Preference Queries[*]

Jan Chomicki

Dept. of Computer Science and Engineering

University at Buffalo

Buffalo, NY 14260-2000

`chomicki@cse.buffalo.edu`

17th April 2017

Abstract

The handling of user preferences is becoming an increasingly important issue in present-day information systems. Among others, preferences are used for *information filtering and extraction* to reduce the volume of data presented to the user. They are also used to keep track of *user profiles* and formulate *policies* to improve and automate decision making.

We propose here a simple, logical framework for formulating preferences as *preference formulas*. The framework does not impose any restrictions on the preference relations and allows arbitrary operation and predicate signatures in preference formulas. It also makes the composition of preference relations straightforward. We propose a simple, natural embedding of preference formulas into relational algebra (and SQL) through a single *winnow* operator parameterized by a preference formula. The embedding makes possible the formulation of complex preference queries, e.g., involving aggregation, by piggybacking on existing SQL constructs. It also leads in a natural way to the definition of further, preference-related concepts like ranking. Finally, we present general algebraic laws governing the winnow operator and its interaction with other relational algebra operators. The preconditions on the applicability of the laws are captured by logical formulas. The laws provide a formal foundation for the algebraic optimization of preference queries. We demonstrate the usefulness of our approach through numerous examples.

1 Introduction

The handling of user preferences is becoming an increasingly important issue in present-day information systems. Among others, preferences are used for *information filtering and extraction* to reduce the volume of data presented to the user. They are also used to keep track of *user profiles* and formulate *policies* to improve and automate decision making.

---

[*]This is an expanded version of the paper [8]. CoRR paper cs.DB/0207093.

The research literature on preferences is extensive. It encompasses preference logics [29, 25, 18], preference reasoning [30, 28, 5], prioritized nonmonotonic reasoning and logic programming [6, 11, 27] and decision theory [12, 13] (the list is by no means exhaustive). However, only a few papers [24, 4, 16, 2, 19, 8, 22, 23], most of them very recent, address the issue of user preferences in the context of database queries. Two different approaches are pursued: qualitative and quantitative. In the *qualitative* approach [24, 4, 16, 8, 22, 23], the preferences between tuples in the answer to a query are specified directly, typically using binary *preference relations*.

Example 1.1 *We introduce here one of the examples used throughout the paper. Consider the relation* $Book(ISBN, Vendor, Price)$ *and the following preference relation* $\succ_1$ *between* Book *tuples:*

> prefer one Book tuple to another if and only if their ISBNs are the same and the Price of the first is lower.

Consider the following instance r_1 *of Book*

ISBN	Vendor	Price
0679726691	*BooksForLess*	*$14.75*
0679726691	*LowestPrices*	*$13.50*
0679726691	*QualityBooks*	*$18.80*
0062059041	*BooksForLess*	*$7.30*
0374164770	*LowestPrices*	*$21.88*

Then clearly the second tuple is preferred to the first one which in turn is preferred to the third one. There is no preference defined between any of those three tuples and the remaining tuples.

In the *quantitative* approach [2, 19], preferences are specified indirectly using *scoring functions* that associate a numeric score with every tuple of the query answer. Then a tuple t_1 is preferred to a tuple t_2 iff the score of t_1 is higher than the score of t_2. The qualitative approach is strictly more general than the quantitative one, since one can define preference relations in terms of scoring functions (if the latter are explicitly given), while not every intuitively plausible preference relation can be captured by scoring functions.

Example 1.2 *There is no scoring function that captures the preference relation described in Example 1.1. Since there is no preference defined between any of the first three tuples and the fourth one, the score of the fourth tuple should be equal to all of the scores of the first three tuples. But this implies that the scores of the first three tuples are the same, which is not possible since the second tuple is preferred to the first one which in turn is preferred to the third one.*

This lack of expressiveness of the quantitative approach is well known in utility theory [12, 13].

In the present paper, we contribute to the qualitative approach by defining a logical framework for formulating preferences and its embedding into relational query languages.

We believe that combining preferences with queries is very natural and useful. The applications in which user preferences are prominent will benefit from applying the modern database technology. For example, in decision-making applications databases may be used to store the space of possible configurations. Also, the use of a full-fledged query language makes it possible to formulate complex decision problems, a feature missing from most previous, non-database, approaches to preferences. For example, the formulation of the problem may now involve quantifiers, grouping, or aggregation. At the same time by explicitly addressing the technical issues involved in querying with preferences present-day DBMS may expand their scope.

The framework presented in this paper consists of two parts: a formal first-order logic notation for specifying preferences and an embedding of preferences into relational query languages. In this way both abstract properties of preferences (like asymmetry or transitivity) and evaluation of preference queries can be studied to a large degree separately.

Preferences are defined using binary *preference relations* between tuples. Preference relations are specified using first-order formulas. We focus mostly on *intrinsic* preference formulas. Such formulas can refer only to built-in predicates. In that way we capture preferences that are based only on the values occuring in tuples, not on other properties like membership of tuples in database relations. We show how the latter kind of preferences, called *extrinsic*, can also be simulated in our framework in some cases.

We propose a new relational algebra operator called *winnow* that selects from its argument relation the *most preferred tuples* according to the given preference relation. Although the winnow operator can be expressed using other operators of relational algebra, by considering it on its own we can on one hand focus on the abstract properties of preference relations (e.g., transitivity) and on the other, study special evaluation and optimization techniques for the winnow operator itself. For SQL. we are faced with a similar choice: either the language is appropriately extended with an SQL equivalent of winnow, or the occurrences of winnow are translated into SQL. The first alternative looks more promising; however, in this paper we don't commit ourselves to any specific syntactic expression of winnow in SQL.

We want to capture many different varieties of preference and related notions: *unconditional* vs. *conditional* preferences, *nested* and *hierarchical* preferences, *groupwise* preferences, *indifference*, *iterated* preferences and *ranking*, and *integrity constraints* and *vetoes*.

The main contributions of this paper are as follows:

1. a simple, logical framework for formulating preferences as preference formulas. The framework does not impose any restrictions on the preference relations and allows arbitrary operation and predicate signatures in preference formulas. It also makes the composition of preference relations straightforward.

2. a simple, natural embedding of preference formulas into relational algebra (and SQL) through a single winnow operator parameterized by a preference formula. The em-

3

bedding makes possible the formulation of complex preference queries, e.g., involving aggregation, by piggybacking on existing SQL constructs. It also leads in a natural way to the definition of further, preference-related concepts like ranking.

3. general algebraic laws governing the winnow operator and its interaction with other relational algebra operators. The preconditions on the applicability of the laws are captured by logical formulas. The laws provide a formal foundation for the algebraic optimization of preference queries.

In Section 2, we define the basic concepts of preference relation, preference formula, and the winnow operator. We also introduce several examples that will be used throughout the paper. In Section 3, we study the basic properties of preference relations. In Section 4, which contains the main technical contributions of the paper, we present the main properties of the winnow operator, characterize its expressive power, and outline – for completeness – a number of evaluation algorithms that were proposed elsewhere. In Section 5, we explore the composition of preferences. In Section 6, we show how the winnow operator together with other constructs of relational algebra and SQL makes it possible to express a wide variety of preference queries. In Section 7, we show how iterating the winnow operator provides a ranking of tuples and introduce a weak version of the winnow operator that is helpful for preference relations that are not strict partial orders. We discuss related work in Section 8 and conclude with a brief discussion of further work in Section 9. All the non-trivial proofs are given.

2 Basic notions

We are working in the context of the relational model of data. We assume two infinite domains: D (uninterpreted constants) and N (numbers). We do not distinguish between different numeric domains, since it is not necessary for the present paper. When necessary, we assume that database instances are finite. (Some results hold without the finiteness assumption.) Additionally, we have the standard built-in predicates. In the paper, we will move freely between relational algebra and SQL.

2.1 Basic definitions

Preference formulas are used to define binary preference relations.

Definition 2.1 *Given a relation schema $R(A_1 \cdots A_k)$ such that U_i, $1 \leq i \leq k$, is the domain (either D or N) of the attribute A_i, a relation $\succ$ is a* preference relation over R *if it is a subset of $(U_1 \times \cdots \times U_k) \times (U_1 \times \cdots \times U_k)$.*

Intuitively, $\succ$ will be a binary relation between pairs of tuples from the same (database) relation. We say that a tuple t_1 *dominates* a tuple t_2 in $\succ$ if $t_1 \succ t_2$.

Typical properties of the relation $\succ$ include:

- *irreflexivity*: $\forall x. x \not\succ x$,

- *asymmetry*: $\forall x, y. x \succ y \Rightarrow y \not\succ x$,

- *transitivity*: $\forall x, y, z. (x \succ y \wedge y \succ z) \Rightarrow x \succ z$,

- *negative transitivity*: $\forall x, y, z. (x \not\succ y \wedge y \not\succ z) \Rightarrow x \not\succ z$,

- *connectivity*: $\forall x, y. x \succ y \vee y \succ x \vee x = y$.

The relation $\succ$ is: a *strict partial order* if it is irreflexive, asymmetric and transitive; a *total order* if it is a connected strict partial order; a *weak order* if it is is a negatively transitive strict partial order. At this point, we do not assume any properties of $\succ$, although in most applications it will satisfy at least the properties of a strict partial order.

Definition 2.2 *A* preference formula (pf) $C(t_1, t_2)$ *is a first-order formula defining a preference relation* $\succ_C$ *in the standard sense, namely*

$$t_1 \succ_C t_2 \text{ iff } C(t_1, t_2).$$

An intrinsic preference formula (ipf) *is a preference formula that uses only built-in predicates.*

We will limit our attention to preference relations defined using preference formulas. By using the notation $\succ_C$ for a preference relation, we assume that there is an underlying preference formula C.

Ipfs can refer to equality ($=$) and inequality ($\neq$) when comparing values that are uninterpreted constants, and to the standard set of built-in arithmetic comparison operators when referring to numeric values (there are no function symbols). We will call an ipf that references only arithmetic comparisons ($=, \neq, <, >, \leq, \geq$) *pure comparison*. Without loss of generality, we will assume that ipfs are in DNF (Disjunctive Normal Form) and quantifier-free (the theories involving the above predicates admit quantifier elimination). A formula in DNF is called k-DNF if it has at most k disjuncts.

In this paper, we mostly restrict ourselves to ipfs and preference relations defined by such formulas. The main reason is that ipfs define *fixed*, although possibly infinite, relations. As a result, they are computationally easier and more amenable to syntactic manipulation that general pfs. For instance, transitively closing an ipf results in a finite formula (Theorem 5.3), which is typically not the case for pfs. However, we formulate in full generality the results that hold for arbitrary pfs.

We define now an algebraic operator that picks from a given relation the set of the *most preferred tuples*, according to a given preference formula.

Definition 2.3 *If R is a relation schema and C a preference formula defining a preference relation $\succ_C$ over R, then the* winnow operator *is written as $\omega_C(R)$, and for every instance r of R:*

$$\omega_C(r) = \{t \in r \mid \neg \exists t' \in r.\ t' \succ_C t\}.$$

A preference query is a relational algebra query containing at least one occurrence of the winnow operator.

2.2 Examples

The first example illustrates how preference queries are applied to *information extraction*: here obtaining the best price of a given book.

Example 2.1 *Consider the relation $Book(ISBN, Vendor, Price)$ from Example 1.1. The preference relation $\succ_{C_1}$ from this example can be defined using the formula C_1:*

$$(i, v, p) \succ_{C_1} (i', v', p') \equiv i = i' \wedge p < p'.$$

The answer to the preference query $\omega_{C_1}(Book)$ provides for every book the information about the vendors offering the lowest price for that book. For the given instance r_1 of $Book$, applying the winnow operator ω_{C_1} returns the tuples

ISBN	Vendor	Price
0679726691	*LowestPrices*	*$13.50*
0062059041	*BooksForLess*	*$7.30*
0374164770	*LowestPrices*	*$21.88*

Note that in the above example, the preferences are applied groupwise: *separately for each book. Note also that due to the properties of $<$, the preference relation $\succ_{C_1}$ is irreflexive, asymmetric and transitive.*

The second example illustrates how preference queries are used in *automated decision making* to obtain the most desirable solution to a (very simple) configuration problem.

Example 2.2 *Consider two relations $Wine(Name, Type)$ and $Dish(Name, Type)$ and a view $Meal$ that contains possible meal configurations*

```
CREATE VIEW Meal(Dish,DishType,Wine,WineType) AS
  SELECT * FROM Wine, Dish;
```

Now the preference for white wine in the presence of fish and for red wine in the presence of meat can be expressed as the following preference formula C_2 over $Meal$:

$$
\begin{aligned}
(d, dt, w, wt) \succ_{C_2} (d', dt', w', wt') \equiv \ & (d = d' \wedge dt = {}'\text{fish}' \wedge wt = {}'\text{white}' \\
& \wedge dt' = {}'\text{fish}' \wedge wt' = {}'\text{red}') \\
& \vee (d = d' \wedge dt = {}'\text{meat}' \wedge wt = {}'\text{red}' \\
& \wedge dt' = {}'\text{meat}' \wedge wt' = {}'\text{white}')
\end{aligned}
$$

Notice that this will force any white wine to be preferred over any red wine for fish, and just the opposite for meat. For other kinds of dishes, no preference is indicated. This is an example of a relative *preference. Consider now the preference query $\omega_{C_2}(Meal)$. It will pick the most preferred meals, according to the above-stated preferences. Notice that in the absence of any white wine, red wine can be selected for fish.*

The above preferences are conditional, since they depend on the type of the dish being considered. Note that the relation $\succ_{C_2}$ in this example is irreflexive and asymmetric. Transitivity is obtained trivially because the chains of $\succ_{C_2}$ are of length at most 2. Note also that the preference relation is defined without referring to any domain order.

Note also that the meals with a wine which is neither red nor white but, e.g., rosé, are not related through $\succ_{C_2}$ to the meals with either of those kinds of wine. Therefore, the preference query $\omega_{C_2}(Meal)$ will return also the meals involving such wines, as they are not dominated by other meals. If this is undesirable, one can express an absolute preference for white wine for fish (and red wine for meat) using the formula C_3:

$$
\begin{aligned}
(d, dt, w, wt) \succ_{C_3} (d', dt', w', wt') \quad \equiv \quad & (d = d' \wedge dt = {}'\text{fish}' \wedge wt = {}'\text{white}' \\
& \wedge dt' = {}'\text{fish}' \wedge wt' \neq {}'\text{white}') \\
& \vee (d = d' \wedge dt = {}'\text{meat}' \wedge wt = {}'\text{red}' \\
& \wedge dt' = {}'\text{meat}' \wedge wt' \neq {}'\text{red}')
\end{aligned}
$$

Similarly, an unconditional preference for red wine for any kind of meal can also be defined as a first-order formula C_4:

$$
(d, dt, w, wt) \succ_{C_4} (d', dt', w', wt') \quad \equiv \quad d = d' \wedge wt = {}'\text{red}' \wedge wt' \neq {}'\text{red}'.
$$

3 Properties of preference queries

3.1 Preference relations

Since pfs can be essentially arbitrary formulas, no properties of preference relations can be assumed. So our framework is entirely neutral in this respect.

In the examples above, the preference relations were strict partial orders. This is likely to be the case for most applications of preference queries. However, there are cases where such relations fail to satisfy one of the properties of partial orders. We will see in Section 6 when irreflexivity fails. For asymmetry: We may have two tuples t_1 and t_2 such that $t_1 \succ t_2$ and $t_2 \succ t_1$ simply because we may have one reason to prefer t_1 over t_2 and another reason to prefer t_2 over t_1. Similarly, transitivity is not always guaranteed [20, 25, 12, 18]. For example, t_1 may be preferred over t_2 and t_2 over t_3, but the gap between t_1 and t_3 with respect to some heretofore ignored property may be so large as to prevent preferring t_1 over t_3. Or, transitivity may have to be abandoned to prevent cycles in preferences. However, transitivity is essential for the correctness of the algorithms that compute winnow (Section 4).

It is not difficult to check the properties of a preference relation defined using a pure comparison ipf.

Theorem 3.1 *If a preference relation is defined using a pure comparison ipf in DNF, it can be checked in PTIME for irreflexivity and asymmetry. If the ipf is also in k-DNF for some fixed k, then the preference relation can be checked in PTIME for transitivity, negative transitivity, and connectivity.*

Proof: We discuss first asymmetry, the remaining properties can be handled in a similar way. If $t_1 \succ t_2$ is defined as $D_1 \vee \ldots \vee D_m$ and $t_2 \succ t_1$ as $D'_1 \vee \ldots \vee D'_m$, we can write down the negation of asymmetry as $(D_1 \vee \ldots \vee D_m) \wedge (D'_1 \vee \ldots \vee D'_m)$. This formula is satisfiable iff at least one of m^2 formulas $\phi_{i,j} \equiv D_i \wedge D'_j$, $i,j = 1, \ldots, m$, is satisfiable. Each formula $\phi_{i,j}$ is a conjunction of atomic formulas involving arithmetic comparison predicates. Thus its satisfiability can be checked in PTIME using the methods of [17]. Testing for transitivity, negative transitivity and connectivity requires writing down the negation of a DNF formula and distributing the negation inside. The restriction to k-DNF guarantees that we have again a polynomial number of PTIME satisfiability problems. $\qquad\square$

Theorem 3.2 *If a preference relation $\succ_C$ over R is a strict partial order, then for every finite, nonempty instance r of R, $\omega_C(r)$ is nonempty.*

If the properties of strict partial orders are not satisfied, then Theorem 3.2 may fail to hold and the winnow operator may return an empty set, even though the relation to which it is applied is nonempty. For instance, if $r_0 = \{t_0\}$ and $t_0 \succ t_0$ (violation of irreflexivity), then the winnow operator applied to r_0 returns an empty set. Similarly, if two tuples are involved in a violation of asymmetry, they may block each other from appearing in the result of the winnow operator. Also, if the relation r is infinite, it may happen that $\omega_C(r) = \emptyset$, for example if r contains all natural numbers and the preference relation is the standard ordering $>$.

The winnow operator is not *monotone* or *anti-monotone*.

Example 3.1 *Consider the following preference formula C_6:*

$$x \succ_{C_6} y \equiv x = a \wedge y = b.$$

Then

$$b = \omega_{C_6}(\{b\}) \not\subseteq \omega_{C_6}(\{a, b\}) = a.$$

Thus monotonicity and anti-monotonicity fail.

However, a form of monotonicity with respect to the preference formula parameter holds for winnow.

Theorem 3.3 *If $\succ_{C_1}$ and $\succ_{C_2}$ are preference relations over a relation schema R, and the formula*

$$\forall t_1, t_2 [C_1(t_1, t_2) \Rightarrow C_2(t_1, t_2)]$$

is valid, then for all instances r of R, $\omega_{C_2}(r) \subseteq \omega_{C_1}(r)$. If $\succ_{C_1}$ and $\succ_{C_2}$ are strict partial orders, then the converse also holds.

Proof: The first part is obvious. To see that the second part also holds, assume that for all relations r, $\omega_{C_2}(r) \subseteq \omega_{C_1}(r)$ but $C_1 \not\Rightarrow C_2$. Thus, $C_1 \wedge \neg C_2$ is satisfiable, and there are two tuples t_1 and t_2 such $t_1 \succ_{C_1} t_2$ but $t_1 \not\succ_{C_2} t_2$. Consider now the instance $r_{12} = \{t_1, t_2\}$. Then $\omega_{C_1}(r_{12}) = \{t_1\}$ but $t_2 \in \omega_{C_2}(r_{12})$, a contradiction. $\qquad\square$

Several properties of winnow follow directly from the definition (the first is listed in [22], although in a less general context):

Proposition 3.1 *For every preference relations $\succ_{C_1}$ and $\succ_{C_2}$ over a relation schema R and every instance r of R:*

$$\omega_{C_1 \vee C_2}(r) = \omega_{C_1}(r) \cap \omega_{C_2}(r)$$
$$\omega_{False}(r) = r$$
$$\omega_{True}(r) = \emptyset.$$

3.2 Indifference

There is a natural notion of *indifference* associated with our approach: two tuples t_1 and t_2 are *indifferent* ($t_1 \sim_C t_2$) if neither is preferred to the other one, i.e., $t_1 \not\succ_C t_2$ and $t_2 \not\succ_C t_1$.

Proposition 3.2 *For every preference relation $\succ_C$, every relation r and every tuple $t_1, t_2 \in \omega_C(r)$, we have $t_1 = t_2$ or $t_1 \sim_C t_2$.*

It is a well-known result in decision theory [12, 13] that in order for a preference relation to be representable using scoring functions the relation has to be a weak order. This implies, in particular, that the corresponding indifference relation (defined as above) has to be *transitive*. This is not the case for the preference relation $\succ_{C_1}$ defined in Example 1.1.

4 The winnow operator

In this section, we study various properties of the winnow operator: expressive power, monotonicity, commutativity and distributivity. Formulating such properties is essential for the evaluation and optimization of preference queries. We also briefly discuss some evaluation methods for winnow.

Although, as we show, the winnow operator can be expressed in relational algebra, its explicit use makes possible a clean separation of preference formulas from other aspects of the query. This has several advantages. First, the properties of preference relations can be studied in an abstract way. Second, specialized query evaluation methods for the winnow operator can be developed. Third, algebraic properties of that operator can be formulated, in order to be used in query optimization.

4.1 Expressive power

The winnow operator can be expressed in relational algebra, and thus does not add any expressive power to it. Perhaps more surprisingly, winnow can be used to simulate set difference.

Theorem 4.1 *Relational algebra with winnow replacing set difference has the same expressive power as standard relational algebra.*

9

Proof: Clearly, the winnow operator is first-order definable. Thus any relational algebra query with winnow can be translated to relational calculus, and then back to relational algebra (without winnow). Such a construction is, however, mainly of theoretical importance.

From a practical point of view, we show now the translation of the winnow operator $\omega_C(r)$ for $C = D_1 \vee \ldots \vee D_k$ which is a pure comparison ipf formula in DNF. Each D_i, $i = 1, \ldots, k$, is a formula over free variables t_1 and t_2. It can be viewed as a conjunction $D_i \equiv \phi_i \wedge \psi_i \wedge \gamma_i$ where ϕ_i refers only to the variables of t_1, ψ_i to the variables of t_2, and γ_i to the variables of both t_1 and t_2. The formula ϕ_i has an obvious translation to a selection condition Φ_i over R, and the formula ψ_i a similar translation to a selection condition Ψ_i over $\varrho(R)$, where ϱ is a renaming of R. The formula γ_i can similarly be translated to a join condition Γ_i over R and $\varrho(R)$. Then

$$\omega_C(R) = \varrho^{-1}(\varrho(R) - \pi_{\varrho(R)}(\bigcup_{i=1}^{k}(\sigma_{\Phi_i}(R) \underset{\Gamma_i}{\bowtie} \sigma_{\Psi_i}(\varrho(R)))))$$

where ϱ^{-1} is the inverse of the renaming ϱ.

We show now how to simulate the set difference operator $R - S$ using winnow. Assume that R (and S) have the set of attributes X of arity k. Then

$$R - S = \pi_X(\sigma_{B \neq 0}(\omega_{C_5}(R \times \{1\} \cup S \times \{0\})))$$

where B is the last attribute of $R \times \{1\}$ and

$$(x_1, \ldots, x_k, b) \succ_{C_5} (x_1', \ldots, x_k', b') \equiv x_1 = x_1' \wedge \cdots \wedge x_k = x_k' \wedge b = 0 \wedge b' = 1.$$

This works as follows. Think of the attribute B as a tag. All the tuples in R (resp. S) are tagged with 1 (resp. 0). If a tuple is in $R \cap S$, then there are two copies of it in $R \times \{1\} \cup S \times \{0\}$: one tagged with 1, the other with 0. The latter one is preferred according to $\succ_{C_5}$. Finally, the selection $\sigma_{B \neq 0}$ eliminates all the tuples in S, keeping the tuples that are only in R. □

4.2   Evaluating winnow

For completeness, we show here several algorithms that can be used to compute the result of the winnow operator $\omega_C(r)$. The first is a simple nested-loops algorithm (Figure 1). The second is BNL, an algorithm proposed in [4] in the context of *skyline queries*, a specific class of preference queries, but the algorithm is considerably more general (Figure 2). The third [9] is a variant of the second, in which a presorting step is used (Figure 3). All the algorithms used a fixed amount of main memory (a *window*). However, for the algorithm NL, this is not made explicit, since it is irrelevant for the properties of the algorithm that are of interest here. Our emphasis is not on the algorithms themselves – they are much more completely described and analyzed in the original papers – but rather on determining their scope. We will identify the classes of preference queries to which each of them is applicable.

---

1. open a scan S_1 on r;

2. for every tuple t_1 returned by S_1:

   (a) open a scan S_2 on r;

   (b) for every tuple t_2 returned by S_1:

        if $t_2 \succ_C t_1$, then close S_2 and goto 2d;

   (c) output t_1;

   (d) close S_2;

3. close S_1.

Figure 1: NL: Nested Loops

---

1. initialize the window W and the temporary table F to empty;

2. make r the input;

3. repeat the following until the input is empty:

   (a) for every tuple t :
   - t is dominated by a tuple in $W \Rightarrow$ ignore t,
   - t dominates some tuples in $W \Rightarrow$ eliminate the dominated tuples and insert t into W,
   - t is incomparable with all tuples in $W \Rightarrow$ insert t into W (if there is room), otherwise add t to F;

   (b) output the tuples from W that were added there when F was empty,

   (c) make F the input, clear F.

Figure 2: BNL: Blocked Nested Loops

The NL algorithm is correct for any preference relation $\succ_C$. In principle, the preference relation might even be reflexive, since the algorithm compares a tuple with itself. The BNL and SFS algorithms require the preference relation to be a strict partial order (for BNL this is noted in [4]). The algorithms require irreflexivity, because they do not compare a tuple with itself. Neither do they handle correctly symmetry: the situation where there are two tuples t_1 and t_2 such that $t_1 \succ_C t_2$ and $t_2 \succ_C t_1$. In this case, BNL will break the tie depending on the order in which the tuples appear, and SFS will fail altogether, being unable to produce a topological sort. To see the necessity of transitivity, consider the following example.

Example 4.1 *The preference relation C_0 is defined as follows:*

$$x \succ_{C_0} y \equiv x = a \wedge y = b \vee x = b \wedge y = c.$$

---

1. topologically sort r according to $\succ_C$;

2. make r the input;

3. initialize the window W and the temporary table F to empty;

4. repeat the following until the input is empty:

    (a) for every tuple t in the input:

- t is dominated by a tuple in W $\Rightarrow$ ignore t,
- t is incomparable with all tuples in W $\Rightarrow$ insert t into W (if there is room), otherwise add t to F;

    (b) output the tuples from W.

    (c) make F the input, clear F.

Figure 3: SFS: Sort-Filter-Skyline

Now let us suppose that the window has room for only one tuple, and the tuples arrive in the following order: a, b, c. Then a will be in the window, and b will be discarded, which prevents b from blocking c. Therefore, BNL will output a (correctly) and c (incorrectly). Such an example can be easily generalized to any fixed window size, simply be assuming that a and b are separated in the input by sufficiently many values different from a, b and c.

4.3 Algebraic laws

We present here a set of algebraic laws that govern the commutativity and distributivity of winnow w.r.t. relational algebra operators. This set constitutes a formal foundation for rewriting preference queries using the standard strategies like *pushing selections down*. We prove the soundness of the introduced laws. In the cases of selection, projection, union and difference, we show that the preconditions on the applicability of the laws are not only sufficient but also necessary. In the remaining cases, we show that the violations of the preconditions lead to the violations of the laws. In most interesting cases, the preconditions can also be efficiently checked.

We adopt the set-based view of relational algebra operators and leave exploring the multiset-based view for future research.

4.3.1 Commutativity of winnow

We establish here a sufficient condition for winnow to be commutative. Commutativity is a fundamental property that makes it possible to move the winnow operator around in preference queries.

Theorem 4.2 *If C_1 and C_2 are preference formulas over a schema R such that*

- *the formula $\forall t_1, t_2[C_1(t_1, t_2) \Rightarrow C_2(t_1, t_2)]$ is valid, and*

12

- $\succ_{C_1}$ and $\succ_{C_2}$ are strict partial orders,

then for all finite instances r of R:

$$\omega_{C_1}(\omega_{C_2}(r)) = \omega_{C_2}(\omega_{C_1}(r)) = \omega_{C_2}(r).$$

Proof: We prove here the first equality; the second can be proved in a similar way.

Assume $t \notin \omega_{C_2}(\omega_{C_1}(r))$ and $t \in \omega_{C_1}(\omega_{C_2}(r))$. Then also $t \in \omega_{C_2}(r)$. There are two possibilities: (1) $\exists t' \in \omega_{C_1}(r)$ such that $t' \succ_{C_2} t$. But then $t' \in r$, which contradicts the fact that $t \in \omega_{C_2}(r)$. (2) $t \notin \omega_{C_1}(r)$. But then by Theorem 3.3 $t \notin \omega_{C_2}(r)$, a contradiction.

Assume $t \notin \omega_{C_1}(\omega_{C_2}(r))$ and $t \in \omega_{C_2}(\omega_{C_1}(r))$. Then also $t \in \omega_{C_1}(r)$. There are two possibilities: (1) $\exists t' \in \omega_{C_2}(r)$ such that $t' \succ_{C_1} t$. But then also $t' \in r$, which contradicts the fact that $t \in \omega_{C_1}(r)$. (2) $t \notin \omega_{C_2}(r)$. Still $t \in r$, since otherwise $t \notin \omega_{C_1}(r)$. Therefore, $\exists t' \in r$ such that $t' \succ_{C_2} t$. Now because $\succ_{C_2}$ is a strict partial order and r is finite, we can choose $t' \in \omega_{C_2}(r)$. If $t' \in \omega_{C_1}(r)$, then in view of the fact that $t \in \omega_{C_1}(r)$ and $t' \succ_{C_2} t$, we get a contradiction. On the other hand, if $t' \notin \omega_{C_1}(r)$, then by Theorem 3.3 we get $t' \notin \omega_{C_2}(r)$, a contradiction. $\square$

Consider now what happens if the assumptions in Theorem 4.2 are relaxed.

Example 4.2 *Let $Emp(EmpNo, YearEmployed, Salary)$ be a relation schema. Define the following preference relations over it:*

$$(e, y, s) \succ_{C_1} (e', y', s') \equiv s > s'$$

and

$$(e, y, s) \succ_{C_2} (e', y', s') \equiv y < y'.$$

Clearly, neither $C_1 \Rightarrow C_2$ nor $C_2 \Rightarrow C_1$. The database $r_1 = \{(1, 1975, 100K), (2, 1980, 150K)\}$. Now

$$\omega_{C_1}(\omega_{C_2}(r) = (1, 1975, 100K)) \neq (2, 1980, 150K) = \omega_{C_2}(\omega_{C_1}(r)).$$

Example 4.3 *Consider the following preference relations:*

$$x \succ_{C_1} y \equiv x = a \wedge y = b$$

and

$$x \succ_{C_2} y \equiv x = a \wedge y = b \vee x = b \wedge y = a.$$

Clearly, $C_1 \Rightarrow C_2$. However, $\succ_{C_2}$ is not a strict partial order. We have

$$\omega_{C_1}(\omega_{C_2}(r)) = \emptyset \neq \{a\} = \omega_{C_2}(\omega_{C_1}(r)).$$

In Theorem 4.2, if the preference formula C_2 is a pure comparison ipf in k-DNF, then checking the validity of the formula $\forall t_1, t_2[C_1(t_1, t_2) \Rightarrow C_2(t_1, t_2)]$ can be done in PTIME.

13

4.3.2 Commuting selection and winnow

We identify in Theorem 4.3 below a sufficient and necessary condition under which the winnow operator and a relational algebra selection commute. This is helpful for pushing selections past winnow operators in preference queries. It is well known that moving selections down in the query tree reduces the size of (and the time needed to materialize) intermediate results and has a potential of enabling the use of indexes (if a selection is pushed all the way down to a database relation that has an index matching the selection condition).

Theorem 4.3 *Given a relation schema R, a selection condition C_1 over R and a preference formula C_2 over R, if the formula*

$$\forall t_1, t_2[(C_1(t_2) \wedge C_2(t_1, t_2)) \Rightarrow C_1(t_1)]$$

is valid, then for all instances r of R:

$$\sigma_{C_1}(\omega_{C_2}(r)) = \omega_{C_2}(\sigma_{C_1}(r)).$$

The converse holds under the assumption that $\succ_{C_2}$ is irreflexive.

Proof: We have that:

$$t \in \sigma_{C_1}(\omega_{C_2}(r)) \equiv t \in r \wedge C_1(t) \wedge (\neg \exists t'[t' \in r \wedge C_2(t', t)]).$$

On the other hand:

$$t \in \omega_{C_2}(\sigma_{C_1}(r)) \equiv t \in r \wedge C_1(t) \wedge (\neg \exists t'[t' \in r \wedge C_2(t') \wedge C_2(t', t)]).$$

Clearly, the first formula implies the second. To see that the opposite direction also holds, assume that there is a tuple t_0 such that $t_0 \in r$ and $C_2(t_0, t)$ holds. $C_1(t)$ holds, thus $C_1(t_0)$ holds too, since otherwise the formula $\forall t_1, t_2[(C_1(t_2) \wedge C_2(t_1, t_2)) \Rightarrow C_1(t_1)]$ would not be valid.

To see the necessity of the condition of the theorem, assume that there are tuples t_1 and t_2 such that $C_1(t_2) \wedge C_2(t_1, t_2) \wedge \neg C_1(t_1)$. Then

$$\omega_{C_2}(\sigma_{C_1}(\{t_1, t_2\})) = \{t_2\} \neq \emptyset = \sigma_{C_1}(\omega_{C_2}(\{t_1, t_2\})).$$

The irreflexivity of $\succ_{C_2}$ is necessary to ensure that $\omega_{C_2}(\sigma_{C_1}(\{t_1, t_2\}))$ is nonempty. □

If the preference formula C_2 in Theorem 4.3 is a pure comparison ipf and the selection condition C_1 is in k-DNF and refers only to the arithmetic comparison predicates, then checking the validity of the formula $\forall(C_1(t_2) \wedge C_2(t_1, t_2)) \Rightarrow C_1(t_1)$ can be done in PTIME.

Example 4.4 *Consider the relation $Book(ISBN, Vendor, Price)$ from Example 1.1. The preference relation $\succ_{C_1}$ is defined as*

$$(i, v, p) \succ_{C_1} (i', v', p') \equiv i = i' \wedge p < p'.$$

Consider the query $\sigma_{Price<15}(\omega_{C_1}(Book))$. Now

$$\forall p, p', i, i'[(p' < 15 \wedge i = i' \wedge p < p') \Rightarrow p < 15]$$

is a valid formula, thus by Theorem 4.3

$$\omega_{C_1}(\sigma_{Price<15}(Book)) = \sigma_{Price<15}(\omega_{C_1}(Book)).$$

On the other hand, consider the query $\sigma_{Price>15}$. Then

$$\forall p, p', i, i'[(p' > 15 \wedge i = i' \wedge p < p') \Rightarrow p > 15]$$

is not a valid formula, thus in this case the selection does not commute with winnow. Finally, the query $\sigma_{ISBN=c}$ for any string c commutes with with $\omega_{C_1}(Book)$, because

$$\forall p, p', i, i'[(i = c \wedge i = i' \wedge p < p') \Rightarrow i' = c]$$

is a valid formula.

4.3.3 Commuting projection and winnow

We deal now with projection. For winnow to commute with projection, the preference formula needs to be restricted to the attributes in the projection. We denote by $t[X]$ the tuple $(t[A_1], \ldots, t[A_k])$, where $X = A_1 \cdots A_k$ is a set of attributes.

Definition 4.1 *Given a relation schema R, a set of attributes X of R, and a preference relation $\succ_C$ over R, the restriction $\theta_x(\succ_C)$ of $\succ_C$ to X is a preference relation $\succ_{C'}$ defined using the following formula:*

$$u \succ_{C'} u' \equiv \forall t, t'[(t[X] = u \wedge t'[X] = u') \Rightarrow t \succ_C t'].$$

It is easy to see that if $\succ_C$ is a strict partial order, so is $\theta_x(\succ_C)$.

Theorem 4.4 *Given a relation schema R, a set of attributes X of R, and a preference formula C over R, if the following formulas are valid:*

$$\forall t_1, t_2, t_3[(t_1[X] = t_2[X] \wedge t_1[X] \neq t_3[X] \wedge t_1 \succ_C t_3) \Rightarrow t_2 \succ_C t_3],$$

$$\forall t_1, t_3, t_4[(t_3[X] = t_4[X] \wedge t_1[X] \neq t_3[X] \wedge t_1 \succ_C t_3) \Rightarrow t_1 \succ_C t_4],$$

then for all instances r of R:

$$\pi_X(\omega_C(r)) = \omega_{C'}(\pi_X(r)),$$

where $\succ_{C'} = \theta_x(\succ_C)$ is the restriction of $\succ_C$ to X. The converse holds under the assumption that $\succ_C$ is irreflexive.

Proof: Assume $u \in \pi_X(\omega_C(r))$. Then there exists a tuple $t \in \omega_C(r)$ such that $t[X] = u$. Assume $u \notin \omega_{C'}(\pi_X(r))$. Since $u \in \pi_X(r)$, there exists a tuple $u' \in \pi_X(r)$ such that $u' \succ_{C'} u$ and a tuple $t' \in r$ such that $t'[X] = u'$. Since $u' \succ_{C'} u$, it has to be the case that $t' \succ_C t$, which contradicts the fact that $t \in \omega_C(r)$.

For the opposite direction, assume that $u \in \omega_{C'}(\pi_X(r))$ and $u \notin \pi_X(\omega_C(r))$. Then for each tuple $t \in r$ such that $t[X] = u$, there is another tuple $t' \in r$ such that $t' \succ_c t$ and $t'[X] \neq t[X]$. By the assumption of the theorem, each tuple t' that dominates (in $\succ_C$) one tuple t such that $t[X] = u$, also dominates each such tuple. Also, any two tuples that agree on X dominate the same set of tuples. Therefore, if $u' = t'[X]$, then $u' \succ_{C'} u$, which contradicts the fact that $u \in \omega_{C'}(\pi_X(r))$.

To show the converse, assume that the first condition is violated, i.e., there are three tuples t_1, t_2 and t_3 such that $t_1[X] = t_2[X]$, $t_1[X] \neq t_3[X]$, $t_1 \succ_C t_3$ and $t_2 \nsucc_C t_3$. Let $r_0 = \{t_1, t_2, t_3\}$. Then $t_3 \notin \omega_C(r_0)$, so $\pi_X(\omega_C(r_0)) = \{t_1[X]\}$. Now $t_1[X] \nsucc_{C'} t_3[X]$ (because $t_2 \nsucc_C t_3$) and $t_1[X] \neq t_3[X]$. Thus

$$\omega_{C'}(\pi_X(r)) = \{t_1[X], t_3[X]\} \neq \{t_1[X]\} = \pi_X(\omega_C(r_0)).$$

The violation of the second condition also leads to a contradiction in a similar way. □

If the preference formula C in Theorem 4.4 is a pure comparison ipf in k-DNF then checking the validity of the assumption of this theorem can be done in PTIME. If C is a pure-comparison ipf, then C' can be presented in an equivalent, quantifier-free form.

Example 4.5 *Consider again the preference relation $\succ_{C_1}$ from Example 1.1:*

$$(i, v, p) \succ_{C_1} (i', v', p') \equiv i = i' \wedge p < p'$$

over the relation schema $Book(ISBN, Vendor, Price)$. Then the relation $C' = \theta_{ISBN, Price}(\succ_{C_1})$ is defined as

$$(i, p) \succ_{C'} (i', p') \equiv \forall t, t'[(t[X] = (i, p) \wedge t'[X] = (i', p')) \Rightarrow t \succ_{C_1} t'] \equiv i = i' \wedge p < p'.$$

This confirms the intuition that the projection does not affect this particular preference relation. It is easy to see that the condition of Theorem 4.4 is also satisfied, so winnow commutes with projection in this case.

4.3.4 Distributing winnow over Cartesian product

For winnow to distribute (in a modified form) with the Cartesian product, the preference formula needs to be in a special form. The form turns out to be the *Pareto composition*, well known in multi-attribute utility theory [13]. Preference queries involving Pareto composition are quite common: the skyline queries [4] without `DIFF` attributes are of this form.

Definition 4.2 *Given two relation schemas R_1 and R_2, a preference relation $\succ_{C_1}$ over R_1 and a preference relation $\succ_{C_2}$ over R_2, the Pareto composition $P(\succ_{C_1}, \succ_{C_2})$ of $\succ_{C_1}$ and $\succ_{C_2}$ is a preference relation $\succ_{C_0}$ over the Cartesian product $R_1 \times R_2$ defined as:*

$$(t_1, t_2) \succ_{C_0} (t'_1, t'_2) \equiv t_1 \succeq_{C_1} t'_1 \wedge t_2 \succeq_{C_2} t'_2 \wedge (t_1 \succ_{C_1} t'_1 \vee t_2 \succ_{C_2} t'_2),$$

where

$$x \succeq_C y \equiv x \succ_C y \vee x = y.$$

Clearly, if $\succ_{C_1}$ and $\succ_{C_2}$ are strict partial orders, so is $P(\succ_{C_1}, \succ_{C_2})$.

Theorem 4.5 *Given two relation schemas R_1 and R_2, a preference relation $\succ_{C_1}$ over R_1 and a preference relation $\succ_{C_2}$ over R_2, for any relations r_1 and r_2 which are instances of R_1 and R_2, resp., the following property holds:*

$$\omega_{C_0}(r_1 \times r_2) = \omega_{C_1}(r_1) \times \omega_{C_2}(r_2),$$

where $C_0 = P(\succ_{C_1}, \succ_{C_2})$.

Proof: Assume $(t_1, t_2) \in \omega_{C_0}(r_1 \times r_2)$ but $(t_1, t_2) \notin \omega_{C_1}(r_1) \times \omega_{C_2}(r_2)$. Then $t_1 \notin \omega_{C_1}(r_1)$ or $t_2 \notin \omega_{C_2}(r_2)$. Assume the first. Since $(t_1, t_2) \in r_1 \times t_2$ and $t_1 \in r_1$, there must be a tuple $t_1' \in r_1$ such that $t_1' \succ_{C_1} t_1$. Then the tuple $(t_1', t_2) \in r_1 \times r_2$ and $(t_1', t_2) \succ_{C_0} (t_1, t_2)$ which contradicts the fact that $(t_1, t_2) \in \omega_{C_0}(r_1 \times r_2)$. The second case is symmetric.

Assume now that $(t_1, t_2) \in \omega_{C_1}(r_1) \times \omega_{C_2}(r_2)$ and $(t_1, t_2) \notin \omega_{C_0}(r_1 \times r_2)$. Then there is a tuple $(t_1', t_2') \in r_1 \times r_2$ such that $(t_1', t_2') \succ_{C_0} (t_1, t_2)$. Consequently, $t_1' \succ_{C_1} t_1$ or $t_2' \succ_{C_2} t_2$. Both cases lead to a contradiction with the fact that $(t_1, t_2) \in \omega_{C_1}(r_1) \times \omega_{C_2}(r_2)$. $\square$

We show now that a slight variation of the Pareto composition, even though it appears to be more natural, fails to achieve the distributivity of winnow over product.

Example 4.6 *Define a different composition $\succ_{C_0'}$ of two preference relations $\succ_{C_1}$ and $\succ_{C_2}$ as follows:*

$$(t_1, t_2) \succ_{C_0'} (t_1', t_2') \equiv t_1 \succ_{C_1} t_1' \wedge t_2 \succ_{C_2} t_2'.$$

Consider the following preference relations:

$$x \succ_{C_1} y \equiv x \succ_{C_2} y \equiv x > y.$$

Then if $r_1 = \{1\}$ and $r_2 = \{1, 2\}$, then

$$\omega_{C_1}(r_1) \times \omega_{C_2}(r_2) = \{(1, 2)\} \neq \{(1, 1), (1, 2)\} = \omega_{C_0'}(r_1 \times r_2).$$

4.3.5 Distributing winnow over union and difference

It is possible to distribute winnow over union or difference only in the trivial case where the preference relation is an anti-chain. We call two relation schemas *compatible* if they have the same number of attributes and the corresponding attributes have the same domains.

Theorem 4.6 *Given two compatible relation schemas R and S and an irreflexive preference relation $\succ_C$ over R, we have for every relation r and s*

$$\omega_C(r \cup s) = \omega_C(r) \cup \omega_C(s)$$

and

$$\omega_C(r - s) = \omega_C(r) - \omega_C(s)$$

if and only if $\succ_C = \emptyset$.

Proof: Clearly, if $\succ_C = \emptyset$ and $\succ_C$ is irreflexive, then

$$\omega_C(r) \cup \omega_C(s) = r \cup s = \omega_C(r \cup s).$$

To show that this is a necessary condition, assume that $\succ_C \neq \emptyset$. Then there are two tuples t_1 and t_2 such that $t_1 \succ_C t_2$. Now

$$\omega_C(\{t_1, t_2\}) = \{t_1\} \neq \{t_1, t_2\} = \omega_C(\{t_1\}) \cup \omega_C(\{t_2\}).$$

The proof for difference is similar. □

5 Composition of preferences

Preference relations may be composed in many different ways. In general, we distinguish between *multi-dimensional* and *uni-dimensional* composition. In multi-dimensional composition, we have a number of preference relations defined over several database relation schemas, and we define a preference relation over the Cartesian product of those relations. An example is Pareto composition (Definition 4.2). Another example is lexicographic composition. In uni-dimensional composition, a number of preference relations over a single database schema are composed, producing another preference relation over the same schema. Examples include: Boolean and prioritized composition (discussed below).

Since in our framework preference relations are defined by first-order preference formulas, any first-order definable composition of preference relations leads again to first-order preference formulas, which in turn can be used as parameters of the winnow operator. The composition does not even have to be first-order definable, as long as it produces a (first-order) preference formula. We'll see an example of the latter later in section when we discuss transitive closure.

5.1 Boolean composition

Union, intersection and difference of preference relations are obviously captured by the Boolean operations on the corresponding preference formulas. For example, the following formula captures the preference $\succ_{C_0} = \succ_{C_1} \cap \succ_{C_2}$:

$$x \succ_{C_0} y \equiv x \succ_{C_1} y \wedge x \succ_{C_2} y.$$

Table 1 summarizes the preservation of properties of relations by the appropriate Boolean composition operator.

5.2 Preference hierarchies

It is often the case that preferences form hierarchies. For instance, I may have a general preference for red wine but in specific cases, e.g., when eating fish, this preference is overridden by the one for white wine. Also a preference for less expensive books (Example 1.1) can be overridden by a preference for certain vendors.

	Union	Intersection	Difference
Irreflexivity	Yes	Yes	Yes
Asymmetry	No	Yes	Yes
Transitivity	No	Yes	No

Table 1: Properties Preserved by Boolean Composition

Definition 5.1 *Consider two preference relations $\succ_{C_1}$ and $\succ_{C_2}$ defined over the same schema U. The* prioritized composition *$\succ_{C_{1,2}} = \succ_{C_1} \triangleright \succ_{C_2}$ of $\succ_{C_1}$ and $\succ_{C_2}$ is defined as:*

$$t_1 \succ_{C_{1,2}} t_2 \equiv t_1 \succ_{C_1} t_2 \vee (t_2 \nsucc_{C_1} t_1 \wedge t_1 \succ_{C_2} t_2).$$

The prioritized composition $\succ_{C_1} \triangleright \succ_{C_2}$ has the following intuitive reading: *prefer according to $\succ_{C_2}$ unless $\succ_{C_1}$ is applicable.*

Example 5.1 *Continuing Example 1.1, instead of the preference relation $\succ_{C_1}$ defined there as follows:*

$$(i, v, p) \succ_{C_1} (i', v', p') \equiv i = i' \wedge p < p',$$

we consider the relation $\succ_{C_0} \triangleright \succ_{C_1}$ where $\succ_{C_0}$ is defined by the following formula C_0:

$$(i, v, p) \succ_{C_0} (i', v', p') \equiv i = i' \wedge v = {}'\text{BooksForLess}' \wedge v' = {}'\text{LowestPrices}'.$$

Assume the preference relation $\succ_{C_{0,1}} = \succ_{C_0} \triangleright \succ_{C_1}$ (the definition of $\succ_{C_{0,1}}$ is easily obtained from the formulas C_0 and C_1 by substitution). Then $\omega_{C_{0,1}}(r_1)$ returns the following tuples

ISBN	Vendor	Price
0679726691	*BooksForLess*	*$14.75*
0062059041	*BooksForLess*	*$7.30*
0374164770	*LowestPrices*	*$21.88*

Note that now a more expensive copy of the first book is preferred, due to the preference for 'BooksForLess' over 'LowestPrices'. However, 'BooksForLess' does not offer the last book, and that's why the copy offered by 'LowestPrices' is preferred.

Theorem 5.1 *If $\succ_{C_1}$ and $\succ_{C_2}$ are preference relations, so is $\succ_{C_{1,2}}$. If $\succ_{C_1}$ and $\succ_{C_2}$ are both irreflexive or asymmetric, so is $\succ_{C_{1,2}}$.*

However, a relation defined as the prioritized composition of two transitive preference relations does not have to be transitive.

Example 5.2 *Consider the following preference relations:*

$$a \succ_{C_1} b, b \succ_{C_2} c.$$

Both $\succ_{C_1}$ and $\succ_{C_2}$ are trivially transitive. However, $\succ_{C_1} \triangleright \succ_{C_2}$ is not.

Theorem 5.2 *Prioritized composition is associative:*

$$(\succ_{C_1} \rhd \succ_{C_2}) \rhd \succ_{C_3} \equiv \succ_{C_1} \rhd (\succ_{C_2} \rhd \succ C_3)$$

and distributes over union:

$$\succ_{C_1} \rhd (\succ_{C_2} \cup \succ_{C_3}) \equiv (\succ_{C_1} \rhd \succ_{C_2}) \cup (\succ_{C_1} \rhd \succ_{C_3}).$$

Thanks to the associativity and distributivity of $\rhd$, the above construction can be generalized to an arbitrary finite partial *priority order* between preference relations. Such an order can be viewed as a graph in which the nodes consist of preference relations and the edges represent relative priorities (there would be an edge $(\succ_{C_1}, \succ_{C_2})$ in the situation described above). To encode this graph as a single preference relation, one would construct first the definitions corresponding to individual paths from roots to leaves, and then take a disjunction of all such definitions.

There are many other ways of combining preferences. For instance, the paper [3] defines an infinite family of uni-dimensional composition operators for preference relations on the basis of two basic operators. Since all the definitions are first-order, every preference relation defined in the framework of [3] can also be defined in ours. In [3], it is proved that the operators in the defined family exhaust all operators satisfying a number of intuitively plausible postulates. It turns out that the operator $\rhd$ defined above cannot be captured in the framework of [3], because it violates one of those postulates: it does not preserve transitivity.

5.3 Transitive closure

We address here the issue of *transitively closing* a preference relation. We have seen an example (Example 1.1) of a preference relation that is already transitive. However, there are cases when we expect the preference relation to be the *transitive closure* of another preference relation which is not transitive.

Example 5.3 *Consider the following relation:*

$$x \succ y \equiv x = a \wedge y = b \vee x = b \wedge y = c.$$

In this relation, a and c are not related though there are contexts in which this might be natural. (Assume I prefer to walk than to drive, and to drive than to ride a bus. Thus, I also prefer to walk than to ride a bus.)

In our framework, we can specify the preference relation $\succ_{C^*}$ to be the *transitive closure* of another preference relation $\succ$ defined using a first-order formula. This is similar to transitive closure queries in relational databases. However, there is an important difference. In databases, we are computing the transitive closure of a *finite* relation, while here we are transitively closing an infinite relation defined using a first-order formula.

Definition 5.2 *The* transitive closure *of a preference relation* $\succ_C$ *over a relation schema* R *is a preference relation* $\succ_{C*}$ *over* R *defined as:*

$$t_1 \succ_{C*} t_2 \text{ iff } t_1 \succ_C^n t_2 \text{ for some } n \geq 0,$$

where:

$$t_1 \succ_C^1 t_2 \equiv t_1 \succ_C t_2$$
$$t_1 \succ_C^{n+1} t_2 \equiv \exists t_3.\ t_1 \succ_C t_3 \wedge t_3 \succ_C^n t_2.$$

Clearly, in general Definition 5.2 leads to infinite formulas. However, as Theorem 5.3 shows, in many important cases the preference relation $\succ_{C*}$ will in fact be defined by a finite formula.

Theorem 5.3 *If a preference relation* $\succ_C$ *is defined using a pure comparison ipf, the transitive closure* $\succ_{C*}$ *of* $\succ_C$ *is also defined using a pure comparison ipf and that definition can be effectively obtained.*

Proof: The computation of the transitive closure can in this case be formulated as the evaluation of Datalog with order or gap-order (for integers) constraints. Suppose $\succ_C$ is defined as:

$$x \succ_C y \equiv \alpha_1(x, y) \vee \cdots \vee \alpha_n(x, y).$$

Then the Datalog program that computes the formula C^* defining $\succ_{C*}$ looks as follows:

$$T(x, y) \leftarrow \alpha_1(x, y).$$
$$\cdots$$
$$T(x, y) \leftarrow \alpha_n(x, y).$$
$$S(x, y) \leftarrow T(x, y).$$
$$S(x, y) \leftarrow T(x, z), S(z, y).$$

The evaluation of this program terminates [21, 26] and its result, collected in S, represents the desired formula.

$\square$

An analogous result holds if instead of arithmetic comparisons we consider equality constraints over an infinite domain [21].

Example 5.4 *Continuing Example 5.3, we obtain the following preference relation* $\succ_{C*}$ *by transitively closing* $\succ_C$:

$$x \succ_{C*} y \equiv x = a \wedge y = b \vee x = b \wedge y = c \vee x = a \wedge y = c.$$

Theorem 5.3 is not in conflict with the well-known non-first order definability of transitive closure on finite structures. In the latter case it is shown that there is no finite first-order formula expressing transitive closure for arbitrary (finite) binary relations. In Theorem 5.3 the relation to be closed, although possibly infinite, is fixed (since it is defined using the given ipf). In particular, given an encoding of a fixed finite binary relation using an ipf, the transitive closure of this relation is defined using another ipf.

The transitive closure of a irreflexive (resp. asymmetric) preference relation may fail to be irreflexive (resp. asymmetric).

6   Applications and extensions

We show here how to use winnow to express special classes of preference queries: skylines and queries involving scoring functions, and how to use winnow together with other operators of the relational algebra to express more complex decision problems involving preferences. We consider the following: integrity constraints, extrinsic preferences, and aggregation.

6.1   Special classes of preference queries

6.1.1   Skylines

Skyline queries [4] find all the tuples in a relation that *are not dominated by any other tuples in the same relation in all dimensions.* This is exactly the notion of Pareto composition (Definition 4.2) in an arbitrary number of dimensions.

Figure 4 shows an example of a two-dimensional skyline where the dominance relationship is $>$. The skyline elements are marked with thick black dots.
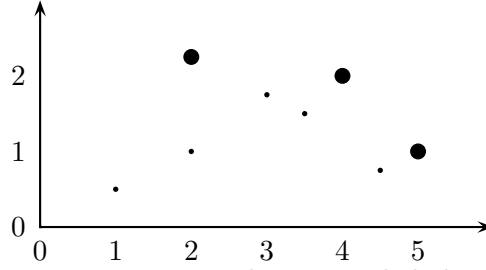

Figure 4: Two-dimensional skyline

[4] propose to write skyline queries using the following extension to SQL:

```
SELECT ... FROM ... WHERE ...
GROUP BY ... HAVING ...
SKYLINE OF A1 [MIN | MAX | DIFF]
         ...
         An [MIN | MAX | DIFF]
```

The values of a `MIN` attribute are minimized, those of a `MAX` attribute maximized. A `DIFF` attribute indicates that tuples with *different* values of that attribute are incomparable. The `SKYLINE` clause is applicable after all other SQL clauses.

Clearly, skylines can be expressed using winnow. The winnow is applied to an SQL view that expresses the non-skyline constructs in a skyline query. The preference formula is easily obtained from the `SKYLINE` clause. For example:

`SKYLINE OF A DIFF, B MAX, C MIN`

in a relation R is equivalent to $\omega_C(R)$ where

$$(x, y, z) \succ_C (x', y', z') \equiv x = x' \land y \geq y' \land z \leq z' \land (y > y' \lor z < z').$$

22

Finally, we note that $\omega_{C_1}(Book)$ from Example 2.1 is also a skyline query in which the skyline clause looks as follows:

```
SKYLINE OF ISBN DIFF, PRICE MIN.
```

6.1.2 Queries involving scoring functions

Sometimes a relation schema R comes with a scoring function that associates a value $f(t)$ with every tuple t in a possible instance of R. Now finding the tuples that maximize a scoring function f can be done by computing $\omega_{C_f}(r)$ for the given instance r of R, where

$$t \succ_{C_f} t' \equiv f(t) > f(t').$$

This approach can be generalized to compute not only the top scoring tuples but also those whose score differs from the top score by at most a given value or a given percentage. For example, the tuples that differ from the top score by at most d are computed by $\omega_{C_{f-d}}(r)$, where

$$t \succ_{C_{f-d}} t' \equiv f(t) - d > f(t').$$

Queries that return the tuples with *top-N* scores [7] can also be captured using winnow together with SQL, using the approach described later in this section. Essentially, for each tuple t we will determine using SQL the number $n(t)$ of tuples with higher scores than t and use the expression $N - n(t)$, where is the number of tuples in the relation, to define a new scoring function. This function is then used to define a preference relation as in the preceding paragraph. It appears, however, that in terms of the efficiency of query evaluation this approach will be inferior to the approach in which top-N queries are supported directly by the query engine.

Formally, we say that a real-valued function f over a schema R *represents* a preference relation $\succ_C$ over R iff

$$\forall t_1, t_2 \ [t_1 \succ_C t_2 \text{ iff } f(t_1) > f(t_2).$$

As pointed out earlier, not every preference relation which is a strict partial order can be expressed using a scoring function. A necessary condition is that the relation be a *weak order* [13]. We can ask for the motivation behind this notion of representation. It is easy to show that

Theorem 6.1 *A real-valued function f represents a preference relation $\succ_C$ iff for every finite instance r of R, the set $\omega_C(r)$ is equal to the set of tuples of r assuming the maximum value of u.*

Thus, if a scoring function does not represent a preference relation, that fact can be detected by winnow evaluated over some instance.

There are other, weaker forms of representation of preference relations by scoring functions. For instance, if we only require that

$$\forall t_1, t_2 \ [t_1 \succ_C t_2 \Rightarrow f(t_1) > f(t_2),$$

then for every strict partial order there is a scoring function representing it. However, in this case we can only guarantee that the set of tuples in a given instance r that maximize f is a *subset* of $\omega_C(r)$.

6.2 Integrity constraints

There are cases when we wish to impose a constraint on the result of the winnow operator. In Example 1.1, we may say that we are interested only in the books under \$15. In Example 2.2, we may restrict our attention only to the meat or fish dishes (note that currently the dishes that are not meat or fish do not have a preferred kind of wine). In the same example, we may ask for a specific number of meal recommendations.

In general, we need to distinguish between *local* and *global* constraints. A local constraint imposes a condition on the components of a single tuple, for instance `Book.Price<$15`. A global constraint imposes a condition on a set of tuples. The first two examples above are local constraints; the third is global. To satisfy a global constraint on the result of the winnow operator, one would have to construct a maximal subset of this answer that satisfies the constraint. Since in general there may be more than one such subset, the required construction cannot be described using a single relational algebra query. On the other hand, local constraints are easily handled, since they can be expressed using selection. In general, it matters whether the selection is applied before or after the winnow operator. Theorem 4.3 identifies sufficient and necessary conditions for winnow and selection to commute.

Example 6.1 *Consider the situation where we have a specific preference ordering for cars, e.g., prefer BMW to Chevrolet, but also have a limited budget (captured by a selection condition). Then clearly, selecting the most desirable affordable car will not give the same result as selecting the most desirable cars if they are affordable.*

A *veto* expresses a prohibition on the presence of a specific set of values in the elements of the answer to a preference query and thus can be viewed as a local constraint. To veto a specific tuple $w = (a_1, \ldots, a_n)$ in a relation S (which can be defined by a preference query) of arity n, we write the selection:

$$\sigma_{A_1 \neq a_1 \vee \cdots \vee A_n \neq a_n}(S).$$

6.3 Intrinsic vs. extrinsic preferences

So far we have talked only about *intrinsic* preference formulas. Such formulas establish the preference relation between two tuples purely on the basis of the values occurring in those tuples. *Extrinsic* preference formulas may refer not only to built-in predicates but also to

other constructs, e.g., database relations. In general, extrinsic preferences can use a variety of criteria: properties of the relations from which the tuples were selected, properties of other relations, or comparisons of aggregate values, and do not even have to be defined using first-order formulas.

It is possible to express some extrinsic preferences using the winnow operator together with other relational algebra operators using the following multi-step strategy:

1. using a relational query, combine all the information relevant for the preference in a single relation,

2. apply the appropriate winnow operator to this relation,

3. project out the extra columns introduced in the first step.

The following example demonstrates the above strategy, as well as the use of aggregation for the formulation of preferences.

Example 6.2 *Consider again the relation $Book(ISBN, Vendor, Price)$. Suppose for each book a preferred vendor (there may be more than one) is a vendor that sells the* maximum total *number of books. Clearly, this is an extrinsic preference since it cannot be established solely by comparing pairs of tuples from this relation. However, we can provide the required aggregate values and connect them with individual books through new, separate views:*

```
CREATE VIEW BookNum(Vendor,Num) AS
  SELECT B1.Vendor, COUNT(DISTINCT B1.ISBN)
  FROM Book B1
  GROUP BY B1.Vendor;

CREATE VIEW ExtBook(ISBN,Vendor,Num) AS
  SELECT B1.ISBN, B1.Vendor, BN.Num
  FROM Book B1, BookNum BN
  WHERE B1.Vendor=BN.Vendor;
```

Now the extrinsic preference is captured by the query

$$\pi_{ISBN,Vendor}(\omega_{C_5}(ExtBook))$$

where the preference formula C_5 is defined as follows:

$$(i, v, n) \succ_5 (i', v', n') \equiv i = i' \wedge n > n'.$$

Example 6.3 *To see another example of extrinsic preference, consider the situation in which we prefer any tuple from a relation R over any tuple from a relation S which is disjoint from R. Notice that this is truly an extrinsic preference, since it is based on where the tuples come from and not on their values. It can be handled in our approach by* tagging

the tuples with the appropriate relation names (easily done in relational algebra or SQL) and then defining the preference relation using the tags. If there is a tuple which belongs both to R and S, then the above preference relation will fail to be irreflexive and the simulation using intrinsic preferences will not work. Note also that an approach similar to tagging was used in Example 2.2 (wine and dish types play the role of tags).

Example 6.4 *Suppose user preferences are stored in a database relation Pref(A,B). Then once can define an extrinsic preference relation:*

$$x \succ_{Pref} y \equiv Pref(x, y).$$

Such a preference relation cannot be defined using a pure comparison ipf, because the transitive closure of a preference relation defined using an ipf is finite (Theorem 5.3), while that of $\succ_{Pref}$ is infinite.

7 Iterated preferences and ranking

We show here that the framework presented so far can be further developed to capture other preference-related concepts like ranking. We also present a variant of winnow suitable to preference relations that are not partial orders.

7.1 Ranking

A natural notion of *ranking* is implicit in our approach. A ranking is defined using *iterated preference.*

Definition 7.1 *Given a preference relation $\succ$ defined by a pf C, the n-th iteration of the winnow operator ω_C in r is defined as:*

$$\omega_C^1(r) = \omega_C(r)$$
$$\omega_C^{n+1}(r) = \omega_C(r - \bigcup_{1 \leq i \leq n} \omega_C^i(r))$$

For example, the query $\omega_C^2(r)$ computes the set of "second-best" tuples.

Example 7.1 *Continuing Example 1.1, the query $\omega_{C_1}^2(r_1)$ returns*

ISBN	Vendor	Price
0679726691	*BooksForLess*	*$14.75*

and the query $\omega_{C_1}^3(r_1)$ returns

ISBN	Vendor	Price
0679726691	*QualityBooks*	*$18.80*

Therefore, by iterating the winnow operator one can *rank* the tuples in a given relation instance.

Theorem 7.1 *If a preference relation $\succ_C$ over a relation schema R is a strict partial order, then for every finite instance r of R and every tuple $t \in r$, there exists an i, $i \geq 1$, such that $t \in \omega_C^i(r)$.*

Proof: Assume there is a tuple $t_0 \in \omega_C(r)$ such that for all $i \geq 1$, $t_0 \notin \omega_C^i(r)$. Select the least i_0 such that $\forall i \geq i_o$, $\omega_C^i(r) = \emptyset$ (such an i_0 always exists due to the finiteness of r). Clearly, $t_0 \notin \omega_C^{i_0}(r)$, thus $t \in r - \bigcup_{1 \leq i \leq i_0 - 1} \omega_C^i(r)$. Then there must be a tuple t_1 such that $t_1 \succ_C t_0$ and $t_1 \in r - \bigcup_{1 \leq i \leq i_0 - 1} \omega_C^i(r)$ (otherwise $t_0 \in \omega_C^{i_0}(r)$). Since $\succ_C$ is a strict partial order, there has to be an infinite increasing chain in r, a contradiction with the finiteness of r. $\square$

We define now the ranking operator $\eta_C(R)$.

Definition 7.2 *If R is a relation schema and C a preference formula defining a preference relation $\succ_C$ over R, then the* ranking operator *is written as $\eta_C(R)$, and for every instance r of R:*

$$\eta_C(r) = \{(t, i) \mid t \in \omega_C^i(r)\}.$$

One can now study the algebraic properties of the ranking operator, that parallel those that we established for winnow in Section 4. We list here only one property which is the most important one from a practical point of view: commutativity of selection with ranking. In this context, ranking enjoys identical properties to winnow.

Theorem 7.2 *Given a relation schema R, a selection condition C_1 over R and a preference formula C_2 over R, if the formula*

$$\forall t_1, t_2 [(C_1(t_2) \wedge C_2(t_1, t_2)) \Rightarrow C_1(t_1)]$$

is valid, then for all instances r of R:

$$\sigma_{C_1}(\eta_{C_2}(r)) = \eta_{C_2}(\sigma_{C_1}(r)).$$

The converse holds under the assumption that $\succ_{C_2}$ is irreflexive.

Proof: The proof is by induction on tuple rank. The base case follows from Theorem 4.3 and the inductive case from the observation that

$$\sigma_{C_1}(\omega_{C_2}^{n+1}(r)) = \sigma_{C_1}(\omega_{C_2}(r - \bigcup_{1 \leq i \leq n} \omega_C^i(r))) = \omega_{C_2}(\sigma_{C_1}(r - \bigcup_{1 \leq i \leq n} \omega_C^i(r)))$$

which is equal to

$$\omega_{C_2}(\sigma_{C_1}(r) - \sigma_{C_1}(\bigcup_{1 \leq i \leq n} \omega_C^i(r))) = \omega_{C_2}(\sigma_{C_1}(r) - \bigcup_{1 \leq i \leq n} \sigma_{C_1}(\omega_C^i(r))).$$

under the assumptions of the theorem. $\square$

7.2 Weak winnow

If a preference relation is not a strict partial order, then Theorems 3.2 and 7.1 may fail to hold. A number of tuples can block each other from appearing in the result of any iteration of the winnow operator. However, even in this case there may be a weaker form of ranking available.

Example 7.2 *Consider Examples 1.1 and 5.1. If the preference formula C' is defined as $C_0 \vee C_1$, then the first two tuples of the instance r_1 block each other from appearing in the result of $\omega_{C'}(r_1)$, since according to C_0 the first tuple is preferred to the second but just the opposite is true according to C_1. Intuitively, both those tuples should be preferred to (and ranked higher) than the third tuple. But since neither the first not the second tuple is a member of $\omega_{C'}(r_1)$, none of the first three tuples can be ranked.*

To deal with preference relations that are not strict partial orders, we define a new, weaker form of the winnow operator. We relax the asymmetry and irreflexivity requirements but preserve transitivity.

To define this operator, we notice that as long as the preference relation $\succ_C$ is transitive, we can use it to define another preference relation $\succ_{C_>}$ which is a strict partial order:

$$x \succ_{C_>} y \equiv x \succ_C y \wedge y \not\succ_C x.$$

Definition 7.3 *If R is a relation schema and $\succ_C$ a transitive preference relation over R, then the weak winnow operator is written as $\psi_C(R)$ and for every instance r of R, $\psi_C(r) = \omega_{C_>}(r)$.*

It follows from the definition that

$$\psi_C(r) = \{t \in r \mid \forall t' \in r.\ t \succ_C t' \vee t' \not\succ_C t\}.$$

Thus the weak winnow operator returns all the tuples that are dominated only by the tuples that they dominate themselves.

Example 7.3 *Considering Example 7.2, we see that the query $\psi_{C'}(r_1)$ returns now*

ISBN	Vendor	Price
0679726691	*BooksForLess*	*$14.75*
0679726691	*LowestPrices*	*$13.50*
0062059041	*BooksForLess*	*$7.30*
0374164770	*LowestPrices*	*$21.88*

Below we formulate a few properties of the weak winnow operator. Using Theorems 3.3 and 3.2 (notice that $C_> \Rightarrow C$), we immediately obtain the following theorem.

Theorem 7.3 *If R is a relation schema and $\succ_C$ a transitive preference relation over R, then:*

28

- *for every instance r of R, $\omega_C(r) \subseteq \psi_C(r)$.*

- *for every finite, nonempty relation instance r of R, $\psi_C(r)$ is nonempty.*

One can define the iteration of the weak winnow operator similarly to that of the winnow operator (Definition 7.1).

Theorem 7.4 *If a preference relation $\succ_C$ over a relation schema R is transitive, then for every finite instance r of R and for every tuple $t \in r$, there exists an i, $i \geq 1$, such that $t \in \psi_C^i(r)$.*

8 Related work

8.1 Preference queries

[24] originated the study of *preference queries*. It proposed an extension of the relational calculus in which preferences for tuples satisfying given logical conditions can be expressed. For instance, one could say: *Among the tuples of R satisfying Q, I prefer those satisfying P_1; among the latter I prefer those satisfying P_2.* Such a specification was to mean the following: Pick the tuples satisfying $Q \wedge P_1 \wedge P_2$; if the result is empty, pick the tuples satisfying $Q \wedge P_1 \wedge \neg P_2$; if the result is empty, pick the remaining tuples of R satisfying Q. This can be simulated in our framework as the relational algebra expression $\omega_{C^*}(\sigma_Q(R))$ where C^* is an ipf defined in the following way:

1. obtain the formula C defining a preference relation $\succ$

$$t_1 \succ t_2 \equiv P_1(t_1) \wedge P_2(t_1) \wedge P_1(t_2) \wedge \neg P_2(t_2) \vee P_1(t_1) \wedge \neg P_2(t_1) \wedge \neg P_1(t_2),$$

2. transform C into DNF to obtain an ipf C', and

3. close the result transitively to obtain an ipf C^* defining a transitive preference relation $\succ^*$ (as described in Section 5).

Other kinds of logical conditions from [24] can be similarly expressed in our framework. Maximum/minimum value preferences (as in Example 1.1) are handled in [24] through the explicit use of aggregate functions. The use of such functions is implicit in the definition of our winnow operator.

Unfortunately, [24] does not contain a formal definition of the proposed language, so a complete comparison with our approach is not possible. It should be noted, however, that the framework of [24] seems unable to capture very simple conditional preferences like the ones in Examples 2.2 and 5.3. Also, it can only handle strict partial orders of bounded depth (except in the case where aggregate functions can be used, as in Example 1.1). Hierarchical or iterated preferences are not considered.

[16] was one of the sources of inspiration for the present paper. It defines *Preference Datalog*: a combination of Datalog and clausally-defined preference relations. Preference

Datalog captures, among others, the class of preference queries discussed in [24]. The declarative semantics of Preference Datalog is based on the notion of *preferential consequence*, introduced earlier by the authors in [15]. This semantics requires preferences to be reflexive and transitive. Also, the operational semantics of Preference Datalog uses specialized versions of the standard logic program evaluation methods: bottom-up [16] or top-down [15]. In the context of database queries, the approach proposed in the present paper achieves similar goals to that of [15] and [16], remaining, however, entirely within the relational data model and classical first-order logic. Finally, [15, 16] do not address some of the issues we deal with in the present paper like transitive closure of preferences, prioritized composition or iterated preferences (a similar concept to the last one is presented under the name of "relaxation"). More importantly, the issues of embedding the framework into a real relational query language and optimizing preference queries are not addressed.

[22, 23] propose an (independently developed) framework similar to the one presented in this paper and in [8]. A formal language for formulating preference relations is described. The language has a number of base preference constructors and their combinators (Pareto and lexicographic composition, intersection, disjoint union and others). Clearly, all of those can be captured in our framework. On the other hand, [22, 23] do not consider the possibility of having arbitrary operation and predicate signatures in preference formulas, and do not identify any specific classes of preference formulas. Neither do they consider extrinsic preferences, complex preferences involving aggregation, or ranking. However, the embedding into relational query languages they use is identical to ours (it is called Best Match Only, instead of winnow). While some possible rewritings for preference queries are presented in [22], abstract properties of winnow that we described in Section 4 are not identified. Finally, [23] describes an implementation of the framework of [22] using a language called Preference SQL, which is translated to SQL, and several deployed applications.

[4] introduces the *skyline* operator and describes several evaluation methods for this operator. As shown in Section 6, skyline is a special case of winnow. It is restricted to use a pure comparison ipf which is a conjunction of pairwise comparisons of corresponding tuple components. So in particular Example 2.2 does not fit in that framework. Some examples of possible rewritings for skyline queries are given but no general rewriting rules are formulated.

[2] uses quantitative preferences in queries and focuses on the issues arising in combining such preferences. [19] explores in this context the problems of efficient query processing. Since the preferences in this approach are based on comparing the scores of individual tuples under given scoring functions, they have to be intrinsic. However, the simulation of extrinsic preferences using intrinsic ones (Section 6) is not readily available in this approach because the scoring functions are not integrated with the query language. So, for instance, Example 6.2 cannot be handled. In fact, even for preference relations that satisfy the property of transitivity of the corresponding indifference relation, it is not clear whether the scoring function capturing the preference relation can be defined intrinsically (i.e., the function value be determined solely by the the values of the tuple components). The general construction of a scoring function on the basis of a preference relation [12, 13] does not provide such a

definition. So the exact expressive power of the quantitative approach to preference queries remains unclear.

8.2 Preferences in logic and artificial intelligence

The papers on *preference logics* [29, 25, 18] address the issue of capturing the common-sense meaning of preference through appropriate axiomatizations. Preferences are defined on formulas, not tuples, and with the exception of [25, 10] limited to the propositional case. [25] proposes a modal logic of preference, and [10] studies preferences in the context of relation algebras. The application of the results obtained in this area to database queries is unclear.

The papers on *preference reasoning* [30, 28, 5] attempt to develop practical mechanisms for making inferences about preferences and solving decision or configuration problems similar to the one described in Example 2.2. A central notion there is that of *ceteris paribus* preference: preferring one outcome to another, all else being equal. Typically, the problems addressed in this work are propositional (or finite-domain). Such problems can be encoded in the relational data model and the inferences obtained by evaluating preference queries. A detailed study of such an approach remains still to be done. We note that the use of a full-fledged query language in this context makes it possible to formulate considerably more complex decision and configuration problems than before.

The work on *prioritized logic programming and nonmonotonic reasoning* [6, 11, 27] has potential applications to databases. However, like [16] it relies on specialized evaluation mechanisms, and the preferences considered are typically limited to rule priorities.

9 Conclusions and future work

We have presented a framework for specifying preferences using logical formulas and its embedding into relational algebra. As the result, preference queries and complex decision problems involving preferences can be formulated in a simple and clean way.

Clearly, our framework is limited to applications that can be entirely modeled within the relational model of data. Here are several examples that do not quite fit in this paradigm:

- preferences defined between *sets* of elements;

- *heterogenous* preferences between tuples of different arity or type (how to say I prefer a meal without a wine to a meal with one in Example 2.2?);

- preferences requiring nondeterministic choice. We believe this is properly handled using a nondeterministic choice [14] or witness [1] operator.

In addition to addressing the above limitations, future work directions include:

- evaluation and optimization of preference queries, including cost-based optimization;

- extrinsic preferences;

31

- defeasible and default preferences;

- preference elicitation.

Acknowledgments

References

[1] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[2] R. Agrawal and E.L. Wimmers. A Framework for Expressing and Combining Preferences. In *ACM SIGMOD International Conference on Management of Data*, pages 297–306, 2000.

[3] H. Andreka, M.D. Ryan, and P.-Y. Schobbens. Operators and Laws for Combining Preference Relations. *Journal of Logic and Computation*, 12(1):13–53, 2002.

[4] S. Börzsönyi, D. Kossmann, and K. Stocker. The Skyline Operator. In *IEEE International Conference on Data Engineering*, pages 421–430, 2001.

[5] C. Boutilier, R. I. Brafman, H. H. Hoos, and D. Poole. Reasoning with Conditional Ceteris Paribus Preference Statements. In *Symposium on Uncertainty in Artificial Intelligence*, 1999.

[6] G. Brewka and T. Eiter. Preferred Answer Sets for Extended Logic Programs. *Artificial Intelligence*, 109(1-2):297–356, 1999.

[7] M.J. Carey and D. Kossmann. On Saying Enough Already! in SQL. In *ACM SIGMOD International Conference on Management of Data*, pages 219–230, 1997.

[8] J. Chomicki. Querying with Intrinsic Preferences. In *International Conference on Extending Database Technology*, pages 34–51, 2002.

[9] J. Chomicki, P. Godfrey, J. Gryz, and D. Liang. On Skyline Computation. Submitted for publication.

[10] M. Cristani. Many-sorted Preference Relations. In *International Conference on Principles of Knowledge Representation and Reasoning*, 2002.

[11] J. P. Delgrande, T. Schaub, and H. Tompits. Logic Programs with Compiled Preferences. In *European Conference on Artificial Intelligence*, 2000.

[12] P. Fishburn. Preference Structures and their Numerical Representations. *Theoretical Computer Science*, 217:359–383, 1999.

[13] P.C. Fishburn. *Utility Theory for Decision Making*. Wiley & Sons, 1970.

[14] F. Giannotti, S. Greco, D. Sacca, and C. Zaniolo. Programming with Non-determinism in Deductive Databases. *Annals of Mathematics and Artificial Intelligence*, 19(3-4), 1997.

[15] K. Govindarajan, B. Jayaraman, and S. Mantha. Preference Logic Programming. In *International Conference on Logic Programming*, pages 731–745, 1995.

[16] K. Govindarajan, B. Jayaraman, and S. Mantha. Preference Queries in Deductive Databases. *New Generation Computing*, pages 57–86, 2001.

[17] S. Guo, W. Sun, and M.A. Weiss. Solving Satisfiability and Implication Problems in Database Systems. *ACM Transactions on Database Systems*, 21(2):270–293, 1996.

[18] S. O. Hansson. Preference Logic. In D. Gabbay, editor, *Handbook of Philosophical Logic*, volume 8. 2001.

[19] V. Hristidis, N. Koudas, and Y. Papakonstantinou. PREFER: A System for the Efficient Execution of Multiparametric Ranked Queries. In *ACM SIGMOD International Conference on Management of Data*, pages 259–270, 2001.

[20] R.G. Hughes. Rationality and Intransitive Preferences. *Analysis*, 40:132–134, 1980.

[21] P. C. Kanellakis, G. M. Kuper, and P. Z. Revesz. Constraint Query Languages. *Journal of Computer and System Sciences*, 51(1):26–52, August 1995.

[22] W. Kiessling. Foundations of Preferences in Database Systems. In *International Conference on Very Large Data Bases*, 2002.

[23] W. Kiessling and G. Koestler. Preference SQL - Design, Implementation, Experience. In *International Conference on Very Large Data Bases*, 2002.

[24] M. Lacroix and P. Lavency. Preferences: Putting More Knowledge Into Queries. In *International Conference on Very Large Data Bases*, pages 217–225, 1987.

[25] S. M. Mantha. *First-Order Preference Theories and their Applications*. PhD thesis, University of Utah, 1991.

[26] P. Z. Revesz. A Closed-Form Evaluation for Datalog Queries with Integer (Gap)-Order Constraints. *Theoretical Computer Science*, 116:117–149, 1993.

[27] C. Sakama and K. Inoue. Prioritized Logic Programming and its Application to Commonsense Reasoning. *Artificial Intelligence*, 123:185–222, 2000.

[28] S-W. Tan and J. Pearl. Specification and Evaluation of Preferences under Uncertainty. In *International Conference on Principles of Knowledge Representation and Reasoning*, 1994.

[29] G. H. von Wright. *The Logic of Preference*. Edinburgh University Press, 1963.

[30] M. P. Wellman and J. Doyle. Preferential Semantics for Goals. In *National Conference on Artificial Intelligence*, pages 698–703, 1991.