Contents lists available at SciVerse ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

Interactive skyline queries [☆]

Jongwuk Lee [a], Gae-won You [a], Seung-won Hwang [a,*], Joachim Selke [b], Wolf-Tilo Balke [b]

[a] Department of Computer Science and Engineering, Pohang University of Science and Technology (POSTECH), Pohang, Republic of Korea
[b] Institut für Informationssysteme, Technische Universität Braunschweig, Braunschweig, Germany

A R T I C L E I N F O

Article history:
Received 22 October 2009
Received in revised form 29 November 2010
Accepted 2 April 2012
Available online 24 April 2012

Keywords:
Skyline query
Preference elicitation

A B S T R A C T

When issuing user-specific queries, users often present vague and imprecise information needs. Skyline queries with an intuitive query formulation mechanism identify the most interesting objects for *incomplete* user preferences. However, the applicability of skyline queries suffers from a severe drawback because incomplete user preferences often lead to an impractical skyline size. To address this problem, we develop an *interactive preference elicitation* framework – while user preferences are collected at each iteration, the framework iteratively updates skylines. In this process, the framework aims to both minimize user interaction and maximize skyline reduction size, while the query formulation is still intuitive. All that users need to do is thus to answer a few well-chosen questions generated from the framework. We validate the effectiveness and efficiency of our framework in extensive experimental settings, and demonstrate that a few questions are enough to acquire a skyline with a manageable size.

1. Introduction

Querying in databases and information systems has evolved beyond SQL-style exact match queries. Modern query languages and processing algorithms account for vague and imprecise information needs. As prime techniques, top-k retrieval and skyline queries have been recently introduced.

In top-k retrieval, a query consists of an integer k and a *utility function*, which assigns a numerical score to each object. The database system returns a ranked list of k objects scoring the highest. Thus, top-k queries always provide a focused and manageable result set, but users usually find it difficult to come up with a utility function that resembles their personal preferences.

In contrast, skyline queries do not require users to specify a single all-inclusive utility function. A skyline query consists of a set of utility functions, each of which resembles a single aspect of object quality. Typically, each utility function possesses a simple structure, e.g., a sorted list, and can come with natural preference order. In other words, each utility function can directly reflect a *qualitative* user preference on a single attribute defined in the database's relational schema.

The skyline query is to return a set of *Pareto-optimal* objects, called a *skyline*. Specifically, an object a can be a *skyline object* if there exists no other object b dominating a – an object a is said to *dominate* another object b if a scores better than b with respect to at least one utility function, and a does not score worse than b with respect to any other utility functions. Note that only qualitative order induced by each utility function matters for the skyline query, which makes it more intuitive.

The following example illustrates a typical skyline query.

☆ This paper is based on and significantly extends preliminary work [31] presented at DEXA 2008.
* Corresponding author.
E-mail address: swhwang@postech.ac.kr (S.-w. Hwang).

0020-0255/$ - see front matter © 2012 Elsevier Inc. All rights reserved.
http://dx.doi.org/10.1016/j.ins.2012.04.007

Table 1
A toy dataset in Examples 1 and 2.

ID	Type	Color	Brand	Price	Speed
1	Convertible	Red	Ferrari	80	190
2	Sedan	Blue	Ferrari	150	160
3	Convertible	Blue	Honda	100	160
4	Sedan	Red	Honda	70	200
5	Roadster	Blue	Honda	200	150

Example 1. Consider a customer named Alice shopping for a car that should be optimal with respect to five attributes: *type*, *color*, *brand*, *price*, and *speed* (Table 1). Alice prefers convertibles to sedans, sedans to roadsters, red cars to blue ones, and Ferraris to Hondas. Also, price and speed are naturally ordered by "the cheaper the better" and "the faster the better" respectively. Her preferences can thus be represented by "*type*: convertible $\succ$ sedan $\succ$ roadster; *color*: red $\succ$ blue; *brand*: Ferrari $\succ$ Honda; *price*: minimize, *speed*: maximize", where $\succ$ denotes a *dominance* relationship. We can rule out cars 2, 3, and 5, because they are all dominated by car 1. The remaining cars 1 and 4 make up skyline objects.

The skyline query in Example 1 is useful for identifying objects satisfying user preferences, but it does not scale to attributes having a large domain without natural order. In that case, users are required to define large preference orderings within their queries, which is very demanding and tedious. Thus, this manual preference elicitation for *complete* user preferences can make the paradigm of skyline query useless.

To overcome this problem, we allow users to present skyline queries with *incomplete* user preferences, as illustrated by the following example.

Example 2. Alice is shopping for a car, but now she solely specifies her incomplete preferences on some attributes. There are several ways to leave out some preference information as follows: (1) "*type*: convertible $\succ$ sedan; *color*: red $\succ$ blue; *brand*: Ferrari $\succ$ Honda; *price*: minimize, *speed*: maximize". The skyline is cars 1, 4, and 5, because her preferences do not offer whether roadsters are better than convertibles or sedans; (2) another query with incomplete preferences is "*type*: convertible $\succ$ sedan $\succ$ roadster; *color*: ?; *brand*: Ferrari $\succ$ Honda; *price*: minimize, *speed*: maximize", where? is no preference information. The incomplete preferences yield cars 1–4 as the skyline.

Example 2 implies that a large number of different skyline queries can involve incomplete preferences. Clearly, the user want to get the skyline whose size is as small as possible, while providing as little preference information as necessary. In practical scenarios, the amount of preference information available at query time is usually limited. On the other hand, when the system requires the user to ask complex or complete preferences before querying, the user often need to make considerable efforts to specify her preferences in sufficient detail.

A solution to this problem would be to provide only the "most informative" preferences within the query. However, because the user does not know what objects are contained in the database system, it is impossible for the user to choose informative preferences. On the other hand, the database system knows the objects stored well.

In this paper, we thus study how the user and the database system can refine skyline queries involving attributes without natural order in an interactive way. Specifically, we devise a preference elicitation framework, which not only minimizes user interaction but also maximizes skyline reduction size. Because user-specific preferences are not known *a priori*, our proposed framework is based on a *probabilistic* approach for *missing knowledge* about user preferences, and iteratively requires the user to elicit her preferences.

To summarize, this paper makes the following contributions:

- We state a problem definition for our framework that captures the notion of informative preferences (Section 2).
- We propose an effective elicitation framework and discuss its efficient implementation (Section 3).
- We extend our framework for general datasets including both attributes with and without natural order (Section 4).
- We evaluate the effectiveness and efficiency of the proposed algorithms (Section 5).

2. Problem statement

This section states preliminaries to address the preference elicitation problem. We first introduce notations used to define preference queries, skyline queries, and preference elicitation. Table 2 summarizes the notations used in this paper.

2.1. Preference queries

Preference queries are commonly used to retrieve the "best" tuples. Throughout this paper, we deal with a relational schema that consists of d attributes, i.e., $A_1, A_2, \ldots, A_d$. The schema's domain is denoted by $\mathcal{D}$, i.e., $\mathcal{D} = D_1 \times D_2 \times \cdots \times D_d$, where D_i is the domain of attribute A_i. A dataset $\mathcal{O}$ is a finite subset of $\mathcal{D}$, i.e., $\mathcal{O} \subseteq \mathcal{D}$. As usual in databases, $\mathcal{O}$ is a multiset, implying that $\mathcal{O}$ could contain duplicate tuples.

Table 2
The list of notations.

Symbol	Definition
$\mathcal{A}$	Attribute set
d	Number of attributes
A_i	ith attribute
D_i	Domain of A_i, where $m_i = \|D_i\|$
$\mathcal{D}$	Global domain, i.e., $\mathcal{D} = D_1 \times \cdots \times D_d$
$\mathcal{O}$	A dataset, i.e., $\mathcal{O} \subseteq \mathcal{D}$
n	Cardinality, i.e., $n = \|\mathcal{O}\|$
P_i	Pareto preference relation on D_i
$\succ_i$	Strict preference on D_i
$\sim_i$	Equivalence on D_i
$\succeq_i$	Weak preference on D_i
$\\|_i$	Missing preferential knowledge on D_i
$\mathcal{P}$	Pareto preference relation on $\mathcal{D}$
$\succ$	Strict preference on $\mathcal{D}$
$\sim$	Equivalence on $\mathcal{D}$
$\succeq$	Weak preference on $\mathcal{D}$
$\\|$	Missing preferential knowledge on $\mathcal{D}$
(A_i,x,y)	Question Q about x's and y's preference relation on A_i, where $x,y \in D_i$
$\mathcal{P}^Q$	Updated preference relation after asking question Q
S	Skyline of $\mathcal{O}$ w.r.t $\mathcal{P}$
S^Q	Skyline of $\mathcal{O}$ w.r.t $\mathcal{P}^Q$
$S^Q_\succ$	Skyline of $\mathcal{O}$ w.r.t $\mathcal{P}^Q$ assuming Q will be answered by $\succ$
$S^Q_\sim$	Skyline of $\mathcal{O}$ w.r.t $\mathcal{P}^Q$ assuming Q will be answered by $\sim$
$S^Q_\prec$	Skyline of $\mathcal{O}$ w.r.t $\mathcal{P}^Q$ assuming Q will be answered by $\prec$
ESR(Q)	Expected skyline reduction of question Q given prior knowledge (Definition 9)

In preference-based retrieval, a dataset $\mathcal{O}$ is commonly called a set of *(actual) alternatives*, whereas $\mathcal{D}$ is referred to as *data space* or a set of *possible alternatives*. We call n and d the *cardinality* and *dimensionality* for $\mathcal{O}$ respectively.

Typically, because n is very large and the user needs specific information, a focused retrieval strategy is required to find a preferably small subset of $\mathcal{O}$ satisfying user's information needs. To address this problem, a classical SQL requires the queries to specify hard constraints, e.g., Boolean conditions, defined by a subset of $\mathcal{D}$.

In contrast, preference queries deal with information needs that cannot be expressed with strict constraints. With preference-oriented query languages, e.g., PreferenceSQL [26] or winnow operator [14], the user is able to define *preference order* on $\mathcal{D}$. That is, given any attribute values $x,y \in D_i$, x is preferred to y, both are of equal preference, or y is preferred to x. The preference-based queries are answered by only the "best" alternatives in $\mathcal{O}$ satisfying user-specific preferences.

2.2. Skyline queries

One of the most popular preference-based query paradigms is a *skyline query*. The query induces a set of r relevant attributes $A_{i_1},\ldots c, A_{i_r}$ along with r *Pareto relations* $P_{i_1},\ldots c, P_{i_r}$ presenting user preferences, where $P_{i_j} \subseteq D_{i_j} \times D_{i_j}$ for any $j = 1,\ldots,r$. For the sake of representation, we will assume that all d attributes in $\mathcal{A}$ are of interest to the user.

Basically, a Pareto relation P_i is a binary relation representing *weak order*. Given an attribute value pair $(x,y) \in P_i$, it means x is preferred to or equal to y, denoted as $x \succeq_i y$. The asymmetric part and symmetric part of weak order, denoted as $\succ_i$ and $\sim_i$, correspond to *strict order* and *equivalence*, respectively.

Using the Pareto relation, we present possible relationships between attribute values. First, $(x,y) \in P_i$ means that x *dominates* y on D_i, denoted as $x \succeq_i y$. Second, given $(y,x) \in P_i$, it means x is dominated by y, denoted as $x \preceq_i y$, where the asymmetric part, denoted as $x \prec_i y$, means x is less preferred to y on D_i. As the last case, if both $(x,y) \in P_i$ and $(y,x) \in P_i$ hold, x and y are equally preferred to the user. That is, x and y are *equivalent* on D_i, denoted as $x \sim_i y$.

Based on the relationships, we formally define the notions of *dominance*, *incomparability*, and *skyline*, respectively.

Definition 1 (*Dominance*). Given two objects $a,b \in \mathcal{O}$, an object a *dominates* another object b on $\mathcal{A}$ if and only if $\forall D_i \in \mathcal{D} : a_i \succeq_i b_i$ and $\exists D_j \in \mathcal{D} : a_j \succ_j b_j$.

Definition 2 (*Incomparability*). Given two objects $a,b \in \mathcal{O}$, a and b are *incomparable* on $\mathcal{A}$, if and only if $\exists D_i \in \mathcal{D} : a_i \not\succeq_i b_i$ and $\exists D_j \in \mathcal{D} : b_j \not\succeq_j a_j$.

Definition 3 (*Skyline*). An object a is a *skyline object* on $\mathcal{A}$ if and only if a is not dominated by any other object b on $\mathcal{A}$. Given a dataset $\mathcal{O}$, a *skyline* is a set of skyline objects.

In practice, the user cannot manually specify all single-attribute preferences. Specifically, numerical attributes can possess natural preference order, e.g., *price* and *speed*. However, categorical attributes, e.g., *color* and *brand*, do not have a pre-defined preference order. The user thus needs to specify her preferences manually.

Because we cannot reasonably assume that all single-attribute preferences have been specified completely, we account for *missing knowledge*, representing undefined user preferences, by dropping the requirement of *totality*. Given two attribute values $x,y \in D_i$, let $x \parallel_i y$ denote missing knowledge on (x,y). Note that the values on missing knowledge are *incomparable* to each other, and cannot exploit *transitivity* with other preference relationships.

We provide an example using the toy dataset (Table 1). Suppose that incomplete user preferences are represented by "*type*: convertible $\succ$ sedan $\succ$ roadster; *color*: ?; *brand*: Ferrari $\succ$ Honda; *price*: minimize, *speed*: maximize". Because user preferences for *color* are unknown, cars 1 and 2 do not dominate each other regardless of other attributes. In other words, cars with different colors are incomparable to each other.

Meanwhile, there exists a special case where dominance relationships hold between two objects, even when user preferences are unknown for some attributes. Specifically, when user preferences on *color* are unknown, the dominance relationship between cars 1 and 4 sharing the same colors can be determined based on user preferences on *type* and *brand*. Section 3 will discuss how to deal with the dominance relationships between objects in detail.

Without loss of generality, let $\mathcal{P}$ denote an aggregated Pareto relation with respect to $P_1,\ldots,P_d$. Also, to present user preferences on $\mathcal{P}$, we will consistently use notations $\succeq$, $\preceq$, $\succ$, $\prec$, $\sim$, $\not\succ$, and $\parallel$.

2.3. Preference elicitation

For missing preference knowledge, we perform *interactive preference elicitation*. Instead of confronting the user with a lot of arbitrary questions about her unknown preferences, the database system analyzes the current skyline, and derives which questions could reduce the skyline size as much as possible.

We discuss how to model preference elicitation for collecting the most "informative" user preferences. Specifically, the database system first analyzes missing knowledge based on the current skyline, and then considers preference elicitation in an interactive fashion. When a question Q is selected from the system, the user solely answers which one of $x \succ_i y$, $x \prec_i y$, and $x \sim_i y$ holds. Formally:

Definition 4 (*Question*). A question Q with respect to attribute A_i is a triple (A_i,x,y), where $x,y \in D_i$. The answer to Q is a preferential relationship that holds between x and y on D_i. That is, in order to answer Q, the user states exactly one of $x \succ_i y$, $x \prec_i y$, and $x \sim_i y$.

User preferences are consistently collected, and the current skyline is updated accordingly. This process continues until either the size of the skyline is small enough to satisfy user's information needs, or there exist no more informative questions. In the latter case, additional information is not required to reduce the skyline size.

This elicitation process may also contain some initial information on user's preferences. Specifically, they may have domain-specific preferences shared by all users, e.g., a preference for low prices, or personalized preference information based on a user profile derived from previous interaction with the system. For efficiency, we can thus initialize pre-defined preferences to the system.

We model our framework with an iterative elicitation method. We capture the system's preferential knowledge, i.e., $\mathcal{P} = \{P_1,\ldots,P_d\}$, before asking a question. Let S denote the skyline with respect to $\mathcal{P}$. Also, the updated system after answering a question Q is described by $\mathcal{P}^Q = \left\{P_1^Q,\ldots c, P_d^Q\right\}$, where the corresponding skyline is represented by S^Q.

We formally state our preference elicitation model as follows.

Definition 5 (*Preference elicitation*). Given a Pareto relation $\mathcal{P}$, preference elicitation consists of the following steps:

1. Choose an attribute A_i and attribute values $x,y \in D_i$, where $x \parallel_i y$ holds. Construct the question $Q = (A_i,x,y)$.
2. Ask the question Q, and record user's answer.
3. If the answer is $x \succ_i y$, define P_i^Q to be the transitive closure of the relation $P_i \cup \{(x,y)\}$. If $x \sim_i y$, define P_i^Q as the transitive closure of $P_i \cup \{(x,y),(y,x)\}$. Otherwise, P_i^Q is the transitive closure of the relation $P_i \cup \{(y,x)\}$. The other relations are not changed, i.e., $\forall j \in [1,d],\ j \neq i : P_j^Q = P_j$.

2.4. Quality of elicitation processing

The hardest part of preference elicitation is to determine the most informative question. Some question Q may result in a large decrease of skyline size when stepping from S to S^Q, while other questions might not help to decrease the skyline size at all. For example, assuming that we know nothing about user preferences, statistics show that a dataset $\mathcal{O}$ contains roughly as many blue cars as red cars. In this case, it would be a reasonable strategy to ask the first question about the preference relationship between "red" and "blue".

Based on this intuition, we formally define our elicitation method as follows:

Definition 6 (*Elicitation method*). An *elicitation method* returns a question Q based on a deterministic algorithm that takes relations $P_1,\ldots,c,P_d$, a dataset $\mathcal{O}$, and possibly some additional background information as inputs.

Clearly, the quality of an elicitation method depends highly on the distribution of actual alternatives and user's true preferences.

We then define a key property of the elicitation method to find the most informative question at each step.

Definition 7 (*Step-optimal elicitation*). An elicitation method is called *step-optimal*, if a question Q maximizing the difference $|S| - |S^Q|$ is chosen over all possible questions.

In practice, a step-optimal elicitation method cannot be built without knowing actual user preferences. However, we may have some prior information about what she might answer in response to each question, where her possible answers to the question Q are three cases: $x \succ_i y$, $x \prec_i y$, or $x \sim_i y$.

Based on this observation, we introduce a *probabilistic* approach to the elicitation method.

Definition 8 (*Answer probabilities*). Given a question $Q = (A_i,x,y)$, $\Pr(x \succ_i y|Q)$ denotes the probability that the user elicits $x \succ_i y$ for Q. Similarly, $\Pr(x \prec_i y\,|\,Q)$ and $\Pr(x \sim_i y\,|\,Q)$ are the probabilities when answering $x \prec_i y$ and $x \sim_i y$ for Q. As always, the sum of all these probabilities is one: $\Pr(x \succ_i y\,|\,Q) + \Pr(x \prec_i y|Q) + \Pr(x \sim_i y\,|\,Q) = 1$.

According to user preferences, we define the notion of *expected step-optimal elicitation*.

Definition 9 (*Step-optimality in expectation*). Let $S^Q_\succ$ be the skyline where user's answer has been $x \succ_i y$ for $Q = (A_i,x,y)$. Analogously, define the expressions $S^Q_\prec$ and $S^Q_\sim$ for $x \prec_i y$ and $x \sim_i y$ respectively. Then, an elicitation method is called *step-optimal in expectation*, if Q maximizes the difference $|S| - \left(\Pr\left(x\succ_i y|Q \cdot |S^Q_\succ| + \Pr(x\prec_i y|Q)\cdot\left|S^Q_\prec\right|\right) + \Pr(x\sim_i y|Q)\cdot\left|S^Q_\sim\right|\right)$ over all possible questions. We refer to this number as the *expected skyline reduction* ESR (Q) with respect to Q.

When there exists no prior information about these probabilities available, we cannot estimate the probability for user preferences. In this case, we assume that user preferences are in uniform distribution. All probabilities are thus equal, i.e., $\Pr(x\succ_i y|Q) = \Pr(x\prec_i y|Q) = \Pr(x\sim_i y|Q) = \frac{1}{3}$, for any question Q.

Note that, if we have background knowledge on user preferences, the probabilities for user preferences can be adapted. Section 3.3 will discuss the preference-adaptive method in detail.

3. A general elicitation framework

In this section, we develop a general preference elicitation framework. For brevity, we first assume that all d attributes in $\mathcal{A}$ consist of categorical attributes. We will later extend our framework for general datasets including both categorical and numerical attributes (Section 4).

Given user-specific retrieval size k, our framework consists of the following four steps:

1. *Question selection*: Based on a current Pareto relation $\mathcal{P}$ and background information on user preferences, choose a question $Q = (A_i,x,y)$ among missing preferences. In particular, to minimize user interaction, our framework aims to select Q maximizing expected skyline reduction size (Definition 9).
2. *Preference elicitation*: A question Q is shown to the user. Based on the user answer for Q, we update $\mathcal{P}$ into $\mathcal{P}^Q$.
3. *Disqualifying dominated objects*: For $\mathcal{P}^Q$, dominated objects are removed from S, which yields S^Q.
4. *Presentation of results*: Check the size of S^Q that is shown to the user, and iterate until $|S^Q| \leqslant k$ or no more informative questions are available.

At step 3, we can reduce skyline size using the given user preferences on Q. Specifically, we can categorize possible user preferences into two types. First, when user's answer is strict order such as $\succ_i$ or $\prec_i$, this would result in a large decrease of skyline size using *ceteris paribus* semantics (assuming that a dataset follows the conditions in Section 3.2). That is, given $x \succ_i y$, we can prune out all objects $(b_1,\ldots,b_{i-1},y,b_{i+1},\ldots,b_d)$ if there exists an object $(a_1,\ldots,a_{i-1},x,a_{i+1},\ldots,a_d)$ such that $\forall D_j \in \mathcal{D}/D_i : a_j \succeq_j b_j$. With a similar method, when a user preference is $x \prec_i y$, we can prune out all dominated objects. Second, when a user preference is equivalent, i.e., $x \sim_i y$, we can prune out all objects $(b_1,\ldots,b_{i-1},y,b_{i+1},\ldots,b_d)$ if there exists an object $(a_1,\ldots,a_{i-1},x,a_{i+1},\ldots,a_d)$ such that $\forall D_j \in \mathcal{D}/D_i : a_j \succeq_j b_j$, and $\exists D_k \in \mathcal{D}/D_i : a_k \succ_k b_k$.

We can also infer some missing knowledge from a newly specified preference. For example, suppose that the current user preference is $y \succ_i z$, and other preferences such as $x \parallel_i y$ and $x \parallel_i z$ are unknown. Given $Q = (A_i,x,y)$, when a user preference is specified by $x \sim_i y$, we can derive $x \succ_i z$ from existing preferences $x \sim_i y$ and $y \succ_i z$ using *transitivity* although the preference between x and z is unknown. We can thus reduce skyline size using both $x \succ_i z$ and $y \succ_i z$.

At step 4, our framework then updates S to S^Q iteratively, and checks the size of updated skylines. In this process, we can only guarantee that any object a is in the final skyline when there exists no object b dominating a. For instance, suppose that user preferences in Table 1 are given: "red $\succ_{\text{color}}$ blue"; "Ferrari $\succ_{\text{brand}}$ Honda", where categorical attributes are only considered (Table 1). In this case, car 1 can be the final skyline regardless of other preferences. When current skyline objects are incomparable to each other, and the user preferences on missing knowledge do not affect current skyline objects, we can decide the final skyline.

3.1. Algorithm OptPrune

This section presents a step-optimal elicitation algorithm, called OptPrune. We first present a step-optimal question selection. We then explain how to dynamically update user preferences, and discuss how to maintain current skyline objects over such changes. Lastly, we iteratively perform this procedure until the termination condition is satisfied.

3.1.1. Finding step-optimal questions

We deal with identifying a step-optimal question based on a current preference relation and a skyline. Our approach first enumerates a set Ω of all possible pairs, and then identifies one pair maximizing the expected skyline reduction size:

$$Q = \text{argmax}_{Q^* \in \Omega} \text{ESR}(Q^*)$$

In our toy example (Fig. 2), we would enumerate all five possible pairs. Among these pairs, we then choose the question $Q = (\text{color}, \text{red}, \text{blue})$, which has the highest expected skyline reduction size.

According to our prior analysis, an update on preference relation $\mathcal{P}^Q$ and skyline S^Q incurs a cost of $O(m^2)$ and $O(n'n'')$ at each iteration, respectively. The overall complexity is thus $O(dm^2 \cdot (m^2 + n'n''))$, where dm^2 is an upper bound for the number of possible questions.

3.1.2. Updating preference elicitation

Given a question $Q = (A_i, x, y)$, a user states her preference as one of possible preference types: $x \succ_i y$, $x \prec_i y$, or $x \sim_i y$. For simplicity, for all attributes assume that the size of a domain D_i is m.

Conceptually, we introduce an m-by-m *adjacency matrix* to represent incomplete user preferences. Let M_i be the matrix representing user preferences on D_i, where the entry $M_i(x,y)$ represents the preference relationship between the xth and yth attribute values, where D_i has an arbitrary but fixed attribute ordering. Each entry $M_i(x,y)$ on M_i can represent one of four different cases: *dominating*, *dominated*, *equivalence*, and *missing*.

- *Dominating*: if the xth value is preferred to the yth value on D_i, i.e., $x \succ_i y$, then $M_i(x,y) = 1$.
- *Dominated*: if the xth value is less preferred to the yth value on D_i, i.e., $x \prec_i y$, then $M_i(y,x) = -1$.
- *Equivalence*: if the xth and yth values are equivalent, i.e., $x \sim_i y$, then $M_i(x,y) = 0$.
- *Missing*: if the xth and yth values are missing knowledge, i.e., $x \parallel_i y$, then $M_i(x,y) = \infty$.

A matrix M_i is initialized by setting all its diagonal elements to 0, and the other elements to ∞. Given a user preference, we then update the matrix for the corresponding user preferences, e.g., if $x \succ_i y$, then $M_i(x,y) = 1$ and $M_i(y,x) = -1$ are updated. The matrix easily illustrates user preferences, but most entries on M_i may be ∞ representing missing values, which incurs a *sparse* matrix.

We thus consider a *compressed* matrix representation to prevent unnecessary storage and computation in our implementation. Specifically, we only store entries for dominating or equivalence specified by the user, based on which a *preference graph* representing incomplete user preferences is built up.

More specifically, we present how to update this structure in every elicitation step. Given a user preference $M_i(x,y)$, we have to update both $M_i(w,x)$ and $M_i(y,z)$, where w and z are any *reachable* ancestor and descendent node for x and y, respectively. Given $M_i(x,y)$, the time complexity of updating the preference graph is $O(m^2)$. More precisely, we need to update only entries connected by both ancestors and descendants, i.e., $O(\#\ \textit{of ancestors} \cdot \#\ \textit{of descendants})$.

To illustrate this, Fig. 1 describes two preference graphs, before and after stating a user preference "sedan $\succ_{\text{type}}$ roadster". A directed arrow is a strict preference $\succ_{\text{type}}$ and a bi-directed arrow is an equivalent preference $\sim_{\text{type}}$. By traversing the preference graph on M_i, we can update all reachable ascendants to "sedan" (dashed arrows). We can also update all reachable descendants from "roadster". For example, given "sedan $\succ_{\text{type}}$ roadster", we first add an edge for $M_{\text{type}}(\text{sedan}, \text{roadster}) = 1$, and then traverse the graph to update all reachable entries such that $M_{\text{type}}(\text{convertible}, \text{roadster}) = 1$, $M_{\text{type}}(\text{convertible}, \text{sports car}) = 1$, and $M_{\text{type}}(\text{sedan}, \text{sports car}) = 1$.

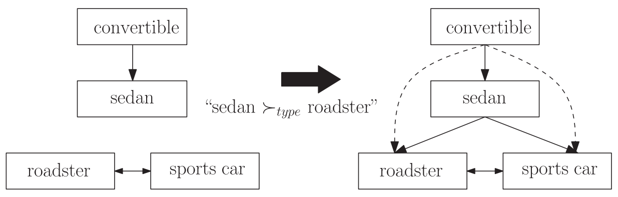
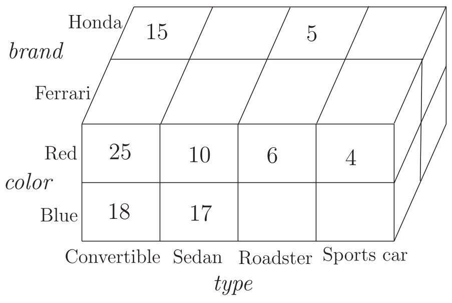


Fig. 1. Illustration of changing a preference graph (dashed arrows are newly added dominance relationships by transitivity).



(a) A three-dimensional data cube

cell	*cardinality*
(convertible, red, Ferrari)	25
(convertible, blue, Ferrari)	18
(convertible, red, Honda)	15
(sedan, red, Ferrari)	10
(sedan, blue, Ferrari)	17
(roadster, red, Ferrari)	6
(roadster, red, Honda)	5
(sports car, red, Ferrari)	4

(b) A data table

Fig. 2. Illustration of a data cube using a toy car dataset.

3.1.3. Disqualifying dominated objects

After updating a preference relation $\mathcal{P}$, we have to update a skyline S^Q as well. As an efficient data structure for pruning, we leverage a *data cube* [22], which allows $\mathcal{O}$ to collect multiple objects that share the same categorical values into a single *cell*. This structure is well-suited for categorical attributes with low-cardinality domains, because objects tend to share values, which are aggregated into a compact single cell in the data cube. With this data cube, we can only identify a skyline with a set of cells.

Fig. 2 describes a data cube as a compact representation of the current dataset $\mathcal{O}$. Given a user preference "convertible $\succ_{\text{type}}$ sedan", we remove all dominated cells from the data cube.

Basically, we can remove the dominated cells based on *ceteris paribus* semantics ('all-else-being-equal') [23], i.e., two objects only differ from a single attribute value on the user preference and the other attributes are equal. For instance, when the user preference is "red $\succ_{\text{color}}$ blue", we can remove (convertible, blue, Ferrari) and (sedan, blue, Ferrari), which are dominated by (convertible, red, Ferrari) and (sedan, blue, Ferrari) respectively (Fig. 2b). We formally state the special case of dominance relationships:

Definition 10 (*Ceteris paribus dominance*). Given a preference $a_i \succ_i b_i$, a dominates b on $\mathcal{D}$ if $\forall D_j \in \mathcal{D} \setminus D_i : a_j \sim_j b_j$.

We can also remove dominated cells by the transitivity. For instance, suppose current user preferences are "convertible $\succ_{\text{type}}$ sedan" and "roadster $\sim_{\text{type}}$ sports car" (Fig. 1). Given "sedan $\succ_{\text{type}}$ roadster", we can infer the other preferences, e.g., "convertible $\succ_{\text{type}}$ roadster", by transitivity, which can be exploited for removing dominated cells. In this process, the overall time complexity is $O(n' \cdot n'')$, where n' and n'' are the number of cells including current skyline objects and the number of cells with a newly preferred value, respectively.

For efficient implementation, we leverage multiple sorted lists [5] to check a final skyline while reducing dominated cells. Specifically, we exploit a sorted list for objects based on a preference graph. First, we match each attribute value into a preference value – attribute values with missing preferences correspond to the smallest values, and other attribute values correspond to level numbers when traversing the preference graph in breadth-first search. Second, we sort current objects in increasing order of preference values. Lastly, we access attribute values for each sorted list in a round-robin manner until any object is accessed on all sorted lists. We elaborate on this procedure as follows:

1. Construct d sorted lists based on preference graphs, i.e., $L_1, \ldots, L_d \leftarrow \{\}$. Each node in a sorted list consists of an object identifier and an integer value representing preferences. The values with missing relationships to the other values are zero. The other attribute values correspond to level numbers in breadth-first search. We then sort identifiers in increasing order of their corresponding values. As a tie-breaker, we use object identifiers.
2. Initialize a data table K, which stores objects with an identifier and its values. Also, initialize a counter $i = 1$, representing an attribute id accessed.
3. Get the next object a by sorted access on L_i. If $a \in K$, update its ith value. Otherwise, add a and its ith value into K. For objects with the same values, we can access such objects using the data cube at once. We then update i into $i = (i \bmod d) + 1$.
4. Repeat the above process until any object b in K has all attribute values. If the termination condition is satisfied, we check additional skyline candidates that have same values for b on L_i, and add the candidates into K.
5. Determine a *watermark object* that appears on all attributes in K, based on which we perform pair-wise dominance tests, and rule out dominated objects. Recall that we need to consider incomparability for values with missing relationships. For efficient pair-wise comparisons, we can exploit access the order of objects on L_i [5].

In this process, we can identify final skyline objects regardless of additional preference elicitation in an *eager* or a *lazy* fashion. Specifically, while the eager approach finalizes some skyline objects during disqualifying non-skyline objects in a

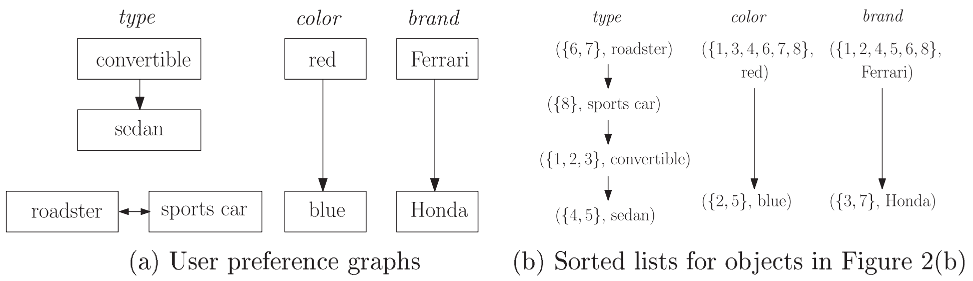


Fig. 3. Illustration of constructing sorted lists based on user preference graphs.

"progressive" manner, the lazy approach waits the final skyline until all the pruning procedures are completed. The eager approach thus has the benefit of avoiding redundant computation on final skylines, but incurs additional dominance comparisons, which is the trade-off between two approaches. (We evaluate the efficiency of two approaches in Section 5.3.3.).

We continue to use a toy dataset (Fig. 2). The user preferences are given as: "convertible $\succ_{type}$ sedan"; "roadster $\sim_{type}$ sports car"; "red $\succ_{color}$ blue"; "Ferrari $\succ_{brand}$ Honda" (Fig. 3a). Based on the preference graphs, we can construct sorted lists for objects in Fig. 2b (Fig. 3b). The object ids are granted according to the order of objects in the data table (Fig. 2b). We then access objects in a round-robin manner, and identify a watermark object. After that, we perform pair-wise dominance tests, where missing relationships such as "convertible $\|_{type}$ roadster", "convertible $\|_{type}$ sports car", "sedan $\|_{type}$ roadster", and "sedan $\|_{type}$ sports car" have to be considered regardless of the order of sorted lists. In this process, an object (convertible, red, Ferrari) is decided as a watermark object, and objects $\{2,3,4,5,7\}$ are removed. The current skyline is thus $\{1,6,8\}$, and its size is 35.

3.1.4. Overall procedure of OptPrune

Putting all three pieces together, we describe our algorithm OptPrune as follows.

1. *Initialization:* Initialize the given dataset $\mathcal{O}$ into a data cube, and the initial skyline S includes all data cells.
2. *Question selection:* Enumerate all possible questions and identify step-optimal question Q with the highest expected skyline reduction.
3. *Preference elicitation:* Given a question $Q = (A_i, x, y)$, the user provides the corresponding preference relation, based on which all preference relations are updated.
4. *Disqualifying dominated objects:* Given the updated preference graph, we remove all dominated objects and update S^Q.
5. *Iteration*: Repeat steps 2–4 until the skyline size is smaller than or equal to user-specified retrieval size k, or no more informative questions are available.

To continue the illustration with our toy dataset, we describe the overall procedure after choosing a step-optimal question $Q = (\text{color}, \text{red}, \text{blue})$. Assume that a user specifies her preference as "red $\succ_{color}$ blue". A user preference graph is then updated for $M_{color}(\text{red}, \text{blue}) = 1$. Because no such graph exists for "color", we only construct the graph for "red $\succ_{color}$ blue". After that, based on the ceteris paribus condition, we can prune out dominated cells from the data cube, i.e., (convertible, blue, Ferrari) and (sedan, blue, Ferrari), which are dominated by (convertible, red, Ferrari) and (sedan, red, Ferrari), respectively. The size of skyline S^Q thus narrows down to 65. Algorithm OptPrune iteratively continues the processing until the termination condition is satisfied.

Although OptPrune achieves step-optimality, it incurs high computational costs to choose an optimal question, e.g., 106 s for 100k objects as Section 5 reports. In the following, we thus develop an efficient heuristic question selection method.

3.2. Algorithm MaxPrune

We now develop a heuristic algorithm, called MaxPrune, which is significantly faster than OptPrune on running time, but does not compromise step-optimality much. Specifically, our approximate solution pursues step-optimality in a restrictive problem setting with the following two assumptions.

- *Non-sparsity:* There exists at least one object for every possible value combination. In other words, not all cells in the data cube are empty for all possible combinations.
- *Independence:* All attribute values are statistically independent.

We observe that these assumptions lead to the following desirable properties.

- *Step-optimality:* Algorithm MaxPrune is step-optimal if the above two assumptions hold. Also, although some cells can be empty in general problem settings, our empirical analysis in Section 5 reports that MaxPrune is close to OptPrune.

Table 3
The probability for each attribute in a toy car dataset.

Type	Prob.	Color	Prob.	Brand	Prob.
Convertible	0.4	Red	0.6	Ferrari	0.7
Sedan	0.3	Blue	0.4	Honda	0.3
Roadster	0.2	–		–	
Sports car	0.1	–		–	

- *Efficiency:* To select a step-optimal question, the assumptions enable us to only consider a question for two objects with the highest cardinality on each domain D_i. As a result, we can reduce the size of question candidates from $d\ m^2$ to d.

More specifically, among d question candidates, MaxPrune only considers a question with the highest expected skyline reduction.

The correctness of the above observations can be seen as follows. For ease of explanation, let $a(A_i)$ be the attribute value of an alternative a on A_i.

Lemma 1 (Cell cardinality in expectation). *Assuming that attributes are independent, the cell cardinality of $a(a_1,\ldots,a_d)$ is expected to $|\{a|a(A_i)=a_i\}| \times \prod_{j=1,j\neq i}^{d}\Pr(a(A_j)=a_j)$, where $|\{a|a(A_i)=a_i\}|$ represents the cardinality.*

Proof. The cell cardinality of $a(a_i,\ldots,a_d)$ is expected by $n \times \prod_{i=1}^{d}\Pr(a(A_i)=a_i)$, where n is the cardinality of $\mathcal{O}$. When $|\{a|a(A_i)=a_i\}|$ is known, we can leverage $n \times \Pr(a(A_i)=a_i)$ instead of $|\{a|a(A_i)=a_i\}|$. As a result, the cell cardinality of $a(a_1,\ldots,a_d)$ can be expected to $|\{a|a(A_i)=a_i\}| \times \prod_{j=1,j\neq i}^{d}\Pr(a(A_j)=a_j)$.

Based on Lemma 1, we derive how to select the question maximizing the expected skyline reduction. Basically, given user preference $x \succ_i y$ with respect to $Q=(A_i,x,y)$, we can prune out all objects $(a_1,\ldots,a_{i-1},y,v_{i+1},\ldots,a_d)$ if there exists an object $(b_1,\ldots,b_{i-1},x,b_{i+1},\ldots,b_d)$ such that $\forall D_j \in \mathcal{D}/A_i : b_j \succeq_j a_j$. In other words, we can prune out dominated objects by the ceteris paribus condition or the transitivity of user preferences.

We then consider the skyline reduction size for the given user preference $x \succ_i y$. Because the non-sparsity condition ensures that the if-clause is satisfied, we can only consider the skyline reduction size by the ceteris paribus condition. The skyline reduction size can simply be represented by $|\{a|a(A_i)=y\}| \times \prod_{j=1,j\neq i}^{d}\Pr(a(A_j)=a_j)$. The expected skyline reduction ESR (Q) for $Q=(A_i,x,y)$ can be represented by $\frac{1}{3} \times |\{a|a(A_i)=x\}| \times \prod_{j=1,j\neq i}^{d}\Pr(a(A_j)=a_j) + \frac{1}{3} \times |\{a|a(A_i)=y\}| \times \prod_{j=1,j\neq i}^{d}\Pr(a(A_j)=a_j)$. For simplicity, we ignore an equivalent preference $x \sim_i y$, because the equivalent preference does not reduce skyline size by the ceteris paribus condition. Also, because the probability $\prod_{j=1,j\neq i}^{d}\Pr(a(A_j)=a_j)$ is a constant for all questions on A_i, the optimal question maximizing the expected skyline reduction can be found by identifying the one with maximum cardinality $|\{a|a(A_i)=x\}| + |\{a|a(A_i)=y\}|$.

To illustrate this, we describe how to select the question with a toy dataset in Table 3. We first find questions with the highest cardinality for each attribute, i.e., (Type, convertible, sedan), (Color, red, blue), and (Brand, Ferrari, Honda). Among these questions, we determine a question $Q=(\text{color},\text{red},\text{blue})$, because the expected skyline reduction based on the cell cardinality is the highest, i.e., $\frac{1}{3} \times 0.6 + \frac{1}{3} \times 0.4$.

We analyze the performance of the algorithm MaxPrune. In order to select the question Q, md values are accessed once to find a pair of values with the highest cardinality. For the question selection, the time complexity of MaxPrune is thus $O(m\ d)$. Note that the MaxPrune algorithm is independent of accessing the actual dominance relationships to identify a step-optimal question. The time complexity of MaxPrune is thus $O(md + m^2 + n'\ n'')$ by combining the computation time $O(m^2)$ for finding a question and the computation time $O(m^2 + n'n'')$ for maintaining skyline objects.

To sum up, the overall cost significantly reduces from $O(m^2 d \cdot (n'n'' + m^2))$ to $O(m\ d + m^2 + n'n'')$, e.g., 106–6 s as Section 5 reports.

3.3. Preference-adaptive algorithms

We extend our two proposed algorithms into algorithms OptPruneQF and MaxPruneQF with background knowledge on distribution $\mathcal{Q}$ for user preferences.

When computing the expected skyline reduction, instead of setting the probability, each value on a sample is chosen equally, i.e., with probability $\frac{1}{3}$, we may apply our knowledge on user preferences obtained from prior query logs, such as *query frequency*. That is, the more frequently a value occurs, the higher the preferred chance of this value is. Specifically, when the distribution $\mathcal{Q}$ has different probability values according to query frequencies, the probability a value is chosen is calculated as the relative values on the sample. For instance, the frequency for "Ferrari" and "Honda" is p_f and p_h, respectively. The probability of choosing "Ferrari" over "Honda" is $p_f/(p_f + p_h)$, while that of choosing "Honda" is $p_h/(p_f + p_h)$.

Our notion of expected skyline reduction ESR (Q) can be straightforwardly extended. Let the query frequency of x and y on $Q=(A_i,x,y)$ be p_a and p_b, respectively. The relative probability is $p_b/(p_a + p_b)$ and $p_a/(p_a + p_b)$. The expected pruning cardinality is calculated with relative values, instead of 1/3. In other words, algorithms OptPruneQF and MaxPruneQF can consider both the pruning cardinality and the probability based on user preference distribution.

4. Extension for generalized datasets

This section shows how to extend our proposed algorithms to efficiently support a general dataset with both categorical and numerical attributes. For instance, a car dataset has both numerical attributes such as *price* and *speed* and categorical attributes such as *color* and *brand*. We conducted a survey on the UCI real-life data archive[1] and found that out of 150 datasets, the majority (59%) includes categorical attributes. Among these sets, 61% include numerical attributes as well. Our goal is thus to support such datasets with both a set of numerical attributes and a set of categorical attributes. For ease of illustration, we distinguish attribute set $\mathcal{A}$ into a *categorical attribute set* $\mathcal{A}^C$ and a *numerical attribute set* $\mathcal{A}^N$.

Contrary to categorical attributes, numerical attributes can easily be represented by complete ordering with respect to user preferences. For instance, user preference can be stated that the cheapest car is preferred, i.e., min(*price*) such that $\forall a, b \in \mathcal{O} : a \succeq_{\text{price}} b$ if $a(price) \leqslant b(price)$. Similarly, the user can state her preference of maximizing *speed*, e.g., max(*speed*), or having values around an ideal price, e.g., *around*(*price* = \$10,000), which corresponds to complete order such that $a \succeq_{\text{price}} b$ if $|a(price) - 10{,}000| \leqslant |b(price) - 10{,}000|$. In clear contrast, when the user preference is stated on categorical attributes, the user needs to elicit user preferences for $\sum_{A_i \in \mathcal{A}^C} \binom{|D_i|}{2}$ possible questions in the worst case scenario.

We thus propose a *pre-pruning procedure* using numerical attributes without requiring the elicitation of user preferences on categorical attributes. Specifically, for objects sharing the same categorical values, e.g., objects in the same data cube cell, we first perform dominance tests with respect to numerical attributes. If an object is dominated by another object with respect to numerical attributes, then such dominated objects can be pruned out. With the pre-pruning, we would update the data cube for categorical attributes in Fig. 2.

To illustrate this, we describe an example for a pre-pruning procedure. Given a toy dataset in Table 4, suppose that a user preference on numerical attributes are specified by "*price*: minimize, *speed*: maximize". For each cell with the same categorical values, we can easily remove dominated objects 1 and 5, where car 3 dominates car 1, and car 4 dominates car 5.

We formally state this pre-pruning procedure as follows:

Lemma 2 (Pre-pruning procedure). *We can prune out dominated object b satisfying two conditions: (i) all categorical attribute values are equal such that* $\forall A_i \in \mathcal{A}^C : a(A_i) = b(A_i)$*; (ii) a dominates b with respect to numerical attributes such that* $\forall A_j \in \mathcal{A}^N : a(A_j) \succeq_j b(A_j)$ *and* $\exists A_k \in \mathcal{A}^N : a(A_k) \succ_k b(A_k)$.

Proof. According to the dominance notion, since a is dominated by b, b cannot be a skyline object, regardless of the preference elicitation over categorical domains.

Recall that the numerical attributes are only used for the pre-pruning procedure, and they are not leveraged for building the data cube. In other words, objects with the same categorical values are stored in a single cell of the data cube, and numerical attributes are used in checking the dominance tests between cells.

After this pre-pruning, we consider a set of remaining objects as base skyline S. That is, we exploit user preferences for numerical attributes to reduce the size of an initial skyline before any elicitation on categorical attributes even begins. Note that if the size of the base skyline is already less than or equal to user-specified retrieval size k, i.e., $|S| \leqslant k$, our framework can be terminated without eliciting user preferences over categorical attributes.

To implement this idea, we can straightforwardly extend our pre-pruning processing to the Sort-First-Skyline (SFS) algorithm [17]. Assume that each numerical attribute is normalized to the unit interval [0, 1], where smaller values are better. We first sort the objects by an aggregated score of numerical values using a monotone function, i.e., $f(a) = \sum_{A_i \in \mathcal{A}^N} a(A_i)$. This ordering preserves the following property – if $f(a) \leqslant f(b)$, then a is not dominated by b [17]. This property helps reduce the number of dominance tests, as illustrated below:

1. Initialize a list L including current skyline objects as an empty set, i.e., $L \leftarrow \{\}$.
2. Sort objects in decreasing order for $f(a) = \sum_{A_i \in \mathcal{A}^N} a(A_i)$, and sequentially access objects in sorted order.

   (a) Let an object b be a current accessed object. For each object $a \in L$, iteratively perform dominance tests between a and b as follows:

      i. For $\mathcal{A}^C$, we first check whether categorical values between a and b are equal.
      ii. If the values are different, b is not dominated by a regardless of numerical attributes, and we can skip the dominance test on $\mathcal{A}^N$.
      iii. Otherwise, we additionally check whether b is dominated by a on $\mathcal{A}^N$.
      iv. If $a \succ b$, stop performing additional dominance tests. Otherwise, repeat steps (i)–(iii) until all objects in L perform dominance tests for b.

   (b) If b is not dominated by any other object $a \in L$, then b becomes a skyline object, and is inserted into L. Otherwise, b cannot be a skyline object, where b is pruned out.

[1] http://mlr.cs.umass.edu/ml/datasets.html.

Table 4
A toy dataset for two cells.

ID	Type	Color	Brand	Price	Speed
1	Convertible	Red	Ferrari	120	180
2	Convertible	Red	Ferrari	150	200
3	Convertible	Red	Ferrari	100	190
4	Sedan	Blue	Honda	150	180
5	Sedan	Blue	Honda	200	150

We repeat this processing until all objects in sorted order are accessed once. For efficiency, we can organize a list L with a data cube structure. We group objects on L by categorical attributes, and sort objects with the same categorical attributes. This optimization can remove unnecessary dominance tests for numerical attributes between objects.

Lastly, we analyze the cost of the pre-pruning processing. The cost of sorting is $O(|\mathcal{A}^N| \cdot n \cdot \log n)$. The cost of identifying skyline is $O(d \cdot n \cdot |S|)$, where $|S|$ is the size of remaining skyline objects after the pre-pruning procedure. The overall cost is thus $O(n \cdot (|\mathcal{A}^N| \cdot \log n + d \cdot |S|))$. After this pre-pruning, our proposed framework then iteratively elicits user preferences on categorical attributes until at most k skyline objects are identified.

5. Experiments

This section validates the effectiveness and efficiency of the algorithms OptPrune, MaxPrune, OptPruneQF, and MaxPruneQF using various synthetic datasets. We implemented all our algorithms in C++. Our experiments were carried out on an Intel Xeon machine with 3.20 GHz dual processors and 1 GB RAM running Linux.

5.1. Data generation

For extensive evaluations, we generated synthetic datasets by varying experiment settings, including data size n, dimensionality d, number of distinct attribute values m, user-specified retrieval size k% (of N), as described in Table 5. To evaluate the proposed elicitation methods, suppose that the synthetic datasets only have categorical attributes except for Section 5.3.4. Specifically, for every attribute, we randomly generated m distinct attribute values and query frequency, according to Uniform and Zipfian distributions, varying skewness from $z = 0$ (uniform) to $z = 2$.

5.2. Preference generation

We synthetically generated user preferences and preference elicitation according to *a priori* knowledge, i.e., query frequency $\mathcal{Q}$, to compare our frameworks with and without $\mathcal{Q}$. Specifically, we randomly generated query frequencies for each dimension based on Zipfian distribution with skewness $z' = [0,2]$. Then, we are based on user interaction on $\mathcal{Q}$ to prefer values in descending order of query frequencies for each dimension.

Note, when the descending order of query frequencies coincides with the descending order of cardinality, i.e., when two orders are perfectly correlated, the behaviors of OptPrune(MaxPrune) and OptPruneQF(MaxPruneQF) will be identical. We thus observed their behaviors over the varying correlations. In particular, we adopted *Kendall τ distance dist*, a widely-adopted metric to quantify the correlation between two orderings by query frequency $\mathcal{X}_i$ and cardinality $\mathcal{Y}_i$, and vary *dist* in our experiments, which is defined as follows:

$$dist = \frac{1}{n}\sum_{1}^{n}\mathcal{K}_i(\mathcal{X}_i, \mathcal{Y}_i),$$

$$\mathcal{K}_i(\mathcal{X}_i, \mathcal{Y}_i) = \frac{\sum_{(p,q)}|\mathcal{X}_i(p) > \mathcal{X}_i(q) \wedge \mathcal{Y}_i(p) < \mathcal{Y}_i(q)|}{m(m-1)/2},$$

where (p,q) is every possible pair of values and $\mathcal{X}_i(p)$ and $\mathcal{Y}_i(p)$ are positions of the orderings.

Table 5
Parameters for experimental setup.

Parameter	Value: default
Database size n	[1 K,100 K]: 10 K
Dimensionality d	[3,7]: 4
Number of distinct values m	[3,7]: 4
Retrieval size k (%)	[1,20]: 5
Data skewness z	[0,2]: 1
Query frequency skewness z'	[0,2]: 1
Kendall τ distance *dist*	[0,1]: 0.5

5.3. Experimental results

We report our experimental results in extensive synthetic datasets. Table 5 describes our experimental settings. Meanwhile, we compare our algorithms with Random and Oracle as follows:

- Random: This scheme randomly selects a question from an attribute domain selected in a round robin manner.
- Oracle: This scheme implements a theoretically optimal framework, knowing user feedback in advance, to compute the exact pruning cardinality of each question, which is unfeasible in any real-life application.

In particular, we adopt the following performance metrics:

- *Effectiveness:* We use the number of iterations until we identify the k% best results, averaged over 100 runs.
- *Efficiency:* We measure the runtime performance of our framework at each iteration, averaged over 100 runs.

5.3.1. Effectiveness of proposed algorithms

This section evaluates proposed algorithms over varying parameters. Fig. 4 reports the effectiveness of our algorithms by comparing Random and Oracle.

First, Fig. 4a reports our results over varying retrieval size k%. Observe that, algorithms OptPrune and MaxPrune minimizing the number of user interaction significantly outperform Random, which randomly chooses questions. Additionally, MaxPruneQF and OptPruneQF exploiting prior knowledge enhance the effectiveness even further. For instance, both algorithms MaxPrune and OptPrune save about 38% compared to Random when $k = 20\%$. In addition,F MaxPruneQF and OptPruneQF save about 11% more. In particular, our heuristics MaxPrune and MaxPruneQF show comparable performance with OptPrune and OptPruneQF respectively, but incur significantly less cost as we will discuss in Section 5.3.2. Our approximations are also comparable to Oracle, which achieves global optimality by ideally knowing user preferences.

Second, our algorithms dominate Random over varying m and d illustrated in Fig. 4b and c respectively. Observe also that our frameworks scale more gracefully compared to Random. For instance, when $m = 7$ in Fig. 4b, our frameworks save about 50% from Random, while saving more than 20% when $m = 3$.

Third, Fig. 4d validates the effectiveness over varying correlation distance. As discussed above, *dist* means how user preferences are consistent with query frequency. As *dist* is close to one, our elicitation methods based on query frequency can be more effective. Ideally, OptPruneQF and MaxPruneQF behave identically when $d = 1$, which can be observed from Fig. 4d. We can thus validate that OptPruneQF and MaxPruneQF can achieve more similar behaviors with Oracle as the correlation of query frequency and actual user preference is higher.

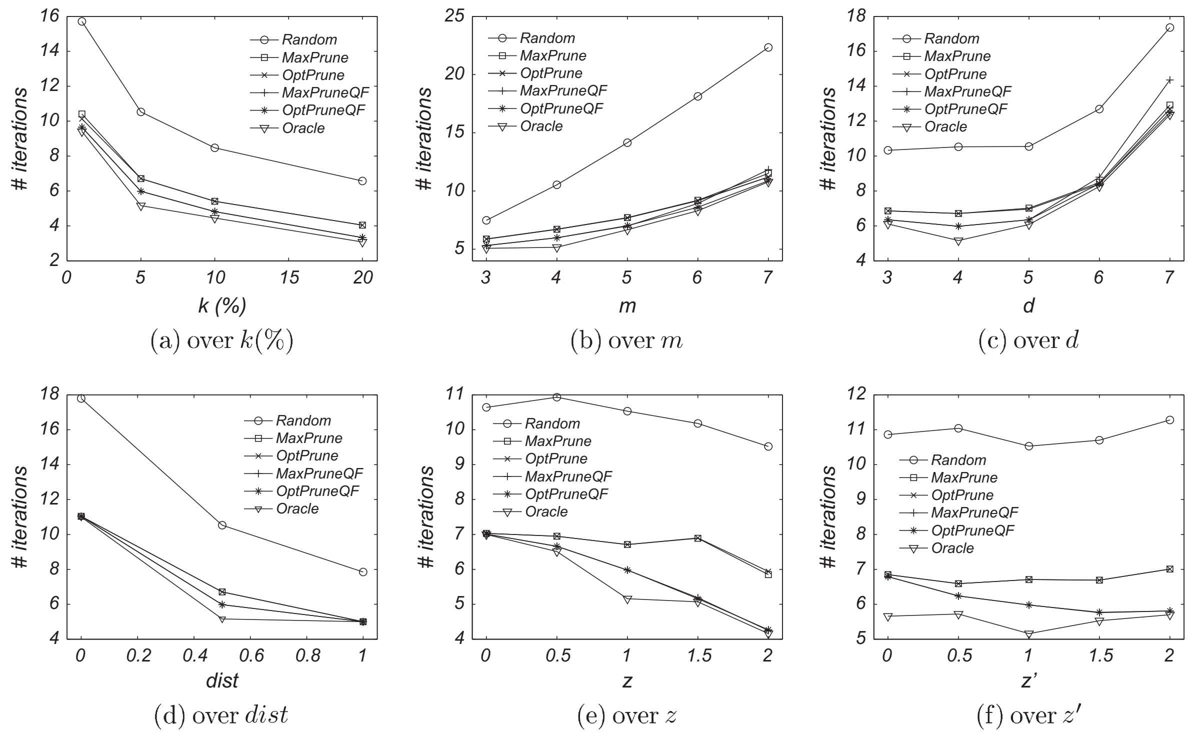


Fig. 4. Number of iterations over varying parameters.

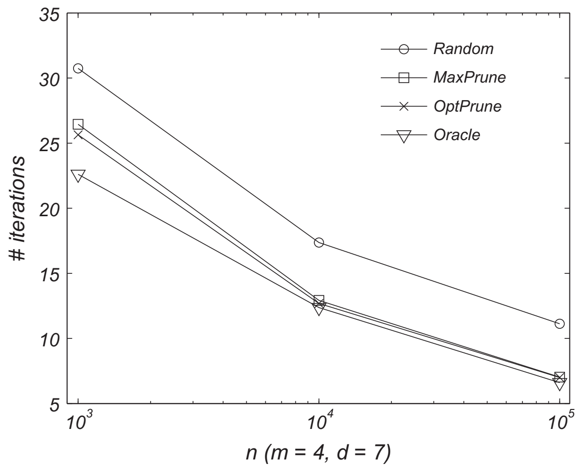


Fig. 5. Number of iterations over # objects.

Fourth, Fig. 4e and f validate the effectiveness with respect to skewness of datasets and query frequency. Observe that the effectiveness of our proposed algorithms increases as the skewness increases, especially for MaxPruneQF and OptPruneQF. For instance, when $z = 2$ both MaxPruneQF and OptPruneQF save 55% compared to Random. Similarly, the frameworks save about 48% of Random when $z' = 2$.

Fifth, Fig. 5 validates the effectiveness over varying data size n. Note that as the number of objects increases, the datasets are "non-sparse" (see Section 3.2), which is a desirable problem setting for MaxPrune by design. Fig. 5 confirms this intuition. As the non-sparsity (or the number of occupied cells) increases over increasing data size d, the performance gap between MaxPrune and OptPrune decreases and eventually converges to zero when $n \geqslant 10^4$.

Lastly, Fig. 6 validates the pruning effectiveness of our methods by illustrating the number of current skyline objects after each iteration over large scale datasets (a) $n = 100$ K, $m = 4$, and $d = 7$ (b) $n = 100$ K, $m = 7$, and $d = 4$, where the y-axis is log-scaled. Note that, the algorithms are terminated until no more informative questions are available. That is, while the number of iterations can be different depending on elicitation methods, each algorithm should have the same final skyline. Observe that our algorithms significantly outperform Random by maximizing the expected skyline reduction. For instance, in Fig. 6a, at the 5th iteration, both OptPrune and MaxPrune reduce the skyline size to below 12% while that of Random still remains at 25%. In particular, OptPrune and MaxPrune can save about 5 and 30 iterations compared to Random.

To summarize, our evaluation results validate that our proposed algorithms are comparably effective to Oracle, and they scale over all parameters. Our approximations, MaxPrune and MaxPruneQF, also perform comparably to Oracle, especially for dense datasets (Fig. 5) where its restricted problem assumption, e.g., non-sparsity, is met. Also, the relative performance gain over Random is more prominent in the presence of the skewness in terms of data and query frequency.

5.3.2. Efficiency of proposed algorithms

This section evaluates the efficiency of our proposed algorithms. Fig. 7 validates the efficiency of our methods, which correspond to Fig. 6. We report average response time for each iteration until the final skyline is determined.

Specifically, the response time of MaxPrune is comparable to OptPrune and Random at all iterations, and MaxPrune outperforms OptPrune and Random at every iteration. In particular, observe that MaxPrune significantly outperforms OptPrune, up to orders of magnitude, and performs comparably to Random. Note that both MaxPrune and OptPrune outperform Random after a certain number of iterations. For instance, after the 17th iteration, OptPrune begins to deal with a much smaller sample pool, and outperforms Random from this point on. While this may look counter-intuitive, this phenomenon is due to the higher pruning power of our proposed algorithms. As more data objects are pruned, a pool of samples gets smaller, which reduces the overhead of both sample selection and disqualified data pruning, performed at each iteration. Similarly, Fig. 7b reports similar results for datasets with $n = 100$ K, $m = 7$, and $d = 4$.

5.3.3. Efficiency of disqualifying non-skylines

This section evaluates the efficiency of disqualifying non-skyline objects. Specifically, we used a synthetic datasets when $d = 7$ and $m = 4$. We set other parameters as default values.

Specifically, we compare two approaches, i.e., eager vs. lazy, discussed in Section 3.1.3. Recall that an eager approach, finalizing some skyline objects during pruning non-skyline objects, incurs additional cost for finalizing some of the current skyline objects, but avoids redundant computation on the final skyline. The advantage of eager approach is clear, as we know more about user preferences, we can efficiently save the redundant computation for the final skyline. In contrast, when the information on user preferences is small, very few skylines can be finalized, and additional cost is thus mostly wasted.

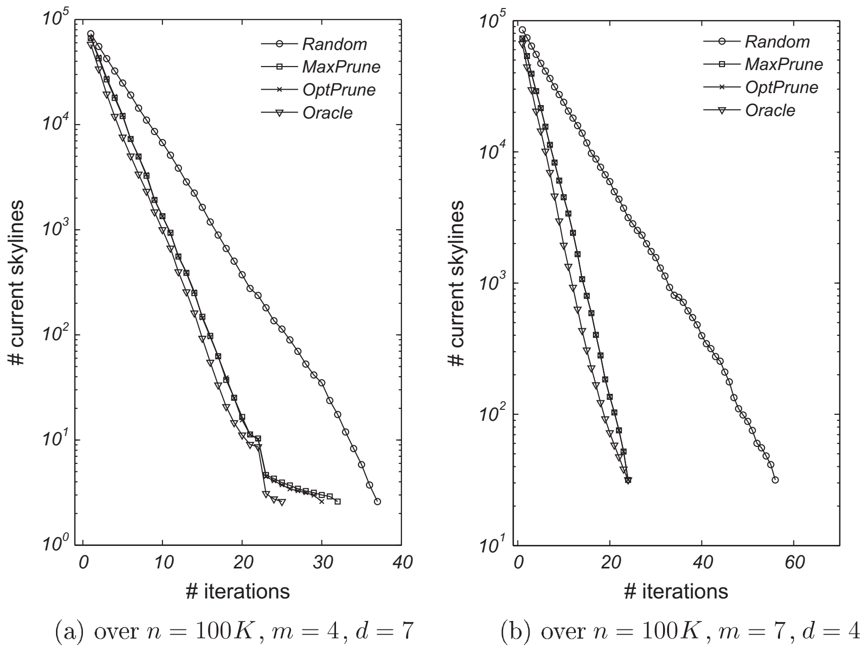


Fig. 6. Number of current skylines for each iteration.

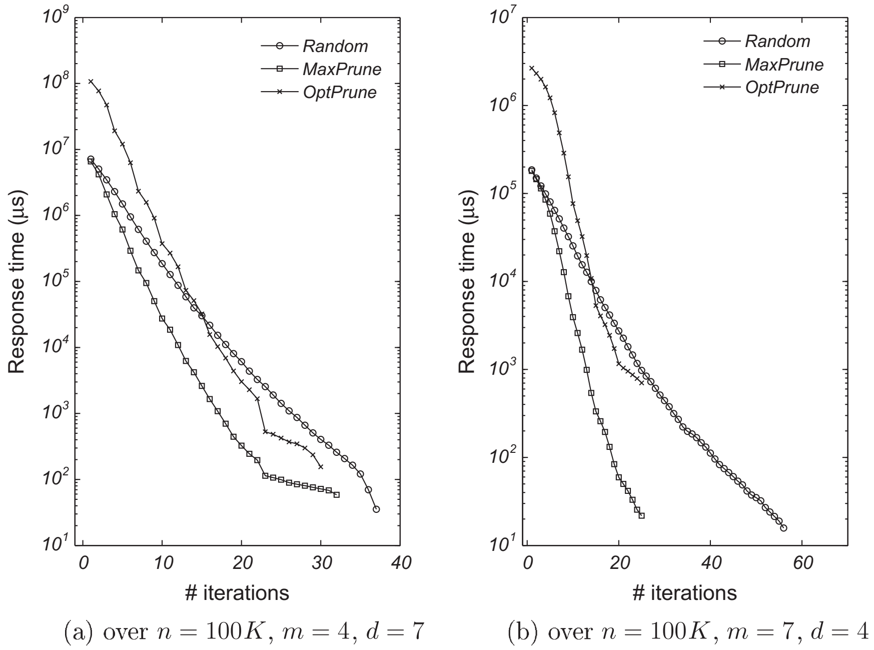


Fig. 7. Response time for each iteration.

We use the number of dominance comparisons as the measure, which mainly affects overall response time in CPU-intensive skyline algorithms [32,47].

Fig. 8 shows the trade-off between the two approaches. In early iterations, additional cost used in the eager approach to finalize skylines is mostly wasted, while the eager approach becomes more effective when we know more about user preferences (iterations >9). Based on this observation, we adopt lazy approach, as we assume users have very limited capability to specify their preferences (i.e., our target scenario corresponds to the early iterations).

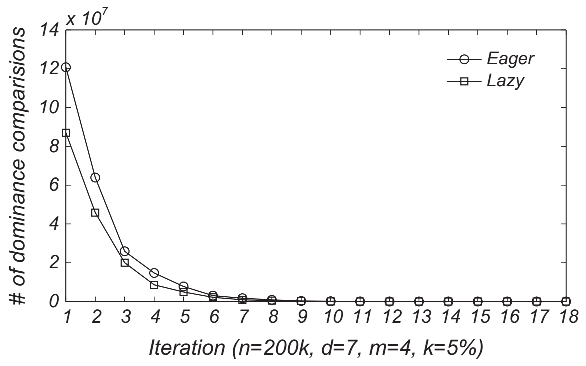


Fig. 8. Comparisons between two approaches.

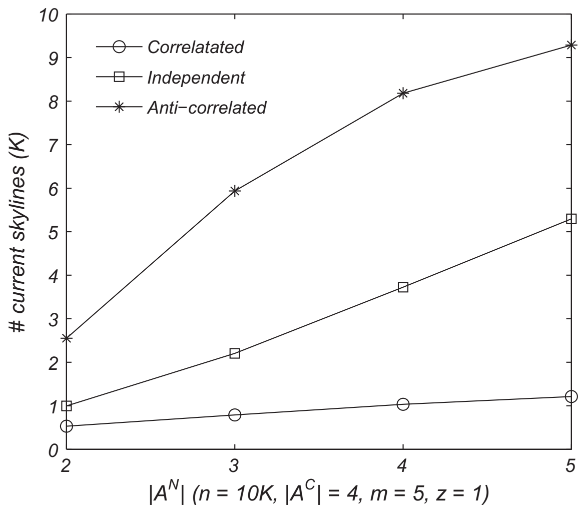


Fig. 9. Number of skylines over $|\mathcal{A}^N|$.

5.3.4. Effectiveness of pre-pruning procedure

This section evaluates the effectiveness of the pre-pruning procedure over generalized data settings. Fig. 9 reports the number of skylines over varying dimensionality for numerical attributes. We generate synthetic datasets including both categorical and numerical attributes over varying $|\mathcal{A}^N|$ and distributions, and keep the remaining parameters constant, i.e., $n = 10$ K, $|\mathcal{A}^C| = 4,\ m = 5,\ z = 1$. Note that the distributions of numerical attributes are independent of categorical attributes, and every numerical attribute has a domain of non-negative rational numbers, normalized to $[0,1]$.

Observe that our pre-pruning procedure is the most effective for correlated datasets, i.e., for every $|\mathcal{A}^N|$, we can prune out about 90 % objects without preference elicitation for categorical attributes. For the remaining attributes, the pruning power using numerical attributes highly depends on dimensionality, i.e., as the dimensionality increases, the number of pruning objects decreases. This implies that the size of skylines exponentially increases when $|\mathcal{A}^N|$ is high. Therefore, we can validate that our pre-pruning procedure can achieve a significant reduction when the size of skylines for only numerical attributes is small.

6. Related work

In this section, we first review research efforts on modeling user preferences. We then discuss existing skyline algorithms in numerical and categorical data, and distinguish our contributions from existing work.

6.1. Preference foundation

There has been a lot of research effort to represent user preferences. First, Kießling [26,27] proposed a preference model, which represents qualitative preferences as binary preference relations. Similarly, Chomicki [13,14] developed a preference model, which consists of a basic preference *winnow* operator and its combinators. These preference models showed that qualitative preference models are more intuitive than quantitative alternatives [19,25]. Furthermore, recent research has studied *incomplete* user preferences. Chen and Pu [12] surveyed user preference elicitation systems. In addition, Chomicki [15,16] showed how to adapt preference queries when user preferences are iteratively refined. However, neither work addressed how specifically to collect user-specific preferences to minimize user interaction.

6.2. Skyline computation over numerical domains

Skyline queries were first studied as *maximal vectors* [29] with theoretical analysis. Later, Börzsönyi et al. [7] introduced skyline queries in the database community, and proposed three basic skyline computations: block nested loop (BNL), divide-and-conquer (D& C), and B-tree-based algorithms. Next, Tan et al. [39] proposed *progressive* skyline computation using auxiliary structures such as bitmaps or sorted lists. Kossmann et al. [28] proposed the nearest neighbor (NN) algorithm, which iteratively partitions the data space with respect to the nearest neighbor object from the origin on the subspaces. Papadias et al. [36] developed the branch and bound skyline (BBS) algorithm, which improves the NN algorithm by achieving I/O optimality. Meanwhile, unlike BBS and NN leveraging R-tree indices, Chomicki et al. [17] developed the sort-filter-skyline (SFS) algorithm leveraging pre-sorted lists for efficient dominance tests. Godfrey et al. [21] proposed the linear elimination-sort algorithm for skyline (LESS) with attractive average-case asymptotic complexity of $O(d \cdot n)$ for d-dimensional space with n data points. Recently, Morse et al. [35] proposed the lattice-based skyline algorithm (LS), which computes the skyline over datasets with low-cardinality domains. Also, Lee et al. [32] proposed skyline computation using ZBtree, storing data based on a Z-order curve. More recently, Zhang et al. [47] proposed the object-based space partitioning skyline algorithm (OSPS) without pre-computed indices, which considers both dominance and incomparability.

Meanwhile, there have been active research efforts to understand and address the curse of dimensionality problem, i.e., the size of skyline increases exponentially by dimensionality d. The exact size was estimated as $\Theta((\ln n)^{d-1}/(d-1)!)$ for n data objects in [6], assuming distinct value condition and attribute independence. Recent work [11,20] confirmed the curse of dimensionality, in a more general setting without such assumptions. To address this problem, [24,37,38,44,46] studied data structures for materializing skylines in varying subspaces. First, Yuan et al. [46] proposed *skycube* structure, which can also be represented in a compacter form *compressed skycube* (CSC) structure [44]. Second, Pei et al. [37,38] studied how to represent each skyline with its *decisive subspace*. Recently, Kailasam et al. [24] proposed efficient skycube algorithms for datasets with low-cardinality domains.

Building upon these structures, some research work [9,10,30,40] proposed skyline ranking algorithms to identify the most relevant k skylines, using the following ranking metrics. Chan et al. [10] first proposed *skyline frequency*, measuring the number of subspaces that a point is a skyline point, and later proposed *k-dominant skylines* [9] which finds common skyline objects in all k-dimensional subspaces and ranks objects in the increasing order of k. While these two metrics generate the same ranking for all users, Lee et al. [30] proposed a personalized skyline ranking algorithm based on user-specific qualitative preferences. More recently, Vlachoua and Vazirgiannis [40] proposed a skyline ranking algorithm, which constructs a *skyline graph* using the dominance relationships between skyline objects for different subspaces.

6.3. Skyline computation over categorical domains

While most skyline algorithms focus mainly on numerical domains, recent work [18,45] has extended the applicability for categorical data. In the skyline literature, Chan et al. [8] proposed skyline computation over partially-ordered domains, i.e., incomplete preferences on categorical data. Balke et al. [3,4] studied how to identify skylines over qualitative partial orders and how to identify ideal partial results, or "prime cuts", supporting the notions of *equivalence* and *indifference*, and later inter-attribute equivalence as well [1]. Recently, Lofi et al. [33,34] proposed the *trade-off skyline*, which identifies more meaningful skylines using additional trade-off user preferences after computing a skyline for incomplete user preferences. However, these works do not consider how to obtain, or elicit, user preferences.

For such elicitation, user interface was explored in [2], though this work does not study effective question selection. Recently, Wong et al. [41–43] studied dynamic preferences on the categorical domain. Specifically, [42] first studied the task of mining a set of user preferences for each qualifying skyline object. Later, [41] studied skyline computation for qualitative user preference in the form of $a_i \succeq_i b_i \succeq_i *$, sharing similar motivation with our work. More recently, [43] proposed a *compressed ordered skyline tree* (CST) to efficiently support both [41,42]. In clear contrast, our framework, supporting any arbitrary partial orders, generalizes this problem, and enables users to elicit their preferences in an arbitrary incomplete order.

6.4. Our work

This paper focused on addressing efficient preference elicitation for skyline queries on generalized data environments. While our preliminary work [31] focused on an optimal elicitation algorithm in a restricted problem setting, this paper extended the proposed method for a general problem setting without such restriction, and discussed its efficient implementation in detail. Also, we extended the applicability of the proposed algorithms for datasets including both numerical and categorical attributes.

7. Conclusions

This paper studied a preference elicitation method for skyline queries. To minimize user interaction, we introduced the notion of the expected skyline reduction, based on which we collected user feedback that maximizes skyline reduction size. We proved that this procedure guarantees step-optimality. In addition, we extended our framework to generalized datasets including both categorical and numerical attributes. Lastly, we extensively validated our proposed algorithms over varying experimental settings, and demonstrated the effectiveness and efficiency of our proposed algorithms.

Acknowledgements

This work was supported by the IT R&D program of MKE/KEIT [10041244, SmartTV 2.0 Software Platform].

References

[1] Wolf-Tilo Balke, Ulrich Güntzer, Christoph Lofi, Eliciting matters – controlling skyline sizes by incremental integration of user preferences, in: International Conference on Database Systems for Advanced Applications, 2007, pp. 551–562.
[2] Wolf-Tilo Balke, Ulrich Güntzer, Christopher Lofi, User interaction support for incremental refinement of preference-based queries, in: IEEE International Conference on Research Challenges in Information Science, 2007, pp. 209–220.
[3] Wolf-Tilo Balke, Ulrich Güntzer, Wolf Siberski, Exploiting indifference for customization of partial order skylines, in: IEEE International Database Engineering and Applications Symposium, 2006, pp. 80–88.
[4] Wolf-Tilo Balke, Ulrich Güntzer, Wolf Siberski, Getting prime cuts from skylines over partially ordered domains, in: Datenbanksysteme in Business, Technologie und Web, 2007, pp. 64–81.
[5] Wolf-Tilo Balke, Ulrich Güntzer, Jason Xin Zheng, Efficient distributed skylining for web information systems, in: International Conference on Extending Database Technology, 2004, pp. 256–273.
[6] Jon Louis Bentley, H.T. Kung, Mario Schkolnick, Clark D. Thompson, On the average number of maxima in a set of vectors and applications, Journal of the Association for Computing Machinery 25 (4) (1978) 536–543.
[7] Stephan Börzsönyi, Donald Kossmann, Konrad Stocker, The skyline operator, in: IEEE International Conference on Data Engineering, 2001, pp. 421–430.
[8] Chee-Yong Chan, Pik-Kwang Eng, Kian-Lee Tan, Stratified computation of skylines with partially-ordered domains, in: ACM SIGMOD International Conference on Management of Data, 2005, pp. 203–214.
[9] Chee-Yong Chan, H.V. Jagadish, Kian-Lee Tan, Anthony K.H. Tung, Zhenjie Zhang, On high dimensional skylines, in: International Conference on Extending Database Technology, 2006, pp. 478–295.
[10] Chee-Yong Chan, H.V. Jagadish, Kian-Lee Tan, Authony K.H. Tung, Zhenjie Zhang, Finding k-dominant skyline in high dimensional space, in: ACM SIGMOD International Conference on Management of Data, 2006, pp. 503–514.
[11] Surajit Chaudhuri, Nilesh Dalvi, Raghav Kaushik, Robust cardinality and cost estimation for skyline operator, in: IEEE International Conference on Data Engineering, 2006, pp. 64–73.
[12] Li Chen, Pearl Pu, Survey of preference elicitation methods, in: EPFL Technical Report, 2004.
[13] Jan Chomicki, Querying with intrinsic preferences, in: International Conference on Extending Database Technology, 2002, pp. 34–51.
[14] Jan Chomicki, Preference formulas in relational queries, ACM Transactions on Database Systems 28 (4) (2003) 427–466.
[15] Jan Chomicki, Iterative modification and incremental evaluation of preference queries, in: Foundations of Information and Knowledge Systems, 2006, pp. 63–82.
[16] Jan Chomicki, Database querying under changing preferences, Annals of Mathematics and Artificial Intelligence 50 (1–2) (2007) 79–109.
[17] Jan Chomicki, Parke Godfery, Jarek Gryz, Dongming Liang, Skyline with presorting, in: IEEE International Conference on Data Engineering, 2003, pp. 717–719.
[18] Josep Domingo-Ferrer, Agusti Solanas, A measure of variance for hierarchical nominal attributes, Information Sciences 178 (24) (2008) 4644–4655.
[19] Peter C. Fishburn, Utility Theory for Decision Making, Number 18 in Publications in Operations Research, Wiley, 1970.
[20] Parke Godfrey, Skyline cardinality for relational processing, in: Foundations of Information and Knowledge Systems, 2004, pp. 78–97.
[21] Parke Godfrey, Ryan Shipley, Jarek Gryz, Maximal vector computation in large data sets, in: International Conference on Very Large Data Bases, 2005, pp. 229–240.
[22] Jim Gray, Adam Bosworth, Andrew Layman, Hamid Pirahesh, Data cube: a relational aggregation operator generalizing group-by, cross-tab, and sub-totals, in: IEEE International Conference on Data Engineering, 1996, pp. 152–159.
[23] Sven Ove Hansson, What is ceteris paribus preference?, Journal of Philosophical Logic 25 (3) (1996) 307–332
[24] Gayathri Tambaram Kailasam, Jinseung Lee, Jae-Won Rhee, Jaewoo Kang, Efficient skycube computation using point and domain-based filtering, Information Sciences 180 (7) (2010) 1090–1103.
[25] Ralph L. Keeney, Howard Raiffa, Decisions with Multiple Objectives Preferences and Value Tradeoffs, Wiley, 1976.
[26] Werner Kießling, Foundations of preferences in database systems, in: International Conference on Very Large Data Bases, 2002, pp. 311–322.
[27] Werner Kießling, Gerhard Köstler, Preference SQL: design, implementation, experience, in: International Conference on Very Large Data Bases, 2002, pp. 990–1001.
[28] Donald Kossmann, Frank Ramsak, Steffen Rost, Shooting stars in the sky: an online algorithm for skyline queries, in: International Conference on Very Large Data Bases, 2002, pp. 275–286.
[29] H.T. Kung, F. Luccio, F.P. Preparata, On finding the maxima of a set of vectors, Journal of the Association for Computing Machinery 22 (4) (1975) 469–476.
[30] Jongwuk Lee, Gae-won You, Seung-won Hwang, Personalized top-k skyline queries in high-dimensional space, Information Systems 34 (1) (2009) 45–61.
[31] Jongwuk Lee, Gae-won You, Seung-won Hwang, Joachim Selke, Wolf-Tilo Balke, Optimal preference elicitation for skyline queries over categorical domains, in: Database and Expert Systems Applications, 2008, pp. 610–624.
[32] Ken C.K. Lee, Baihua Zheng, Huajing Li, Wang-Chien Lee, Approaching the skyline in z order, in: International Conference on Very Large Data Bases, 2007, pp. 279–290.
[33] Christoph Lofi, Wolf-Tilo Balke, Ulrich Güntzer, Efficiently performing consistency checks for multi-dimensional preference trade-offs, in: IEEE International Conference on Research Challenges in Information Science, 2008, pp. 271–278.
[34] Christoph Lofi, Ulrich Güntzer, Wolf-Tilo Balke, Efficient computation of trade-off skylines, in: International Conference on Extending Database Technology, 2010, pp. 597–608.
[35] Michael Morse, Jignesh M. Patel, H.V. Jagadish, Efficient skyline computation over low-cardinality domains, in: International Conference on Very Large Data Bases, 2007, pp. 267–278.
[36] Dimitris Papadias, Yufei Tao, Greg Fu, Bernhard Seeger, An optimal and progressive algorithm for skyline queries, in: ACM SIGMOD International Conference on Management of Data, 2003, pp. 467–478.

[37] Jian Pei, Ada Wai-Chee Fu, Xuemin Lin, Haixun Wang, Computing compressed multidimensional skyline cubes efficiently, in: IEEE International Conference on Data Engineering, 2007, pp. 96–105.
[38] Jian Pei, Wen Jin, Martin Ester, Yufei Tao, Catching the best views of skyline: a semantic approach based on decisive subspaces, in: International Conference on Very Large Data Bases, 2005, pp. 253–264.
[39] Kian-Lee Tan, Pin-Kwang Eng, Beng Chin Ooi, Efficient progressive skyline computation, in: International Conference on Very Large Data Bases, 2001, pp. 301–310.
[40] Akrivi Vlachoua, Michalis Vazirgiannis, Ranking the sky: discovering the importance of skyline points through subspace dominance relationships, Data and Knowledge Engineering 69 (9) (2010) 943–964.
[41] Raymond Chi-Wing Wong, Ada Wai-Chee Fu, Jian Pei, Yip Sing Ho, Tai Wong, Yubao Liu, Efficient skyline querying with variable user preferences on nominal attributes, in: International Conference on Very Large Data Bases, 2008, pp. 1032–1043.
[42] Raymond Chi-Wing Wong, Jian Pei, Ada Wai-Chee Fu, Ke Wang, Mining favorable facets, in: ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2007, pp. 804–813.
[43] Raymond Chi-Wing Wong, Jian Pei, Ada Wai-Chee Fu, Ke Wang, Online skyline analysis with dynamic preferences on nominal attributes, IEEE Transactions on Knowledge and Data Engineering 21 (1) (2009) 35–49.
[44] Tian Xia, Donghui Zhang, Refreshing the sky: the compressing skycube with efficient support for frequent updates, in: ACM SIGMOD International Conference on Management of Data, 2006, pp. 491–502.
[45] Gae-won You, Seung-won Hwang, Hwanjo Yu, Supporting personalized ranking over categorical attributes, Information Sciences 178 (18) (2008) 3510–3524.
[46] Yidong Yuan, Xuemin Lin, Qing Liu, Wei Wang, Jeffery Xu Yu, Qing Zhang, Efficient computation of the skyline cube, in: International Conference on Very Large Data Bases, 2005, pp. 241–252.
[47] Shiming Zhang, Nikos Mamoulis, David W. Cheung, Scalable skyline computation using object-based space partitioning, in: ACM SIGMOD International Conference on Management of Data, 2009, pp. 483–494.