Evaluation of Conditional Preference Queries

Fabíola S.F. Pereira, Sandra de Amo

Universidade Federal de Uberlândia, Brazil
deamo@ufu.br
fabfernandes@comp.ufu.br

Abstract. The need for incorporating preference querying in database technology is a very important issue in a variety of applications ranging from e-commerce to personalized search engines. A lot of recent research work has been dedicated to this topic in the artificial intelligence and database fields. Several formalisms allowing preference reasoning and specification have been proposed in the AI domain. On the other hand, in the database field the interest has been focused mainly in extending standard SQL with preference facilities in order to provide personalized query answering. More precisely, the interest in the database context focuses on the notion of *top-k preference query* and on the development of efficient methods for evaluating these queries. A top-k preference query returns k data tuples which are the most preferred according to the user's preference hierarchy. Of course, top-k preference query answering is closely dependent on the particular *preference model* underlying the semantics of the operators responsible for selecting the *best* tuples. In this paper, we consider the *conditional preference queries (cp-queries)* where preferences are specified by a set of rules expressed in a logical formalism. We propose the algorithms BNL** and R-BNL** for evaluating the top-k cp-queries and implement them in the core of the PostgreSQL query processor. An extensive set of experiments demonstrates the efficiency of our method for computing the most preferred tuples according to the user's preferences. We also show the need to integrate the new ***Select-Best*** and ***SelectK-Best*** operators into a database system, rather than translating them into standard SQL queries.

Categories and Subject Descriptors: H. Information Systems [**H.2 Database Management**]: H.2.3 Languages—*Query Languages*

Keywords: conditional preferences, preference queries, query evaluation, top-k queries

1. INTRODUCTION

Recently, a lot of interest arose in the artificial intelligence and database communities concerning the topic of preference elicitation, modeling and reasoning. In fact, due to the huge amount of information users are faced up to daily, the development of formalisms for preference specification and reasoning as well as tools allowing to cope with real user preferences turn out to be essential tasks in AI and DB fields. A lot of work has been done in this area so far by researchers in both communities. From a AI point of view, the interest is centered mainly in developing powerful mechanisms for preference reasoning [Boutilier et al. 2004; Brafman et al. 2006; Wilson 2004]. On the other hand, database researchers are concerned specially in developing preference query languages with high declarative and expressive power for personalized database applications [Kießling and Köstler 2002; Chomicki 2003; Hristidis et al. 2001; Koutrika et al. 2006]. In this context, the interest is essentially focused in enhancing standard query languages with a new operator enabling to return the most preferred tuples according to a given preference ordering.

The *skyline queries* introduced in [Borzsonyi et al. 2001] and more generally, the *pareto preference queries* [Kießling and Köstler 2002; Preisinger and Kießling 2007; Preisinger et al. 2006] are examples of queries enhanced with a *Best* constructor returning the most preferred tuples. In [de Amo and Ribeiro 2009], we introduced the preference query language CPref-SQL allowing to express a larger

We thank to the Brazilian Research Agencies CNPq and CAPES for supporting this work.

class of user preferences, which we called *conditional preference queries*. Conditional preference queries (cp-queries for short) involve a more sophisticated notion of preference between tuples. Cp-queries differ from the *skyline queries* [Borzsonyi et al. 2001] and the more general *pareto queries* [Papadias et al. 2005; Yiu and Mamoulis 2007; Kießling and Köstler 2002] mainly on the following aspect: *local* preferences over attribute values are not *absolute*, that is, they depend on how other attributes are instantiated.

In [de Amo and Ribeiro 2009], a ***Select-Best*** operator has been proposed in this new database preference context as well as the algorithm BNL* to evaluate it. The main goal of this initial work was to introduce the new query language and a first method for evaluating it. No implementation of BNL* has been proposed then. In this paper we propose the algorithm BNL** for evaluating the ***Select-Best*** operator. As the former BNL* algorithm, BNL** follows a *Block Nested Looping* strategy, but differs from BNL* in the way it performs the *dominance test* between two tuples.

As it was pointed out by different authors [Yiu and Mamoulis 2007; Hristidis et al. 2001; Papadias et al. 2005], the size of the set of the most preferred tuples cannot be controlled by the user. Consequently, two drawbacks arise immediately: (1) The preference rules specified by the users may be too restrictive, and so the set of the most preferred tuples may contain very few tuples. It would be interesting to provide mechanisms allowing to relax the user requirements in order to produce the *second* most preferred tuples, and so on; (2) In some cases, the preference rules are too permissive, and so the set of the most preferred tuples may be as large as the entire dataset. Consequently the user will be faced up with a huge set of answers and will have to examine them in order to select those which he regards as more relevant. In order to save the user this burden, it would be useful to provide an operator which can produce a set of tuples which is reasonably interesting to the user and whose size he can control.

In [Hristidis et al. 2001], the *top-k queries* have been introduced in a particular preference context, where the tuples are ranked using a *score* function f, and a tuple t is said to be more preferred to a tuple t' if and only if $f(t) > f(t')$. So, the top-k queries return the k tuples with higher scores. This approach is not always feasible in practice, since it depends on the user's good will in providing the tuples' scores. In [Papadias et al. 2005], the *top-k dominating queries* have been introduced, in the context of *pareto queries*, where the preference relation between tuples depends on the intrinsic characteristic of the data (the attribute values of each tuple) instead of an external score function. The top-k dominating queries return the set of tuples dominating the largest number of tuples in the dataset. It is important to emphasize that top-k queries and top-k dominating queries are different concepts and so may return different answers.

In this paper, we introduce the top-k conditional preference queries (top-k cp-queries) and the ***SelectK-Best*** operator for evaluating these queries. ***SelectK-Best*** combines the functionalities of the Top-N operator of [Carey and Kossman 1997] with a ranking strategy based on a topological sort of the set of tuples, according to the preference hierarchy. The following motivating example aims at illustrating our notion of top-k conditional preference queries.

A Motivating Example

A top-k cp-query incorporates the usual *hard constraints* (declared inside the WHERE clause) as well as *soft constraints* specified by a set of preference rules (*cp-rules* for short). We present below a typical example of such a query.

Example 1.1 Let us suppose we are given a database relation *MOVIES* storing information about movies with attributes T (for Title), G (for Genre), D (for Director), Y (Year (decade)) and A (Actor). The following statements express my preferences about movies: (1) I prefer those movies produced in the 90's rather than those produced in the 80's, if both belongs to the same category (genre); (2) For Woody Allen's films (wa) of the same genre and decade, I prefer those staring the actress Charlotte Rampling (cr) than those staring Mia Farrow (mf); (3) For the comedies produced in the

80's I prefer those directed by Joel Cohen (jc) than those directed by Woody Allen (wa); (4) For the movies produced in the 80's I prefer dramas to comedies.

I would like to list 4 titles of the films which most fulfill my wishes among those stored in the database, but I don't want romances to figure in my list. The corresponding top-k cp-query is:

```
  CREATE PREFERENCES MyPrefs FROM MOVIES AS
      Y=90 > Y=80 [T,D,A] AND
      IF D=wa THEN A=cr > A=mf [T] AND
      IF G=comedy and Y=80 THEN D=jc > D=wa [T,A] AND
      IF Y=80 THEN G=drama > G=comedy [T]

SELECT T FROM MOVIES
WHERE G <> romance
ACCORDING TO PREFERENCES (4, MyPrefs)
```

In this query, the hard constraint is G <> *romance* and the soft constraints are given by the rules *MyPrefs*. Note that the parameter $K = 4$ also appear inside the clause ACCORDING TO PREFERENCES, since it is part of the soft constraint expressed by the preference rules. In general, if K is not specified the most preferred tuples will be returned. Notice that each rule r declared inside the CREATE PREFERENCES statement is followed by a list of attributes within square brackets. This declaration establishes a necessary condition for two tuples to be compared, namely: t_1 and t_2 can be compared using a rule r if $t_1[X] = t_2[X]$ for each attribute *not* appearing in the corresponding list inside square brackets. For the attributes outside brackets, the *ceteris paribus* (all else being equal) semantics of [Boutilier et al. 2004] is considered. Notice for instance that in the first rule, the attribute G does not appear between brackets. This means that in order to compare two tuples t_1 and t_2 they both must coincide in the attribute G (category), following the condition established in statement (1). Thus, the declaration of a set of attributes between brackets gives more flexibility to the preference order semantics.

The evaluation of the ***SelectK-Best*** involves the evaluation of the most preferred tuples first. This task is achieved by the ***Select-Best*** operator. The semantics of this operator is the same as the *winnow* operator of [Chomicki 2003], returning the tuples which are not dominated by others. It is well known that such an operator does not enhance the expressive power of SQL, since it can be expressed in relational algebra [Chomicki 2003], by means of a re-writing procedure responsible to translate preference queries into a standard SQL-compliant query, after proper creation of a temporary relation [Kießling and Köstler 2002]. So, a first approach to implement the cp-queries is simply by translating them into a SQL query.

On the other hand, an explicit definition of ***Select-Best*** enables us to separate preference issues from the other aspects of the query, allowing to design specific and efficient algorithms to implement this operator. Following this line, the most common approach for preference evaluation in database systems is the *on-top* approach [Chan et al. 2005a; 2005b; Preisinger and Kießling 2007; Preisinger et al. 2006; Lin et al. 2007; Yiu and Mamoulis 2007; 2009]. In this approach the preference operator is evaluated as a user-defined function and its evaluation is completely separated from the database. So, it does not interact with the other internal operations involved in the SQL query execution plans (selections, joins, projections). The advantage of this approach relies on its simplicity. Nevertheless, its efficiency is not guaranteed, since it does not interact with the internal execution plan and so, cannot be treated by the query optimizer.

A more efficient approach for preference evaluation is the *built-in* approach where the algorithms for preference evaluation are implemented in the *core* of the database system. This approach has been used in several recent works dedicated to the preference evaluation topic [Borzsonyi et al. 2001; I.F.Ilyas et al. 2004] and its efficiency over the *on-top* approach is undoubtedly proved. Indeed, as far as preference queries are concerned, the design of efficient evaluation methods are essential in real-world applications, where users are often not disposed to wait a longtime for an answer. For that reason, in this paper we chose to follow the *built-in* approach for evaluating the top-k cp-queries. We present the algorithms BNL** and R-BNL** (*Ranked* BNL**) to evaluate the ***Select-Best*** and ***SelectK-Best*** operators respectively as well as their implementations inside the PostgreSQL query processor.

Main Contributions. The main contributions of this paper can be summarized as follows: (1) The introduction of the top-k conditional preference queries (top-k cp-queries); (2) The design of the algorithms BNL** and R-BNL** for evaluating cp-queries and top-k cp-queries respectively; (3) The implementation of both algorithms in the core of the PostgreSQL database system (4) an exhaustive set of experiments comparing top-k cp-queries implemented according to our *built-in* approach and their standard SQL counterpart.

Organization of the Paper. This paper is organized as follows. In Section 2, we discuss some related work. In Section 3, we briefly present the logic formalism introduced in [Wilson 2004] for specifying and reasoning with conditional preferences which constitutes a necessary background for our work. We also present in this section the syntax and semantics of the core CPref-SQL algebra which extends the relational algebra with the ***Select-Best*** and ***SelectK-Best*** operators. Section 4, we present the algorithms BNL** and R-BNL** for evaluating the CPref-SQL operators ***Select-Best*** and ***SelectK-Best*** respectively. In Section 5, we present and discuss the experimental results, comparing our *built-in* approach for top-k cp-query evaluation and the approach using standard SQL queries. Finally, in Section 6 we conclude the paper and discuss our future work.

2. RELATED WORK

Preference Modeling and Reasoning. The research literature on preference modelling, reasoning and eliciting is extensive. The approach of CP-Nets [Boutilier et al. 2004] uses a very simple graphical model which captures users qualitative *conditional preference* over tuples, under a *ceteris paribus* semantics. The approach of TCP-Nets [Brafman et al. 2006] generalizes the CP-Nets by introducing the ability of expressing absolute and relative importance of object attributes. In this paper, we use the approach introduced in [Wilson 2004] for specifying soft constraints inside a CPref-SQL query. This approach uses a logical framework for expressing conditional preference statements. It generalizes the CP-Nets, by relaxing the *ceteris paribus* semantics and allowing more expressiveness, and is orthogonal to TCP-Nets.

Extensions of SQL with Preference Support. In the database area, the problem of enhancing well-known query languages with preference features has been tackled in several recent and important work in the area. A pioneer work concerns the *skyline operator* introduced in [Borzsonyi et al. 2001]. In [Chomicki 2003], a simple logical framework is proposed for expressing preferences by *preference formulae*. These formulae are incorporated into the Relational Algebra and into SQL, through the operator *winnow* parameterized by a preference formula. [Kießling and Köstler 2002] introduced Preference SQL which extends SQL with some built-in base preference constructors and a *pareto accumulation* and a *prioritized accumulation* constructors. The optimizer uses an efficient rewriting procedure which transforms preference queries into standard SQL queries.

The work presented in [Endres and Kießling 2006] is close to ours in the sense that it also proposes a *bridge* between the treatment of preferences in the AI and DB communities, by transforming the TCP-Net queries of [Brafman et al. 2006] into preference database queries. No algorithm for selecting the best tuples has been proposed neither in [Endres and Kießling 2006] neither in further work (to the best of our knowledge).

Algorithms for Preference Query Evaluation. The topic of preference query evaluation has been extensively studied in the literature, in the context of pareto and skyline queries. In [Borzsonyi et al.

2001], the basic BNL (block-nested loop) algorithm has been introduced for evaluating *skyline queries*. Other skyline algorithms have also been presented in this paper, using B-trees and R-trees indexing. In [Chomicki 2003], the algorithm SFS (sort-filter-skyline) has been proposed which outperforms the BNL algorithm for skyline queries evaluation. It explores the idea of sorting the tuples according to a score function. It is known that for any monotone score function f: Tuples $\rightarrow \mathbb{R}$, if $p \in$ Tuples maximizes f then p is a skyline point (most preferred). For more details on skyline with presorting, see [Chomicki et al. 2005].

In [Chan et al. 2005a], the algorithm BNL$^+$ for *pareto query* evaluation was introduced. In this more general setting, the domain of the attribute values are partially ordered [1]. In [Preisinger et al. 2006], the authors proposed the algorithm BNL^{++} for pareto query evaluation in a particular case, where the ordering over the attribute domains is a weak order[2]. This implies nice properties in the better-than graph, which are not verified in the general setting of the algorithm BNL$^+$ of [Chan et al. 2005a]. Due to these nice properties, BNL^{++} is able to identify some important pruning conditions in the better-than graph, which results in a far better performance when compared to the BNL and BNL$^+$ algorithms. Further improvements of the BNL algorithm, in the context of pareto queries evaluation, were achieved in [Chan et al. 2005b] (algorithms BBS$^+$, SDC and SDC$^+$).

The algorithm BNL* we proposed in [de Amo and Ribeiro 2009] for evaluating the ***Select-Best*** operator in the context of cp-queries follows the lines of the BNL algorithm and uses the technique introduced in [Agrawal et al. 1989] for transitive closure evaluation. This algorithm has not been implemented. The algorithm BNL** we propose in the present paper follows the lines of the BNL algorithm as well. In this second proposal, it is coupled with a Datalog Program responsible for effectuating the dominance test between two tuples.

Top-k Queries. The topic of introducing user control in order to bound the answer set has been treated in the pioneer work of [Carey and Kossman 1997], in the context of the Top N queries. A Top N query returns the first N tuples according to the ordering they appear in the database. In [Hristidis et al. 2001] the *top-k queries* have been introduced in a *quantitative* preference model setting, that is, where preference betweeen tuples is expressed by a score function defined over the dataset. The *top-k dominating queries* have been introduced in [Papadias et al. 2005] as an extension of the skyline queries of [Borzsonyi et al. 2001] which were originally designed to return the most preferred tuples, without any user control on the size of the result. A *top-k dominating query* returns the k tuples which dominated the maximum amount of tuples in the database. This concept is orthogonal to the skyline and pareto queries, as well as to our cp-queries. In [Papadias et al. 2005] the authors propose the algorithm BBS (Branch and Bound Skyline) based on nearest-neighbor search on multidimensional access methods for evaluating the top-k dominating queries. In [Yiu and Mamoulis 2007; 2009], a set of algorithms are introduced for top-k dominating queries which significantly outperforms the algorithm BBS of [Papadias et al. 2005].

3. CONDITIONAL PREFERENCE QUERIES

This section aims at presenting the CPref-SQL query language with support to conditional preferences. First of all, we introduce the preference model underlying CPref-SQL in Section 3.1, as presented in [de Amo and Ribeiro 2009]. After that, in Section 3.2 we introduce the preference algebra underlying the language, focusing on the new operator ***SelectK-Best***, that is proposed in this paper.

3.1 The Preference Model

We start by briefly presenting the main concepts and results of [Wilson 2004] concerning a logical formalism for *specifying and reasoning* with preferences. Let $R(A_1, A_2, ..., A_n)$ be a relational schema.

[1]In the more specific case of the skyline queries, the domain of the attribute values are totally ordered.
[2]A strict partial order $\succ$ over a set C is called a *weak order* if it satisfies the *antitransitivity* property: for all $a, b, c \in C$, if $a \not\succ b$ and $b \not\succ c$ then $a \not\succ c$.

For each attribute $A \in R$, let $\mathbf{dom}(A)$ be the finite set of values of A (the domain of A). The set Tup(R)= $\mathbf{dom}(A_1) \times \mathbf{dom}(A_2) \times ... \times \mathbf{dom}(A_n)$ is the set of all possible tuples over R.

Let $\mathcal{L}$ be a set of statements of the form φ: $u \rightarrow (A = a) > (A = a')[W]$, where u is a formula $(A_{i_1} = a_1) \wedge ... \wedge (A_{i_k} = a_k)$, with $A_{i_j} \in R - \{A\}$ and $a_j \in \mathbf{dom}(A_{i_j})$ for all $j \in \{1, ..., k\}$, $a, a' \in \mathbf{dom}(A)$, and $W \subseteq R - \{A, A_{i_1}, ..., A_{i_k}\}$. Such statements are called *conditional preference rules* or *cp-rules* for short. The formula u is called the *condition* of the cp-rule φ. The set of attributes appearing in u is denoted by *Attr(u)*. A *conditional preference theory* (*cp-theory*) over R is a finite set of statements of $\mathcal{L}$.

A conditional preference theory Γ over R induces a *preference ordering* over Tup(R) as follows:

Definition 3.1 (Preference order) Let $\varphi : u \rightarrow (A = a) > (A = a')[W] \in \Gamma$. Let $\mathbf{t} = (\mathbf{b}, \mathbf{c}, a, \mathbf{w})$ and **t'** $= (\mathbf{b}, \mathbf{c}, a', \mathbf{w'}) \in$ Tup(R), where $\mathbf{b}$ is a tuple over *Attr(u)* satisfying the formula u, $\mathbf{c}$ is a tuple over R − (*Attr(u)*$\cup W \cup \{A\}$), $\mathbf{w}$ and **w'** are tuples over W. We say that $\mathbf{t}$ is *preferred* to **t'** according to φ. The set of pairs of tuples ($\mathbf{t}$, **t'**) where $\mathbf{t}$ is preferred to **t'** according to φ is denoted by $>_\varphi$. We denote by $>_\Gamma$ the *transitive closure* of the binary relation $\bigcup_{\varphi \in \Gamma} >_\varphi$.

For each cp-rule φ the induced preference order $>_\varphi$ has a *ceteris paribus* ("everything else equal") semantics with respect to the attributes in $R - (\{A\} \cup W)$. That is, tuples having different values for attributes not in $R - (W \cup \{A\})$ are incomparable according to $>_\varphi$.

Example 3.1 Let *MOVIES* $= \{T, G, D, Y, A\}$ as in Example 1.1 and let us consider the instance of *MOVIES* described in Figure 1. For this relation, for instance, we have $\mathbf{dom}(G) = \{c$ (comedy), a (action), d (drama)$\}$, $\mathbf{dom}(D) = \{wa$ (Woody Allen),jc (Joel Cohen), jm (James Cameron)$\}$, $\mathbf{dom}(Y) = \{$'80', '90', '00'$\}$, $\mathbf{dom}(A) = \{mf$ (Mia Farrow), bp (Bill Paxton), cr (Charlotte Rampling), ...$\}$. The cp-theory $\Gamma = \{\varphi_1, \varphi_2, \varphi_3, \varphi_4\}$ expresses the user preferences of Example 1.1:

$$\begin{aligned}
&\varphi_1\text{: } (Y = 90) > (Y = 80)[\{T, D, A\}],\\
&\varphi_2\text{: } (D = wa) \rightarrow (A = cr) > (A = mf)[\{T\}],\\
&\varphi_3\text{: } (G = c) \wedge (Y = 80) \rightarrow (D = jc) > (D = wa)[\{T, A\}].\\
&\varphi_4\text{: } (Y = 80) \rightarrow (G = d) > (G = c)[\{T\}].
\end{aligned}$$

	TITLE	GENRE	YEARS	DIRECTOR	ACTOR
t_1	Titanic	drama	90	James Cameron	Bill Paxton
t_2	Stardust Memories	drama	80	Woody Allen	Charlotte Rampling
t_3	New York Stories	drama	80	Woody Allen	Mia Farrow
t_4	A Midsummer Night's Sex Comedy	comedy	80	Woody Allen	Mia Farrow
t_5	Crimewave	comedy	80	Joel Coen	Louise Lasser
t_6	Avatar	action	00	James Cameron	Zoe Saldana

Fig. 1. An instance of relation *MOVIES*

The order induced on *MOVIES* by the cp-theory Γ is illustrated by the *better-than-graph* of Figure 2(a). In this figure, an arrow from t to t' means that $t > t'$ (arrows obtained by transitive closure are not shown in the figure). Note for instance that $t_1 > t_2$ according to rule φ_1, $t_2 > t_3$ due to rule φ_2, and $t_1 > t_3$ by transitivity.

Conditional preference order versus Pareto preference order. From Definition 3.1 we can infer two important facts: (1) each individual rule φ induces an ordering over the tuples where *local* preferences over the attribute values of the tuples are not *absolute*, that is, they depend on how other attributes are instantiated; (2) in order to test if two tuples $t = (a_1, a_2, ..., a_n)$ and $t' = (b_1, b_2, ..., b_n)$ are comparable according to a cp-theory $\Gamma = \{\varphi_1, ..., \varphi_k\}$, say $t \succ_\Gamma t'$, we must verify if *there exists* a subset of $\Gamma' \subseteq \Gamma$, $\Gamma' = \{\psi_1, ..., \psi_m\}$, and tuples $s_1, ..., s_{m-1}$ such that $t >_{\psi_1} s_1, s_1 >_{\psi_2} s_2, \; ... \;, s_{m-1} >_{\psi_m} t'$

On the other hand, the pareto preference ordering $\rhd$ underlying the original skyline queries [Borzsonyi et al. 2001], their generalized forms of [Papadias et al. 2005; Yiu and Mamoulis 2007] and the pareto queries of [Chan et al. 2005a; 2005b; Kießling and Köstler 2002; Preisinger et al. 2006] is defined as

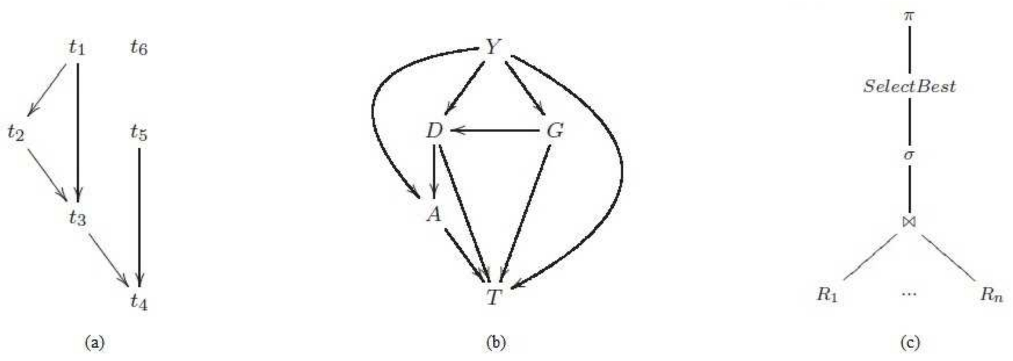

Fig. 2. (a) The better-than-graph for relation *MOVIES*. (b) The dependency graph of cp-theory Γ. (c) Execution plan for a CPref-SQL block.

follows: $t \rhd t'$ if there exists $i \in \{1, ..., n\}$ such that $a_i > b_i$ and for all $j \neq i$ $a_j \not> b_j$. Notice that in this case, the *local* preferences over the attribute values of the tuples are *absolute*: we compare values of the *ith* attribute of t with the values of the corresponding *ith* attribute of t'.

Unlike pareto preference order, the preference order induced by a cp-theory is not necessarily a strict partial order. As discussed in point (2) above, the conditional preference order is obtained by taking the transitive closure of the union of the preference orders specified by each rule individually. The result may be inconsistent, that is, it can be inferred that "I prefer X better than X" ! In fact, $>_\Gamma$ may contain a pair $(\mathbf{t}, \mathbf{t})$, that is, $>_\Gamma$ does not necessarily satisfy irreflexibility. That is not the case with the skyline and pareto queries: The result of pareto accumulation of weak orders over the attribute domains is a strict partial order over the set of tuples (see [Chomicki 2003] for the proof). So, consistency issues have to be carefully taken into account when dealing with cp-queries, since irreflexivity of the preference ordering is not a desirable situation in a database context.

Consistency of a cp-theory. We say that a cp-theory Γ is *consistent* if and only if the induced order $>_\Gamma$ is irreflexive. In [Wilson 2004], a sufficient condition for ensuring consistency of a cp-theory is given. This condition involves testing the acyclicity of the *dependency graph* associated to the cp-theory and its *local consistency.* The dependency graph $G(\Gamma) = (V, E)$ associated to a cp-theory Γ is defined as follows: $V = \{A \mid A \text{ is an attribute appearing in } \Gamma\}$, $E = \{(A, B) \mid$ there exists a cp-rule $\varphi \in \Gamma$ such that A and B appear respectively in the left side and right side of φ or A and B appear respectively in the right side and inside the brackets of $\varphi\}$. For instance, the dependence graph associated to our running example 3.1 is depicted in Figure 2(b). The issue of reasoning with preferences (namely, the design of methods for testing rules consistency and methods for deducing other preference relations from a base set of preference rules) is out of the scope of this paper. A comprehensive study on this topic can be found in [Wilson 2004]. In this paper, we will suppose our cp-theories are consistent. We have implemented the consistency test proposed in [Wilson 2004] in order to guarantee that only consistent cp-rules may be specified by the CREATE PREFERENCES statement. For more details about the consistency test algorithm and its complexity see [Wilson 2004].

In what follows, for the sake of simplifying the notation, we will omit the symbol Γ in the preference ordering notation $>_\Gamma$, assuming that cp-theory Γ is known in the context.

3.2 The CPref-SQL Query Language

In this section we introduce an algebraic operator that selects from a given relation the set of the *top-k tuples* according to a given cp-theory Γ. We denote this operator by ***SelectK-Best***$_\Gamma(K, R)$ or simply ***SelectK-Best***(K, R) where Γ is known in the context. Intuitively, ***SelectK-Best***$_\Gamma(K, R)$ returns the set of K tuples having the minimum number of tuples above them in the preference hierarchy. Besides this new operator, our algebra includes also the ***Select-Best*** operator we introduced in [de Amo and

Ribeiro 2009], which returns the most preferred tuples (those which are not dominated by others).

In order to define rigorously the semantics of the ***SelectK-Best*** operator, we introduce the notion of *level* of a tuple t (denoted $l(t)$) according to cp-theory Γ.

Definition 3.2 (Levels) Let Γ be a cp-theory over the relational schema $R(A_1, ..., A_m)$, r a database instance over R and t a tuple $\in r$. The *level* of t (denoted $l(t)$) *according to* Γ is inductively defined as follows:

if there is no $t' \in r$ such that $t' > t$, we define $l(t) = 0$.
otherwise $l(t) = \max \{l(t') \mid t' > t\} + 1$ □

SelectK-Best(K, r) returns the set of K tuples with the smallest levels.

Example 3.2 Let us consider the relation *MOVIES* depicted in Figure 1 and the cp-theory Γ given in Example 3.1. The preference ordering enforced by this cp-theory over the tuples in *MOVIES* is depicted in the *better-than-graph* of Figure 2(a). We have: $l(t_1) = l(t_5) = l(t_6) = 0$, $l(t_2) = 1$, $l(t_3) = max\{l(t_2), l(t_1)\} + 1 = 2$, $l(t_4) = max\{l(t_1), l(t_2), l(t_3), l(t_5)\} + 1 = 3$.

SelectK-Best$(4, MOVIES) = \{t_1, t_5, t_6, t_2\}$, ***SelectK-Best***$(2, MOVIES) = \{t_1, t_5\}$. We remark that $l(t_5) = l(t_6) = 0$, but t_6 is not considered in the answer, since it comes after t_5 in the relation *MOVIES*.

Note that ***SelectK-Best***$(0, MOVIES) = \emptyset$ and ***Select-Best***$(MOVIES) = \{t_1, t_5, t_6\}$ which is the set of the most preferred tuples, those which are not dominated. □

It is easy to show that if $t > t'$ then $l(t) < l(t')$. The reverse implication does not hold[3].

The CPref-SQL query language is an extension of the standard SQL with the two new preference constructors ***Select-Best*** and ***SelectK-Best***. The simple query block of CPref-SQL is given below. Notice that hard and soft conditions are incorporated into a unique simple query block.

```
SELECT <attribute-list>
FROM <tables>
WHERE <where-conditions> (% hard conditions)
ACCORDING TO PREFERENCES [K] <list-of-cprules> (% soft conditions)
GROUP BY <attribute-list>
ORDER BY <attribute-list>
```

The parameter <list-of-cprules> inside the clause `ACCORDING TO PREFERENCES` is a list of cp-rules declared in the forme *IF ... THEN ...*, each rule separated by the boolean connective *AND*. These rules may be declared inside a `CREATE PREFERENCES` command (as illustrated in our running example 1.1). The canonical execution plan associated to a CPref-SQL block (without aggregate constructors) is shown in Figure 2(c). σ, π and $\bowtie$ denote the usual relational algebra operators *Selection, Projection*, and *Join* respectively.

Definition 3.3 (Top-k cp-query) A *top-k cp-query* is a CPref-SQL query containing the `ACCORDING TO PREFERENCES` clause. □

We have implemented the consistent test proposed in [Wilson 2004] in order to prevent the user to declare an inconsistent set of cp-rules. So, only consistent set of rules are allowed as parameter for the `ACCORDING TO PREFERENCES` clause. The parameter K is a non-negative integer and is optional. If K is not provided, the query is evaluated using the ***Select-Best*** operator and returns the set of most preferred tuples, without specifying the size of this set. If $K \geq 0$ is given, then the evaluation

[3]In fact, the reverse implication holds iff the preference ordering $>$ is a *weak order*, which is not the case for the preference ordering induced by a cp-theory.

uses the ***SelectK-Best*** operator. For instance, the top-k cp-query (Definition 3.3) of Example 1.1 executed over the relation *MOVIES* depicted in Figure 1 returns $\{t_1, t_5, t_6, t_2\}$ (according to the better-than-graph illustrated in Figure 2(a)).

4. THE ALGORITHMS BNL** AND R-BNL**

In this section we present the algorithms BNL** and R-BNL** for implementing the operators ***Select-Best*** and ***SelectK-Best*** respectively. As remarked in Section 1, the new operators do not increase the expressive power of SQL, since they can be translated into a SQL query. As a matter of fact, any operator designed to produce the non-dominated tuples can be expressed in SQL ([Chomicki 2003]). However, unlike the skyline queries and the pareto queries which can be expressed in the standard SQL-92, our cp-queries goes beyond the expressive power of classical relational algebra, necessarily involving recursion in its specification. That is due to the transitive closure operation involved in the definition of the preference ordering associated to a cp-theory (this operation is not needed in the skyline and pareto queries as explained in Section 3.1). The SQL:99 command corresponding to the top-k cp-query of our running example 1.1 is described below.

```
CREATE OR REPLACE VIEW Rules (T, G, Y, D, A, tit, gen, yea, dir, act) AS
     (SELECT * FROM MOVIES M, MOVIES M1
      WHERE M.G=M1.G and M.Y=90 and M1.Y=80)
      UNION
     (SELECT * FROM MOVIES M, MOVIES M1
      WHERE M.Y=M1.Y and M.G=M1.G and M.D=wa and M1.D=wa and M.A=cr and M1.A=mf)
      UNION
     (SELECT * FROM MOVIES M, MOVIES M1
      WHERE M.G=comedy and M1.G=comedy and M.Y=80 and M1.Y=80 and M.D=jc and M1.D=wa)
      UNION
     (SELECT * FROM MOVIES M, MOVIES M1
      WHERE M.G=drama and M1.G=comedy and M.Y=80 and M1.Y=80 and M.D=M1.D and M.A=M1.A);

WITH RECURSIVE Recursion (tit, gen, yea, dir, act, T, G, Y, D, A) AS (
     (SELECT * FROM Rules )
      UNION
     (SELECT M.T, M.G, M.Y, M.D, M.A, R.T, R.G, R.Y, R.D, R.A
      FROM Rules M, Recursion R
      WHERE M.tit=R.tit and M.gen=R.gen and M.yea=R.yea and M.dir=R.dir and M.act=R.act))
     SELECT * FROM MOVIES WHERE G <> romance
      EXCEPT
     SELECT R.T, R.G, R.Y, R.D, R.A FROM Recursion R;
```

4.1 The algorithm BNL**

Before presenting the algorithm BNL**, we will discuss one essential point of its code, the *dominance test*. The *Dominance Problem* is stated as follows: *Given a cp-theory* Γ *over a relational schema* $R = (A_1, ..., A_n)$ *and two tuples* t_1 *and* t_2 *over R, decide if* $t_1 >_\Gamma t_2$. We remind that t_1 may dominate t_2 because this is directly inferred from a particular cp-rule in Γ, or because this is inferred by transitivity. The dominance test establishes a *preference order* between t_1 and t_2 (see Definition 3.1).

Thus, we implemented a Datalog Program which is responsible for executing the dominance test. The program is built in the following way:

(1) for each cp-rule $r \in \Gamma$: $A_{i_1} = a_1 \wedge ... \wedge A_{i_k} = a_k \rightarrow A_j = b_1 > A_j = b_2[A_{j_1}, ..., A_{j_m}]$, we associate a Horn clause p_r:

$pref(x_1, ..., x_n, y_1, ..., y_n) \leftarrow x_{i_1} = a_1, y_{i_1} = a_1, ..., x_{i_k} = a_k, y_{i_k} = a_k, x_j = b_1, y_j = b_2, x_{k_1} = y_{k_1}, ..., x_{k_l} = y_{k_l}$,

where $j \notin i_1, ..., i_k$, the sets $\{j, i_1, ..., i_k\}, \{j_1, ..., j_m\}, \{k_1, ..., k_l\}$ are disjoints and their union is $\{1, ..., n\}$.

(2) we consider two last Horn clauses, responsible for the transitive closure:

$dom(x_1, ..., x_n, y_1, ..., y_n) \leftarrow pref(x_1, ..., x_n, y_1, ..., y_n)$
$dom(x_1, ..., x_n, y_1, ..., y_n) \leftarrow pref(x_1, ..., x_n, z_1, ..., z_n), dom(z_1, ..., z_n, y_1, ..., y_n)$

The test "$t_1 = (a_1, ...a_n) >_\Gamma t_2 = (b_1, ..., b_n)$?" is achieved by executing the goal $\leftarrow dom(a_1, ..., a_n, b_1, ..., b_n)$. In order to simplify the dominance test, one can project the tuples t_1 and t_2 over a set V of attributes as follows: Let $\Gamma = \{\varphi_1, ..., \varphi_p\}$, and for each $j \in \{1, ..., p\}$ let W_j be the set of attributes between brackets in φ_j. Let V = Attr(R) - $\bigcap_1^p W_j$. Let $t'_1 = t_1[V], t'_2 = t_2[V]$. Then, it is easy to see that $t_1 >_\Gamma t_2$ iff $t'_1 >_\Gamma t'_2$, since for each $j \in \{1, ..., p\}$ the values of the attributes $A \in W_j$ are not taken into account when comparing t_1 and t_2 using the cp-rule φ_j.

Let us illustrate the dominance test in our running example:

Example 4.1 Let Γ be the cp-theory of Example 3.1. We have $W_1 \cap W_2 \cap W_3 \cap W_4 = \{T, D, A\} \cap \{T\} \cap \{T, A\} \cap \{T\} = \{T\}$. So $V = \{G, D, Y, A\}$. Let us consider the tuples t_2 and t_4 of relation *MOVIES* of Figure 1. So, it suffices to compare $t_2[V]$ and $t_4[V]$ by asking the goal $dom(drama, 80, wa, cr, comedy, 80, wa, mf)$ to the Datalog Program associated to Γ:

$pref(g_1, y_1, d_1, a_1, g_2, y_2, d_2, a_2) \leftarrow g_1 = g_2, y_1 = 90, y_2 = 80$

$pref(g_1, y_1, d_1, a_1, g_2, y_2, d_2, a_2) \leftarrow d_1 = \text{'}wa\text{'}, d_2 = d_1, a_1 = \text{'}cr\text{'}, a_2 = \text{'}mf\text{'}, g_1 = g_2, y_1 = y_2$

$pref(g_1, y_1, d_1, a_1, g_2, y_2, d_2, a_2) \leftarrow g_1 = \text{'}comedy\text{'}, g_2 = g_1, y_1 = 80, y_2 = y_1, d_1 = \text{'}jc\text{'}, d_2 = \text{'}wa\text{'}$

$pref(g_1, y_1, d_1, a_1, g_2, y_2, d_2, a_2) \leftarrow y_1 = 80, y_2 = y_1, g_1 = \text{'}drama\text{'}, g_2 = \text{'}comedy\text{'}, d_1 = d_2, a_1 = a_2$

$dom(g_1, y_1, d_1, a_1, g_2, y_2, d_2, a_2) \leftarrow pref(g_1, y_1, d_1, a_1, g_2, y_2, d_2, a_2)$

$dom(g_1, y_1, d_1, a_1, g_2, y_2, d_2, a_2) \leftarrow pref(g_1, y_1, d_1, a_1, g_3, y_3, d_3, a_3), dom(g_3, y_3, d_3, a_3, g_2, y_2, d_2, a_2)$

The answer is *'yes'*. □

The algorithm BNL** implementing the ***Select-Best*** is presented in Figure 3. The main procedure **MostPref**(r) returns the most preferred tuples of the relation r. We note that BNL** employs the Block Nested Looping strategy of the BNL algorithm of [Borzsonyi et al. 2001]. The essential part of the algorithm is the dominance test. The dominance test is invoked in each *"dominated by"* and *"dominates"* calls of the procedure ***MostPref***(r). Let us illustrate the execution of BNL** over our *MOVIES* instance r given in Figure 1. In order to illustrate all the stages of the algorithm (specially when the buffer is full), we suppose that the in-memory page W can store only two tuples.

t_1 is inserted into W, by step 7. So, $W = \{t_1\}$.
t_2 is dominated by $t_1 \in W$. So, t_2 is discarded, by step 5.
t_3 is dominated by $t_1 \in W$. So, t_3 is discarded, by step 5.
t_4 is dominated by $t_1 \in W$. So, t_4 is discarded, by step 5.
t_5 is incomparable with $t_1 \in W$. So, t_5 is inserted into W, by step 7: $W = \{t_1, t_5\}$.
t_6 is incomparable with $t_1, t_5 \in W$. Since W is full, $F = \{t_6\}$, by step 7.
By step 8: $S := W$, $W := \emptyset$ and $F = \{t_6\}$ is the new input. The execution returns to Step 4.
t_6 is inserted into W, by step 7.
By step 8: $S := S \cup W$ and $F = \emptyset$.

```
MostPref(r) :
Input: a set $r$ of tuples
Output: A set $S$ containing the most preferred tuples of $r$

1: Clear the in-memory page $W$ and the temporary table $F$
2: make $r$ the input
3: while the input is not empty do
4:     for all tuple $t$ in the input do
5:         if $t$ is dominated by a tuple in $W$ then ignore $t$
6:         if $t$ dominates some tuples in $W$ then elminate the dominated tuples and insert $t$ into $W$
7:         if $t$ is incomparable with all tuples in $W$ then insert $t$ into $W$ if there is room, otherwise add $t$ to $F$
8:     insert in $S$ the tuples of $W$ which were added there when $F$ was empty
9:     make $F$ the input, clear the temporary table
10: return $S$
```

Fig. 3. Algorithm BNL**

By step 9: $F = \emptyset$ is the new input.
As the input is empty, the execution stops and returns $S = \{t_1, t_5, t_6\}$.

So the most preferred tuples are t_1, t_5, t_6.

4.2 The algorithm R-BNL**

The ***SelectK-Best*** operator is implemented by the algorithm R-BNL** (Ranked BNL**), through the procedure **Top**(r, K). The algorithm is described in Figure 4. Remark that R-BNL** produces the most preferred tuples of relation r by calling **Top**$(r, -1)$.

```
Top(r, K) :
Input: a set $r$ of tuples, a number $K \geq -1$
Output: For $K = -1$: a set $S$ containing the most preferred tuples of $r$
        For $K \geq 0$: a set $S$ containing the $K$ top-k less dominated tuples of $r$
1: if $K = -1$ then
2:     $S =$ MostPref $(r)$
3: else if $K = 0$ then
4:     $S = \emptyset$
5: else if $K > 0$ then
6:     $S' =$ topK$(r)$
7:     $S = \emptyset$
8:     $currentLevel = 0$
9:     while $|S| < K$ and $|S| < |S'|$ do
10:        for every tuple $t$ in $S'$ do
11:            if $l(t) = currentLevel$ then            $//l(t) = \max(\{l(s) \mid s \in \text{MorePref}(t)\}) + 1$
12:                $S = S \cup \{t\}$
13:        $currentLevel = currentLevel + 1$
14: return $S$
```

Fig. 4. Algorithm **R-BNL****

```
topK(r) :

1: Clear the in-memory page $W$ and the temporary table $F$
2: make $r$ the input
3: while the input is not empty do
4:     for all tuple $t$ in the input do
5:         for all tuple $t'$ in $W$ do
6:             if $t$ is dominated by $t'$ then add $t'$ into MorePref$(t)$
7:             if $t$ dominates $t'$ then add $t$ into MorePref$(t')$
8:         insert $t$ into $W$ if there is room, otherwise add $t$ to $F$
9:     insert in $S$ the tuples of $W$
10:    make $F$ the input, clear the temporary table
11: return $S$
```

Fig. 5. Algorithm **topK**

Let us first illustrate the execution of **Top**(r,4), where r is given in Figure 1. Again, we suppose that the in-memory page W can store only two tuples. At step 4, the procedure **topK**(r) is called. Notice that the dominance test is also invoked for each *"dominated by"* and *"dominates"* calls from procedure ***topK***(r). It is responsible for evaluating the list MorePref(t_i) of each tuple t_i in the input r. MorePref(t_i) is the list of all tuples $t_j \in r$ that dominates t_i, i.e., MorePref(t_i) $= \{t_j \in r \mid t_j > t_i\}$. After obtaining the lists MorePref(t_i) for each $t_i \in r$, the second step of the algorithm scans r to get the K tuples with lowest levels (top-k less dominated). Remind (see Definition 3.2) that the level of a tuple t_i is computed as:
$l(t_i) = 0$, for MorePref(t_i) $= \emptyset$
$l(t_i) = \max(\{l(t_j) \mid t_j \in \text{MorePref}(t_i)\}) + 1$, for MorePref($t_i$) $\neq \emptyset$

At step 6, the procedure **topK**(r) is called. Its execution is described below.

t_1 is inserted into W, by step 8. So, $W = \{t_1\}$.
t_2 is dominated by $t_1 \in W$. So, by step 6, MorePref(t_2) $= [t_1]$ and $W = \{t_1, t_2\}$ at step 8.
t_3 is dominated by $t_1, t_2 \in W$. So, by step 6, MorePref(t_3) $= [t_1, t_2]$. Since W is full, $F = \{t_3\}$ at step 8.
t_4 is dominated by $t_1, t_2 \in W$. So, by step 6, MorePref(t_4) $= [t_1, t_2]$. By step 8, $F = \{t_3, t_4\}$.
t_5 is incomparable with $t_1, t_2 \in W$. So, by step 8, $F = \{t_3, t_4, t_5\}$.
t_6 is incomparable with $t_1, t_2 \in W$. So, by step 8, $F = \{t_3, t_4, t_5, t_6\}$.
By step 9: $S := W$.
By step 10: $W := \emptyset$ and $F = \{t_3, t_4, t_5, t_6\}$ is the new input. The execution returns to step 4.
t_3 is inserted into W, by step 8. So, $W = \{t_3\}$.
t_4 is dominated by $t_3 \in W$. So, by step 6, MorePref(t_4) $= [t_1, t_2, t_3]$ and $W = \{t_3, t_4\}$ at step 8.
t_5 dominates t_4. So, by step 6, MorePref(t_4) $= [t_1, t_2, t_3, t_5]$. Since W is full, $F = \{t_5\}$, by step 8 .
t_6 is incomparable with $t_3, t_4 \in W$. So, $F = \{t_5, t_6\}$ at step 8.
By step 9: $S := S \cup W$.
By step 10: $W := \emptyset$ and $F = \{t_5, t_6\}$ is the new input. The execution returns to step 4.
t_5 is inserted into W, by step 8.
t_6 is incomparable with $t_5 \in W$. So, by step 8, $W = \{t_5, t_6\}$.
By step 9: $S := S \cup W$.
By step 10: $F = \emptyset$ is the new input.
Since the input is empty, $S := \{t_1, t_2, t_3, t_4, t_5, t_6\}$ with their respective MorePref lists.

The next steps of the execution of **Top**(r,4) are illustrated below.

At step 6, $S' = \{t_1, t_2, t_3, t_4, t_5\}$, at step 7 $S = \emptyset$ and at step 8 $currentLevel = 0$.
level of t_1 is computed, by step 11. So, $l(t_1) = 0 = currentLevel$ and $S = \{t_1\}$, by step 12.
By step 11, $l(t_2)$ is computed: $l(t_2) = max(l(t_1)) + 1 = 1$. So, $l(t_2) \neq currentLevel$.
By step 11, $l(t_3)$ is computed: $l(t_3) = max(l(t_1), l(t_2)) + 1 = 2$. So, $l(t_3) \neq currentLevel$.
By step 11, $l(t_4)$ is computed: $l(t_4) = max(l(t_1), l(t_2), l(t_3), l(t_5)) + 1 = 3 \neq currentLevel$.
$l(t_5) = 0 = currentLevel$, by step 11. So, $S = \{t_1, t_5\}$, by step 12.
By step 11, $l(t_6)$ is computed: $l(t_6) = 0 = currentLevel$ and, by step 12: $S = \{t_1, t_5, t_6\}$.
$currentLevel = 1$ at step 13. The execution returns to step 9.
At step 11, $l(t_2) = 1 = currentLevel$. So, $S = \{t_1, t_5, t_6, t_2\}$, by step 12.
As $|S| = K = 4$, by step 9, the execution stops and returns $S = \{t_1, t_5, t_6, t_2\}$.

5. EXPERIMENTAL RESULTS

All the experiments have been performed on a 2.2GHz AMD Turion X2 Ultra Dual-Core PC, with 4GB of main memory, running Linux. We have used *psql* as front-end to PostgreSQL 8.4, connecting to the server on a local host, to handle the queries. The source code of CPref-SQL can be downloaded from *http://www.lsi.ufu.br/downloads/cprefsql/*.

5.1 Experimental Environment

Datasets. The experiments have been performed using the TPC-H benchmark[4]. TPC-H is a specific benchmark for ad-hoc querying workloads. Its schema consists of eight tables modelling a typical retail environment. Tables such as *customer*, *supplier*, *part* and *partsupp* contain information about items that retail companies buy from their suppliers and sell to their customers, while *nation* and *region* are small tables containing only a few rows. These tables amount to about 15% of the total database. The two largest tables, *lineitem* and *orders* contribute to the remaining 85% of the total database size.

The TPC-H benchmark provides a dataset generator (dbgen) which creates tuples in the tables of the TPC-schema, based on a scale factor SF[5]. The size of the tables, excepting tables *nation* and *region* is proportional to the scale factor. The datasets used in our experiments were generated with SF = 0.001, 0.005, 0.01, 0.05 and 0.1. Thus, the sizes of our datasets are 1MB, 5MB, 10MB, 50MB and 100MB, respectively. The default database (DB-DEF) has size of 10MB.

Queries. We have adapted the benchmark queries to the preference context in order to use them in our experiments. This adaptation has been achieved by using the following criteria: *(a)* removal of aggregate functions since they are not supported by our current implementation of the ***Select-Best*** and ***SelectK-Best*** operators; *(b)* insertion of the preference clause (ACCORDING TO PREFERENCES) and *(c)* changes on the terms of the WHERE clause in order to vary the reduction factor of the selection operation. The benchmark queries Q3, Q5, Q10 and Q18 we have considered in the experiments enabled an expressive variation on the number of rows submitted to the preference operator. More precisely: 215, 2333, 5756 and 10606 tuples for queries Q3, Q5, Q10 and Q18 respectively. Query Q5 was taken as the default one.

Preferences. The preference rules have been built by taking into account two main features: the number of rules and the *depth level*. The antecedents of the preference rules considered in the experiments are a boolean composition of two atomic formulas, that is, the rules are of the form $(A = a \wedge B = b \rightarrow ...)$. The number of rules used in the experiments varies from 2 to 40 rules. The depth level involves the interaction between the preference rules through transitive closure. For instance, let $\Phi = \{\varphi_1,\ \varphi_2\}$, where φ_1: $A = a_1 \wedge C = c_1 \rightarrow B = b_1 > B = b_2\ [E]$ and φ_2 : $C = c_1 \wedge D = d_1 \rightarrow B = b_2 > B = b_3\ [E]$. The depth level of Φ is 2, since the maximum length of a path induced by the cp-rules is 2. A path induced by cp-rules is a preference order over tuples induced by the composition of two or more cp-rules. Here, this path has the format $(a_1,b_1,c_1,d_1,X) > (a_1,b_2,c_1,d_1,Y) > (a_1,b_3,c_1,d_1,Z)$. The depth level of the cp-theories we have used in our experiments varies from 1 to 6 levels.

The default cp-theory (prefDEF) has 6 rules and depth level 2. All the cp-theories were built over the attributes of the eight tables of the TCP-H schema we considered.

Buffer Size. For queries execution, we have also taken into account the available main memory buffer of the DBMS, which varied from 1MB to 28MB (the maximum allowed by the operational system). The default size has been taken as 8MB.

5.2 Performance and Scalability

We have tested the performance and scalability of the ***Select-Best*** operator by comparing the top-k cp-queries and their corresponding translations into standard SQL with recursion. We have also tested the ***SelectK-Best*** operator by varying the K parameter in the ACCORDING TO PREFERENCES clause of the benchmark queries. For this latter test, we did not compare the queries with their SQL

[4]http://www.tpc.org/tpch/

[5]The SF factor specifies the database size. For instance, if SF=1 then the size of the eight tables constituting the dataset is 1GB.

counterpart, since the translation is rather cumbersome and the comparison tests would not bring more information than that already obtained by testing the ***Select-Best*** operator (where K = -1).

Performance analysis. In Figure 6(a), we present the results of the experiments comparing the performances of CPref-SQL and SQL when executing the query Q5, with 8MB of available main memory. All the cp-rules have depth level 2 and the number of rules of the cp-theories considered varies from 2 to 40 rules. Note that the performance of the SQL query decreases as the number of rules increases. This can be explained as follows: as the number of rules increases more tuples can be compared. So, the number of tuples returned by the MostPref procedure decreases (i.e. the possibility for two tuples to be incomparable and both be most preferred is lower). If the most preferred tuples fit into the window W of the MostPref procedure (see Figure 3), the algorithm terminates in one or two iterations and so its complexity is $O(n)$, where n is the size of the input. That is not the case with the translated SQL query, involving a *Join* between the tables *Rules* and *Recursion*.

In Figure 6(b), we illustrate the behaviour of query executions when varying the depth level of the cp-theories. This feature has a high impact on the performance of the translated query. As the depth level increases, the table *Recursion* has more tuples, and so the performance of SQL Q5 decreases (due to the *Join* operation between *Rules* and *Recursion*). As the recursion level and, consequently the depth level increases, the performance of SQL decreases.

The preference features considered in Figures 6(a) and 6(b) show that the performances of SQL and CPref-SQL depend on the selectivity factor of the rules, that is, the number of tuples returned by the MostPref procedure. The number of rules and the depth level of the theory have a considerable negative impact on SQL performance. On the other hand, the performance of CPref-SQL has not been affected by varying these two parameters.

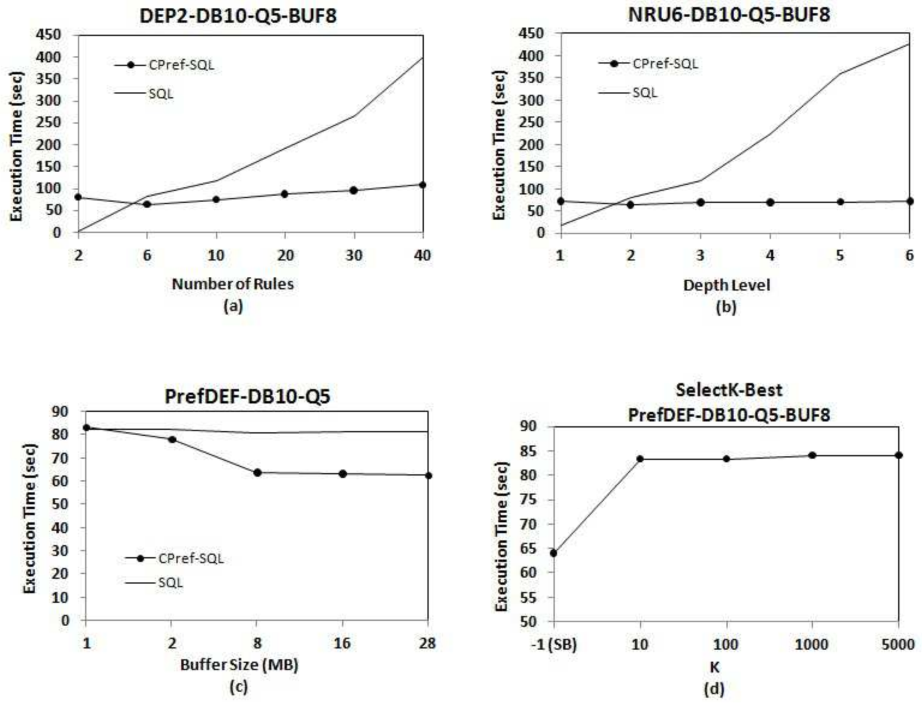

Fig. 6. Performance results

In Figure 6(c) we can see how the amount of available main memory affects the performance of the preference queries executions. We have varied the size of the main memory buffer from 1MB (1% of the size of the database) to 28MB (this is the default RDBMS value and greater than 100% of the size of the default database DB-DEF (10MB)). Clearly, the performance of the CPref-SQL query presented a larger oscillation with respect to the available buffer, due the fact that the MostPref procedure is directly affected by the size of the window W. Note that from 8MB on, the execution time of CPref-SQL is nearly constant, since the most preferred tuples fit in main memory (inside the window W).

We also tested the performance of top-k cp-queries by varying the parameter K (the desired amount of preferred tuples) as shown in Figure 6(d). The performance of the R-BNL** is pratically unchanged for different values of $K > 0$, since on average the algorithm scans only the half part of the relation in order to obtain the K most preferred tuples. As expected, the performance is better for $K = -1$, since for $K > 0$ the algorithm needs extra time to process the different levels.

Scalability results. Figure 7(a) shows how both SQL and CPrefSQL queries scale when the size of the database increases from 1MB to 100MB. Again, we notice that CPref-SQL is more efficient than SQL. Clearly, the amount of most preferred tuples affects the behaviour of the queries. As this amount increases, the performance of the SQL query decreases in a larger rate than the one of the CPref-SQL query.

In Figure 7(b) we can see the behaviour of both query languages when varying the reduction factor of the WHERE clause, that is the number of tuples directly submitted to the preference operator (after the join and selection operations according to the canonical execution plan showed in Figure 2(c). As mentioned before, queries Q3, Q5, Q10 and Q18 return 215, 2333, 5756 and 10606 tuples respectively when executed over the default database DB-DEF (10MB) without the preference clause. The performance of CPref-SQL is far better than SQL performance. For instance, for query Q18 CPref-SQL executes 12 times faster than SQL. Note that as the number of tuples submitted to the preference operator increases, the performance of SQL drastically decreases. This is not the case for CPref-SQL. Its performance decreases in a very low rate.

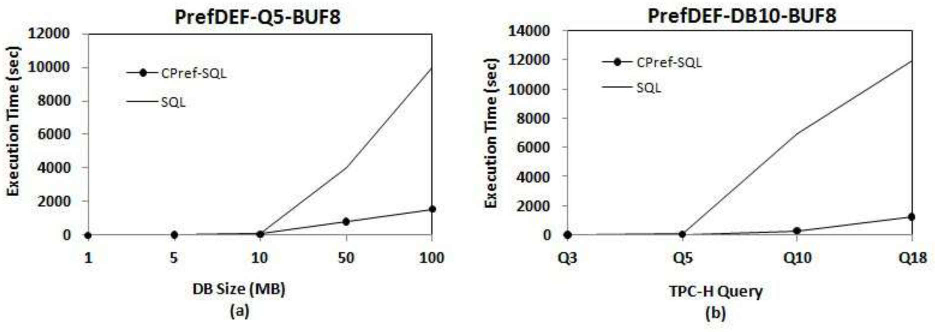

Fig. 7. Scalability results

6. CONCLUSION AND FURTHER WORK

In this paper we have introduced the top-k cp-queries allowing user preferences to be taken into account in the query answering process. We proposed the algorithms BNL** and R-BNL** for evaluating these queries and implemented them in the core of the PostgreSQL query processor. Our experiments showed a considerable better performance and scalability of our operators with respect to their corresponding translation into standard SQL. Of course, a lot of work remains to be done. We intend to follow four directions of research in the future, namely: (1) Improving the performance of the dominance test. The complexity of BNL** is $O(n^2(n^3 + n^2.m))$, where n is the size of the input relation r and

m is the number of rules in the cp-theory. As $n >> m$, then this complexity is O(n^5). In fact, the dominance test (as implemented in BNL**) has a high impact on the complexity of the algorithm. The complexity of the MostPref procedure is $O(n^2.f(n,m))$ in the worst case, where $O(f(n,m))$ is the complexity of the dominance test. In our implementation of the dominance test, $f(n,m) = n^3 + n^2.m$. We intend to employ more efficient techniques to implement the transitive closure. (2) Design of index-based algorithms for evaluating the top-k cp-queries. (3) Incorporating aggregate operations in the CPref-SQL block. (4) Implementing the algorithms under the approach *on-top* and compare the performance with the *built-in* approach proposed in this paper.

REFERENCES

Agrawal, R., Borgida, A., and Jagadish, H. Efficient management of transitive relationships in large data and knowledge bases. *SIGMOD Record* 18 (2): 253–262, 1989.

Borzsonyi, S., Kossmann, D., and Stocker, K. The skyline operator. In *Proceedings of the International Conference on Data Engineering*. Washington, USA, pp. 412–430, 2001.

Boutilier, C., Brafman, R., Hoos, H., and D.Poole. Cp-nets: A tool for representing and reasoning about conditional ceteris paribus preference statements. *Journal of Artificial Intelligence Research* 21 (1): 135–191, 2004.

Brafman, R., Domshlak, C., and Shimony, S. E. On graphical modeling of preference and importance. *Journal of Artificial Intelligence Research* 25 (1): 389–424, 2006.

Carey, M. and Kossman, D. On saying "enough already" in sql. *SIGMOD Record* 26 (2): 219–230, 1997.

Chan, C.-Y., Eng, P.-K., and Tan, K.-L. Efficient processing of skyline queries with partially-ordered domains. In *Proceedings of the International Conference on Data Engineering*. Washington, USA, pp. 190–191, 2005a.

Chan, C.-Y., Eng, P.-K., and Tan, K.-L. Stratified computation of skylines with partially-ordered domains. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. New York, USA, pp. 203–214, 2005b.

Chomicki, J. Preference formulas in relational queries. *ACM Trans. on Database Systems* 28 (4): 427–466, 2003.

Chomicki, J., Godfrey, P., Gryz, J., and Liang, D. Skyline with presorting: Theory and optimizations. In *Proceedings of the Intelligent Information Systems Conference*. Gdansk, Poland, pp. 595–604, 2005.

de Amo, S. and Ribeiro, M. R. Cpref-sql: A query language supporting conditional preferences. In *Proceedings of the Annual ACM Symposium on Applied Computing*. Honolulu, USA, pp. 1573–1577, 2009.

Endres, M. and Kiessling, W. Transformation of tcp-net queries into preference database queries. In *Proceedings of the Multidisciplinary Workshop on Advances in Preference Handling*. Riva del Garda, Italy, pp. 23–30, 2006.

Hristidis, V., Koudas, N., and Papakonstantinou, Y. Prefer: A system for the efficient execution of multi-parametric ranked queries. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. Santa Barbara, USA, pp. 259–270, 2001.

I.F.Ilyas, Aref, W., and Elmagarmid, A. Supporting top-k join queries in relational databases. *The VLDB Journal* 13 (3): 207–221, 2004.

Kiessling, W. and Köstler, G. Preference sql - design, implementation, experiences. In *Proceedings of the International Conference on Very Large Data Bases*. Hong Kong, China, pp. 990–1001, 2002.

Koutrika, G., Simitsis, A., and Ionnadis, Y. Précis: The essence of a query answer. In *Proceedings of the International Conference on Data Engineering*. Atlanta, USA, pp. 69–78, 2006.

Lin, X., Yuan, Y., Zhang, Q., and Zhang, Y. Selecting stars: The k most representative skyline operator. In *Proceedings of the IEEE International Conference on Data Engineering*. Istambul, Turkey, pp. 86–95, 2007.

Papadias, D., Tao, Y., Fu, G., and Seeger, B. Progressive skyline computation in database systems. *ACM Trans. Database Syst.* 30 (1): 41–82, 2005.

Preisinger, T. and Kiessling, W. The hexagon algorithm for pareto preference queries. In *Proceedings of the Multidisciplinary Workshop on Advances in Preference Handling*. Vienna, Austria, 2007.

Preisinger, T., Kiessling, W., and Endres, M. The bnl^{++} algorithm for evaluating pareto preference queries. In *Proceedings of the Multidisciplinary Workshop on Advances in Preference Handling*. Riva del Garda, Italy, pp. 114–121, 2006.

Wilson, N. Extending cp-nets with stronger conditional preference statements. In *Proceedings of the National Conference on Artifical Intelligence*. San Jose, California, pp. 735–741, 2004.

Yiu, M. L. and Mamoulis, N. Efficient processing of top-k dominating queries on multi-dimensional data. In *Proceedings of the International Conference on Very Large Data Bases*. Vienna, Austria, pp. 483–494, 2007.

Yiu, M. L. and Mamoulis, N. Multi-dimensional top-k dominating queries. *The VLDB Journal* 18 (3): 695–718, 2009.