Finding the Best when it's a Matter of Preference

Riccardo Torlone[1] and Paolo Ciaccia[2]

[1] Dip. di Informatica e Automazione, Università Roma Tre
Via della Vasca Navale, 79 - 00146 Roma, Italy
torlone@dia.uniroma3.it
[2] DEIS — CSITE–CNR, Università di Bologna
Viale Risorgimento, 2 - 40136 Bologna, Italy
pciaccia@deis.unibo.it

Abstract. Dealing with user preferences is becoming a widespread issue in novel data-intensive application domains, such as electronic catalogs, e-commerce, multimedia databases, and real estates. Although some recent works have studied the problem under specific assumptions, an understanding of the issues involved in the more general case is still missing. In this paper we assume that user preferences are expressed in a *qualitative* way over the tuples of a relation schema (e.g., *I prefer product A to product B*), which is quite natural from the user point of view and also includes, as a proper sub-case, *quantitative* preferences defined by means of a scoring function. Starting from an analysis of basic properties of (qualitative) preferences, we consider the *Best* operator, which can be used to smoothly embed preferences in queries. We study general properties of this operator and present a practical algorithm for its computation. We also describe a special data structure, called β-tree, that can be used for a rapid evaluation of the Best operator.

1 Introduction

Several modern database application areas nowadays are faced with the problems of managing user preferences. A remarkable example arises in e-commerce Internet sites (marketplaces, electronic shops, and others), which provide information on thousands, even millions, of products. Effectively supporting user search and browsing over such large repositories entails the problem of properly understanding user needs, filtering out unrelevant items, helping the user to formulate the most appropriate queries, and presenting results ranked according to their presumed relevance. Similar issues arise in multimedia databases, when the user looks for objects similar to a given one. In this case, the problem is that the notion of similarity can be a subjective one, thus the system has to "learn" it and then to exploit the acquired knowledge about user preferences to retrieve the most relevant objects [3].

In the database field, the problem of expressing and managing user preferences has received growing attention in the last few years [1, 4, 6, 8, 9, 11–15].

PREFER [12] is a system for the evaluation of multi-parametric ranked queries, which can also exploit the presence of materialized views to speed-up the query evaluation phase. In this approach preferences are expressed *quantitatively* by defining a scoring function that is a weighted linear combination of attributes' values (which have therefore to be numeric). Since the scoring function associates each tuple with a numeric score, tuple t_1 is preferred to t_2 if the score of t_1 is higher than the score of t_2. A more general class of scoring functions is considered by Fagin [8]; however, he only considers the problem of how to integrate in a middleware system rankings obtained from independent systems and ignores the issues arising in the definition and computation of such rankings. Fagin and Wimmers [9] study how to extend scoring functions in a principled way so as to attach user preferences, in the forms of *weights*, to each component score. This approach has also been adopted in [7] to define a fuzzy relational algebra with user preferences. Quantitative preferences are also considered in [1], where the emphasis is on how multiple "preference functions", possibly over different schemas, can be combined within an extensible framework.

Since the expressive power of quantitative preferences is well-known to be limited [10], *qualitative* preferences have also recently been considered from database researchers. Rather than relying on the existence of a scoring function, which indeed amounts to define a linear order over tuples, with qualitative preferences one just assumes that tuples can be compared using some computable function. For instance, the *skyline* operator [4] returns all the tuples which are not *dominated* by any other tuple, where t_1 dominates t_2 if t_1 is at least as good as t_2 in all dimensions and better than t_2 in at least one dimension. For instance, hotel t_1 might be considered to dominate hotel t_2 if t_1 is cheaper than t_2 and both have the same rating. In [4] and [15] several algorithms to evaluate a skyline are considered, many of which rely on the presence of a linear order along each of the dimensions (e.g., hotel price). Along this line, Chomicki [6] has recently considered the case where formulas used to compare the tuple dimensions are restricted to use only built-in predicates, and predicates can be freely combined with logical connectives. He defines a *winnow* operator and mostly concentrates on the properties of the preference language, thus ignoring query processing issues.

In this paper we consider qualitative preferences and, for the sake of generality, disregard issues related to *how* (that is, in which language) preferences are expressed. This means that we do not rely on any linear order on "dimensions" or similar properties. With this in mind, we first argument about some basic properties of qualitative preferences and clarify some issues which are relevant for query evaluation. To this purpose, we consider the *Best* operator, discuss its properties, and then turn to investigate how Best queries can be evaluated. In particular, unlike previous works, we focus on the case where not only the "best" (i.e., highest ranked) tuples are to be computed, but possibly also tuples with a lower rank are to be returned. This leads us to develop a basic algorithm and to introduce the notion of *β-tree*, a data structure that can act as a materialized view for answering subsequent Best queries.

The paper is organized as follows. In Section 2 we introduce the basic notions and investigate some general issues about preferences. In particular, we precisely define the relationship between qualitative and qualitative preferences. In Section 3 we consider the Best operator and study its general properties. Then, in Section 4, we propose an algorithm for computing the Best operator, demonstrate its correctness and discuss possible improvements of this algorithm. We also introduce in this section the notion of β-tree. Finally, in Section 5, we list a set of practical and theoretical open issues.

2 Preferences in Relational Databases

In this section we introduce the basic notion of user preference and present some preliminary results. In order to simplify the notation, we will refer to the relational model but the various notions and results can be extended to more involved data modeds, like an XML-based or an object oriented one. The terminology is mainly inherited from [6].

2.1 Qualitative Preferences

As usual, we denote by $R(X)$ a *relation scheme*, where R is the *name* of the relation and $X = A_1, \ldots, A_k$ is a set of *attributes*. We also assume that each attribute $A \in X$ has associated a set of values D called the *domain* of A. A *tuple* t over a scheme $R(X)$ associates with each $A \in X$ a value taken from its domain. A *relation* r over a scheme $R(X)$ (also called an *instance* of $R(X)$) is a finite set of tuples over $R(X)$.

Preferences over a relational scheme $R(X)$ can be naturally expressed by a collection of pairs of tuples over $R(X)$: each pair specifies the preference of one tuple over another one. These are called *qualitative* preferences [6].

Definition 1 (Preference relation). *A* (qualitative) preference relation $\succ$ *over a relational scheme* $R(X)$ *is a binary relation over* $\prod_{i=1}^{k} D_i$. *We denote a single preference* $(t_1, t_2) \in \succ$ *by* $t_1 \succ t_2$ *and say that* t_1 is preferable *to* t_2.

Example 1. Let us consider the following relation.

	Make	Model	Color	Price
t_1	BMW	330	Black	30K
t_2	Ford	Escort	White	20K
t_3	Toyota	Corolla	Silver	15K
t_4	Ferrari	360	Red	100K

A set of qualitative preferences over it, possibly expressed by a potential customer, can be the following: $t_1 \succ t_2$, $t_3 \succ t_2$, $t_4 \succ t_1$ and $t_3 \succ t_1$.

A preference relation can be naturally represented by means of a directed graph $G_\succ$ that we call *preference graph*. In this graph, nodes correspond to tuples and there is an edge from a node t_1 to a node t_2 if $t_1 \succ t_2$. For instance,

the preference graph for Example 1 is reported in Figure 1. Several properties of preference relations can be expressed and studied in terms of properties of the corresponding graph and for this reason in the following we will sometime switch from a preference relation to its graph representation, and viceversa.

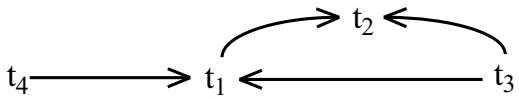

Fig. 1. A preference graph

Note that Definition 1 does not make any a priori hypothesis on the binary relation $\succ$. Usually, in many applications, $\succ$ is at least a (strict) partial order [6]. This means that $\succ$ is irreflexive (we never have $t \not\succ t$), asymmetric (we never have both $t_1 \succ t_2$ and $t_2 \succ t_1$), and transitive (if $t_1 \succ t_2$ and $t_2 \succ t_3$ then also $t_1 \succ t_3$).

Intuitively, while reflexivity does not make sense in this context, asymmetry prevents "inconsistencies" among preferences and transitivity yields a simple way to infer preferences. It has been observed however that, in some practical cases, both asymmetry and transitivity may fail to hold. For instance, when preferences are obtained by taking into account single properties of tuples. Consider as a simple example the following symmetric preferences: *I like Ferrari more than Toyota, but Toyota is preferable to Ferrari for its price*. Similarly, as an example of nontransitive preferences, consider the following: *I prefer the color of car* t_1 *to the color of car* t_2*, and the model of car* t_2 *to the model of car* t_3*, but I have no reason to prefer car* t_1 *to car* t_3. Since however symmetric and, more in general, *cyclic* preferences ($t_1 \succ t_2 \succ \ldots t_n \succ t_1$ for some finite n) lead to ambiguous situations, we also introduce a "stronger" notion of preference under which a tuple t_1 is preferable to a tuple t_2 only if the converse does not hold.

Definition 2 (Strong preference relation). *Given a qualitative preference relation* $\succ$ *over a relation scheme* $R(X)$*, we say that* t_1 *is* strongly preferable *to* t_2*, denoted by* $t_1 \succ\!\!\succ t_2$*, if and only if* $t_1 \succ t_2$ *and* $t_2 \not\succ t_1$.

Example 2. Consider the relation of Example 1 and assume to have $t_1 \succ t_2$, $t_3 \succ t_2$, $t_2 \succ t_4$ and $t_2 \succ t_1$. Then we have $t_3 \succ\!\!\succ t_2$ and $t_2 \succ\!\!\succ t_4$ but $t_1 \not\succ\!\!\succ t_2$ and $t_2 \not\succ\!\!\succ t_1$.

A strong preference relation can be viewed as a preference relation itself for which we have the following result.[3]

Lemma 1. *If* $\succ$ *is transitive then* $\succ\!\!\succ$ *is a partial order.*

Two other natural notions among tuples can be derived from the above definitions. The first one is called indifference since it holds between tuples for which an explicit preference does not exist.

[3] For lack of space all formal results are stated without proof.

Definition 3 (Indifference). *Given a qualitative preference relation $\succ$, an* indifference relation $\|$ *is defined as follows: $t_1 \| t_2$ if neither $t_1 \succ t_2$ nor $t_2 \succ t_1$; if $t_1 \| t_2$ we say that t_1* is indifferent to *t_2.*

We call the second notion likeness as it holds between tuples for which the given preferences do not allow us to establish that one is better than the other.

Definition 4 (Likeness). *Given a qualitative preference relation $\succ$, a* likeness relation $\approx$ *is defined as follows: $t_1 \approx t_2$ if both $t_1 \succ t_2$ and $t_2 \succ t_1$; if $t_1 \approx t_2$ we say that t_1* is like *t_2.*

Example 3. Consider again the relation of Example 1 and assume that now we have the following set of preferences: $t_1 \succ t_2$, $t_4 \succ t_1$, $t_3 \succ t_2$, $t_4 \succ t_2$ and $t_2 \succ t_1$. Then we have $t_1 \| t_3$, $t_3 \| t_4$ and $t_1 \approx t_2$.

It is interesting to note that even if $\succ$ is a partial order, the indifference relation is not necessarily transitive. Consider for instance the case in which we simply have $t_1 \succ t_2$ and $t_3 \succ t_4$, which is trivially a partial order. Then $t_1 \| t_3$, $t_3 \| t_2$ but t_1 and t_2 are not indifferent. In this case, it is probably inappropriate to call it indifference since it would actually represent incomparability among tuples. Conversely, it is easy to show the following.

Lemma 2. *If $\succ$ is transitive then $\approx$ is an equivalence relation, that is, it is a reflexive, symmetric and transitive relation.*

2.2 Combination of Preferences

The need of combining preferences arises in a variety of situations. For instance, two users might want to put together their preferences in order to choose an item that is suitable for both, a user might need to update his/her set of preferences, and so on. Since preferences are binary relations, set-theoretic operators can be immediately applied to them. Moreover, other interesting operators can be defined on them. For instance, when two preference relations $\succ_1$ and $\succ_2$ are both partial orders, one might be interested to derive from them a new preference relation $\succ$ that is still a partial order.

We consider a restricted set of "basic" preference combinations as follows.

Definition 5 (Preference combination). *A basic combination of preferences is one of the following operations.*

- Set operations*: union ($\succ_1 \cup \succ_2$), intersection ($\succ_1 \cap \succ_2$), and difference ($\succ_1 - \succ_2$) of two preference relations $\succ_1$ and $\succ_2$ over the same schema.*
- Inverse*: the inverse of a preference relation $\succ$ is denoted by $\succ^{-1}$ and defined as follows: $t_1 \succ^{-1} t_2$ iff $t_2 \succ t_1$;*
- Transitive closure*: the transitive closure of a preference relation $\succ$ is denoted by $\succ^*$ and is defined in the usual way.*

Starting from these basic operations on preference relations, many other interesting preference relations can be obtained. A notable example is the *priorized composition* [6] , which combines two preference relations by assigning a "higher priority" to one of them. More precisely, the priorized composition of two preference relations $\succ_1$ and $\succ_2$ is defined as follows: t_1 is preferable to t_2 if :(1) $t_1 \succ_1 t_2$ or (2) $t_2 \succ_2 t_1$ and $t_1 \not\succ_1 t_2$. It is easy to see that such a preference relation can be obtained by combining $\succ_1$ and $\succ_2$ as follows: $\succ_1 \cup(\succ_2 - \succ_1^{-1})$.

Finally, we note that a combination of preference relations may not preserve the properties of the original preference relations. With respect to the set operations we have that irreflexivity is always preserved, asymmetry is not preserved by union but is preserved by both intersection and difference, and transitivity is only preserved by intersection [6]. Moreover, it is easy to see that inversion preserves irreflexivity, asymmetry and transitivity whereas the transitive closure does not preserve irreflexivity and asymmetry, but "enforces" transitivity.

2.3 Quantitative Preferences

In this section we discuss about an alternative approach in which the preference for a tuple t over a scheme is measured by a value, called *score*, that specifies (in absolute terms) "how much"we prefer t [1, 5, 10, 12]. According to [6], we call such preferences *quantitative*. Without loss of generality, we can assume that a score is chosen in the set $\mathcal{N}$ of natural numbers.

Definition 6. *A* quantitative preference p *over a relation scheme* $R(X)$ *is a partial function* $p : \prod_{i=1}^{k} D_i \rightarrow \mathcal{N}$. *If* p *is defined on a tuple* t *over* $R(X)$, *we say that* $p(t)$ *is the* score *of* t.

If p is not defined on a tuple t over $R(X)$ then t is *indifferent* with respect to p. A score equal to zero can be used to specify special cases, for instance, the presence of a *veto* over some tuple [1].

Example 4. A possible quantitative preference for the relation in Example 1 may assign the following scores: $p(t_1) = 2$, $p(t_2) = 1$ and $p(t_4) = 2$.

It is easy to see that, given a quantitative preference p over a relation scheme $R(X)$ and an instance r of $R(X)$, we can always define a qualitative preference relation over $R(X)$ such that $t_1 \succ t_2 \Leftrightarrow p(t_1) > p(t_2)$ for all $t_1, t_2 \in r$. However, the converse is not true in general. The following theorem follows by a fundamental result of decision theory [10] and establishes precisely the conditions under which it possible to represent qualitative preferences by means of quantitative ones. Interestingly, it turns out that partial order is not a sufficient condition for such a representability, but if a quantitative representation exists, it is indeed unique. Given a quantitative representation p of a qualitative preference relation $\succ$ ($p(t_1) > p(t_2) \Leftrightarrow t_1 \succ t_2$ for all t_1, t_2 of an instance r), we say that p is *unique up to an ordinal transformation* if for any other quantitative representation p' of $\succ$ we have that $p(t_1) > p(t_2) \Leftrightarrow p'(t_1) > p'(t_2)$.

Theorem 1. *There is a quantitative representation p of a qualitative preference relation $\succ$ over a scheme $R(X)$ for every instance of $R(X)$ if and only if $\succ$ is a partial order for which $\|$ is transitive. In this case, p is unique up to an ordinal transformation.*

The following interesting corollary states that transitivity is a sufficient condition for the representation of strong qualitative preferences.

Corollary 1. *There is a quantitative representation p of a strong qualitative preference relation $\succ\!\!\succ$ over $R(X)$ for every instance of $R(X)$ if and only if both $\succ$ and $\|$ are transitive.*

The above results clearly show that qualitative preferences are more expressive than quantitative ones. Therefore, in the following we will refer to qualitative preferences only.

3 Querying with Preferences

In this section we illustrate and investigate an operator that, combined with the standard operators of relational algebra, can be used to specify queries over a database with preferences. This operator is called *Best* (denoted by $\beta^n_\succ$) and selects the $n-$th ranked set of tuples in terms of preference (for $n = 1$ we obtain the best tuples in absolute terms). Like the other relational algebra operators, it returns a relation as a result.

3.1 The Best Operator

The definition of the Best operator with respect to a preference relation $\succ$ is quite natural: $\beta^1_\succ(r)$ returns all the tuples t of a relation r for which there is no tuple in r better than t according to $\succ$. If the user is not satisfied from the basic result, this operation can be applied iteratively: at each step it returns the best tuples of r, excluding the tuples retrieved in previous steps. This operator is called *winnow* in [6].

Definition 7 (Best operator). *Let r be a relation over a scheme $R(X)$ and let $\succ$ be a qualitative preference relation over $R(X)$. The* Best operator $\beta^n_\succ$ of rank $n > 0$ *is defined as follows:*

- $\beta^1_\succ(r) = \{t \in r \mid \nexists t' \in r, t' \succ t\}$
- $\beta^{n+1}_\succ = \beta^1_\succ(r - \bigcup_{i=1}^n \beta^i_\succ(r))$

Example 5. Consider the preferences of Example 1: $t_1 \succ t_2$, $t_3 \succ t_2$, $t_4 \succ t_1$ and $t_3 \succ t_1$. Then we have $\beta^1_\succ(r) = \{t_4, t_3\}$, $\beta^2_\succ(r) = \{t_1\}$ and $\beta^3_\succ(r) = \{t_2\}$.

Used together with the other operators of relational algebra, the Best operator can be profitably used in queries where preferences are taken into account.

Example 6. Let us consider again Example 1. The Make and the Model of the best cars according to the given preferences whose price is less than $50K$ can be obtained by the following expression: $\pi_{Make,Model}(\beta^1_\succ(\sigma_{Price<50K}(r)))$.

3.2 General Properties of the Best Operator

We now consider some interesting properties of the Best operator. The first one establishes that the Best operator is insensitive on the transitivity of $\succ$.

Theorem 2. *Let $\succ$ be a preference relation over a scheme $R(X)$, then for each instance r of $R(X)$ and each $i > 0$, $\beta^i_{\succ}(r) = \beta^i_{\succ^*}(r)$.*

Another interesting point about the Best operator is the following: we expect that, sooner or later, every object of our database is retrieved by this operator. More precisely, let us denote by $\bar{\beta}^n_{\succ}(r)$ the set of tuples $\bigcup_{i=1}^{n} \beta^i_{\succ}(r)$, that is, the union of the first n applications of the Best operator to r. We introduce the following desirable property.

Definition 8 (Lossiness). *Let $\succ$ be a qualitative preference relation over a scheme $R(X)$. The Best operator $\beta^n_{\succ}$ is* lossless *if there is an integer $k > 0$ such that $\bar{\beta}^{k+1}_{\succ}(r) = \bar{\beta}^k_{\succ}(r) = r$ for every instance r of $R(X)$.*

Put in other words, the lossiness property guarantees that, for every instance r of $R(X)$ and every tuple $t \in r$, there is an integer $i > 0$ such that $t \in \beta^i_{\succ}(r)$. Unfortunately, this property does not hold in general since we could have $\bar{\beta}^{i+1}_{\succ}(r) = \bar{\beta}^i_{\succ}(r) \subset r$, for some $i > 0$, when two tuples, involved in a cyclic chain of preferences, prevent each other to appear in the result. We call i the *fixpoint* of the Best operator.

Example 7. Suppose we have: $t_1 \succ t_2$, $t_2 \succ t_3$, $t_3 \succ t_2$ and $t_3 \succ t_4$. Then we have $\beta^1_{\succ}(r) = \{t_1\}$ but no further tuple can be selected by the Best operator since t_2 and t_3 prevent each other to appear in the result of $\beta^2_{\succ}(r)$ and t_4 cannot be selected since t_3 is preferable to it.

We can however state the following results concerning lossiness.

Lemma 3. *Let $\succ$ be a qualitative preference relation.*

1. *If $\succ$ is acyclic then $\beta^n_{\succ}$ is lossless.*
2. *If $\succ$ is transitive then $\beta^n_{\succ\succ}$ is lossless.*
3. *$\beta^n_{\succ\succ^*}$ is lossless.*

Example 8. Consider again the preferences of Example 7. We have that $t_1 \succ\succ t_2$, $t_2 \approx t_3$ and $t_3 \succ\succ t_4$ and therefore $\beta^1_{\succ\succ}(r) = \{t_1\}$, $\beta^2_{\succ\succ}(r) = \{t_2, t_3\}$ and $\beta^3_{\succ\succ}(r) = \{t_4\}$.

Other interesting properties of the Best operator can be defined and checked by considering the directed graph $G_{\succ}$ associated with the preference relation $\succ$. We recall that a source of a graph is a node with no incoming edges, a path is a sequence of nodes connected by edges, the length of a path composed by n nodes is $n - 1$. We have the following results.

Lemma 4. *Let $\succ$ be a preference relation and $G_{\succ}$ be its preference graph.*

1. $\beta^1_{\succ}(r)$ *returns the sources of* $G_{\succ}$.
2. *If* $G_{\succ}$ *is acyclic then, for every* $t \in r$, $t \in \beta^{k+1}_{\succ}(r)$ *where* k *is the length of the longest path from a source of* $G_{\succ}$ *to* t.
3. *The fixpoint of* $G_{\succ}$ *is equal to the length of the longest path from a source of* $G_{\succ}$ *to a node of* $G_{\succ}$ *involved in a cycle.*
4. *If* $G_{\succ}$ *is acyclic then the fixpoint of* $\beta^n_{\succ}(r)$ *is equal to the length of the longest path of* $G_{\succ}$.

We close this section by observing that a Best operator β_p can also be defined with respect to quantitative preference p in a similar way. It easily turns out that β_p is always lossless.

4 Implementation of the Best Operator

In this section we propose an algorithm for computing the Best operator, demonstrate its correctness, and discuss possible improvements of this algorithm. We then present a special data structure, called β-tree, that can be built while running the algorithm and can be later used to compute fast the Best operator.

Initially, we assume that the preference relation is a partial order. By Theorem 2, the assumption of transitivity does not imply a loss of generality, whereas the hypothesis of asymmetry (as well as irreflexivity) will be later relaxed.

4.1 Intrinsic Complexity

Let us start with a basic result that fixes a lower bound for the computation of the Best operator. The basic operation in this computation consists in verifying whether a tuple is preferable to another: we assume that this task requires constant time. Thus, the time computational complexity will be measured in terms of the number of such comparisons.

Theorem 3. *The computation of the Best operator requires* $O(n^2)$ *time in the worst case, where* n *is the cardinality of* r.

We point out that the above result is valid not only for the first application of the Best operator but for the whole computation, which may require several iterations, until a fixpoint is reached.

4.2 The Basic Algorithm

Although in some cases $O(n^2)$ comparisons are necessarily needed to completely determine the result of the Best operator, it is also true that the actual computational effort might strongly depend on (the structure of) the preference relation and on how an algorithm is able to exploit it. Although a nested-loops algorithm can be easily applied here [4], it is evident that it will perform poorly in the general case, since it will execute $O(n^2)$ comparisons regardless of the underlying preference relation.

In this section we present an algorithm that, unlike nested-loops, can compare tuples multiple times, but it can achieve much faster response times because of the way it organizes its search space.

Let us consider a relation r and a preference relation $\succ$ that is a partial order. The algorithm for the evaluation of $\beta^i_\succ(r)$ is composed by a number of *phases*, one for each iteration of the Best operator. In turn, each phase consists of a set of one or more *scans* over a set C_i of *candidate tuples* which might belong to the output Out_i of the $i-$th phase. At the end of each scan only one tuple is selected and inserted into Out_i.

First phase In the first phase we set $C_1 = r$. Each tuple of C_1 is extracted and processed, in any order but one at time. Let t be first tuple extracted from C_1 in the first scan, which temporarily plays the role of *selected* tuple. We compare t with another tuple t' extracted from C_1. Three cases are possible:

1. $t \| t'$: in this case we just put t' in a set U_1 of *unresolved* tuples and t remains the selected tuple;
2. $t \succ t'$: in this case we put t' in a set $D^t_\succ$, which contains the tuples dominated by t according to $\succ$, and t remains the selected tuple; if t' belongs to another set $D^{t''}_\succ$ it is deleted from it;
3. $t' \succ t$: in this case t' becomes the selected tuple and t is added to the set $D^{t'}_\succ$, which contains the tuples dominated by t' according to $\succ$; if t belongs to another set $D^{t''}_\succ$, it is deleted from it.

The algorithm proceeds by processing in this way all the tuples in C_1.

Let t be the selected tuple we get at the end (that is, when $C_1 = \emptyset$). We can show that there is no tuple among all the processed ones that dominates t and so $t \in \beta^1_\succ(r)$. However, there might be some tuple t' in U_1 dominated by t that has not been compared with t. For this reason, we also compare t with all the tuples in U_1 for which this comparison has no been done yet: if a tuple t'' in U_i is actually dominated by t, we put t'' in the set $D^t_\succ$ and delete it from U_i. At the end of this job, we finally put t in the set Out_1, in which we collect the tuples to be returned as output of the first phase. This completes the first scan.

At this point, if U_1 is not empty, we copy in C_1 all the tuples in U_1, we empty U_1, we select a tuple in C_1 and repeat the whole procedure, i.e. execute another scan at the end of which another tuple will be selected and inserted in Out_1. When, at the end of a scan, we obtain $U_1 = \emptyset$ the first phase is concluded.

Following phases In the second phase, we put in C_2 only the tuples in the sets $D^t_\succ$, for each $t \in Out_1$. Note that this strongly reduces the search space. We then select a tuple from C_2 and proceed as in the first phase. As before, the second phase terminates when the set U_2 is empty.

The subsequent phases proceed similarly. Each phase starts by inserting in C_i only the tuples in the sets $D^t_\succ$, for each $t \in Out_{i-1}$.

We have the following result.

Theorem 4. *The basic algorithm correctly computes, for all i, the result of $\beta^i_\succ$ over a relation r of cardinality n and requires $O(n^2)$ time in the worst case.*

We point out that while the proposed algorithm requires no more than $O(n^2)$ comparisons in the worst case, it exhibits a very good performance in the general case, since the set of candidate tuples is strongly reduced in each phase.

Example 9. Let $r = \{t_1, t_2, t_3, t_4, t_5\}$ and assume we have the following preferences: $t_1 \succ t_2$, $t_4 \succ t_1$, $t_4 \succ t_2$, $t_4 \succ t_3$, $t_5 \succ t_2$ and $t_5 \succ t_3$.

Initially, we set $C_1 = r$. Let t_1 be the first tuple extracted from C_1 and so the initial selected tuple. If we now extract t_2, t_1 remains the selected tuple as $t_1 \succ t_2$ and we set $D_{\succ}^{t_1} = \{t_2\}$. Assume we now extract t_3: since t_1 and t_3 are indifferent, t_1 remains again the selected tuple and we set $U_1 = \{t_3\}$. If we then extract t_4, this tuple becomes the selected tuple as $t_4 \succ t_1$ and we set $D_{\succ}^{t_4} = \{t_1\}$. Finally, we extract t_5 from C_1 that is indifferent to t_4 and so we set $U_1 = \{t_3, t_5\}$. We now need to compare t_4 with t_3, which is in U_1 and has not been compared with it. Since $t_4 \succ t_3$, we add t_3 to $D_{\succ}^{t_4}$, we delete it from U_1, and set $Out_1 = \{t_4\}$. We now repeat the procedure by setting $C_1 = U_1 = \{t_5\}$. Since now C_1 is a singleton, we just set $Out_1 = \{t_4, t_5\}$ and since $U_1 = \emptyset$, this conclude the first phase. In the second phase we put in C_2 only the tuples in $D_{\succ}^{t_4}$, that is: t_1 and t_3, and proceed similarly. Since t_1 and t_3 are indifferent, we get $Out_2 = \{t_1, t_3\}$ and also the second phase is concluded. We then have $C_3 = \{t_2\}$ and so we immediately obtain $Out_3 = \{t_2\}$. The algorithm terminates here as $D_{\succ}^{t_2} = \emptyset$.

4.3 Variants of the Algorithm

In this section we describe some possible modifications of the basic algorithm for the purpose of taking into account more general cases and of improving its efficiency.

Let us first consider the case in which $\succ$ is neither asymmetric nor irreflexive. Point 2 of Lemma 3 suggests us that the most effective way to deal with this case is to resort to the $\succ\!\!\succ$ relation, under which two tuples t_1 and t_2 such that $t_1 \approx t_2$ are returned in the same iteration of the Best operator (see Example 8).

The algorithm can be modified accordingly: at each iteration of the i-th phase, we keep a *set* of selected tuples S_i instead of just one tuple. When a tuple t' is extracted from the set C_i, we compare t' with one tuple t in S_i. Then, besides the three cases considered in the basic algorithm, we can also have $t \approx t'$. In this case, we just add t' to S_i. In the other cases, we proceed as in the basic version of the algorithm, with the only difference that if $t' \succ t$, we set $D_{\succ}^{t'} = D_{\succ}^{t'} \cup S_i$ and $S_i = \{t'\}$. Note that, as before, only one comparison is needed at each iteration with a representative in S_i. This is because, by Lemma 2, $\approx$ is an equivalence relation.

We can claim the following.

Lemma 5. *The modified algorithm correctly computes, for all i, the result of $\beta^i_{\succ\!\!\succ}$ over a relation r of cardinality n and requires $O(n^2)$ time in the worst case.*

Several improvements to the basic algorithm can be done to speed up the computation of the Best operator. We now briefly sketch two of them, with reference to the basic version of the algorithm.

First of all, we note that a crucial operation of the algorithm consists in the extraction of a tuple from the set C_i of candidates. Then, performance can be improved with an appropriate choice of such a tuple based on the preference relation itself. Differently from what it might appear at first sight, it turns out that is convenient to extract tuples form C_i in reversed order of preference. In fact, in this way, the selected tuple changes frequently and, as a consequence, dominated tuples are distributed among several sets $D^t_\succ$. It follows that the cardinality of such sets is minimized. Since the set of candidate tuples is obtained at each phase as union of the sets $D^t_\succ$, for each t in the output of the previous phase, this in turn implies a reduction of the search space.

Another improvement can be obtained by employing special properties of relation $\|$. For instance, when $\|$ is an equivalence relation. In this case, if at a certain iteration of the i-th phase t is the selected tuple and we find $t' \succ t$, t' becomes the selected tuple and we can set $D^{t'}_\succ = D^{t'}_\succ \cup \{t\} \cup U_i$ and empty U_i. We can show that with this change the correctness of the algorithm is preserved and the number of comparisons is generally reduced.

4.4 β-trees

While executing the basic algorithm, a special data structure can be built over the tuples of relation r. We will show that this structure, which we call β-tree, can be profitably used to efficiently compute, for each i, $\beta^i_\succ(r)$. As above, we initially assume that $\succ$ is a partial order.

Definition 9 (β-tree). *A β-tree T_r over a relation r is defined as follows:*

- *T_r has a node t for each $t \in r$ plus a special node $t_\top$;*
- *there is an edge from t_1 and t_2 in T_r if, at the end of the basic algorithm, $t_2 \in D^{t_1}_\succ$;*
- *there is an edge from $t_\top$ to t in T_r if t has no other incoming edge.*

We first note that T_r is indeed a tree having $t_\top$ as root. This easily follows from the fact that, by construction, the sets $D^{t_1}_\succ$ are pairwise disjoint. It should also pointed out that the size of a β-tree can be much less of that of the $\succ$ preference relation, although it contains all the information needed for the computation of the Best operator.

Example 10. A β-tree for the preferences of Example 9 is reported in Figure 2. Note that it represents only partially the given preference relation.

We recall that the *level* of a node n in a tree T is the length of the (unique) path from the root of T to n. Let $L_i(T)$ denote the set of nodes of T at level i (hence, $L_0(T)$ only contains the root of T). The following result establishes the relationship between the topology of T_r and the result of $\beta^i_\succ(r)$.

Theorem 5. *For each relation r, $\beta^i_\succ(r) = L_i(T_r)$.*

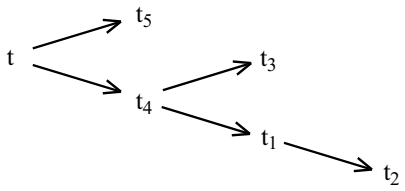

Fig. 2. The β-tree for the preferences of Example 9

The above result suggests a simple way to iteratively compute the Best operator over r from T_r: (1) $\beta^1_\succ(r)$ coincides with the children of $t_\top$ in T_r, and (2) $\beta^i_\succ(r)$ coincides with the children of the tuples returned by $\beta^{i-1}_\succ(r)$ in T_r.

This indicates that, given a relation r and a preference relation $\succ$ over it, it is very useful to pre-compute and store the corresponding β-tree, since it can then be used to efficiently answer queries involving the Best operator over $\succ$.

We conclude the section by mentioning that it is possible to extend the notion of β-tree for considering cyclic (but transitive) preference relations. In such an extension, the nodes of a β-tree represents *sets* of tuples rather than just single tuples. Each set constitutes an equivalence class according to relation $\approx$ (see Lemma 2). It is possible to show that this variant preserves the property specified in Theorem 5.

5 Final Discussion

In this paper we have considered the problem of querying database relations when qualitative preferences are defined over them. Starting from an analysis of basic properties of (qualitative) preferences, we have introduced and studied the *Best* operator, which can be used to embody preferences in queries. Although the computation of this operator has an inherent quadratic complexity, much better results can be obtained depending on the structure of preferences. To this end, since the only available algorithm for qualitative preferences was based on a nested-loops evaluation style, we have introduced an algorithm that tries to limit the number of comparisons among tuples by reducing the "search space" at each iteration. Starting from this algorithm, a number of open problems remain to be solved. We list only a few of them.

- Devising a disk-based algorithm for computing the Best operator on large relations appears to be a challenging problem. For the efficient management of the $D^t_\succ$ sets of dominated tuples a paged index could be considered. Note that this index will eventually converge to the β-tree.
- From a theoretical point of view it would be useful to identify specific patterns of preferences and characterize the algorithm complexity accordingly.
- Devising a specific strategy for performing an effective scan over the set of candidate tuples is also a relevant issue. This seems to be tightly related to the theoretical problem of the minimum number of comparisons needed to infer a partial order.

– A further challenging problem is the dynamic maintenance of β-trees in front of tuples and/or preferences updates. The β-tree is (much) more informative than just keeping the ranks of the tuples, yet it only stores a subset of the preference graph $G_{\succ}$ strictly needed for querying. This saves storage space, yet it makes harder to properly manage updates. We plan to investigate under which circumstances a "local recomputation" of the β-tree is possible, which is also of interest to evaluate the Best operator when preference relations are combined together.

References

1. R. Agrawal and E. L. Wimmers. A Framework for Expressing and Combining Preferences. In *Proc. of the 2000 ACM SIGMOD International Conference on Management of Data, Dallas, USA*, pp. 297–306, 2000.
2. P. Atzeni and V. De Antonellis. Relational Database Theory. *Benjamin/Cummings*, 1995.
3. I. Bartolini, P. Ciaccia, and F. Waas. FeedbackBypass: A New Approach to Interactive Similarity Query Processing. In *Proc. of 27th Intl. Conf. on VLDB, Roma, Italy*, pp. 201–210, 2001.
4. S. Börzsönyi, D. Kossmann, and K. Stocker. The Skyline Operator. In *Proc. 17th Intl. Conf. on Data Engineering, Heidelberg, Germany*, pp. 421–430, 2001.
5. M. Cherniack, M. J. Franklin, and S. B. Zdonik. Data Management for Pervasive Computing. In *Tutorial of 27th Intl. Conf. on VLDB, Roma, Italy*, 2001.
6. J. Chomicki. Querying with Intrinsic Preferences. In *Proc. 8th International Conf. on Extending Database Technology (EDBT'02), Prague, Czech Republic.*, 2002.
7. P. Ciaccia, D. Montesi, W. Penzo, and A. Trombetta. Imprecision and User Preferences in Multimedia Queries: A Generic Algebraic Approach. In *Proc. 1st Int. Symp. on Foundations of Information and Knowledge Systems (FoIKS 2000), Burg (Spreewald), Germany*, pp. 50–71, 2000.
8. R. Fagin. Combining Fuzzy Information from Multiple Systems. In *Proc. 15th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems (PODS'96), Montreal, Canada*, pp. 216–226, 1996.
9. R. Fagin and E.L. Wimmers. Incorporating User Preferences in Multimedia Queries. In *Proc. 6th Intl. Conf. on Database Theory (ICDT'97), Delphi, Greece*, pp. 247–261, 1997.
10. P. C. Fishburn. Preference Structures and Their Numerical Representations. *Theoretical Computer Science*, 217(2): 359–383, 1999.
11. K. Govindarajan, B. Jayaraman, and S. Mantha. Preference Queries in Deductive Databases. *New Generation Computing*, 19(1): 57–86, 2001.
12. V. Hristidis, N. Koudas, and Y. Papakonstantinou. PREFER: A System for the Efficient Execution of Multi-parametric Ranked Queries. In *Proc. of the 2001 ACM SIGMOD International Conference on Management of Data, Santa Barbara, USA*, pages 259–269, 2001.
13. M. Lacroix, P. Lavency. Preferences: Putting More Knowledge into Queries. In *Proc. of 13th Intl. Conf. on VLDB, Brighton, England*, pp. 217–225, 1987.
14. W. Ng. An Extension of the Relational Data Model to Incorporate Ordered Domains. *ACM Transactions on Database Systems*, 26(3), 2001.
15. K. Tan, P. Eng, B. C. Ooi. Efficient Progressive Skyline Computation. In *Proc. 17th Intl. Conf. on Data Engineering, Heidelberg, Germany*, pages 301–310, 2001.