Discovering Relative Importance of Skyline Attributes

Denis Mindolin
Dept. of Computer Science and Engineering
University at Buffalo
Buffalo, NY 14260-2000
mindolin@cse.buffalo.edu

Jan Chomicki
Dept. of Computer Science and Engineering
University at Buffalo
Buffalo, NY 14260-2000
chomicki@cse.buffalo.edu

ABSTRACT

Querying databases with preferences is an important research problem. Among various approaches to querying with preferences, the skyline framework is one of the most popular. A well known deficiency of that framework is that all attributes are of the same importance in skyline preference relations. Consequently, the size of the results of skyline queries may grow exponentially with the number of skyline attributes. Here we propose the framework called *p-skylines* which enriches skylines with the notion of *attribute importance*. It turns out that incorporating relative attribute importance in skylines allows for reduction in the corresponding query result sizes. We propose an approach to discovering importance relationships of attributes, based on user-selected sets of superior and inferior examples. We show that the problem of checking the existence of and the problem of computing an optimal p-skyline preference relation covering a given set of examples are NP-complete and FNP-complete, respectively. However, we also show that a restricted version of the discovery problem – using only superior examples to discover attribute importance – can be solved efficiently in polynomial time. Our experiments show that the proposed importance discovery algorithm has high accuracy and good scalability.

1. INTRODUCTION

Efficient user preference management is a crucial part of any successful sales-oriented business. Knowing *what* customers like and more importantly *why* they like that and what they *will* like in the future is an essential part of the modern risk management process.

One of the most popular preference handling models is *preferences as skyline relations* [2]. The skyline preference relation is defined on top of a set of preferences over individual attributes. It represents the *Pareto improvement* principle: *a tuple o_1 is preferred to a tuple o_2 iff o_1 is as good as o_2 in all the attributes, and o_1 is strictly better than o_2 in at least one attribute.* The set of the most preferred tuples according to this principle is called a *skyline*.

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the VLDB copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Very Large Data Base Endowment. To copy otherwise, or to republish, to post on servers or to redistribute to lists, requires a fee and/or special permission from the publisher, ACM.
VLDB '09, August 24-28, 2009, Lyon, France
Copyright 2009 VLDB Endowment, ACM 000-0-00000-000-0/00/00.

Example 1 *Assume we have the following five cars on sale.*

id	make	price	year
t_1	*ford*	*30k*	*2007*
t_2	*bmw*	*45k*	*2008*
t_3	*kia*	*20k*	*2007*
t_4	*ford*	*40k*	*2008*
t_5	*bmw*	*50k*	*2006*

Assume also Mary wants to buy a car and her preferences over automobile attributes are as follows.

$\succ_{make}$	*BMW is better than Ford, Ford is better than Kia*
$\succ_{year}$	*the car should be as new as possible*
$\succ_{price}$	*the car should be as cheap as possible*

This skyline preference relation is denoted by $\succ_1$. Then the corresponding skyline *is*

id	make	price	year
t_1	*ford*	*30k*	*2007*
t_2	*bmw*	*45k*	*2008*
t_3	*kia*	*20k*	*2007*
t_4	*ford*	*40k*	*2008*

An important deficiency of the skyline framework is its inability to represent differences in the relative importance of attributes. In real life scenarios, it is often the case that benefits in one attribute may outweigh losses in one or more attributes. For instance, given cars that differ in age and price, for some people the age is crucial while the price is secondary. Therefore, the price has to be considered only when benefits in age cannot be obtained, i.e., when the ages are equal. However, according to the Pareto improvement principle, the age and the price are of equal importance and thus have to be considered together when comparing tuples.

Another drawback of the skyline framework is that the size of a skyline may be exponential in the number of attribute preferences involved [10]. This computational issue is caused by the looseness of the Pareto improvement principle. *Pareto improvement* implies that if a tuple o_1 is better than o_2 in one attribute, then the existence of any attribute in which o_2 is better than o_1 makes the tuples *incomparable*. Thus, every additional attribute increases the number of such incomparable tuples. It has been shown that skyline sizes become unmanageable for large data sets [1].

Example 2 *Assume that Mary decides that* year *is more important than* make *and* price*, which in turn are equally important. Thus, regardless of the values of* make *and* price*, a newer car is always better than an old one. At the same time, given two cars of the same age, one needs to compare their* make *and* price *to determine the better one. Denote*

this preference relation as $\succ_2$. *The most preferred tuples according to* $\succ_2$ *are*

id	make	price	year
t_2	*bmw*	*45k*	*2008*
t_4	*ford*	*40k*	*2008*

Namely, t_2 *and* t_4 *are better than all other tuples in* `year`*,* t_2 *is better than* t_4 *in* `make`*, and* t_4 *is better than* t_2 *in* `price`.

Here we propose the *p-skyline* framework which generalizes the skyline framework, and addresses its computational and semantical limitations. The skyline semantics is enriched with the notion of *attribute importance* in a natural way. Given an attribute A being more important then an attribute B, we assume that a tuple with a better value of A is *unconditionally* preferred to all tuples with worse values of A, regardless of their values of B. However, given a tuple with the same value of A, the one with a better value of B is preferred. When dealing with equally important attributes, we use the Pareto improvement principle. Therefore, skyline queries are also representable in this framework.

It turns out that adding attribute importance to skyline relations makes more tuples comparable. This results in a significant reduction in the size of the corresponding query results. Therefore, incorporating attribute importance into skyline relations allows not only to model user preferences more accurately but also to make the size of the corresponding query results more manageable.

So far, we have illustrated only the drawbacks of the skyline semantics. However, one of its main properties, which has made it so popular, is the simplicity of representing preferences. Namely, one needs to provide only a set of atomic preferences to specify the corresponding preference query. For p-skylines, an additional piece of information – the relative importance of the attributes – has to be provided by the user. However, requiring users to describe attribute importance explicitly seems impractical for several reasons. First, the number of required comparisons may be rather large: one has to compare all attributes pairwise. The second issue is even more serious – users themselves may be not fully aware of their own preferences.

The major contribution of our paper is an alternative approach to discovering attribute importance relationships, based on *user feedback*. The type of feedback we discuss here consists of two sets of tuples belonging to a given set: *superior examples* [14], i.e., the *desirable* tuples, and *inferior examples* [14] i.e., the *undesirable* tuples. Given the sets of superior and inferior examples, we consider the problem of the existence of a p-skyline relation which covers the corresponding example sets. The next problem we tackle is computing a p-skyline relation that covers the example sets and *fits user preferences optimally*. We show that the existence problem is `NP`-complete in general, and the problem of computing optimal p-skyline relations is `FNP`-complete.

However, restricted versions of these problems – using only superior examples to discover p-skyline relations – bring computational benefits. First, we show a polynomial-time approach for checking the existence of a p-skyline relation covering a given set of superior examples. Second, we provide a polynomial time algorithm for computing p-skyline relations which optimally fit user preferences. We also provide techniques for improving the algorithm performance. The experimental evaluation of the algorithms on real-life and synthetic data sets demonstrates their high accuracy and scalability. Proofs of the results are provided in [20].

2. BASIC NOTATIONS

Below we describe some concepts of a variant of the binary relation preference framework [6] which we adopt here.

Let $\mathcal{A} = \{A_1, ..., A_n\}$ be a finite set of attributes (a relation schema). Let every attribute $A_i \in \mathcal{A}$ be associated with an *infinite domain* $\mathcal{D}_{A_i}$. The domains considered here are rationals and uninterpreted constants (numerical or categorical). Let $\mathcal{U} = \prod_{A_i \in \mathcal{A}} \mathcal{D}_{A_i}$. Given a tuple $o \in \mathcal{U}$, we denote the value of its attribute A_i as $o.A_i$.

Definition 1 *Let* A *be an attribute from the set* $\mathcal{A}$*. Then* an atomic preference relation over A *is a* total order $\succ_A$ *which is a subset of* $\mathcal{D}_A \times \mathcal{D}_A$.

Definition 2 *A* preference relation *is a* strict partial order (SPO) *binary relation which is a subset of* $\mathcal{U} \times \mathcal{U}$. *A* preference formula *is a quantifier-free first-order formula with two free tuple variables, built from comparisons* $(=, \neq, >, \geq)$. *Given a preference formula* f*, the preference relation it represents is denoted as* $\succ_f$.

Notice that we do not require preference relations to be finite. An example of a preference formula is

$$o_1 \succ_1 o_2 \equiv o_1.\texttt{year} \geq o_2.\texttt{year} \wedge o_1.\texttt{price} \leq o_2.\texttt{price} \wedge$$
$$(o_1.\texttt{make} = BMW \wedge o_2.\texttt{make} = Ford \vee o_1.\texttt{make} = Ford \wedge$$
$$o_2.\texttt{make} = Kia \vee o_1.\texttt{make} = BMW \wedge o_2.\texttt{make} = Kia \vee$$
$$o_1.\texttt{make} = o_2.\texttt{make}) \wedge (o_1.\texttt{year} \neq o_2.\texttt{year} \vee$$
$$o_1.\texttt{price} \neq o_2.\texttt{price} \vee o_1.\texttt{make} \neq o_2.\texttt{make})$$

representing $\succ_1$ from Example 3.

Working with preferences, the two most common tasks are 1) *dominance testing*: checking if a tuple is preferred to another one, according to a given preference relation, 2) *computing the most preferred tuples* in a given set, according to a given preference relation. The former problem is solved easily here by checking if the formula representing the preference relation evaluates to true for the given pair of tuples. To deal with the latter problem, the *winnow* relational algebra operator has been proposed [6, 15].

Definition 3 *Let* $\succ$ *be preference relation over* $\mathcal{U}$*. Then the* winnow *operator is written as* $w_\succ(\mathcal{A})$*, and for every instance* r *of* $\mathcal{A}$ *(i.e., a subset of* $\mathcal{U}$*):*

$$w_\succ(r) = \{t \in r \mid \neg\exists t' \in r \ . \ t' \succ t\},$$

2.1 p-skyline relations

Here we describe an extension of the skyline framework [2]. Let $\mathcal{H}$ be the set containing the atomic preference relation $\succ_A$ for every attribute $A \in \mathcal{A}$. Preferences in this framework are combined using *Pareto accumulation* [15] (which represents equal importance of the preference relations being combined) and *prioritized accumulation* [15] (which represents different importance of the preference relations being combined). A *preference relation* combined using these operators is called *a prioritized skyline preference relation* or simply a *p-skyline relation*.

Definition 4 *For a subset* S *of* $\mathcal{A}$*, let the relation* I_S *be the set of all pairs of tuples whose values of the attributes in* S *are equal, i.e.,*

$$I_S = \{(o, o') \mid o, o' \in \mathcal{U} \ \wedge \forall X \in S \ . \ o.X = o'.X\}.$$

Let also $Var(\succ)$ *be the set of all relevant attributes. A preference relation* $\succ$ *is* a p-skyline relation *if one of the following holds:*

1. $\succ$ *is induced by an* atomic *preference relation* $\succ_A \in \mathcal{H}$

$$\succ \equiv \{(o, o') \mid o, o' \in \mathcal{U} \ . \ o.A \succ_A o'.A\}.$$

Then $Var(\succ) = \{A\}$.

2. $\succ$ *is a* prioritized accumulation $\succ_{c1}$ & $\succ_{c2}$ *of p-skyline relations* $\succ_{c1}$ *and* $\succ_{c2}$*, defined as*

$$\succ \equiv \{(o, o') \mid o, o' \in \mathcal{U} \ . \ o \succ_{c1} o' \vee I_{Var(\succ_{c1})}(o, o') \wedge o \succ_{c2} o'\},$$

where $Var(\succ_{c1}) \cap Var(\succ_{c2}) = \emptyset$. *Then* $Var(\succ) = Var(\succ_{c1}) \cup Var(\succ_{c2})$.

3. $\succ$ *is a* Pareto accumulation $\succ_{c1} \otimes \succ_{c2}$ *of p-skyline relations* $\succ_{c1}$ *and* $\succ_{c2}$*, defined as*

$$\succ \equiv \{(o, o') \mid o, o' \in \mathcal{U} \ . \ o \succ_{c1} o' \wedge I_{Var(\succ_{c2})}(o, o') \vee o \succ_{c2} o' \wedge I_{Var(\succ_{c1})}(o, o') \vee o \succ_{c1} o' \wedge o \succ_{c2} o'\},$$

where $Var(\succ_{c1}) \cap Var(\succ_{c2}) = \emptyset$. *Then* $Var(\succ) = Var(\succ_{c1}) \cup Var(\succ_{c2})$.

According to this definition, with each p-skyline relation we associate the set of relevant attributes Var which are used to compose the entire relation. It follows from the definition that every atomic preference may appear only once in a p-skyline relation definition. [16] shows that p-skyline relations are preference relations, i.e., strict partial orders.

Proposition 1 *[15] The operators* $\otimes$ *and* & *are associative. The operator* $\otimes$ *is symmetric.*

Since the accumulation operators are associative, we extend them from binary to n-ary operators. Denote the set of all p-skyline relations, each composed from *all members of* $\mathcal{H}$, by $\mathcal{F}_{\mathcal{H}}$. Such relations are called *full p-skyline relations*. All p-skyline relations we deal with further are considered to be full. It is easy to see that the *skyline* preference relation (denoted as $sky_{\mathcal{H}}$) is a p-skyline relation constructed as a Pareto accumulation of the atomic preferences $\mathcal{H}$. Thus, p-skylines are indeed a generalization of skylines.

Example 3 *The preference relations* $\succ_1$ *and* $\succ_2$ *from Examples 1 and 2, resp., can be represented as the following p-skyline relations*

$$\succ_1 = \succ_{year} \otimes \succ_{price} \otimes \succ_{make},$$
$$\succ_2 = \succ_{year} \ \& \ (\succ_{price} \otimes \succ_{make}).$$

The main difference between the skyline and the p-skyline frameworks is the ability to represent relative importance of atomic preferences (or the corresponding attributes) in the latter one. It is intuitive to represent a p-skyline relation as a directed graph showing relative importance of the atomic preferences used to compose the p-skyline relation. We call such graphs *p-graphs*. Their nodes are members of the set of attributes $\mathcal{A}$, and directed edges go from more important to less important attributes. Formally, p-graphs are defined as follows.

Definition 5 *Let* $\succ$ *be a p-skyline relation. Then the p-graph representing* $\succ$ *(denoted as* $\Gamma_\succ$*) is a binary relation over* $\mathcal{A}$*. The sets of nodes* $N(\Gamma_\succ)$ *and edges* $E(\Gamma_\succ)$ *of* $\Gamma_\succ$ *are defined inductively as follows:*

1. *if* $\succ$ *is induced by an atomic preference over* A*, then* $N(\Gamma_\succ) = \{A\}$, $E(\Gamma_\succ) = \emptyset$;
2. *if* $\succ = (\succ_2 \ \& \ \succ_2)$*, then* $N(\Gamma_\succ) = N(\Gamma_{\succ_1}) \cup N(\Gamma_{\succ_2})$*, and* $E(\Gamma_\succ) = E(\Gamma_{\succ_1}) \cup E(\Gamma_{\succ_2}) \cup \{(A, B) \mid A \in N(\Gamma_{\succ_1}), B \in N(\Gamma_{\succ_2})\}$;
3. *if* $\succ = (\succ_1 \otimes \succ_2)$*, then* $N(\Gamma_\succ) = N(\Gamma_{\succ_1}) \cup N(\Gamma_{\succ_2})$*, and* $E(\Gamma_\succ) = E(\Gamma_{\succ_1}) \cup E(\Gamma_{\succ_2})$.

Example 4 *The relations* $\succ_1$ *and* $\succ_2$ *(Example 3) are p-skyline relations, and their p-graphs are shown in Figure 1.*

(a) $\Gamma_{\succ_1}$ (b) $\Gamma_{\succ_2}$

Figure 1: p-graphs of $\succ_1$ and $\succ_2$

Given a p-graph $\Gamma_\succ$ of a p-skyline relation $\succ$, denote the set of all children of a node $A \in \mathcal{A}$ as $Ch_{\Gamma_\succ}(A)$. We denote an edge from X to Y in $\Gamma_\succ$ as $\Gamma_\succ \models X \to Y$.

It turns out that p-graphs not only naturally capture the notion of relative attribute importance induced by p-skyline relations, but also can be efficiently used to describe essential properties of such relations. We use these properties extensively in the study of the problem of p-skyline relation discovery discussed further on. The following theorem shows how to check if a directed graph is a p-graph.

THEOREM 1 `(SPO+Envelope)`
A directed graph Γ *with the node set* $N(\Gamma) = \mathcal{A}$ *is a p-graph of some p-skyline relation iff*

1. Γ *is transitive and irreflexive, i.e., a strict partial order* `(SPO)`*, and*
2. Γ *satisfies the* `Envelope` *property:*

$$\forall A, B, C, D \in \mathcal{A}, \text{ all different}$$
$$\Gamma \models A \to B \wedge \ \Gamma \models C \to D \wedge \Gamma \models C \to B \Rightarrow$$
$$\Gamma \models C \to A \vee \Gamma \models A \to D \vee \Gamma \models D \to B$$

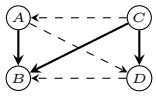


Figure 2: The `Envelope` property

The graph properties shown in Theorem 1 are justified as follows. Recall that an edge in a p-graph captures the difference in the importance of two atomic preferences. Thus, irreflexivity of p-graphs implies that an atomic preference relation cannot be more important than itself. Transitivity of p-graphs captures the transitivity of the importance relationship. The `Envelope` property follows from the fact that each atomic preference relation can have only one occurrence in the definition of a p-skyline preference relation.

The p-skyline relation corresponding to a p-graph Γ can be constructed by a recursive decomposition of Γ into pairs of disjoint subgraphs (Pareto accumulation) or pairs of subgraphs in which every node of the first one has an outgoing edge to every node of the second one (prioritized accumulation).

The next theorem shows that checking the containment of full p-skyline relations can be reduced to the problem of checking the containment of the corresponding p-graphs.

THEOREM 2 *Let* $\succ_1, \succ_2$ *be two members of* $\mathcal{F}_{\mathcal{H}}$ *. Then*

1. $\succ_1 = \succ_2 \ \Leftrightarrow \ E(\Gamma_{\succ_1}) = E(\Gamma_{\succ_2})$
2. $\succ_1 \subset \succ_2 \ \Leftrightarrow \ E(\Gamma_{\succ_1}) \subset E(\Gamma_{\succ_2})$.

The graph $\Gamma_{sky_{\mathcal{H}}}$, containing no edges, is the least among all p-graphs corresponding to p-skyline relations in $\mathcal{F}_{\mathcal{H}}$. Theorem 2 implies that $sky_{\mathcal{H}}$ is the least among all p-skyline relations in $\mathcal{F}_{\mathcal{H}}$ viewed as sets of pairs.

2.2 p-skyline query evaluation

For completeness, we outline here some methods of p-skyline query evaluation. We have shown that the p-skyline framework is a generalization of the framework of skylines. Therefore, some skyline algorithms can be adopted to evaluate p-skyline queries: BNL [2] and SFS [7]. The former algorithm is suitable for any preference relation, and thus can be applied to p-skyline relations as well. To apply the latter algorithm, one needs to construct a weak order of tuples compatible with the p-skyline relation. This can be done by constructing a p-graph $T_{\succ'}$ which is a total order of attributes containing the original p-graph $\Gamma_\succ$. By Theorem 2, $\succ \subseteq \succ'$, and it is easy to check that $\succ'$ is a weak order.

The definition of p-skyline relations is based on the *accumulation operators* which are the foundation of the Preference SQL language [17]. Thus, to evaluate a p-skyline query, one could convert it to a Preference SQL statement, using certain *base preference constructors* to represent the atomic preference relations. A number of optimization techniques have been developed for such queries [12].

3. P-SKYLINE DISCOVERY

In this section, we focus on the problem of discovering p-skyline preference relations, given user feedback. This problem can be decomposed into two subproblems. The first consists of discovering the *relevant attributes* in $\mathcal{A}$ and the corresponding atomic preference relations. The second problem consists of discovering the *structure* of the p-skyline relation that captures user preferences, given a set of atomic preference relations. Namely, we need to find an appropriate combination of the accumulation operators needed to construct the appropriate p-skyline relation. Here we focus on the second problem. Some methods of solving the first problem are discussed in Section 5.

3.1 Discovering relative attribute importance

Preference discovery is based on some form of feedback from a target user. We use the following scenario [14]. Given a finite set of tuples $\mathcal{O} \subseteq \mathcal{U}$, a user has to partition it into disjoint subsets: the set G of tuples she confidently likes (*superior* examples), the set W of tuples she confidently dislikes (*inferior* examples), and the set of remaining tuples about which she is not sure. We assume here that if a tuple o is inferior, then there is at least one superior example which the user likes more than o. This assumption is justified by a general principle that a user considers something bad because she knows of a better alternative.

Now having subsets G and W of a set of tuples $\mathcal{O}$, we want to find a possible p-skyline relation which implies G being superior and W being inferior examples in $\mathcal{O}$. Formally, we want to compute $\succ \in \mathcal{F}$ such that 1) $G \subseteq w_\succ(\mathcal{O})$ (i.e., the tuples in G are among the most preferred tuples in $\mathcal{O}$ according to $\succ$), 2) for every tuple o' in W, there is a tuple o in G such that $o \succ o'$ (i.e., o' is an inferior example). Such a p-skyline relation $\succ$ is called *favoring G and disfavoring W in $\mathcal{O}$*.

Thus, the first problem we consider is the existence of favoring/disfavoring p-skyline relations.

Problem 1. *Given a set of atomic preference relations $\mathcal{H}$, a set of superior examples G, and a set of inferior examples W in a set $\mathcal{O}$, determine if there exists a p-skyline relation $\succ \in \mathcal{F}_\mathcal{H}$ favoring G and disfavoring W in $\mathcal{O}$.*

Example 5 *Take the car inventory data and the atomic preferences $\mathcal{H} = \{\succ_{make}, \succ_{year}, \succ_{price}\}$ from Example 1. Assume $G = \{t_4\}$, $W = \{t_3\}$. We discover $\succ \in \mathcal{F}_\mathcal{H}$ favoring G and disfavoring W. First, $\succ_{make}$ cannot be more important than all other atomic preferences since then t_2 and t_5 dominate t_4, and thus t_4 is not superior. Moreover, $\succ_{price}$ cannot be more important than the other atomic preferences because then t_3 and t_1 dominate t_4. However, if $\succ_{year}$ is more important than the other atomic preferences, then t_4 dominates t_3 and no other tuple dominates t_4 in $\succ_{year}$. At the same time, both t_2 and t_4 are the best according to $\succ_{year}$, but t_2 may dominate t_4 in $\succ_{make}$. Therefore, $\succ_{make}$ should not be more important than $\succ_{price}$. Thus, the following p-skyline relation favors G and disfavors W in $\mathcal{O}$*

$$\succ_2 = \succ_{year} \ \& \ (\succ_{price} \otimes \succ_{make})$$

The set of the best tuples in $\mathcal{O}$ according to $\succ_2$ is $\{t_2, t_4\}$.

Generally there may be zero or more members of $\mathcal{F}_\mathcal{H}$ favoring G and disfavoring W in $\mathcal{O}$. When more than one of them exist, we pick a maximal one (in the set theoretic sense). Larger preference relations imply more dominated tuples and fewer most preferred ones. Consequently, the result of $w_\succ(\mathcal{O})$ gets more manageable due to its decreasing size. Moreover, maximizing $\succ$ corresponds to minimizing $w_\succ(\mathcal{O}) - G$, which implies more precise correspondence of $\succ$ to the real user preferences. A maximal p-skyline relation favoring G and disfavoring W in $\mathcal{O}$ is called *optimal*.

Thus, the second problem considered here is computing an optimal p-skyline relation favoring G and disfavoring W.

Problem 2. *Given a set of atomic preference relations $\mathcal{H}$, a set of superior examples G, and a set of inferior examples W in a set $\mathcal{O}$, discover an optimal p-skyline relation $\succ \in \mathcal{F}_\mathcal{H}$ favoring G and disfavoring W in $\mathcal{O}$.*

Example 6 *Take G, W, and $\succ_2$ from Example 5. Note that in order to make t_4 dominate t_2, we need to make* `price` *more important than* `year`*. As a result, the relation*

$$\succ_3 = \succ_{year} \ \& \ \succ_{price} \ \& \ \succ_{make}$$

also favors G and disfavors W in $\mathcal{O}$, but the set of best tuples in $\mathcal{O}$ according to $\succ_3$ is $\{t_4\}$. Moreover, $\succ_3$ is optimal*. This is because no other p-skyline relation favoring G and disfavoring W contains $\succ_3$ since the p-graph of $\succ_3$ is a total order of* {`year`, `price`, `make`} *and thus is a maximal SPO.*

Below we show the constraints which should be satisfied by the p-graph $\Gamma_\succ$ of a p-skyline relation $\succ$ favoring G and disfavoring W in $\mathcal{O}$. Let us consider the notion of favoring G in $\mathcal{O}$ first. For any member $o' \in G$ to be in the set of the most preferred tuples of $\mathcal{O}$, o' must not be dominated by any tuple in $\mathcal{O}$. That is,

$$\forall o \in \mathcal{O}, o' \in G \ . \ o \not\succ o' \tag{1}$$

Dominance testing can be done easily.

THEOREM 3 *Let $\succ$ be a member of $\mathcal{F}_\mathcal{H}$ and o, o' be different members of $\mathcal{U}$. Let $B(o_1, o_2) = \{A \mid o_1.A \succ_A o_2.A\}$. Then*

$$o \succ o' \Leftrightarrow Ch_{\Gamma_\succ}(B(o, o')) \supseteq B(o', o)$$

In other words, o is preferred to o' iff every attribute in which o' is better than o is less important than at least one attribute in which o is better than o'.

Note that all less important attributes for an attribute are its descendants in the p-graph. Since p-graphs are transitive, all the descendants of an attribute are also its children.

Using Theorem 3, we can rewrite (1) as

$$\forall o \in \mathcal{O}, o' \in G \;.\; Ch_{\Gamma_\succ}(B(o,o')) \not\supseteq B(o',o) \qquad (2)$$

Note that no tuple can be preferred to itself by irreflexivity of $\succ$. Therefore, a p-skyline relation favoring G in $\mathcal{O}$ should satisfy $(|\mathcal{O}|-1)\cdot|G|$ *negative* constraints τ in the form:

$$\tau : Ch_{\Gamma_\succ}(\mathcal{L}_\tau) \not\supseteq \mathcal{R}_\tau$$

where $\mathcal{L}_\tau = B(o,o'), \mathcal{R}_\tau = B(o',o)$. We denote this set of constraints as $\mathcal{N}(G,\mathcal{O})$.

Example 7 *Take Example 1. Then any p-skyline relation* $\succ \in \mathcal{F}_\mathcal{H}$ *favoring* $G = \{t_3\}$ *in* $\mathcal{O}$ *has to satisfy each negative constraint below*

$t_1 \not\succ t_3$	$Ch_{\Gamma_\succ}(\{\texttt{make}\}) \not\supseteq \{\texttt{price}\}$
$t_2 \not\succ t_3$	$Ch_{\Gamma_\succ}(\{\texttt{make, year}\}) \not\supseteq \{\texttt{price}\}$
$t_4 \not\succ t_3$	$Ch_{\Gamma_\succ}(\{\texttt{make, year}\}) \not\supseteq \{\texttt{price}\}$
$t_5 \not\succ t_3$	$Ch_{\Gamma_\succ}(\{\texttt{make}\}) \not\supseteq \{\texttt{price,year}\}$

Now consider the notion of disfavoring W in $\mathcal{O}$. According to the definition, a p-skyline relation $\succ$ disfavors W in $\mathcal{O}$ iff the following holds

$$\forall o' \in W \;\exists o \in G \;.\; o \succ o' \qquad (3)$$

Following Theorem 3, it can be rewritten as a set of *positive constraints* $\mathcal{P}(W,G)$

$$\forall o' \in W \bigvee_{o_i \in G} Ch_{\Gamma_\succ}(B(o_i,o')) \supseteq B(o',o_i). \qquad (4)$$

Therefore, in order for $\succ$ to disfavor W in $\mathcal{O}$, its p-graph $\Gamma_\succ$ has to satisfy $|W|$ positive disjunctive constraints.

Example 8 *Take Example 1. Then any p-skyline relation* $\succ \in \mathcal{F}_\mathcal{H}$ *favoring* $G = \{t_1, t_3\}$ *and disfavoring* $W = \{t_4\}$ *in* $\mathcal{O}$ *has to satisfy the constraint*

$$t_1 \succ t_4 \vee t_3 \succ t_4$$

which is equivalent to the following positive constraint

$$Ch_{\Gamma_\succ}(\{\texttt{price}\}) \supseteq \{\texttt{year}\} \vee Ch_{\Gamma_\succ}(\{\texttt{price}\}) \supseteq \{\texttt{year,make}\}$$

Summarizing: in order to construct a p-skyline relation $\succ$ favoring G and disfavoring W in $\mathcal{O}$, we need to construct an attribute importance graph $\Gamma_\succ$ satisfying `SPO+Envelope`, $\mathcal{N}(G,\mathcal{O})$, and $\mathcal{P}(W,G)$. By Theorem 2, a p-graph of an optimal $\succ$ is *maximal* among all graphs satisfying `SPO+Envelope`, $\mathcal{N}(G,\mathcal{O})$, and $\mathcal{P}(W,G)$.

3.2 Using superior and inferior examples

In this section, we study the problem of computing a favoring/disfavoring p-skyline preference relation. We start with the problem of its existence.

THEOREM 4 *Problem 1 is* NP*-complete.*

We prove Theorem 4 by a reduction from `SAT`. To address Problem 2, we develop a method of computing *minimal extensions* of p-skyline relations discussed below.

Definition 6 *A relation* $\succ' \in \mathcal{F}_\mathcal{H}$ *is a* minimal extension *of* $\succ \in \mathcal{F}_\mathcal{H}$ *if* $\succ \subset \succ'$ *and there is no* $\succ'' \in \mathcal{F}_\mathcal{H}$ *such that* $\succ \subset \succ'' \subset \succ'$.

Similarly, we can define *minimal extensions* of $\Gamma_\succ$. We note that by Theorems 1 and 2, a p-graph $\Gamma_{\succ'}$ of a minimal extension $\succ'$ of $\succ \in \mathcal{F}_\mathcal{H}$ is also a *minimal extension* satisfying `SPO+Envelope` of the p-graph $\Gamma_\succ$.

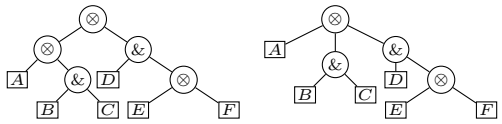
3.2.1 Computing minimal extensions of a p-skyline relation

Here we develop a set of rewriting rules which allow to construct *all* minimal extensions of a given p-skyline relation. The rules are applied to a *syntax tree* of a p-skyline relation. Given a p-skyline relation $\succ$, its syntax tree $T_\succ$ corresponds to its definition using accumulation operators. Every leaf node of such a syntax tree is an attribute. Each non-leaf node is labeled with an accumulation operator and corresponds to the result of applying this operator. Due to associativity of $\otimes$ and & and symmetry of $\otimes$, the same p-skyline relation may have more than one syntax tree. Syntax trees representing the same p-skyline relation are called *equivalent*. A p-skyline syntax tree is called *normalized* if every non-leaf node is labeled differently from its parent. Clearly, for every p-skyline relation, there is a normalized p-skyline syntax tree. However, a normalized syntax tree is not unique for a p-skyline relation due to symmetry of $\otimes$.

Example 9 *Let a p-skyline relation be*

$$\succ_4 = \succ_A \;\otimes\; (\succ_B \;\&\; \succ_C) \;\otimes\; (\succ_D \;\&\; (\succ_E \;\otimes\; \succ_F))$$

Instances of its unnormalized and normalized syntax trees are shown in Figure 3.



(a) Unnormalized (b) Normalized

Figure 3: p-skyline syntax tree of $\succ_4$

Every node of a syntax tree is itself a root of another syntax tree. Let us associate with every node C of a syntax tree the set $Var(C)$ of attributes which are the leaf descendants of C in the parse tree. Essentially, $Var(C)$ corresponds to $Var(\succ_C)$ introduced in Definition 4.

Note that we use two graph notations for p-skyline relations: p-graphs and p-skyline syntax trees. Although they represent different concepts, there is a relationship between them, shown informally in Figure 4. For every pair of paths in a syntax tree going from a $\otimes$ -node to leaf nodes, no edges exist between the nodes in the different paths. For every pair of paths in a syntax tree going from a & -node to leaf nodes, the p-graph has edges going from all the leaf nodes on the left path to all the leaf nodes on the right path. These relationships follow from Definition 5.

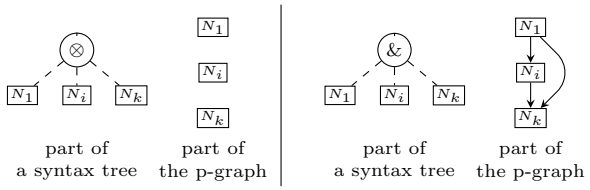
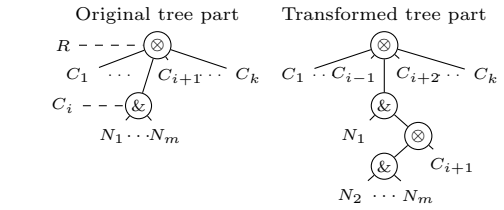
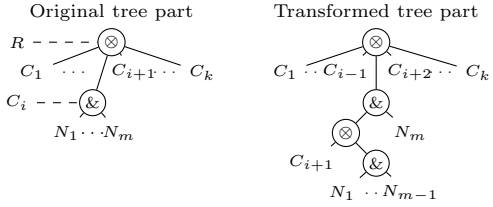


Figure 4: Syntax tree and p-graph correspondence

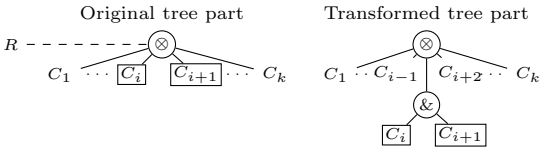
The method of computing all minimal extensions we propose here operates directly on p-skyline relations represented as p-skyline syntax trees. In particular, we show a set of rewriting rules of p-skyline syntax trees such that every unique application of a rule from this set results in a *unique minimal extension* of the given p-skyline relation. If *all* minimal extensions of a p-skyline relation are needed, then one needs to apply to the syntax tree *every* rule in *every* possible way.



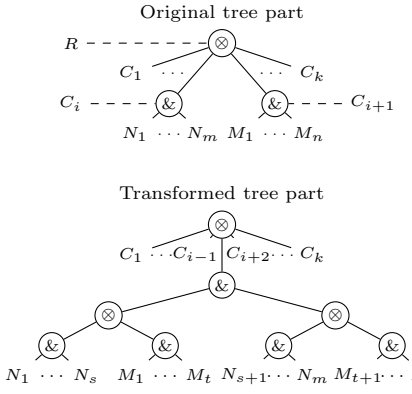
(a) $Rule_1(T_\succ, C_i, C_{i+1})$



(b) $Rule_2(T_\succ, C_i, C_{i+1})$



(c) $Rule_3(T_\succ, C_i, C_{i+1})$



(d) $Rule_4(T_\succ, C_i, C_{i+1}, s, t)$

$\boxed{C_i}$ - leaf node

C_i - leaf or non-leaf node

Figure 5: Syntax tree rewriting rules

The rewriting rules are shown in Figure 5. On the left hand side, we show a part of a normalized syntax tree of the original p-skyline relation. On the right hand side, we show how the corresponding part is modified in the syntax tree of the resulting relation. We assume that the remaining part of the syntax tree is left unchanged. All the rewriting rules perform actions with two children C_i and C_{i+1} of a $\otimes$-node R of a syntax tree. For the sake of simplicity, C_i and C_{i+1} are shown in Figure 5 as consecutive children. However, in the rules we assume that they may be any pair of child nodes of the same $\otimes$-node.

It can be easily checked (Figure 4) that all the rules only *add* edges to the p-graph of the original preference relation. That means (Theorem 2) that they *extend* the original p-skyline relation.

Observation 1 *Given a p-skyline relation $\succ \in \mathcal{F}_\mathcal{H}$, a single application of a rule from $\{Rule_1, \ldots, Rule_4\}$ to a normalized syntax tree $T_\succ$ adds the following edges to $\Gamma_\succ$:*

- $Rule_1(T_\succ, C_i, C_{i+1})$: $\{XY \mid X \in Var(N_1), Y \in Var(C_{i+1})\}$
- $Rule_2(T_\succ, C_i, C_{i+1})$: $\{XY \mid X \in Var(C_{i+1}), Y \in Var(N_m)\}$
- $Rule_3(T_\succ, C_i, C_{i+1})$: $\{(C_i, C_{i+1})\}$
- $Rule_4(T_\succ, C_i, C_{i+1}, s, t)$: $\{XY | X \in \cup_{p\in 1\ldots s} Var(N_p), Y \in \cup_{q\in t+1\ldots n} Var(M_q)\} \cup \{XY | X \in \cup_{p\in 1\ldots t} Var(M_p), Y \in \cup_{q\in s+1\ldots m} Var(N_q)\}$.

We note that every & - and $\otimes$ -node in a p-skyline syntax tree has to have at least two child nodes. This is because the operators & and $\otimes$ must have at least two arguments. However, as a result of a rewriting rule application, some & - and $\otimes$ -nodes may have only one child node, i.e., the syntax tree becomes invalid. To make it valid, we remove all nodes C with a single child and connect the child directly to the parent of C, as shown in Figure 6.

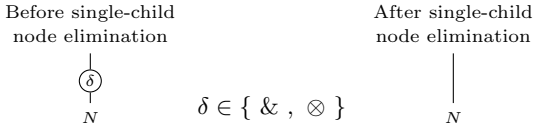


Figure 6: Single-child node elimination

An important property of the set of rules in Figure 5 is shown in the next proposition. It says that an application of any rule in Figure 5 results in a minimal extension of the p-skyline relation, and every minimal extension may be obtained using a *single* application of one of these rules.

Proposition 2 *Let $\succ \in \mathcal{F}_\mathcal{H}$, and $T_\succ$ be a normalized syntax tree of $\succ$. Then $\succ'$ is a* minimal extension *of $\succ$ iff a syntax tree $T_{\succ'}$ of $\succ'$ is obtained from $T_\succ$ by a single application of a rule $Rule_1, \ldots, Rule_4$.*

Proposition 2 has several corollaries which we use further.

Corollary 1 *A syntax tree of every minimal extension of $\succ \in \mathcal{F}_\mathcal{H}$ may be computed in time $\mathcal{O}(|\mathcal{A}|)$.*

In Corollary 1, we assume the adjacency-list representation of syntax trees. The total number of nodes in a tree is linear in the number of its leaf nodes $|\mathcal{A}|$. Thus the number of edges in $T_\succ$ is $\mathcal{O}(|\mathcal{A}|)$. The rewriting of $T_\succ$ using any rule requires removing and adding $\mathcal{O}(|\mathcal{A}|)$ edges of $T_\succ$.

Corollary 2 *Every $\succ \in \mathcal{F}_\mathcal{H}$ has at most $\mathcal{O}(|\mathcal{A}|^4)$ different minimal extensions.*

The justification for Corollary 2 is as follows. The set of minimal-extension rules is complete due to Proposition 2. Every rule operates with two nodes – C_i and C_{i+1} – of a syntax tree. Hence, the number of such node pairs is $\mathcal{O}(|\mathcal{A}|^2)$. $Rule_4$ also relies on some partitioning of the sequence of child nodes of C_i and C_{i+1}. The total number of such partitionings is $\mathcal{O}(|\mathcal{A}|^2)$. As a result, the total number of different rule applications is $\mathcal{O}(|\mathcal{A}|^4)$.

THEOREM 5 *Problem 2 is* FNP*-complete*

We note that Problem 2 is a functional problem. The complexity class FNP for functional problems is a counterpart of the class NP for decision problems. The membership of Problem 2 in FNP follows from Corollaries 1 and 2. To prove its hardness, we use a reduction from FSAT.

In view of Theorems 4 and 5, we consider next more restricted versions of Problems 1 and 2. We will assume that there are no inferior examples ($W = \emptyset$).

3.3 Using superior examples only

By the definition of the problem of discovering favoring/disfavoring p-skyline relations, a user is required to identify both superior and inferior examples in a set of tuples. In many scenarios, users are only indirectly involved in the process of identifying such classes. For instance, the click-through rate may be used to measure the popularity of products. Using this metric, it is easy to find the superior examples – the tuples with the highest click-through rate. However, the problem of identifying inferior examples – those which the user confidently dislikes – is harder. Namely, low click-through rate may mean that a tuple is inferior, as well as that the user does not know about it, or it does not satisfy the search criteria. Thus, there is a need for discovering p-skyline relations based on superior examples only.

In this section, we consider restrictions of Problems 1 and 2 assuming that $W = \emptyset$.

Problem 1′. *Given a set of atomic preference relations $\mathcal{H}$ and a set of superior examples G in a set $\mathcal{O}$, determine if there exists $\succ \in \mathcal{F}_{\mathcal{H}}$ favoring G in $\mathcal{O}$.*

Problem 2′. *Given a set of atomic preference relations $\mathcal{H}$ and a set of superior examples G in a set $\mathcal{O}$, discover an optimal p-skyline relation $\succ \in \mathcal{F}_{\mathcal{H}}$ favoring G in $\mathcal{O}$.*

As we show further, these problems are computationally simpler than the corresponding problems for favoring/disfavoring p-skyline relations (if $P \neq NP$). We show polynomial time algorithms for both of them.

Consider Problem 1′ first. Theorem 2 implies that the *skyline* preference relation is the *least* in the class $\mathcal{F}_{\mathcal{H}}$. It is also well known that smaller preference relations result in larger sets of the most preferred tuples. Therefore, the relation $\succ$ in $\mathcal{F}_{\mathcal{H}}$ which results in the largest set $w_{\succ}(\mathcal{O})$ is the skyline preference relation $sky_{\mathcal{H}}$.

Proposition 3 *There exists a p-skyline relation $\succ \in \mathcal{F}_{\mathcal{H}}$ favoring G in $\mathcal{O}$ iff $G \subseteq w_{\mathrm{sky}_{\mathcal{H}}}(\mathcal{O})$.*

Proposition 3 implies that to solve Problem 1′, one needs to evaluate a skyline algorithm over $\mathcal{O}$ and check if the result contains G. This clearly can be done in *polynomial time*.

In order to efficiently solve Problem 2′, we present a polynomial time algorithm below. Before starting to describe the algorithm, let us consider the constraints which should be satisfied by the p-graph $\Gamma_{\succ}$ of such a discovered relation $\succ$. Since we do not deal with inferior examples here, the set of constraints sufficient to guarantee favoring G in $\mathcal{O}$ is `SPO+Envelope` and $\mathcal{N}(G, \mathcal{O})$. Furthermore, to make the relation $\succ$ optimal favoring G in $\mathcal{O}$, $\Gamma_{\succ}$ has to be a *maximal* graph satisfying these constraints.

The approach of constructing optimal favoring p-skyline relations we propose here is based on iterative rewriting of normalized syntax trees. We use a subset of the rewriting rules shown in Figure 5. We assume that the provided set of superior examples G satisfies Proposition 3, i.e., $G \subseteq w_{sky_{\mathcal{H}}}(\mathcal{O})$. First, we generate the set of negative constraints $\mathcal{N}(G, \mathcal{O})$. The p-skyline relation we start with is $sky_{\mathcal{H}}$ since it is the least relation in $\mathcal{F}_{\mathcal{H}}$ favoring G in $\mathcal{O}$. In every iteration of the algorithm, we pick an atomic preference in $\mathcal{H}$ and apply to the current p-skyline relation's syntax tree a fixed set of rewriting rules. As a result, we obtain a "locally maximal" p-skyline relation satisfying *the given set* $\mathcal{N}(G, \mathcal{O})$ *of negative constraints*. Eventually, this technique produces a maximal p-skyline relation satisfying $\mathcal{N}(G, \mathcal{O})$.

Let us describe what we mean by "locally maximal".

Definition 7 *Let M be a nonempty subset of $\mathcal{A}$. A p-skyline relation $\succ \in \mathcal{F}_{\mathcal{H}}$ which favors G in $\mathcal{O}$ such that $E(\Gamma_{\succ}) \subset M \times M$ is called* M-favoring G in $\mathcal{O}$. *A maximal relation among all of such relations is called* optimal.

By Definition 7, the edge set of a p-graph of every optimal M-favoring relation is maximal among all the p-graphs of M-favoring relations. Note that if M is a singleton, the edge set of a p-graph $\Gamma_{\succ}$ of an optimal M-favoring relation $\succ$ is empty, i.e., $\succ = sky_{\mathcal{H}}$. If $M = \mathcal{A}$, then an optimal p-skyline relation M-favoring G in $\mathcal{O}$ is also optimal favoring G in $\mathcal{O}$. Thus, if we had a method of transforming an optimal M-favoring to an optimal $(M \cup \{A\})$-favoring p-skyline relation for any attribute A, we could construct an optimal p-skyline relation favoring G in $\mathcal{O}$ by induction.

To do this transformation, we use the following subset of the rules shown in Figure 5: 1) $Rule_1$ if $C_{i+1} = A$ or $N_1 = A$; 2) $Rule_2$ if $C_{i+1} = A$ or $N_m = A$; and 3) $Rule_3$ if $C_i = A$ or $C_{i+1} = A$. These rules have the following properties. First, given a p-skyline relation $\succ$, they add only the edges to $\Gamma_{\succ}$ which go to or from the node A. Second, any optimal $(M \cup \{A\})$-favoring p-skyline relation $\succ'$ can be constructed from any optimal M-favoring p-skyline relation $\succ$ (s.t. $\succ \subset \succ'$) by some consecutive application of these rules. Notice that $Rule_4$ is not used here because any its application adds to a p-graph edges going from *at least* two and to *at least* two attributes. At the same time, not every consecutive application of the rules above results in an optimal $(M \cup \{A\})$-favoring relation. Further in Algorithm 4 we show a rule application strategy which allows to construct optimal $(M \cup \{A\})$-favoring p-skyline relations. That strategy is exhaustive, i.e., we apply the rules while the resulting relation satisfies $\mathcal{N}(G, \mathcal{O})$.

3.3.1 Efficient constraint checking

We have shown in Observation 1 that the edge set of the p-graph of a transformed p-skyline relation monotonically increases. Therefore, while we transform a p-skyline relation $\succ$, for every negative constraint τ, we can just drop the elements of $\mathcal{R}_\tau$ which already have incoming edges from $\mathcal{L}_\tau$.

Proposition 4 *Let $\succ \in \mathcal{F}_{\mathcal{H}}$ satisfy a system of negative constraints $\mathcal{N}$. Construct the system of negative constraints $\mathcal{N}'$ from $\mathcal{N}$ in which every $\tau' \in \mathcal{N}'$ is created from a member τ of $\mathcal{N}$ in the following way*

- $\mathcal{L}_{\tau'} = \mathcal{L}_\tau$
- $\mathcal{R}_{\tau'} = \mathcal{R}_\tau - \{Y \in \mathcal{R}_\tau \mid \exists X \in \mathcal{L}_\tau \,.\, \Gamma_{\succ} \models X \to Y\}$

Then any $\succ' \in \mathcal{F}_{\mathcal{H}}$, which is a superset of $\succ$, satisfies $\mathcal{N}$ iff $\succ'$ satisfies $\mathcal{N}'$.

A system of negative constraints $\mathcal{N}'$ obtained as a result of the transformation shown in Proposition 4 is called *minimal w.r.t.* $\succ$. The next proposition summarizes the constraint checking rules over a minimal system of negative constraints.

Proposition 5 *Let $\succ \in \mathcal{F}_{\mathcal{H}}$ satisfy a system of negative constraints $\mathcal{N}$, and $\mathcal{N}$ be minimal w.r.t. $\succ$. Let $\succ' \in \mathcal{F}_{\mathcal{H}}$ be obtained from $\succ$ using one of the rewriting rules. Denote the* added *parents and children of A in $\Gamma_{\succ'}$ as P_A and C_A correspondingly. Then $\succ'$ violates $\mathcal{N}$ iff there is a constraint $\tau \in \mathcal{N}$ s.t.*

- $\mathcal{R}_\tau = \{A\} \wedge P_A \cap \mathcal{L}_\tau \neq \emptyset$, *or*
- $A \in \mathcal{L}_\tau \wedge \mathcal{R}_\tau \subseteq C_A$

3.3.2 Algorithm for p-skyline relation discovery

Here we present an algorithm for constructing an optimal p-skyline relation favoring G in $\mathcal{O}$. This algorithm can only be used if $G \subseteq w_{sky_{\mathcal{H}}}(\mathcal{O})$, otherwise, by Proposition 3, no such p-skyline relation exists.

The algorithm provides a strategy for applying the p-skyline syntax tree rewriting rules discussed above. The function `push` (Algorithm 1) takes four arguments: a set M of attributes, a normalized syntax tree T of an M-favoring p-skyline relation, the current attribute A, and a system of negative constraints $\mathcal{N}$ minimal w.r.t. $\succ$ ($\succ$ here is a p-skyline relation represented by the syntax tree T). It returns true if a rewriting rule has been applied to T without violating $\mathcal{N}$, and false if no rewriting rule can be applied to T without violating $\mathcal{N}$. When the function returns, $\mathcal{N}$ is minimal w.r.t. the p-skyline relation represented by the modified syntax tree, and T is normalized.

The goal of `push` is to find an appropriate rewriting rule which adds to the current p-graph edges going from M to A or vice versa. The function has two branches: the parent of the node A in the syntax tree T is a &-node (i.e., we may apply $Rule_1$ where N_1 is A or $Rule_2$ where N_m is A), or a $\otimes$-node (i.e., we may apply $Rule_1$ or $Rule_2$ where C_{i+1} is A, or $Rule_3$ where C_i or C_{i+1} is A). In the first branch (line 2-14), we distinguish between applying $Rule_1$ (line 3-8) and $Rule_2$ (line 9-14). It is easy to notice that with the parameters specified above, the rules are exclusive. However, the application patterns are similar. First, we find an appropriate child C_{i+1} of R (lines 4 and 10). It is important for $Var(C_{i+1})$ to be a subset of M because we want to add edges going from M to A or from A to M. Then we check if the corresponding rule application will not violate $\mathcal{N}$. To do that, we use the function `checkConstr` (lines 5 and 11) as per Proposition 5. If a rule application does not violate $\mathcal{N}$, we apply the corresponding rule (lines 6 and 12) and minimize $\mathcal{N}$ w.r.t. the p-skyline which is the result of the rewriting (Proposition 4). To do that, we use the function `minimize`.

The second branch of `push` is similar to the first one and different in only the rewriting rules applied. As a result, it is easy to notice that `push` checks every possible rule application not violating $\mathcal{N}$ and adds to the p-graph only the edges going from A to the elements of M or vice versa.

The function `discover` (Algorithm 4) is the entry point of the entire algorithm. It takes four arguments: a set of superior examples G, the entire set of tuples $\mathcal{O}$, the set $\mathcal{H}$ of atomic preference relations, and the set of relevant attributes $\mathcal{A}$. It iteratively constructs an optimal p-skyline relation favoring G in $\mathcal{O}$. In each iteration, it calls `push` repeatedly until it returns false, i.e., until every possible application of rewriting rules violates $\mathcal{N}$. After every rule application, it performs normalization of the modified syntax tree, which is required by Proposition 2. Thus, the syntax tree T obtained in the `repeat` loop represents an optimal $(M \cup \{A\})$-favoring p-skyline relation. Note that the `repeat` loop eventually terminates since every rewriting rule pushes A down the syntax tree, and the number of non-leaf nodes in a p-skyline syntax tree cannot be more than $|\mathcal{A}| - 1$.

THEOREM 6 *The function* `discover` *returns a syntax-tree of a maximal p-skyline relation satisfying the system of negative constraints $\mathcal{N}$. Its running time is $O(|\mathcal{N}| \cdot |\mathcal{A}|^3)$.*

Theorem 6 implies that if $\mathcal{N}$ is $\mathcal{N}(G, \mathcal{O})$ then `discover` returns an optimal p-skyline relation favoring G in $\mathcal{O}$.

Algorithm 1 `push`(T, M, A, $\mathcal{N}$)

Require: T is normalized
1: **if** the parent of A in T is of type &
2: $C_i :=$ parent of A in T; $R :=$ parent of C_i in T;
3: **if** A is the first child of C_i
4: **for** each child C_{i+1} of R s.t. $Var(C_{i+1}) \subseteq M$
5: **if** checkConstr($\mathcal{N}$, A, $\emptyset$, $Var(C_{i+1})$)
6: apply $Rule_1(T, C_i, C_{i+1})$
7: $\mathcal{N} := minimize(\mathcal{N}, Var(A), Var(C_{i+1}))$
8: **return** *true*
9: **else if** A is the last child of C_i
10: **for** each child C_{i+1} of R s.t. $Var(C_{i+1}) \subseteq M$
11: **if** checkConstr($\mathcal{N}$, A, $Var(C_{i+1})$, $\emptyset$)
12: apply $Rule_2(T, C_i, C_{i+1})$
13: $\mathcal{N} := minimize(\mathcal{N}, Var(C_{i+1}), Var(A))$
14: **return** *true*
15: **else** // the parent of A in T is of type $\otimes$
16: $R :=$ parent of A in T;
17: **for** each child C_i of R s.t. $Var(C_i) \subseteq M$
18: **if** C_i is of type &
19: $N_1 :=$ first child of C_i, $N_m :=$ last child of C_i
20: **if** checkConstr($\mathcal{N}$, A, $Var(N_1)$, $\emptyset$)
21: apply $Rule_1(T, C_i, A)$
22: $\mathcal{N} :=$minimize($\mathcal{N}$, $Var(N_1)$, $Var(A)$)
23: **return** *true*
24: **else if** checkConstr($\mathcal{N}$, A, $\emptyset$, $Var(N_m)$)
25: apply $Rule_2(T, C_i, A)$
26: $\mathcal{N} :=$ minimize($\mathcal{N}$, $Var(A)$, $Var(N_m)$)
27: **return** *true*
28: **else** // C_i is a leaf node, since T is normalized
29: **if** checkConstr($\mathcal{N}$, A, $Var(C_i)$, $\emptyset$)
30: apply $Rule_3(T, C_i, A)$
31: $\mathcal{N} :=$minimize($\mathcal{N}$, $Var(C_i)$, $Var(A)$)
32: **return** *true*
33: **else if** checkConstr($\mathcal{N}$, A, $\emptyset$, $Var(C_i)$
34: apply $Rule_3(T, A, C_i)$
35: $\mathcal{N} :=$minimize($\mathcal{N}$, $Var(A)$, $Var(C_i)$)
36: **return** *true*
37: **return** *false*

Algorithm 2 `checkConstr`($\mathcal{N}$, A, P_A, C_A)

1: **for** each $\tau \in \mathcal{N}$
2: **if** $\mathcal{R}_\tau = \{A\} \wedge P_A \cap \mathcal{L}_\tau \neq \emptyset \vee A \in \mathcal{L}_\tau \wedge \mathcal{R}_\tau \subseteq C_A$
3: **return** *false*
4: **return** *true*

In our implementation of the algorithm, all sets of attributes are represented as bitmaps of fixed size $|\mathcal{A}|$. Similarly, every negative constraint τ is represented as a pair of bitmaps corresponding to $\mathcal{L}_\tau$ and $\mathcal{R}_\tau$.

3.3.3 Reducing the number of negative constraints

As we showed in Theorem 6, the running time of the function `discover` linearly depends on the size of the system of negative constraints $\mathcal{N}$. If we construct it as shown in Section 3.1, it will contain $(|\mathcal{O}| - 1) \cdot |G|$ constraints. In this section, we propose two methods of reducing the size of $\mathcal{N}$. Both methods are based on applying the skyline operator.

Algorithm 3 `minimize`($\mathcal{N}$, U, D)

1: **for** each constraint τ in $\mathcal{N}$
2: **if** $U \cap \mathcal{L}_\tau \neq \emptyset$
3: $\mathcal{R}_\tau \leftarrow \mathcal{R}_\tau - D$
4: **return** $\mathcal{N}$

Algorithm 4 `discover`$(G, \mathcal{O}, \mathcal{H}, \mathcal{A})$

Require: $G \subseteq w_{sky_{\mathcal{H}}}(\mathcal{O})$
1: $\mathcal{N} = \mathcal{N}(G, \mathcal{O})$
2: T = a normalized syntax tree of $sky_{\mathcal{H}}$
3: M = any attribute from $\mathcal{A}$
4: **for** each attribute A in $\mathcal{A} - M$
5: **repeat**
6: r = push(T, M, A, $\mathcal{N}$);
7: Normalize the syntax tree T
8: **until** r is false
9: $M = M \cup \{A\}$
10: **return** T

However, the first method applies it to the tuple set $\mathcal{O}$, and the second to the set of constraints itself.

Note that each negative constraint is used to show that a tuple should not be preferred to a superior example. We also know that the relation $sky_{\mathcal{H}}$ is the least in $\mathcal{F}_{\mathcal{H}}$. By definition of the winnow operator, for every $o' \in \mathcal{O} - w_{sky_{\mathcal{H}}}(\mathcal{O})$ there is a tuple $o \in w_{sky_{\mathcal{H}}}(\mathcal{O})$ s.t. o is preferred to o' according to $sky_{\mathcal{H}}$. Since $sky_{\mathcal{H}}$ is the least relation in $\mathcal{F}_{\mathcal{H}}$, the same o is preferred to o' according to every member of $\mathcal{F}_{\mathcal{H}}$. Thus, in order to guarantee favoring G in $\mathcal{O}$, the system of negative constraints needs to contain only the constraints showing that the tuples in $w_{sky_{\mathcal{H}}}(\mathcal{O})$ are not preferred to the superior examples. The size of such a system of negative constraints is $(|w_{sky_{\mathcal{H}}}(\mathcal{O})| - 1) \cdot |G|$.

Another way to reduce the size of a system of negative constraints is based on the following fact. Let us take two negative constraints $\tau, \tau' \in \mathcal{N}$ such that $\mathcal{L}_{\tau'} \subseteq \mathcal{L}_{\tau}$, $\mathcal{R}_{\tau} \subseteq \mathcal{R}_{\tau'}$, and at least one of these relationships is strict. It is easy to check that τ strictly implies τ'. Thus, the constraint τ' is *redundant* and may be deleted from $\mathcal{N}$. This idea can also be expressed as follows:

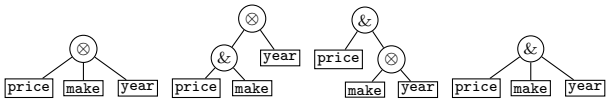
τ strictly implies τ' iff $\mathcal{L}_{\tau'} \subseteq \mathcal{L}_{\tau} \wedge (\mathcal{A} - \mathcal{R}_{\tau'}) \subseteq (\mathcal{A} - \mathcal{R}_{\tau})$ and at least one containment is strict.

Represent τ as the bitmap representation of $(\mathcal{A} - \mathcal{R}_{\tau})$ appended to the bitmap representation of $\mathcal{L}_{\tau}$. We assume that a bit is set to 1 iff the corresponding attribute is in the corresponding set. Denote such a representation as $bitmap(\tau)$.

Example 10 *Let* $\mathcal{L}_{\tau} = \{A_1, A_3, A_5\}$, $\mathcal{R}_{\tau} = \{A_2\}$, $\mathcal{L}_{\tau'} = \{A_1, A_5\}$, $\mathcal{R}_{\tau'} = \{A_2, A_4\}$. *Let* $\mathcal{A} = \{A_1, \ldots, A_5\}$. *Then* $bitmap(\tau) = 10101\ 10111$ *and* $bitmap(\tau') = 10001\ 10101$.

Consider $bitmap(\tau)$ as a vector with $2 \cdot |\mathcal{A}|$ dimensions. From the negative constraint implication rule, it follows that τ strictly implies τ' iff $bitmap(\tau)$ and $bitmap(\tau')$ satisfy the *Pareto improvement principle*, i.e., *the value of every dimension of* $bitmap(\tau)$ *is greater or equal to the corresponding value in* $bitmap(\tau')$, *and there is at least one dimension whose value in* $bitmap(\tau)$ *is greater than in* $bitmap(\tau')$. Therefore, the set of all non-redundant constraints in $\mathcal{N}$ corresponds to the *skyline* of the set of bitmap representations of all constraints in $\mathcal{N}$. Moveover, $bitmap(\tau)$ can have only two values in every dimension. Thus, an algorithm for computing skylines over low cardinality domains (e.g. [21]) can be used to compute the set of non-redundant constraints.

Example 11 *Take* $\mathcal{O}$ *and* $\mathcal{H}$ *from Example 1, and* G *from Example 7. Let us find an optimal* $\succ \in \mathcal{F}_{\mathcal{H}}$ *favoring* G *in* $\mathcal{O}$ *by running* `discover`. *Among the constraints in Example 7, only* $\mathcal{N} = \{\tau : Ch_{\Gamma_{\succ}}(\{\texttt{make}, \texttt{year}\}) \not\supseteq \{\texttt{price}\}\}$ *is nonredundant. Moreover, it cannot be further minimized because* $\mathcal{R}_{\tau}$ *is a singleton. Consider the attributes in the order:* `make`, `price`, *and* `year`.



(a) (b) (c) (d)

Figure 7: Example 11

We run `discover`. *The tree* T *(line 2) is shown in Figure 7(a). The initial value of* M *is* {`make`}. *First, we call* `push`$(T, M, \texttt{price}, \mathcal{N})$. *The parent of* `price` *is the* $\otimes$*-node (Figure 7(a)), so we go to line 16 of* `push`, *where* R *is set to the* $\otimes$*-node (Figure 7(a)). After* C_i *is set to the node* `make` *in line 17, we go to line 29 because it is a leaf node. The* `checkConstr` *test in line 29 fails because* $\mathcal{N}$ *prohibits the edge* `make` $\rightarrow$ `price`. *Hence, we go to line 33 where the* `checkConstr` *test succeeds. We apply* $Rule_3(T, \texttt{price}, C_i)$, `push` *returns true, and the resulting syntax tree* T *is shown in Figure 7(b). Next time we call* `push`$(T, M, \texttt{price}, \mathcal{N})$ *(line 6 of* `discover`*), we get to line 4 of* `push`. *Since* `year` $\notin M$, *we immediately go to line 37 and return* $false$. *In* `discover` *we set* M *to* {`make`, `price`} *and call* `push`$(T, M, \texttt{year}, \mathcal{N})$. *There we go to line 16 (*R *is set to the* $\otimes$*-node in Figure 7(b)),* C_i *is set to the &-node (Figure 7(b)), we apply* $Rule_1(T, C_i, \texttt{year})$ *(the resulting tree* T *is show in Figure 7(c)), and true is returned. When* $push(T, M, \texttt{year}, \mathcal{N})$ *is called next time, we first go to line 16,* R *is set to the* $\otimes$*-node (Figure 7(c)), and* C_i *to the node* `make`. *Then* $Rule_3(T, C_i, \texttt{year})$ *is applied (line 30) resulting in the tree* T *shown in Figure 7(d), and true is returned.* `push`$(T, M, year, \mathcal{N})$ *is called once again from* `discover`, *but it returns* $false$, *and thus the tree in Figure 7(d) is the final one. According to the corresponding p-skyline relation,* t_3 *dominates all other tuples in* $\mathcal{O}$.

The final p-skyline relation constructed in Example 11 is a prioritized accumulation of all the atomic preference relations. This is due to the fact that $\mathcal{N}$ contained only one constraint. When more constraints are involved, a discovered p-skyline relation generally also has occurrences of Pareto accumulation.

4. EXPERIMENTS

We have performed extensive experimental study of the proposed framework. The algorithms were implemented in Java. The experiments were ran on Intel Core 2 Duo CPU 2.1 GHz with 2.0GB RAM. The experiments were run on four data sets: one real-life and three synthetic.

4.1 Experiments with real-life data

In this section, we focus on experimenting with the accuracy of the p-skyline relation discovery algorithm and the reduction of skyline sizes achieved by modeling user preferences using p-skyline relations. We used a data set $\mathcal{O}$ which stores statistics of NHL players [22] and contains 9395 tuples. We used three sets of relevant attributes $\mathcal{A}$ of 12, 9, and 6 attributes, respectively. The sizes of the corresponding skylines were 568, 114, and 33. In this experiment, we randomly generated 100 p-skyline relations $\succ_{fav}$ and computed the sets $w_{\succ_{fav}}(\mathcal{O})$. For each $w_{\succ_{fav}}(\mathcal{O})$, we randomly picked 5 tuples from it, and used them as superior examples G_{fav} to discover three different optimal p-skyline relations $\succ$ favoring G_{fav} in $\mathcal{O}$ using the proposed algorithm. Out of those three relations, we picked the one resulting in $w_{\succ}(\mathcal{O})$ of the smallest size. Then we added 5 more tuples

from $w_{\succ_{fav}}(\mathcal{O})$ to G_{fav} and repeated the same procedure. We kept adding tuples to G_{fav} from $w_{\succ_{fav}}(\mathcal{O})$ until G_{fav} reached $w_{\succ_{fav}}(\mathcal{O})$.

To measure the accuracy of the p-skyline discovery algorithm, we computed the following three values: (i) the measure of similarity of $w_{\succ}(\mathcal{O})$ and $w_{\succ_{fav}}(\mathcal{O})$

$$acc\text{-}ratio = \frac{|w_{\succ}(\mathcal{O}) \cap w_{\succ_{fav}}(\mathcal{O})|}{|w_{\succ_{fav}}(\mathcal{O})|},$$

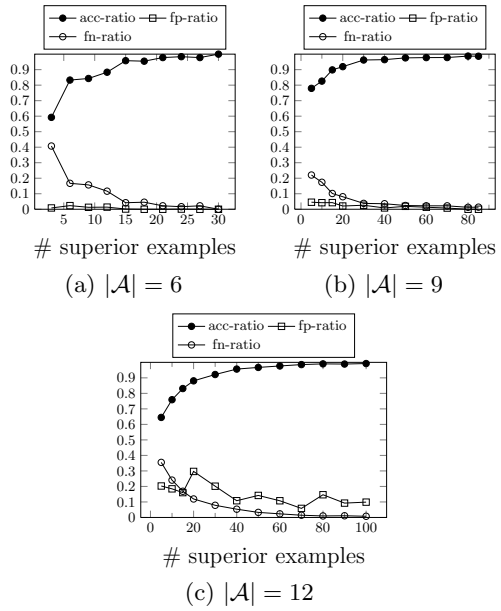
(ii) the ratio of tuples which should *have not* been in $w_{\succ}(\mathcal{O})$

$$fp\text{-}ratio = \frac{|w_{\succ}(\mathcal{O}) - w_{\succ_{fav}}(\mathcal{O})|}{|w_{\succ_{fav}}(\mathcal{O})|},$$

and (iii) the ratio of tuples which should *have* been in $w_{\succ}(\mathcal{O})$

$$fn\text{-}ratio = \frac{|w_{\succ_{fav}}(\mathcal{O}) - w_{\succ}(\mathcal{O})|}{|w_{\succ_{fav}}(\mathcal{O})|}.$$

We plotted the average values of these measures in Figure 8. As we can observe, when the sets of superior examples G_{fav} used to discover $\succ$ were of size 15 or more, the average values of *acc-ratio* were greater than 0.83 for every set $\mathcal{A}$ we used. We note that *acc-ratio* converges to 1 faster for smaller $\mathcal{A}$ due to the smaller size of the corresponding skyline. At the same time, even for $|\mathcal{A}| = 12$, *acc-ratio* becomes close to 1 (> 0.95) when the number of superior examples were 40 or more. The fact that *fn-ratio* starts from comparatively large values can be justified by the fact that the discovery algorithm computes an *optimal* relation favoring G_{fav} in $\mathcal{O}$. Thus, when G_{fav} is small, it is not sufficient to exactly capture the corresponding preference relation $\succ_{fav}$. However, when we increase the number of superior examples, the value of *fn-ratio* consistently goes down.



(a) $|\mathcal{A}| = 6$ (b) $|\mathcal{A}| = 9$

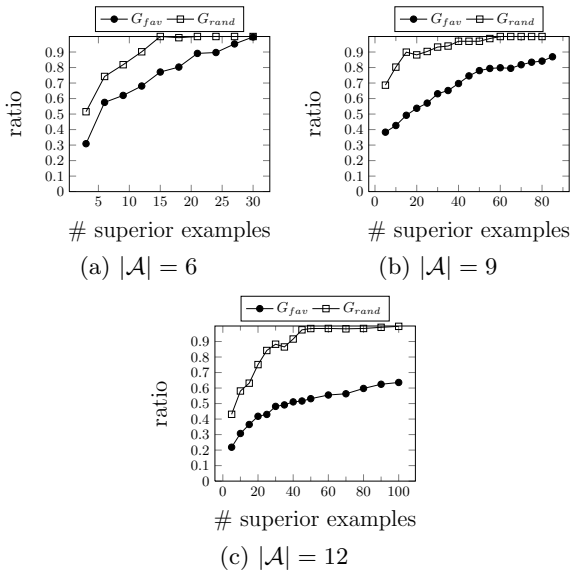
(c) $|\mathcal{A}| = 12$

Figure 8: Accuracy of p-skyline discovery

In Figure 9, we use a similar setup to show how the size of a p-skyline query result varies as a function of the number of superior examples. Here we used two kinds of superior example sets: sets G_{fav} which were drawn from the sets of the best tuples according to real p-skyline relations (as discussed above), and sets G_{rand} which were drawn *randomly* from the skyline. Hence, a set G_{rand} may not be favored by any p-skyline relation (besides $sky_{\mathcal{H}}$, of course). We use these sets to discover p-skyline relations $\succ$ favoring them. In Figure 9, we plot

$$ratio = \frac{|w_{\succ}(\mathcal{O})|}{|w_{sky_{\mathcal{H}}}(\mathcal{O})|},$$

which shows the difference in the sizes of results of p-skyline and skyline queries. As this figure suggests, using larger sets of relevant attributes generally results in smaller *ratio* values. Moreover, p-skyline query results are smaller when superior examples are similar (i.e., G_{fav}) rather then arbitrary (i.e., G_{rand}).



(a) $|\mathcal{A}| = 6$ (b) $|\mathcal{A}| = 9$

(c) $|\mathcal{A}| = 12$

Figure 9: p-skyline size reduction

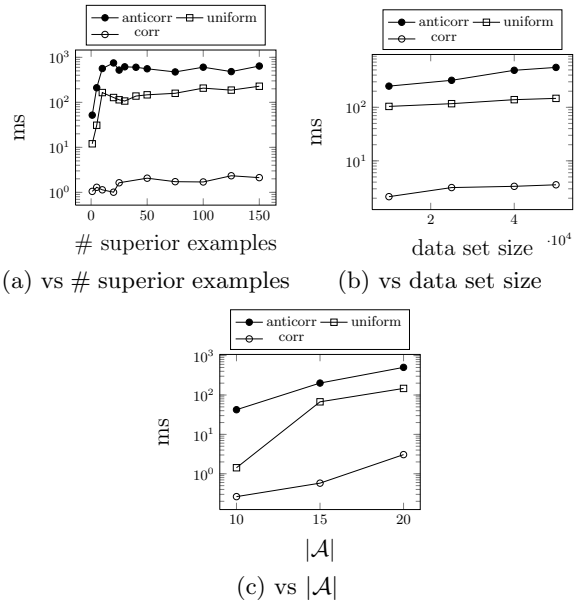
4.2 Experiments with synthetic data

The main focus of this section is to experiment with the scalability of the proposed approach and the reduction of large skyline sizes, achieved by modeling user preferences using p-skyline relations. We used three synthetic data sets $\mathcal{O}$ here: correlated, anticorrelated, and uniform. Each of them contained 50000 tuples. We used three different sets $\mathcal{A}$ of 10, 15, and 20 relevant attributes. For each of them, we picked different sets of superior examples G. We constructed such sets of *similar* tuples, where similarity was measured in terms of Euclidean distance. Given a set G, we used `discover` to compute optimal p-skyline relations $\succ$ favoring G. In Figure 10, we plot dependency of the average running time of `discover` on the number of superior examples $|G|$ used to discover a p-skyline relation (Figure 10(a), $|\mathcal{O}| = 50000$, $|\mathcal{A}| = 20$), the size of $\mathcal{O}$ (Figure 10(b), $|G| = 50$, $|\mathcal{A}| = 20$), and the number $|\mathcal{A}|$ of relevant attributes (Figure 10(c), $|\mathcal{O}| = 50000$, $|G| = 50$). The measured time does not include the time to construct the system of negative constraints and find the non-redundant constraints in it. According to our experiments, the preprocessing time predominantly depends on the performance of the skyline computation algorithm.

According to Figure 10(a), the algorithm running time increases until the size of G reaches 20. After that, it does not

vary much. This is due to the fact that the algorithm performance depends on the number of negative constraints used. We use only *non-redundant* constraints for discovery. As we show further, the dependence of the size of a system of non-redundant constraints on the number of superior examples has a pattern similar to Figure 10(a).

The growth of running time with the increase in the data set size (Figure 10(b)) is justified by the fact that the number of negative constraints depends on skyline size (Section 3.3.3). For the data sets used in the experiment, skyline sizes grow with the sizes of the corresponding data sets. Similarly, the running time of the algorithm grows with the number of relevant attributes (Figure 10(c)), due to the increase in the corresponding skyline size.



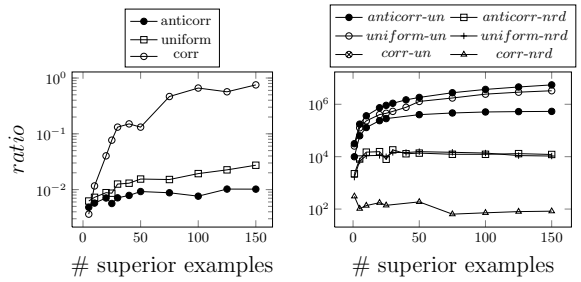
(a) vs # superior examples (b) vs data set size

(c) vs $|\mathcal{A}|$

Figure 10: Performance of p-skyline discovery

In Figure 11(a), we show how the size of the p-skyline query result varies with the number of superior examples used to discover a p-skyline relation $\succ$. We compare it with the size of the corresponding skyline and plot the value of *ratio* defined in the previous section. Here we used the anticorrelated, the uniform, and the correlated data sets of 50000 tuples each. The number of relevant attributes was 20. The size of the corresponding skylines was: 41716 (anticorrelated), 37019 (uniform), and 33888 (correlated). For the anticorrelated and the uniform data sets, the values of *ratio* quickly reach a certain bound and then grow slowly with the number of superior examples. This bound is approximately 1% of the skyline size (i.e., about 350 tuples) for both data sets. At the same time, the growth of *ratio* for the correlated data set is faster. Note that the values of *ratio* are generally lower for synthetic data sets in comparison to the real-life data set in the previous section. This is due to the larger set of relevant attributes and larger skyline sizes in the synthetic case.

In Figure 11(b), we show how the number of negative constraints depends on the number of superior examples used to construct them. For every data set, we plot two measures: the number of *unique* negative constraints in $\mathcal{N}(G, w_{sky_{\mathcal{H}}}(\mathcal{O}))$ (*anticorr-un*, *uniform-un*, and *corr-un*, resp.) and the number of *unique non-redundant* constraints



(a) p-skyline size reduction (b) Constraint # reduction

Figure 11: Synthetic data experiments

in the corresponding system (*anticorr-nrd*, *uniform-nrd*, and *corr-nrd*, resp.). Note that the reduction in the constraint number achieved using the methods we proposed in Section 3.3.3 is significant. Namely, for the anticorrelated data set and G of size 150, the total number of constraints in $\mathcal{N}(G, \mathcal{O})$ was approximately $7.5 \cdot 10^6$. Among them, about $5.5 \cdot 10^6$ were unique in $\mathcal{N}(G, w_{sky_{\mathcal{H}}}(\mathcal{O}))$. However, less than 1% of them (about $1.2 \cdot 10^4$) were non redundant.

The experiments show that incorporating attribute importance into skyline relations allows for significant reduction in query result size. The algorithm for p-skyline relation discovery has good scalability in terms of the data set size and the number of relevant attributes. The algorithm has a high accuracy even for small sets of superior examples used to discover p-skyline relations.

5. RELATED WORK

The p-skyline framework is based on the preference construction approach introduced in [15]. There, preferences are constructed from user-selected *preference constructors* using Pareto and prioritized accumulation operators. In the current work, we restrict them by requiring every atomic preference to be a total order and to be used *at most once* to construct a p-skyline relation. This allows for an intuitive way of capturing relative attribute importance using p-graphs. As far as we know, the problem of relative attribute importance in this framework has not been addressed yet.

An approach to mine preferences aggregated using the accumulation operators has been proposed in [13]. Web server logs are used to discover atomic preferences and accumulation operators used to aggregate them. The mining approach is based on statistical properties of log data – more preferable tuples appear more frequently. The authors provide heuristics for discovering atomic preferences on categorical and numerical domains.

A variation of the skyline framework which addresses the problem of large skyline size was considered in [23]. The authors propose to use *subspace skylines* (i.e., skylines over subsets of relevant attributes) whose sizes tend to be smaller than the size of the skyline over the entire set of relevant attributes. They also address the problem of finding subsets of relevant attributes guaranteeing a set of superior tuples to be in the corresponding subspace skyline. This work is orthogonal to the approach we propose here. Namely, we assume that the set of relevant attributes is *fixed* and show how to construct a maximal p-skyline relation favoring a set of tuples. A similarity between the frameworks is that for a certain class of p-skyline relations, the p-skyline queries may be represented as pipelined subspace skyline queries. In particular, those are the p-skyline relations whose p-graphs are weak orders (i.e., negatively transitive SPO). At the same

time, this does not apply to all p-skyline relations since the class of `SPO+Envelope` is more general than weak orders (e.g., $\succ_4$ from Example 9).

A framework of preference discovery which is complementary to the approach we propose here was presented in [14]. [14] was the first paper to introduce the scenario of mining preferences using superior and inferior examples. In that work, preferences are modeled as skyline relations. The authors focus on mining unknown atomic preferences using superior and inferior examples. They try to discover minimal (in terms of relation size) finite atomic preference relations. The authors show that the problem of existence of such relations is NP-complete, and the problem of computing them is NP-hard. They also provide two heuristics for computing such preferences. We note that these heuristics are greedy and may fail to return a satisfying set of preferences for certain sets of superior/inferior tuples. That approach and the approach we present here are different in the following sense. First, [14] deals with skyline relations, and thus all atomic preferences are considered to be equally important. The focus of our work is to discover differences in attribute importance. Second, the authors of [14] focus on mining minimal finite atomic preferences. In contrast, we are interested in computing maximal preference relations since this guarantees a better fit to a provided set of superior examples. At the same time, our work and [14] complement each other. Namely, when atomic preferences are not provided explicitly by the user, the approach [14] may be used to discover them.

Another approach of preference relation discovery in the skyline framework is proposed in [18]. That work is motivated by the problem of large skyline sizes. The authors propose to reduce skyline sizes through revising skyline relations by supplying additional tuple relationships: preference and equivalence. The relationships are obtained from user's answers to simple comparison questions. [18] provides an algorithm for constructing such questions.

A method of representing attribute priorities similar to p-graphs is presented in [4]. The authors identify two kinds of importance edges between attributes: *conditional* and *unconditional*. Preferences are modeled as conditional *ceteris paribus* statements.

In quantitative preference frameworks [8], attribute priorities are represented as *weight coefficients* in numeric utility functions. A number of methods have been proposed to elicit such utility functions [5, 3, 9]

6. CONCLUSIONS AND FUTURE WORK

In this paper, we studied the problem of discovering attribute importance in skylines. We proposed a framework which intuitively captures relative attribute importance using the notion of *p-skyline preference relation*. We showed that the problem of discovering optimal p-skyline preference relations based on superior and inferior examples is intractable in general. We also proposed a polynomial time algorithm for discovering such preference relations based on superior examples only. The experimental evaluation of the proposed framework using synthetic as well as real-life data shows its practicality.

According to our experiments, modeling user preferences as p-skyline relations allows for significant reduction in skyline size. At the same time, the superior tuples a user has to provide to discover such a relation have to accurately describe it. However, when the skyline is large, expecting that a user will be able to explore it entirely is impractical. Hence, there is a need to develop methods of suggesting good candidates for superior examples. Some promising measures of goodness of a skyline tuple are its *representativeness* [19], its *entropy value* [11], or the number of p-skyline relations favoring the tuple. This direction has to be thoroughly investigated. Another interesting direction of future work is to generalize our attribute importance discovery approach to *variable* sets of relevant attributes.

7. REFERENCES

[1] W.-T. Balke, W. Siberski, and U. Güntzer. Getting prime cuts from skylines over partially ordered domains. In *BTW*, pages 64–81, 2007.

[2] S. Börzsönyi, D. Kossmann, and K. Stocker. The skyline operator. In *ICDE*, pages 421–430, 2001.

[3] C. Boutilier. A POMDP formulation of preference elicitation problems. In *AAAI/IAAI*, pages 239–246, 2002.

[4] R. I. Brafman and C. Domshlak. Introducing variable importance tradeoffs into CP-nets. In *UAI*, 2002.

[5] U. Chajewska, D. Koller, and R. Parr. Making rational decisions using adaptive utility elicitation. In *AAAI/IAAI*, pages 363–369, 2000.

[6] J. Chomicki. Preference formulas in relational queries. *ACM Trans. Database Syst.*, 28(4):427–466, 2003.

[7] J. Chomicki, P. Godfrey, J. Gryz, and D. Liang. Skyline with presorting. In *ICDE*, pages 717–719, 2003.

[8] P. Fishburn. *Utility Theory for Decision-Making*. John Wiley & Sons, New York, 1970.

[9] K. Gajos and D. S. Weld. Preference elicitation for interface optimization. In *UIST*, pages 173–182, 2005.

[10] P. Godfrey. Skyline cardinality for relational processing. In *FoIKS*, volume 2942, pages 78–97, 2004.

[11] P. Godfrey, R. Shipley, and J. Gryz. Algorithms and analyses for maximal vector computation. *VLDB J.*, 16(1):5–28, 2007.

[12] B. Hafenrichter and W. Kießling. Optimization of relational preference queries. In *ADC*, pages 175–184, 2005.

[13] S. Holland, M. Ester, and W. Kießling. Preference mining: A novel approach on mining user preferences for personalized applications. In *PKDD*, pages 204–216, 2003.

[14] B. Jiang, J. Pei, X. Lin, D. W. Cheung, and J. Han. Mining preferences from superior and inferior examples. In *KDD*, pages 390–398, 2008.

[15] W. Kießling. Foundations of preferences in database systems. In *VLDB*, pages 311–322, 2002.

[16] W. Kießling. Preference queries with SV-semantics. In *COMAD*, pages 15–26, 2005.

[17] W. Kießling and G. Köstler. Preference SQL - Design, Implementation, Experiences. In *VLDB*, pages 990–1001, 2002.

[18] J. Lee, G. won You, S. won Hwang, J. Selke, and W.-T. Balke. Optimal preference elicitation for skyline queries over categorical domains. In *DEXA*, pages 610–624, 2008.

[19] X. Lin, Y. Yuan, Q. Zhang, and Y. Zhang. Selecting stars: The k most representative skyline operator. In *ICDE*, pages 86–95. IEEE, April 2007.

[20] D. Mindolin and J. Chomicki. P-Skyline Preference Relations. http://mindolin.info/pskyline/.

[21] M. D. Morse, J. M. Patel, and H. V. Jagadish. Efficient skyline computation over low-cardinality domains. In *VLDB*, pages 267–278, 2007.

[22] NHL.com stats. `http://www.nhl.com/ice/playerstats.htm`.

[23] 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 Syst.*, 31(4):1335–1381, 2006.