MDCs of Points after Comparing a to Other Points

Package	MDC
a	-
c	-
d	{(T, H), (G, R)}
e	{(T, M), (G, R)}
f	{(T, M), (G, W)}

processing c. Similarly, we process the remaining iterations. It can be verified that the final MDCs are as shown in Table 4.

The algorithm consists of two parts. The first part is computing the skyline points with respect to R. One can adopt any existing algorithm (e.g., [7] and [6]) for computing the skyline for partially ordered domains. Let us analyze the complexity of the second part of our algorithm. It takes nested loops to compare each pair of skyline points, p_i and p_j. There are $O(k^2)$ possible pairs, where k is the number of skyline points with respect to R. For each p_i quasi-dominating p_j, the minimum condition is computed with cost $O(m')$, where m' is the number of nominal dimensions. The minimum conditions are inserted into the set of MDCs if the current set does not have a weaker condition. The insertion takes cost $O(l)$, where l is the maximum number of MDCs of a skyline point. Thus, the overall complexity is $O((m' + l)k^2)$.

4 The CST Approach

As strongly indicated in the previous studies on data warehousing and online analytical processing, an effective approach to online query answering is to materialize the answers in a compact and well-organized way. In this section, we study how to store the skylines with respect to various refinement orders in a compact data structure.

A straightforward approach is to store the refinement order lattice explicitly, as shown in Fig. 1. At each node of the lattice, we also store the corresponding skyline. However, as can be observed in the figure, there is substantial redundancy in the lattice: multiple orders in the lattice may share the same skyline, and a point may appear in the skylines of multiple orders. Removing the redundancy not only saves space but also speeds up the computation.

To keep our discussion easy to follow, let us first consider the case where there is only one nominal dimension. We will consider the general case where there are multiple partially ordered dimensions later in Section 4.4.

TABLE 4
MDCs of Points after Comparing c to Other Points

Package	MDC
a	-
c	-
d	{(G, R)}
e	{(T, M), (G, R)}
f	{(T, M), (G, W)}, {(H, M), (G, W)}

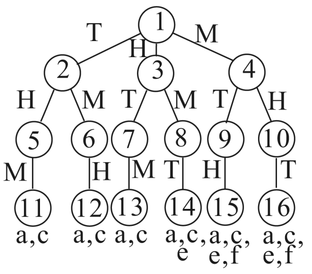


Fig. 2. An OST.

Without loss of generality, let D_m be the partially ordered nominal attribute. In the following discussion, the template order on D_m is set to $R_m = \emptyset$. Our discussion can be easily extended to the general case where $R_m \neq \emptyset$.

4.1 OST and CST

Fig. 2 shows an **ordered skyline tree** (OST) of Table 1. In an OST, each edge is associated with a value in D_m, and each leaf node corresponds to a total order. If the path from the root to a leaf node N is $C_{i_1} - \cdots - C_{i_l}$, then the order corresponding to N is $C_{i_1} \prec \cdots \prec C_{i_l}$.

An OST only stores the skyline with respect to every total order (at the leaf nodes) on D_m. It is compact since it does not store the skyline with respect to any partial order on D_m. Based on an OST, one can find the skyline with respect to any partial order on D_m, and an efficient algorithm will be presented in Section 4.3.

There is still redundancy in an OST. For example, a and c appear in all leaf nodes in Fig. 2, since a and c are in the skylines with respect to any total orders. This redundancy can be reduced by collecting the common skyline points from leaf nodes and storing them in the shared internal nodes. For example, a and c can be moved from all leaf nodes to the root node (node 1). After the adjustment, if a node in the OST contains no skyline points and no child node, it even can be removed. Fig. 3 shows the resulting CST.

4.2 CST Construction

Let us consider how to construct a CST efficiently. First of all, we compute all skyline points with respect to the template order R. One can adopt any existing algorithm (e.g., [7] and [6]) for computing the skyline for partially ordered domains. In our running example, if $R_m = \emptyset$, then these skyline points are a, c, e, and f. According to Theorem 1, only those points can be in the skylines with respect to refinements of the template. The set of skyline points with respect to R can be further divided into two subsets: the **global skyline point set** $\mathcal{G}$ and the **order-sensitive skyline point set** $\mathcal{D}'$. A point p is in the set $\mathcal{G}$ if p is

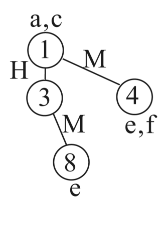


Fig. 3. A CST.

in the skyline of subspace $D_1 \times \cdots \times D_{m-1}$, i.e., the subspace of the totally preordered attributes, *and there exists no other point p' taking the same values as p on all those totally preordered attributes but different values from p on D_m*. The other skyline points are in the set $\mathcal{D}'$. For example, in Table 1, the global skyline set $\mathcal{G} = \{a, c\}$, and the order-sensitive skyline set $\mathcal{D}' = \{e, f\}$.

It is easy to see that a point p in $\mathcal{G}$ cannot be dominated in any refinement order. Thus, the points in $\mathcal{G}$ are in the skylines with respect to any refinement orders. $\mathcal{G}$ should be stored at the root node of the CST.

Next, the descendants of the root node in the CST are generated in a *breadth-first* manner. In particular, we need to find the skyline points with respect to different refinements of the template from the order-sensitive skyline set $\mathcal{D}'$. Thus, each node is associated with a *node skyline set*.

4.2.1 Node Skyline Set

As shown in Fig. 3, each node N in the CST is associated with a **node skyline set** $S(N)$. For clarity, if the context of the node N is clear, we simply write S. A point x appears in $S(N)$ in CST if and only if *x is a skyline point in all the leaf descendants of N in the (correspondence) OST* and *x is not a skyline point in at least one leaf descendant of a sibling of N in OST.* In Fig. 3, let N be node 4. $S(N) = \{e, f\}$ in CST because e and f are skyline points in all the leaf descendants of node 4 in the (correspondence) OST, and for a sibling of the node (e.g., node 2), some of the leaf descendants (e.g., node 11) contain neither e nor f.

For a nonroot node N in CST, let $v(N)$ be the value associated to the edge between node N and the parent node of N in CST. Let $R(N)$ be the node order of N in CST, which is defined as follows: Suppose the path from the root to a node N is $C_{i_1} - \cdots - C_{i_k}$. Let $\mathcal{C} = \{C_{i_1}, \ldots, C_{i_k}\}$. We define the order at node N as the total order on $\mathcal{C}$, $C_{i_1} \prec \cdots \prec C_{i_k}$, and the partial order on D_m, $C_{i_x} \prec C_j$, where $x \in [1, k]$, and $C_j \notin \mathcal{C}$. For example, node 8 in the figure represents the total order on $\mathcal{C}$, $H \prec M$, and the partial orders $H \prec T$ and $M \prec T$. Besides, each child node of the root represents partial orders. For example, node 4 represents $M \prec T$ and $M \prec H$. The following lemma will be useful in our algorithm.

Lemma 1. *Let N be a nonroot node in a CST. Then, for each point $p \in S(N)$, $p.D_m = v(N)$.*

Proof. Assume that on the contrary, a point x stored in $S(N)$ does not have a D_m value $x.D_m$ equal to $v(N)$. By definition, x is a skyline point of all leaf descendants of N in OST, and it is not a skyline point of at least one leaf descendant L' of one sibling N' of N. Since x is not a skyline point of L', the value of $x.D_m$ must appear along the path from the root to L', where the ordering for $x.D_m$ determines that there exists a point q that dominates x. Since x is a skyline point for all descendants of N, the value of $v(N')$ of any node N'' between the root of OST to N must not be $x.D_m$ since otherwise, q will dominate x at all descendants of N''. Therefore, in some leaf descendant L of N, the value of $v(L)$ must also be $x.D_m$ such that at L, x will also be dominated by the same point q. Hence, x cannot be a skyline point of all leaf descendants of N, and we arrive at a contradiction. □

For example, in Fig. 3, let N be node 4. $v(N) = Mozilla$. $S(N)$ contains e and f, which have value $Mozilla$.

4.2.2 Algorithm

At the beginning of the breadth-first construction of CST, for each child N of the root, we are given a set P of candidates $= \mathcal{D}'$. By Lemma 1, we only need to check whether any candidate with a D_m value equal to $v(N)$ is a skyline point with respect to the order at N. If so, it is inserted into $S(N)$. Then, all candidates with D_m values equal to $v(N)$ are removed from P. The remaining data points in P will be considered as the candidate data points in all the descendants of node N. If P becomes empty, no descendant of N needs to be generated.

The growth of the CST tree continues in a breadth-first manner until no more new descendant can be generated. This is followed by a compression process that repeatedly removes any leaf node with an empty node skyline set. At the end, all leaf nodes in the tree contain a nonempty node skyline set, and it is the resulting CST tree.

At each node N, the node skyline set $S(N)$ is determined from the candidate set. The following lemma, which follows from the monotonic property in Theorem 1, suggests how we compute it. Let $\mathcal{U}(N)$ be the set of data points in the node skyline sets of all ancestors of N. $\mathcal{U}(N)$ can be found easily by traversing from N to the root.

Lemma 2. *If a data point x in the candidate set P of N has a D_m value of $v(N)$ and is not dominated by any data points in $\mathcal{U}(N)$ (with respect to $R(N)$), then x is a skyline point with respect to $R(N)$.*

A point x that satisfies the condition in the above lemma is inserted into $S(N)$. The mechanisms to find $S(N)$ given $\mathcal{U}(N)$ will be described in Section 4.2.3.

The following corollary, which follows from Theorem 1, can help to further trim down the CST construction cost.

Corollary 1. *Let N be a node. Suppose x is a data point in the candidate set P of N with a D_m value equal to $v(N)$. If $x \notin S(N)$, then x is not a skyline point in all leaf descendants of all siblings of N in the OST.*

The corollary follows because in any descendant N' of all siblings of N, if $v(N') = v(N)$, then the order corresponding to N' is a refinement of the partial order corresponding to N. In our algorithm, as shown in Algorithm 2, after processing all the children of the same parent, any points x found to be a nonskyline point in any child node is deleted from the candidate sets of the children (lines 15 and 16 in Algorithm 2).

Algorithm 2 shows the algorithm for constructing CST. Typically, only a small subset of the nodes in OST is explored. In the running example, only nine nodes are explored, out of 16 nodes in the OST.

Algorithm 2. Algorithm for constructing CST
```
1: // Expansion
2: $S$ of root $\leftarrow \mathcal{G}$, where $\mathcal{G}$ is the global skyline set
3: $P$ of root $\leftarrow \mathcal{D}'$, where $\mathcal{D}'$ is the order-sensitive skyline
   set
4: for each level $l$ of OST $\mathcal{T}$ do
```

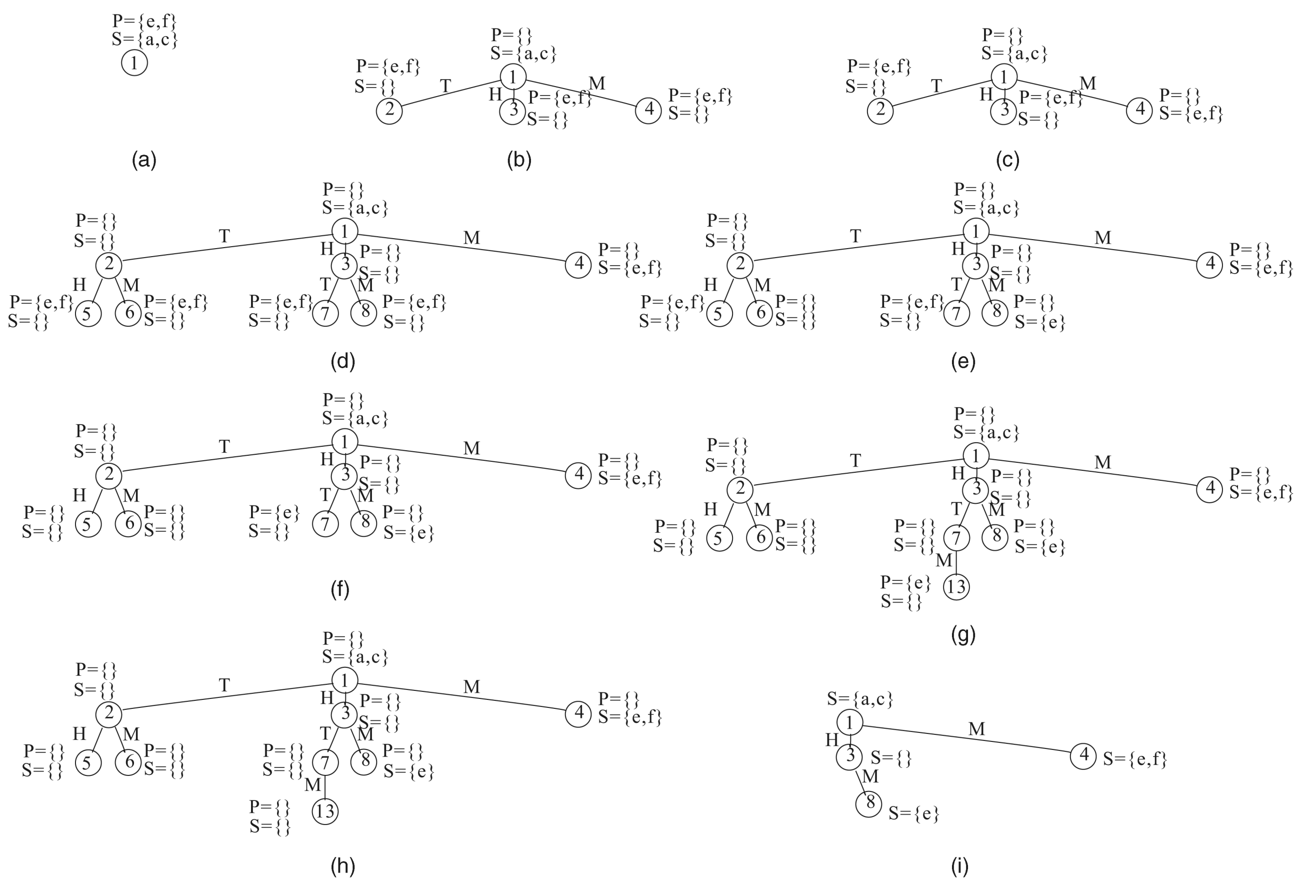


Fig. 4. CST construction (one nominal attribute).

```
5:     for each node $N_{par}$ in level $l$ do
6:       let $P_{par}$ be the candidate set $P$ of node $N_{par}$
7:       // $\overline{S}_{sib}$ will store all nonskyline points in all
         siblings $N_i$
8:       $\overline{S}_{sib} \leftarrow \emptyset$
9:       for each child node $N_i$ of node $N_{par}$ with respect to
         the template order $R$ do
10:        clone all data points in $P_{par}$ to the candidate set $P$
           of node $N_i$
11:        find the node skyline set $S$ and the node
           nonskyline set $\overline{S}$ of node $N_i$ from the data points in
           $P$ with $D_m$ value equal to $v(N_i)$
12:        $\overline{S}_{sib} \leftarrow \overline{S}_{sib} \cup \overline{S}$
13:        discard $\overline{S}$ from $P$
14:      end for
15:      $P_{par} \leftarrow \emptyset$
16:      for each child node $N_i$ of node $N_{par}$ do
17:        remove all data points in $\overline{S}_{sib}$ from $P$ of $N_i$
18:      end for
19:    end for
20:  end for
21:  // Compression
22:  while there is a leaf node $L$ with an empty node skyline
     set $S$ do
23:    remove $L$
24:  end while
```

Example 6. Let us illustrate the algorithm with the example in Table 1. Let the template $R = \emptyset$. Let $P(N)$ be the candidate set P of a node N. The global skyline set $\mathcal{G} = \{a, c\}$, and the order-sensitive skyline set $\mathcal{D}' = \{e, f\}$. Hence, P(root) $= \{e, f\}$, and S(root) $= \{a, c\}$, as shown in Fig. 4a. Then, we expand the root node by cloning P in the root node to P in the children (i.e., node 2, node 3, and node 4), as shown in Fig. 4b. After that, $P(rootnode)$ is cleared. For each child node N, we process the data points with a hotel-group value equal to $v(N)$. Since e and f have a hotel-group value of "M," they are processed in node 4. They are found to be skyline points with respect to the order of node 4. Thus, $S(node4) = \{e, f\}$. The two points are removed from $P(node4)$. The result is shown in Fig. 4c.

Moving one level downward, P of node 2 is migrated to the P sets of node 5 and node 6. Similarly, we generate child nodes 7 and 8 for node 3. The P of node 4 is empty and needs no processing. The result of this step is shown in Fig. 4d. At this level, only node 6 and node 8 are processed. e and f are not skyline points with respect to the order of node 6 since a dominates both e and f with respect to the order $T \prec M$. Hence, $S(node6) = \{\}$. e is a skyline point, and f is not with respect to the order of node 8. $S(node8)$ becomes $\{e\}$. The updated tree is shown in Fig. 4e. Since e and f are not skyline points in node 6, by Corollary 1, e and f are not skyline points in any leaf descendants of its siblings (i.e., any descendants of node 5). Thus, e and f are removed from $P(node5)$. Similarly, f is not a skyline point in node 8. By Corollary 1,

f can be removed from $P(node7)$. The resulting tree is shown in Fig. 4f.

Similarly, node 13 is generated as shown in Fig. 4g. e is not a skyline point with respect to the order of node 13, and the tree in Fig. 4h is obtained. After the compression step, the final CST is shown in Fig. 4i.

The following lemma gives a bound on the overhead in node exploration during the construction of the CST.

Lemma 3. *In the CST construction algorithm, for the parent N_p of any leaf node in the final CST, no node N beyond the grandchildren of N_p in the (correspondence) OST will be explored by Algorithm 2.*

Proof. Consider a data point x in the candidate set P of a nonroot node N in the OST, where N does not appear in the (correspondence) final CST. By Corollary 1, if x is still in the candidate set P of N, then x is a skyline point of N', where N' is a sibling node of the parent of N. This is because if x is not a skyline point of any sibling of the parent node, then it will be pruned from the candidate set P. Hence, the grandparent of N is in the OST and also the correspondence CST. Thus, N is not beyond the grandchildren of N_p in the OST, where N_p is the parent of a leaf node in the final CST. □

4.2.3 Implementation Details

In Algorithm 2, we need to compute the node skyline set $S(N)$ given $\mathcal{U}(N)$. We propose the following implementation. We call this approach **NomSky-Scan**. To decide if x is dominated by any point in $\mathcal{U}(N)$, we scan all data points in $\mathcal{U}(N)$ and compare with x. Let n_P be the number of data points x with values $x.D_m = v(N)$ in the candidate set P of N. The runtime of this approach is $O(n_P|\mathcal{U}(N)|)$. Let n_{SKY} be the greatest number of skyline points with respect to any total order on D_m. Since $|\mathcal{U}(N)| = O(n_{SKY})$, the runtime is $O(n_P n_{SKY})$.

4.3 Query Evaluation

CST supports both skyline queries and viewpoint queries.

4.3.1 Skyline Queries

Skyline queries aim to find all skyline points with respect to a given refinement R' of the template R. We propose an algorithm here with two main steps. The first step is to *extract all possible maximal linear orders* in R'. A total order R'' is a maximal linear order in R' if 1) R'' is a total order in R' and 2) no superset of R'' is also a total order in R'. If R' is a partial order, then there can be multiple maximal linear orders. For example, if $R' = \{(T,H),(H,G),(T,M)\}$, there are two maximal linear orders $\{(T,H),(H,G)\}$ and $\{(T,M)\}$.

For each maximal linear order $R'' : C_{i_1} \prec C_{i_2} \prec \ldots \prec C_{i_k}$, we find the *skyline points with respect to* R''. Let $\mathcal{C}$ be $\{C_{i_1}, C_{i_2}, \ldots, C_{i_k}\}$. First, we initialize a variable, $X_{R''}$, storing the skyline with respect to R'', with $S(N_{root})$, where N_{root} is the root of the CST. Then, we perform two steps:

Step 1. Let $\mathcal{P}$ be the path $C_{i_1} - C_{i_2} - \ldots - C_{i_k}$ in the CST. For each node N along $\mathcal{P}$, if N exists, then add $S(N)$ to $X_{R''}$ ($X_{R''} \leftarrow X_{R''} \cup S(N)$). For example, if $R'' = (T,H)$, then, from Fig. 4i, we find only points a and c since there is no path $T - H$ and the node skyline points of the root node are a and c.

Step 2. Next, we find the additional skyline points with values that do not appear in $\mathcal{C}$ with respect to R''. Such points are not dominated by others, and they do not dominate others. We will find these additional skyline points as follows: For each value $C_j \notin \mathcal{C}$, we look for any child node N of the root of CST with $v(N) = C_j$. The points in $S(N)$ are added to $X_{R''}$. For example, if $R'' = (T,H)$, from Fig. 4i, we find additional data points (i.e., e and f) from the points stored in node 4 with a D_m value of $M \notin \{T,H\}$.

The above process is repeated with other possible maximal linear orders. The intersection of the resulting sets of skyline points is the required answer since the result should satisfy all maximal linear orders simultaneously. For example, if the refinement $R' = \{(T,H),(T,M)\}$, there are two maximal orders $\{(T,H)\}$ and $\{(T,M)\}$. For $\{(T,H)\}$, we find a, c, e, and f. Similarly, for $\{(T,M)\}$, we find a and c. The intersection of these two sets is a and c, which is the final skyline points for order R.

The complexity of the algorithm depends on the number of maximal linear orders and the sizes of their skyline sets, which are typically quite small compared to the data size. We shall show in our experiments that the querying evaluation is very efficient.

4.3.2 Viewpoint Queries

A viewpoint query is to find all the refinements that are skyline viewpoints for a given data point x. Here, we discuss how to find the results from a CST. Let q be the number of different values in D_m. Given a node N in the CST associated with a path from the root $C_{i_1} - C_{i_2} - \ldots - C_{i_k}$, where $k \leq q$, node N is associated with a unique partial order $R(N)$. We call a set $BO(N)$ of binary orders on the attribute values a *canonical binary order set* if the transitive closure of $BO(N)$, denoted by $BO^+(N)$, is equivalent to $R(N)$ and no subset of $BO(N)$ has this equivalence property. $BO^+(N)$ can be easily computed by repeatedly extending the set $BO(N)$ with any binary order that results from the transitivity of two binary orders in the set. For instance, let node 11 be N. $BO(N) = \{(T,H),(H,M)\}$. By the transitivity property of binary conditions in $BO(N)$, $BO^+(N) = \{(T,H),(T,M),\ (H,M)\}$, which is equivalent to $R(N)$. The use of the canonical order set is to minimize the overhead in listing the relationships in an order. Let $\mathcal{C} = \{C_{i_1}, C_{i_2}, \ldots, C_{i_k}\}$. Given node N, we compute the *canonical binary orders set* by the following steps:

1. Generate binary orders among the values in $\mathcal{C}$: for each $j \in [1, k-1]$, generate $(C_{i_j}, C_{i_{j+1}})$.
2. Generate the binary orders between C_{i_k} and the values in $D_m - \mathcal{C}$. Let $D_m - \mathcal{C}$ be $\{C_{i_{k+1}}, C_{i_{k+2}}, \ldots, C_{i_q}\}$, for each $j \in [k+1, q]$, generate (C_{i_k}, C_{i_j}).

Let the set of canonical binary orders of node N be $BO(N)$. Consider node 4 in our running example. We obtain (M,T) and (M,H) in the second step. That is, $BO(N_4) = \{(M,T),(M,H)\}$. Similarly, $BO(N_8) = \{(H,M),(M,T)\}$.

Let $\mathcal{N}$ be the set of nodes storing a data point x. The main observation here is that suppose a data point x is in the node skyline sets S's of nodes $\in \mathcal{N}$ in CST, then any refinement that satisfies all the binary conditions in $BO(N)$ of any node $N \in \mathcal{N}$ is a skyline viewpoint for x.

Other than the above viewpoints, we must also include any refinement that does not contain any relationship

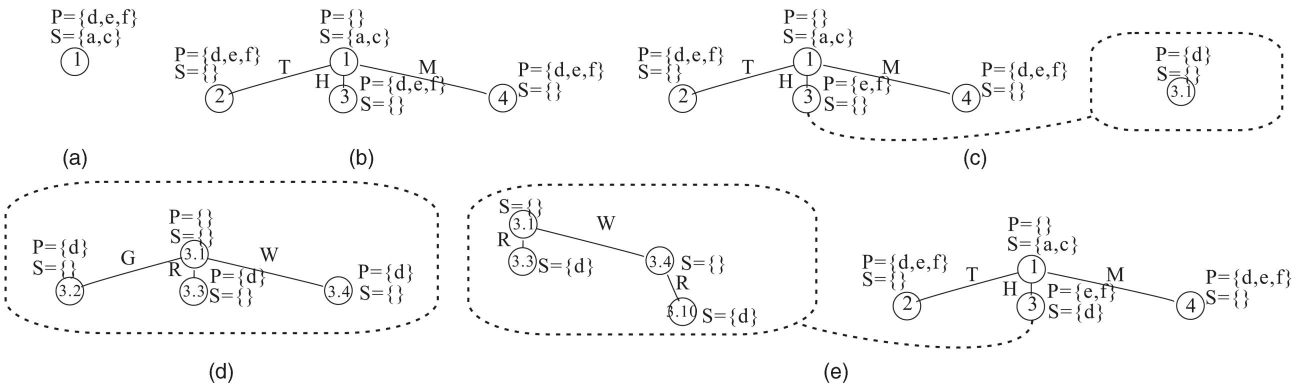


Fig. 5. CST construction (two nominal attributes).

involving $x.D_m$. In such a refinement order, x is not dominated by any point since there is no value in D_m that dominates $x.D_m$. We define a *special order condition* $\{\neg x.D_m\}$, which corresponds to this kind of order. Since there are two possible scenarios that x is a skyline point, we conclude that a skyline viewpoint for x must satisfy either all the binary orders of at least one of the $BO(N)$ or the special order condition. The answer to the viewpoint query is given by the disjunction of $\vee_{N\in\mathcal{N}} BO^+(N)$ and the special order $\{\neg x.D_m\}$. Hence, for a refinement to be a skyline viewpoint for e, the necessary and sufficient condition is given by $\{(M,T),(M,H)\}^+ \vee \{(H,M),(M,T)\}^+ \vee \{\neg M\}$. Note that in the real implementation, we do not need to compute $BO^+(N)$. We just keep $BO(N)$ because if a refinement R' satisfies all the binary conditions in $BO(N)$, R' is already a viewpoint of a data point x.

Let $\mathcal{N}$ be the set of nodes in CST containing a data point x in their node skyline sets. With the implementation of a header table in which each data point x is linked with a list of nodes containing x in the node skyline set, the search for $\mathcal{N}$ requires $O(|\mathcal{N}|)$ time. The computation of a canonical binary order set takes $O(q)$ time. Thus, the overall runtime is $O(|\mathcal{N}|q)$.

4.4 Multiple Partially Ordered Attributes

So far, we have assumed that there is only one nominal attribute in the data set. Now, we relax the assumption and allow multiple nominal attributes. We will discuss the required modifications in the algorithm with an example in Table 2. Fig. 5 illustrates some major steps in the construction of the corresponding CST.

The resulting CST consists of a primary tree based on one nominal attribute, and in the primary tree, each node can be linked to a nested tree that is based on a second nominal attribute. The number of nominal attributes determines the maximum possible levels of nesting. In our running example, a nested tree is created for node 3. Based on hotel group, we find a set X of data points with a hotel-group value equal to $Horizon$ in the candidate set P of node 3 (i.e., d). Then, we build a *nested tree* based on airline as shown in the dotted region in Figs. 5c, 5d, and 5e. We can build a nested tree by first setting P of the root node (i.e., node 3.1) by the data points in X. After that, we perform a similar construction step in the nested tree by expanding the children with values G, R, and W. Each node N in the nested tree has a corresponding order $R(N)$. For example, for the root node (node 3.1), the order is given by $\{(H,M),(H,T)\}$, which is the same as $R(node3)$, while $R(node3.4) = \{(W,R),(W,G),(H,M),(H,T)\}$. We extend the meaning of $v(N)$ for node N in the nested tree to be a set of values for the set of nominal attributes. For example, $v(node3) = \{Horizon, nil\}$, and $v(node3.4) = \{Horizon, Wings\}$. Then, we process the nodes and find the skyline points according to the correspondence orders.

In the nested tree for a node N, we also perform the compression step at the end to repeatedly remove any leaf node with an empty node skyline set. After constructing the nested tree, we insert all skyline points in the nested tree (e.g., d) into the node skyline set $S(N)$ in node N (e.g., node 3). This insertion does not affect the process of finding the node skyline set in a descendant node. For example, when we process each descendant node N' of node 3, $\mathcal{U}(N')$ also contains d. Each node N'' in the nested tree based on airline (expanded from N') is processed with respect to $\mathcal{U}(N')$, which also contains d. In this process, we consider whether a data point x is a skyline point with respect to $R(N')$; even if d is not a skyline point with respect to $R(N')$, this does not affect the skyline membership of x.

Lemma 4. *If a data point x in the candidate set P of a node N in CST has a nominal value set of $v(N)$ and is not dominated by any data points in $\mathcal{U}(N)$ (with respect to $R(N)$), then x is a skyline point with respect to $R(N)$.*

Example 7. Let us illustrate the algorithm with the example in Table 2. Let the template $R = \emptyset$. We know that the global skyline set $\mathcal{S} = \{a,c\}$ and the order-sensitive skyline set $\mathcal{D}' = \{d,e,f\}$. We set P and S of the root node to $\{d,e,f\}$ and $\{a,c\}$, respectively, as shown in Fig. 5a. Then, according to attribute hotel group, we expand the root node by cloning all data points of P in the root node to P in the children (i.e., node 2, node 3, and node 4), as shown in Fig. 5b; P is cleared in the root node. For each child node N, we construct a nested CST based on value $v(N)$. For example, node 3 has the node value equal to "H." Data d matches this value. We remove d from the P of node 3 and assign it to P of the root node of the nested CST (i.e., node 3.1 in Fig. 5c). The nested CST is expanded and then compressed according to attribute airline, as shown in Figs. 5d and 5e. The construction steps of the other nodes are similar to that of node 3.

The extension of both skyline queries and viewpoint queries is straightforward: 1) In the skyline queries, suppose the refinement contains the order on hotel group and the order on airline, whenever the algorithm visits a node in the CST based on hotel group (which corresponds to the order on hotel group), it traverses the "nested" tree and visits the nodes in the "nested" tree (which correspond to the order on airline). Similarly, the complexity of the skyline queries is dependent on the number of maximal linear orders and the sizes of their skyline sets. The runtime is $O(x^{m'} \times |S|)$, where x is the number of maximal linear orders for an attribute, m' is the number of nominal attributes, and $|S|$ is the average size of their skyline sets, with the bitwise implementation of the intersection of two sets. Similarly, x and $|S|$ are typically small, and thus, the query time is quite short, which can be found in our experiments. 2) In the viewpoint queries of a data point x, we also find all nodes that contain x in their node skyline sets and determine the viewpoints accordingly. Similarly, the search for $\mathcal{N}$ requires $O(|\mathcal{N}|)$ time, where $\mathcal{N}$ is the set of nodes in CST containing a data point x in their node skyline sets. Here, the runtime of computing a canonical binary order set is $O(q^{m'})$, where q is the number of possible values in a nominal attribute, and m' is the number of nominal attributes. The overall runtime is thus $O(|\mathcal{N}|q^{m'})$.

5 Empirical Study

We have conducted our experiments on a Pentium IV 3.2-GHz PC with 2-Gbyte 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 [5]. As this data set generates only numeric attributes, we have modified the program to generate both numeric attributes and nominal attributes. The numeric attributes are generated as in [5]. The nominal attributes are generated according to a Zipfian distribution [15]. By default, we set the Zipfian parameter $\theta = 1$. We generated 500,000 tuples with three numeric dimensions and one nominal dimension. The total number of dimensions is equal to the number of numeric dimensions plus the number of nominal dimensions. The number of classes in a nominal dimension was set to five. We adopted a template where the most frequent class in a nominal dimension has a higher priority than each of the other classes.

We denote the MDC materialization approach by *MDC.* In the CST method, Section 4.2.3 can be implemented with the NomSky-Scan approach. For the sake of comparison, based on R-tree [18], we also implement two additional approaches, namely, *NomSky-Rtree* and *NomSky-MultiRtree,* when we compute the node skyline set $S(N)$ given $\mathcal{U}(N)$. In the NomSky-Rtree approach, a *single* R-tree is built and indexed on the totally ordered attributes in $\mathcal{D}'$. The branch-and-bound (BBS) algorithm in [18] is adopted to compute $S(N)$. In NomSky-MultiTree, instead of one R-tree, we build an R-tree for each value of the nominal attributes. Similarly, the BBS algorithm can be applied here. For the R-tree-based algorithms that we have tested, each node occupies a page size of 4 Kbytes.

We compare the algorithms in terms of the execution time of MDC construction and CST construction. Also, we compare the storage sizes of the materialized MDC, OST, and CST generated by the algorithms. The execution time of two kinds of queries—*skyline query* and *viewpoint query*—- will be analyzed.

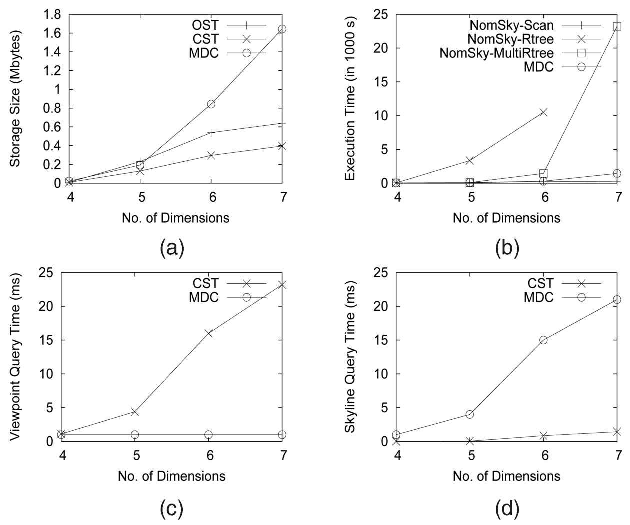


Fig. 6. Scalability with respect to dimensionality where the number of numeric attributes is fixed to three.

5.1 Synthetic Data Sets

Three types of data sets are generated as described in [5]: 1) *independent data sets*, 2) *correlated data sets*, and 3) *anticorrelated data sets.* The description of these data sets can be found in [5]. For the interest of space, we only show the experimental results for the anticorrelated data sets. The results for the independent data sets and the correlated data sets are similar, but the execution times in those data sets are much shorter.

Effect of the number of dimensions. We fixed the number of numeric attributes to be three while varying the number of nominal attributes from one to four. Fig. 6a shows that the storage sizes of MDC, OST, and CST increase with the number of nominal attributes. The increase rate of MDC is greater than the increase rate of OST/CST. This is because for each data point stored by MDC, the minimum disqualifying conditions are stored independently, while OST/CST make use of a *sharing* technique for common ancestors for different orders. CST is more compressed compared with OST when the number of nominal attributes is higher.

In Fig. 6b, the execution time of the CST construction step of NomSky-Scan is the smallest, and the execution time of the MDC construction is the second smallest, while the execution time of NomSky-Rtree is the greatest. By Corollary 1, the CST construction step can prune data points in the candidate sets P. Therefore, NomSky-Scan can take fewer data points into consideration for computing the skyline points in S of the current node. However, the R-tree-based algorithms build R-tree(s) based on all the data points in $\mathcal{D}'$ and cannot take advantage of the pruning. It is noted that the NomSky-Rtree algorithm performed slower than the NomSky-MultiRtree algorithm. This is because NomSky-Rtree mixes the data points of different values of the nominal attribute in a node. When we traverse a node in OST, the single-tree approach needs to select the data points with the node value, which slows down the performance.

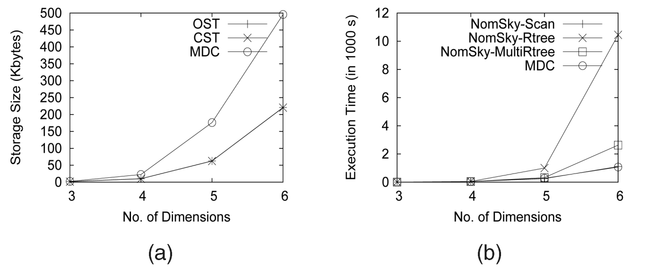


Fig. 7. Scalability with respect to dimensionality where the number of nominal attributes is fixed to one. The curves of OST and CST are similar and overlapping in (a), and the curves of MDC and NomSky-Scan are similar and overlapping in (b).

The viewpoint query time (Fig. 6c) and the skyline query time (Fig. 6d) increase with the number of nominal dimensions because when the number of nominal dimensions increases, we need to traverse more "nested" trees in CST. The viewpoint query time of MDC is faster than CST because the structure of MDC itself stores the viewpoint, but CST has to find it from the tree. The skyline query time of MDC is longer than CST because MDC has to scan all data points stored to check the skyline membership, but the structure of CST itself stores all the skyline points with respect to a query.

The effects of variations in the number of numeric attributes have also been studied, as shown in Fig. 7, where we fix the number of nominal attributes to one and vary the number of numeric attributes from two to five. As expected, both the execution time and the storage size increase with the number of numeric dimension.

Effect of the number of tuples. In this experiment, the number of tuples was varied from 250,000 to 1,000,000. Fig. 8a shows that the storage sizes of OST and CST both increase with the number of tuples. When there are more tuples in the anticorrelated data set, it is more likely that more points will become skyline data points, and the storage size increases. The construction time of CST of all algorithms increases almost linearly with the number of tuples, as shown in Fig. 8b. Again, NomSky-Scan has the fastest execution time compared with the R-tree-based algorithm and the MDC algorithm since it benefits from the pruning of the candidate sets. As the viewpoint query time and the skyline query time were very short, there was not much impact when we varied the number of tuples, as shown in Figs. 8c and 8d. Similarly, the skyline query time of MDC is larger than that of CST.

Effect of the template. We next studied the effect of different templates. Let v_c be the most frequent value in the nominal attribute. We started the experiment with a template that was an empty set. Then, one binary order was added to the template at a time. The binary order to be added was $v_c \prec v_i$, where v_i was a value in the nominal attribute, $v_i \neq n_c$, and v_i was not in the current template. For each addition, we measured the effect on the storage sizes of CST and MDC and the execution times of the CST and MDC construction. Figs. 9a and 9b show that the storage sizes and execution times decrease when the number of binary orders in the template increases. This can be explained by the fact that with more binary orders, there will be more restrictions in the node traversal. Thus, the

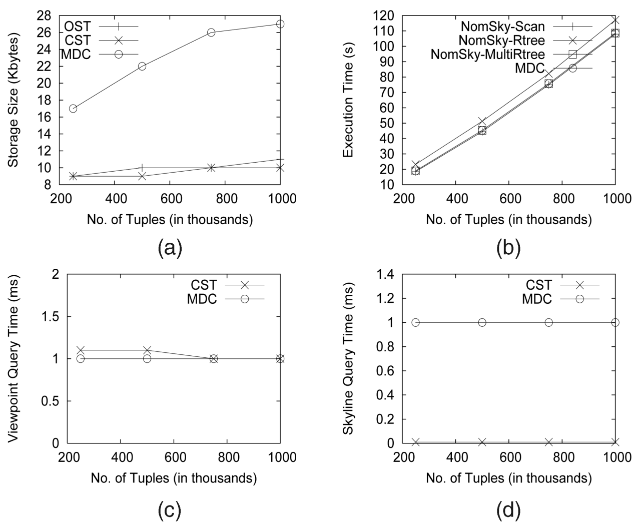


Fig. 8. Scalability with respect to database size.

number of nodes to be traversed is fewer. This reduces the storage sizes and the construction time of MDC and CST.

Let us call the above set of templates (A). We have also conducted the experiments with two other kinds of templates: (B) the templates where the least frequent value has a higher priority than the other values and (C) the templates in which the "median" frequent value has a higher priority than the other values, where the "median" frequent value is the value ranked in the middle of the values. The storage size and the compression ratio of CST both decrease from templates (C) to (B) to (A). There is a higher chance that there are more data points in the more frequent class. In other words, if we order this class at a lower priority in the template, we generate a CST with more data points in the levels near the leaf level. That means that we need to store more data points in OST and CST.

Effect of the cardinality of nominal attributes. Fig. 10 shows the results with the variation of the cardinality of a nominal attribute. In Fig. 10a, the storage size of MDC, OST, and CST increases with the cardinality. The number of permutations of different classes/values in a nominal attribute increases with the cardinality. As the number of permutations increases, it is more likely that the OST and CST contain more nodes, yielding a larger storage size. Also, the curve of MDC increases with the cardinality. But the increase rate of MDC is smaller than that of OST/CST. This is because MDC is independent of the permutation of different values in a nominal attribute. As there are more classes, the number of the minimum disqualifying

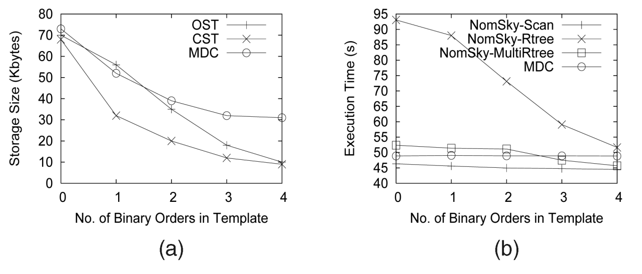


Fig. 9. Scalability with respect to template size.

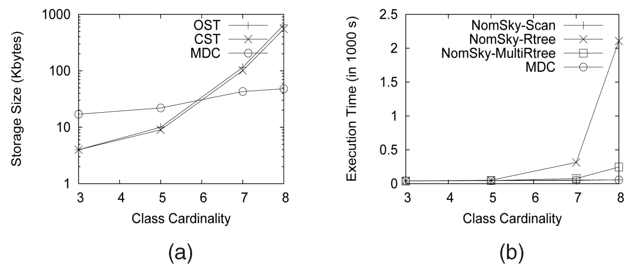


Fig. 10. Scalability with respect to the cardinality of the nominal attribute.

conditions for a data point is greater, yielding the increase of the storage of MDC. In Fig. 10b, the execution time of all algorithms increases with cardinality. The skyline query time and the viewpoint query time of CST and MDC increases with the cardinality because we have to process more values. Similar as before, the skyline query time of CST is shorter than MDC, and the viewpoint query time of MDC is shorter than CST. For example, the skyline query times of CST and MDC are less than 1 ms and about 1.6 ms, respectively, when the cardinality is equal to eight. The viewpoint query times of CST and MDC are about 3.4 ms and 1 ms, respectively, when the cardinality is equal to eight.

5.2 Real Data Sets

To demonstrate the usefulness of our methods, we ran our algorithms on two real data sets. The first set is *Nursery*, which is publicly available from the UC Irvine Machine Learning Repository.[2] *Nursery* was derived from a hierarchical decision model originally developed to rank applications for nursery schools in Ljubljana and Slovenia where the rejected applications frequently needed an objective explanation. Each tuple is an application to the nursery schools. Semantically, if an application is in the skyline, it can be considered a good candidate. Different nursery schools may have different order preferences on the nominal attributes.

In the Nursery data set, there are 12,960 instances and 8 attributes. We transformed six attributes (parents, has_nurs, housing, finance, social, and health) to numeric attributes because their values are ordered. For example, the "social" attribute has three possible values: nonproblematic, slightly problematic, and problematic, we matched them to the numbers 0, 1, and 2, respectively. The remaining two attributes are the form of the family (e.g., incomplete family and foster family) and the number of children, since there is no trivial order on their values, they would be our nominal attributes. (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 class cardinality of both nominal attributes is equal to four. We have adopted the default setting in this experiment. The results in the performance are similar to those for the synthetic data sets. The storage sizes of OST and CST are 575 Kbytes and 314 Kbytes, respectively. The compression ratio is 54.6 percent. The execution times of NomSky-Scan, NomSky-Rtree, and NomSky-MultiRtree are 94 seconds, 1,246 seconds, and 266 seconds, respectively. The skyline order query time is also similar to previous experiments.

2. http://kdd.ics.uci.edu/.

Our second real data set is *Automobile*, also adopted from the UC Irvine Machine Learning Repository. Our goal is to study the utility of viewpoint queries. We have chosen four attributes in the experiments, namely, "symboling," "normalized-losses," "price," and "make." Attribute "symboling" is the assigned numeric insurance risk rating. The smaller the value is, the safer it is. Attribute "normalized-losses" is the relative average loss payment per insured vehicle year. If the value is smaller, the loss will be smaller. Hence, the only nominal attribute is "make," representing the car brand names. The data set size was small, there were only 205 tuples, and therefore, the computation times for queries were negligibly small. We are interested to see the meaning and utilization of the skyline viewpoints for different data points. Three car brand names are chosen for our study, namely, Honda, Mitsubishi, and Toyota. We would find the disqualifying condition for three data points that belong to these three brand names, respectively:

1. Our first selected data was a Honda car that was a skyline point in some orders. The disqualifying condition we discovered is the following Make order: $Toyota \prec Honda$.
2. The second data was a Mitsubishi car, and the MDCs were the following Make orders: $Honda \prec Mitsubishi$ and $Toyota \prec Mitsubishi$.
3. Finally, we chose a Toyota car, and it gave an empty disqualifying condition. In fact, the reason was that this car model had the lowest price among all cars in the database. Therefore, it was in the global skyline set.

From the above results, a salesperson for the first car should not try to promote the car to a customer that prefers Toyota to Honda, but he may promote it to a customer who prefers Mitsubishi to Honda, since the car has some other advantage that can be attractive. The second car should be promoted to someone who prefers Mitsubishi. The third car can be promoted to any customer.

6 Related Work

Skyline queries have been studied since the 1960s in the theory field where skyline points are known as *Pareto sets* and *admissible points* [12] or *maximal vectors* [4]. However, earlier algorithms such as [4] and [3] 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 [5].

We can categorize the existing work into two major groups—*full-space skyline queries* and *subspace skyline queries*.

Many efficient methods have been proposed for full-space skyline queries that return the set of skyline points in a specific space. Some representative methods include a bitmap method [22], a nearest neighbor (NN) algorithm [16], and BBS skyline method [18].

Recently, skyline computation has been extended to subspaces. Subspace skyline queries return the skylines in all subspaces [26], [20], [21], [25], [19].

Most of the existing studies assume that only numeric attributes are present. Some recent studies [7], [6], [8], [9], [10], [2], [1] consider partially ordered attributes. For example, [7] and [6] transform each partially ordered attribute into two integer attributes. Then, the conventional skyline algorithms can be applied. Reference [8] studies the problem of estimating the cost of the skyline operator involving partially ordered attributes.

Nevertheless, most existing work assumes that *each attribute has only one order: either a total or a partial order.* 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.

Some latest works consider skylines on nominal attributes. References [9] and [10] study how to specify a query based on preferences on nominal attributes. When preferences change, the query results can be incrementally refined. In [2], a user can specify some values in nominal attributes as an equivalence class to denote the same "importance" for those values. Whenever a value v has a higher preference than a value v' in the equivalence class, v also has a higher preference than all the other values in the equivalence class. Reference [1] is an extension of [2]. In [1], whenever a user performs a query and obtains the results, if s/he finds that there are a lot of irrelevant results, s/he can modify the query by adding more conditions in the query so that the result set is smaller to suit her/his need.

Reference [13] studies how to find skyline points in a dynamic data set containing moving objects, each of which is associated with some spatial-related attributes and a non-spatial-related attributes. The dynamic skyline studied in [13] is based on spatial-related attributes that should satisfy the triangle inequality. However, the dynamic skyline studied in this paper is based on nominal attributes whose values can have any arbitrary order. The problems in [13] and here are essentially different, and the methods proposed in [13] cannot be applied here.

To summarize, there are two major differences of our work from other works. First, in this paper, since we study the refined skyline queries for *online* queries, we have to materialize the skylines of every order in a compact way so that the query result can be returned online and the storage of the materialization can be minimized. Second, we study the problem of viewpoint queries. The output is a representation for some conditions that a given data point is (or is not) a skyline point. No other researcher has addressed the problems studied here.

In the preliminary version of this paper [24], we proposed viewpoint queries. However, [24] only focuses on viewpoint queries, but this paper addresses both viewpoint queries and refined skyline queries thoroughly. Recently, [23] explored refined skyline queries with *implicit preferences*, which are a special case of the preferences studied in this paper. One example of an implicit preference is "$v_1 \prec v_2 \prec *$," which denotes that value v_1 is the most preferable, v_2 is the second most preferable, and all other values (denoted by "$*$") are less preferable than v_1 and v_2. The preferences studied in this paper are in form of any partial orders and thus are more general than implicit preferences. Most recently, [14] studied the problem of mining user preferences using skyline and nonskyline examples provided by user feedback.

7 Conclusion

We consider the problem of online skyline analysis with dynamic preferences on nominal attributes. We present a concise model of the problem, which captures the essential properties relating the changes of skylines due to the changes of orders. We also introduce the skyline viewpoint queries and propose two methods for the queries. The first method is based on the MDC. In the second approach, we propose a compact data structure CST to materialize and index the skylines with respect to various orders on the nominal attributes. We have conducted experiments to show that the proposed algorithms are effective and efficient. Future works include incremental update handling and the subspace skyline problem with dynamic preferences on the nominal attributes.

Acknowledgments

The authors are grateful to the anonymous reviewers for their constructive comments on this paper. The research of Raymond Chi-Wing Wong and Ada Wai-Chee Fu was supported in part by the RGC Earmarked Research Grant of HKSAR CUHK 4120/05E and 4118/06E. Jian Pei's research has been supported in part by an NSERC Discovery Grant and an NSERC 2008 Discovery Accelerator Supplement (DAS). 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.

References

[1] W.-T. Balke, U. Güntzer, and C. Lofi, "Eliciting Matters—-Controlling Skyline Sizes by Incremental Integration of User Preferences," *Proc. 12th Intl Conf. Database Systems for Advanced Applications (DASFAA '07),* pp. 551-562, 2007.

[2] W.-T. Balke, U. Güntzer, and W. Siberski, "Exploiting Indifference for Customization of Partial Order Skylines," *Proc. 10th Int'l Database Eng. and Applications Symp. (IDEAS '06),* pp. 80-88, 2006.

[3] J.L. Bentley, K.L. Clarkson, and D.B. Levine, "Fast Linear Expected-Time Algorithms for Computing Maxima and Convex Hulls," *Proc. First Ann. ACM-SIAM Symp. Discrete Algorithms (SODA '90),* pp. 179-187, 1990.

[4] J.L. Bentley, H.T. Kung, M. Schkolnick, and C.D. Thompson, "On the Average Number of Maxima in a Set of Vectors and Applications," *J. ACM,* vol. 25, no. 4, pp. 536-543, 1978.

[5] S. Börzsönyi, D. Kossmann, and K. Stocker, "The Skyline Operator," *Proc. 17th Int'l Conf. Data Eng. (ICDE '01),* pp. 421-430, 2001.

[6] C.Y. Chan, P.-K. Eng, and K.-L. Tan, "Efficient Processing of Skyline Queries with Partially-Ordered Domains," *Proc. 21st Int'l Conf. Data Eng. (ICDE '05),* pp. 190-191, 2005.

[7] C.Y. Chan, P.-K. Eng, and K.-L. Tan, "Stratified Computation of Skylines with Partially-Ordered Domains," *Proc. ACM SIGMOD '05,* pp. 203-214, 2005.

[8] S. Chaudhuri, N.N. Dalvi, and R. Kaushik, "Robust Cardinality and Cost Estimation for Skyline Operator," *Proc. 22nd Int'l Conf. Data Eng. (ICDE '06),* p. 64, 2006.

[9] J. Chomicki, "Iterative Modification and Incremental Evaluation of Preference Queries," *Proc. Int'l Symp. Foundations of Information and Knowledge Systems (FoIKS '06),* pp. 63-82, 2006.

[10] J. Chomicki, "Database Querying under Changing Preferences," *Annals of Math. and Artificial Intelligence,* vol. 50, nos. 1-2, pp. 79-109, 2007.

[11] J. Chomicki, P. Godfrey, J. Gryz, and D. Liang, "Skyline with Presorting," *Proc. 19th Int'l Conf. Data Eng. (ICDE '03),* pp. 717-816, 2003.

[12] O. Barndorff-Nielsen et al., "On the Distribution of the Number of Admissable Points in a Vector Random Sample," *Theory of Probability and Its Application,* vol. 11, no. 2, 1966.

[13] Z. Huang, H. Lu, B.C. Ooi, and A.K.H. Tung, "Continuous Skyline Queries for Moving Objects," *IEEE Trans. Knowledge and Data Eng.,* vol. 18, no. 12, pp. 1645-1658, Dec. 2006.

[14] B. Jiang, J. Pei, X. Lin, D.W.-L. Cheung, and J. Han, "Mining Preferences from Superior and Inferior Examples," *Proc. ACM SIGKDD,* 2008.

[15] N.L. Johnson, S. Kotz, and A.W. Kem, *Univariate Discrete Distributions.* Wiley-Interscience, 1992.

[16] D. Kossmann, F. Ramsak, and S. Rost, "Shooting Stars in the Sky: An Online Algorithm for Skyline Queries," *Proc. 28th Int'l Conf. Very Large Data Bases (VLDB '02),* pp. 275-286, 2002.

[17] D. Papadias, Y. Tao, G. Fu, and B. Seeger, "An Optimal and Progressive Algorithm for Skyline Queries," *Proc. ACM SIGMOD '03,* pp. 467-478, 2003.

[18] D. Papadias, Y. Tao, G. Fu, and B. Seeger, "Progressive Skyline Computation in Database Systems," *ACM Trans. Database Systems,* vol. 30, no. 1, pp. 41-82, 2005.

[19] J. Pei, A.W.-C. Fu, X. Lin, and H. Wang, "Computing Compressed Multidimensional Skyline Cubes Efficiently," *Proc. 23rd Int'l Conf. Data Eng. (ICDE '07),* pp. 96-105, 2007.

[20] J. Pei, W. Jin, M. Ester, and Y. Tao, "Catching the Best Views of Skyline: A Semantic Approach Based on Decisive Subspaces," *Proc. 31st Int'l Conf. Very Large Data Bases (VLDB '05),* pp. 253-264, 2005.

[21] J. Pei, Y. Yuan, X. Lin, W. Jin, M. Ester, Q. Liu, W. Wang, Y. Tao, J.X. Yu, and Q. Zhang, "Towards Multidimensional Subspace Skyline Analysis," *ACM Trans. Database Systems,* vol. 31, no. 4, pp. 1335-1381, 2006.

[22] K.-L. Tan, P.-K. Eng, and B.C. Ooi, "Efficient Progressive Skyline Computation," *Proc. 27th Int'l Conf. Very Large Data Bases (VLDB '01),* pp. 301-310, 2001.

[23] R.C.-W. Wong, A.W.-C. Fu, J. Pei, Y.S. Ho, T. Wong, and Y. Liu, "Efficient Skyline Querying with Variable User Preferences on Nominal Attributes," *Proc. 34th Int'l Conf. Very Large Data Bases (VLDB),* 2008.

[24] R.C.-W. Wong, J. Pei, A.W.-C. Fu, and K. Wang, "Mining Favorable Facets," *Proc. ACM SIGKDD '07,* pp. 804-813, 2007.

[25] T. Xia and D. Zhang, "Refreshing the Sky: The Compressed Skycube with Efficient Support for Frequent Updates," *Proc. ACM SIGMOD '06,* pp. 491-502, 2006.

[26] Y. Yuan, X. Lin, Q. Liu, W. Wang, J.X. Yu, and Q. Zhang, "Efficient Computation of the Skyline Cube," *Proc. 31st Int'l Conf. Very Large Data Bases (VLDB '05),* pp. 241-252, 2005.

Raymond Chi-Wing Wong received the BSc, MPhil, and PhD degrees in computer science and engineering from the Chinese University of Hong Kong (CUHK) in 2002, 2004, and 2008, respectively. He joined the Department of Computer Science and Engineering, Hong Kong University of Science and Technology, Kowloon, Hong Kong, as an assistant professor in 2008. From 2004 to 2005, he worked as a research and development assistant under an R&D project funded by ITF and a local industrial company called Lifewood. He received at least 18 awards. Within five years, he has published at least 19 conference papers (e.g., SIGKDD, VLDB, and ICDM) and at least eight journal/chapter papers (e.g., *Journal of Data Mining and Knowledge Discovery (DAMI)*, *IEEE Transactions on Knowledge and Data Engineering (TKDE),* and *VLDB Journal*). He reviewed papers from conferences and journals related to data mining and database, including the VLDB conference, *VLDB Journal*, *TKDE*, *ACM Transactions on Knowledge Discovery from Data*, ICDE, SIGKDD, ICDM, *DAMI*, DaWaK, PAKDD, EDBT, and the *International Journal of Data Warehousing and Mining.* He also gave presentations in international conferences such as VLDB '07, SIGKDD '07, SIGKDD '06, ICDM '05, SDM '05, PAKDD '04, and ICDM '03. His research interests include database, data mining and security. He is a member of the IEEE.

Jian Pei received the PhD degree in computing science from Simon Fraser University, Burnaby, British Columbia, Canada, in 2002. He is currently an associate professor in the School of Computing Science, Simon Fraser University. His research interests can be summarized as developing effective and efficient data analysis techniques for novel data intensive applications. Currently, he is interested in various techniques of data mining, data warehousing, online analytical processing, and database systems, as well as their applications in Web search, sensor networks, bioinformatics, privacy preservation, software engineering, and education. His research has been supported in part by the Natural Sciences and Engineering Research Council of Canada (NSERC), the US National Science Foundation (NSF), Microsoft, IBM, Hewlett-Packard Co. (HP), the Canadian Imperial Bank of Commerce (CIBC), and the SFU Community Trust Endowment Fund. He has published prolifically in refereed journals, conferences, and workshops. He is an associate editor of the *IEEE Transactions on Knowledge and Data Engineering.* He has served regularly on the organization committees and the program committees of many international conferences and workshops and has also been a reviewer for the leading academic journals in his fields. He is the recipient of the British Columbia Innovation Council 2005 Young Innovator Award, an IBM Faculty Award (2006), an IEEE Outstanding Paper Award (2007), and an NSERC 2008 Discovery Accelerator Supplement (DAS). He is a senior member of the ACM and the IEEE.

Ada Wai-Chee Fu received the BSc degree in computer science from the Chinese University of Hong Kong in 1983 and the MSc and PhD degrees in computer science from Simon Fraser University, Canada, in 1986 and 1990, respectively. She worked at Bell Northern Research, Ottawa, from 1989 to 1993 on a wide-area distributed database project. She joined the Department of Computer Science and Engineering, Chinese University of Hong Kong, Shatin, Hong Kong, in 1993. Her research interests include database systems and data mining. She is a member of the IEEE.

Ke Wang received PhD degree from the Georgia Institute of Technology. He is currently a professor in the School of Computing Science, Simon Fraser University, Burnaby, British Columbia, Canada. He has taught in the areas of database and data mining. His research interests include database technology, data mining and knowledge discovery, machine learning, and emerging applications, with recent interests focusing on the end use of data mining. This includes explicitly modeling the business goal and exploiting user prior knowledge (such as extracting unexpected patterns and actionable knowledge). He is interested in combining the strengths of various fields such as database, statistics, machine learning, and optimization to provide actionable solutions to real-life problems. He has published in database, information retrieval, and data mining conferences, including SIGMOD, SIGIR, PODS, VLDB, ICDE, EDBT, SIGKDD, SDM, and ICDM. He is an associate editor of the *IEEE Transactions on Knowledge and Data Engineering*, an editorial board member for the *Journal of Data Mining and Knowledge Discovery.* He is a senior member of the IEEE.