Preference Formulas in Relational Queries

JAN CHOMICKI
University at Buffalo, Buffalo, New York

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, for example, 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 interactions 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.

Categories and Subject Descriptors: F.4.1 [**Mathematical Logic and Formal Languages**]: Mathematical Logic; H.2.3 [**Database Management**]: Languages—*query Languages*; H.2.4 [**Database Management**]: Systems—*query processing*

General Terms: Languages, Theory

Additional Key Words and Phrases: Preference queries, preferences, query optimization, relational algebra

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.

The support of National Science Foundation under grant IIS-0307434 is gratefully acknowledged.
Author's address: Department of Computer Science and Engineering, 201 Bell Hall, University at Buffalo, State University of New York, Buffalo, NY 14260-2000; email: chomicki@cse.buffalo.edu.

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

Fig. 1. The Book relation.

The research literature on preferences is extensive. It encompasses preference logics [von Wright 1963; Mantha 1991; Hansson 2001], preference reasoning [Wellman and Doyle 1991; Tan and Pearl 1994; Boutilier et al. 1999], prioritized nonmonotonic reasoning and logic programming [Brewka and Eiter 1999; Delgrande et al. 2000; Sakama and Inoue 2000], and decision theory [Fishburn 1999; Fishburn 1970] (the list is by no means exhaustive). The research on preferences in the context of database queries goes back a long time [Lacroix and Lavency 1987; Kießling and Güntzer 1994; Köstler et al. 1995]. Nevertheless, only recently this area has attracted broader interest in the database research community [Agrawal and Wimmers 2000; Börzsönyi et al. 2001; Govindarajan et al. 2001; Hristidis et al. 2001; Chomicki 2002; Kießling 2002; Kießling and Hafenrichter 2002; Kießling and Köstler 2002]. Two different approaches have been pursued: qualitative and quantitative. In the *qualitative* approach [Lacroix and Lavency 1987; Kießling and Güntzer 1994; Köstler et al. 1995; Börzsönyi et al. 2001; Govindarajan et al. 2001; Chomicki 2002; Kießling 2002; Kießling and Hafenrichter 2002; Kießling and Köstler 2002], 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 article. Consider the relation *Book*(*ISBN*, *Vendor*, *Price*) and the following preference relation $\succ_{C_1}$ 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 instance r_1 of *Book* in Figure 1.
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 [Agrawal and Wimmers 2000; Hristidis et al. 2001], 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. However, 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 [Fishburn 1999, 1970]. On the other hand, preferences expressed using scoring functions can also be expressed directly, as long as the functions are explicitly defined. Qualitative approaches often offer various ways to express such preferences, as described, for example, in [Kießling 2002] and Section 7 of this article.

In the present article, 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 relational DBMS technology enhanced with a comprehensive support for preference queries. It will become possible to formulate as queries much more sophisticated search and configuration problems involving quantifiers, grouping, or aggregation. There are already promising applications of preference queries in the area of *personalized* search engines and shopping agents [Kießling and Köstler 2002]. Further potential applications include *logical user profiles* in the form of *materialized preference views*. Such profiles could help to partially automate (or guide) the customer's decision process and also provide useful customer information to vendors. Other potential applications involve “soft” catalogs [Stolze 2000] and knowledge-based recommendation systems [Burke 1999].

The framework presented in this article 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, 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 article, 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, *absolute* vs. *relative* preferences, *nested* and *hierarchical* preferences, *groupwise* preferences, *indifference*, preference *composition*, *iterated* preferences and *ranking*, and *integrity constraints* and *vetoes*.

The main contributions of this article 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 embedding makes possible the formulation of complex preference queries, for example, 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. The notion of winnow, first proposed in Chomicki [2002], was subsequently used in Torlone and Ciaccia [2002] under the name of the *Best* operator. A similar concept was independently proposed in Kießling [2002] and, in a more restricted form, in Börzsönyi et al. [2001].
(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 particular, the laws are applicable to the optimization of skyline queries [Börzsönyi et al. 2001], queries with scoring functions [Agrawal and Wimmers 2000; Hristidis et al. 2001], and queries with preference constructors [Kießling 2002; Kießling and Köstler 2002].

The restriction to the relational model of data is not, in our opinion, a serious limitation of the presented results. Although the data of interest may often be nonrelational, preference is a *relational* notion par excellence. For instance, the information about a book may be stored as an XML document. However, document structure is rarely used to compare one document to another. Rather, the relevant *relational* information, for example, price or year of publication, is extracted from the documents to yield fixed-arity tuples of attribute values. (Even Preference XPath, an extension of XPath with preference constructors [Kießling et al. 2001], seems to largely conform to the relational paradigm.)

In Section 2, we define the basic concepts of preference relation, preference formula, and winnow. 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, we explore the composition of preferences. (This section can be skipped if the reader is primarily interested in preference queries.) In Section 5, we describe some basic properties of winnow and outline—for completeness—a number of evaluation algorithms that were proposed elsewhere. In Section 6, which contains the main technical contributions of the article, we present the algebraic properties of the winnow operator. In Section 7, 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 8, we introduce the ranking operator whose semantics is defined using the iteration of winnow. In Section 9, we define a relaxed version of winnow that is helpful for preference relations that are not strict partial orders. We discuss related work in Section 10 and conclude with a brief discussion of further work in Section 11. All the nontrivial proofs are given.

2. BASIC NOTIONS

We are working in the context of the relational model of data. For concreteness, we consider two infinite domains: $\mathcal{D}$ (uninterpreted constants) and $\mathcal{Q}$ (rational numbers). Other domains could be considered as well without influencing most of the results of the article. 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 article, 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 $\mathcal{D}$ or $\mathcal{Q}$) 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 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*: $\forall x, y.\ x \succ y \vee y \succ x \vee x = y$.

Clearly, those properties are not independent. Asymmetry implies irreflexivity. Irreflexivity and transitivity imply asymmetry. However, it is often convenient to talk about them separately.

The relation $\succ$ is:

—a *strict partial order* if it is irreflexive and transitive (thus also asymmetric);
—a *weak order* if it is a negatively transitive strict partial order;
—a *total order* if it is a connected 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 *strict partial order*. Some results that we formulate in this article require that at least some of the properties of strict partial order hold. This will be always explicitly stated. The restriction to weak orders—which is rather strong—is necessary if preference relations are to be represented using scoring functions. This issue is discussed in more depth in Section 7.

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.

Because we consider two specific domains, $\mathcal{D}$ and $\mathcal{Q}$, we will have two kinds of variables, $\mathcal{D}$-variables and $\mathcal{Q}$-variables, and two kinds of atomic formulas:

—*equality constraints*: $x = y$, $x \neq y$, $x = c$, or $x \neq c$, where x and y are $\mathcal{D}$-variables, and c is an uninterpreted constant;

—*rational-order constraints*: $x\theta y$ or $x\theta c$, where $\theta \in \{=, \neq, <, >, \leq, \geq\}$, x and y are $\mathcal{Q}$-variables, and c is a rational number.

Without loss of generality, we will assume that ipfs are in DNF (Disjunctive Normal Form) and quantifier-free (the theories involving the above domains admit quantifier elimination). We also assume that atomic formulas are closed under negation (also satisfied by the above theories). An ipf whose all atomic formulas are equality (respectively, rational-order) constraints will be called an *equality* (respectively *rational-order*) ipf.

In this article, we mostly restrict ourselves to ipfs and preference relations defined by such formulas. The main reason is that ipfs are a special case of general constraints [Kuper et al. 2000], and 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 4.10), 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 2.4. Consider the relation *Book*(*ISBN*, *Vendor*, *Price*) (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* (Figure 1), 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 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 2.5. 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. Transitivity is obtained trivially because the chains of $\succ_{C_2}$ are of length at most 2. 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, for example, 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:

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

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

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

3. PREFERENCE AND INDIFFERENCE

3.1 Properties of 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 7 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 [Hughes 1980; Mantha 1991; Fishburn 1999; Hansson 2001]. 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. Composing preference relations may also lead to violations of asymmetry or transitivity (see Section 4). However, transitivity is essential for the correctness of some of the algorithms that compute winnow (Section 5).

Due to the restricted syntax of ipfs, one can effectively check whether a preference relation defined by an ipf is a strict partial order. To provide a more fine-grained analysis, we assume that the size of a preference formula C (over a relation R) in DNF is characterized by two parameters: $width(C)$—the number of disjuncts in C, and $span(C)$—the maximum number of conjuncts in a disjunct of C. Namely, if $C = D_1 \vee \cdots \vee D_m$, and each $D_i = C_{i,1} \wedge \cdots \wedge C_{i,k_i}$, then $width(C) = m$ and $span(C) = \max\{k_1, \ldots, k_m\}$.

THEOREM 3.1. *If a preference relation is defined using an ipf C containing only atomic constraints over the same domain and such that $width(C) \leq m$, $span(C) \leq n$, and the time complexity of checking satisfiability of a conjunctive ipf with n conjuncts is in $O(T(n))$, then the time complexity of checking the properties of the preference relation is in:*

— *$O(m\, T(n))$ for irreflexivity;*
— *$O(m^2\, T(n))$ for asymmetry;*
— *$O(m^2\, n^m\, T(\max(m, n)))$ for transitivity;*
— *$O(m\, n^{2m}\, T(\max(m, n)))$ for negative transitivity;*
— *$O(k\, n^{2m}\, T(m))$ for connectivity (where k is the arity of R).*

Additionally, the time complexity of checking whether a preference formula C_1 implies a preference formula C_2, where $width(C_1) \leq m$, $width(C_2) \leq m$, $span(C_1) \leq n$, and $span(C_2) \leq n$, is in $O(m\, n^m\, T(\max(m, n)))$.

PROOF. All of those results are proved in the same way. The property is negated, yielding a satisfiability problem. We discuss first asymmetry, the remaining properties can be handled in a similar way. If $C(t_1, t_2) = D_1 \vee \cdots \vee D_m$ and $C(t_2, t_1) = D'_1 \vee \cdots \vee D'_m$, we can write down the negation of asymmetry as $(D_1 \vee \cdots \vee D_m) \wedge (D'_1 \vee \cdots \vee D'_m)$. This formula is satisfiable iff at least one of m^2 conjunctive formulas $\phi_{i,j} = D_i \wedge D'_j$, $i, j = 1, \ldots, m$, is satisfiable. Testing for transitivity, negative transitivity, connectivity and implication requires writing down the negation of a DNF formula and distributing the negation inside. □

Guo et al. [1996] contain several results about checking satisfiability of conjunctive formulas. For instance, in the case of rational-order formulas, this problem is shown to be solvable in $O(n)$. This implies the following two corollaries.

COROLLARY 3.2. *Checking the properties of a preference relation defined by a rational-order ipf in DNF can be done in time:*

— *$O(m\, n)$ for irreflexivity;*
— *$O(m^2\, n)$ for asymmetry;*
— *$O(m^2\, n^m\, \max(m, n))$ for transitivity;*
— *$O(m\, n^{2m}\, \max(m, n))$ for negative transitivity;*
— *$O(k\, m\, n^{2m})$ for connectivity.*

Even more specifically, if a preference formula is conjunctive ($m = 1$), then all of the above properties can be checked in at most cubic time.

COROLLARY 3.3. *Implication of rational-order preference formulas can be checked in $O(m\, n^m\, \max(m, n))$ time.*

We note that the complexity here is measured *in the size of the preference formula*. The latter is unlikely to be large in practice. This should be contrasted with the *data* complexity measure [Vardi 1982]—commonly used to characterize the cost of evaluating queries—where the input, a database, can be huge.

Table I. Properties Preserved by Boolean Composition

	Union	Intersection	Difference
Strict partial order	No	Yes	No
Weak order	No	No	No
Total order	No	No	No

For equality constraints, the situation is very much alike. Checking satisfiability of conjunctive formulas can be done in almost-linear $O(n\,\alpha(n))$ time, where α is a very slowly growing function. This can be shown [Creignou et al. 2001] by adapting a unification algorithm that explicitly constructs equivalence classes of variables and constants using UNION-FIND, for example, that of Huet [Huet 1976; Knight 1989]. Thus, the complexity bounds obtained for checking various properties of equality ipfs will be very close to those obtained for rational-order ipfs. Therefore, in the rest of the paper we will only derive the latter in detail.

3.2 Indifference

Every preference relation $\succ_C$ generates an indifference relation $\sim_C$: two tuples t_1 and t_2 are *indifferent* ($t_1 \sim_C t_2$) if neither is preferred to the other one, that is, $t_1 \not\succ_C t_2$ and $t_2 \not\succ_C t_1$. Note that if the preference relation $\succ_C$ is irreflexive, we have for every tuple t, $t \sim_C t$.

PROPOSITION 3.4. *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 utility theory [Fishburn 1999, 1970] 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. COMPOSITION OF PREFERENCES

Preference relations may be composed in many different ways. In general, we distinguish between *unidimensional* and *multidimensional* composition. In unidimensional 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). We also consider the transitive closure of a preference relation. In multidimensional 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. Examples given below include Pareto and lexicographic composition (Definition 4.13). Moreover, we study the preservation of different kinds of orders by different composition operators. The results are summarized in Tables I and II.

In our framework, preference relations are defined by first-order preference formulas, therefore any first-order definable composition of preference relations leads again to first-order preference formulas. The composition does not even have to be first-order definable, as long as it produces a (first-order) preference

Table II. Properties Preserved by Other Kinds of Composition

	Prioritized composition	Pareto composition	Lexicographic composition
Strict partial order	No	No	No
Weak order	Yes	No	Yes
Total order	Yes	No	Yes

formula. We'll see an example of the latter later in this section when we discuss transitive closure.

4.1 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 $\succ_{C_0} = \succ_{C_1} \cap \succ_{C_2}$:

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

We characterize below the preservation of the properties of preference relations (identified in Section 2) by the Boolean operations.

PROPOSITION 4.1. *Strict partial order is preserved by intersection but not by union or set difference.*

PROOF. For union, consider

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

The relations $\succ_{C_1}$ and $\succ_{C_2}$ are strict partial orders, but the relation $\succ_{C_1} \cup \succ_{C_2}$ is not transitive (and thus not a partial order). For set difference, consider

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

The relation $\succ_{C_1} - \succ_{C_2}$ is not transitive (and thus not a partial order). □

PROPOSITION 4.2. *Weak order is not preserved by any Boolean operation.*

PROOF. To show that weak order is not preserved by union, we cannot simply refer to the first counterexample from the proof of Proposition 4.1. The orders used there were not weak, because they violated negative transitivity. The counterexample needs to be more elaborate. Consider:

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

The relations $\succ_{C_1}$ and $\succ_{C_2}$ are weak orders, but $\succ_{C_1 \vee C_2}$ is not because it is not transitive:

$$a \succ_{C_1 \vee C_2} b, b \succ_{C_1 \vee C_2} c, a \not\succ_{C_1 \vee C_2} c.$$

For intersection and difference, consider:

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

Then $\succ_{C_1 \wedge C_2} = \{(a, b)\}$ is not a weak order since it violates negative transitivity:

$$a \not\succ_{C_1 \wedge C_2} c, c \not\succ_{C_1 \wedge C_2} b, a \succ_{C_1 \wedge C_2} b.$$

Similarly, the difference of $\succ_{C_1}$ and $\succ_{C_2}$ consists only of the pair (a, c) and is not a weak order. □

PROPOSITION 4.3. *Total order is not preserved by any Boolean operation.*

PROOF. Take a total order and its inverse. Their union violates asymmetry. Their intersection is empty and thus not a total order. For difference, consider the difference of any total order and itself, whose result is empty. □

4.2 Prioritized Composition

It is often the case that preferences are prioritized. For instance, I may have a general preference for red wine but in specific cases, for example, 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. This situation may also occur in a multiagent environment where the preferences of one agent override those of another agent.

Definition 4.4. Consider two preference relations $\succ_{C_1}$ and $\succ_{C_2}$ defined over the same schema R. The *prioritized composition* $\succ_{C_{1,2}} \equiv \succ_{C_1} \rhd \succ_{C_2}$ of $\succ_{C_1}$ and $\succ_{C_2}$ is a preference relation over R defined as:

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

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

Example 4.5. 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 $\succ_{C_{0,1}} \equiv \succ_{C_0} \rhd \succ_{C_1}$ where $\succ_{C_0}$ is defined by the following formula C_0:

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

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.

PROPOSITION 4.6. *Weak and total orders are preserved by prioritized composition. Strict partial order is not preserved by prioritized composition.*

PROOF. The first two facts are shown by case analysis. The third can be obtained as for union (Proposition 4.1): prioritized composition does not have to preserve transitivity for arbitrary strict partial orders. □

THEOREM 4.7. *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, Andreka et al. [2002] define an infinite family of unidimensional 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 Andreka et al. [2002] can also be defined in ours. In Andreka et al. [2002], 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 Andreka et al. [2002], because it violates one of those postulates: it does not in general preserve transitivity.

4.3 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 4.8. 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 $\succ_C$ 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 4.9. The *transitive closure* of a preference relation $\succ_C$ over a relation schema R is a preference relation $\succ_{C^*}$ over R defined as:

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

where:

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

Clearly, in general Definition 4.9 leads to infinite formulas. However, as Theorem 4.10 shows, in the cases we consider in this article, the preference relation $\succ_{C*}$ will in fact be defined by a finite formula.

THEOREM 4.10. *If a preference relation $\succ_C$ is defined using an (equality or rational-order) ipf C, the transitive closure $\succ_{C^*}$ of $\succ_C$ is also defined using an (equality or rational-order) 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 Constraint Datalog. Suppose C is in DNF and thus $\succ_C$ 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:

$$\begin{aligned} & T(x, y) \leftarrow \alpha_1(x, y). \\ & \ldots \\ & T(x, y) \leftarrow \alpha_n(x, y). \\ & S(x, y) \leftarrow T(x, y). \\ & S(x, y) \leftarrow T(x, z), S(z, y). \end{aligned}$$

The evaluation of this program terminates [Kanellakis et al. 1995] and its result, collected in S, represents the desired formula. □

Example 4.11. Continuing Example 4.8, 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 4.10 is not in conflict with the well-known nonfirst-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 4.10, the relation to be closed, although possibly infinite, is fixed and finitely representable (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 use of Datalog in Theorem 4.10 is at a meta-level. The Datalog program computes a preference formula, which can in turn be used as a parameter of winnow. Datalog is *not* used for query evaluation. (Moreover, this is Constraint Datalog, not relational Datalog. The latter can to a large degree be handled by SQL:1999 systems.)

4.4 Multidimensional Composition

In decision theory, various ways of defining a preference relation on the Cartesian product of two relations have been studied. The two most common ones are Pareto and lexicographic composition.

Definition 4.12. Given two relation schemas R_1 and R_2 and preference relations $\succ_{C_1}$ over R_1 and $\succ_{C_2}$ 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

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

Definition 4.13. Given two relation schemas R_1 and R_2 and preference relations $\succ_{C_1}$ over R_1 and $\succ_{C_2}$ over R_2, the *lexicographic composition* $L(\succ_{C_1}, \succ_{C_2})$ *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:

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

The preservation properties of different kinds of orders with respect to Pareto and lexicographic composition are formulated below.

PROPOSITION 4.14. *Strict partial order is not preserved by Pareto or lexicographic composition.*

PROOF. Consider the following preference relations $\succ_{C_1}$ and $\succ_{C_2}$ (and the associated indifference relations):

$$c_1 \succ_{C_1} a_1, a_1 \sim_{C_1} b_1, b_1 \sim_{C_1} c_1, a_2 \succ_{C_2} b_2, b_2 \succ_{C_2} c_2.$$

Assuming $C_0 = P(\succ_{C_1}, \succ_{C_2})$, we have:

$$(a_1, a_2) \succ_{C_0} (b_1, b_2), (b_1, b_2) \succ_{C_0} (c_1, c_2)$$

but $(a_1, a_2) \sim_{C_0} (c_1, c_2)$, violating the transitivity of $\succ_{C_0}$. In the same example, $L(\succ_{C_1}, \succ_{C_2})$ is identical to $P(\succ_{C_1}, \succ_{C_2})$, and thus also fails to be transitive. □

It is easy to see that if $\succ_{C_1}$ and $\succ_{C_2}$ are weak orders, then both $P(\succ_{C_1}, \succ_{C_2})$ and $L(\succ_{C_1}, \succ_{C_2})$ are strict partial orders. In fact, lexicographic composition preserves weak and total orders.

PROPOSITION 4.15. *Weak and total orders are preserved by lexicographic composition but not by Pareto composition.*

PROOF. The first two facts are shown by case analysis. To see the failure of Pareto composition in preserving weak and total orders, consider the example in which:

$$t_1 \succ_{C_1} t_1'' \succ_{C_1} t_1'$$

and

$$t_2' \succ_{C_2} t_2 \succ_{C_2} t_2''.$$

Thus $\succ_{C_1}$ and $\succ_{C_2}$ are total orders (and thus also weak). If $C_0 = P(\succ_{C_1}, \succ_{C_2})$ then

$$(t_1, t_2) \sim_{C_0} (t_1', t_2'),\ (t_1', t_2') \sim_{C_0} (t_1'', t_2'')$$

but

$$(t_1, t_2) \succ_{C_0} (t_1'', t_2'').$$

Thus the indifference $\sim_{C_0}$ is not transitive, which means that $\succ_{C_0}$ is not a weak order (and thus not a total order as well). □

5. THE WINNOW OPERATOR

In this section, we study various properties of the winnow operator, including nonemptiness, monotonicity, and expressive power. In the next section, we discuss the algebraic properties of winnow: commutativity and distributivity. Establishing 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.

5.1 Basic Properties

Several properties of winnow, also identified in Kießling and Köstler [2002], follow directly from Definition 2.3.

PROPOSITION 5.1. *For every preference relations $\succ_{C_1}$ and $\succ_{C_2}$ over a relation schema R and every instance r of R:*

$$\begin{aligned}
&\omega_{C_1}(r) \subseteq r\\
&\omega_{C_1}(\omega_{C_1}(r)) = \omega_{C_1}(r)\\
&\omega_{C_1 \vee C_2}(r) = \omega_{C_1}(r) \cap \omega_{C_2}(r)\\
&\omega_{False}(r) = r\\
&\omega_{True}(r) = \emptyset.
\end{aligned}$$

Note that $\omega_{C_1 \wedge C_2}(r) = \omega_{C_1}(r) \cup \omega_{C_2}(r)$ does *not* hold in general.

5.2 Nonemptiness

THEOREM 5.2. *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.*

We show now that violating the premisses in Theorem 5.2 results in the violation of the hypothesis. First, consider relaxing irreflexivity of $\succ_C$. Thus there is a tuple t_0 such that $t_0 \succ_C t_0$. But then $\omega_C(\{t_0\}) = \emptyset$. Second, if two tuples are involved in a violation of asymmetry, they block each other from appearing in the result of the winnow operator. Third, without transitivity a preference relation may be asymmetric but have a cycle of length three or more, resulting in the result of the winnow being empty. Finally, 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 $>$.

5.3 Monotonicity

The winnow operator is not *monotone* or *antimonotone* with respect to its relation argument.

Example 5.3. 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 antimonotonicity fail.

However, partial antimonotonicity holds:

PROPOSITION 5.4. [CHERNOFF 1954]. *If a preference relation $\succ_C$ over R is a strict partial order, then for every pair of instances r_1 and r_2 of R:*

$$r_1 \subseteq r_2 \ \Rightarrow\ r_1 \cap \omega_C(r_2) \subseteq \omega_C(r_1).$$

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

THEOREM 5.5. *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 $\succ_{C_2}$ is irreflexive, 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 $t_2 \notin \omega_{C_1}(r_{12})$ but $t_2 \in \omega_{C_2}(r_{12})$, a contradiction. □

Theorem 5.5 implies that the *query containment* problem for winnow is identical to the *implication* problem for preference formulas. For example, if for the class of preference formulas considered the implication problem is decidable, then query containment for winnow can also be effectively tested.

5.4 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.

By *standard relational algebra*, we mean relational algebra with the following operators: selection, projection, Cartesian product, union, set difference and renaming.

THEOREM 5.6. *The expressive power of the standard relational algebra does not change if set difference is replaced by winnow.*

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 $C = D_1 \vee \cdots \vee D_k$ which is an (equality or rational-order) ipf 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

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

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

Finally, we show 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

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

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 (respectively, S) are tagged with 1 (respectively 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 $\sigma_{B=1}$ eliminates all the tuples in S, keeping the tuples that are only in R. □

5.5 Evaluating Winnow

Winnow is somewhat similar to the set difference operation in the sense that a tuple is in the result of winnow if it is not *killed* by another tuple. For set difference, killing means encountering another occurrence of the *same* tuple in the second relation. For winnow, it means encountering an occurrence of a *dominating* tuple. Although it has only one argument, winnow is more like a binary operation than a unary one, because it requires comparing each tuple to multiple (possibly all) tuples from the same input relation.

We show here several algorithms that can be used to compute the result of the winnow operator $\omega_C(R)$. Not surprisingly, the first is a simple nested-loops

```
(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 $S_2$:
        if $t_2 \succ_C t_1$, then goto 2d;
    (c) output $t_1$;
    (d) close $S_2$;
(3) close $S_1$.
```

Fig. 2. NL: Nested Loops.

```
(1) clear the window $W$ and the temporary table $F$;
(2) make $r$ the input;
(3) 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 $t$,
        —$t$ 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 the temporary table.
```

Fig. 3. BNL: Blocked Nested Loops.

algorithm (Figure 2). The second is BNL, an algorithm proposed in Börzsönyi et al. [2001] in the context of *skyline queries*, a specific class of preference queries, but the algorithm is considerably more general (Figure 3). The third [Chomicki et al. 2003] is a variant of the second, in which a presorting step is used (Figure 4). All the algorithms use 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.

The NL algorithm is correct for any preference relation $\succ_C$. In principle, the preference relation might even be reflexive, since the algorithm explicitly compares a tuple with itself. The BNL and SFS algorithms require the preference relation be a strict partial order (for BNL this is noted in Börzsönyi et al. [2001]). 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.

```
(1) topologically sort $r$ according to $\succ_C$;
(2) make $r$ the input;
(3) clear the window $W$ and the temporary table $F$;
(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 $W$, clear $W$.
    (c) make $F$ the input, clear the temporary table.
```

Fig. 4. SFS: Sort-Filter-Skyline.

Example 5.7. The preference relation C_0 is defined as follows:

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

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 killing c. Therefore, BNL will output a (correctly) and c (incorrectly). This example can be easily generalized to any fixed window size, simply by assuming that a and b are separated in the input by sufficiently many values different from a, b and c.

6. ALGEBRAIC PROPERTIES OF WINNOW

We present here a set of algebraic laws that govern the commutativity and distributivity of winnow with respect to 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.

6.1 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. Unfortunately, it seems that commutativity requires idempotence, that is, two consecutive occurrences of winnow commute if they can be collapsed to one of them.

THEOREM 6.1. *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*
— *$\succ_{C_1}$ 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 5.5, $t \notin \omega_{C_2}(r)$, 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 5.5 we get $t' \notin \omega_{C_2}(r)$, a contradiction. □

Consider now what happens if the assumptions in Theorem 6.1 are relaxed.

Example 6.2. 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

$$\omega_{C_1}(\omega_{C_2}(r_1)) = \{(1, 1975, 100K)\} \neq \{(2, 1980, 150K)\} = \omega_{C_2}(\omega_{C_1}(r_1)).$$

Example 6.3. Consider the instance $r_2 = \{a, b\}$ and 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

$$\omega_{C_1}(\omega_{C_2}(r_2)) = \emptyset \neq \{a\} = \omega_{C_2}(\omega_{C_1}(r_2)).$$

Theorem 3.1 gives a general characterization of the time complexity of checking implication and the strict partial order property of preference formulas, thus it can be directly used to establish the computational cost of checking the precondition for the commutativity of winnow. In particular, it follows from Corollary 3.3 that checking this precondition can be done in time quadratic in the size of the formulas, if the formulas are conjunctions of rational-order constraints.

6.2 Commuting Selection and Winnow

We identify in Theorem 6.4 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). In an operation like winnow that requires comparing pairs of tuples from the same relation, it is "doubly" beneficial to push a selection down, since that reduces the number of tuples on both sides of the comparisons. Only in extreme cases, it would be advantageous to pull a selection up through a winnow.

THEOREM 6.4. *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:

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

Clearly, the first formula implies the second. To see that the opposite implication also holds, assume $t \notin \sigma_{C_1}(\omega_{C_2}(r))$. There are three cases: $C_1(t)$ does not hold, $t \notin r$, and $C_1(t)$ holds but there is a tuple $t_0 \in r$ such that $C_2(t_0, t)$. In the first two cases, it is immediately clear that $t \notin \omega_{C_2}(\sigma_{C_1}(r))$. In the third case, $C_1(t_0)$ holds too, because 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. However, then $t_0 \in \sigma_{C_1}(r)$, which implies that $t \notin \omega_{C_2}(\sigma_{C_1}(r))$.

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 formulas C_1 and C_2 in Theorem 6.4 are rational-order ipfs in DNF, it follows from Theorem 3.1 that the time complexity of checking the validity of the formula $\forall(C_1(t_2) \wedge C_2(t_1, t_2)) \Rightarrow C_1(t_1)$ is in $O(m_1\, m_2\, n_1^{m_1} \max(n_1, n_2, m_1))$ time, where $m_1 = \mathit{width}(C_1)$, $m_2 = \mathit{width}(C_2)$, $n_1 = \mathit{span}(C_1)$, and $n_2 = \mathit{span}(C_2)$. So if both C_1 and C_2 are conjunctive ($m_1 = m_2 = 1$), checking the precondition of Theorem 6.4 can be done in quadratic time.

Example 6.5. 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 6.4

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

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.

6.3 Commuting Projection and Winnow

We deal now with projection. For winnow to commute with projection, the preference formula—which is the parameter of the winnow—needs to be restricted to the attributes in the projection. Below we propose a restriction of this kind. 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 6.6. 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 $\succ_{C'}$ over $\pi_X(R)$ 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 $\theta_x(\succ_C)$.

In the following theorem, a precondition is imposed on the given preference relation to allow winnow to commute with projection. The precondition captures the intuition that the preference formula *depends only on the attributes in the projection*. Clearly, if the formula uses the projected-out attributes in an essential way, the winnow will usually fail to commute with projection.

THEOREM 6.7. *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, that is, 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 $t_2 \not\succ_C t_3$. 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 $t_1[X] \not\succ_{C'} t_3[X]$ (because $t_2 \not\succ_C t_3$) 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 formula C in Theorem 6.7 is a rational-order ipf in DNF, it follows from Theorem 3.1 that checking the precondition in Theorem 6.7 can be done in time $O(m\, k\, n^m \max(k, m, n))$, where $m = width(C)$, $n = span(C)$, and $k = |X|$. So if C is conjunctive ($m = 1$) and X fixed, this task can be accomplished in quadratic time. If C is an ipf, then C' can be presented in an equivalent, quantifier-free form.

Example 6.8. 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 preference relation

$$\succ_{C'} = \theta_{ISBN,Price}(\succ_{C_1})$$

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 6.7 is also satisfied, so winnow commutes with projection in this case.

6.4 Distributing Winnow over Cartesian Product

For winnow to distribute (possibly in a modified form) over the Cartesian product of two relations, the preference formula—the parameter of the winnow—needs to be decomposed into the formulas that will distribute into the argument relations. Thus, it is natural to consider in this context preference relations that are themselves built by multidimensional composition. In Section 3, we have defined two kinds of multidimensional composition: Pareto and lexicographic, both well known in multi-attribute utility theory [Fishburn 1970]. Preference queries involving Pareto composition are quite common: the skyline queries [Börzsönyi et al. 2001] without DIFF attributes, discussed in Section 7, are of this form.

THEOREM 6.9. *Given two relation schemas R_1 and R_2, an irreflexive preference relation $\succ_{C_1}$ over R_1 and an irreflexive 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, respectively, 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 $(t_1, t_2) \in r_1 \times r_2$ 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 $(t_1', t_2) \succ_{C_0} (t_1, t_2)$ (we use irreflexivity of $\succ_{C_2}$ to infer $t_2 \sim_{C_2} t_2$), 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 $(t_1, t_2) \in \omega_{C_1}(r_1) \times \omega_{C_2}(r_2)$. □

Theorem 6.9 makes it possible to derive an interesting corollary that validates the transformation rule pushing winnow through Cartesian product. Assume that $\succ_{C_2}$ is the empty relation, and $\sim_{C_2}$ is the complete relation (i.e., contains every pair of tuples). Then, $\succ_{C_0}$ is defined purely in terms of the first dimension:

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

In this case Theorem 6.9 implies that for every r_1 and r_2

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

because $\omega_{C_2}(r_2) = r_2$ for every instance r_2. So we can say that winnow with a one-dimensional preference formula can be pushed down the appropriate argument of the product.

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 6.10. 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

$$\omega_{C_1}(r_1) \times \omega_{C_2}(r_2) = \{(1, 2)\} \neq \{(1, 1), (1, 2)\} = \omega_{C'_0}(r_1 \times r_2).$$

For lexicographic composition, we obtain the same property as for Pareto composition. Its proof parallels that of Theorem 6.9.

THEOREM 6.11. *Given two relation schemas R_1 and R_2, an irreflexive preference relation $\succ_{C_1}$ over R_1 and an irreflexive 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 $C_0 = L(\succ_{C_1}, \succ_{C_2})$.

6.5 Distributing Winnow over Union and Difference

For completeness, we show that it is possible to distribute winnow over union or difference only in the trivial case where the preference relation is an empty set. We call two relation schemas *compatible* if they have the same number of attributes and the corresponding attributes have the same domains.

THEOREM 6.12. *Given two compatible relation schemas R and S and an asymmetric preference relation $\succ_C$ over R, we have for every instance r of R and every instance s of 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 $\succ_C = \emptyset$, 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 $t_1 \succ_C t_2$, and $t_2 \not\succ_C t_1$ (by asymmetry). 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. □

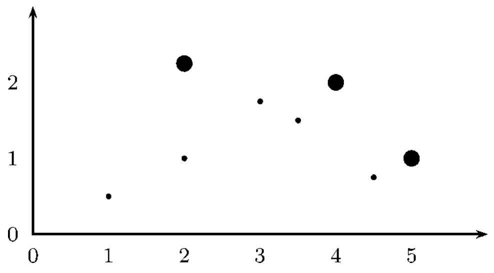

Fig. 5. Two-dimensional skyline.

7. 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.

7.1 Special Classes of Preference Queries

7.1.1 *Skylines.* *Skyline queries* [Börzsönyi et al. 2001] 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 4.12) in an arbitrary number of dimensions. Typically, skyline queries are formulated in the context of multidimensinal Euclidean space where the dominance relationship is $>$ or $<$. Figure 5 shows an example of a skyline in two-dimensional Euclidean space where the dominance relationship is $>$. The skyline elements are marked with thick black dots.

Börzsönyi et al. [2001] 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 nonskyline constructs in a skyline query. The corresponding rational-order ipf is easily obtained from the SKYLINE clause.

Example 7.1. The skyline query

```
SELECT * FROM R
SKYLINE OF A DIFF, B MAX, C MIN
```

is equivalent to $\omega_C(R)$ where

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

We note that $\omega_{C_1}(Book)$ from Example 2.4 is also a skyline query in which the skyline clause looks as follows:

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

Thus, clearly skylines cannot be captured using scoring functions (Example 1.1).

In view of the above, the algebraic laws that characterize the properties of winnow are also applicable to skylines. Thus, Corollary 3.3 implies that if a selection condition is a rational-order ipf, then checking whether it commutes with a skyline can be done in $O(m_1\, k\, n_1^{m_1} \max(n_1, k))$ time, where m_1 is the width of the selection condition, n_1 its span, and k the dimension of the skyline space. Thus, if the selection condition is conjunctive ($m_1 = 1$), checking the commutativity precondition can be done in cubic time.

7.1.2 *Queries Involving Scoring Functions.* We compare here two different ways of representing preferences: a *qualitative* one using binary preference relations and a *quantitative* one using scoring (also called utility) functions.

Definition 7.2. A real-valued function f over a schema R *represents* a preference relation $\succ_C$ over R iff

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

In other words, f is an *order-isomorphism*.

We can ask for the motivation behind this notion of representation. It is easy to show that

THEOREM 7.3. *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 f.*

Thus in this case a particularly simple, linear-time method of computing $\omega_C(r)$ becomes possible: determine the maximum value of f in r and return the tuples in r that assume it. And vice-versa, finding the tuples in an instance r that maximize a scoring function f can be done by computing $\omega_C(r)$. Moreover, the algebraic laws for winnow presented in Section 5 become applicable to queries with scoring functions.

Theorem 7.3 implies that if a scoring function does not represent a preference relation, that fact can be detected by winnow evaluated over some instance.

Example 7.4. Consider again Example 1.1. Let $r_1 = \{t_1, t_2, t_3, t_4, t_5\}$ be the instance from that example where t_1 is the first tuple etc. We can construct a scoring function f with the property that *in the given instance* r_1 it is maximized in exactly those tuples that are in $\omega_{C_1}(r_1)$. For example, let $f(t_1) = 0.9$, $f(t_2) = 1.0$, $f(t_3) = 0.8$, $f(t_4) = 1.0$ and $f(t_5) = 1.0$. Then f is maximized in t_2, t_4 and t_5, and $\omega_{C_1}(r_1) = \{t_2, t_4, t_5\}$. However, in the instance $r_1' = \{t_1, t_3, t_4, t_5\}$, the function f is maximized in t_4 and t_5, while $\omega_{C_1}(r_1') = \{t_1, t_4, t_5\}$.

Unfortunately, as pointed out earlier, not every preference relation that is a strict partial order can be expressed using a scoring function. A necessary condition is that the relation be a *weak order* [Fishburn 1970]. Therefore, the approach using preference relations is strictly more general than the one that uses scoring functions.

Winnow can be used not only to compute 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-K* scores [Carey and Kossmann 1997; Bruno et al. 2002] 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 N 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-K queries are supported directly by the query engine.

Finally, we note that there are other, weaker forms of representation than Definition 7.2. 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 [Fishburn 1970]. Such a function is an *order-homomorphism*. However, in that case, we can only guarantee that the set of tuples in a given instance r that maximize f is a *subset* of $\omega_C(r)$.

7.2 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 2.5, 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 6.4 identifies sufficient and necessary conditions for winnow and selection to commute.

Example 7.5. Consider the situation where we have a specific preference ordering for cars, for example, 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:

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

7.3 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, for example, 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 query constructs using the following multistep 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 7.6. 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 the *Book* 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:

$$(i, v, n) \succ_{C_5} (i', v', n') \equiv i = i' \wedge n > n'.$$

Example 7.7. 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:

```
SELECT $A_1, \ldots, A_n, 'r'$ FROM $R$
UNION
SELECT $A_1, \ldots, A_n, 's'$ FROM $S$.
```

Then the preference relation is defined using the tags:

$$(x_1, \ldots, x_n, t) \succ_C (x'_1, \ldots, x'_n, t') \equiv t = 'r' \wedge t' = 's'.$$

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 2.5 (wine and dish types play the role of tags).

Example 7.8. Suppose user preferences are stored in a database relation *Pref(A,B)*. Then one can define an extrinsic preference relation:

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

Such a preference relation cannot be defined using an intrinsic preference formula, because each ipf is true of a fixed set of tuples, and one can always choose the instance of *Pref* to be different from that set.

8. ITERATED PREFERENCES AND RANKING

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

Definition 8.1. Given a preference relation $\succ$ defined by a pf C, the nth *iteration* of the winnow operator ω_C in r is defined as:

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

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

Example 8.2. Continuing Example 1.1, $\omega^2_{C_1}(r_1)$ returns

ISBN	*Vendor*	*Price*
0679726691	BooksForLess	$14.75

and $\omega^3_{C_1}(r_1)$ 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 8.3. *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 $t \in \omega^i_C(r)$.*

PROOF. Assume there is a tuple $t_0 \in r$ such that for all $i \geq 1$, $t_0 \notin \omega^i_C(r)$. Select the least i_0 such that $\forall i \geq i_o$, $\omega^i_C(r) = \emptyset$ (such an i_0 always exists due to the finiteness of r). Clearly, $t_0 \in r - \bigcup_{1\leq i\leq i_0-1} \omega^i_C(r)$. Because $t_0 \notin \omega^{i_0}_C(r)$, there must be a tuple t_1 such that $t_1 \succ_C t_0$ and $t_1 \in r - \bigcup_{1\leq i\leq i_0-1} \omega^i_C(r)$ (otherwise, $t_0 \in \omega^{i_0}_C(r)$). But $t_1 \notin \omega^{i_0}_C(r)$ either. 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. □

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

Definition 8.4. 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:

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

Note that the ranking operator is defined inductively using recursion with non-stratified negation. It cannot be expressed in SQL, even using the recursive definitions of SQL:1999 [Eisenberg and Melton 1999]. On the other hand, ranking can be efficiently computed using extended versions of the evaluation algorithms BNL and SFS (Section 5). For instance, SFS can use multiple windows, one for each rank, and compare input tuples with the contents of each window (in rank order). This will be discussed in more detail in the full version of Chomicki et al. [2003].

One can study the algebraic properties of the ranking operator that parallel those that we established for winnow in Section 5. 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 8.5. *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 6.4 and the inductive case from the observation that

$$\sigma_{C_1}\left(\omega^{n+1}_{C_2}(r)\right) = \sigma_{C_1}\left(\omega_{C_2}\left(r - \bigcup_{1\leq i\leq n} \omega^i_{C_2}(r)\right)\right) = \omega_{C_2}\left(\sigma_{C_1}\left(r - \bigcup_{1\leq i\leq n} \omega^i_{C_2}(r)\right)\right),$$

which is equal to

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

By the inductive assumption, the last expression is equal to

$$\omega_{C_2}\left(\sigma_{C_1}(r) - \bigcup_{1\leq i\leq n} \omega^i_{C_2}(\sigma_{C_1}(r))\right) = \omega^{n+1}_{C_2}(\sigma_{C_1}(r)). \quad \square$$

9. RELAXED WINNOW

If a preference relation is not a strict partial order, then Theorems 5.2 and 8.3 may fail to hold. This may occur, for example, if the preference relation is obtained through composition (Section 4). However, even in this case there may be weaker forms of winnow and ranking available.

Example 9.1. Consider Examples 1.1 and 4.5. 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, relaxed form of the winnow operator. We relax the irreflexivity requirement but preserve transitivity. Thus, we also allow violations of asymmetry.

To define the relaxed winnow 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 9.2. If R is a relation schema and $\succ_C$ a transitive preference relation over R, then the *relaxed winnow operator* is written as $\psi_C(R)$ and for every instance r of R, $\psi_C(r) = \omega_{C_>}(r)$.

As the following theorem shows, the relaxed winnow operator returns all the tuples that are dominated only by the tuples that they dominate themselves.

THEOREM 9.3. *If R is a relation schema and $\succ_C$ a transitive preference relation over R, then for every instance r of R*

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

PROOF. We have

$$t \in \psi_C(r) \equiv t \in \omega_{C_>}(r) \equiv t \in r \wedge \neg\exists t' \in r.\ t' \succ_{C_>} t \equiv$$
$$t \in r \wedge \neg\exists t' \in r.\ t' \succ_C t \wedge t \not\succ_C t' \equiv t \in r \wedge \forall t' \in r.\ t' \not\succ_C t \ \vee\ t \succ_C t'. \quad \square$$

Example 9.4. Considering Example 9.1, 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 relaxed winnow operator. We note that because of Definition 9.2, the implementation algorithms and the properties of winnow (including the algebraic properties) immediately carry over to relaxed winnow. Using Theorems 5.2 and 5.5 (notice that $C_> \Rightarrow C$), we obtain the following theorem.

THEOREM 9.5. *If R is a relation schema and $\succ_C$ a transitive preference relation over R, then:*

—*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 relaxed winnow operator similarly to that of the winnow operator (Definition 8.1).

THEOREM 9.6. *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 $t \in \psi^i_C(r)$.*

10. RELATED WORK

10.1 Preference queries

Lacroix and Lavency [1987] 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: Pick the tuples of R 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 tuples satisfying $Q \wedge \neg P_1 \wedge P_2$. This can be simulated in our framework in a very natural fashion as

$$\omega_{C_2}(\omega_{C_1}(\sigma_Q(R)))$$

where

$$C_1(t_1, t_2) \equiv P_1(t_1) \wedge \neg P_1(t_2)$$
$$C_2(t_1, t_2) \equiv P_2(t_1) \wedge \neg P_2(t_2).$$

Other kinds of logical conditions from Lacroix and Lavency [1987] can also be expressed in our framework. Maximum/minimum value preferences (as in Example 1.1) are handled in Lacroix and Lavency [1987] through the explicit use of aggregate functions. The use of such functions is implicit in the definition of the winnow operator.

Unfortunately, Lacroix and Lavency [1987] 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 Lacroix and Lavency [1987] seems unable to capture very simple conditional preferences like the ones in Examples 2.5 and 4.8. 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). Composition or iteration of preferences is not considered. Neither is addressed the issue of algebraic optimization of preference queries.

It is interesting to note that a logical approach to preferences, similar to the one proposed in the present article, was first studied in the context of deductive databases in Kießling and Güntzer [1994] and Köstler et al. [1995] and—independently—in Govindarajan et al. [1995, 2001]. All of those papers propose extending Datalog with clausally defined preference relations. Govindarajan et al. [2001] demonstrate that this approach subsumes, among others, the class of preference queries discussed in Lacroix and Lavency [1987]. The cited papers present declarative and operational semantics for the proposed Datalog extensions. The operational semantics extends the standard bottom-up [Köstler et al. 1995; Govindarajan et al. 2001] or top-down [Govindarajan et al. 1995] evaluation method for logic programs. In the context of database queries, the approach proposed in the present article achieves similar goals to that of Kießling and Güntzer [1994], Köstler et al. [1995], and Govindarajan et al. [1995, 2001], remaining, however, entirely within the relational data model and SQL, and not requiring a specialized query evaluation engine. Moreover, Kießling and Güntzer [1994], Köstler et al. [1995], and Govindarajan et al. [1995, 2001] do not address some of the issues we deal with in the present paper like transitive closure of preferences, prioritized composition, or ranking (a similar concept to the last one is presented in Govindarajan et al. [1995] 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.

Kießling et al. [Kießling 2002; Kießling and Köstler 2002; Kießling and Hafenrichter 2002] propose an independently developed framework similar to the one presented in this article and in Chomicki [2002]. 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). In this language, all the examples we have presented in Section 2 can be expressed. Clearly, all the preference constructors and their combinators can be captured in our framework. On the other hand, Kießling [2002] and Kießling and Köstler [2002] do not consider the possibility of having arbitrary constraints in preference formulas.

Example 10.1. The following preference relation cannot be expressed in the framework of Kießling et al.:

$$(x, y) \succ_C (x', y') \equiv x = y \wedge x' \neq y'.$$

However, the above framework has been recently extended with a facility to define preference relations using arbitrary Boolean tests (W. Kießling, 2003, personal communication). Moreover, Kießling et al. allow only very limited versions of extrinsic ("persistent") preferences and transitive closure. The embedding into relational query languages they use is identical to ours (it is called Best Match Only, instead of winnow). Thus, although complex preferences involving aggregation are not explicitly considered in Kießling [2002] and Kießling and Köstler [2002], such preferences can be formulated in that framework.

While some possible rewritings for preference queries are presented in Kießling [2002] and some algebraic laws in Kießling and Hafenrichter [2002], those results are mostly less general that those we gave in Section 6. In particular, Theorems 6.1 and 6.4 have no corresponding results in Kießling [2002], Kießling and Köstler [2002], and Kießling and Hafenrichter [2002], while Theorems 6.7, 6.9 and 6.11 are significantly more general that the corresponding results in [Kießling and Hafenrichter 2002]. On the other hand, Kießling and Hafenrichter [2002] contains several results about pushing a unary operator or winnow into a subtree headed by a binary operator or winnow, without removing the occurrence of the top-level operator. We didn't consider this kind of transformation. Also, Kießling and Hafenrichter [2002] has a result about pushing winnow through selection which is not a special case of Theorem 6.4. However, that does not contradict the necessity of the precondition used in that theorem, since the proposed transformation is applicable only if the argument of the selection is of a special form. Further work in this direction is presented in Kießling and Hafenrichter [2003]. The implementation of winnow and ranking is also studied in Torlone and Ciaccia [2002, 2003].

In general, the logical formulation of preferences in our approach makes it possible to view the preconditions for the application of algebraic laws involving winnow as *logical validity problems*. For instance, we can capture the connection between a selection condition and a preference criterion, since both are expressed as logical formulas. This is not directly possible in the approach of Kießling et al. On the other hand, the constructs proposed by Kießling et al. can be formulated logically, enabling the application of our results in that context as well. Kießling and Köstler [2002] describes an implementation of the framework of Kießling [2002] using a language called Preference SQL, which is translated to SQL, and several deployed applications.

Börzsönyi et al. [2001] introduces the *skyline* operator and describes several evaluation methods for this operator. As shown in Section 7, skyline is a special case of winnow. It is restricted to use an ipf which is a conjunction of pairwise comparisons of corresponding tuple components. So, in particular, Example 2.5 does not fit in that framework. Some examples of possible rewritings for skyline queries are given in Börzsönyi et al. [2001] but no general rewriting rules are formulated. Algortithms for computing skylines are also described in Kossmann et al. [2002], Papadias et al. [2003], and [Chomicki et al. 2003].

The algorithms in Kossmann et al. [2002] and Papadias et al. [2003] are based on nearest-neighbor computations and thus do not seem to generalize beyond skyline queries.

Agrawal and Wimmers [2000] uses quantitative preferences (scoring functions) in queries and focuses on the issues arising in combining such preferences. Hristidis et al. [2001] explores in this context the problems of efficient query processing using materialized views. As pointed out repeatedly in the present paper (and well known in decision theory [Fishburn 1970]), the approach based on scoring functions is inherently less expressive than the one based on preference relations. In particular, skyline queries cannot be captured using scoring functions. Moreover, since the quantitative approach is based on comparing the scores of individual tuples under given scoring functions, the preferences represented in this way have to be intrinsic. However, the simulation of extrinsic preferences using intrinsic ones (Section 7) is not readily available in this approach because the scoring functions are not integrated with the query language. So, for instance, Example 7.6 cannot be handled. In fact, even for preference relations that satisfy the property of transitivity of the corresponding indifference relation (and thus representable using scoring functions), 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 [Fishburn 1999, 1970] does not provide such a definition. Moreover, it is not clear how to *compose* scoring functions to achieve an effect similar to various preference relation composition operators, for example, those discussed in Section 4.

10.2 Preferences in Logic and Artificial Intelligence

The papers on *preference logics* [von Wright 1963; Mantha 1991; Hansson 2001] 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 Mantha [1991] and Cristani [2002] limited to the propositional case. Mantha [1991] proposes a modal logic of preference, and Cristani [2002] 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* [Wellman and Doyle 1991; Tan and Pearl 1994; Boutilier et al. 1999] attempt to develop practical mechanisms for making inferences about preferences and solving decision or configuration problems similar to the one described in Example 2.5. A central notion there is that of *ceteris paribus* preference: preferring one outcome to another, all else being equal. Typically, the problems addressed in that work are propositional (or finite-domain). It appears that 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* [Brewka and Eiter 1999; Delgrande et al. 2000; Sakama and Inoue 2000] has potential applications to databases. However, like Govindarajan et al. [2001], it relies on specialized evaluation mechanisms, and the preferences considered are typically limited to rule priorities.

11. 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 where the preferences 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 2.5?);
—preferences requiring nondeterministic choice. We believe this is properly handled using a nondeterministic choice [Giannotti et al. 1997] or witness [Abiteboul et al. 1995] operator.

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

—evaluation and optimization of preference queries, including cost-based optimization;
—materialized preference views (views defined using preference queries);
—extrinsic preferences;
—multiagent preferences and preference revision [Wong 1994];
—preference query elicitation (how to construct preference queries based on user input).

ACKNOWLEDGMENTS

This paper is dedicated to the memory of Javier Pinto whose premature death prevented him from participating in this research. The numerous discussions on preferences with Svet Braynov, Agnieszka Grabska, Jarek Gryz, Bharat Jayaraman, and Jorge Lobo were highly appreciated. Thanks also go to Werner Kießling for his detailed comments and to the referees for their useful suggestions.

REFERENCES

Abiteboul, S., Hull, R., and Vianu, V. 1995. *Foundations of Databases*. Addison-Wesley, Reading, Mass.

Agrawal, R. and Wimmers, E. L. 2000. A framework for expressing and combining preferences. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. ACM, New York, 297–306.

ANDREKA, H., RYAN, M., AND SCHOBBENS, P.-Y. 2002. Operators and laws for combining preference relations. *J. Logic Comput. 12*, 1, 13–53.

BÖRZSÖNYI, S., KOSSMANN, D., AND STOCKER, K. 2001. The skyline operator. In *Proceedings of the IEEE International Conference on Data Engineering*. IEEE Computer Society Press, Los Alamitos, Calif., 421–430.

BOUTILIER, C., BRAFMAN, R. I., HOOS, H. H., AND POOLE, D. 1999. Reasoning with conditional ceteris paribus preference statements. In *Proceedings of the Symposium on Uncertainty in Artificial Intelligence*.

BREWKA, G. AND EITER, T. 1999. Preferred answer sets for extended logic programs. *Artif. Intel. 109*, 1-2, 297–356.

BRUNO, N., CHAUDHURI, S., AND GRAVANO, L. 2002. Top-*k* selection queries over relational databases: Mapping strategies and performance evaluation. *ACM Trans. Datab. Syst. 27*, 2 (June), 153–187.

BURKE, R. 1999. Integrating knowledge-based and collaborative-filtering recommender systems. In *Proceedings of the AAAI Workshop on AI and Electronic Commerce*.

CAREY, M. AND KOSSMANN, D. 1997. On saying enough already! in SQL. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. ACM, New York, 219–230.

CHERNOFF, H. 1954. Rational selection of decision functions. *Econometrica 22*, 422–443.

CHOMICKI, J. 2002. Querying with intrinsic preferences. In *Proceedings of the International Conference on Extending Database Technology*. Springer-Verlag, Lecture Notes in Computer Science, vol. 2287, Springer-Verlag, New York, 34–51.

CHOMICKI, J., GODFREY, P., GRYZ, J., AND LIANG, D. 2003. Skyline with presorting. In *Proceedings of the IEEE International Conference on Data Engineering*. Poster. IEEE Computer Society Press, Los Alamitos, Calif.

CREIGNOU, N., HERMANN, M., AND PICHLER, R. 2001. Complexity of Constraint Solving Problems. In *Proceedings of the International Conferences on Constraint Programming and Logic Programming*. Tutorial Notes.

CRISTANI, M. 2002. Many-sorted preference relations. In *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning*.

DELGRANDE, J. P., SCHAUB, T., AND TOMPITS, H. 2000. Logic programs with compiled preferences. In *Proceedings of the European Conference on Artificial Intelligence*.

EISENBERG, A. AND MELTON, J. 1999. SQL:1999, formerly known as SQL3. *ACM SIGMOD Record 28*, 1, 131–138.

FISHBURN, P. C. 1970. *Utility Theory for Decision Making*. Wiley, New York.

FISHBURN, P. C. 1999. Preference structures and their numerical representations. *Theoret. Comput. Sci. 217*, 359–383.

GIANNOTTI, F., GRECO, S., SACCA, D., AND ZANIOLO, C. 1997. Programming with nondeterminism in deductive databases. *Ann. Math. Artif. Intel. 19*, 3–4.

GOVINDARAJAN, K., JAYARAMAN, B., AND MANTHA, S. 1995. Preference logic programming. In *Proceedings of the International Conference on Logic Programming*. 731–745.

GOVINDARAJAN, K., JAYARAMAN, B., AND MANTHA, S. 2001. Preference queries in deductive databases. *New Gen. Comput.*, 57–86.

GUO, S., SUN, W., AND WEISS, M. 1996. Solving satisfiability and implication problems in database systems. *ACM Trans. Datab. Syst. 21*, 2, 270–293.

HANSSON, S. O. 2001. Preference logic. In *Handbook of Philosophical Logic*, D. Gabbay, Ed. Vol. 8.

HRISTIDIS, V., KOUDAS, N., AND PAPAKONSTANTINOU, Y. 2001. PREFER: A system for the efficient execution of multiparametric ranked queries. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. ACM, New York, 259–270.

HUET, G. 1976. Ph.D. dissertation. Univ. de Paris VII, Paris, France.

HUGHES, R. 1980. Rationality and intransitive preferences. *Analysis 40*, 132–134.

KANELLAKIS, P. C., KUPER, G. M., AND REVESZ, P. Z. 1995. Constraint query languages. *J. Comput. Syst. Sci. 51*, 1 (Aug.), 26–52.

KIEßLING, W. 2002. Foundations of preferences in database systems. In *Proceedings of the International Conference on Very Large Data Bases*.

KIEßLING, W. AND GÜNTZER, U. 1994. Database reasoning—A deductive framework for solving large and complex problems by means of subsumption. In *Proceedings of the 3rd Workshop on*

Information Systems and Artificial Intelligence. Lecture Notes in Computer Science, vol. 777, Springer-Verlag, New York, 118–138.

Kießling, W. and Hafenrichter, B. 2002. Optimizing preference queries for personalized web services. In *Proceedings of the IASTED International Conference on Communications, Internet and Information Technology*. Also Tech. Rep. 2002-12, July 2002, Institute of Computer Science, University of Augsburg, Germany.

Kießling, W. and Hafenrichter, B. 2003. Algebraic optimization of relational preference queries. Tech. Rep. 2003-1, Institut für Informatik, Universität Augsburg.

Kießling, W., Hafenrichter, B., Fischer, S., and Holland, S. 2001. Preference XPATH—A query language for E-commerce. In *Proceedings of the 5th International Conference Wirtschaftsinformatik*. Augsburg, Germany, 43–62.

Kießling, W. and Köstler, G. 2002. Preference SQL - Design, implementation, experience. In *Proceedings of the International Conference on Very Large Data Bases*.

Knight, K. 1989. Unification: A multidisciplinary survey. *ACM Comput. Surv. 21*, 1, 93–124.

Kossmann, D., Ramsak, F., and Rost, S. 2002. Shooting stars in the sky: An online algorithm for skyline queries. In *Proceedings of the International Conference on Very Large Data Bases*.

Köstler, G., Kießling, W., Thöne, H., and Güntzer, U. 1995. Fixpoint iteration with subsumption in deductive databases. *J. Intel. Inf. Syst. 4*, 123–148.

Kuper, G., Libkin, L., and Paredaens, J., Eds. 2000. *Constraint Databases*. Springer-Verlag.

Lacroix, M. and Lavency, P. 1987. Preferences: Putting More Knowledge Into Queries. In *Proceedings of the International Conference on Very Large Data Bases*. 217–225.

Mantha, S. M. 1991. First-order preference theories and their applications. Ph.D. thesis, University of Utah.

Papadias, D., Tao, Y., Fu, G., and Seeger:, B. 2003. An optimal and progressive algorithm for skyline queries. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. ACM, New York, 467–478.

Sakama, C. and Inoue, K. 2000. Prioritized logic programming and its application to commonsense reasoning. *Artif. Intel. 123*, 185–222.

Stolze, M. 2000. Soft navigation in product catalogs. *Int. J. Digital Lib. 3*, 1, 60–66.

Tan, S.-W. and Pearl, J. 1994. Specification and evaluation of preferences under uncertainty. In *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning*.

Torlone, R. and Ciaccia, P. 2002. Which are my preferred items? In *Proceedings of the Workshop on Recommendation and Personalization in E-Commerce*.

Torlone, R. and Ciaccia, P. 2003. Management of user preferences in data intensive applications. In *Proceedings of the 11th Italian Symposium on Advanced Database Systems (SEBD)*.

Vardi, M. Y. 1982. The complexity of relational query languages. In *Proceedings of the ACM Symposium on Theory of Computing*. ACM, New York, 137–146.

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

Wellman, M. P. and Doyle, J. 1991. Preferential semantics for goals. In *Proceedings of the National Conference on Artificial Intelligence*. 698–703.

Wong, S. T. C. 1994. Preference-based decision making for cooperative knowledge-based systems. *ACM Trans. Inf. Syst. 12*, 4, 407–435.

Received September 2002; revised February 2003; accepted July 2003